Learning Computer Science by Studying Computer Science Education

A Plan of Concentration in Computer Science

Richard Scruggs

Jim Mahoney, sponsor

December, 2010

Marlboro College, Marlboro, Vermont

Acknowledgements

This plan project would still be a few half-formed ideas if it weren't for the help of several people:

Thank you to Jim Mahoney, for sponsoring me through this undertaking – and for supporting my coming back to Marlboro to do it. I think I finally believe you about test-driven development, although it certainly took me long enough.

Thank you to Jon Mack, for going out on a limb to work with me on educational theory, and thank you for Sean Conley, for taking on a new challenge and helping me with the education papers. Thanks to John Arhin, for turning what started as an ordinary graph theory tutorial into a nearly semester-long discussion of the traveling salesman problem.

Finally, thank you to Daniella Nahmias, for the mental support, but also for reading the papers, helping with the data collection and formatting, getting the articles and books… everything.

This is my Plan of Concentration in computer science. It consists of several parts, with grading percentages as follows:

Plan Project – 40%:

One paper on computer science curriculum and one on educational theory in computer science education.

Other Plan Component – 30%:

An implementation of the traveling salesman algorithm.

Independent Plan Work – 30%:

Computer science exams in programming languages, discrete math, and algorithms.

Table of Contents

Computer Science Education Papers

Traveling Salesman Project

Computer Science Proficiency Exams

A Discussion of Undergraduate Computer Science Curricula

It has become common sport to criticize computer science curricula. CIOs complain about skills lacking in their new hires, programmers lament the perceived dumbing down of the discipline, and computer science educators talk of how the curriculum must be changed to weather a crisis and fit the broadening of the field (Dewar & Schonberg, 2008; Cassel et al., 2008). The critics tend to focus on a few facets of the field as a whole--most typically programming-related areas--and use a few apparent flaws to deride the computer science curriculum as a whole. Meanwhile, many of the people who are trying to change CS curricula are arguing that small facets of the field should be changed while not addressing wider effects of those small changes. Before continuing to change the CS curriculum piecemeal, however, a firm grounding in educational principles should be established. Such a foundation would allow the field to slowly evolve in order to stay current, while helping CS educators understand how their changes both fit and change the wider context and goals of the field.

Curricular theory is underutilized in computer science education. While modern curricular theory often gets bogged down in details of definition and implementation, the field as a whole has identified and attempted to answer many questions which are increasingly relevant to computer science education. Mark Guzdial (2005) states that CS educational researchers should build on the pedagogic foundation created by educationalists and educational psychologists (in Hazzan et al.). The field of curriculum is very similar; indeed, as computer science is changing faster than most fields, it is even more important that those who shape it understand the purpose of and reason for their changes (Soh et al., 2007). Curricular theorists, while giving more of their time to the questions of what should be learned and how to organize it, have produced a great deal of discourse on how curricular change should be effected. For this

reason, it is important that computer science educators have a solid understanding of the efforts of curricular theorists and how those efforts may be best applied when constructing or modifying their own curricula.

Petre (2005) points out that computer science education lacks a "driving theory," and as such must borrow from other disciplines (in Hazzan et al.). She states that what is important is that borrowing is done properly, with the borrowing educator understanding the context of the borrowed method and making certain that the method works. CS education does borrow from some educational theory, but curricular discussions in CS rarely make reference to recent theorists. Many of these theorists offer more directly useful ideas and talk more about the realities of curriculum design than their earlier colleagues.

The Current Process of Curricular Change

At present, there are two approaches to undergraduate CS curricular change: centralized bodies that attempt to change the direction of the field as a whole, and small-scale efforts that attempt to change the curriculum in one class, or at one school. The former is led by the Association for Computing Machinery and its curricular recommendations.

It is difficult to speak to the scope and success of small-scale curricular change. Randolph et al. (2008) review several hundred recent computer science education articles and come to the conclusion that nearly half of these articles discuss a "new way to organize a course". From this, it is evident that there are many educators discussing and implementing curricular change on a small scale. Many of the articles published, however, only discuss the author's experience implementing his new ideas in one course, with one posttest to determine success or failure; there are rarely follow-up articles. Hence, it is unknown whether these small changes take root or if the experiments were merely one-time events.

In his dissertation, Randolph (2007) discusses the presence of a literature review in computer science education articles. He theorizes that the majority of such articles do not perform a literature review, or perform an inadequate review, but is unable to conclusively support this. If most CS education articles do not in fact cite each other, it is even more difficult to determine whether their ideas are having any effect outside of their small test areas.

Program accreditation is one way that a central body might shape undergraduate CS education. In the United States, ABET is the organization that accredits many science and engineering programs, including computer science. Like the ACM, they are attempting to split the all-encompassing CS degree into several other programs, each with a more clearly expressed set of goals. Their primary focus for CS is that graduates should be able to stay current and competent in the field throughout their professional lives (Reif and Mathieu, 2009).

Reif and Mathieu, who work with ABET, discuss the current scope of CS program accreditation, coming to the conclusion that while only a minority of CS programs are accredited, accreditation is becoming more widespread. Of the programs I examined, five had ABET accreditation, but their requirements differed from each other as much as they differed from requirements of other schools in the same category.

Further, ABET's mission and strategic plan suggest that while they are interested in "[promoting] quality and innovation in education," they are an organization that adapts its role to fit the needs of its constituencies, not an organization that attempts to direct their educational goals or practices through accreditation standards or requirements. If ABET tried to take more of a leadership role in the determining of CS curriculum, I do not think that they would have a great deal of success - convincing hundreds of large, powerful universities that they must change the content of their programs is at best an exceedingly difficult task.

The Association for Computing Machinery [ACM] constructs undergraduate curricula recommendations for five computing disciplines: computer science, computer engineering, software engineering, information systems, and information technology. It also offers curricular recommendations for computer science in liberal arts colleges, and recommendations for computer science at two-year schools. Looking at a small sample of computer science curricula, I found that the ACM CS recommendations bear general resemblance to CS curricula at most schools, but few schools adhere to the ACM recommendations closely. Further, the ACM's recommendations are constructed with the input of faculty at only a small percentage of computer science programs. Nevertheless, their recommendations are the closest thing that computer science has to a unified curriculum.

The ACM CS recommendations state several guiding principles that they attempted to follow in the creation of the recommendations and a list of characteristics that they believe that all computer science graduates should possess, but they never explicitly state the purposes of their curriculum. Their listed characteristics suggest that students should understand both a high-level overview and lower-level implementation of theory, and that students should be able to adapt and keep pace with the field (63). Although they do not explicitly say so, it is evident from examining their characteristics and the general tone of their report that the main purpose of their curriculum is to prepare students for jobs in the computing field.[1]

An Examination of Undergraduate Curricula

While the ACM curricular recommendations provide a comprehensive framework for a computer science curriculum, curricula often change significantly from design to implementation. Before discussing CS curriculum, it is important to make that distinction and to

[1] In the Computing Curricula 2005 report, they do have one sentence: "It is important that the computing disciplines attract quality students from a broad cross section of the population and prepare them to be capable and responsible professionals, scientists, and engineers" (Shackleford et al., 1).

evaluate both planned curriculum and implemented curriculum. I examined the curricula of sixteen computer science programs, four leading to an associate's degree in computer science, twelve leading to a bachelor's degree in computer science. Four curricula were from liberal arts colleges, four were from large universities, four from schools [research universities] highly ranked by US News for computer science or computer engineering, and four from community colleges. The Carnegie classifications of the schools produce a similar grouping to what I am using, but do not differentiate all the top tier schools from the large research universities (Carnegie Foundation). Although there were some curricular similarities within the four groups, there was a great deal of variation across the sixteen curricula examined—and almost as much variation across the curricula of the twelve schools that offered bachelor's degrees.

The community colleges were clearly the most career-oriented of the four groups. Classes offered included "MFC Windows Programming," "C# with .NET," and "HTML Programming"—all skills that would be directly applied in the workforce. Some math courses were offered, but none of the community colleges examined required any. There were, however, CS courses that seemed to go beyond simple job skills; all of the schools offer data structures courses, with one requiring it, and two offer computer organization courses, with one requiring it. The total number of CS-related[2] courses required for the associate's degree ranges from three to seven.

The large universities had the most required CS-related classes of any group, from twelve to thirty-four.[3] Their curricula, likely aided by the greater number of classes, also achieved the best coverage of the ACM recommendations. Unlike the community colleges, the large universities do not offer courses designed to teach directly applicable job skills. Their curricula

[2] For the purposes of this examination, I count CS, electrical engineering, and math courses as "CS-related".
[3] The hypothetical ACM curriculum for research universities in the 2001 report has 22 courses, omitting the science electives (47).

seem designed to give students as much of a grounding in the different subfields of CS as possible, with a strong emphasis on programming and implementation—although there is certainly ample theory to be found. In this, as well, they follow the ACM recommendations.

Liberal arts colleges required the fewest CS-related classes of the four-year groups, from seven to eleven.[4] Their curricula bear some resemblance to the ACM recommendations for small departments and the Liberal Arts Computer Science Consortium recommendations, but they pare down the core even further to allow for broader studies. They require few or no math classes, and their classes seem to focus more on the theory of CS than the implementation, although some of them emphasize implementation through a large capstone project. Liberal arts colleges, as would be expected from their general ethos, attempt to educate their CS students to understand what they see as the nature and theory of the discipline.

The top-ranked research universities require between thirteen and twenty-four CS-related classes. In general, curricula at top-tier schools bears more resemblance to the liberal arts colleges than the large research universities, with a lot of emphasis on theory, a small core, and many electives. They require several math classes, but not more than the large universities. Some of them also require large capstone projects, or independent research. They seem to be educating for research more than for ordinary work in the field; many of their courses emphasize theoretical, mathematical aspects of computer science before particular programming implementations.

This examination immediately shows the importance of one contextual factor. The examined schools generally share a great deal of curricular similarity with others in their category, and their curricula differ significantly from those at schools in other categories.

[4] The ACM 2001 hypothetical curriculum for small departments has 14 courses (50); the 2007 curriculum by the Liberal Arts Computer Science Consortium has 11 courses.

More important to this paper is what the examination reveals about the process of curricular formation at these schools. The ACM recommendations are broadly followed, but curricular change is driven less by the direction of a central body, and more by factors internal to each school or department. Unlike the ACM, individual departments rarely publicize the aims of their computer science programs, so it is difficult to compare aims with results. Further, it is difficult to determine exactly why computer science curricula as implemented differ from the hypothetical curricula proposed by the ACM.

Curricular change has never been easy to affect, particularly in the realm of higher education. It is difficult to publicize any new changes, and publicity alone has relatively little effect. In 1975, Kelly determined that, five years after the introduction of a new [British, secondary school, science] curriculum, 10-20% of schools at which it was advertised had adopted it—some of them incorrectly—and 20% of the target schools were unaware of it (cited in Barrow, 1984). This can be compared to a smaller computer science case in higher education. Ni et al. (2010) interviewed eight instructors to determine how much they had adopted from a series of workshops on a media computation course: they found that of the eight, four had not adopted from the course, one hoped to adopt it soon, one had adopted part of the course, another had adopted the course, and one instructor's entire department had adopted the course. The two cases are not strictly comparable—the introduction to the new material and breadth of the new material varies greatly—but there is ample evidence to suggest that preparing, publicizing, and even teaching new material are far from sufficient to ensure its adoption. Worse yet, Gornitzka, Kogan, and Amaral discuss several studies which indicate that policies as adopted often differ from policies as implemented (7).

The ACM recommendations do not always address the difficulty of curricular change, particularly large-scale change. In the Computing Curricula 2005 Overview Report, the ACM argues for a split between the subfields of CS so that those subfields better reflect the uses of CS in the outside world. However, many of the schools that they are trying to convince have already started offering some of the subfields as specializations within an ordinary CS degree, and as such would be resistant to adopting a different solution. CS degrees, as compared to degrees in IT and IS, have an established cachet in the computing industry, and it would be difficult to give that to another degree. Students also have certain expectations relating to their CS degrees, and transferring those expectations to degrees in information systems, or even software engineering, would be nontrivial.

Given these difficulties, it is evident that curricular change is a difficult process in any discipline. While curricular theorists and researchers have not solved all the difficulties of implementation, they do offer many strategies that are helpful. In the next section, I will introduce several curricular theories and talk about how they might apply to CS curriculum.

Reconciling the Process of Curricular Change with Educational Theory and Practice

There are many approaches championed by curricular theorists that explain how curricular change should be enacted. However, before attempting to change a curriculum, one should understand that curriculum. What are the questions that educational theorists ask when examining curricula? Tyler (1949), in his *Basic Principles of Curriculum and Instruction*, offers four questions to be answered when constructing a curriculum:

1. What educational purposes should the school seek to attain?
2. What educational experiences can be provided that are likely to attain these purposes?
3. How can these educational experiences be effectively organized?
4. How can we determine whether these purposes are being attained? (p. 1)

While Tyler's questions are intended to be used when constructing a curriculum, they are sufficiently general that they may be asked of an existent curriculum as well. Tyler has received a great deal of attention from curricular theorists, and his objectives-focused model is the model most commonly used in computer science. Many more recent educational theorists, however, see Tyler's perspective as limiting, and instead favor a more holistic, philosophical approach to curriculum development.

Mannheim (1962) was an early educational sociologist who supported the idea that a simple, objectives-based approach does not sufficiently describe educational interactions: "It is not enough to say that this or that educational system or theory or policy is good. We have to determine for what it is good, for which historical aims it stands and whether we want this educational result" (p. 44). Educational and curricular theorists took a similar idea from philosophers: "To make anything other than random guesses about the curriculum of any institution, we need to know […] about the context of curriculum" (Grundy, 1987, p. 7). This view is shared by many within curricular theory—the success or failure of a curriculum can only be evaluated within its context.

The context of computer science curriculum is itself a misleading phrase: computer science, as a field, is sufficiently multifaceted so as to have many contexts and many curricula. This is evidenced by the earlier examination of computer science curricula. The ACM does offer curricular recommendations for several contexts—'ordinary' CS programs, CS in liberal arts colleges, and CS in two-year schools, but it is difficult to find out whether these contexts accurately divide CS as a whole. A larger curricular study would provide data from more categories of schools that could then be compared to ACM classifications; such data could point to other factors that may influence contextual boundaries of CS education.

With the idea of context, it often is easier to modify curriculum on a smaller-scale, more individual basis. Lawrence Stenhouse (1975), a British curricular theorist, was known for his belief that teachers should themselves act as curricular researchers; while he focuses his argument on secondary schools, I believe it is even more applicable to higher education. The average professor, having more control over the content and structure of her course than her colleague in a high school, is in a much better position to bring about change from within—and, as Randolph et al. (2008) discuss, she often acts to do so.

Even with the importance of context and the difficulty of setting standardized aims for CS curriculum as a whole, the aims of a particular CS curriculum should be carefully considered. While some of the curricular theorists discussed in this section are opposed to concrete objectives, a curriculum should certainly have general aims. In the next section, I will discuss several factors that may influence the aims of a particular computer science curriculum.

What Should the Goals of Undergraduate CS Curriculum Be?

Curricular theorists often say that curriculum should serve the best interests of the students then define those interests to fit their needs. Today, though, we have a great deal of data on the modern university student, and instead of defining the needs of the student, we can use this data to attempt to discern those needs. In computer science, though, there are other pertinent needs beyond those of the student: Industry needs a supply of workers who have certain computer-related skills and universities need students who can thrive in graduate schools and ultimately expand the frontiers of the discipline. Computer science educators seem to pay better attention to these needs, sometimes equating them with those of the students; while they should not be equated, they should certainly be addressed. The Bureau of Labor Statistics [BLS] offers data that can be used to depict the CS workforce and their educational needs.

Bureau of Labor Statistics Data

The Bureau of Labor Statistics produces data relating to employment. Unlike NCES, they offer a more detailed breakdown of computer-related occupations, splitting them into several categories, but they offer no data about the people working those categories and their educational backgrounds. I have examined the computer-related fields within their category of "Computer and Mathematical Science Occupations," excluding the strictly mathematical occupations within that category, and have also included "computer science teachers, postsecondary." I have excluded "computer support specialists" as, based on the job description and needed qualifications, the majority of employees in that occupation likely have degrees in IT rather than CS.

Bureau of Labor Statistics Data on Computer Science-Related Occupations

15-1011	Computer and information scientists, research	26,610
15-1021	Computer programmers	394,230
15-1031	Computer software engineers, applications	494,160
15-1032	Computer software engineers, systems software	381,830
15-1051	Computer systems analysts	489,890
15-1061	Database administrators	115,770
15-1071	Network and computer systems administrators	327,850
15-1081	Network systems and data communications analysts	230,410
15-1099	Computer specialists, all other	191,780
25-1021	Computer science teachers, postsecondary	32,520
	Total	2,685,050

Of the CS-related occupations indexed by the BLS, about half are either programming or software engineering, to an extent validating those who are alarmed at a perceived lack of programming skills.

Many of the employees in software engineering have degrees in computer science ("Computing Careers"). The percentage of employees in the various subfields does not match up very closely with the amount of instructional time either recommended by the ACM or observed in the above examination of curricula. If CS curriculum is indeed meant to prepare students for immediate job placement in the field it could dovetail far better with the needs of employers. On the other hand, employers' future needs are harder to ascertain and the ACM's recommendations may prepare students for productive careers better than for immediate hiring.

Tan and Venable (2008) suggest that while CS curriculum might be more focused on the teaching of vocational skills, the goal of the university at large is to generally educate all of its students. While all CS curricula that I examined leave general education to the purview of the university, they all cover some aspects of the field that are theoretical enough to have little vocational application. The question of the precise balance would seem more contextual than anything else—as they should, schools adjust the balance between career vocational skills and general education to fit their goals.

The desires of students are difficult to ascertain. Cohen and Kisker (2010) state that 75% of college students go to college for career training, or to make money, but that reflects the goals of all college students, not merely computer science majors. The National Center for Education Statistics offers data that can be examined more specifically. 78% of surveyed computer science majors said it was important to be financially well off, not significantly different from the average of 77%. After two years, 71% of computer science majors said it was important, slightly

higher than the average of 68%. 39% of computer science majors expected that a bachelor's degree would be the highest degree they would receive, above the average of 32%; an additional 33% of computer science majors expected to terminate with a master's, closer to the average of 31%. Two years later, 39% still expected to terminate with a bachelor's degree, but those planning to terminate with a master's had dropped to 25%, below the average of 29%. When considering students earning two-year degrees, 56% of them chose their school to learn job skills, above the average of 47%.[5]

This data does not translate well to precise student needs, but it does generally suggest that many computer science students, slightly more than in other majors, are studying computer science for career-related skills. According to a report by Georgetown University's Center on Education and the Workforce, 51% of the "computing and mathematical science occupations" require bachelor's degrees and an additional 20% require master's degrees (Carnevale et al., 2010). The report goes on to state that job prospects in these occupations are bright, suggesting that computer science students' career hopes are not misplaced.

Computer science students' specific hopes within the larger field are harder to determine, but those hopes are also less valuable. Yasuhara (2008) states that many students' initial perceptions of computer science are shallow and do not accurately reflect the field as a whole. However, it would still be useful to have more data about the tracks students pursue within computer science.

Ideal Computer Science Curricula

I should make clear one important caveat before discussing any ideal curriculum. Tyler, as many theorists, discusses curriculum as though it were a blank slate and its designer were

[5] These comparisons are difficult to make accurately as NCES combines computer science with information systems in their major categories.

implementing it for the first time; his exploration of ideal goals does not make allowances for a preexisting curriculum. Curricular recommendations that recognize—as the ACM's do—that they will not be implemented on blank canvases but instead will be strongly affected by current practice are much better prepared for adoption (Grundy 6).

Discussions of ideal curricula differ among curriculum theorists and educators. The former tend to be more idealistic, while the latter are more constrained by what currently exists and what they believe they can change. I fall with the latter group, hence the mentions of existent CS curricula in this section. I believe that it would be exceedingly difficult to implement an ideal CS curriculum from scratch at a university with an existing program. As the overwhelming majority of CS curricular change occurs at such universities, any list of ideal goals for CS curriculum should be based in reality.

While difficult, creating an ideal curriculum or curricular modification for a certain context is only the first step in the process of curricular change. The planned modification must be put in practice in order to be of use, and as computer scientists know, implementation of new ideas is often quite a challenge.

When implementing curricular ideas, it is critical that a professor understands the import and context of his changes. One can study articles written by CS educators who have reorganized courses or introduced new ones. In the articles I have examined - which admittedly is by no means a complete collection, or even a properly representative sample - I have not seen such considerations discussed. This omission would not only affect the larger aims of CS; it also suggests that insufficient attention is being paid to context in general.

Affecting curricular change requires consciousness of context, as Grundy states. Educators must understand the forces already at work on the curriculum that they are attempting

to modify, and if they can work against as few of those forces as possible, their changes will have a much better chance of success. In this case, the ACM might meet with better results if, instead of attempting to push students to new degree fields, they work with the specializations that universities already use. Professors attempting to make smaller changes should also understand what is driving them to make those changes and how they can best work with the students, their department, and any outside forces to effect their desired change.

Conclusion

Curricular change in undergraduate computer science education is inevitable. New developments in computer science will make it necessary for universities to educate students in new areas of the field; changing preferences will obsolete other areas. As change happens, it is important that computer science educators are well-equipped to both cope with it and enact it. Many curricular theorists discuss how curricula should be constructed and modified, offering suggestions of particular factors that should be considered.

Much of the existent literature about computer science curricula pays insufficient attention to the effects of context. For example, If a professor at a small liberal arts school saw an article about symmetric multiprocessing and wished to introduce it to his students, he would likely be most interested in making sure his students understood the underlying theory—that work can be parallelized for a multiprocessor machine, and that multiprocessing brings certain advantages and disadvantages. He could add his new ideas to his class with relatively little effort. On the other hand, if a professor at a large research university saw the same article and wished to add it to the curriculum at her school, she would likely be more interested in specific applications of multiprocessing, might have her students program a project that parallelized an algorithm, and would have a much more difficult time getting the material into the curriculum.

The many factors that influence context make it very likely that curricular development and modification will continue to be conducted on a school-by-school basis. Given this, it is important that computer science educators at each school have a clear understanding of the factors acting on computer science education as a whole and the factors acting on their school. As curricular theorists have framed and discussed many of the questions that should be asked when working with curricula, computer science educators should consider their perspective carefully when modifying their own curricula.

References

About Swathmore CS. (n.d.). *Swarthmore CS home page*. Retrieved December 6, 2010, from http://www.cs.swarthmore.edu/program/about.html

Accredited Programs Search. (n.d.). ABET, Inc. Retrieved November 15, 2010, from http://www.abet.org/AccredProgramSearch/AccreditationSearch.aspx.

Barrow, R. (1984). *Giving Teaching Back to Teachers: a Critical Introduction to Curriculum Theory.* New York: Wheatsheaf.

Best Undergraduate Engineering Programs – Best Colleges – Education – US News. (2010). *U.S. News and World Report*. Retrieved December 6, 2010, from http://colleges.usnews.rankingsandreviews.com/best-colleges/spec-doct-engineering

Best Undergraduate Engineering Programs – Best Colleges – Education – US News. (2010). *U.S. News and World Report*. Retrieved December 6, 2010, from http://colleges.usnews.rankingsandreviews.com/best-colleges/spec-engineering

BHCC. (n.d.). *Bunker Hill Community College*. Retrieved December 6, 2010, from http://www.bhcc.mass.edu/inside.php?navID=468&programID=13&year=2009

Bureau of Labor Statistics. (2008, May). Occupational Employment and Wage Estimates: National Cross-Industry Estimates. *Bureau of Labor Statistics.* Retrieved May 10, 2010 from ftp://ftp.bls.gov/pub/special.requests/oes/oesm08nat.zip.

Carnegie Foundation for the Advancement of Teaching. (n.d.). *Carnegie Foundation for the Advancement of Teaching.* Retrieved December 6, 2010, from http://www.carnegiefoundation.org/

Carnevale, A. P., Smith, N. & Strohl, J., (2010, June). *Help Wanted: Projections of Jobs and Education Requirements Through 2018.* Retrieved from http://cew.georgetown.edu/jobs2018/

Casper College ~ Casper, Wyoming. (n.d.). *Casper College, Casper, Wyoming, USA - Education for a Lifetime!*. Retrieved December 6, 2010, from http://www.caspercollege.edu/computer_science/index.html

Cassel, L., Clements, A., Davies, G., Guzdial, M., McCauley, R., McGettrick, A., … Cross, J. (2008, December). Computer Science Curriculum 2008. *Proceedings from Association for Computing Machinery, IEEE Computer Society*. Retrieved from ACM Digital Library.

Chang, C., Denning, P., Cross, J., Engel, G., Sloan, R., Carver, D., … Wolz, U. (2001, December) Computing Curricula 2001: Computer Science. *Proceedings from Association for Computing Machinery, IEEE Computer Society*. Retrieved from ACM Digital Library

Cohen, A. & Kisker, C. (2009). *The Shaping of American Higher Education: Emergence and Growth of the Contemporary System* (2nd ed.). Jossey-Bass.

Computer Science Major - Mathematical Sciences - Departments - College of Arts and Sciences - Lewis & Clark. (n.d.). *Welcome to Lewis & Clark*. Retrieved December 6, 2010, from http://www.lclark.edu/college/departments/mathematical_sciences/majors/computer_science_major/

Computer Science, Caltech - Requirements. (n.d.). *Computer Science, Caltech*. Retrieved December 6, 2010, from http://www.cs.caltech.edu/academics/opt_info.html

Computing Careers » Software Engineering. (n.d.). *Computing Careers*. Retrieved December 6, 2010, from http://computingcareers.acm.org/?page_id=12

Degree Requirements. (n.d.). *University of Illinois at Urbana-Champaign Computer Science*. Retrieved December 6, 2010, from https://agora.cs.illinois.edu/display/undergradProg/Degree+Requirements

Department of Electrical Engineering and Computer Science - EECS. (n.d.). *University of Central Florida*. Retrieved December 6, 2010, from http://www.eecs.ucf.edu/index.php?id=majorsacademics/undergraduate/computerscience

Dewar, R. B.K., & Schonberg, E., (2008). Computer Science Education: Where Are the Software Engineers of Tomorrow? *Crosstalk: The Journal of Defense Software Engineering January* Software Technology Support Center. Retrieved May 10, 2010, from <http://www.stsc.hill.af.mil/crossTalk/2008/01/0801DewarSchonberg.html>.

Gornitzka, A., Amaral, A., & Kogan, M., (2005). *Reform and Change in Higher Education Analysing Policy Implementation*. Dordrecht: Springer.

Grundy, S., (1987). *Curriculum: Product or Praxis?* London: Falmer.

Hazzan, O., Almstrum, V. L., Guzdial, M. & Petre, M. Challenges to Computer Science Education Research. *Proceedings of SIGCSE, 05,* 191-192. Retrieved from ACM Digital Library.

Informal Guide to Computer Science -- Major Requirements. (n.d.). *Williams College Computer Science Department*. Retrieved December 6, 2010, from http://www.cs.williams.edu/ifg/fullguide_7.html

Irving Valley College Pages - Computer Science. (n.d.). *Irving Valley College - From Possibility to Actuality*. Retrieved December 6, 2010, from http://www.ivc.edu/cis/pages/default.aspx

Lansing Community College. (n.d.). *Lansing Community College - Where Success Begins*. Retrieved December 6, 2010, from http://www.lcc.edu/mathematics/computer_science/curriculum/

Liberal Arts Computer Science Consortium. (2007). A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science. *Journal on Educational Resources in Computing (JERIC)*, 7(2). Retrieved December 6, 2010, from http://cs.wellesley.edu/~pmetaxas/LACS2007report.pdf

Mannheim, K., (1962). *An Introduction to the Sociology of Education*. Ed. W. A. Stewart. London: Routledge & Kegan Paul.

MIT Course Catalog: Department of Electrical Engineering and Computer Science. (n.d.) *MIT*. Retrieved December 6, 2010, from http://web.mit.edu/catalog/degre.engin.elect.html

National Center for Educational Statistics. (see below).

Ni, L., McKlin, T., and Guzdial, M., (2010, March) "How Do Computing Faculty Adopt Curricular Innovations? The Story from Instructors." ACM Digital Library. *Proceedings of SIGCSE '10*, 544-548. Retrieved from ACM Digital Library.

NYU Computer Science Department. (n.d.). *Computer Science Majors*. Retrieved December 6, 2010, from http://cs.nyu.edu/web/Academic/Undergrad/majors.html

Ohio State University. (n.d.). *Undergraduate majors*. Retrieved December 6, 2010, from http://majors.osu.edu/

Program: Computer Science - Oberlin College (n.d.). *Oberlin College.* Retrieved December 6, 2010, from http://catalog.oberlin.edu/preview_program.php?catoid=18&poid=2064&bc=1

Programs: Computer Science. (n.d.). *Rose-Hulman Institute of Technology*. Retrieved December 6, 2010, from www.rose-hulman.edu/Catalog0910/program-cs.htm

Randolph, J. (2007). *Computer science education research at the crossroads: A methodological review of computer science education research: 2000-2005.* Retrieved from http://www.archive.org/details/randolph_dissertation.

Randolph, J., Julnes, G., Sutinen, E., & Lehman, S. (2008). A Methodological Review of Computer Science Education Research. *Journal of Information Technology Education, 7*, 135-162.

Reif, H. L., & Mathieu, R. G., (2009) Global Trends in Computing Accreditation. *Computer* Nov. 2009: 102-04.

Shackelford, R., Cross, J. H., Davies, G., Impagliazzo, J., Kamall, R., LeBlanc, R., … Topi, H. (2005, September) Computing Curricula 2005: The Overview Report. *ACM, AIS, IEEE,* 30 Retrieved from ACM Digital Library.

Soh, L., Samal, A., & Nugent, G. (2007). An Integrated Framework for Improved Computer Science Education: Strategies, implementations, and results. *Computer Science Education*, *17*(1), 59-83.

Stenhouse, L. (1975). *An introduction to curriculum research and development*. London: Heinemann.

Surendra, N. C., & Denton J. W., (2009) Designing IS Curricula for Practical Relevance: Applying Baseball's "Moneyball" Theory. *Journal of Information Systems Education* 20(1). 77-85. Retrieved from WilsonWeb.

Tan, G., & Venables, A. (2008). Survival Mode: The Stresses and Strains of Computing Curricula Review." *Journal of Information Technology Education: Innovations in Practice 7.* 33-43.

Tyler, R. W. (1949). *Basic Principles of Curriculum and Instruction*. Chicago: University of Chicago.

Undergraduate Program, Computer Science Department - USC Viterbi School of Engineering. (n.d.). *Computer Science Department - USC Viterbi School Of Engineering*. Retrieved December 6, 2010, from http://www.cs.usc.edu/current-students/undergrad.html

Yasuhara, K. (2008). *Viewpoints from the Doorstep: Pre-major Interest in and Perceptions of Computer Science.* Retrieved from http://staff.washington.edu/yasuhara//cv/publications/dissertation/Yas08.dissertation-Viewpoints_from_the_doorstep.pdf

A note on the data from the National Center for Education Statistics: The data used in this paper was from the 2003-04 Beginning Postsecondary Students Longitudinal Study, First Follow-up (BPS:04/06). It was created with the aid of the Data Analysis System, http://nces.ed.gov/dasol/. The precise variables used to create each table are given on each table. The variance estimation was Balanced Repeated Replication (BRR).

A note on the school curricula: This data was collected in May, 2010, from the most current curricula listed on each school's Web site. The Carnegie classification data and ABET accreditation data were collected in November, 2010, from the Web sites of the Carnegie Foundation and ABET, respectively. The liberal arts colleges and community colleges were selected as a broad sample of their respective categories, covering various sizes and locations; the large research universities chosen were the two largest private and public universities by single campus size; and the top tier schools were selected as a broad sample of schools highly ranked by US News.

