# Find the Sneaky Path

CS404 Project, Fall Semester 2016
Version: October 31, 2016.
Due: Saturday November 26, 2016 (or earlier)

**Summary**

Finding an optimal path in a network is not always the shortest path, shortest distance, or cheapest cost, but rather: most scenic, least traffic, get-away path, and so on. This project explores finding a *SneakyPath*.

**Motivation**

Commercial Airlines, roads, and computer and communication networks often model their routes as graphs. There are other examples in social networks and cloud computing, and so on. In such a network, nodes are the cities, or intersections, or server nodes (sources of demand, such as traffic or communication sites), and the links connect hem. If you want to go from one node to another (by plane, by car, or routing internet traffic or information packet), then you need a plan: source, destination, and a planned path in between. Let us explain this in terms of an road network between cities as the motivating application, but keep in mind that this project might have additional uses. When planning a car-trip, say from $a$ to $b$, most people would want to find the shortest route, and there are several algorithms known that will help you find the shortest path. But this may not always be the criterion used. Some people may be interested in the least travelled road, to avoid as much congestion as possible. They want to avoid other cars as much as possible, they do not want to see other cars, they do not other cars to see them, or whatever. May be you are towing a 12-foot wide boat, and you do not want to inconvenience other drivers, so you look for the least travelled road. Even in computer networks this might be of interest: lowest probability of dropped packets. We will call such a path a *SneakyPath*.

You will be given a number of different input scenario's. Each scenario will have

1. The size $n$ of the system (the total number of cities),
2. an adjacency matrix $\mathbf{E}$ with edge-weights. The $(i,j)^{\text{th}}$ entry $\mathbf{E}[i,j]$ represents the traveling time on the (direct) edge between the two specific nodes, $i$ and $j$.

The matrix $\mathbf{E}$ is will allow you to compute the shortest path from $a$ to $b$, and several algorithms could be used for this. The shortest path might take several edges (or links), and if you are the only one on the road, then this is the path you will take. However, you are not the only person/car on the road. You will also be given

3. a flow matrix $\mathbf{F}$, whose $(i,j)^{\text{th}}$ entry $\mathbf{F}[i,j]$ represents the number of (other) cars that travel the entire path from node $i$ to node $j$ every hour.

For the sake of argument, say that there are $\mathbf{F}[21,47] = 115$ cars traveling on the path from node $v_{21}$ to node $v_{47}$, then there are $115$ cars on each edge of that path. Suppose edge $(v_{34}, v_{38})$ is on the shortest path from node $v_{21}$ to node $v_{47}$, then there are also $115$ cars/hour on the edge $(v_{34}, v_{38})$ due to the flow $\mathbf{F}[21,47]$. But is very well possible that the edge $(v_{34}, v_{38})$ is also on the shortest path from node $v_{17}$ to node $v_{29}$, and that $\mathbf{F}[17,29] = 75$. This means that there are at least $190$ cars/hour on the edge $(v_{34}, v_{38})$. Thus for each edge $(i,j)$, you can calculate the actual carried traffic load from all source-destination pairs on that edge, because everybody takes their shortest path. Put all this information in a new matrix $\mathbf{L}$, each entry representing the total load on the edge. Finally, you will be given

4. the starting node $a$ and the terminal node $b$

And for this particular pair, you are asked to calculate (and print):

a) The *SneakyPath* from $a$ to $b$, such that the total number of other cars on the road encountered is as small as possible,

b) the edge on this *SneakyPath* with the lowest number of other cars,

c) the edge on this *SneakyPath* with the highest number of other cars,

d) the average number of other cars on the *SneakyPath*, averaged over the number of links on the path.

This is however a course on algorithms, and you also have to report on the algorithms you used to actually find the information asked above. In particular,

e) The worst case time complexity for the algorithm(s) you used,

f) the measured CPU-usage of the programs when it solved each case,

g) the empirical validation that your programs have the correct asymptotic complexity.