Category	**Community Colleges**			
School	Casper College	Lansing Community College	Bunker Hill Community College	Irvine Valley College
Carnegie data				
Level	2-year	2-year	2-year	2-year
Control	Public	Public	Public	Public
Enrollment	3,799	19,471	7,821	9,865
Classification	Category	Category	Category	Category
Undergrad Instructional Program:	Assoc: Associate's	Assoc: Associate's	Assoc: Associate's	Assoc: Associate's
Graduate Instructional Program:	(not applicable)	(not applicable)	(not applicable)	(not applicable)
Enrollment Profile:	ExU2: Exclusively undergraduate two-year	ExU2: Exclusively undergraduate two-year	ExU2: Exclusively undergraduate two-year	ExU2: Exclusively undergraduate two-year
Undergraduate Profile:	Mix2: Mixed part/full-time two-year	PT2: Higher part-time two-year	PT2: Higher part-time two-year	PT2: Higher part-time two-year
Size and Setting:	M2: Medium two-year	VL2: Very large two-year	M2: Medium two-year	L2: Large two-year
Basic	Assoc/Pub-R-M: Associate's--Public Rural-serving Medium	Assoc/Pub-R-L: Associate's--Public Rural-serving Large	Assoc/Pub-U-MC: Associate's--Public Urban-serving Multicampus	Assoc/Pub-S-MC: Associate's--Public Suburban-serving Multicampus
ABET Accredited	No	No	No	No
Total CS-Related Classes	4	3	5	7
Intro	N/A	N/A	CIT120 Intro to Computer Science & Object Oriented Programming	N/A
			CIT239 Introduction to JAVA	
CS Core Requirements	COSC 1030 Computer Science I	CPSC 230 - Algorithms and Computing w/C++	CIT285 Advanced Java	CS 1 Introduction to Computer Systems 4
	COSC 2030 Computer Science II	CPSC 231 - Computing and Data Structures	CIT242 Data Structures	
	COSC 2150 Computer Organization	CPSC 260 - Computer Science Structures		

Category	**Community Colleges**			
School	Casper College	Lansing Community College	Bunker Hill Community College	Irvine Valley College
Required Choices	One of:	N/A	One of:	Four of:
	COSC 2300 Discrete Structures		MAT291 Linear Algebra	CS 6A Computer Discrete Mathematics I
	COSC 2403 Linux with X-Windows		CIT237 C++ Programming	CS 6B Computer Discrete Mathematics II
	COSC 2404 Java and Java Script Programming			CS 36 C Programming
	COSC 2405 MFC Windows Programming in C++			CS 37 C++ Programming
	COSC 2406 Java Programming			CS 38 WWW/Intranet w/ Java
	COSC 2409 Programming: Topic			CS 50A HTML Programming

Category	**Research Universities**			
School	New York University	University of Southern California	The Ohio State University	University of Central Florida
Carnegie data				
Level	4-year or above	4-year or above	4-year or above	4-year or above
Control	Private not-for-profit	Private not-for-profit	Public	Public
Enrollment	39408	32160	50995	42465
Classification	Category	Category	Category	Category
Undergrad Instructional Program:	A&S+Prof/HGC: Arts & sciences plus professions, high graduate coexistence	Bal/HGC: Balanced arts & sciences/professions, high graduate coexistence	Bal/HGC: Balanced arts & sciences/professions, high graduate coexistence	Prof+A&S/HGC: Professions plus arts & sciences, high graduate coexistence
Graduate Instructional Program:	CompDoc/MedVet: Comprehensive doctoral with medical/veterinary	CompDoc/MedVet: Comprehensive doctoral with medical/veterinary	CompDoc/MedVet: Comprehensive doctoral with medical/veterinary	CompDoc/NMedVet: Comprehensive doctoral (no medical/veterinary)
Enrollment Profile:	MU: Majority undergraduate	MU: Majority undergraduate	HU: High undergraduate	HU: High undergraduate
Undergraduate Profile:	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	FT4/MS/HTI: Full-time four-year, more selective, higher transfer-in	FT4/MS/HTI: Full-time four-year, more selective, higher transfer-in	MFT4/S/HTI: Medium full-time four-year, selective, higher transfer-in
Size and Setting:	L4/HR: Large four-year, highly residential	L4/R: Large four-year, primarily residential	L4/R: Large four-year, primarily residential	L4/NR: Large four-year, primarily nonresidential
Basic	RU/VH: Research Universities (very high research activity)	RU/VH: Research Universities (very high research activity)	RU/VH: Research Universities (very high research activity)	RU/H: Research Universities (high research activity)
ABET Accredited		Yes	Yes	Yes
Total CS-Related Classes	12	23	34	15

Category	Research Universities			
School	New York University	University of Southern California	The Ohio State University	University of Central Florida
Intro	V22.0101 Introduction to Computer Science	CSCI 101L Fundamentals of Computer Programming	CSE 201 Elementary Computer Programming (not needed for major)	COP 3223 Intro to Programming with C
				COP 3330 Intro to OO Programming with Java
		ENGR 102 Engineering Freshmen Academy		COP 3502 Computer Science I
CS Core Requirements	V22.0102 Data Structures	MATH 225 Linear Algebra and Differential Equations	MA 366 Discrete Mathematical Structures I	COP 3503 Computer Science II
	V22.0201 Computer Systems Organization	EE 364 Introduction to Probability and Statistics	MA 566 Discrete Mathematical Structures II	EEL 3801 Computer Organization
	V22.0202 Operating Systems	Engineering and Computer Science	STAT 427 Introduction to Probability and Statistics for Engineering and the Sciences I	COP 3402 Systems Software
	V22.0310 Basic Algorithms	CSCI 102L Data Structures	STAT 428 Introduction to Probability and Statistics for Engineering and the Sciences II	COT 3100 Intro to Discrete Structures
	V63.0121 Calculus I	CSCI 200 Object Oriented Programming	CSE 221 Software Development Using Components,	COP 4331 Procs. for OO Development
	V63.0120 Discrete Mathematics	CSCI 201L Principles of Software Development	CSE 222 Development of Software Components	EEL 4768 Intro to Computer Architecture
		CSCI 271 Discrete Methods in Computer Science	CSE321 Case Studies in Component-Based Software	COP 4020 Programming Languages
		CSCI 303 Design and Analysis of Algorithms	CSE360 Introduction to Computer Systems	COP 4600 Introduction to Operating Systems

Category	Research Universities			
School	New York University	University of Southern California	The Ohio State University	University of Central Florida
		CSCI 377 Introduction to Software Engineering	CSE 541 Elementary Numerical Methods	COT 4210 Discrete Computational Structures
		CSCI 402x Operating Systems	CSE 560 Systems Software Design, Development, and Documentation	
		CSCI 477ab Design and Construction of Large Software Systems	CSE 601 Social and Ethical Issues in Computing	
		EE 101 Introduction to Digital Logic	CSE 625 Introduction to Automata and Formal Languages	
		EE 106Lx Introduction to Computer Engineering/Computer Science	CSE 655 Principles of Programming Languages	
		EE 201L Introduction to Digital Circuits	CSE 660 Introduction to Operating Systems	
Core Requirements (cont.)		EE 357 Basic Organization of Computer Systems	CSE 670 Introduction to Database Systems I	
		MATH 125 Calculus I	CSE 675.01 Introduction to Computer Architecture	
		MATH 126 Calculus II	CSE 680. Introduction to Analysis of Algorithms and Data Structures	
		MATH 226 Calculus III	Elec Eng 206 Switching Circuits Laboratory	
			EE 261 Introduction to Logic Design	
			EE 300 Electrical Circuits	
			EE 309 Electrical Circuits Laboratory	
			EE 320 Electronic Devices and Controls	

Category	Research Universities			
School	New York University	University of Southern California	The Ohio State University	University of Central Florida
			EE 567 Microprocessor/Microcontroller Laboratory I	
Required Choices	Five of:	Four of:	Technical Elective Option (7 or 8 classes, some including not CSE classes):	Three CS electives
	400-level CS classes	CSCI 300, Introduction to Intelligent Agents Using Science Fiction	Software Systems	
	Calc II	CSCI 351, Programming and Multimedia on World Wide Web	Hardware-Software Systems	Two math electives
	Linear Algebra	CSCI 445, Introduction to Robotics	Information Systems	
		CSCI 459, Computer Systems & Applications Modeling Fundamentals	Information and Computation Assurance	
		CSCI 460, Introduction to Artificial Intelligence	Individualized Option:	
		CSCI 464, Foundations of Exotic Computation	Graphics/Animation Track	
		CSCI 480, Computer Graphics	AI Track	
		CSCI 485, File and Database Management	Advanced Studies Track	
Required Choices (cont.)		CSCI 490x, Directed Research	Business Information Systems Track	
		CSCI 499; Special Topics		

Category	**Research Universities**			
School	New York University	University of Southern California	The Ohio State University	University of Central Florida
		EE 450, Introduction to Computer Networks		
		EE 454L, Introduction to Systems Design Using Microprocessors		
		EE 459L		
		EE 465, Probabilistic Methods in Computer Systems Modeling		
		EE 477L		
		EE 490x, Directed Research		
		EE 499; Special Topics		
		MATH 458. Numerical Methods		

Category	**Liberal Arts**			
School	Lewis & Clark College	Williams College	Oberlin College	Swarthmore College
Carnegie data				
Level	4-year or above	4-year or above	4-year or above	4-year or above
Control	Private not-for-profit	Private not-for-profit	Private not-for-profit	Private not-for-profit
Enrollment	3259	2050	2857	1474
Classification	Category	Category	Category	Category
Undergrad Instructional Program:	A&S-F/NGC: Arts & sciences focus, no graduate coexistence	A&S-F/SGC: Arts & sciences focus, some graduate coexistence	A&S-F/SGC: Arts & sciences focus, some graduate coexistence	A&S-F/NGC: Arts & sciences focus, no graduate coexistence
Graduate Instructional Program:	Postbac-Prof/Other: Postbaccalaureate professional (other dominant fields)	Postbac-A&S: Postbaccalaureate, arts & sciences dominant	S-Postbac/Other: Single postbaccalaureate (other field)	(not applicable)
Enrollment Profile:	MU: Majority undergraduate	VHU: Very high undergraduate	VHU: Very high undergraduate	ExU4: Exclusively undergraduate four-year
Undergraduate Profile:	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in
Size and Setting:	S4/HR: Small four-year, highly residential	S4/HR: Small four-year, highly residential	S4/HR: Small four-year, highly residential	S4/HR: Small four-year, highly residential
Basic	Bac/A&S: Baccalaureate Colleges--Arts & Sciences	Bac/A&S: Baccalaureate Colleges--Arts & Sciences	Bac/A&S: Baccalaureate Colleges--Arts & Sciences	Bac/A&S: Baccalaureate Colleges--Arts & Sciences
ABET Accredited	No	No	No	No
Total CS-Related Classes	11	8	7	9
Intro	CS 171: Computer Science I	CS 134 Introduction to Computer Science		CPSC 021 – Intro to CS

Category	**Liberal Arts**			
School	Lewis & Clark College	Williams College	Oberlin College	Swarthmore College
	CS 172: Computer Science II	CS 136 Data Structures and Advanced Programming.		
CS Core Requirements	Computational Mathematics - CS 230	CS 237 Computer Organization	CSCI 210: Computer Organization	CPSC 035 – Data Structures and Algorithms
	Computer Architecture and Assembly Languages - CS 277	CS 256 Algorithms	CSCI 275: Programming Abstractions	CPSC 097 – Advanced Topic
	Algorithm Design and Analysis - CS 383	CS 334 Principles of Programming Languages	CSCI 280: Introduction to Algorithms	
	Statistical Concepts and Methods - Math 255	CS 361 Theory of Computation	CSCI 383: Theory of Comp Science	
Required Choices	Five of:	Two of:	Three of:	One of:
	CS 363: Operating Systems	315 : Bioinformatics and Biological Physics	311 - Databases	CPSC 033 – Computer Organization
	CS 367: Computer Graphics	323: Software Engineering	331 - Compilers	CPSC 052 – Principles of Computer Organization
	CS 369: Artificial Intelligence	336: Computer Networks	307 - Programming Languages	
	CS 373: Programming Language Structures	337: Introduction to Computer Science	342 - Computer Networks	One of:
	CS 393: Computer Networks	338: Parallel Processing	341 - Operating Systems	CPSC 037 – Structure and Interpretation of Computer Programs

Category	**Liberal Arts**			
School	Lewis & Clark College	Williams College	Oberlin College	Swarthmore College
	CS 465: Theory of Computation	371: Computer Graphics	357 - Computer Graphics	CPSC 075 – Principles of Compiler Design and Construction
	CS 467: Advanced Computer Graphics	373: Artificial Intelligence	343 - Information Security	
	CS 487: Advanced Algorithms	432: Operating Systems	364 - Artificial Intelligence	One of:
	CS 488: Software Development	434: Compiler Design	317 - Computer Architecture	CPSC 041 – Algorithms
	CS 495: Topics in Computer Science		333 - Natural Language Processing	CPSC 046 – Theory of Computation
Required Choices (cont.)			347 - Software Engineering	
			365 - Advanced Algorithms	Three of (no overlap w/ previous):
				CPSC 040 - Graphics
				CPSC 041 - Algorithms
				CPSC 044 - Databases
				CPSC 045 - OS
				CPSC 046 – Theory of Computation
				CPSC 052 - Architecture
				CPSC 063 - AI
				CPSC 065 – Natural Language Processing
				CPSC 067 – Information Retrieval
				CPSC 072 – Computer Vision

Category	**Liberal Arts**			
School	Lewis & Clark College	Williams College	Oberlin College	Swarthmore College
				CPSC 075 – Compiler Design
				CPSC 081 – Adaptive Robotics
				CPSC 082 – Mobile Robotics
				CPSC 087 – Distributed Computing

Category	**Top Tier**			
School	California Institute of Technology	Rose-Hulman Institute of Technology	Massachusetts Institute of Technology	University of Illinois at Urbana-Champaign
Carnegie data				
Level	4-year or above	4-year or above	4-year or above	4-year or above
Control	Private not-for-profit	Private not-for-profit	Private not-for-profit	Public
Enrollment	2171	1904	10320	40687
Classification	Category	Category	Category	Category
Undergrad Instructional Program:	A&S+Prof/HGC: Arts & sciences plus professions, high graduate coexistence	(Special focus institution)	Bal/HGC: Balanced arts & sciences/professions, high graduate coexistence	Bal/HGC: Balanced arts & sciences/professions, high graduate coexistence
Graduate Instructional Program:	Doc/STEM: Doctoral, STEM dominant	(Special focus institution)	CompDoc/NMedVet: Comprehensive doctoral (no medical/veterinary)	CompDoc/MedVet: Comprehensive doctoral with medical/veterinary
Enrollment Profile:	MGP: Majority graduate/professional	VHU: Very high undergraduate	MGP: Majority graduate/professional	MU: Majority undergraduate
Undergraduate Profile:	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	(Special focus institution)	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in	FT4/MS/LTI: Full-time four-year, more selective, lower transfer-in
Size and Setting:	S4/HR: Small four-year, highly residential	(Special focus institution)	L4/HR: Large four-year, highly residential	L4/R: Large four-year, primarily residential
Basic	RU/VH: Research Universities (very high research activity)	Spec/Engg: Special Focus Institutions--Schools of engineering	RU/VH: Research Universities (very high research activity)	RU/VH: Research Universities (very high research activity)
ABET Accredited	No	Yes	Yes	No
Total CS-Related Classes	21	21	13	18

Category	Top Tier			
School	California Institute of Technology	Rose-Hulman Institute of Technology	Massachusetts Institute of Technology	University of Illinois at Urbana-Champaign
Intro	CS 1 Intro. to Computation	CSSE 120 Introduction to Software Development		CS 125 Intro to Computer Science
	CS 2 Intro. to Programming Methods	CSSE 220 Object-Oriented Software Development		
CS Core Requirements	Ma/CS 6 a Intro. to Discrete Math	CSSE 230 Data Structures and Algorithm Analysis	6.042J Mathematics for Computer Science	CS 173 Discrete Structures
	CS 21 Decidability and Tractability	CSSE 232 Computer Architecture	6.005 Elements of Software Construction	CS 210 Ethical and Professional Issues in CS
	CS 24 Intro. to Computing Systems	CSSE 304 Programming Language Concepts	6.006 Introduction to Algorithms	CS 225 Data Structure and Software Principles
	CS 38 Introduction to Algorithms	CSSE 332 Operating Systems	6.033 Computer Systems Engineering	CS 231 Computer Architecture I
	E 11 Technical Communication	CSSE 333 Database Systems - SQL	6.034 Artificial Intelligence	CS 232 Computer Architecture II
	E 10	CSSE 371 Software Requirements and Specification	6.046 Design and Analysis of Algorithms	CS 241 System Programming
		CSSE 374 Software Architecture and Design		CS 242 Programming Studio

Category	Top Tier			
School	California Institute of Technology	Rose-Hulman Institute of Technology	Massachusetts Institute of Technology	University of Illinois at Urbana-Champaign
		CSSE 473 Design and Analysis of Algorithms		CS 373 (was CS 273) Theory of Computation
		CSSE 474 Theory of Computation		MATH 461 Probability Theory
		ECE 130 Introduction to Logic Design		MATH 415 Applied Linear Algebra
		ECE 332 Computer Architecture II		
		MA 275 Discrete and Combinatorial Algebra I		(assuming CS track)
		MA 221 Differential Equations and Matrix Algebra		CS 357 Numerical Methods I
CS Core Requirements (cont.)		MA 375 Discrete and Combinatorial Algebra II		CS 421 Programming Languages and Compilers
		MA 381 Introduction to Probability with Applications to Statistics		CS 473 Algorithms
Required Choices	Major project or thesis	Four CS electives	One of:	Two of:
		CS electives	18.03 Differential Equations	400-level classes
	63 CS units		18.06 Linear Algebra	
				Senior Project

Category	**Top Tier**			
School	California Institute of Technology	Rose-Hulman Institute of Technology	Massachusetts Institute of Technology	University of Illinois at Urbana-Champaign
	36 units in MA, ACM, CS		One of:	
			6.141 Robotics Science and Systems	One, two-class specialization, selected from:
			6.172 Performance Engineering of Software Systems	Systems
			6.035 Computer Language Engineering	Databases
			6.813 User Interface Design & Implementation	Graphics
				Human-Computer Interaction
			Two advanced subjects	Languages
			Three exploratory classes	Artificial Intelligence
				Security
				Networking

Purpose (1st school) 2004: job skills			
	No	Yes	
	(%)	(%)	
Estimates			
Total	52.9698	47.0302	
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	54.7658	45.2342	
Humanities	58.1276	41.8724	
Social/behavioral sciences	60.9317	39.0683	
Life sciences	73.9506	26.0494	
Physical sciences	57.9494	42.0506	
Math	‡	‡	
Computer/information science	43.6935	56.3065	
Engineering	59.5767	40.4233	
Education	58.1749	41.8251	
Business/management	62.1498	37.8502	
Health	49.8663	50.1337	
Vocational/technical	40.8887	59.1113	
Other technical/professional	50.2618	49.7382	
Standard Error (BRR)			
Total	1.0618	1.0618	
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	1.9981	1.9981	
Humanities	2.6549	2.6549	
Social/behavioral sciences	2.9651	2.9651	
Life sciences	5.3386	5.3386	
Physical sciences	7.0806	7.0806	
Math	‡	‡	
Computer/information science	4.8075	4.8075	
Engineering	6.0671	6.0671	
Education	3.5816	3.5816	
Business/management	2.1134	2.1134	
Health	2.5411	2.5411	
Vocational/technical	3.3525	3.3525	
Other technical/professional	2.9354	2.9354	

Weighted Sample sizes(n/1,000s)			
Total	2215.6374		
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	437.5167		
Humanities	190.6211		
Social/behavioral sciences	118.5513		
Life sciences	32.6606		
Physical sciences	28.8961		
Math	‡		
Computer/information science	75.1797		
Engineering	39.9486		
Education	107.6768		
Business/management	236.3672		
Health	184.3408		
Vocational/technical	87.747		
Other technical/professional	150.2807		

	Importance 2004: being financially well off	
	No	Yes
	(%)	(%)
Estimates		
Total	23.3	76.7
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	21.9	78.1
Humanities	29.2	70.8
Social/behavioral sciences	25.9	74.1
Life sciences	24.6	75.4
Physical sciences	29.3	70.7
Math	32.8	67.2
Computer/information science	21.9	78.1
Engineering	24.6	75.4
Education	32.8	67.2
Business/management	16.6	83.4
Health	21.8	78.2
Vocational/technical	19.8	80.2
Other technical/professional	22.2	77.8
Standard Error (BRR)		
Total	0.46	0.46
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	1.15	1.15
Humanities	1.7	1.7
Social/behavioral sciences	1.37	1.37
Life sciences	2.12	2.12
Physical sciences	3.88	3.88
Math	5.64	5.64
Computer/information science	2.77	2.77
Engineering	3.64	3.64
Education	2.2	2.2
Business/management	1.08	1.08
Health	1.58	1.58
Vocational/technical	2.18	2.18
Other technical/professional	1.61	1.61

Weighted Sample sizes(n/1,000s)		
Total	3,832.70	
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	605.9	
Humanities	391.6	
Social/behavioral sciences	337.8	
Life sciences	124.3	
Physical sciences	61.1	
Math	24.4	
Computer/information science	124.6	
Engineering	102.9	
Education	231.8	
Business/management	495.8	
Health	286.7	
Vocational/technical	136.6	
Other technical/professional	294.5	

	Importance 2006: being financially well off	
	No	Yes
	(%)	(%)
Estimates		
Total	32.2	67.8
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	29.6	70.4
Humanities	39.8	60.2
Social/behavioral sciences	38.4	61.6
Life sciences	42.4	57.6
Physical sciences	37	63
Math	54.2	45.8
Computer/information science	28.6	71.4
Engineering	26.9	73.1
Education	47	53
Business/management	25.2	74.8
Health	30	70
Vocational/technical	27.6	72.4
Other technical/professional	31.4	68.6
Standard Error (BRR)		
Total	0.51	0.51
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	1.4	1.4
Humanities	1.46	1.46
Social/behavioral sciences	1.57	1.57
Life sciences	2.57	2.57
Physical sciences	3.69	3.69
Math	5.81	5.81
Computer/information science	2.74	2.74
Engineering	2.17	2.17
Education	2.21	2.21
Business/management	1.25	1.25
Health	1.68	1.68
Vocational/technical	2.27	2.27
Other technical/professional	1.75	1.75

Weighted Sample sizes(n/1,000s)		
Total	3,832.70	
Major when last enrolled (12 cat) in 2006		
Undeclared or not in a degree program	605.9	
Humanities	391.6	
Social/behavioral sciences	337.8	
Life sciences	124.3	
Physical sciences	61.1	
Math	24.4	
Computer/information science	124.6	
Engineering	102.9	
Education	231.8	
Business/management	495.8	
Health	286.7	
Vocational/technical	136.6	
Other technical/professional	294.5	

	Highest degree expected, 2003-04		
	No degree or certificate	Certificate	Associate's degree
	(%)	(%)	(%)
Estimates			
Total	1.0629	4.7807	9.4589
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	1.5981	6.4293	11.1388
Humanities	0.3603	3.7838	5.521
Social/behavioral sciences	0.1152	2.3711	3.1417
Life sciences	0.0419	1.3079	2.0076
Physical sciences	0.5701	6.9495	3.2244
Math	0	0	1.5744
Computer/information science	0.9088	3.0311	10.7322
Engineering	0.0384	4.2655	6.5924
Education	0.0693	0.7648	6.3743
Business/management	0.3081	2.5209	6.3522
Health	0.8716	4.3515	13.8551
Vocational/technical	1.4166	7.2528	13.5457
Other technical/professional	0.3184	5.0515	10.6365
Standard Error (BRR)			
Total	0.1291	0.2716	0.4273
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	0.366	0.6351	1.1459
Humanities	0.0947	0.6343	0.9297
Social/behavioral sciences	0.0669	0.4578	0.5394
Life sciences	0.051	0.7399	0.7433
Physical sciences	0.228	1.7637	1.4054
Math	0	0	1.5447
Computer/information science	0.526	0.9621	2.1058
Engineering	0.0416	1.4246	1.4648
Education	0.0833	0.2493	1.3507
Business/management	0.1284	0.3872	0.7107
Health	0.5382	0.7571	1.3036
Vocational/technical	0.6282	1.4184	2.4919
Other technical/professional	0.1135	0.7905	1.3625

Bachelor's degree (%)	Post-BA/post-MA certificate (%)	Master's degree (%)	Doctoral degree (%)	Professional degree (%)
31.706	0.414	35.4061	11.9008	5.2705
33.8251	0.2798	31.4976	11.4101	3.8213
31.6523	0.231	40.3548	13.8309	4.266
23.8314	0.305	40.4562	20.1498	9.6296
17.1797	0.0118	24.537	28.8159	26.0983
19.8097	0.8179	32.9621	26.5399	9.1265
25.7062	0	46.5176	22.5246	3.6771
38.9682	0.8753	33.1643	9.6087	2.7115
27.1385	0.2074	41.9053	17.5515	2.301
26.5975	0.9907	51.2915	10.5596	3.3524
31.2546	0.4757	45.3982	9.7773	3.913
32.7144	0.6366	31.8397	10.2105	5.5206
35.8378	0.1861	30.6882	6.2482	4.8245
32.638	0.2019	37.7924	9.0413	4.3199
0.5651	0.0674	0.5524	0.3831	0.2159
1.4082	0.1142	1.3027	1.2236	0.4865
1.8745	0.072	1.8537	1.0436	0.5519
1.5248	0.1558	1.509	1.282	0.8735
1.9666	0.0118	2.0986	2.1995	1.999
3.4771	0.631	4.5504	3.172	2.006
4.7096	0	5.6543	4.4413	1.5671
2.9391	0.81	2.6192	2.8014	1.0011
2.6985	0.2217	3.1214	2.1001	0.8094
1.9639	0.3797	2.1369	1.0812	0.7551
1.4023	0.1482	1.546	0.7512	0.5058
1.6161	0.2498	1.5531	1.1167	0.7684
2.7555	0.2046	3.2022	1.2225	1.332
1.6488	0.114	1.9291	1.1839	0.6333

Weighted Sample sizes(n/1,000s)			
Total	3832.6796		
Major when last enrolled (12 cat) in 2006			
Undeclared or not in a degree program	605.9075		
Humanities	391.6472		
Social/behavioral sciences	337.7749		
Life sciences	124.2804		
Physical sciences	61.1235		
Math	24.3861		
Computer/information science	124.6423		
Engineering	102.887		
Education	231.751		
Business/management	495.8342		
Health	286.6998		
Vocational/technical	136.578		
Other technical/professional	294.5007		

Highest degree expected, 2006									
	No degree	Undergrad certificate	AA/AS	BA/BS	Post-BA/BS	MA/MS	Post-MA/MS	Professional	Doctoral
	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)
Estimates									
Total	4.8528	6.4415	12.2807	31.5536	0.7614	29.6356	0.4284	5.7049	8.3411
Major when last enrolled (12 cat) in 2006									
Undeclared or nondegree	4.9395	9.7276	13.7125	33.5139	0.6624	25.9679	0.2729	4.9293	6.274
Humanities	2.0765	3.908	7.154	34.656	0.9945	34.3373	0.2565	6.8606	9.7566
Social/behavioral sciences	1.5239	2.4374	6.2595	21.0731	0.8412	35.7214	0.8669	11.7991	19.4775
Life sciences	0.159	2.3815	1.7728	17.3755	1.384	25.2292	1.0564	29.2344	21.4073
Physical sciences	2.8292	8.0858	4.3676	14.1006	0	30.0167	0.1798	12.7415	27.6787
Math	0.1612	1.9872	1.2661	20.9466	0.128	54.1184	0	4.6413	16.7512
Computer/information science	4.4723	2.4442	17.5124	39.8122	1.2553	25.3483	0.1861	0.741	8.2281
Engineering	3.1662	6.0711	5.1796	30.3135	0.5809	37.8199	0.4218	4.44	12.0069
Education	2.1011	2.8212	6.1554	25.3227	0.7718	51.0096	0.9034	1.9383	8.9766
Business/management	1.8626	2.0717	8.8933	37.8647	0.8668	37.7785	0.5409	3.6361	6.4854
Health	1.3706	6.8479	19.6969	31.1506	0.9399	27.5177	0.7408	4.4601	7.2754
Vocational/technical	3.6452	10.2472	19.0992	36.9081	1.2111	21.6918	0.0776	4.527	2.5928
Other technical/professional	3.4174	6.8306	12.9864	35.5368	0.6908	30.9277	0.4034	4.214	4.9928

Standard Error (BRR)									
Total	0.3443	0.3267	0.4746	0.5227	0.099	0.5068	0.0542	0.2092	0.2868
Major when last enrolled (12 cat) in 2006									
Undeclared or nondegree	0.976	0.8183	1.0403	1.3135	0.1916	1.2328	0.1308	0.6563	0.7533
Humanities	0.436	0.5227	0.9116	1.7746	0.4115	1.7937	0.1019	0.7677	0.9113
Social/behavioral sciences	0.354	0.4519	0.8515	1.2346	0.2426	1.4181	0.236	0.9925	1.2141
Life sciences	0.1583	0.9921	0.6409	2.3664	0.7658	2.058	0.5774	2.1893	1.8671
Physical sciences	0.9924	1.8887	1.7028	2.6405	0	4.6995	0.1802	2.3943	3.7098
Math	0.1669	1.4185	1.105	4.4468	0.1277	5.4665	0	2.1021	3.5384
Computer/information science	1.701	0.7798	2.4885	3.0884	0.4678	2.6497	0.1417	0.4549	1.6785
Engineering	1.545	3.0294	1.0867	3.1368	0.2812	3.089	0.2753	1.055	1.881
Education	0.6458	0.693	1.1819	1.9494	0.295	2.3102	0.3032	0.5528	1.1654
Business/management	0.3594	0.381	0.929	1.4106	0.368	1.4317	0.1468	0.4554	0.7403
Health	0.324	0.9794	1.9046	1.8549	0.3777	1.5231	0.3402	0.6918	0.7925
Vocational/technical	0.7489	1.6966	2.9384	2.893	0.4569	2.5906	0.0751	0.9949	0.8337
Other technical/professional	0.7205	1.0224	1.4835	1.8087	0.2827	1.7558	0.1913	0.599	0.7073

Weighted Sample sizes(n/1,000s)	
Total	3832.68
Major when last enrolled (12 cat) in 2006	
Undeclared or nondegree	605.9075
Humanities	391.6472
Social/behavioral sciences	337.7749
Life sciences	124.2804
Physical sciences	61.1235
Math	24.3861
Computer/information science	124.6423
Engineering	102.887
Education	231.751
Business/management	495.8342
Health	286.6998
Vocational/technical	136.578
Other technical/professional	294.5007

The Relationship between Educational Theory and Educational Practice in Undergraduate Computer Science Education

For decades, educational researchers have argued that the gap between educational theory and practice must be bridged. Researchers and practitioners have attempted to define roles that they should play in the educational process, and entire subfields have sprung up in an attempt to translate educational theory to educational policy. Despite these efforts, the gap has persisted across all levels of education.

The theory-practice gap, while significant, is even more complicated at the level of higher education: each field has its own educationalists who sit between the practitioners of the field and the more general educational theorists. The classic gap between theory and practice is, in higher education, fracturing into two: a gap between the specialized educationalists of each field and the general educational community, and a gap between those educationalists and their corresponding practitioners.[1] These gaps make it more difficult to effect change in educational practice, and serve to further remove practitioners and theorists from one another. The gaps also lead to redundancy in educational research, with educationalists of each field attempting to solve many of the same problems.

Computer science educational researchers are beginning to argue against this isolated approach (see Almstrum in Hazzan et al., Hazzan in Almstrum et al.). However, they are still early in the process of response, and it seems likely that their response will only address the isolation of computer science educators from the larger educational community. While addressing that isolation is beneficial, an approach that integrated the existing educational structure with the work of both educational researchers and educationalists from different

[1] A note on terminology used in this paper: Educational researchers are those who produce and discuss general educational theory. 'Educationalist' refers to those in each field who write about the educational process in that field. 'Practitioner' refers to the professors, adjuncts, or TAs who actually teach the classes.

disciplines would allow the harnessing of the benefits presented by the current model while ameliorating many of its shortcomings.

Guzdial, in Almstrum et al., summarizes the problem:

> The real challenge to computing education is to avoid the temptation to re-invent the wheel. Computers are a revolutionary human invention, so we might think that teaching and learning about computers requires a new kind of education. That's completely false: The basic mechanisms of human learning haven't changed in the last 50 years.
>
> Too much of the research in computing education ignores the hundreds of years of education, cognitive science, and learning sciences research that have gone before us.

This is not the first time that computer science has had difficulties with reinventing the wheel—in "One Man's View of Computer Science," Richard Hamming states:

> "[…] one of my major complains about the computer field is that whereas Newton could say 'If I have seen a little farther than others, it is because I have stood on the shoulders of giants,' I am forced to say, 'Today we stand on each other's feet.' Perhaps the central problem we face in all of computer science is how we are to get to the situation where we build on top of the work of others rather than redoing so much of it in a trivially different way."

Randolph et al. (2008), in a review of computer science education research, offer more explanation of the drawbacks of ignoring the work of educational researchers. Their paper concludes that computer science educationalists often perform their research with practices that the educational and psychological research communities have established as ineffective, thus making the educationalists' results insufficiently validated. In his dissertation, Randolph (2006) also examines computer science education research and looks for the presence of literature reviews in articles. His findings are inconclusive, but he believes that the majority of computer science education research is conducted without a proper preceding literature review. If Randolph is correct, the problem noticed by Guzdial is broader than he describes it: computer

science educators, along with ignoring the work of educational theorists and cognitive researchers, also ignore each other's work.

The solution supported by Guzdial, Randolph, and others is that computer science educationalists should build their work atop that of others. Petre and Hazzan, among others, write as though computer science educationalists must "borrow" ideas from other disciplines and the question at hand has become that of what should be borrowed and how it should be borrowed (Petre in Hazzan et al., Hazzan in Almstrum et al.). This argument, though, promotes a unidirectional flow of ideas and research—the educationalists would borrow without offering in return the ideas built on the work of the lenders.

Instead of solving the problem, then, this merely moves it: the educationalists will not have to reinvent the wheel, but without the educationalists' researchers being published in general educational journals, the educational researchers will. That in and of itself is not terrible—educational researchers have been working as they have for many years and have produced a great deal of useful research and theory in that time. However, the two theory-practice gaps magnify the problem: computer science educationalists will not only be borrowing from educational researchers, behavioral researchers, and psychologists, but also from educationalists in many other fields. The sheer number of educationalists in different fields makes it critical that these educationalists build on each other's work rather than duplicating it.

The discussion of whether computer science educationalists properly build on the work of education theorists or each other is moot if computer science educational practitioners do not pay attention to the work of those educationalists. Unfortunately, there is little data on the factors that contribute to computer science educational practice. Tutty et al. (2008) discuss the teaching of IT academics, citing an article stating that many academics in accounting are unaware of the

majority of educational methods. Almstrum's (2005) statement that colleagues may "convey the attitude that educational research is not *real* research" suggests that computer science educational practice is not significantly shaped by educational theory either. Ni et al. (2010) discuss the implementation of curricular innovation in computer science, finding that even after a new computer science course was presented to several professors in a series of workshops, the majority of the professors did not even adopt ideas from the course. In order for computer science educationalists to play a larger role in determining the practice of computer science education, it is important that these educationalists work to see their ideas implemented.

Computer science educationalists recognize the challenge of selecting and adapting others' research to better their own, but have not discussed the difficulties of how to best present their own results and discoveries to ease adaptation by others. This is less of an issue when results are presented for implementation by another educator in a similar situation, but if results are being shared with a larger audience, it becomes much more important. The recommendations made by Randolph et al. provide a base for such a presentation—results will not be useful to others if they cannot be firmly established as valid under certain conditions—but the later steps are not as clear. The question of how to present new ideas or results so that others will adapt them is an open one in the world of education, but there are several reasons why a three-tiered system such as that which exists can be beneficial to the interactions between theory and practice.

One common criticism of theorists by practitioners is that the theorists are often too removed from the actual practice, and thus their theory is often inapplicable to real situations. Educationalists, often being practitioners in their respective fields, are better informed about the realities of practice in those fields. If they were to work to communicate those realities to the

educational theorists, the theorists would learn how their ideas played out in practice and the educationalists would be able to advantage themselves of educational theory that was produced by better-informed theorists.

Similarly, one difficulty in the implementation of educational theory by practitioners is that actual implementation is a complicated process—theory must often be promoted, modified, or even required by policymakers before it is enacted in schools. The teacher-researcher movement supports the idea that each individual practitioner should assess the suitability of new theory to fit their needs and modify theory, but this is often difficult in a higher education environment, where each professor is expected to be a research in his or her own field and does not have time to also be a research-practitioner. Meanwhile, those who favor centralized direction of theory are faced with the decentralized authority and different contexts of higher education, making it difficult to prescribe a theory that would both work and be adopted in every context.

The educationalists of each field, however, in standing between the practitioners and the educational theorists, can act both as teacher-researchers and as educational theorists. In computer science, there are educationalists at large research universities, small liberal arts schools, and almost every situation in between. If these educationalists avail themselves of educational theory and translate it so that it fits their situations, they can then present it to other practitioners in more concrete forms, thus greatly increasing the chances of its adoption.

Although computer science is only now developing its process of educational research, its situation is far from unique. Computer science educationalists should look to the experiences of educationalists in other fields to gain insight into how they can best build their own system of educational research. While creating such a system, however, they should do more than just

borrow the work of others; they should contribute their own work to the educational conversation. If their experiences are then helpful to other educationalists and educational theorists, they will have done more than just dealing with the gap between educational theory and practice—they will have helped to close it.

References

Almstrum, V. L., Hazzan, O., Ginat, D., & Morley, T. Import and Export to/from Computing Science Education: The Case of Mathematics Education Research. *Proceedings of ITiCSE'02*, 193-194. Retrieved from ACM Digital Library.

Hamming, R. W. (2007). One man's view of computer science. In *ACM Turing award lectures*. Retrieved from ACM Digital Library.

Hazzan, O., Almstrum, V. L., Guzdial, M. & Petre, M. Challenges to Computer Science Education Research. *Proceedings of SIGCSE, 05*, 191-192. Retrieved from ACM Digital Library.

Ni, L., McKlin, T., and Guzdial, M., (2010, March) "How Do Computing Faculty Adopt Curricular Innovations? The Story from Instructors." ACM Digital Library. *Proceedings of SIGCSE '10*, 544-548. Retrieved from ACM Digital Library.

Randolph, J. (2007). *Computer science education research at the crossroads: A methodological review of computer science education research: 2000-2005.* Retrieved from http://www.archive.org/details/randolph_dissertation.

Randolph, J., Julnes, G., Sutinen, E., & Lehman, S. (2008). A Methodological Review of Computer Science Education Research. *Journal of Information Technology Education, 7*, 135-162.

Tutty, J., Sheard., J., & Avram, C. (2008). Teaching in the current higher education environment: perceptions of IT academics. *Computer Science Education* 18 (3), 171-185.

Programming Project Postmortem

My initial plan for this project was to write a program that would calculate shortest distances for round-the-world air itineraries. Round-the-world plane tickets are offered by the three major airline alliances and allow the purchaser to plan their own itinerary and visit a number of destinations. These tickets are priced by total flight distance - Star Alliance, for example, offers 29,000, 34,000, and 39,000 mile tiers. The purchaser can then visit up to a certain number of cities, typically fifteen or less, as long as his or her itinerary satisfies certain other minor conditions. To save money, the purchaser would want to travel the shortest possible route, but the creation of such a route is difficult. I wanted to write a program that would ask the user for the cities that they wished to visit and then give them the shortest itinerary that visited those cities.

From a programming perspective, the problem has several aspects. The airlines offer data about their flight schedules; this data must be processed to convert it to a form such that a traveling salesman algorithm may be run on it to find the shortest itinerary. After finding that itinerary, the data would then be presented to the user. I wanted to get the airlines' flight data, which is available as a PDF, process it with regular expressions to get all the distances of the existing flights, run Djikstra's algorithm on the resulting data to create a complete graph of cities, use the Lin-Kernighan heuristic to find the shortest path that visited all the cities, and output the data as a web page that drew the route with Google Maps.

Unfortunately, I bit off a good bit more than I could chew, and only a minority of this project ended up being finished. I started by implementing Djikstra's algorithm, as I thought it was the simplest part of the project. It's currently the only part that works correctly and is finished. When I started to write the code for the Lin-Kernighan heuristic, I discovered that many

of the descriptions of the heuristic that I was using left various implementation details up in the air. I made assumptions for the details, but in the process of programming and debugging, I found that my assumptions were flawed. Due to some terrible coding style choices, it took me far longer than it should have to discover this, and by the time I realized what I had done wrong, a lot of the structure of the code depended on things that would have to be changed.

Before I talk in more detail about the traveling salesman portion of the project, I will introduce the problem in a little more detail. The traveling salesman problem is one of the more studied problems in computer science. It can be modeled with a traveling salesman who needs to visit several cities and return to his starting city in the shortest possible distance. It has a number of applications-the most obvious having to do with routing, but others include circuit design and even telescope aiming (Applegate et al.). The problem can either be exactly solved or approximately solved; as I had read that exact solvers were more complex to implement, I decided to write an approximate solver-with the small size of the problems I anticipated solving, I would have been able to run the solver several times from different starting tours, which would have produced an optimal tour with high probability. I chose to implement the Lin-Kernighan heuristic as it seemed to strike a good balance between performance and complexity.

The Lin-Kernighan heuristic (which is different from the Kernighan-Lin algorithm) was devised by Lin and Kernighan in 1973. It is a tour improvement method, meaning it starts with a random tour visiting all the cities then improves that tour as much as it can. The simplest such method is 2-opt, which swaps pairs of edges in the tour so as to shorten it. A tour that cannot be further improved by 2-opt swaps is called 2-optimal. 3-opt is a similar method, making swaps of trios of edges. 3-opt can overcome some local minima that 2-opt cannot, but it is only another step. Lin and Kernighan noticed that while 3-opt produced shorter tours than 2-opt, their returns

diminished when expanding to 4-opt, and the computation required to calculate optimal tours increased significantly with the larger number of swaps (Lin and Kernighan).

Their idea was to create a k-opt algorithm that could make 2 swaps, 3, or more as the situation demanded. Essentially, as long as the total gain of the swaps was positive, they could add more swaps to their current attempt. This allowed them to make some swaps that added to the tour length, enabling them to overcome local minima better than other methods. Even in their paper, where the heuristic is first presented, they immediately present refinements that allow it to find better solutions while examining fewer tours and edges (Lin and Kernighan).

When I started working on this project, I expected that the implementation of the traveling salesman heuristic itself would not be very difficult—after all, it was a problem that had a great deal of discussion; there were bound to be several implementations I could look at to see how it was done. I thought that the project was more about the sum of the parts, and the traveling salesman implementation itself would be a matter of looking at a discussion of the algorithm and implementing some pseudocode.

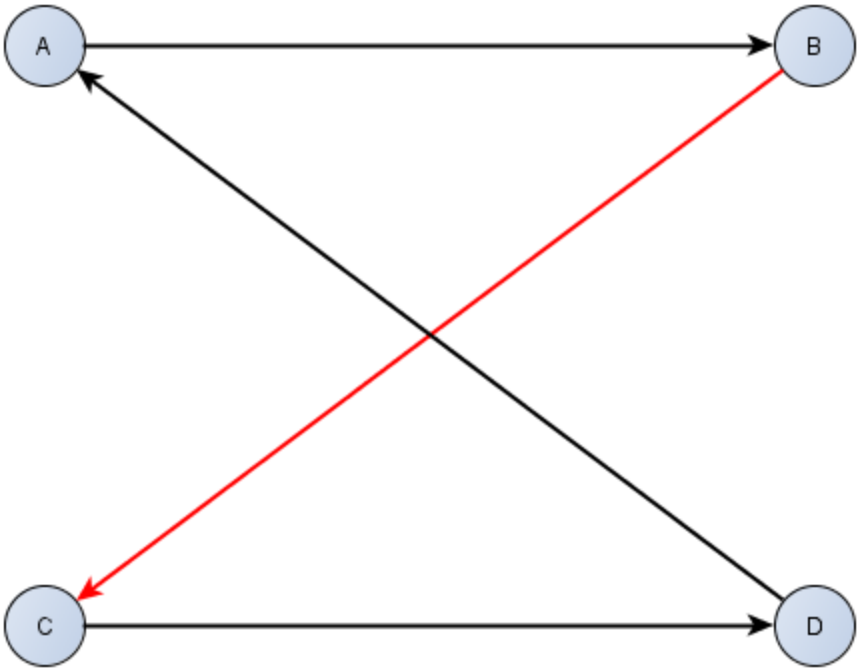
Once I had written code for Djikstra's algorithm, I already had some structure to my project, so when I started reading about Lin-Kernighan, I thought about how I could implement it on top of what I already had. That was probably my first mistake-the structure that worked for Djikstra's did not work as well for Lin-Kernighan.

I started writing code for Lin-Kernighan without fully understanding the algorithm-although I thought I did—and I didn't understand how I would deal with the implementation details that the various papers and pseudocodes did not really mention. In fact, I didn't realize that some of those implementation details, particularly those having to do with roads, existed at all. There are indeed many papers that discuss the Lin-Kernighan heuristic, but many of them do

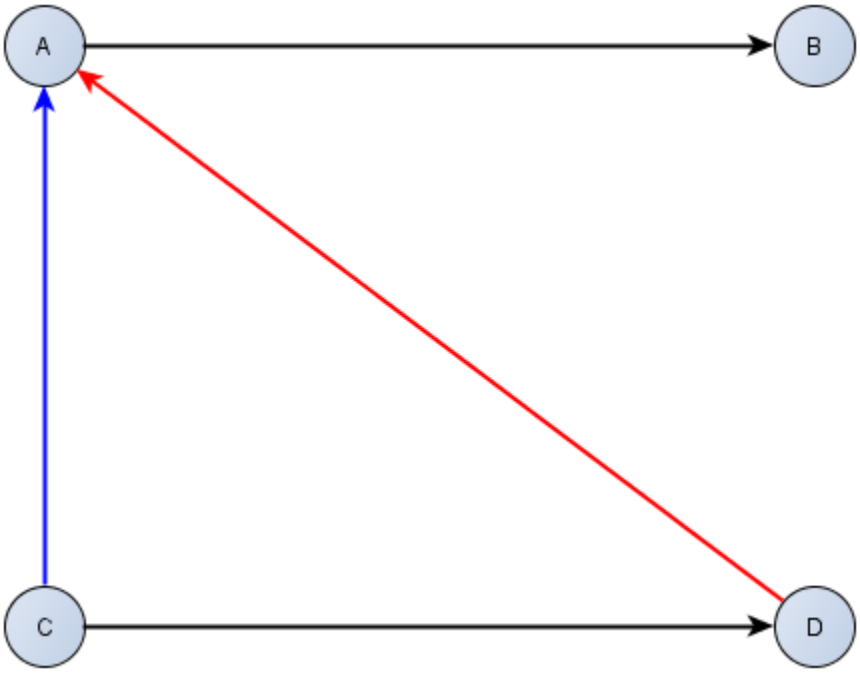
not offer implementation details or pseudocode. Worse yet, as the Lin-Kernighan heuristic is dated, most of the papers that do offer implementations have made some attempt to improve or modify it; this makes it difficult to look at another paper when having difficulties understanding one author's explanation.

I'd never tried to program a project of any magnitude before, and while I'd been told time and time again that it's impossible to keep track of everything in your head, I still tried to. This wasn't a problem when implementing Djikstra's as the code was small enough that I could keep it in my head, but once I started with the TSP, it became a serious issue. I should have decided what each class was meant to do first, but I wasn't totally sure what I needed each class to do-and as I implemented more of Lin-Kernighan, what I thought I needed from each class changed. Since I didn't have tests, or even a clear specification for classes, the interfaces changed; I attempted to change the rest of the code to keep up with that, but quite a few bugs were created.

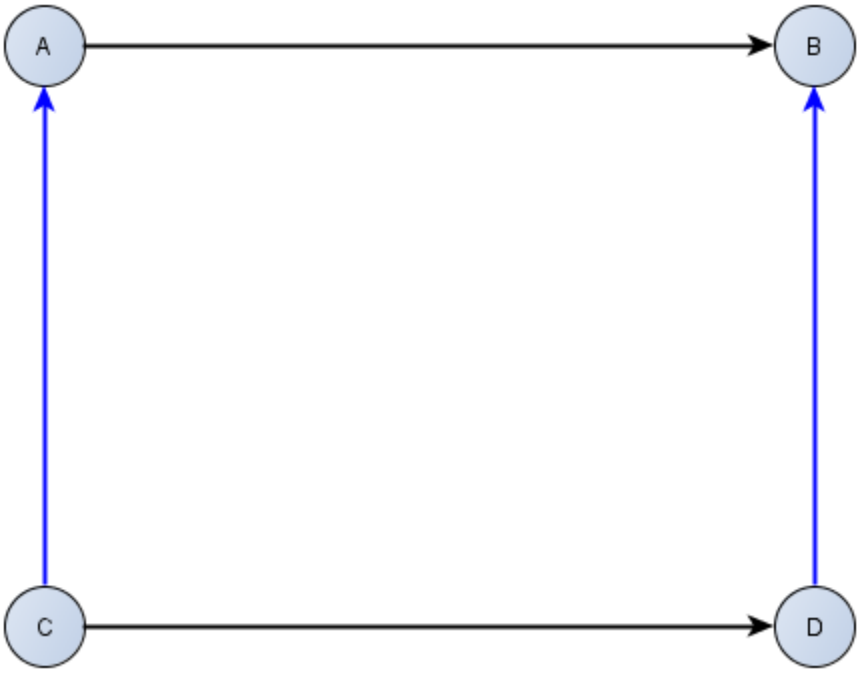
It didn't actually take that long to get some code for Lin-Kernighan, but there were many bugs in it at first. I had to get rid of quite a few of them before the program worked enough for me to figure out that there were flaws in its implementation of Lin-Kernighan itself. The pseudocodes of Lin-Kernighan that I had seen were usually not clear on exactly how they implemented the connections between cities. I had initially used directional Road objects as that was intuitive for implementing Djikstra's, but using directional objects when doing Lin-Kernighan creates issues when closing tours-see below.



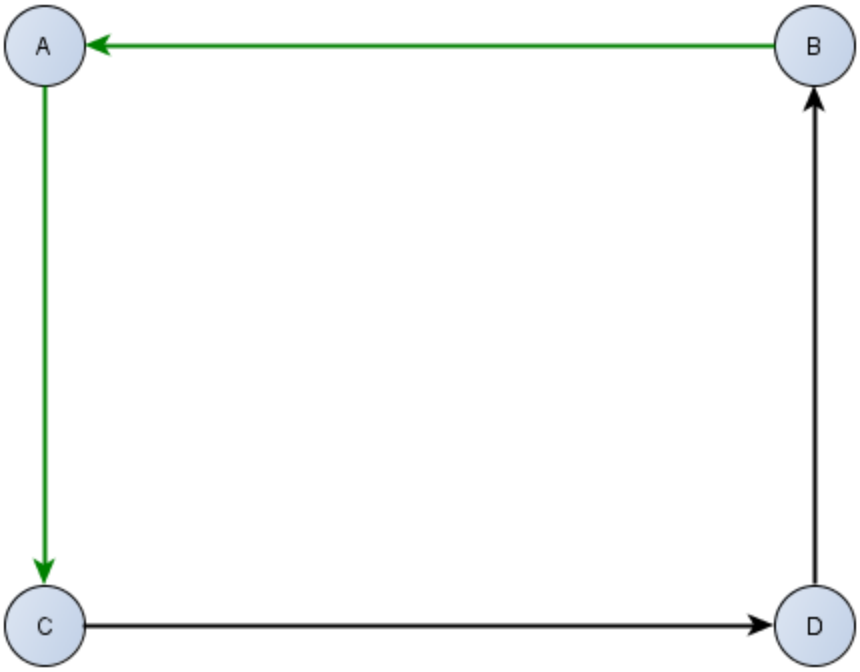
The above is a four-city instance. The highlighted red path from B to C is being cut. Depending on the implementation, either a path from C to A or from B to D would be added; the directional issues appear in either case. I'll add a path from C to A.



After adding the blue path from C to A, there are three paths connected to A. Either A to B or D to A can be removed, but removing A to B closes off part of the tour and isolates it from the last city. Thus, I next remove D to A. As I will have modified roads going to all the cities at this point, I will then close the tour by connecting D to B.



At this point, there are two paths entering B and leaving it, and two paths leaving C and none entering it. Half of the tour should be reversed to make the tour valid.



The green paths were reversed, making the tour valid. Unfortunately, the structure I had in my program did not allow for easy reversing of directional paths, so I sought another solution.

My initial solution was to replace the Roads with nondirectional Connector objects (see second version of ‘tsp.py’). However, after reworking much of the rest of the code to handle those objects, I discovered that modifying tours with nondirectional edges was not that much easier than modifying tours with directional edges-it became more difficult to figure out where exactly I was when I was walking the tour to cut and add edges.

I next attempted to implement a simpler 2-opt solver. When building it on top of the Connectors, I had difficulty figuring out which ends had to be reconnected, and when building it on top of the Roads, I ran into the same issue with closing tours creating cycles. At that point, the semester was starting to end, and I ran out of time to get the code working.

Designing a program that does one thing is significantly different from designing a program that performs several functions that do not necessarily depend on each other. When

designing this project, I did not have a process that dealt with this well: I thought about the suitability of the structure for only the part of the project that I was currently working on, but then once I started working on another part, I assumed that the structure I had would work for that as well. I should have either planned for the needs of each part - which would have been exceedingly difficult - or not planned on any carryover between the parts, instead devising new structures that stored and provided access to the data in such ways that it was easy to deal with it in that part, then programming functions that translated the data between the parts. Had I done that, changing the data structure for the traveling salesman solver would have only affected the solver, not the code for Djikstra's algorithm or the rest of the problem. In retrospect, I should have stopped trying to use the same Road structure I had for Djikstra's altogether and instead written something else that better fit the needs of Lin-Kernighan.

The bigger problem, though, happened when trying to implement the Lin-Kernighan heuristic. Implementing an algorithm that one doesn't understand is always going to be challenging, but I think that there are several things I could have done differently that would have ameliorated some of the difficulties. First, my understanding of exactly how the heuristic worked changed several times, and each time I tried to change my code to fit what I thought should be happening. However, each time, I had to reread the code, try to figure out what I had thought was happening, and try to change it to fit what I now thought should happen. Writing a clear specification for each function and class helps when writing new functions that build on those others, but when writing code for a poorly-understood algorithm, I think it is very important to describe not only what the code should do, but how it is doing that.

Properly dividing the code would also have made it easier to modify and fix, although there were several factors that made it difficult to do this. Subdividing an algorithm that is poorly

understood often leads to—and did, in this case, lead to—functions that weren't divided on a proper boundary and have to clumsily modify data that should be modified elsewhere. In attempting to implement the Lin-Kernighan heuristic, though, each pseudocode I saw divided the problem differently and I didn't understand why they divided it where they did. I again was too wedded to the divisions I first made and had difficulty significantly changing how the code worked.

I see flexibility as one of the cornerstones of computer science. In the course of working on this project, there were several instances where I considered throwing out much of the code and starting over, but each time, remembering how long it had taken to create the existent code, I decided to keep the code there was and make it work somehow. By the time I realized I would be better off throwing it out, I did not have the time to rewrite it and the numerous issues it had prevented me from getting a real solver working at all.

References

Applegate, D. L., Bixby, R., Chvátal, V., & Cook, W. (2006). *The traveling salesman problem: a computational study*. Princeton: Princeton University Press.

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21, 498–516. Retrieved December 6, 2010, from the JSTOR database.

Helsgaun, K. (2000). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Researc*h, 126(1), 106-130. Retrieved December 6, 2010, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.4908&rep=rep1&t

Johnson, David S., & McGeoch, L. A. (1997). The Traveling Salesman Problem: A Case Study in Local Optimization. In E. H. Aarts, & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (p. 215–310). Chichester: Wiley.

Round the World Fare – Star Alliance. (n.d.). *Star Alliance*. Retrieved December 6, 2010, from http://www.staralliance.com/en/fares/round-the-world-fare/

This is a Python implementation of the Lin-Kernighan heuristic for solving the Euclidean traveling salesman problem. Various parts of this code do not work as intended; other parts of the code may not work at all.

It was programmed for Python 2.7 and was not tested on other versions of Python.

Richard Scruggs
December, 2010

==========

This program is intended to solve a particular class of traveling salesman problems - mapping the shortest path for a round the world air itinerary.

There are two different versions of this code: the older version uses directional Road objects, the newer uses nondirected Connector objects. Neither works, but they don't work for different reasons, so I have included both.

When you run tests.py, it will ask you if you'd like to load a file or use random data. I've provided two data files, 'three' and 'twelve', each with the indicated number of cities. The order in which they appear in the file (and thus the default tour the program will create) is also the order of the shortest tour - for 'three', the tour distance is about 57; for 'twelve' it's about 151.

If you'd like to create your own small instance, there are two methods to do so: The Windows port of Concorde provides a GUI that can be used to add cities to a map and save that map, which this program can then read. Alternately, the city data files may be edited by hand. The file format the program uses is very simple and is described at readCities in 'utils.py'.

'djikstra.py' also offers a test function that times how long it takes to run Djikstra's algorithm on some cities.

```
from utils import *
import copy
from itertools import permutations

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

"""This is the module that implements the TSP solving functions. It is the
later version, so its functions deal with Connectors rather than with Roads."""

class TSP():
    """This class ties together the TSP. It loads all the precomputed data and
    contains methods to do all the preprocessing before handing it off to a
    TSPSolver. After it is solved, it then tidies the result and prints it
    out."""
    #At least, that's what it would ideally do.

class TSPSolver():
    """This is a superclass for a solver for TSPs. It lays out all the methods
    that a solver should implement. At present, it only exists to raise errors
    if those methods are not implemented."""
    def __init__(self):
        raise NotImplementedError()
    def solve(self, cities):
        """Each Solver should be able to create a solution from just a list of
        cities (with populated Reachables)."""
        raise NotImplementedError()

class BruteForce(TSPSolver):
    """Solves a TSP by checking all possible tours to find the shortest. This is
    O(n!), so it is essentially useless on tours of more than eight or nine
    cities."""
    def __init__(self):
        pass
    def solve(self, cityList):
        if isinstance(cityList, Cities):
            cityList = cityList.cityList
        perms = permutations(cityList)
        shortest = Tour(perms.next())
        for perm in perms:
            tempTour = Tour(perm)
            if tempTour < shortest:
                shortest = tempTour
        return shortest

class TwoOpt(TSPSolver):
    """This TSP solver attempts to solve the problem with 2-opt - removing two
    roads and replacing them with two others that connect those four cities
    differently in an attempt to create a shorter tour."""
    def __init__(self):
        pass
    def solve(self, cities):
        """Returns a 2-optimal tour of the given cities."""
        if isinstance(cities, Tour):
            T = copy.copy(cities)
        else:
            T = Tour(cities)
        return self.twoOptSolve(T)
    def twoOptSolve(self, T):
        """Calculates the two-optimal path for the given tour."""
```

```
            initialDistance = T.updateDistance()
            for optOne in T.insideRoads[:]:
                for optTwo in T.insideRoads[:]:
                    if shares(optOne, optTwo):
                        continue
                    #I'm being clever. If a given pair is valid, the other one isn't
                    T.remove(optOne)
                    T.remove(optTwo)
                    swapOne = optOne.endpoints[0].getRoad(optTwo.endpoints[0])
                    swapTwo = optOne.endpoints[1].getRoad(optTwo.endpoints[1])
                    if not (swapOne in T or swapTwo in T):
                        T.add(swapOne)
                        T.add(swapTwo)
                        if T.isValid():
                            if T.updateDistance() < initialDistance:
                                return twoOptSolve(T)
                            T.remove(swapOne)
                            T.remove(swapTwo)
                            T.add(optOne)
                            T.add(optTwo)
                            continue
                        T.remove(swapOne)
                        T.remove(swapTwo)
                    swapOne = optOne.endpoints[0].getRoad(optTwo.endpoints[1])
                    swapTwo = optOne.endpoints[1].getRoad(optTwo.endpoints[0])
                    T.add(swapOne)
                    T.add(swapTwo)
                    if T.updateDistance() < initialDistance:
                        return twoOptSolve(T)
                    T.remove(swapOne)
                    T.remove(swapTwo)
                    T.add(optOne)
                    T.add(optTwo)

    class SimpleLK(TSPSolver):
        """This TSP solver is a simple version of the Lin-Kernighan heuristic,
        without any of the enhancements discussed by Lin and Kernighan in their
        orignial paper. I did add one enhancement mentioned by Helsgaun - when it
        finds a new shorter tour, it is supposed to start solving that tour
        immediately."""
        def __init__(self):
            """This Java-ish model probably isn't the best."""
            pass
        def solve(self, cities):
            """Returns the shortest tour that visits all the given cities."""
            if isinstance(cities, Tour):
                T = copy.copy(cities)
            else:
                T = Tour(cities)
            if not T.isValid():
                print "Invalid Tour."
                return T
            return self.recursiveSearch(T, T.insideRoads[:])

        def recursiveSearch(self, T, cutRoads, visited=[], gain=0):
            """Performs a recursive Lin-Kernighan search to find the shortest tour.

            The search is performed like this: Each road i is removed, one at a
            time. I get a list of roads shorter than i from i's destination to any
            other cities. I try adding each of those roads to the tour; their
            destinations then have three roads connected to them. I check to see if
            the two pre-existing roads can be cut and the other ends of those roads
            can be connected to the starting city - the source of i - to produce a
            valid tour. If it can and the valid tour is shorter, the search is
            restarted with that tour; if the valid tour is longer, the search
            deepens, trying to find another cut and add that will produce a shorter
            tour."""
            for cutRoad in cutRoads:
                gain += cutRoad.distance
                if visited:
                    addRoads = T.getAddRoads(visited[-1], cutRoad.distance)
                else :
                    addRoads = T.getAddRoads(cutRoad.endpoints[0], cutRoad.distance)
```

```
                T.remove(cutRoad)
                visited.append(cutRoad.otherEnd(visited[-1]))
                for x in range(len(addRoads)):
                    tempAdd = addRoads.pop()
                    gain -= tempAdd.distance
                    nextCuts, shortCuts = getCuts(T, visited, gain, tempAdd)
                    #getCuts adds tempAdd to T.
                    if shortCuts:
                        return self.recursiveSearch(T, T.insideRoads)
                    if nextCuts:
                        U = copy.copy(T)
                        recurseBest = self.recursiveSearch(U, nextCuts, visited, gain)
                        if recurseBest < T:
                            return self.recursiveSearch(recurseBest, recurseBest.insideRoads,
                        gain += tempAdd.distance
                    T.remove(tempAdd)
                    visited.pop()
                gain -= cutRoad.distance
                T.add(cutRoad)
                visited.pop()
                visited.pop()
            if not T.isValid():
                #I'm under the assumption here that I only need to return a valid
                #tour if I was passed one. Otherwise, I need to return something
                #that definitely won't be passed up as it's longer than what there was.
                return float('Inf')
            return T #if our starting tour is best.

    def getCuts(tour, visited, gain, addRoad):
        """Returns the possible cuts from addRoad's destination, paired with their
        closing roads. If a closing road is found that shortens the tour, it returns
        an empty list of cuts and a True. The tour will have been modified in the
        testing process - and we just jump out when we see the new one is shorter -
        so it doesn't need to return the particular cut or close."""
        #AddRoad should not be added to the tour yet or we might return it as a
        #possible cut. Note that we add it here and don't remove it after.
        cuts = []
        closeDistances = []
        for road in tour.insideRoads:
            if addRoad.otherEnd(visited[-1]) in road.endpoints:
                if (road.endpoints[0] not in visited) and (road.endpoints[1] not in visited):
                    cuts.append(road)
        #Now, cuts is a list of roads from addRoad.destCity to anywhere we haven't
        #modified.
        tour.add(addRoad)
        visited.append(addRoad.otherEnd(visited[-1]))
        #For each road in cuts, I try to remove it and close the tour.
        for road in cuts:
            tour.remove(road)
            visited.append(road.otherEnd(visited[-1]))
            closeRoad = visited[-1].getRoad(visited[0])
            tour.add(closeRoad)
            visited.append(closeRoad.otherEnd(visited[-1]))
            cuts.remove(road)
            if tour.isValid():
                if (road.distance + closeRoad.distance < gain):
                    return [],True
                else:
                    pass
                cuts.append(road)
            tour.add(road)
            visited.pop()
            tour.remove(closeRoad)
            visited.pop()
        tour.updateDistance
        return cuts, False
```

```
#!/usr/bin/env python
import random
from math import sqrt
import bisect

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

"""This module provides many utility functions that the rest of the code uses.
This is the later version of this file, so many of the functions it implements
deal with Connectors rather than with Roads."""

def measureDistance(cityA, cityB):
    """Returns the distance between city A and city B using the Pythagorean
    theorem."""
    return sqrt((max(cityA.y, cityB.y) - min(cityA.y, cityB.y))**2 + \
        (max(cityA.x, cityB.x) - min(cityA.x, cityB.x))**2)

def shares(connectorA, connectorB):
    """Returns True if the connectors share an endpoint, False otherwise."""
    for endpoint in connectorA.endpoints:
        if endpoint in connectorB.endpoints:
            return True
    return False

def createConnections(cities, connectedness):
    """Creates connections among the cities. Connectedness should be between 0
    and 1, and controls how many other cities each city is connected to.
    Destroys the passed list or Cities object and returns a new one."""
    #Note that I want each city connected to at least one other. That won't
    #always happen with this.
    if isinstance(cities, Cities):
        cities = cities.cityList
    newCities = []
    while cities:
        city = cities.pop()
        for destination in cities:
            if (connectedness > random.random()):
                tempDistance = measureDistance(city, destination)
                city.reachables.append(Road(city, destination, [city, destination], tempDistance
                destination.reachables.append(Road(destination, city, [destination, city], tempD
        newCities.append(city)
    return Cities(newCities)

def readCities(filename):
    """Reads cities from the given filename. Returns a Cities object.
    As far as input format, I'm trying to follow the format used by the Windows
    version of Concorde, for simplicity, but I want more things than it expects,
    so I'm tacking things on in places it doesn't notice.

    The file is expected to be formatted like:
    6 0 0.75
    21.239837 89.716312 New York
    17.987805 79.787234 Atlanta
    17.987805 68.794326 Philadelphia
    25.406504 79.787234 Chicago
    25.406504 69.858156 Houston
    22.256098 59.929078 Miami

    with the first line being: cityCount segmentCount connectedness
    I'll read cityCount and use it to not read segments; I do not yet read or
    write segments.
    Connectedness is something I tacked on; Concorde ignores it; my code does
    not require it but will use it if present.