## Project expectations

You need to explore and expand the algorithms for finding the information as outlined above, and you should feel free to explore additional algorithms that you feel are appropriate. The goal is to find an efficient algorithm for the problem and to implement them efficiently. You can write separate algorithms/programs for the separate parts, or use the same one, whichever is most appropriate for the problem. You need to write a report in which you address the following issues:

1. Explain the reasoning for using the algorithms you implemented (as opposed to alternate choices), explain any supporting algorithms and data structures. Obvious potential choices are Dijkstra's SPA, and Floyd-Marshall SPA.
2. Explain the algorithm you used and implemented to actually generate the paths between nodes, so that the load per edge can be calculated. If there were two ideas you were working on, what are the pro's and con's? You may use secondary metrics, such as ease of programming, ease of software maintenance, portability, and so on.
3. Provide convincing arguments that your algorithms find a correct *SneakyPath* or demonstrate by counter example that it does not always find a *SneakyPath* and explain why this should be acceptable.
4. Present an analysis of your algorithms for best-case and worst-case time complexities as well as space complexities. Make sure to mention the key-and-basic operation you are using.
5. Implement the algorithms that is best suited (or acceptably suited) for networks where $n$ is bounded above by $1,000$. Best-suited means better performance.
6. Provide convincing arguments that your have implemented your algorithms correctly and efficiently with the appropriate data structure.
7. Perform a timing study of your implementation and show that this conforms with your (pre-implementation) time complexity study. In other words, measure the run times of your programs to experimentally verify your analytical findings. The least you can do is a plot (size $n$ versus observed times).
8. For the cases that will be provided, present the *SneakyPath*.
9. Feel free to add test cases yourself that illustrate certain algorithmic or program features
10. Indicate whether or not you believe that there is a better (i.e. faster run-time) solution with an explanation on why you believe this to be the case or not to be the case.
11. As a last item, reflect back on this project and on the design and data structure decisions you made. How did you handle any unforeseen situations and any 'newly discovered' problems? Now that you have done the project, what have you learned? If you have the opportunity to do this project again, perhaps with additional requirements, what would you do the same, and what would you do differently? What are the lasting lessons you have learned? What are the lessons for future students?

### Input

Some time next week, I will make available one or more input data files that contain $n$, $a$, $b$, $\mathbf{E}$ and $\mathbf{F}$. The format will be the three numbers: $n$, $a$ and $b$, after which you get a number of entries of '$E, i, j, E[i, j]$' or '$F, i, j, F[i, j]$'. Several more input files will follow a few days before the due date, too late for a re-design of the project.

### Strong Advice

CS404 is a course on algorithm design, and I would expect that this would be the most time consuming task. Particularly, because you are only tasked with 'What' needs to be accomplished, and you yourself have to decide on the 'How'. This part is the most rewarding phase, yet also the most frustrating phase of the project, since it is impossible to predict how long this takes for anyone, and it is impossible to realize how close you may be to a solution. Just do not panic, and go back to the basics: What do you currently have, where do you need to go, and how can you use what you learned in previous courses to your advantage. Structure your thinking, document what you could try and have tried already. Once you get the big idea, the programming is less challenging due to your strong programming and coding skills, but the interpretation of the results and the actual writing of the report will take some time again. So set yourself realistic milestones, and build in extra time to recover from setbacks. The key is: Get started. Start thinking.

### Project Report

When all is said and done, we expect a project report where you have the opportunity to fully present your findings and conclusions, together with your algorithms, data structures and programs, and where you are asked to highlight any novelty of design/implementation. This report gives you the opportunity to create "talking points" when you are going for an interview, and something tangible to show. See the last page for details on the report.

### Getting Help

The primary point of contact for this project is Priya. Her office hours are Wednesdays from 2pm – 4pm and Fridays from 12noon – 2pm in room FH 560H. You can also make an appointment with her by sending an email: pn535@mail.umkc.edu

### Friendly reminder

The letter grade for the entire course will be based highest of the following:

- If your project grade is below $60\%$, then your (letter) grade for the course will be an $F$.
- If your project grade is larger than $60\%$, then your course percentage is calculated as indicated on the syllabus.

# Generic Expectation for Projects in CS404

The description below is written for all projects assigned in CS404, regardless of the semester.
If anything conflicts, then the specific requirements for the current semester take priority over these generic ones.
Version: Fall Semester, 2016.

### Generic Goals and Expected Outcomes

Certain components of this project are used specifically to assess your attainment toward meeting the expected student learning outcomes for our degree programs as required by both our outside accreditation agencies and our internal assessment committees. Some of these learning outcomes are taught in this course, whereas others have been taught in prior courses, and you will be able to demonstrate your mastery of them. For instance, you have learned various programming skills in prior course work, and this project gives you an opportunity to demonstrate your skills, or even expand and learn new features. I expect that you can do so with self-study, or with the help of the TA. This course is taught independent of a particular computer language.

Generically, projects in CS404 are designed around the expected course outcomes. Keep in mind, that a particular project may not be suited to demonstrate all outcomes. Some outcomes will be tested during quizzes and exams.

- Demonstrate the ability to evaluate trade-offs in design choices Your report should indicate at least two algorithmic choices that you considered, before settling on the one you implemented.
- Demonstrate the ability to select the proper data structure to solve a problem Your report should indicate at least two data structure choices you considered for parts of the program, before settling on the one you implemented.
- Demonstrate the ability to use probability to analyze design options
- Demonstrate the ability to design an efficient algorithm What are the best- and worst-case time complexities for your algorithms?
- Demonstrate the ability to make ethical decisions Ethical decisions are expected every day from every professional in our field. You will be asked to submit a statement concerning your adherence to student conduct.
- Demonstrate the ability to gather any additional information that may be needed Consult other sources and, perhaps, discuss with friends and class mates. Please cite all sources appropriately. For this project, cite any and all materials that you found useful in this work.
- Demonstrate the ability to communicate in writing Your report is expected to be complete, correct from a mathematical and computing perspective, and clear in presentation, with correct English grammar and usage. It also means that any programs you write in support of your findings are written according to standard software engineering practice: well designed solutions, readable and self-documenting code, header files, style, and appropriate use of language and library features. Note that any choice you make is important and subject to evaluation. Please indicate and elaborate on any choice(s) you have made that contributed significantly to the success of the project, and that you are a little bit proud of.

### Generic Algorithm and Program Design

The teaching philosophy behind project based learning is that the requirements for such a project are somewhat open-ended: You are given a problem to solve, without a strict guidance on 'how' to solve it. As such, there is room to display your particular problem solving skills, your creativity, and your thoroughness. There are still a fair number of decisions to be made, and you should make judicious choices. Please document the reasons why you made your decisions. Present the issue or option, discuss the pros and cons of each option, and present the reasoning behind your decision.

Following the design of the algorithm comes the choice of which programming language you should select to implement it. You should chose the language based on the inherent capabilities of the language and your own familiarity with it. You are welcome to use the programming language of your choice for this project, as long as the teaching staff (teaching assistant and/or the teacher) can run it on software available in our (Virtual) Labs, and as long as you are willing to explain and demonstrate to us (and teach us) your specific language. The following languages are pre-approved and recommended for a project like this: C, C++, Maple, Matlab, and Python. All other languages need to be pre-approved by explaining (in writing) why you prefer the proposed language above the suggested ones. Remember, the algorithm needs to be designed first, before looking for a language. Whichever language you choose, you should take advantage of the subroutines that are available in the supporting Standard Libraries. Proficiency in programming is a prerequisite for the course, although there are opportunities to hone or expand your specific programming skill set.

Regardless of the programming language you use, I expect you to turn in "well structured" and "well documented code" according to the standards you have been taught in previous classes and according to the best practices for the language you have chosen. Important considerations for efficient and high quality programs are structure and simplicity. Efficient programs are usually not the most straightforward, and this must be explained and (self-) documented in the program itself. Neatness counts as well. If you want others to use your program (or if you want to re-use and modify your own program in a few months), they must be easy to understand, well-organized, coherent. Application programs are frequently modified by yourself or others. Future users (and your managers and instructors) must be able to read and understand your programs easily. A well-written program reflects a well-conceived design, and such a program is almost always simpler and more efficient.

I expect that you accept responsibility for the programming proficiencies that were covered in previous courses and I expect you to perform at the level of either 'Acceptable" or 'Exemplary' in the rubric below. In particular, all programs should compile, should be without runtime errors, and provide correct output. Please do not expect much partial credit if your program falls into the 'Should be better', regardless how much time it has taken you.

| Trait | Exemplary | Acceptable | Should be better |
|---|---|---|---|
| Specifications | The program works and meets all of the specifications. | The program works and produces the correct results and displays them correctly. It also meets most of the other specifications. | The program produces correct results but does not display them correctly or produces incorrect results. |
| Readability | The code is exceptionally well organized and very easy to follow. | The code is fairly easy to read. | The code is readable only by someone who knows what it is supposed to be doing, or is poorly organized and very difficult to read. |

**Generic Correctness and TestCases**

For all algorithms, provide convincing arguments as to why they are correct. Similarly, for all programs and routines, provide convincing arguments and test cases that your implementation is correct. Correctness here means two things: the output is correct, AND the performance is correct. For the output to be correct, you need to develop your own test cases, much like "Test Driven Software Development". You do not need to show the actual input-output map for your test cases, but you must report which test cases you generated, verified, and validated. From your previous course work, you have learned how to generate test cases to test for correctness. We are adding to this list and expect you to test for both correct implementation and correct according to predicted runtime performance. You do not need to report on these explicitly in your report but you must report that you have performed them.