```

```
        I believe that the default Concorde save format is .qs - although it doesn't
        use that extension and I'm not sure what exactly the specifications of the
        format are.
        """
        cities = Cities([])
        f = open(filename)
        line = f.readline()
        line = line.split()
        if len(line) > 2:
            connectedness = float(line[-1])
        else:
            connectedness = 0
        cityCount = int(line[0])
        for x in range(cityCount):
            line = f.readline()
            if not line:
                break
            #The below call is being clever. If lineParse gets a name from the line,
            #the string is never used; otherwise the string is used as a city name.
            cities.append(City(*lineParse(line, str(x))))
        f.close()
        return cities, connectedness

    def lineParse(line, name):
        """Parses the given line to get values that can be used to create a City.
        Uses defaultName for the city's name if unable to find one on the line.
        >>> lineParse("17.987805 79.787234 Atlanta", "0")
        ([17.987805, 79.787234], 'Atlanta')

        >>> lineParse("17.987805 79.787234", "0")
        ([17.987805, 79.787234], '0')
        """
        words = line.split()
        x = float(words[0])
        y = float(words[1])
        if len(words) > 2:
            name = " ".join(words[2:])
        return [[x,y], name]

    def YesOrNo(question, default=-1):
        """Asks the user the passed question. Returns True if their answer is yes,
        False if it isn't. If default is passed, returns it on null input."""
        answer = raw_input(question)
        while True:
            answer = answer.strip()
            if answer == "":
                if default != -1:
                    return default
                print "Your answer was not parsed properly. Please enter 'yes' or 'no'."
                answer = raw_input(question)
                continue
            answer = answer.lower()
            if answer[0] == 'y':
                return True
            if answer[0] == 'n':
                return False
            print "Your answer was not parsed properly. Please enter 'yes' or 'no'."
            answer = raw_input(question)

    class Cities:
        """A class for a group of cities. This allows various list-like methods of
        accessing the group, but is mostly just a wrapper around a list of City
        objects."""
        def __init__(self, cityList):
            self.cities = {}
            for city in cityList:
                self.cities[city.name] = city
            self.cityList = cityList
        def __getitem__(self, key):
            if type(key) is int:
                return self.cityList[key]
            elif type(key) is str:
                return self.cities[key]
        def __len__(self):
            """Returns the number of cities in the given Cities object."""
            return len(self.cityList)
        def __setitem__(self, key, value):
            raise NotImplementedError("Use .append() to add to a CityMap.")
```

```
155         def __delitem__(self, key):
156             """Deletes the given item from the collection of cities. Does not tidy
157             reachables, however."""
158             if type(key) is str:
159                 self.cityList.remove(self.cities[key])
160                 del self.cities[key]
161             else:
162                 del self.cities[key.name]
163                 self.cityList.remove(key)
164         def __iter__(self):
165             """Returns an iterator over the cities in the given Cities object."""
166             return self.cityList.__iter__()
167         def __contains__(self, item):
168             """Returns True if the group of cities has a particular city instance,
169             or if a group of cities has a city with the given name."""
170             return (item in self.cityList) or (item in self.cities.keys())
171         def __str__(self):
172             "Returns a string suitable for exporting with city coordinates."
173             outStr = ""
174             for city in self.cityList:
175                 outStr += str(city)
176                 outStr += "\n"
177             return outStr
178         def append(self, item):
179             """Adds the given City to the Cities object."""
180             if isinstance(item, City):
181                 self.cities[item.name] = item
182                 self.cityList.append(item)
183             else:
184                 raise ValueError("Cannot add non-City to Cities.")
185         def pop(self):
186             """Returns and removes the first City in the given Cities object."""
187             city = self.cityList[0]
188             self.remove(city)
189             return city
190         def remove(self, item):
191             """Removes the given City from the Cities object either by name or by
192             instance."""
193             self.__delitem__(item)
194         def index(self, item):
195             """Returns the index of the given City in the underlying list of the
196             Cities object."""
197             if type(item) is str:
198                 return self.cityList.index(self.cities[item])
199             return self.cities.index(item)
200         def hasOrphans(self):
201             """Returns True if there are any cities that are not connected to any
202             others. Does not test for disconnected clusters."""
203             for city in self.cityList:
204                 if len(city.reachables) == 1:
205                     return True
206             return False
207 
208     class Tour(Cities):
209         """A Tour is a structure that stores cities and provides methods to make the
210         implementation of the traveling salesman solver easier.
211 
212         A Tour is an itinerary that visits each city once and returns to its source.
213         This class stores a list of Connectors that are used in that itinerary and
214         the Connectors that connect the cities in the tour but are not used in the
215         itinerary. It provides methods to add and remove Connectors, and check to
216         see if a given tour is valid."""
217         def __init__(self, cityList, randomize=False):
218             """Creates a Tour from the passed list of cities. At this point, it's
219             assumed that each city has a fully populated list of reachables."""
220             #The actual 'order' of the tour is clumsy to access at present.
221             self.cities = {}
222             if isinstance(cityList, Cities):
223                 cityList = cityList.cityList #that could be named better...
224             for city in cityList:
225                 self.cities[city.name] = city
226             self.cityList = cityList
227             if randomize:
228                 self.shuffle()
229             self.insideRoads, self.outsideRoads = getRoads(self.cityList)
230             self.updateDistance()
231         def __cmp__(self, other):
232             """Compares two Tours by their distance. Assumes both are valid."""
```

```
            return cmp(self.distance, other)
        def __str__(self):
            """Returns a string that describes the tour. Expects the tour to be valid.
            >>> T
            <utils.Tour instance at 0x0279FF80>
            >>> print T
            A tour of 3 cities, with total distance 57.
            The cities, in the order they are visited:
            New York, Boston, Washington, New York.

            (note that the first city is at both ends; this was not by design but
            I thought it looked better and decided to keep it.)
            """
            outStr = "A tour of %d cities, with total distance %d.\n" % (len(self.cityList), self.di
            outStr += "The cities, in the order they are visited:\n"
            insideCopy = self.insideRoads[:]
            startCity = insideCopy[0].endpoints[0]
            walkCity = insideCopy[0].endpoints[1]
            walkRoad = insideCopy[0]
            visited = [startCity]
            while walkCity != startCity:
                visited.append(walkCity)
                insideCopy.remove(walkRoad)
                for road in insideCopy:
                    if walkCity in road.endpoints:
                        walkRoad = road
                        found = True
                        break
                walkCity = road.otherEnd(walkCity)
            visited.append(self.insideRoads[0].endpoints[0])
            for city in visited:
                outStr += (city.name + ", ")
            outStr = outStr[:-2] + "."
            return outStr
        def add(self, road):
            """Adds the given road to the tour, removing it from tour.outsideRoads
            and adding it to tour.insideRoads."""
            self.outsideRoads[road.endpoints[0].name].remove(road)
            self.outsideRoads[road.endpoints[1].name].remove(road)
            self.insideRoads.append(road)
        def getRoad(self, cityA, cityB):
            """Returns the road from cityA to cityB."""
            return cityA.getRoad(cityB)
        def isValid(self):
            """Returns True if the tour is valid - that is, if all the cities in it
            are visited exactly once and if it's closed."""
            '''Walking a tour of Connectors is more complicated than walking one of
            Roads. I start with the first road in insideRoads and get a startCity,
            walkCity, and walkRoad from that. I also keep track of the visited
            cities to avoid loops. I look for the other road that contains walkCity,
            set it as walkRoad, and remove the old walkRoad from the list of roads
            to examine. If all roads but the inital one are removed from the list
            after a walk in this fashion, the tour is valid.'''
            insideCopy = self.insideRoads[:]
            startCity = insideCopy[0].endpoints[0]
            walkCity = insideCopy[0].endpoints[1]
            walkRoad = insideCopy[0]
            visited = [startCity]
            while walkCity != startCity:
                found = False
                if walkCity in visited:
                    return False
                visited.append(walkCity)
                insideCopy.remove(walkRoad)
                for road in insideCopy:
                    if walkCity in road.endpoints:
                        walkRoad = road
                        found = True
                        break
                if not found:
                    return False
                walkCity = walkRoad.otherEnd(walkCity)
            return len(insideCopy) == 1
        def getAddRoads(self, city, distance=float('Inf')):
            """Returns the roads from the given city with lengths less than
            distance."""
            addRoads = []
            for road in self.outsideRoads[city.name]:
```

```
311                 if road.distance < distance:
312                     addRoads.append(road)
313                 else:
314                     break
315             return addRoads
316         def printRoads(self):
317             """Prints the roads in the tour. Meant to be used on invalid tours,
318             where str() will fail."""
319             print "Roads in tour:"
320             for road in self.insideRoads:
321                 print road
322             print
323         def shuffle(self):
324             """Shuffles the cityList. Only used at init as the road structure is not
325             changed."""
326             random.shuffle(self.cityList)
327         def remove(self, road):
328             """Removes the given road from the tour, assuming it is present. Does
329             not check to see if the road is in the tour first."""
330             self.insideRoads.remove(road)
331             bisect.insort(self.outsideRoads[road.endpoints[0].name], road)
332             bisect.insort(self.outsideRoads[road.endpoints[1].name], road)
333         def updateDistance(self):
334             """Updates the tour's distance. Returns that distance."""
335             distance = 0
336             for road in self.insideRoads:
337                 distance += road.distance
338             self.distance = distance
339             return distance
340 
341     def getRoads(cityList):
342         """When passed the given cityList, returns two lists: the list of Roads that
343         are needed to connect the given cities in a closed circuit in the given
344         order, and the list of Roads that are not."""
345         outsideRoads = []
346         for city in cityList:
347             outsideRoads.extend(city.reachables)
348         removeDuplicates(outsideRoads)
349         insideRoads = []
350         for x in range(len(cityList) - 1):
351             tempRoad = cityList[x].getRoad(cityList[x+1])
352             insideRoads.append(tempRoad)
353             outsideRoads.remove(tempRoad)
354         tempRoad = cityList[-1].getRoad(cityList[0])
355         insideRoads.append(tempRoad)
356         outsideRoads.remove(tempRoad)
357         outsideDict = {}
358         for city in cityList:
359             outsideDict[city.name] = []
360         for road in outsideRoads:
361             outsideDict[road.endpoints[0].name].append(road)
362             outsideDict[road.endpoints[0].name].append(road)
363     #       bisect.insort(outsideDict[road.endpoints[0].name], road)
364     #       bisect.insort(outsideDict[road.endpoints[1].name], road)
365         return insideRoads, outsideDict
366 
367     def removeDuplicates(duplicatedList):
368         """Removes duplicates from the given list, in place. Assumes the items in
369         the list are comparable, but not hashable."""
370         #from the Python FAQ.
371         duplicatedList.sort()
372         last = duplicatedList[-1]
373         for i in range(len(duplicatedList) -2, -1, -1):
374             if last == duplicatedList[i]:
375                 del duplicatedList[i]
376             else:
377                 last = duplicatedList[i]
378 
379     class DistanceMatrix():
380         """A DistanceMatrix is a matrix that stores the shortest distance from each
381         city to each other city. It is used when running Djikstra's algorithm.
382 
383         The data is not actually stored as a matrix, but as a multidimensional
384         Python dictionary. Each source city maps to a dictionary of destination
385         cities; each destination city maps to a Road that goes from the source to
386         the destination."""
387         def __init__(self, cities):
388             """Creates a distance matrix with data from the given cities. Sets all
```

```
            unknown distances to infinity."""
            self.matrix = {}
            for source in cities:
                self.matrix[source.name] = {}
                for dest in cities:
                    self.matrix[source.name][dest.name] = Road(source, dest, [], float('Inf'))
            for source in cities:
                for road in source.reachables:
                    self.setRoad(source, road.destCity, road)
        def __str__(self):
            """Returns a string representation of the given distance matrix."""
            cityNames = self.matrix.keys()
            cityNames.sort()
            outStr = "\t"
            #first line
            for name in cityNames:
                outStr += name
                outStr += "\t"
            outStr += "\n"
            for source in cityNames:
                outStr += source
                outStr += "\t"
                for dest in cityNames:
                    outStr += "%.2f" % float(self.getDistance(source, dest))
                    outStr += "\t"
                outStr += "\n"
            return outStr
        def getDistance(self, source, dest):
            """Returns the distance from source to dest; source and dest can be
            cities or names of cities."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            return self.matrix[source][dest].distance
        def getRoad(self, source, dest):
            """Returns the road from source to dest. If the matrix is newly created,
            it may return infinity instead of a Road object."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            return self.matrix[source][dest]
        def setRoad(self, source, dest, road):
            """Sets the road from source to destination in the matrix to the given
            road."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            self.matrix[source][dest] = road
        def roadsFrom(self, source):
            """Returns a list of roads leading from source to any other cities."""
            if isinstance(source, City):
                source = source.name
            roadList = self.matrix[source].values()
            return roadList
        def toConnectors(self):
            """Turns all of the Roads in the given matrix to Connectors."""
            sources = self.matrix.keys()
            list.sort(sources)
            for source in sources:
                dests = self.matrix[source].keys()
                for dest in dests:
                    if source < dest:
                        self.matrix[source][dest] = self.matrix[source][dest].toConnector()
                    else:
                        self.matrix[source][dest] = self.matrix[dest][source]
        def values(self):
            """Returns all the destination dictionaries in the given matrix."""
            return self.matrix.values()

    class CityMap(Cities):
        """A CityMap is a class that provides some structure for storing a
        two-dimensional "map" of cities. It stores the size of the map and can print
        a representation of it."""
        def __init__(self, cityList, mapX, mapY, connectedness=None):
            self.connectedness = connectedness
```

```
            self.cities = {}
            for city in cityList:
                self.cities[city.name] = city
            if isinstance(cityList, Cities):
                self.cityList = cityList.cityList #Sorry.
            else:
                self.cityList = cityList #Again, sorry.
            self.x = mapX
            self.y = mapY
        def printMap(self):
            """Constructs and prints a graphical representation of the city
            locations. Does not feature connections on the map yet."""
            printList = []
            for x in range(self.y):
                printList.append("." * self.x)
            for city in self.cityList:
                printList[city.y] = printList[city.y][:city.x] + "X" + printList[city.y]
[city.x+1:]
            for line in printList:
                print line
        def __str__(self):
            "Returns a string suitable for exporting with city coordinates."
            outStr = ""
            outStr += "Map Width:\t%d\nMap Height:\t%d\n" % (self.x, self.y)
            outStr += "Cities:\n"
            for city in self.cityList:
                outStr += str(city)
                outStr += "\n"
            return outStr

    def getDistance(path):
        """Returns the length of the given path - a list of cities. Expects each
        city in the path to have a distance to at least the next one."""
        distance = 0
        for x in range(len(path) - 1):
            distance += path[x].getRoad(path[x+1]).distance
        return distance

    class Connector:
        """A class for Connectors. A Connector is an undirected segment from one
        city to another."""
        def __init__(self, cityA, cityB, path, distance=None):
            """Inititalizes a connector. Sorts the passed cities and reverses the
            path as necessary to ensure a connector between two cities will be the
            same regardless of order."""
            self.endpoints = [cityA, cityB]
            self.endpoints.sort()
            if self.endpoints[0] != path[0]:
                path.reverse()
            self.path = path
            if not distance:
                distance = getDistance(path)
            self.distance = distance
        def __cmp__(self, other):
            """Compares two Connectors by distance."""
            return cmp(self.distance, other)
        def __str__(self):
            """Prints a string representation of the connector."""
            #From Jim
            return " %s - %s (%.2f) " % \
                (self.endpoints[0].name, self.endpoints[1].name, self.distance)
        def __eq__(self, other):
            """Returns True if the other object is a connector that connects the
            same two cities.

            Note that if there is a Connector A connecting New York and Chicago,
            and a Road B connecting New York and Chicago, A == B will evaluate to
            True, but B == A will not."""
            if isinstance(other, Connector):
                return self.endpoints == other.endpoints
            return False
        def otherEnd(self, city):
            """When given one end of a connector, returns the other. Doesn't check
            to see if the passed city _is_ one end of the connector."""
            if city is self.endpoints[0]:
                return self.endpoints[1]
            return self.endpoints[0]

```

```
    class Road(Connector):
        """A class for roads; a road is a directed path from one city to another."""
        def __init__(self, cityA, cityB, path, distance=0):
            """Creates a Road. The two cities are the endpoints, path is a list of
            cities that are traveled through. It cannot be null; should always
            contain at least two cities - one iff cityA is cityB."""
            #all the different accessors for endpoints were added at different times
            #as I thought I needed them. Most are no longer used.
            self.endpoints = [cityA, cityB]
            self.endpoints.sort()
            self.source = [cityA, cityA.name]
            self.destination = [cityB, cityB.name]
            self.sourceName = cityA.name
            self.destName = cityB.name
            self.sourceCity = cityA
            self.destCity = cityB
            self.path=path
            if not distance:
                distance = getDistance(path)
            self.distance = distance
        def __eq__(self, other):
            """Returns True if the two roads are the same. See the note at
            Connector.__eq__."""
            return isinstance(other, Road) and self.path == other.path
        def __str__(self):
            """Prints a string representation of the given road."""
            #From Jim
            return " %s -> %s (%.2f) " % \
                (self.sourceName, self.destName, self.distance)
        def toConnector(self):
            """Returns a Connector with the same endpoints as the given road."""
            return Connector(self.sourceCity, self.destCity, self.path, self.distance)

    class City:
        """A class for cities. A city has a name and x-y coordinates as a location,
        although the coordinates are not used for anything other than to create
        roads as yet. Cities also store a list of the other cities that are
        reachable from them and which roads lead to those other cities."""
        def __init__(self, location, name=None, continent=None):
            """Creates a City. Location is an (x,y) tuple of coordinates, name is
            self-explanatory, and continent has to do with a specific constraint of
            the problem that is not implemented.

            >>> testCity = City((5,10), "New York")
            >>> print testCity
            City New York at (5, 10), connected to 1 cities: New York.
            """
            #Note that this init routine adds a road from the new city to itself.
            self.x = location[0]
            self.y = location[1]
            self.continent = continent
            self.name = name
            self.reachables = [Road(self, self, [self], 0)]
        def __cmp__(self, other):
            """Compares two City objects by name."""
            return isinstance(other, City) and cmp(self.name, other)
        def __str__(self):
            """Returns a string representation of the given city and which other
            cities are reachable from it."""
            reachableString = ""
            for road in self.reachables:
                reachableString += road.destName
                reachableString += ", "
            reachableString = reachableString[:-2]
            outStr = "City %s at (%d, %d), connected to %d cities: %s." % \
            (self.name, self.x, self.y, len(self.reachables), reachableString)
            return outStr
        def getRoad(self, destination):
            """Returns the Connector from this city to the given destination."""
            for connector in self.reachables:
                if destination in connector.endpoints:
                    return connector
            raise Exception("There is no connector from %s to %s." % (str(self), str(destination)))
        def reachableCities(self):
            """Returns a list of cities that are reachable from the current city.
            Does not work with the Connector interface currently in place."""
            cityList = []
            for road in self.reachables:
```

```
622                 cityList.append(road.destCity)
623             return cityList
```

```
#!/usr/bin/env python
from utils import *
from itertools import permutations
import copy

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

"""This is the module that implements the TSP solving functions. It is the
earlier version, so its functions deal with Roads rather than with
Connectors."""

class TSP():
    """This class ties together the TSP. It loads all the precomputed data and
    contains methods to do all the preprocessing before handing it off to a
    TSPSolver. After it is solved, it then tidies the result and prints it
    out."""
    #At least, that's what it would ideally do.

class TSPSolver():
    """This is a superclass for a solver for TSPs. It lays out all the methods
    that a solver should implement. At present, it only exists to raise errors
    if those methods are not implemented."""
    def __init__(self):
        raise NotImplementedError()
    def solve(self, cities):
        """Each Solver should be able to create a solution from just a list of
        cities (with populated Reachables)."""
        raise NotImplementedError()

class BruteForce(TSPSolver):
    """Solves a TSP by checking all possible tours to find the shortest. This is
    O(n!), so it is essentially useless on tours of more than eight or nine
    cities."""
    def __init__(self):
        pass
    def solve(self, cities):
        """Returns an optimal tour that visits the given cities. As stated
        above, calling this on tours longer than nine cities takes forever."""
        if isinstance(cityList, Cities):
            cityList = cityList.cityList
        perms = permutations(cityList)
        shortest = Tour(perms.next())
        for perm in perms:
            tempTour = Tour(perm)
            if tempTour < shortest:
                shortest = tempTour
        return shortest

class TwoOpt(TSPSolver):
    """This TSP solver attempts to solve the problem with 2-opt - removing two
    roads and replacing them with two others that connect those four cities
    differently in an attempt to create a shorter tour."""
```

```
65          def __init__(self):
66              pass
67          def solve(self, cities):
68              """Returns a 2-optimal tour of the given cities."""
69              if isinstance(cities, Tour):
70                  T = copy.copy(cities)
71              else:
72                  T = Tour(cities)
73              return self.twoOptSolve(T)
74          def twoOptSolve(self, T):
75              """Calculates the two-optimal path for the given tour."""
76              initialDistance = T.updateDistance()
77              for optOne in T.insideRoads[:]:
78                  for optTwo in T.insideRoads[:]:
79                      if shares(optOne, optTwo):
80                          continue
81                      #We're being clever. If a given pair is valid, the other one isn
82                      T.remove(optOne)
83                      T.remove(optTwo)
84                      swapOne = optOne.sourceCity.getRoad(optTwo.sourceCity)
85                      swapTwo = optOne.destCity.getRoad(optTwo.destCity)
86                      if not (swapOne in T or swapTwo in T):
87                          T.add(swapOne)
88                          T.add(swapTwo)
89                          if T.updateDistance() < initialDistance:
90                              return twoOptSolve(T)
91                          T.remove(swapOne)
92                          T.remove(swapTwo)
93                      T.add(optOne)
94                      T.add(optTwo)
95              return T
96
97      class SimpleLK(TSPSolver):
98          """This TSP solver is a simple version of the Lin-Kernighan heuristic,
99          without any of the enhancements discussed by Lin and Kernighan in their
100         orignial paper. I did add one enhancement mentioned by Helsgaun - when it
101         finds a new shorter tour, it is supposed to start solving that tour
102         immediately."""
103         def __init__(self):
104             """This Java-ish model probably isn't the best."""
105             pass
106         def solve(self, cities):
107             """Returns the shortest tour that visits all the given cities."""
108             if isinstance(cities, Tour):
109                 return self.recursiveSearch(cities, cities.insideRoads[:])
110             T = Tour(cities)
111             return self.recursiveSearch(T, T.insideRoads)
112         def recursiveSearch(self, T, cutRoads, visited=[], gain=0):
113             """Performs a recursive Lin-Kernighan search to find the shortest tour.
114
115             The search is performed like this: Each road i is removed, one at a
116             time. I get a list of roads shorter than i from i's destination to any
117             other cities. I try adding each of those roads to the tour; their
118             destinations then have three roads connected to them. I check to see if
119             the two pre-existing roads can be cut and the other ends of those roads
120             can be connected to the starting city - the source of i - to produce a
121             valid tour. If it can and the valid tour is shorter, the search is
122             restarted with that tour; if the valid tour is longer, the search
123             deepens, trying to find another cut and add that will produce a shorter
124             tour."""
125             for cutRoad in cutRoads:
126                 visited.append(cutRoad.sourceCity)
127                 #The above lets the function be more purely recursive. (For all
128                 #calls after the first, the sourceCity will be the same)
129                 gain += cutRoad.distance
130                 addRoads = T.getAddRoads(cutRoad.destCity, cutRoad.distance)
```

```
131                 T.remove(cutRoad)
132                 for x in range(len(addRoads)):
133                     tempAdd = addRoads.pop()
134                     gain -= tempAdd.distance
135                     nextCuts, shortCuts = getCuts(T, visited, gain, tempAdd)
136                     #getCuts modifies T, which isn't totally clear.
137                     if shortCuts:
138                         return self.recursiveSearch(T, T.insideRoads)
139                     if nextCuts:
140                         U = copy.copy(T)
141                         recurseBest = self.recursiveSearch(U, nextCuts, visited, gair
142                         if recurseBest < T:
143                             return self.recursiveSearch(recurseBest, recurseBest.insi
144                         gain += tempAdd.distance
145                     T.remove(tempAdd)
146                 gain -= cutRoad.distance
147                 T.add(cutRoad)
148                 visited.remove(cutRoad.sourceCity)
149             if not T.isValid():
150                 #I'm under the assumption here that I only need to return a valid
151                 #tour if I was passed one. Otherwise, I need to return something
152                 #that definitely won't be passed up as it's longer than what there wa
153                 return float('Inf')
154             return T #if our starting tour is best.
155 
156     def getCuts(tour, visited, gain, addRoad):
157         """Returns the possible cuts from addRoad's destination, paired with their
158         closing roads. If a closing road is found that shortens the tour, it returns
159         an empty list of cuts and a True. The tour is modified in the testing
160         process - and we just jump out when we see the new one is shorter - so it
161         doesn't need to return the particular cut or close."""
162         #addRoad should not be added to the tour yet or this might return it as a
163         #possible cut. Note that it's added here, modifying the tour.
164         cuts = []
165         closeDistances = []
166         for road in tour.insideRoads:
167             if addRoad.destCity in road.endpoints:
168                 if (road.destCity not in visited) and (road.sourceCity not in visitec
169                     cuts.append(road)
170         #Now, cuts is a list of roads from addRoad.destCity to any city not yet
171         #modified.
172         tour.add(addRoad)
173         #For each road in cuts, I try to remove it and close the tour. If the tour
174         #produced thus is valid and shorter than the current shortest, this function
175         #jumps out; otherwise it returns a list of cuts that produce valid but not
176         #shorter tours.
177         for road in cuts:
178             tour.remove(road)
179             closeRoad = tour.getRoad(road.otherEnd(addRoad.destCity), visited[0])
180             tour.add(closeRoad)
181             cuts.remove(road)
182             if tour.isValid():
183                 if (road.distance + closeRoad.distance < gain):
184                     return [],True
185                 cuts.append(road)
186             tour.add(road)
187             tour.remove(closeRoad)
188         tour.updateDistance
189         return cuts, False
```

```
#!/usr/bin/env python
import random
from math import sqrt
import random

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

"""This module provides many utility functions that the rest of the code uses.
This is the older version of this file, so many of the functions it implements
deal with Roads rather than with Connectors."""

def measureDistance(cityA, cityB):
    "Returns the distance between city A and city B using the Pythagorean theorem."
    return sqrt((max(cityA.y, cityB.y) - min(cityA.y, cityB.y))**2 + \
        (max(cityA.x, cityB.x) - min(cityA.x, cityB.x))**2)

def shares(connectorA, connectorB):
    """Returns True if the connectors share an endpoint, False otherwise."""
    for endpoint in connectorA.endpoints:
        if endpoint in connectorB.endpoints:
            return True
    return False

def createConnections(cities, connectedness):
    """Creates connections among the cities. Connectedness should be between 0
    and 1, and controls how many other cities each city is connected to.
    Destroys the passed list or Cities object and returns a new one."""
    #Note that we want each city connected to at least one other. That won't always
    #happen with this.
    if isinstance(cities, Cities):
        cities = cities.cityList
    newCities = []
    while cities:
        city = cities.pop()
        for destination in cities:
            if (connectedness > random.random()):
                tempDistance = measureDistance(city, destination)
                city.reachables.append(Road(city, destination, [city, destination], tempDis
                destination.reachables.append(Road(destination, city, [destination, city],
        newCities.append(city)
    return Cities(newCities)

def readCities(filename):
    """Reads cities from the given filename. Returns a Cities object.
    As far as input format, I'm trying to follow the format used by the Windows
    version of Concorde, for simplicity, but I want more things than it expects,
    so I'm tacking things on in places it doesn't notice.

    The file is expected to be formatted like:
    6 0 0.75
    21.239837 89.716312 New York
    17.987805 79.787234 Atlanta
    17.987805 68.794326 Philadelphia
    25.406504 79.787234 Chicago
    25.406504 69.858156 Houston
    22.256098 59.929078 Miami

    with the first line being: cityCount segmentCount connectedness
    I'll read number_cities and use it to not read segments; I do not yet read
    or write segments.
```

```
        Connectedness is something I tacked on; Concorde ignores it; my code does
        not require it but will use it if present.

        I believe that the default Concorde save format is .qs - although it doesn't
        use that extension and I'm not sure what exactly the specifications of the
        format are.
        """
        cities = Cities([])
        f = open(filename)
        line = f.readline()
        line = line.split()
        if len(line) > 2:
            connectedness = float(line[-1])
        else:
            connectedness = 0
        cityCount = int(line[0])
        for x in range(cityCount):
            line = f.readline()
            if not line:
                break
            #The below call is being clever. If lineParse gets a name from the line,
            #the string is never used; otherwise the string is used as a city name.
            cities.append(City(*lineParse(line, str(x))))
        f.close()
        return cities, connectedness

    def lineParse(line, name):
        """Parses the given line to get values that can be used to create a City.
        Uses defaultName for the city's name if unable to find one on the line.
        >>> lineParse("17.987805 79.787234 Atlanta", "0")
        ([17.987805, 79.787234], 'Atlanta')

        >>> lineParse("17.987805 79.787234", "0")
        ([17.987805, 79.787234], '0')
        """
        words = line.split()
        x = float(words[0])
        y = float(words[1])
        if len(words) > 2:
            name = " ".join(words[2:])
        return [[x,y], name]

    def YesOrNo(question, default=-1):
        """Asks the user the passed question. Returns True if their answer is yes,
        False if it isn't. If default is passed, returns it on null input."""
        answer = raw_input(question)
        while True:
            answer = answer.strip()
            if answer == "":
                if default != -1:
                    return default
                print "Your answer was not parsed properly. Please enter 'yes' or 'no'."
                answer = raw_input(question)
                continue
            answer = answer.lower()
            if answer[0] == 'y':
                return True
            if answer[0] == 'n':
                return False
            print "Your answer was not parsed properly. Please enter 'yes' or 'no'."
            answer = raw_input(question)

    class Cities:
        """A class for a group of cities. This allows various list-like methods of
        accessing the group, but is mostly just a wrapper around a list of City
        objects."""
        def __init__(self, cityList):
            self.cities = {}
            for city in cityList:
                self.cities[city.name] = city
            self.cityList = cityList
        def __getitem__(self, key):
            if type(key) is int:
                return self.cityList[key]
```

```
147             elif type(key) is str:
148                 return self.cities[key]
149         def __len__(self):
150             """Returns the number of cities in the given Cities object."""
151             return len(self.cityList)
152         def __setitem__(self, key, value):
153             raise NotImplementedError("Use .append() to add to a CityMap.")
154         def __delitem__(self, key):
155             """Deletes the given item from the collection of cities. Does not tidy
156             reachables, however."""
157             if type(key) is str:
158                 self.cityList.remove(self.cities[key])
159                 del self.cities[key]
160             else:
161                 del self.cities[key.name]
162                 self.cityList.remove(key)
163         def __iter__(self):
164             """Returns an iterator over the cities in the given Cities object."""
165             return self.cityList.__iter__()
166         def __contains__(self, item):
167             """Returns True if the group of cities has a particular city instance,
168             or if a group of cities has a city with the given name."""
169             return (item in self.cityList) or (item in self.cities.keys())
170         def __str__(self):
171             "Returns a string suitable for exporting with city coordinates."
172             outStr = ""
173             for city in self.cityList:
174                 outStr += str(city)
175                 outStr += "\n"
176             return outStr
177         def append(self, item):
178             """Adds the given City to the Cities object."""
179             if isinstance(item, City):
180                 self.cities[item.name] = item
181                 self.cityList.append(item)
182             else:
183                 raise ValueError("Cannot add non-City to Cities.")
184         def pop(self):
185             """Returns and removes the first City in the given Cities object."""
186             city = self.cityList[0]
187             self.remove(city)
188             return city
189         def remove(self, item):
190             """Removes the given City from the Cities object either by name or by
191             instance."""
192             self.__delitem__(item)
193         def index(self, item):
194             """Returns the index of the given City in the underlying list of the
195             Cities object."""
196             if type(item) is str:
197                 return self.cityList.index(self.cities[item])
198             return self.cities.index(item)
199         def hasOrphans(self):
200             """Returns True if there are any cities that are not connected to any
201             others. Does not test for disconnected clusters."""
202             for city in self.cityList:
203                 if len(city.reachables) == 1:
204                     return True
205             return False
206 
207     class Tour(Cities):
208         """A Tour is a structure that stores cities and provides methods to make the
209         implementation of the traveling salesman solver easier.
210 
211         A Tour is an itinerary that visits each city once and returns to its source.
212         This class stores a list of roads that are used in that itinerary and the
213         roads that connect the cities in the tour but are not used in the itinerary.
214         It provides methods to add and remove roads, and check to see if a given
215         tour is valid."""
216         def __init__(self, cityList, randomize=False):
217             """Creates a Tour from the passed list of cities. At this point, it's
218             assumed that each city has a fully populated list of reachables."""
219             #The actual 'order' of the tour is clumsy to access at present.
220             self.cities = {}
```

```
            if isinstance(cityList, Cities):
                cityList = cityList.cityList #maybe that could be named better...
            for city in cityList:
                self.cities[city.name] = city
            self.cityList = cityList
            self.cityMatrix = DistanceMatrix(cityList)
            if randomize:
                self.shuffle()
            self.insideRoads, self.outsideRoads = getRoads(self.cityList)
            self.updateDistance()
        def __cmp__(self, other):
            """Compares two Tours by their distance. Assumes both are valid."""
            return cmp(self.distance, other)
        def __str__(self):
            """Returns a string that describes the tour. Expects the tour to be valid.
            >>> T
            <utils.Tour instance at 0x0279FF80>
            >>> print T
            A tour of 3 cities, with total distance 57.
            The cities, in the order they are visited:
            New York, Boston, Washington, New York.

            (note that the first city is at both ends; this was not by design but
            I thought it looked better and decided to keep it.)
            """
            outStr = "A tour of %d cities, with total distance %d.\n" % (len(self.cityList), se
            outStr += "The cities, in the order they are visited:\n"
            testCities = []
            for road in self.insideRoads:
                testCities.append(road.sourceCity)
            currentRoad = self.insideRoads[0]
            outStr += (currentRoad.sourceName + ", ")
            for x in range(len(self.insideRoads)):
                currentRoad = self.insideRoads[testCities.index(currentRoad.destCity)]
                outStr += (currentRoad.sourceName + ", ")
            outStr = outStr[:-2] + "."
            return outStr
        def add(self, road):
            """Adds the given road to the tour, removing it from tour.outsideRoads
            and adding it to tour.insideRoads."""
            self.outsideRoads[road.sourceName].remove(road)
            self.insideRoads.append(road)
        def getRoad(self, cityA, cityB):
            """Returns the road from cityA to cityB."""
            return self.cityMatrix.getRoad(cityA, cityB)
        def isValid(self):
            """Returns True if the tour is valid - that is, if all the cities in it
            are visited exactly once and if it's closed."""
            '''How it works - as I don't think it's quite transparent: We start with
            insideRoads, which is a list of roads in any order. As walking that
            would be difficult, I first create a list of the sourceCities in the
            same order as they appear in insideRoads. I can then search that list to
            get the place of the next road in insideRoads, and can walk it that
            way.'''
            visited = self.cityList[:]
            testCities = []
            for road in self.insideRoads:
                testCities.append(road.sourceCity)
            currentRoad = self.insideRoads[0]
            for x in range(len(self.insideRoads)):
                try:
                    visited.remove(currentRoad.sourceCity)
                    currentRoad = self.insideRoads[testCities.index(currentRoad.destCity)]
                except ValueError:
                    return False
            return currentRoad is self.insideRoads[0]
        def getAddRoads(self, city, distance=float('Inf')):
            """Returns the roads from the given city with lengths less than
            distance."""
            addRoads = []
            for road in self.outsideRoads[city.name]:
                if road.distance < distance:
                    addRoads.append(road)
            return addRoads
```

```
295         def printRoads(self):
296             """Prints the roads in the tour. Meant to be used on invalid tours,
297             where str() will fail."""
298             print "Roads in tour:"
299             for road in self.insideRoads:
300                 print road
301             print
302         def shuffle(self):
303             """Shuffles the cityList. Only used at init as it does not affect
304             roads."""
305             random.shuffle(self.cityList)
306         def remove(self, road):
307             """Removes the given road from the tour, assuming it is present. Does
308             not check to see if a road is in the tour first."""
309             self.insideRoads.remove(road)
310             self.outsideRoads[road.sourceName].append(road)
311         def updateDistance(self):
312             """Updates the tour's distance. Returns that distance."""
313             distance = 0
314             for road in self.insideRoads:
315                 distance += road.distance
316             self.distance = distance
317             return distance
318 
319     def getRoads(cityList):
320         """When passed the given cityList, returns two lists: the list of Roads that
321         are needed to connect the given cities in a closed circuit in the given
322         order, and the list of roads that are not."""
323         outsideRoads = []
324         for city in cityList:
325             outsideRoads.extend(city.reachables)
326         insideRoads = []
327         for x in range(len(cityList) - 1):
328             tempRoad = cityList[x].getRoad(cityList[x+1])
329             insideRoads.append(tempRoad)
330             outsideRoads.remove(tempRoad)
331         tempRoad = cityList[-1].getRoad(cityList[0])
332         insideRoads.append(tempRoad)
333         outsideRoads.remove(tempRoad)
334         outsideDict = {}
335         for city in cityList:
336             outsideDict[city.name] = []
337         for road in outsideRoads:
338             outsideDict[road.sourceName].append(road)
339         return insideRoads, outsideDict
340 
341     def removeDuplicates(duplicatedList):
342         """Removes duplicates from the given list, in place. Assumes the items in
343         the list are comparable, but not hashable."""
344         #from the Python FAQ.
345         duplicatedList.sort()
346         last = duplicatedList[-1]
347         for i in range(len(duplicatedList) -2, -1, -1):
348             if last == duplicatedList[i]:
349                 del duplicatedList[i]
350             else:
351                 last = duplicatedList[i]
352 
353     class DistanceMatrix():
354         """A DistanceMatrix is a matrix that stores the shortest distance from each
355         city to each other city. It is used when running Djikstra's algorithm.
356 
357         The data is not actually stored as a matrix, but as a multidimensional
358         Python dictionary. Each source city maps to a dictionary of destination
359         cities; each destination city maps to a Road that goes from the source to
360         the destination."""
361         def __init__(self, cities):
362             """Creates a distance matrix with data from the given cities. Sets all
363             unknown distances to infinity."""
364             self.matrix = {}
365             for source in cities:
366                 self.matrix[source.name] = {}
367                 for dest in cities:
368                     self.matrix[source.name]
```

```
[dest.name] = Road(source, dest, [], float('Inf'))
            for source in cities:
                for road in source.reachables:
                    self.setRoad(source, road.destCity, road)
        def __str__(self):
            """Returns a string representation of the given distance matrix."""
            cityNames = self.matrix.keys()
            cityNames.sort()
            outStr = "\t"
            #first line#
            for name in cityNames:
                outStr += name
                outStr += "\t"
            outStr += "\n"
            for source in cityNames:
                outStr += source
                outStr += "\t"
                for dest in cityNames:
                    outStr += "%.2f" % float(self.getDistance(source, dest))
                    outStr += "\t"
                outStr += "\n"
            return outStr
        def getDistance(self, source, dest):
            """Returns the distance from source to dest; source and dest can be
            cities or names of cities."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            return self.matrix[source][dest].distance
        def getRoad(self, source, dest):
            """Returns the road from source to dest. If the matrix is newly created,
            it may return infinity instead of a Road object."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            return self.matrix[source][dest]
        def setRoad(self, source, dest, road):
            """Sets the road from source to destination in the matrix to the given
            road."""
            if isinstance(source, City):
                source = source.name
            if isinstance(dest, City):
                dest = dest.name
            self.matrix[source][dest] = road
        def roadsFrom(self, source):
            """Returns a list of roads leading from source to any other cities."""
            if isinstance(source, City):
                source = source.name
            roadList = self.matrix[source].values()
            return roadList
        def values(self):
            """Returns all the destination dictionaries in the given matrix."""
            return self.matrix.values()

    class CityMap(Cities):
        """A CityMap is a class that provides some structure for storing a
        two-dimensional "map" of cities. It stores the size of the map and can print
        a representation of it."""
        def __init__(self, cityList, mapX, mapY, connectedness=None):
            self.connectedness = connectedness
            self.cities = {}
            for city in cityList:
                self.cities[city.name] = city
            if isinstance(cityList, Cities):
                self.cityList = cityList.cityList #Sorry.
            else:
                self.cityList = cityList #Again, sorry.
            self.x = mapX
            self.y = mapY
        def printMap(self):
            """Constructs and prints a graphical representation of the city
            locations. Does not feature connections on the map yet."""
```

```
442             printList = []
443             for x in range(self.y):
444                 printList.append("." * self.x)
445             for city in self.cityList:
446                 printList[city.y] = printList[city.y][:city.x] + "X" + printList[city.y]
[city.x+1:]
447             for line in printList:
448                 print line
449         def __str__(self):
450             "Returns a string suitable for exporting with city coordinates."
451             outStr = ""
452             outStr += "Map Width:\t%d\nMap Height:\t%d\n" % (self.x, self.y)
453             outStr += "Cities:\n"
454             for city in self.cityList:
455                 outStr += str(city)
456                 outStr += "\n"
457             return outStr
458
459     def getDistance(path):
460         """Returns the length of the given Path. Expects each city in the path to
461         have a distance to at least the next one."""
462         distance = 0
463         for x in range(len(path) - 1):
464             distance += path[x].getRoad(path[x+1]).distance
465         return distance
466
467     class Connector:
468         """A class for Connectors. A Connector is an undirected segment from one
469         city to another."""
470         def __init__(self, cityA, cityB, path, distance=None):
471             """Inititalizes a connector. Sorts the passed cities and reverses the
472             path as necessary to ensure a connector between two cities will be the
473             same regardless of order."""
474             self.endpoints = [cityA, cityB]
475             self.endpoints.sort()
476             if self.endpoints[0] != path[0]:
477                 path.reverse()
478             self.path = path
479             if not distance:
480                 distance = getDistance(path)
481         def __cmp__(self, other):
482             """Compares two Connectors by distance."""
483             return cmp(self.distance, other)
484         def __str__(self):
485             """Prints a string representation of the connector."""
486             #From Jim
487             return " %s - %s (%.2f) " % \
488                 (self.endpoints[0], self.endpoints[1], self.distance)
489         def __eq__(self, other):
490             """Returns True if the other object is a connector that connects the
491             same two cities.
492
493             Note that if there is a Connector A connecting New York and Chicago,
494             and a Road B connecting New York and Chicago, A == B will evaluate to
495             True, but B == A will not."""
496             if isinstance(other, Connector):
497                 return self.endpoints == other.endpoints
498             return False
499         def otherEnd(self, city):
500             """When given one end of a connector, returns the other. Doesn't check
501             to see if the passed city _is_ one end of the connector."""
502             if city is self.endpoints[0]:
503                 return self.endpoints[1]
504             return self.endpoints[0]
505
506     class Road(Connector):
507         """A class for roads; a road is a directed path from one city to another."""
508         def __init__(self, cityA, cityB, path, distance=0):
509             """Creates a Road. The two cities are the endpoints, path is a list of
510             cities that are traveled through. It cannot be null; should always
511             contain at least two cities - one iff cityA is cityB."""
512             #all the different accessors for endpoints were added at different times
513             #as I thought I needed them. Most are no longer used.
514             self.endpoints = [cityA, cityB]
```

```
            self.endpoints.sort()
            self.source = [cityA, cityA.name]
            self.destination = [cityB, cityB.name]
            self.sourceName = cityA.name
            self.destName = cityB.name
            self.sourceCity = cityA
            self.destCity = cityB
            self.path=path
            if not distance:
                distance = getDistance(path)
            self.distance = distance
        def __eq__(self, other):
            """Returns True if the two roads are the same. See the note at
            Connector.__eq__."""
            if isinstance(other, Road):
                return self.path == other.path
            return False
        def __str__(self):
            """Prints a string representation of the given road."""
            #From Jim
            return " %s -> %s (%.2f) " % \
                (self.sourceName, self.destName, self.distance)
        def toConnector(self):
            """Returns a Connector with the same endpoints as the given road."""
            return Connector(self.sourceCity, self.destCity, self.path, self.distance)

    class City:
        """A class for cities. A city has a name and x-y coordinates as a location,
        although the coordinates are not used for anything other than to create
        roads as yet. Cities also store a list of the other cities that are
        reachable from them and which roads lead to those other cities."""
        def __init__(self, location, name=None, continent=None):
            """
            >>> testCity = City((5,10), "New York")
            >>> print testCity
            City New York at (5, 10), connected to 1 cities: New York.
            """
            #Note that this init routine adds a road from the new city to itself.
            self.x = location[0]
            self.y = location[1]
            self.continent = continent
            self.name = name
            self.reachables = [Road(self, self, [self], 0)]
        def __cmp__(self, other):
            """Compares two City objects by name."""
            return cmp(self.name, other)
        def __str__(self):
            """Returns a string representation of the given city and which other
            cities are reachable from it."""
            reachableString = ""
            for road in self.reachables:
                reachableString += road.destName
                reachableString += ", "
            reachableString = reachableString[:-2]
            outStr = "City %s at (%d, %d), connected to %d cities: %s." % \
            (self.name, self.x, self.y, len(self.reachables), reachableString)
            return outStr
        def getRoad(self, destination):
            """Returns the Road from this city to the given destination."""
            for road in self.reachables:
                if destination in road.destination:
                    return road
            raise Exception("There is no road from %s to %s." % (str(self), str(destination)))
        def reachableCities(self):
            """Returns a list of Roads that are reachable from the given city."""
            cityList = []
            for road in self.reachables:
                cityList.append(road.destCity)
            return cityList
```

```
#!/usr/bin/env python

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010

"""

"""This module implements Djikstra's algorithm and several related functions."""

from utils import *
import heapq

def testDjikstra(auto=False, verbose=True):
    """A quick and simple test routine. Generates some cities, runs Djikstra's
    on them. Not a substitute for real testing; just here so that this does
    something from the command line."""
    if auto:
        cities = genDefault()
    else:
        cities = genCities()
    start = time.time()
    matrix = extendedDjikstra(cities)
    populateReachables(cities, matrix)
    finish = time.time()
    if verbose:
        print cities
        print matrix
    print "%d cities processed. %f seconds taken to run Djikstra's." % (len(cities), finish - start)

def populateReachables(cities, distances):
    """Replaces all of the reachables in cities with the roads in distances."""
    distances.toConnectors()
    for city in cities:
        temp = distances.roadsFrom(city.name)
        temp.sort()
        temp = temp[1:]
        city.reachables = temp

def extendedDjikstra(cities):
    """Runs an extended version of Djikstra's algorithm to produce, in essence,
    a complete graph over the given set of cities. Djikstra's algorithm normally
    produces a minimal spanning tree from a given starting node; this version
    does the same thing, but uses each city in turn as a starting node. Note
    that this function does not replace the reachables in each city. Call
    populateReachables to do that."""
    distances = DistanceMatrix(cities)
    for startCity in cities:
        unvisitedCities = cities.cityList[:]
        currentCity = startCity
        currentDistance = 0 #distance from startCity to currentCity
        getDistance = distances.getDistance
        getRoad = distances.getRoad
        setRoad = distances.setRoad
        shortestDistances = startCity.reachables[:]
        heapq.heapify(shortestDistances)
        while True:
            for road in currentCity.reachables:
                if (currentDistance + road.distance) < (getDistance(startCity, road.destCity)):
                    newRoad = joinRoads(getRoad(startCity, currentCity), road)
                    setRoad(startCity, road.destCity, newRoad)
                    heapq.heappush(shortestDistances, newRoad)
            unvisitedCities.remove(currentCity) #we've checked all the roads, so we're done with this city
            currentCity = shortestHeap(shortestDistances, unvisitedCities)
            if currentCity == None: #the check to see if we've visited all cities.
                break
            currentDistance = getDistance(startCity, currentCity)
    return distances

def shortestHeap(shortestDistances, unvisitedCities):
    """Returns the nearest city from shortestDistances that is in unvisitedCities."""
    if not unvisitedCities:
        return None
    while shortestDistances:
        road = heapq.heappop(shortestDistances)
```

```
85              if road.destCity in unvisitedCities:
86                  return road.destCity
87          #Execution should not get here.
88          return None
89
90      def joinRoads(roadA, roadB):
91          """Returns a new Road from the start of roadA to the end of roadB."""
92          newPath = roadA.path[:]
93          newPath.extend(roadB.path)
94          newPath.remove(roadA.destCity)
95          newDistance = roadA.distance + roadB.distance
96          return Road(roadA.sourceCity, roadB.destCity, newPath, newDistance)
97
98      if __name__ == '__main__':
99          "If the user invokes it on the command line, it does a quick test."
100         from randomdatagen import *
101         import time
102         import cProfile
103         cProfile.run("testDjikstra(auto=True, verbose=False)")
```

```
#!/usr/bin/env python

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

"""
This module generates a random set of cities which are suitable for TSP tests.
"""
import random
from utils import *

def genDefault():
    """Returns a set of cities created with some default values."""
    return genCities(10,10,10,0.5)

def genCities(mapX=None, mapY=None, cityCount=None, connectedness=None, cityMap=False):
    """Returns a cityList constructed to fit the supplied parameters.

    MapX and MapY are the size of the map in those dimensions, cityCount is the
    number of cities that should be generated. Connectedness is a value between
    0 and 1 that represents how connected the cities are - if it is 1, the
    cities are 100% connected and there is a road from each city to every
    other; if it is 0.5, there is a 50% chance that there is a road from one
    city to another, so on.
    If cityMap is True, a CityMap object is returned instead of a Cities
    object."""
    if (not ((mapX and mapY) and (cityCount and connectedness))):
        mapX, mapY, cityCount, connectedness = getParams()
    cityLocations = []
    names = (map(str, range(cityCount)))
    for x in range(cityCount):
        while True:
            newLocation = (random.randrange(mapX), random.randrange(mapY))
            if newLocation not in cityLocations:
                cityLocations.append(newLocation)
                break
    cityList = map(City, cityLocations, names)
    cities = createConnections(cityList, connectedness)
    if cityMap:
        cities = CityMap(cities, mapX, mapY, connectedness)
    return cities

def getParams():
    """Asks the user for some map parameters; returns them. Returns defaults if
    user hits enter. Said defaults may cause problems if the user takes some but
    not others.."""
    try:
        mapX = int(raw_input("Enter the (integer) width of the map: "))
    except ValueError:
        print "Error - Using 10."
        mapX = 10
    try:
        mapY = int(raw_input("Enter the (integer) height of the map: "))
    except ValueError:
        print "Error - Using 10."
        mapY = 10
    try:
        cityCount = int(raw_input("Enter the (integer) number of cities the map should have: "))
    except ValueError:
        print "Error - Using 10."
        cityCount = 10
    try:
        connectedness = float(raw_input("How connected should the cities initially be? (1 = all connected, 0.5 = 50% co
nnected, etc.): "))
    except ValueError:
        print "Error - Using 0.5."
        connectedness = 0.5
    if (cityCount > mapX * mapY):
        raise Exception("Too many cities for map.")
    return mapX, mapY, cityCount, connectedness

if __name__ == '__main__':
    """If the user invokes it on the command line, generates some cities, puts
    them on a map, and prints the map."""
    cities = genCities(cityMap=True)
    print cities
    cities.printMap()
```

```
#!/usr/bin/env python

from utils import *
from djikstras import *
from tsp import *
from randomdatagen import *
from itertools import permutations

"""
This is a part of a Python implementation of the Lin-Kernighan heuristic for
solving the Euclidean traveling salesman problem. Various parts of this code do
not work as intended; other parts of the code may not work at all.

That said, all of this is licensed under the Creative Commons Attribution 3.0
Unported License (see http://creativecommons.org/licenses/by/3.0/ for license
text), so if you think you can take it and make it into something useful, go for
it.

Richard Scruggs
December, 2010
"""

def main():
    """This is a terribly disorganized test routine that is meant to either
    generate or read some cities, run Djikstra's algorithm on them, create a
    tour that visits them all, and then run a traveling salesman solver to find
    the shortest tour."""
    print("This is a TSP test routine.")
    print("If you have a data file, please enter the filename, or enter 'r' to generate random data."
    filename = raw_input("Filename: ")
    if filename == "r":
        cities = genCities()
    else:
        cities, connectedness = readCities(filename)
        if not connectedness:
            print("There was no value for connectedness in the given file.")
            try:
                connectedness = float(raw_input("How connected should the cities be? (1 = all connect
 etc.): "))
            except ValueError:
                print "Error - Using 1.0."
                connectedness = 1.0
        cities = createConnections(cities, connectedness)
    print("Running the extended Djikstra's algorithm on the cities.")
    matrix = extendedDjikstra(cities)
    populateReachables(cities, matrix)
    if YesOrNo("Would you like to run a complete test? ", False):
        if len(cities) > 10:
            print("Too many cities to run a complete test on.")
            return
        solve = completeSolve(cities)
        if solve[0]:
            print("Complete solve successfully completed.")
        else:
            print("Solve failed.")
            print("These two tours did not match (Other tours may also not be solving correctly.):")
            print solve[1][0]
            print solve[1][1]
    else:
        print("Creating a tour of the cities.")
        T = Tour(cities)
        print "The created tour:"
        print str(T)
        print("Initializing the solver.")
        solver = BruteForce()
        print("Solving the TSP.")
        shortest = solver.solve(T)
        print shortest
        return shortest

def completeSolve(cityList):
    """Takes a list of cities and tries to solve them in every possible
    permutation. Returns True if all permutations solve to the same distance,
    False and which two tours failed otherwise. Note, obviously, that this is n!
    time, so don't do this on lists of more than a few cities. Also, since this
    is testing all permutations, a few of them will be the optimum tour without
    solving, just to make sure we get it in there."""
    if isinstance(cityList, Cities):
        cityList = cityList.cityList
```

```
        T = Tour(cityList)
        solver = DumbSolver()
        T = solver.solve(T)
        distance = T.distance
        perms = permutations(cityList)
        for perm in perms:
            U = solver.solve(Tour(list(perm)))
            if U.distance != distance:
                return False, [T, U]
        return [True]

    def testTourValidity(cityList):
        """Checks to see if all the valid tours made from a given cityList return
        valid. Exists to test Tour.isValid."""
        perms = permutations(cityList)
        for perm in perms:
            T = Tour(list(perm))
            if not T.isValid():
                return False
        return True

    if __name__ == '__main__':
        solution = main()
    else:
        T = main()
```

First computer science plan exam :
 programming language skills and concepts
 for Richard Scruggs
 from Jim Mahoney
 on Nov 2 2010

This is an open, take home exam: books or web sources are OK as long as you cite them explicitly and as long as they aren't a drop-in solution to the problem. Don't ask other people for help. Your job is to convince us you understand this stuff.

This is to be done during the week of Nov 2 - Nov 9;
the due date is 10am on Nov 9, before our tutorial.

As always with my exams, if you think there's a mistake in one of the questions or it doesn't make sense, you can (a) ask for clarification, and/or (b) make and state an explicit interpretation and do the problem that way. (Again: the point is to demonstrate your understanding, not to get the "right answer" per se.)

For any code submitted, try to follow typical 'best practices' including language typical docs, tests, and coding style.
The harder I have to work to figure out what's what, the less I will be impressed, eh?

I) Write six short programs, one in each of C, lisp, and python
    (using any version or variant you wish; just be clear about
    your choices) for each of the following two problems.

    At least one of the six should be done with an object oriented
    approach, and at least one with a functional approach.

    In each case, include appropriate tests and documentation
    for the language and problem; it should be clear what it does
    and how to run it.

    * problem A (from projecteuler.net) :

      A palindromic integer reads the same both ways in base ten.
      The largest palindrome made from the product of two 2-digit
      numbers is 9009 = 91 * 99. Find the largest palindrome made
      from the product of two 3-digit numbers.

    * problem B :

      Read and interpret a geneological data file of this form :

      --- data.txt ---
      name: Tom Jones (12338)
      born: 12/10/2004
      died: -
      dad: Fred Smith (78223)

```
      mom:  Irene Smith (23456)

      name: Irene Thompson (23456)
      born: 1/2/1952
      died: 3/18/2006
      dad:  -
      mom:  -

      -----------------

      The number in parens is a unique id for that person;
      the name may vary across records.  (In the example above,
      Irene was born Irene Thompson but in Tom's record is listed
      with her married name.)

      The program should be able to store a representation of the data
      and output some sort of report; the details I leave up to you.
      For example, it could print the age and siblings of each person,
      or print a representation (ascii or otherwise) of the family tree.

      Create some (fake) sample data to feed it, with at least
      ten people over three generations.  You may use graphic
      or other libraries if you wish; if so, include them.


II)  Explain the concepts behind following programming buzz words,
     across all three programming languages.
       (a) First organizing them into groups showing which are variations
           of the same concept, or variations across languages,
         or opposites.
       (b) Then for each group, give example code snips from various
           languages to illustrate them.
       (c) Discuss the concepts behind each group, and how the words
           and meaning are similar or different across the languages.
      Be clear that I *don't* want you to just define each of these
words;
      instead, I want you to use them as the starting point for a
conversation
      with examples about the ideas the words are trying to convey in C,
      Python, and Lisp.

     * API
     * argument
     * class
     * closure
     * collection
     * comment
     * compiled
     * data structure
     * dynamic
     * exception
     * function
     * global
     * hash
```

* inheritance
* interface
* interpreted
* iterate
* lexical
* link
* list
* macro
* method
* namespace
* object
* overload
* scope
* package
* pattern
* pass by reference
* pass by value
* recursion
* side effect
* stack overflow
* static
* symbol
* syntactic sugar
* test
* throw
* variable
* vector

III) Discuss the strengths and weaknesses of python, C, and lisp as you see them. What sorts of problems are well suited for each language, and why? Be specific in your comparisons, giving examples that justify your comparisons and conclusions.

General readme and attribution for any sources that were referenced:

1:
All of these can be run as you would expect - for Python, "python filename.py",for Lisp, "clisp filename.lisp", for C, compile with GCC and run. 1-2.c expects datafile.txt as the only argument, while 1-2.lisp and 1-2.py ask for the name to be entered once the program starts.

1-1: Python: I just used the online Python docs, http://docs.python.org/index.html The code should run on recent Python 2.x; 3.x replaced range() with xrange() and removed xrange(). I prefer to use the iterator in all cases rather than using the list - and hardly anyone uses Python 3.

For Lisp, I used the HyperSpec, http://www.lispworks.com/documentation/common-lisp.html and Seibel's Practical Common Lisp, http://gigamonkeys.com/book/ I did not excessively test the code on different Lisp compilers; it runs on clisp 2.49 and 2.37; I assume it would run on most versions of Lisp. I'm not convinced that it's "functional" enough--more of a hybrid, really—but it is recursive.

For C, I used the C library reference from http://www.cplusplus.com/reference/clibrary/
The code compiles cleanly with -Wall under GCC 4.5.1; it should compile with any ANSI C compiler.

1-2:
Assumption for all of these: people's ID numbers have no (significant) leading 0s.

Python: Again, just online Python docs.