- Test for arguments in the correct order,
- Test for arguments within the accepted range, on the borderline, and out-of-range. Note that an argument may have

multiple fields (or other composites, like a linear list, tree, etc). Again, test for in, on, and out of range fields (or test for empty composite, trivial composite, very large composite). Note that an argument may be of variant type, test for concrete subtypes.

- Test for the correct number of arguments (fewer, equal, and larger). Note that an argument may have multiple fields.
- Make a deliberate choice of input arguments so that all execution paths are exercised. In additional to a choice for which no execution path is/was specified.
- Make a deliberate choice of input arguments so that all output options or actions are exercised in addition to a choice for which no output/action is/was specified.
- If correct answers are not always unique, test for several of these.
- If the output is a number of objects, each perhaps with multiple fields, or of variant types, then test these results in the calling program as indicated above.
- Use the above ideas to suggest additional test cases based on problem specific, algorithm specific, data structure specific, and language specific features.
- Scrutinize for (and experiment with) implementation choices that do (or might) impact timing performance, such as 'passing-by-value', when 'passing-by-reference' is warranted, making local copies of some arguments to improve page-density, and others.
- Add counters to your implementation that count the number of times that "key-and-basic" operations are executed.
- Add SVC-calls to your implementation to track the actual CPU-time charged to your implementation. This depends very much on your platform, configuration, cache-size, RAM-memory size, virtual space, and-so-on. Each language and each platform provides such a development and management tool (it is called CPTime in C++ environment).

## Deliverables – Fall Semester 2016

*Aim to write a report that you could proudly incorporate in a portfolio of accomplishments that potential future employers might ask for.*

I expect a type-written report with the sections below. Your report should have (and will be graded on) the appropriate substance and rigor with the results, and be well structured in terms of grammar, syntax, and style. The overall presentation and clarity of the report is important, so please mark your sections clearly, and put them in the correct order; it also makes grading easier. You should use the listed 'Generic Goals and Expected Outcomes' above, and the 'project report deliverables' below, as a guide for naming your (sub-)sections. Describe the steps you took to convince yourself that the algorithms are correct. Describe the steps you took to convince yourself that your programs are a correct implementation of your algorithms (i.e. test cases).

Feel free to use outside sources and material that are free and publicly available; just make sure to cite them appropriately. All figures should have captions and clearly labeled axes. Equations are preferably typeset with a proper equation editor. The report includes a *cover page* that identifies the project and your name, and a declaration that you have adhered to the policy on academic honesty that I am asking you to sign. The report must also include a page (if appropriate and most likely as an appendix) that serves as a README file for your programs, where you identify the computer language you used and the compiler and other environmental information that we need to know to run your programs. This is also the place to let us know the status of the programs in your project. If it is not quite fully complete. (For example, I was not able to run it on your test-data, but it worked fine on my own test data with $5$ elements. Or: there is an unexplained bug for file sizes larger than $n = 100$.) Please note, that a bug in an algorithm is more serious than a bug in a program. An undocumented bug in a program is more serious than a documented bug.

Your Project Report should have the following identifiable sections:

**Title page**

CS404, Fall Semester 2016: Find the Sneaky Path

> I understand and have adhered to the rules regarding student conduct. In particular, any and all material, including algorithms and programs, have been produced and written by myself. Any outside sources that I have consulted are free, publicly available, and have been appropriately cited. I understand that a violation of the code of conduct will result in a zero (0) for this assignment, and that the situation will be discussed and forwarded to the Academic Dean of the School for any follow up action. It could result in being expelled from the university.
>
> //Your Name// //This Date//

Please print and sign this page. Subsequently scan it back in and attach it to your report.

**Introduction** (5%) Section in which you state the problem in your own words or paraphrased from this description. This makes the report self contained so that you (and others, like employers) can read the rest with ease, even next semester/year.