For Lisp, the same references as for 1-1 above. I also used split-sequence, from http://www.cliki.net/SPLIT-SEQUENCE - it seems rather odd to me that Lisp, with its comparatively large standard library, has no string split or tokenizing function. I put the code right in the file as I was having issues importing the package properly across platforms.

For C, I used the same references as above along with http://c-faq.com/. There is one more assumption made to ease the string processing: People are expected to have a first name and a last name.
For whatever reason, the C program does not work as designed or expected. It'll read in the data and print it out properly, but running qsort does unexpected things. The order of the data is changed; the resulting order is repeatable and does vary when the comparison function is modified, but the resulting order is _not_ sorted with the passed comparison function.

2: I did not use 'stack overflow', 'macro', and 'pattern'; I added 'return value' to the list.

Code examples taken from Practical Common Lisp, http://www.scriptol.com/programming/fibonacci.php, and the Python docs.

3:
The Lisp benchmark in the paper is for the initial, iterative version of the program. The later, recursive version is slower.

```python
#!/usr/bin/env python

#Programming Language Skills and Concepts exam
#Problem 1
#Part 1
#Richard Scruggs
#11/4/2010
#Licensed under the Creative Commons Attribution 3.0 Unported License.
#see http://creativecommons.org/licenses/by/3.0/ for license text.

def isPalindromic(string):
    """Returns True if the given string is a palindrome, False otherwise.

    >>> isPalindromic("racecar")
    True

    >>> isPalindromic("test")
    False
    """
    for a,b in zip(iter(string), reversed(string)):
        if a != b:
            return False
    return True

def largestProductPalindrome(maxFactor=100, minFactor=0):
    """Returns a list with the largest palindromic product that can be produced
    with factors greater than minFactor and less than maxFactor and the factors
    that are multiplied to produce that product. Returns [0,0,0] if no
    palindromes are found.
    Note that maxFactor and minFactor behave as in range(); maxFactor is
    excluded but minFactor is included.

    >>> largestProductPalindrome()
    [9009, 91, 99]

    >>> largestProductPalindrome(1000, 100)
    [906609, 913, 993]
    """
    #I spent way too long profiling this - which probably ended up making the
    #code harder to follow, but did make it faster.
    #To explain a bit of it: I reverse the factor lists so I can avoid
    #unnecessary multiplications and calls to isPalindromic. When checking each
    #pair of factors, I only multiply them out as long as their product is
    #bigger than the current maxPalindrome; once they get smaller than it, I
    #break from that cycle of the inner loop and start again with a new x and y.
    #
    #Also, note that I only test y-factors that are larger than the current
    #x-factor. This tends to find larger palindromes faster and reduces
    #duplicate multiplications.

    maxPalindrome = [0,0,0]
    for x in reversed(xrange(minFactor, maxFactor)):
        for y in reversed(xrange(x, maxFactor)):
            if x*y > maxPalindrome[0]:
                if isPalindromic(str(x*y)):
                    maxPalindrome = [x*y, x, y]
            else:
                break
    return maxPalindrome

def main():
    """This main function just takes the general functions written above and
    applies the required constrains."""
    print("This program uses Python to calculate the largest palindromic number produced from two three-digit factor
s.")
    print("Calculating...")
    maxPalindrome = largestProductPalindrome(1000, 100)
    print("The largest palindrome is %d, formed by multiplying %d and %d." % (maxPalindrome[0], maxPalindrome[1], ma
xPalindrome[2]))

main()
```

```
1   ;Programming Language Skills and Concepts exam
2   ;Problem 1
3   ;Part 1
4   ;Richard Scruggs
5   ;11/5/2010
6   ;Licensed under the Creative Commons Attribution 3.0 Unported License.
7   ;see http://creativecommons.org/licenses/by/3.0/ for license text.
8
9   (defun is-palindrome (string)
10    "Returns T if the given string is a palindrome, NIL otherwise."
11    (string-equal string (reverse string)))
12
13  (defun largest-palindrome-product (max-factor &optional (min-factor 100) (largest-palindrome 0))
14    "Calculates the largest palindromic number that can be produced with two factors less than max-factor and great
    er than min-factor."
15    (do ((x max-factor (1- x)))
16        ((< x min-factor) largest-palindrome)
17      (do ((y max-factor (1- y)))
18      ((< y x))
19        (if (> (* x y) largest-palindrome)
20        (if (is-palindrome (write-to-string (* x y)))
21            (return-from largest-palindrome-product (largest-palindrome-product max-factor min-factor (* x y))))
22        (return)))))
23
24  (defun main ()
25    "Applies the specific constraints required by the problem to the more general functions. Prints the result."
26    (format t "This program uses Lisp to calculate the largest palindromic number produced from two three-digit fac
    tors.~%")
27    (format t "Calculating...~%")
28    (format t "The largest palindrome is ~a.~%" (largest-palindrome-product 1000)))
29
30  (main)
```

```c
/*
Programming Language Skills and Concepts exam
Problem 1
Part 1
Richard Scruggs
11/4/2010
Licensed under the Creative Commons Attribution 3.0 Unported License.
see http://creativecommons.org/licenses/by/3.0/ for license text.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int is_palindrome (int test_number){
    /*Returns 1 if the given number is palindromic, 0 otherwise.*/
    char test_string[11]; //2^32 is 10 digits long.
    sprintf(test_string, "%d", test_number);
    int length = strlen(test_string);
    int half = length/2;
    int end_pointer = length - 1;
    int i;
    for (i=0; i<half; i++){
        if (test_string[i] != test_string[end_pointer-i])
            return 0;
    }
    return 1;
}

int largest_product_palindrome (int max_factor, int min_factor){
    /*Calculates the largest palindromic number that can be produced with two
    factors less than max_factor and greater than min_factor."*/
    int max_palindrome = 0;
    int x,y;
    for (x=max_factor; x>min_factor; x--){
        for (y=max_factor; y>=x; y--){
            if ((x * y) > max_palindrome){
                if (is_palindrome(x*y)){
                    max_palindrome = x * y;
                }
            }
            else{
                break;
            }
        }
    }
    return max_palindrome;
}

int main(){
    /*Applies the specific constraints required by the problem to the more general functions. Prints the result.*/
    printf("This program uses C to calculate the largest palindromic number produced from two three-digit factors.\n
");
    printf("Calculating...\n");
    printf("The largest palindrome is %d.\n", largest_product_palindrome(1000, 100));
    return 0;
}
```

```
#!/usr/bin/env python

#Programming Language Skills and Concepts exam
#Problem 1
#Part 2
#Richard Scruggs
#11/8/2010
#Licensed under the Creative Commons Attribution 3.0 Unported License.
#see http://creativecommons.org/licenses/by/3.0/ for license text.

from datetime import date

class Person():
    """A class for people. Stores name, ID number, birth and death dates, and
    parents."""
    def __init__(self, nameID, born=None, died=None, dad=None, mom=None):
        """Initializes a Person object. Expects [name, ID], date, date, [name,
        ID], [name, ID]."""
        self.id = nameID[1]
        self.name = nameID[0]
        self.born = born
        self.died = died
        if mom:
            self.mom = mom[1]
        else:
            self.mom = mom
        if dad:
            self.dad = dad[1]
        else:
            self.dad = dad
    def age(self):
        """Returns the age of the Person. If they are still alive, it is
        calculated as of today; else it is their age at death."""
        if not self.born:
            return None
        if self.died:
            return ((self.died - self.born).days / 365)
        return ((date.today() - self.born).days / 365)

def readPeople(filename):
    """Reads People from the given filename. Returns a list of Person
    objects, or False if there is an error opening the file."""
    try:
        infile = open(filename)
    except IOError:
        return False
    people = []
    tempQualities = []
    while True:
        for x in range(5):
            tempQualities.append(infile.readline().split(':')[1].strip())
        people.append(makePerson(tempQualities))
        tempQualities = []
        if not infile.readline():
            break
    return people

def cmpAge(personA, personB):
    """Compares the given two people by age. Just a wrapper for cmp."""
    return cmp(personA.age(), personB.age())

def makePerson(tempQualities):
    """Makes a Person object from the simply separated list tempQualities."""
    longerQualities = []
    for quality in tempQualities:
        if '(' in quality:
```

```
67                splits = quality.split('(')
68                longerQualities.append([splits[0].strip(), int(splits[1].strip(')'))])
69            elif '/' in quality:
70                longerQualities.append(makeDate(quality))
71            else:
72                longerQualities.append(None)
73        return Person(*longerQualities)
74
75    def makeDate(dateString):
76        """Takes the given dateString and returns a proper date.date object for it.
77
78        >>> makeDate("12/10/2004")
79        datetime.date(2004, 12, 10)"""
80        tempParts = dateString.split("/")
81        for x in range(len(tempParts)):
82            tempParts[x] = int(tempParts[x])
83        dateParts = [tempParts[2]]
84        dateParts.extend(tempParts[:2])
85        return date(*dateParts)
86
87    def printAges(people):
88        """Takes a list of Persons. Prints them, sorted by age."""
89        people.sort(cmpAge)
90        print "Here are the people in the data file, sorted by age."
91        for person in people:
92            print "%s: %d" % (person.name, person.age())
93
94    def main():
95        """Asks the user for a filename and wraps the rest of the functions."""
96        filename = raw_input("Enter data filename: ")
97        people = readPeople(filename)
98        if not people:
99            print("Error reading from file %s. Check filename." % filename)
100           return
101       else:
102           printAges(people)
103
104   main()
```

```
1    ;Programming Language Skills and Concepts exam
2    ;Problem 1
3    ;Part 2
4    ;Richard Scruggs
5    ;11/9/2010
6    ;Licensed under the Creative Commons Attribution 3.0 Unported License.
7    ;see http://creativecommons.org/licenses/by/3.0/ for license text.
8
9    (defvar *people* nil)
10
11   (defun add-person (person)
12     "Pushes the given person into *people*."
13     (push person *people*))
14
15   (defun person-form (person-qualities)
16     "Creates a plist with the passed person-qualities."
17     (list :name (caar person-qualities) :id (cadar person-qualities)
18       :born (second person-qualities) :died (third person-qualities)
19       :mom (if (fourth person-qualities)
20            (second (fourth person-qualities))
21            nil)
22       :dad (if (fifth person-qualities)
23            (second (fifth person-qualities))
24            nil)))
25
26   (defun read-people (filename)
27     "Reads people from the passed data file. Puts them into *people*. Returns the nu
     at there's at least one person in the file."
28     (with-open-file (stream filename :if-does-not-exist nil)
29       (loop
30        (add-person (make-person
31             (loop for i from 1 upto 5 collecting
32               (string-trim " " (second (split-
sequence #\: (read-line stream nil)))))))
33        (if (not (read-line stream nil))
34        (return-from read-people (length *people*))))))
35
36   (defun make-person (person-qualities)
37     "Takes the split list of qualities. Performs some additional parsing and uses pe
     rmatted plist."
38     (person-form (loop for item in person-qualities collecting
39       (if (search "(" item)
40           (parse-name item)
41           (if (search "-" item)
42           nil
43           item)))))
44
45   (defun parse-name (name-id)
46     "Takes a name-id number string and returns a list with a name and id, parsed."
47     (loop for item in (split-sequence #\( name-id)
48       if (search ")" item) collect (parse-integer (string-trim "()" item))
49       else collect (string-trim " " item)))
50
51   (defun prompt-read (prompt)
52     "Prints the given prompt and returns the user's response. From Practical Common
53     (format *query-io* "~a: " prompt)
54     (force-output *query-io*)
55     (read-line *query-io*))
56
57   (defun print-ancestry ()
58     "Prints the ancestors of the people in *people*."
59     (loop for person in *people* do
60       (format t "Printing ancestors of ~a~%" (getf person :name))
61       (if (has-parentsp person)
62           (print-ancestors person)
```

```
 63            (format t "No ancestors.~%"))
 64        (format t "~%")))
 65
 66    (defun print-ancestors (person)
 67      "Prints the ancestors of the given person."
 68      (print-person person)
 69      (loop for ancestor in (get-parents person)
 70        if ancestor do (print-ancestors ancestor)))
 71
 72    (defun has-parentsp (person)
 73      "Returns T if the given person has parents who have records, nil otherwise."
 74      (if (get-parents person)
 75          T
 76          nil))
 77
 78    (defun get-parents (person)
 79      "Returns a list with the parents of the passed person."
 80      (loop for id in (list (getf person :mom) (getf person :dad))
 81        if (person-existsp id) collect (get-person id)))
 82
 83    (defun person-existsp (id)
 84      "Returns T if a person with the given id has an entry, nil otherwise. Used to se
 85      (if id
 86          (if (get-person id) T)
 87          nil))
 88
 89    (defun get-person (id)
 90      "Returns the person with the given id, or nil if the person does not exist."
 91      (loop for person in *people*
 92        do (if (eq (getf person :id) id)
 93            (return-from get-person person)))
 94      nil)
 95
 96    (defun print-person (person &optional (short T))
 97      "Prints a representation of the given person - by default, just their name and i
 98      (if short
 99          (format t "~a (~a).~%" (getf person :name) (getf person :id))
100          (format t "~{~a:~10t~a~%~}~%" person)))
101
102    (defun main ()
103      "Wraps the other functions, reading people from the file and printing the ancest
104      (setf *people* nil)
105      (let ((people-count (read-people (prompt-read "Enter data filename"))))
106        (if people-count
107          (print-ancestry)
108          (format t "Unable to open file. Check filename.~%"))))
109    ;
110    ;
111    ;My code ends here (other than the (main) at the bottom)
112    ;
113    ;
114
115    (defun split-sequence (delimiter seq &key (count nil) (remove-empty-
subseqs nil) (from-end nil) (start 0) (end ni
       l) (test nil test-supplied) (test-not nil test-not-
supplied) (key nil key-supplied))
116      "Return a list of subsequences in seq delimited by delimiter.
117
118    If :remove-empty-subseqs is NIL, empty subsequences will be included
119    in the result; otherwise they will be discarded.  All other keywords
120    work analogously to those for CL:SUBSTITUTE.  In particular, the
121    behaviour of :from-end is possibly different from other versions of
122    this function; :from-end values of NIL and T are equivalent unless
123    :count is supplied. The second return value is an index suitable as an
124    argument to CL:SUBSEQ into the sequence indicating where processing
125    stopped."
126      (let ((len (length seq))
```

```
127            (other-keys (nconc (when test-supplied
128                                 (list :test test))
129                               (when test-not-supplied
130                                 (list :test-not test-not))
131                               (when key-supplied
132                                 (list :key key)))))
133        (unless end (setq end len))
134        (if from-end
135            (loop for right = end then left
136                  for left = (max (or (apply #'position delimiter seq
137                                             :end right
138                                             :from-end t
139                                             other-keys)
140                                      -1)
141                                  (1- start))
142                  unless (and (= right (1+ left))
143                              remove-empty-subseqs) ; empty subseq we don't want
144                  if (and count (>= nr-elts count))
145                  ;; We can't take any more. Return now.
146                  return (values (nreverse subseqs) right)
147                  else
148                  collect (subseq seq (1+ left) right) into subseqs
149                  and sum 1 into nr-elts
150                  until (< left start)
151                  finally (return (values (nreverse subseqs) (1+ left))))
152          (loop for left = start then (+ right 1)
153                for right = (min (or (apply #'position delimiter seq
154                                            :start left
155                                            other-keys)
156                                     len)
157                                 end)
158                unless (and (= right left)
159                            remove-empty-subseqs) ; empty subseq we don't want
160                if (and count (>= nr-elts count))
161                ;; We can't take any more. Return now.
162                return (values subseqs left)
163                else
164                collect (subseq seq left right) into subseqs
165                and sum 1 into nr-elts
166                until (>= right end)
167                finally (return (values subseqs right))))))
168
169    (defun split-sequence-if (predicate seq &key (count nil) (remove-empty-
subseqs nil) (from-end nil) (start 0) (end
         nil) (key nil key-supplied))
170      "Return a list of subsequences in seq delimited by items satisfying
171    predicate.
172
173    If :remove-empty-subseqs is NIL, empty subsequences will be included
174    in the result; otherwise they will be discarded.  All other keywords
175    work analogously to those for CL:SUBSTITUTE-IF.  In particular, the
176    behaviour of :from-end is possibly different from other versions of
177    this function; :from-end values of NIL and T are equivalent unless
178    :count is supplied. The second return value is an index suitable as an
179    argument to CL:SUBSEQ into the sequence indicating where processing
180    stopped."
181      (let ((len (length seq))
182            (other-keys (when key-supplied
183                          (list :key key))))
184        (unless end (setq end len))
185        (if from-end
186            (loop for right = end then left
187                  for left = (max (or (apply #'position-if predicate seq
188                                             :end right
189                                             :from-end t
190                                             other-keys)
191                                      -1)
```

```
192                     (1- start))
193                unless (and (= right (1+ left))
194                            remove-empty-subseqs) ; empty subseq we don't want
195                if (and count (>= nr-elts count))
196                ;; We can't take any more. Return now.
197                return (values (nreverse subseqs) right)
198                else
199                collect (subseq seq (1+ left) right) into subseqs
200                and sum 1 into nr-elts
201                until (< left start)
202                finally (return (values (nreverse subseqs) (1+ left))))
203          (loop for left = start then (+ right 1)
204                for right = (min (or (apply #'position-if predicate seq
205                                       :start left
206                                       other-keys)
207                                     len)
208                                 end)
209                unless (and (= right left)
210                            remove-empty-subseqs) ; empty subseq we don't want
211                if (and count (>= nr-elts count))
212                ;; We can't take any more. Return now.
213                return (values subseqs left)
214                else
215                collect (subseq seq left right) into subseqs
216                and sum 1 into nr-elts
217                until (>= right end)
218                finally (return (values subseqs right))))))
219
220    (defun split-sequence-if-not (predicate seq &key (count nil) (remove-empty-
subseqs nil) (from-end nil) (start 0)
       (end nil) (key nil key-supplied))
221      "Return a list of subsequences in seq delimited by items satisfying
222    (CL:COMPLEMENT predicate).
223
224    If :remove-empty-subseqs is NIL, empty subsequences will be included
225    in the result; otherwise they will be discarded.  All other keywords
226    work analogously to those for CL:SUBSTITUTE-IF-NOT.  In particular,
227    the behaviour of :from-end is possibly different from other versions
228    of this function; :from-end values of NIL and T are equivalent unless
229    :count is supplied. The second return value is an index suitable as an
230    argument to CL:SUBSEQ into the sequence indicating where processing
231    stopped."                ; Emacs syntax highlighting is broken, and this helps: "
232      (let ((len (length seq))
233        (other-keys (when key-supplied
234                      (list :key key))))
235        (unless end (setq end len))
236        (if from-end
237            (loop for right = end then left
238                  for left = (max (or (apply #'position-if-not predicate seq
239                                             :end right
240                                             :from-end t
241                                             other-keys)
242                                      -1)
243                                  (1- start))
244                  unless (and (= right (1+ left))
245                              remove-empty-subseqs) ; empty subseq we don't want
246                  if (and count (>= nr-elts count))
247                  ;; We can't take any more. Return now.
248                  return (values (nreverse subseqs) right)
249                  else
250                  collect (subseq seq (1+ left) right) into subseqs
251                  and sum 1 into nr-elts
252                  until (< left start)
253                  finally (return (values (nreverse subseqs) (1+ left))))
254          (loop for left = start then (+ right 1)
255                for right = (min (or (apply #'position-if-not predicate seq
256                                       :start left
```

```
                      other-keys)
                 len)
                 end)
           unless (and (= right left)
                       remove-empty-subseqs) ; empty subseq we don't want
           if (and count (>= nr-elts count))
           ;; We can't take any more. Return now.
           return (values subseqs left)
           else
           collect (subseq seq left right) into subseqs
           and sum 1 into nr-elts
           until (>= right end)
           finally (return (values subseqs right))))))

;;; clean deprecation

(defun partition (&rest args)
  (apply #'split-sequence args))

(defun partition-if (&rest args)
  (apply #'split-sequence-if args))

(defun partition-if-not (&rest args)
  (apply #'split-sequence-if-not args))

(define-compiler-macro partition (&whole form &rest args)
  (declare (ignore args))
  (warn "PARTITION is deprecated; use SPLIT-SEQUENCE instead.")
  form)

(define-compiler-macro partition-if (&whole form &rest args)
  (declare (ignore args))
  (warn "PARTITION-IF is deprecated; use SPLIT-SEQUENCE-IF instead.")
  form)

(define-compiler-macro partition-if-not (&whole form &rest args)
  (declare (ignore args))
  (warn "PARTITION-IF-NOT is deprecated; use SPLIT-SEQUENCE-IF-NOT instead")
  form)

(pushnew :split-sequence *features*)

(main)
```

```c
/*
Programming Language Skills and Concepts exam
Problem 1
Part 2
Richard Scruggs
11/9/2010
Licensed under the Creative Commons Attribution 3.0 Unported License.
see http://creativecommons.org/licenses/by/3.0/ for license text.*/

/*
Known bugs: qsort does not sort the data correctly.
The program does not exit cleanly. The use of exit() from main suppresses the
segfault, but examination in gdb still shows execution going out of main() at
the end. I suspect that I'm not cleaning up a bit of memory and the automatic
attempt at same isn't working properly--exit() bypasses that attempt.*/

#define DATEMAX 11
#define NAMEMAX 30
#define LINEMAX 100
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct person{
    char first_name[NAMEMAX];
    char last_name[NAMEMAX];
    int id;
    char birth[DATEMAX];
    char death[DATEMAX];
    int mom;
    int dad;
};

typedef struct person person;

int print_people(person** people, int people_count);
int free_people(person** people, int people_count);
int person_compare(const void* person_1, const void* person_2);
int read_people (char* filename, person** people, int* people_count, int* people_cap);
int insert_person(char person_data[6][LINEMAX], person** people, int people_count);

int main (int argc, char* argv[]){
    /*Takes the given filename and passes it, with some initialized variables,
    to read_people. Sorts and prints the prople, then frees the memory they
    use.*/
    if (argc != 2){
        printf("Usage: %s datafile\n", argv[0]);
        return 0;
    }
    person* people;
    int people_count = 0;
    int people_cap = 5;
    if ((read_people(argv[1], &people, &people_count, &people_cap)) == -1){
        printf("Error reading people from file. Check filename.\n");
        printf("Exiting.\n");
        return 1;
    }
    qsort(&people, people_count, sizeof(person*), person_compare);
    //the above does not behave as it should.
    print_people(&people, people_count);
    free_people(&people, people_count);
    exit (0);
}

int print_people(person** people, int people_count){
    /*Prints people_count of the passed people[], starting from people[0] to
    standard out, one person per line, with a newline after the set.*/
    int i;
    for (i=0; i<people_count; i++){
            printf("%15s %15s (%d), born: %11s, died: %11s. Mom id: %6d. Dad id: %6d.\n",
                people[i]->first_name, people[i]->last_name, people[i]->id, people[i]->birth,
                people[i]->death, people[i]->mom, people[i]->dad);
    }
    printf("\n");
    return 0;
}

int free_people(person** people, int people_count){
    /*Frees the memory used by people_count people, starting from people[0].*/
    int i;
    for (i=0; i<people_count; i++){
        free(people[i]);
    }
    return 0;
}

int person_compare(const void* person_1, const void* person_2){
    /*Expects pointers to persons, cast as void*. Compares their names, last
    first.*/
    person* person_a = (person* ) person_1;
    person* person_b = (person* ) person_2;
    if (strncmp(person_a->last_name, person_b->last_name, NAMEMAX) != 0)
```

```
93              return strncmp(person_a->last_name, person_b->last_name, NAMEMAX);
94          return strncmp(person_a->first_name, person_b->first_name, NAMEMAX);
95      }
96
97      int read_people (char* filename, person** people, int* people_count, int* people_cap){
98          /*Reads persons from the given filename into people. Modifies people_count
99          to reflect the number read, and people_cap to reflect the capacity of
100         people. Returns 0 on success, -1 on failure.*/
101         FILE* in_file;
102         if ((in_file = fopen(filename, "r")) == NULL)
103             return -1;
104         /*I use a temporary array, read five lines into it, pass it to
105         insert_person, check to see it's reached EOF, and repeat.*/
106         char temp_person[6][LINEMAX];
107         *people = (person *) malloc(*people_cap * sizeof(person*));
108         int place_count = 0;
109         while (fgets(temp_person[place_count], LINEMAX, in_file) != NULL){
110             if (*people_count >= *people_cap){
111                 *people_cap = *people_cap * 2;
112                 *people = (person *) realloc(*people, *people_cap * sizeof(person*));
113             }
114             place_count++;
115             if (place_count == 5){
116                 insert_person(temp_person, people, *people_count);
117                 *people_count = *people_count + 1;
118                 if (fgets(temp_person[place_count], LINEMAX, in_file) == NULL){
119                     break;
120                 }
121                 place_count = 0;
122             }
123         }
124         fclose(in_file);
125         return 0;
126     }
127
128     int insert_person(char person_data[6][LINEMAX], person** people, int people_count){
129         /*Mallocs space for a new person and inserts them in people[people_count].*/
130         people[people_count] = (person*) malloc(sizeof(person));
131         sscanf(person_data[0], "%*s %s %s (%d)\n", people[people_count]->first_name, people[people_count]->last_name, &p
    eople[people_count]->id);
132         sscanf(person_data[1], "%*s %s\n", people[people_count]->birth);
133         sscanf(person_data[2], "%*s %s\n", people[people_count]->death);
134         sscanf(person_data[3], "%*s %*s %*s (%d)\n", &people[people_count]->dad);
135         sscanf(person_data[4], "%*s %*s %*s (%d)\n", &people[people_count]->mom);
136         return 0;
137     }
```

Category number (0 = omitted)	* = given, # = added, x = omitted	Term
0	x	stack overflow
0	x	macro
0	x	pattern
1	*	argument
1	*	function
1	*	global
1	*	pass by reference
1	*	pass by value
1	*	variable
1	*	dynamic
1	*	static
1	*	namespace
1	*	scope
1	*	closure
1	*	symbol
1	*	lexical
2	*	collection
2	*	list
2	*	vector
2	*	data structure
2	*	hash
3	*	compiled
3	*	interpreted
3	*	link
4	*	overload
4	*	API
4	*	class
4	*	inheritance
4	*	interface
4	*	method
4	*	object
4	*	package
5	*	iterate
5	*	recursion
5	*	side effect
5	#	return value
6	*	comment
6	*	syntactic sugar
6	*	exception
6	*	test
6	*	throw

Discussion of Categorized Terms

Category 1

This category consists of terms relating to functions, variables, and scope.

Examples:

A function in Python that takes several variables as arguments:

```
def sum(a, b, c):
     return a+b+c
```

The same function in C, illustrating the difference between static typing (C) and dynamic typing (Python):

```
int sum(int a, int b, int c){
     return a + b + c
}
```

Defining a global (dynamic) variable and a lexical variable in Lisp:

```
(defvar *two* 2) ;global/dynamic scope
(let ((two 2))) ;lexical scope
```

A warning when defining a function from a file in Lisp when a function with that name has already been defined at the top level:

```
DEFUN/DEFMACRO: redefining function MAIN in
/mnt/big/code/1-1.fas, was defined in top-level
```

A function in Lisp that adds two numbers that are declared as fixnums:

```
(defun add (x y)
     (declare (fixnum x y))
     (+ x y))
```

Discussion:

This category can be easily grouped into several subcategories to make discussion easier. Variables in all the languages are passed by value, although C's pointers offer the option of passing by reference. Also, all languages encourage using lexically scoped variables but offer the option of global variables—note, though, that Lisp's lexical scoping, with `(let)` is perhaps more explicit than C or Python. Functions and their arguments are relatively similar among the three languages, although C's static typing creates a different syntax. While C and Python are clearly dynamically and statically typed respectively, Lisp offers a choice between the two: a variable is usually dynamically typed, but a programmer has the option of declaring a variable's type, which the compiler will then use to optimize.

Category 2

This category consists of data structures and related terms.

Examples:

The list example here is a little messy; Lisp and Python support more types of collections than C, so I show a list for them and an array for C.

```
testList = [4, 5, 6] #a list in Python.
```

```
(list 4 5 6) ;a list in Lisp.
int[3] array = {4,5,6}; //an array in C.
```

A data structure in C:

```
struct person {
     char* name;
     int age;
}
```

A dictionary in Python:

```
testDictionary = {'sape': 413, 'guido': 412, 'jack': 409}
```

Discussion:

The data structures available and encouraged in each language differ significantly. Lisp encourages use of lists; Python offers several data structures but does not strongly support any over the others; and while C offers arrays, many C programs implement their own data structures.

Category 3

This category's terms relate to how code written in a language is actually executed on a computer.

Examples:

Compiling a file with clisp (with the clisp header removed for readability):

```
[shaggy@shaggy-pc code]$ clisp -c 1-1.lisp
;; Compiling file /mnt/big/code/1-1.lisp ...
;; Wrote file /mnt/big/code/1-1.fas
0 errors, 0 warnings
Bye.
```

Discussion:

The definitions of these terms do not vary by language; the difference is which of the terms apply. The typical distinction, whether a language is compiled or interpreted, is becoming increasingly muddied—if it ever was clear. While C is a typical example of a compiled language, there exist C interpreters, and while Python is a typical example of an interpreted language, it may be compiled. Lisp, as it often does, does not fit neatly into either box: it can be compiled, but some interpreters only can compile it to bytecode. Also, it's typically run in a read-eval-print loop, something usually characteristic of an interpreted language.

Category 4

This category consists of terms relating to classes, objects, and libraries.

Examples:

An example of operator overloading in Python:

```
>>> date.today() - date(2001,5,16)
datetime.timedelta(3464)
>>> 5 - 4
```

```
1
#the - operator works on numbers and date objects
```

A class that inherits from another class in Lisp (with class contents elided):

```
(defclass bank-account () ...)
(defclass checking-account (bank-account) ...)
```

Including code from another file in C:

```
#include <stdio.h>
```

Discussion:

The role of objects and libraries differs greatly between the languages. C, of course, has no support for objects, and Lisp has no native support for objects, only adding a rudimentary implementation in later dialects such as CommonLisp. Python, meanwhile, has a well-developed object system.

There are many packages available for Python, Lisp, and C, although the method of accessing those packages—and even the definition of what consists of a package—varies. C allows one to use functions from other files, CommonLisp offers the ASDF packaging system, and Python includes many packages in its extended library and has many more online.

Category 5

The terms in this category relate to behavior of functions.

Examples:

The archetypal, recursive Fibonacci function in Python:

```
def fib(n):
     if n < 2:
          return 1
     else:
          return fib(n-1) + fib(n-2)
```

An iterative Fibonacci function in C:

```
int fib(int n)
{
  int first = 0, second = 1;
  int tmp;
  while (n--)
    {
      tmp = first+second;
      first = second;
      second = tmp;
    }
  return first;
}
```

A function with a side effect and a return value:

```
(defun test-print ()
     (format t "Hello.~%"))
```

Discussion:

The biggest difference between the three languages among the concepts in this category is that is Lisp, everything has a return value. Iteration and recursion are supported by all the languages, as are side effects. From a philosophical standpoint, Lisp functions are encouraged to use the return values when possible and avoid side effects unless necessary, C's limitations on return values forces greater reliance on side effects, and Python holds a middle ground.

Category 6

The terms in this category are things that are helpful, but not integral to running code.

Examples:

A doctest in Python:

```
>>> isPalindromic("test")
False
```

A simple exception handler in Lisp:

```
(handler-case
    (progn
      (do-stuff)
      (do-more-stuff))
  (some-exception (se) (recover se)))
```

Syntactic sugar in C, easing array access:

```
int array[100]; //an empty array with space for 100 ints
array[0] = 5;
```

Discussion:

This category, as stated above, consists of features that make things easier when programming. Comments and syntactic sugar make it easier to understand what code is doing, tests ensure that the code actually work as it should, and exception handling makes sure that when code does not execute as expected, program execution continues in a predictable manner.