**Design and Analysis of the algorithm(s)** (25%) A section in which you present your algorithms with supporting data structures. Provide convincing arguments that your algorithms finds the correct shortest paths, finds the correct loading on each link, and finds the correct SneakyPath for the given $a$ and $b$. This includes an analysis in terms as

best-case and worst-case time complexities as well as space complexities. Make sure to mention the key-and-basic operation you are using. The analysis is expected to have appropriate rigor and communicated clearly, so that a student who is currently in CS303 is able to follow your derivation. In your narrative, explain any design and implementation decisions you made and why you believe these were good choices. In particular, highlight any special or innovative features and focus on the performance impact of your decisions. Describe any modifications that you had to make to any standard algorithm as presented in the textbook, in class, or those you found elsewhere (give proper citations). Highlight any key insight or neat design feature that is particularly helpful for efficiency in this project, and those that were particularly enlightening to you.

**Implementation and testing of your algorithm** (25%) First, you need to tell us what language you use to implement your algorithms, and why you have decided on this particular language. Also, in this section you provide evidence that your program is running and gives correct results, probably for small test cases you create yourself by hand. If you needed to make significant changes to the original algorithm or original program, then you report these in this section, and you report which steps you took to convince yourself (as well as the *TA* and *me*) why your program is a correct implementation of the algorithm, and report on the various test data you created. We expect the code to be transparent and 'mean and lean'. A listing of all your programs (both .h and .cpp files) are in an appendix. Elaborate when appropriate. For instance: if there are two closely related data structures, then how did you decide between them? How and why did you break ties? Why did you rely on routines from the STL with unknown performance, rather than writing your own and understand it's performance? Highlight any key insight or neat design or implementation feature that is particularly helpful for the efficiency in this project, and those that were particularly enlightening to you.

**Validate your pre-implementation analysis with a timing study of your program** (15%) Perform a timing study of your implementation and show that this conforms with your (pre-implementation) time complexity study. In other words, measure the run times of your programs to experimentally verify your analytical findings. This is preferably done in one or more tables, where you present, for each input, the value of $n$, the predicted time complexity of your algorithm and the measured time consumption of your programs. Now is the time for the analysis and interpretation: Are these timings consistent with the predicted times? Can you daw any conclusions from these? If not, you may need to generate additional test cases. Can you draw conclusions if you also include the smaller test cases? Based upon your work (you may need to do additional runs): how long would your algorithm/program run if you double the input size? You may need to copy this information from the output as generated by your program. Include in an appendix the (unedited) outputs as generated by your program(s) for the required test cases that will be provided.

**Epilogue** (10%) In this section, you reflect back on the design and data structure decisions you made. How did you handle any unforeseen situations and any 'newly discovered' problems? Now that you have done the project, what have you learned? If you have the opportunity to do this project again, perhaps with additional requirements, what would you do the same, and what would you do differently? What are the lasting lessons you have learned? What are the lessons for future students?

**Written communication skills** (10%) Organization, readability, structure, style and presentation of the report. Use correct English (grammar, syntax, spelling, and so on) and use section headings that will make it easy for us to find the outcomes we are looking for.

**Appendix A: Program Listing** (5%) Here provide a 'Program Listing'. Include both .h and .cpp files. Programs should be documented according the standards from CS201/CS201L/CS303. State which compiler and which machine was used to produce the results. We expect your program to be readable and documented internally: identification block; Internal documentation, proper indentation, meaningful variables, ...

**Appendix B: Output** Provide outputs exactly as generated by the programs you include in appendix A.

**How to Submit** Your complete package (report and programs) must be submitted in a single zip-file through Blackboard. Should this not work, then emailing an zip-file is okay as well. Your file should be called `SneakyFS16LLLLFFFF`, where you substitute the first four letters of your last name for `LLLL`, and similarly the first four of your first name for `FFFF`. The primary point of contact for this project is the TA, and you should direct all questions to the TA, whose office and hours are announced in a separate cover through Blackboard, and whose email address was listed in case you need to make an appointment outside the office hours. If you contact the TA, you could be asked to explain your design and/or implementation in person, during which you should demonstrate the workings of your programs and your understanding of the project and its implementation. Your programs must run (and be correct) for newly supplied data. The project grade will depend on what you submit.

**Late Submission** The late policy on project is 10% off the total grade if late within one day, 20% off the grade for two days late, 30% off the grade for three days late. Projects that are submitted more than three days late will no longer be graded.

**Sample input**

Will follow shortly on Blackboard