These terms, more than the other categories, are reasonably similar across the languages. Comments and syntactic sugar are universal, and while C does not have proper exception handling as the other languages do, it is expected that a C programmer should anticipate and attempt to check for error conditions.

C, Lisp, and Python

"Comparisons of programming languages," says Paul Graham, "either take the form of religious wars or undergraduate textbooks so determinedly neutral that they're really works of anthropology."[1] I believe that Graham is mistaken—not only is anthropology often far from neutral, but it is also possible to write a comparison of programming languages that mildly expresses an opinion or preference. In this paper, I will attempt to compare Python, C, and Lisp in such a fashion.

Python, C, and Lisp can be grouped together in many different ways. C is the most imperative of the set, Lisp the most extensible, and Python is the most interpreted. All of them are sufficiently general-purpose that one may use them for any purpose, but each also has certain strengths and weaknesses that suit it more for one task over another.

I will start with language performance, as I find it more clearly defined than the other differences between the languages. The most well-known strength of C, its speed, can be illustrated with a benchmark of the three short programs written for question 1-1 of this exam. These programs calculated the largest palindromic number with three-digit factors—a task that can be sped up somewhat with clever tricks, but one that requires a great deal of multiplication nonetheless. I wrote programs to do this in Python, Lisp, and C; the method used to perform the calculation is relatively similar among the languages. I timed all three versions of the program with the `time` utility in Linux. The execution times are displayed below.

3-digit factors	Python	Lisp[2]	C

[1] http://www.paulgraham.com/avg.html, footnote 6.

[2] Clisp, the Lisp implementation used, does not compile to native code. I had it compile to bytecode, but Lisp is still handicapped here.

User	0.160s	0.060s	0.003s
Sys	0.013s	0.037s	0.000s
Total	0.173s	0.097s	0.003s

The C code is faster by two orders of magnitude. However, it's generally acknowledged that C's advantage in machine runtime is countered by a disadvantage in programmer-time. In this case, the C code is about twice as long as the corresponding Lisp and Python code, and suffers from various limitations that are not present in either of the other versions.

This is perhaps the best illustration of the performance argument. When writing a program that will be run a few times and where the speed differential is relatively insignificant, it's typically faster to write in languages like Lisp or Python that handle more of the small implementation details—the fact that the C program runs a second faster doesn't matter when it takes thirty minutes longer to write it. The extension of this, then, is the belief that when writing code that must run as fast as possible, C or a similarly low-level language should be used.

I do not hold to this latter belief, however. Take the example of ITA Software, a startup that was recently acquired by Google.[3] ITA writes software to solve a computationally hard problem – finding the cheapest airline tickets. Their software, written largely in Lisp,[4] is fast enough to be able to look across an entire month to find the cheapest ticket. ITA's method reflects the current climate of computation—many hard computational problems that are solved today are solved not with brute force but with carefully designed algorithms. The fact that Lisp often runs more slowly than equivalent code in C is made irrelevant by the higher-level features

[3] http://www.google.com/press/ita/.
[4] This was stated in a 2001 email, see http://www.paulgraham.com/carl.html.

of the language which enable programmers to more effectively attack problems. Python, on the other hand, does not add features that Lisp does not have, but sacrifices speed to make many of Lisp's features easier to understand and use.[5] Ruby takes another step along that curve, being even slower than Python, but also adding more features from Lisp.

Deciding what language to use on performance alone, though, steps close to premature optimization, which is the root of all evil.[6] Another important factor when choosing a language is one's intended audience. Whenever programming for more than personal use, one ought to consider who else will examine and use the code, and whether those people will be able to easily understand it. In this regard, I consider library size, code size, and code complexity all to be important. The larger the library, the less likely that the audience is familiar with every function used. The larger the code size, the less likely that the reader can hold it in her head at once—good organization can help with this problem, but not eliminate it entirely. Finally, the more complex the code, the more likely the reader won't understand what it does. Of the languages discussed here, I see, on average, Lisp having the largest standard library, C the smallest; Lisp having the most compact code, C the least; and Python having the least complex code. Whether C or Lisp is more complex depends more on the audience and the particular code in question; I would argue that, in general, Lisp is more complex to the uninitiated, C is more complex otherwise.

C, Python, and Lisp all encourage certain styles of programming, each of which has its own strengths and weaknesses. Python, for instance, supports the philosophy that there's one way to do it, and its code is often as easy to read as pseudocode. Python code is also easier to extend and update than its C equivalent—I have not heard of any Python source code preceded

[5] See Norvig's chart comparing the two languages, http://norvig.com/python-lisp.html.
[6] "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil" – attributed at times to Knuth and C.A.R. Hoare.

by an ASCII skull and crossbones, as the display code for Gosmacs (written in C) famously was.[7] Lisp is more flexible than the other two, but there along with the many right ways to solve a problem, there are many wrong ones. I have not seen as much written about a unified style for C, but it is worth mentioning that it allows and even encourages nonstandard programming practice to a much greater degree than the other two languages.

As a final data point, I believe that Python's popularity as a teaching language is rooted in several factors that are illustrative to this comparison: Python code is easy to read, even for nonprogrammers; Python is sufficiently flexible so as to allow easy demonstration of many key programming concepts; and Python's philosophy that there's one obvious way to do it makes it easier for different people to reach the same result by the same process. Most important, Python is a language that does not require a great deal of translation to get from thought to code. In C, there is a great deal of repetitious code that must be written each time, and it is often cumbersome to write and access data structures. In Lisp, the final code is typically short and elegant, but there is often a great deal of thought required in its design. Coding in Python, on the other hand, is like following its philosophy: most of the time, there is an obvious way to do it, and that way works well enough. This tends to hold true even when one is a novice to Python.

The question of whether to use Python, C, or Lisp—or, indeed, any programming language—for a particular project is not one that can be answered decisively. Sometimes it is important to be able to develop as fast as possible, sometimes it only matters that the code run as fast as possible; most often there is some mix of requirements that requires the choice of language to be a compromise. In this paper, I have attempted to offer some benefits that C, Python, and Lisp offer as compared to one another, and some situations in which one might be preferred.

[7] http://en.wikipedia.org/wiki/Gosmacs

Second computer science plan exam :
 discrete math
   (probability and counting, information theory,
   game theory, graph theory)
 for Richard Scruggs
 from Jim Mahoney
 on Nov 16 2010

This is an open, take home exam: books or web sources are OK as long
as you cite them explicitly and as long as they aren't a drop-in
solution to the problem. Don't ask other people for help. Your job
is to convince us you understand this stuff.

These are straight pen-and-paper questions; none of 'em need coding.

This is to be done during the week of Nov 16 - Nov 23
the due date is 10am on Nov 23, before our tutorial.

As always with my exams, if you think there's a mistake in one of the
questions or it doesn't make sense, you can (a) ask for clarification,
and/or (b) make and state an explicit interpretation and do the
problem that way. (Again: the point is to demonstrate your
understanding, not to get the "right answer" per se.)

---

1. Take several situations from fiction (books/movies/tv...) that
   could plausibly be analysed with game theory (either simultaneous
   selection matrix games or sequential turn tree games) and
   see what the theory says should happen.
   Discuss successes and failures of the theory.

2. Given a language made up of N symbols x_i = (x_1, x_2, ..., x_N)
   each of which appear with probability P(x_i), show that
   the information entropy is highest when the probabilities are equal.
   Explain what this says about randomness, predicitability,
   and information.

3. Show that C(n,k) - C(n-3,k) = C(n-1,k-1) + C(n-2,k-1) + C(n-3,k-1),
   in two different ways, using a combinatoric and algebraic approach.

4. Show that if 10 positive integers sum to 151 then the sum of three
   of these numbers must be at least 46.

5. Show that any finite simple graph, except the trivial graph with a
   single vertex, has two vertices of the same degree.

6. Show that in every finite simple graph the edges can be oriented in
   such a way that the in-degree of each vertex is at most one different
   from its out-degree.

7. Bayes' Theorem says that P(x|y) = P(y|x) * P(x) / P(y).
   Define the terms, derive the result, and explain how
   it is used by computerists.

8. Is it possible to load two dice such that each of the possible totals 2,3,...12 occurs with equal probability when both are rolled?

Discrete Math Exam

Richard Scruggs

November 23, 2010

1: Game Theory Analysis of a Plausible Situation

Take several situations from fiction (books/movies/tv...) that could plausibly be analysed with game theory (either simultaneous selection matrix games or sequential turn tree games) and see what the theory says should happen.
Discuss successes and failures of the theory.

Oddly enough, it seems that the Twilight series can be analyzed with game theory to a reasonable extent. I'm making a few slight modifications to the story to make it a little easier to discuss in this fashion, but the results should still be similar enough.

The central story arc of Twilight is the love triangle between Bella, Edward, and Jacob. Of the latter two, Bella prefers Edward if he is pursuing her, but will settle for Jacob otherwise. However, if both repeatedly pursue her at the same time, she gets mad at one of them for being too competitive. It can be modeled as an iterated prisoner's dilemma: Edward and Jacob each want to pursue when the other does not. If both persue her simultaneously for n turns, Bella gets mad, chooses one at random and will not accept pursuit from that person for m turns.

I've decided to model a winning condition like so: the first one to successfully pursue Bella for w turns wins. For this condition, a 'successful' pursuit is different for each person: Jacob can only successfully pursue if Edward is not, or if they both repeatedly pursue and Bella gets mad at Edward. Edward can successfully pursue her as long as Bella is not mad at him, regardless of whether or not Jacob is pursuing.

In this model - as in the series - the odds are stacked in Edward's favor. He can easily prevent Jacob from ever succeeding by simply pursuing Bella often enough that he always stays in the picture. Indeed, as long as $n > 1$, $m > 0$, and $(m + 1) < w$, he can't lose if he always pursues. While the model is true to the series, from a game theory perspective there is no reason for Jacob to ever pursue - he cannot win unless Edward lets him.

In the series, Jacob's behavior initially follows the model - he only begins to pursue Bella after Edward has stopped his pursuit for a while. Once Edward returns, though, Jacob continues pursuit despite not having a chance at winning. Actually, this would not be difficult to add to the model - make it difficult for a player to stop pursuit once started, depending on how long they have been pursuing - but it does not fit the actions of the series itself.

I am surprised by how well the game theory model holds up in analyzing the situation - the simplicity of the model does not preclude its utility, although that may be more of a commentary on the simplicity of the series than on the utility of the theory. In the series, Jacob does continue pursuit, but he also realizes that he cannot win in a head-to-head competition with Edward and instead focuses on trying to convince Edward not to pursue Bella.

2: Information Entropy of a Language

Given a language made up of N symbols $x_i = (x_1, x_2, ..., x_N)$ each of which appear with probability

$P(x_i)$, show that the information entropy is highest when the probabilities are equal. Explain what this says about randomness, predicitability, and information.

Note: The first part of this problem appears to be pretty close to an example given in the Wikipedia entry for Entropy (information theory). I avoided looking at it after that realization - and my discussion is totally different - but should probably mention it anyway.

I think of information entropy as a measure of how much meaning there is in some given information - although that definition is more useful for compression than for an abstract discussion of entropy. The more one can infer about as-yet-unreceived information, the less entropy it has. If the information is totally random, then it is impossible to infer anything about future information and entropy is at maximum.

Given the language here, if each symbol appears with equal probability, it is essentially random data: one cannot predict the next symbol with any certainty. However, if the probabilities are not equal, one or more symbols will appear more often than others. Since it is known that these symbols appear more often, their information content is less.

The most direct relation between randomness, predictability, and information would be that the more random a language is, the harder it is to predict; the harder it is to predict, the more information is carried in it.

3: Proof

Show that $\binom{n}{k} - \binom{n-3}{k} = \binom{n-1}{k-1} + \binom{n-2}{k-1} + \binom{n-3}{k-1}$, in two different ways, using a combinatoric and algebraic approach.

Algebraic (uncommented, but hopefully clear. LaTeX cuts off the end of a line once, but that portion of the line is unchanged from the previous.)

$$\binom{n}{k} - \binom{n-3}{k} = \binom{n-1}{k-1} + \binom{n-2}{k-1} + \binom{n-3}{k-1}$$

$$\frac{n!}{k!(n-k)!} - \frac{(n-3)!}{k!(n-3-k)!} = \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-2)!}{(k-1)!(n-1-k)!} + \frac{(n-3)!}{(k-1)!(n-k-2)!}$$

$$\frac{n!}{k(n-k)!} - \frac{(n-3)!}{k(n-3-k)!} = \frac{(n-1)!}{(n-k)!} + \frac{(n-2)!}{(n-1-k)!} + \frac{(n-3)!}{(n-k-2)!}$$

$$\frac{n!}{k(n-k)(n-k-1)(n-k-2)} - \frac{(n-3)!}{k} = \frac{(n-1)!}{(n-k)(n-k-1)(n-k-2)} + \frac{(n-2)!}{(n-k-1)(n-k-2)} + \frac{(n-3)!}{(n-k-2)}$$

$$\frac{n(n-1)(n-2)}{k(n-k)(n-k-1)(n-k-2)} - \frac{1}{k} = \frac{(n-1)(n-2)}{(n-k)(n-k-1)(n-k-2)} + \frac{(n-2)}{(n-k-1)(n-k-2)} + \frac{1}{(n-k-2)}$$

$$\frac{n(n-1)(n-2)}{k(n-k)(n-k-1)} - \frac{n-k-2}{k} = \frac{(n-1)(n-2)}{(n-k)(n-k-1)} + \frac{(n-2)}{(n-k-1)} + 1$$

$$\frac{n(n-1)(n-2)}{(n-k)(n-k-1)} - (n-k-2) = \frac{k(n-1)(n-2)}{(n-k)(n-k-1)} + \frac{k(n-2)}{(n-k-1)} + k$$

$$\frac{n(n-1)(n-2)}{(n-k)(n-k-1)} - n + 2 = \frac{k(n-1)(n-2)}{(n-k)(n-k-1)} + \frac{k(n-2)}{(n-k-1)}$$

$$\frac{n(n-1)(n-2)}{(n-k)} - n(n-k-1) + 2(n-k-1) = \frac{k(n-1)(n-2)}{(n-k)} + k(n-2)$$

$$\frac{n(n-1)(n-2)}{(n-k)} - n^2 + 3n - 2 = \frac{k(n-1)(n-2)}{(n-k)}$$

$$\frac{n(n-1)(n-2)}{(n-k)} - \frac{k(n-1)(n-2)}{(n-k)} = n^2 - 3n + 2$$

$$(n-k)\frac{(n-1)(n-2)}{(n-k)} = n^2 - 3n + 2$$

$$(n-1)(n-2) = n^2 - 3n + 2$$

$$n^2 - 3n + 2 = n^2 - 3n + 2$$

Combinatoric:

This is a partly expanded recursive definition of a binomial coefficient. To cite Wikipedia, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$, with the rightmost term of that being expanded several more times in this problem.

It's easier to talk about if the terms are reordered thus: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-2}{k-1} + \binom{n-3}{k-1} + \binom{n-3}{k}$

Talking about each term, the leftmost is a simple $\binom{n}{k}$. The next chooses $k-1$ from $n-1$, thus element n, in essence, is pre-selected as k. My understanding of precisely what is happening for the rest of it is somewhat fuzzier. I can follow it from a recursive point of view, but a precise explanation is more difficult. The next three terms count the instances where n is pre-selected as not k - $\binom{n-2}{k-1}$ represents those instances where $n-1$ is pre-selected as k, $\binom{n-3}{k-1}$, those where $n-2$ is. Finally, $\binom{n-3}{k}$ represents the instances where n through $n-2$ are all pre-selected as not k.

4: Sum of Integers

Show that if 10 positive integers sum to 151 then the sum of three of these numbers must be at least 46.

First, in order for 10 integers to sum to 151, at least one integer i in the set must be greater than or equal to 16: a set of ten integers each of value 15 only sums to 150.

In order for the premise to be false - that is, for the sum of any three numbers in the set to be 45 or less - several things must be true. First, if any number is greater than 15 by x, all but one of the other numbers must be less than or equal to $15 - x$, otherwise, one can select the greater number and two other numbers greater than or equal to $15 - x$ and find a sum greater than or

equal to 46. It was established above that i must be 16 or greater to achieve a sum of 151, thus eight numbers must be 14 or less. $14 \times 8 = 112$; thus the sum of the last two numbers must be at least $151 - 112 = 39$. However, $39 + 14 > 46$.

5: Degree of Vertices of a Graph

Show that any finite simple graph, except the trivial graph with a single vertex, has two vertices of the same degree.

Assume the inverse - that there is a finite simple graph with n vertices, with each vertex having different degree. In order for this to be the case, there must be a vertex v with degree 0 - otherwise there would only be $n-1$ possible degrees for n vertices, ensuring by the pigeonhole principle that there would be two vertices with the same degree. By the same reasoning, there must also be a vertex with degree $n-1$. In order for a vertex to have degree $n-1$, it must be connected to all the other vertices - including the vertex of degree 0. Since the vertex of degree 0 cannot be simultaneously connected to no vertices and the vertex of degree $n-1$, there is a contradiction. Thus, the proposition is true.

6: Edge Orientation of a Simple Graph

Show that in every finite simple graph the edges can be oriented in such a way that the in-degree of each vertex is at most one different from its out-degree.

As a computer scientist, I'm inclined to talk about this in algorithmic terms.

While the graph has cycles, choose a cycle in the graph and orient the edges in it so as to traverse it. This should, in effect, remove the cycle from the graph - each vertex in the cycle will have gained one in-degree and one out-degree.

At the end of this, the graph will not have any cycles, only simple paths, and the vertices in the graph will still have equal in-degree and out-degree. These simple paths do not affect the in-degree and out-degree of any vertices that they pass through - the orientation can simply follow the path, giving the middle vertices one in-degree and one out-degree. The degree of the ends of the paths are affected. If there is only one path entering or leaving a vertex, this does not matter; if there are multiple paths entering or leaving, the fact that the graph has no more cycles becomes important.

Since there are no cycles, one can take a vertex that is a terminus for multiple paths and orient those paths as needed for the terminating vertex's degrees to match up properly - there is no way for the paths to return to parts of the graph being traversed by other connected paths as that would require a cycle. Thus, algorithmically, one can take each vertex that is the terminus of multiple paths, orient those paths so that the terminating vertex's degrees are correct, and repeat for the next one. The order that the multi-terminus vertices are addressed does not matter - it would take a cycle to make it impossible to orient properly. After all the paths from multi-terminus vertices are addressed, the only paths remaining connect single-terminus vertices with equal in-degree and out-degree to each other. These paths can thus be connected in any orientation; they will only differentiate in-degree and out-degree by one.

7: Bayes' Theorem

Bayes' Theorem says that $P(x|y) = \frac{P(y|x)*P(x)}{P(y)}$. Define the terms, derive the result, and explain how it is used by computerists.

See note for question 2 re Wikipedia. Also, for the discussion of Bayesian filtering, I looked at Paul Graham's "A Plan for Spam".

The definition of conditional probability is: $P(x|y) = \frac{(P(y \cap x)}{P(y)}$. This can be substituted into Bayes' Theorem: $P(x|y) = \frac{\frac{(P(x \cap y)}{P(x)} P(x)}{P(y)}$ The simplification is trivial: $\frac{P(x \cap y)}{P(y)}$.

The archetypal example of Bayes' Theorem in computer science would be that of the Bayesian spam filter. Most people receive far more spam than they do legitimate email (ham). A simple filter can look for spammy words and filter out those messages, or look for hammy words and let those messages through. The former approach suffers as spammers can avoid spammy words; the latter suffers as there are far more spams than hams. One can use Bayes' Theorem to more accurately assess whether a message that contains a certain word is spam or ham. In words: the probability that a message is spam given that it contains a certain word is equivalent to the probability that a message contains that word, given that it is spam, multiplied by the probability that any message is spam and divided by the probability that the word occurs.

Once the Bayesian filter can perform this analysis on every word in a given message, it can tell with remarkably high prabability whether that message is spam or ham; given that classification, it can then update the spamminess or hamminess of each word in the message.

8: Loading Dice

Is it possible to load two [ordinary, six-sided] dice such that each of the possible totals $2, 3, ...12$ occurs with equal probability when both are rolled?

No. 2-12 is a total of eleven different outcomes. That means that each total must, with some loading of the dice, occur $\frac{1}{11}$ of the time. Given that, I'll take the edge cases - both 2 and 12 need to come up $\frac{1}{11}$ of the time. In order to get a 2, one must roll two 1s; in order to get a 12, one must roll two 6s. Thus, $P(6_a \cap 6_b) = \frac{1}{11}$ and $P(1_a \cap 1_b) = \frac{1}{11}$. In order to achieve this, at least one die must come up 6 with probability greater than or equal to $\sqrt{\frac{1}{11}} = 0.3015$; at least one die must come up 1 with at least the same probability.

1 and 6 sum to 7; an edge case of another sort. As stated before, we must have at least one heavily weighted 6 and one heavily weighted 1. If the two are on different dice, loading them to achieve the stated result cannot be possible: the probability of rolling a 7 is at least $\sqrt{\frac{1}{11}} \times \sqrt{\frac{1}{11}} = \frac{1}{11}$ with only one permutation of a 1 and 6. The probability of rolling the other permutation of 1 and 6 must be nonzero to roll the 2 and 12; this pushes the probability of rolling a 7 past $\frac{1}{11}$.

If it is possible to weight the dice to achieve this result, then, the heavily weighted 1 and 6 cannot be on different dice. If they are on the same die, $P(1_a) \geq 0.3105$ and $P(6_a) \geq 0.3105$; $P(1_a \cap 1_b) = \frac{1}{11}$ and $P(6_a \cap 6_b) = \frac{1}{11}$. Here, there are a few possibilities to address: Ei-

ther $P(1_a) = P(6_a)$ or $P(1_a)! = P(6_a)$. If $P(1_a) = P(6_a)$, then $P(1_b) = P(6_b)$ as well; thus $P(1_a \cap 1_b) = P(6_a \cap 6_b) P(1_a \cap 6_b) = P(6_a \cap 1_b) = \frac{1}{11}$. As a 7 can be formed from two pairs that both appear with $\frac{1}{11}$ possibility, $P(1_a)$ cannot equal $P(6_a)$.

Finally, if $P(1_a)! = P(6_a)$, one of the probabilities must be greater; assume $P(1_a) > P(6_a)$. Given that, $P(6_b) > P(1_b)$. However, this means that $P(1_a \cap 6_b) > P(1_a \cap 1_b)$, and as $P(1_a \cap 1_b) = \frac{1}{11}$, $P(1_a \cap 6_b) > \frac{1}{11}$ by itself. Thus, it's not possible to load two dice so that each total occurs with equal probability.

Third computer science plan exam :
 algorithms
 for Richard Scruggs
 from Jim Mahoney
 on Nov 30 2010

This is an open, take home exam: books or web sources are OK as long as you cite them explicitly and as long as they aren't a drop-in solution to the problem. Don't ask other people for help. Your job is to convince us you understand this stuff.

This is to be done during the week of Nov 30 - Dec 7
the due date is 10am on Nov 10. (Though you should put up a marker for it and a "it'll be here soon" if that's too close to the plan mail date ... which we haven't set yet.)

As always with my exams, if you think there's a mistake in one of the questions or it doesn't make sense, you can (a) ask for clarification, and/or (b) make and state an explicit interpretation and do the problem that way. (Again: the point is to demonstrate your understanding, not to get the "right answer" per se.)

---

1. Choose any one of the standard sorting algorithms.

   (a) Describe what you expect the O() run time of this to be, and explain why and and on what sort of problems.

   (b) Implement and test it in a language of your choice. (Don't use a built-in or library sorting routine.)

   (c) Run it on various size lists of randomly generated numbers, recording the number of steps (for any reasonable notion of "steps") the algorithm takes to run. Show explicitly with a plot of the data from this "experiment" that the O() behavior is as expected.

2. In a language of your choice, illustrate both a depth-first and breadth-first tree search using a stack and a queue for a small "sliding block" puzzle.

   A sample search might be to get from

start	finish
2 1 3	1 2 3
5 4 6	4 5 6
7 8 .	7 8 .

   where the "." is the empty square; the two possible first moves slide either the 6 or the 8 to bottom right corner.

Is either of these types of tree searches better than the other for this problem?

3. You're given a list of N names and told that you'll need to search the list for a given name M times. The following techniques are suggested:

   * a hash table
   * a heap
   * sequential search
   * sorting followed by binary search

   Discuss the time efficiency of these approaches as the size of N and M vary. Which one would you suggest and why? Would your answer change if you need to change the list of names by adding and deleting names (say, P times) between searches? (You don't have to write any code for this one, though you can if you'd like.)

4. Compare and contrast the LZW and Huffman coding methods for lossless data compression, including their strengths and weaknesses.

   Give an example of each algorithm applied to the same text string. (You can do the example by hand or code it; the point is to demonstrate clearly your understanding of what's going on.)

Algorithms Exam
Richard Scruggs
12/5/2010

1. Choose any one of the standard sorting algorithms.

(a) Describe what you expect the O() run time of this to be, and explain why and on what sort of problems.

(b) Implement and test it in a language of your choice. (Don't use a built-in or library sorting routine.)

(c) Run it on various size lists of randomly generated numbers, recording the number of steps (for any reasonable notion of "steps") the algorithm takes to run. Show explicitly with a plot of the data from this "experiment" that the O() behavior is as expected.

I'm using Quicksort as my chosen sorting algorithm. Quicksort works by choosing a pivot value in the data to be sorted, then comparing each value to the pivot. It then has two lists; values greater than the pivot and values less than the pivot. It then runs recursively on each of the smaller lists to sort them, and combines the sorted lists.

Quicksort, best-case, is $O(n \log n)$. In the optimal case, each pivot is the precise middle of the list, thus each partitioning process divides the list exactly in half. This means that there are log n levels of recursion, each performing about n comparisons. Worst case, each pivot is the greatest or least element in the set, which makes the partition process essentially useless. There are then n levels of recursion, each still performing about n comparisons, pushing the sort to $O(n^2)$. There are several implementation tricks that are used to prevent the pivot from being an end element, but I did not use any of them when programming my simple implementation.

See the attached 'quicksort.py' file for the algorithm itself; see the 'quicksort.pdf' file for the test data and plot.

I tested my implementation of quicksort on random data of sizes $0<n<200000$, in increments of 10000. I then plotted it against $n \log n$ (multiplied by 10^{-6} for scale). I also compared Python's built in sorting function, Timsort. As can be seen on the plot, my implementation follows the trend for $n \log n$ pretty well, but is far worse than Timsort. What I'm really taking away from this is that sorting is a problem that many people have put a lot of effort into. As a result, there is almost always an implementation in the programming language spec itself, and I can trust that implementation to work well in almost all cases. I can easily program an implementation of Quicksort or Heapsort myself, but most of the time, that implementation will be missing various small tricks that speed up performance significantly.

2. In a language of your choice, illustrate both a depth-first and breadth-first tree search using a stack and a queue for a small "sliding block" puzzle.
[example removed]
Is either of these types of tree searches better than the other for this problem?

I wrote code to do this, but discovered that the problem was actually a bit more complicated than I had first thought. I initially generated random orderings of the puzzle, then tried to solve them, but many random orderings are not solvable. The code now disorders the puzzle by making a random sequence of moves before it tries to solve it.

See the attached 'blockpuzzle.py' file for the code for this problem.

Breadth-first search is much easier to use than depth-first for this particular problem. It is easy for a depth-first search to get stuck in a loop and continue making a cycle of moves indefinitely, but a breadth-first search will find the shallowest solution.

This disadvantage is clearly illustrated when running the code several times - it solves each puzzle with breadth-first and depth-first, then prints the number of moves in each search's solution. Depth-first typically has a longer solution, or takes too many moves to find a solution and is cut off. This is particularly evident when breadth-first came up with a solution that required more than a few moves - the chances were much greater that depth-first went down the wrong path and had to be cut off.

If examined positions were stored somehow, the depth-first search would work much better - the algorithm would actually dead-end in some circumstances, and would be forced to find a solution eventually. Similarly, using an iterative deepening depth-first search would also have better results.

3. You're given a list of *N* names and told that you'll need to search the list for a given name *M* times. The following techniques are suggested:

* a hash table
* a heap
* sequential search
* sorting followed by binary search

Discuss the time efficiency of these approaches as the size of *N* and *M* vary. Which one would you suggest and why? Would your answer change if you need to change the list of names by adding and deleting names (say, *P* times) between searches? (You don't have to write any code for this one, though you can if you'd like.)

Assumptions: I'm assuming that I'm using an imperfect hash function that will create collisions, and using a chained hash table to deal with that. I'm also assuming the use of a binary heap as it is the simplest.

Hash table: Hash tables are interesting in that their worst case is *O(n)* to retrieve a single element - if all the inserted elements have collided and chaining is being used. Insertion and removal times echo the insertion and deletion times of the particular list used for chaining, with insertion adding to the end and deletion removing from a certain point. As such, the worst case construction time for a hash table is $O(n^2)$.
However, if the worst-case times of a hash table are an issue, either a terrible hash function is being used or the data is specially constructed to produce a worst case. The insertion time, with a hash table of size *S* and a list with insertion time *O(n)*, is actually *O(1 + n/s)*; thus, while insertion is *O(n)*, in practice the size of the hash table and the distribution of the hash function

should create near constant insertion time, giving a typical creation time of close to *O(n)*. Similarly, deletion time follows the deletion time of the list, but the hash used should make each list very short, giving near constant deletion time.

Heap: The initial creation of the heap is *O(n log n)*. Retrieval time is *O(log n)*, deletion time is *O(log n)*, and insertion time is *O(log n)*.

Sequential search: There is no initial setup time. Retrieval of a name is *O(n)*, deleting a name is *O(n)*, and adding a name is *O(1)*.

Sorting, binary search: In practice, this one looks similar to the heap: it's *O(n log n)* with a decent sort algorithm to sort the list, *O(log n)* to retrieve a name, and *O(log n)* to delete a name. Time efficiency of insertion depends on exactly how it's done - if it's done as a sorted insert, with a binary search, it's *O(log n)*, but if it's just tacked on at the end and the entire list is resorted, the efficiency depends on the sort algorithm and how effective it is at sorting nearly-sorted data.

I'll talk about the heap and the binary search together as, when talking about a binary heap, the two are almost the same for the purposes used here. The creation time of the heap is longer than the creation time of any of the other data structures, but heaps are less affected by worst-case data than hash tables. Other than that, though, these approaches do not offer many advantages over the hash table in this problem.

Sequential search, in a few narrow cases, actually does offer a few advantages over hash tables and heaps. If there are no retrievals or deletions, sequential search is by far the fastest data structure for this problem. Unfortunately, there are very few realistic cases where retrievals or deletions will not be needed, and the *O(n)* cost of every retrieval and deletion makes it prohibitively slow after only a few such operations.

The problem describes a situation that is really almost tailor-made for a hash table. The data never needs to be sorted, it only needs to be arranged for easy retrieval/addition/deletion by keys. It's certainly possible to do this with another data structure, but there really isn't much of a reason to ever do so unless there are memory constraints.

That said, there are several possible caveats to the use of a hash table. If I knew the size of the list of names initially and could choose a proper size for the hash table, I would almost certainly want to use the hash table. Taking the reasonable-case numbers of *O(n)* for initial setup and *O(1)* for retrieval, insertion, and removal, it is by far the fastest method. On the other hand, if I did not know the initial size, or if there were a significant number of later insertions, the hash table might have to be resized to continue to be efficient, which would slow things down somewhat. Even then, though, assuming resizing doubles the size of the table and assuming a constant-time insertion, the total insertion time would not be more than *2 * (n + p)*, and the retrieval, insertion, and removal should stay effectively constant.

The most serious drawbacks to the hash table do not come into play in this problem. First, hash tables take up a fair bit more space than the other data structures. Almost any realistic use of names would use little enough space to begin with that the space overhead of the table would not

be a serious issue. Next, hashing takes longer than sorting if the data that must be hashed is large - in order to hash a large file, for example, the entire file must be examined; in order to sort that file, only enough must be examined to compare the file with others.

4. Compare and contrast the LZW and Huffman coding methods for lossless data compression, including their strengths and weaknesses.

Give an example of each algorithm applied to the same text string. (You can do the example by hand or code it; the point is to demonstrate clearly your understanding of what's going on.)

LZW and Huffman coding have several similarities. Both are based on creating shorter representations of frequently occurring subsequences of data, but the way they create those representations is different. In a nutshell, LZW scans the data stream and builds a dictionary based on the first sequences it finds, while Huffman coding chooses which characters or subsequences to encode based on frequency analysis. LZW generally produces better results, but Huffman can adapt to source data as it is being encoded, letting it perform well on data that has a changing probability distribution. LZW, meanwhile, is unable to discard the dictionary it forms, meaning that it can perform well on data that is consistent throughout, but less well on data that initially has one probability distribution, then switches to another.

One of the disadvantages of Huffman coding is that it typically uses only single characters - or perhaps character pairs - when creating its code table; this can be somewhat ameliorated by encoding the Huffman-encoded data with another compression scheme such as run-length encoding.

In the example of these algorithms, I'm going to encode them very simply. The LZW code does not adjust dictionary size, and I have not written a decoder, just an encoder. The Huffman example was mostly done by hand.
The text string I used is the beginning of *A Tale of Two Cities*:
"it was the best of times, it was the worst of times; it was the age of wisdom, it was the age of foolishness"

I generated the letter frequencies with Python:

```
>>> for letter in plainString:
    freqDict[letter] = freqDict.get(letter, 0) + 1
>>> freqDict
{'a': 6, ' ': 23, 'b': 1, 'e': 10, 'd': 1, 'g': 2, 'f': 5, 'i':
8, 'h': 5, 'm': 3, ',': 2, 'o': 8, 'l': 1, 's': 12, 'r': 1, 't':
12, 'w': 6, ';': 1, 'n': 1}
```

See the attached images for the tree I created with those frequencies.

The corresponding encoding table, as a Python dictionary:

```
{'a': '0111', ' ': '00', 'b': '011010', 'e': 1011, 'd':
'011011', 'g': 100110, 'f': 10100, 'i': 1101, 'h': 10101, 'm':
'01100', ',': 100111, 'o': 1000, 'n': 1001011, 's': 111, 'r':
1001010, 't': '010', 'w': 1100, ';': 1001001, 'l': 1001000}
```

```
>>> codedStr = ""
>>> for char in shorter:
    codedStr += (str(encDict[char]) + " ")

>>> codedStr[:-1] #cutting off the trailing space
'1101 010 00 1100 0111 111 00 010 10101 1011 00 011010 1011 111
010 00 1000 10100 00 010 1101 01100 1011 111 100111 00 1101 010
00 1100 0111 111 00 010 10101 1011 00 1100 1000 1001010 111 010
00 1000 10100 00 010 1101 01100 1011 111 1001001 00 1101 010 00
1100 0111 111 00 010 10101 1011 00 0111 100110 1011 00 1000
10100 00 1100 1101 111 011011 1000 01100 100111 00 1101 010 00
1100 0111 111 00 010 10101 1011 00 0111 100110 1011 00 1000
10100 00 10100 1000 1000 1001000 1101 111 10101 1001011 1011 111
111'
```

Assuming a 7-bit ASCII encoding, the original message would take up

```
>>> len(plainString) * 7
756 #bits
```

After compression, the length of the encoded message is:

```
>>> len(codedStr)
399 #bits
```

(although this does not include the length of the encoding table)

For the LZW encoding, I wrote two small Python functions which are included with the exam.

Encoded with this LZW implementation, and using a maximum codeword length of 7 bits (enough to encode the test string without having to expand the code table), I got an encoded message of length 497 bits.

This is the final state of the code table:

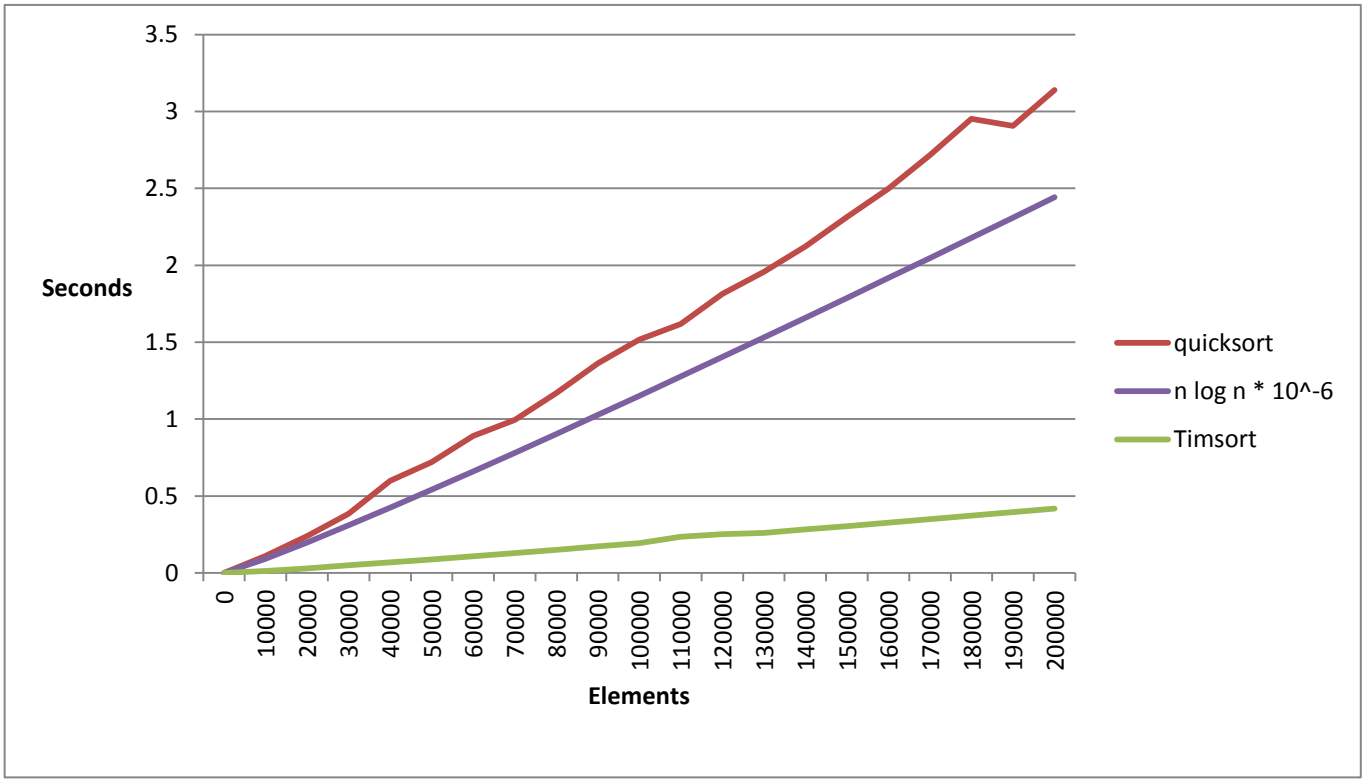
```
>>> table
{' b': '1100010', 't w': '1000101', ' ti': '1011011', ' i':
'1010110', ' o': '1001100', 'ge': '0111111', 'ge ': '0110001', '
w': '1101010', ' t': '1100110', 't o': '1011110', ' ':
'1111110', ' it': '1000110', ',': '1110101', 'li': '0101011',
'th': '1100101', 'e o': '0111110', 'it ': '1010101', 'e a':
'1000001', 'it w': '0110101', 'd': '1111011', ' wa': '1010100',
' wo': '1010000', 'l': '1110011', 't': '1110000', 'h':
'1110111', 't ': '1101011', 'as ': '1010011', ' the': '0110011',
'es': '1100000', 'ess': '0100111', 'rs': '1001110', ';':
'1101110', 'be': '1100001', 'wa': '1101001', 'e ': '1100011',
'wi': '0111100', '; ': '1000111', 'mes': '1001001', 'oo':
'0101101', 'om': '0111000', 'ol': '0101100', 'g': '1111010',
'of': '1011101', 'o': '1110100', 's': '0100110', 'ish':
'0101010', 'w': '1101111', 'es,': '1011000', 'or': '1001111', 's
t': '1000011', 'st ': '1001101', ', i': '0110110', 'm,':
```

```
'0110111', 'e ag': '0110010', 'was': '1000100', 'of w':
'0111101', 'f f': '0101111', 'hn': '0101001', ' tim': '1001010',
' th': '1010010', 'was ': '0110100', 'he': '1100100', 'me':
'1011001', 'b': '1111101', 'f': '1111001', 'n': '1101101', 'r':
'1110001', 'he ': '1010001', 'ag': '1000000', 'do': '0111001', '
of': '0110000', 'is': '0111011', 'it': '1101100', 'f ':
'1011100', 'as': '1101000', 'im': '1011010', 's;': '1001000',
'of ': '1001011', ', ': '1010111', 'ne': '0101000', 's ':
'1100111', 'fo': '0101110', 'a': '1111111', 'e': '1111100', 'i':
'1111000', 'm': '1110110', 'st': '1011111', 'the': '1000010',
'sd': '0111010'}
```

And this is the encoded message:

```
'1111000 1110000 1111110 1101111 1111111 1110010 1111110 1110000
1110111 1111100 1111110 1111101 1111100 1110010 1101011 1110100
1111001 1100110 1111000 1110110 1100000 1110101 1111110 1101100
1101010 1101000 1100110 1100100 1101010 1110100 1110001 1011111
1111110 1011101 1011011 1011001 1110010 1101110 1010110 1101011
1101001 1100111 1100101 1100011 1111111 1111010 1100011 1001011
1101111 1111000 1110010 1111011 1110100 1110110 1010111 1010101
1000100 1010010 1000001 0111111 1001100 1011100 1111001 1110100
1110100 1110011 0111011 1110111 1101101 1100000 0100110'
```

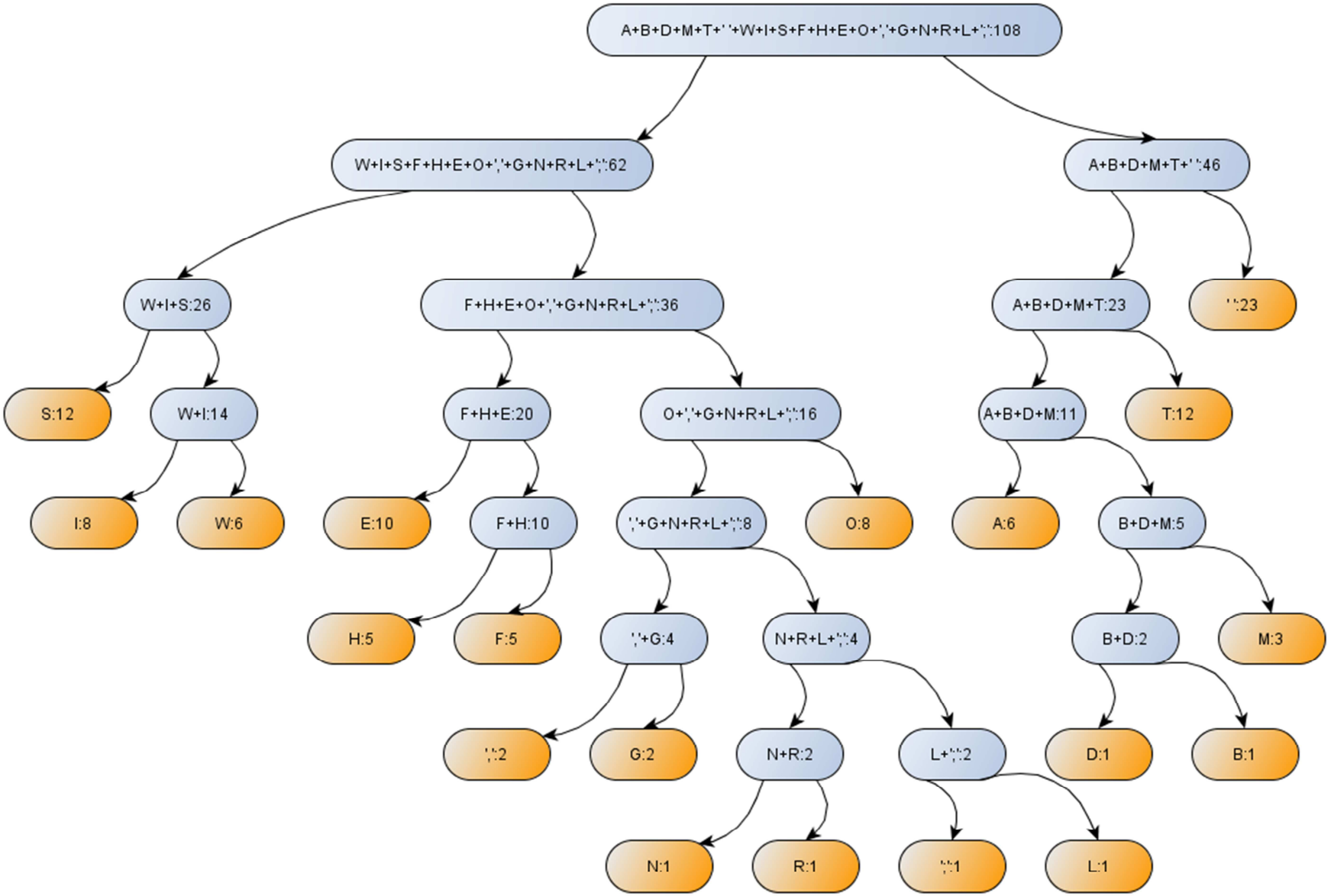
In this example, the Huffman coding produced a shorter message, but as the length of the message increases, LZW should have a greater advantage. Also, it would not be difficult to modify my LZW implementation so that its starting code table was in the same state for every message, thus allowing it to just pass the coded message and save the space the code table would take up.

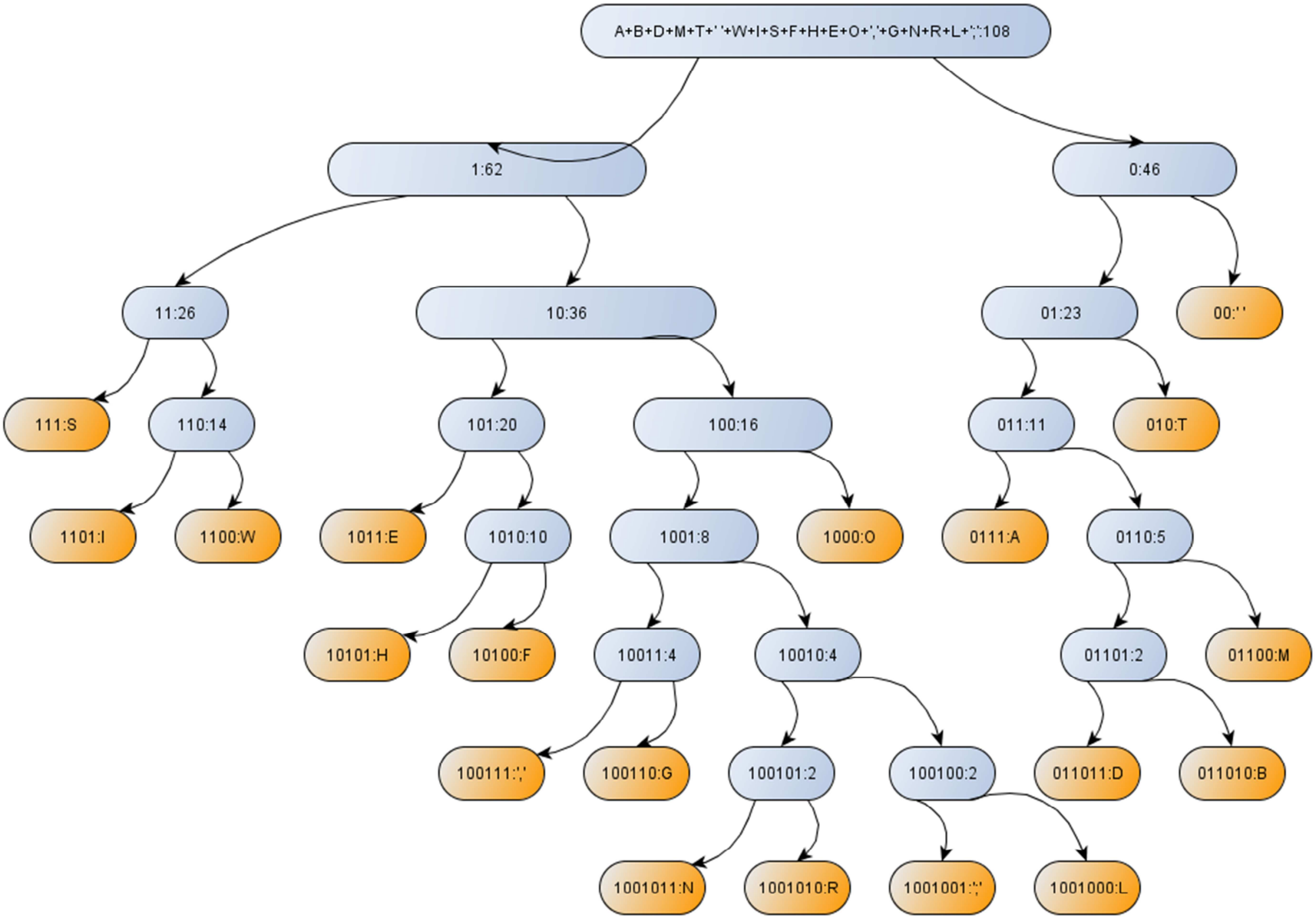


Elements	quicksort	Timsort	n log n * 10^-6
0	0.000001	0.000001	-
10000	0.109463	0.013282	0.092114
20000	0.239633	0.030082	0.198081
30000	0.385177	0.049454	0.30928
40000	0.599323	0.069008	0.423877
50000	0.719258	0.088091	0.541001
60000	0.890108	0.108155	0.660138
70000	0.994673	0.129521	0.78095
80000	1.170019	0.150009	0.903195
90000	1.362924	0.171974	1.026693
100000	1.517209	0.194278	1.151305
110000	1.618466	0.236059	1.276919
120000	1.813367	0.251469	1.403442
130000	1.956763	0.259734	1.5308
140000	2.121195	0.282097	1.658929
150000	2.31349	0.304499	1.787772
160000	2.497842	0.32577	1.917282
170000	2.7176	0.349142	2.047417
180000	2.95106	0.372258	2.178141
190000	2.906974	0.394589	2.309421
200000	3.138662	0.417282	2.441228

```
#!/usr/bin/env python

#quicksort.py

#Algorithms exam
#Problem 1
#Richard Scruggs
#12/3/2010
#Licensed under the Creative Commons Attribution 3.0 Unported License.
#see http://creativecommons.org/licenses/by/3.0/ for license text.

import random
import time

def printTestData(timeList):
    """Prints a formatted table of the data created by testRoutine."""
    print "Results:"
    print "Elements\tTime"
    for timePair in timeList:
        print "%d\t\t%5f" % (timePair[0], timePair[1])

def testRoutine(sorter, minSize, maxSize, interval):
    """Tests the sorting algorithm with random data of a range of sizes. Sorter
    is the function to be tested; the other three arguments are passed to
    range() to generate test data and control the number of tests. Note that one
    is added to maxSize, making the range of test values minSize-maxSize,
    inclusive."""
    timeList = []
    for x in range(minSize,maxSize+1,interval):
        testData = []
        for y in range(10):
            testData.append(randList(x))
        timeList.append([x, timing(sorter, testData)])
    printTestData(timeList)

def timing(f, a):
    """A timing routine, modified from http://www.python.org/doc/essays
/list2str.html
    Expects a function and a list of values for the function. Runs the function
    on each of the list of values and returns the average time taken."""
    t1 = time.clock()
    for data in a:
        f(data)
    t2 = time.clock()
    return round(t2-t1, 5) / len(a)

def quicksort(array):
    """Runs quicksort on the given array, returning the sorted array.

    >>> quicksort([])
    []

    >>> quicksort(range(10))
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    >>> testList = range(10)
    >>> testList.reverse()
    >>> quicksort(testList)
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    >>> testList = randList(10)
    >>> quicksort(testList)
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

```
64          >>> quicksort([4,4,4,5,5,2,2,2])
65          [2, 2, 2, 4, 4, 4, 5, 5]
66          """
67          if len(array) <= 1:
68              return array
69          pivot = array[0]
70          less,greater = [],[]
71          for value in array[1:]:
72              if value < pivot:
73                  less.append(value)
74              else:
75                  greater.append(value)
76          return quicksort(less) + [pivot] + quicksort(greater)
77
78      def randList(n):
79          """Returns a randomly ordered list of integers from 0 to n-1."""
80          randList = range(n)
81          random.shuffle(randList)
82          return randList
83
84      def main():
85          """Runs testRoutine with some default values."""
86          print "Testing Quicksort implementation:"
87          testRoutine(quicksort, 0, 50000, 10000)
88
89      if __name__ == "__main__":
90          import doctest
91          doctest.testmod()
92          main()
```


A+B+D+M+T+' '+W+I+S+F+H+E+O+','+G+N+R+L+';':108
W+I+S+F+H+E+O+','+G+N+R+L+';':62
A+B+D+M+T+' ':46
W+I+S:26
F+H+E+O+','+G+N+R+L+';':36
A+B+D+M+T:23
' ':23
S:12
W+I:14
F+H+E:20
O+','+G+N+R+L+';':16
A+B+D+M:11
T:12
I:8
W:6
E:10
F+H:10
','+G+N+R+L+';':8
O:8
A:6
B+D+M:5
H:5
F:5
','+G:4
N+R+L+';':4
B+D:2
M:3
',':2
G:2
N+R:2
L+';':2
D:1
B:1
N:1
R:1
';':1
L:1


A+B+D+M+T+' '+W+I+S+F+H+E+O+','+G+N+R+L+';':108
1:62
0:46
11:26
10:36
01:23
00:' '
111:S
110:14
101:20
100:16
011:11
010:T
1101:I
1100:W
1011:E
1010:10
1001:8
1000:O
0111:A
0110:5
10101:H
10100:F
10011:4
10010:4
01101:2
01100:M
100111:','
100110:G
100101:2
100100:2
011011:D
011010:B
1001011:N
1001010:R
1001001:';'
1001000:L

```
#!/usr/bin/env python

#blockpuzzle.py

#Algorithms exam
#Problem 2
#Richard Scruggs
#12/3/2010
#Licensed under the Creative Commons Attribution 3.0 Unported License.
#see http://creativecommons.org/licenses/by/3.0/ for license text.

from collections import deque
from random import randint
import copy

class Puzzle():
    """
    A class for Puzzles. This stores the data for a puzzle, its solution, and
    provides various methods to make moves.

    >>> puzzle = Puzzle(3,3,False)
    >>> print puzzle
     1  2  3
     4  5  6
     7  8  0

    """
    def __init__(self, rows, columns, disorder=True, disorderly=30):
        """Initializes a puzzle. Rows and columns determine the size; disorder
        determines whether to randomize the values; disorderly determines how
        many randomizing moves are made."""
        values = range(1, rows*columns)
        values.append(0)
        data,solution = [],[]
        for x in range(rows):
            data.append(values[(x*columns):((x+1)*columns)])
            solution.append(values[(x*columns):((x+1)*columns)])
        self.data = data
        self.rows = rows
        self.columns = columns
        self.solution = solution
        if disorder:
            self.disorder(disorderly)
    def __str__(self):
        """Prints the given puzzle.

        >>> puzzle = Puzzle(3,3,False)
        >>> print puzzle
         1  2  3
         4  5  6
         7  8  0
        """
        outStr = ""
        for row in self.data:
            for column in row:
                outStr += "%2d " % column
            outStr = outStr[:-1]
            outStr += "\n"
        return outStr[:-1]
    def disorder(self, maxDepth, depth=0):
        """Makes maxDepth random moves to disorder the given puzzle."""
        if depth >= maxDepth:
            return True
        moves = deque()
```

```
            addMoves(self, [], moves)
            move = moves.pop()[1]
            selection = randint(0, len(move)-1)
            self.makeMove(move[selection])
            return self.disorder(maxDepth, depth+1)
        def findZero(self):
            """Returns the position of 0 in the given puzzle.

            >>> puzzle = Puzzle(3,3,False)
            >>> puzzle.findZero()
            (2, 2)
            """
            for x in range(self.rows):
                for y in range(self.columns):
                    if self.data[x][y] == 0:
                        return x,y
        def isSolved(self):
            """
            Returns True if the puzzle is solved, False otherwise.
            >>> puzzle = Puzzle(3,3, True, 1)
            >>> puzzle.isSolved()
            False
            >>> puzzle.data = puzzle.solution
            >>> puzzle.isSolved()
            True
            """
            return self.solution == self.data
        def makeMove(self, move):
            """Makes the given move.
            A move is formatted [[sourceRow, sourceColumn], [destRow, destColumn]]
            (although the ordering of source and dest are irrelevant)
            >>> puzzle = Puzzle(3,3,False)
            >>> puzzle.makeMove([[2,2],[2,1]])
            >>> print puzzle
             1  2  3
             4  5  6
             7  0  8
            """
            source = move[0]
            dest = move[1]
            self.data[dest[0]][dest[1]], self.data[source[0]][source[1]] = \
            self.data[source[0]][source[1]], self.data[dest[0]][dest[1]]
        def makeMoves(self, moves):
            """Makes each of the given list of moves."""
            for move in moves:
                self.makeMove(move)
        def unmakeMove(self, move):
            """Unmakes the given move. That is actually the same as making it, so
            this is just syntactic sugar."""
            self.makeMove(move)
        def unmakeMoves(self, moves):
            """Unmakes each of the given list of moves. Note that makeMoves and
            unmakeMoves are expected to operate on the same set of moves, which is
            why this function performs the moves in reverse."""
            tempMoves = moves[:]
            tempMoves.reverse()
            for move in tempMoves:
                self.unmakeMove(move)

    def addMoves(puzzle, path, moves):
        """Expects a puzzle, a path - which is the list of moves that it took to get
        from the start of the puzzle to the current position - and a queue of moves.
        Adds the moves that are possible from the current position to the move
        queue. Does not return anything.

        The exact format of moves is a bit confusing. The queue is a list of
```

```
        movelists; each movelist is a list of two things: the path, and the possible
        moves. The path is explained above; the possible moves are all moves that
        are valid from the position specified by the path.
        To try to explain it better:
        moves=[path, possibles].
        path = [firstmove, secondmove, thirdmove]
        possibles = [possibleA, possibleB, possibleC]
        firstmove = [[0,0],[1,0]], for example
        possibleA = [[]]

        >>> puzzle = Puzzle(3,3,False)
        >>> moves = deque()
        >>> addMoves(puzzle, [], moves)
        >>> moves
        deque([[[], [[[1, 2], [2, 2]], [[2, 1], [2, 2]]]]])
        """
        zeroRow, zeroColumn = puzzle.findZero()
        dest = [zeroRow, zeroColumn]
        tempMoves = []
        if zeroRow+1 < puzzle.rows:
            tempMoves.append([[zeroRow+1, zeroColumn], dest])
        if zeroRow-1 > 0:
            tempMoves.append([[zeroRow-1, zeroColumn], dest])
        if zeroColumn+1 < puzzle.rows:
            tempMoves.append([[zeroRow, zeroColumn+1], dest])
        if zeroColumn-1 >= 0:
            tempMoves.append([[zeroRow, zeroColumn-1], dest])
        moves.append([path, tempMoves])

    def solve(puzzle, depthFirst, maxMoves=1000):
        """Solves the puzzle. Returns True if the puzzle has been solved, False
        if it has made maxMoves moves without solving it. If depthFirst is 1, it
        solves with depth-first; if 0, it solves with breadth-first.

        This solve function actually handles both depth-first and breadth-first
        searches - the only difference is how it accesses the queue of moves. The
        depth-first accesses the queue as a stack, the breadth-first accesses it as
        a queue.
        """
        if puzzle.isSolved():
            return True
        moves = deque()
        addMoves(puzzle, [], moves)
        moveCounter = 0
        if depthFirst:
            getMove = moves.pop
        else:
            getMove = moves.popleft
        while moveCounter < maxMoves:
            nextMove = getMove()
            moveCounter += 1
            puzzle.makeMoves(nextMove[0])
            for move in nextMove[1]:
                puzzle.makeMove(move)
                if puzzle.isSolved():
                    return nextMove[0] + [move]
                addMoves(puzzle, nextMove[0]+[move], moves)
                puzzle.unmakeMove(move)
            puzzle.unmakeMoves(nextMove[0])
        return False

    def main():
        """Generates a random 8-puzzle and tries to solve it."""
        puzzle = Puzzle(3, 3)
        print puzzle
        puzzle2 = copy.deepcopy(puzzle)
```

```
197         print "Solving with breadth-first."
198         solution = solve(puzzle, False, 500)
199         if solution:
200             if isinstance(solution, list):
201                 moves = len(solution)
202                 print "Puzzle solved in %d moves." % moves
203             else:
204                 print "Puzzle started solved."
205         else:
206             print "Puzzle not solved - too many moves tried."
207         print "Solving with depth-first."
208         solution = solve(puzzle2, True, 500)
209         if solution:
210             if isinstance(solution, list):
211                 moves = len(solution)
212                 print "Puzzle solved in %d moves." % moves
213             else:
214                 print "Puzzle started solved."
215         else:
216             print "Puzzle not solved - too many moves tried."
217 
218     if __name__ == "__main__":
219         import doctest
220         doctest.testmod()
221         main()
```

```
#!/usr/bin/env python

#lzw.py

#Algorithms exam
#Problem 4
#Richard Scruggs
#12/3/2010
#Licensed under the Creative Commons Attribution 3.0 Unported License.
#see http://creativecommons.org/licenses/by/3.0/ for license text.

#The code for this problem is less standalone and complete than the code for the
#other problems; I just wrote it to ease the parts of the problem that would be
#repetitive or difficult to do by hand.

def buildCodes(length):
    """Returns string binary representations of the numbers from 0 to
    (2**length)-1. Zero-pads those representations.
    >>> buildCodes(3)
    ['000', '001', '010', '011', '100', '101', '110', '111']"""
    codes = []
    for x in range(2**length,2**(length+1)):
        codes.append(str(bin(x))[3:])
    return codes

def lzw(message, codes):
    """LZW encodes a message. Expects a message and a list of codes as generated
    by buildCodes. If the list of codes is too short for the encoder to create
    a code table for the entire message, this crashes."""
    messageCopy = message[:] #because I destroy messageCopy in encoding
    encTable = {}
    for character in set(message):
        encTable[character] = codes.pop()
    #Now we have a starting encTable - the single characters in the message all
    #have encodings.
    encMessage = ""
    #I perform the actual encoding thus: I pull off the first character in the
    #message as currentEnc then, while the currentEnc + the next character is in
    #the encoding table, add the next character to the encoding string.
    #After the while loop, currentEnc is as long as it can be while still being
    #in the table, so I add a table definition for currentEnc + next character
    #and add the code for currentEnc into the encoded message.
    while messageCopy:
        currentEnc = messageCopy[0]
        while messageCopy[:len(currentEnc)+1] in encTable.keys() and len(currentEnc) < len(messageCopy):
            currentEnc = messageCopy[:len(currentEnc)+1]
        encTable[messageCopy[:len(currentEnc)+1]] = codes.pop()
        encMessage += (encTable[currentEnc])
        messageCopy = messageCopy[len(currentEnc):]
    return encTable,encMessage
```