

## **Uninformed Search Homework**

T. Nathan Mundhenk and Laurent Itti  
University of Southern California  
Department of Computer Science

Copyright (c) 2006 T. Nathan Mundhenk, Laurent Itti

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

**Due: By class Tuesday, January 31, 2006**

**Guidelines:**

This assignment has a written and programming component. You can see how much each part of the assignment is worth by the percentage next to it. For the written part, please turn in typewritten answers. You are not English majors, however blatant spelling and grammatical errors may cost you points, so be sure and run the spell check at least once. Answers on the written part will require justification and a simple yes/no answer will almost certainly garner you little or no points. In general, it is safer to write longer answers than shorter ones, but stay focused as a long but off-topic answer will not work either. This way, we can discern your train of thoughts better and grant partial credit for a wrong answer, if you were on the right track. Graphs must be neat and tidy. Hand-drawn graphs are OK, but computer-drawn graphs (e.g., made with Adobe Illustrator or Microsoft Power Point) are preferred. If you draw a graph by hand, be sure and use a straight edge (ruler) so it looks neat.

For the programming part, you will be provided sample inputs and at least one sample output. Since each homework is checked via an automated Perl script, your output should match the example format *exactly*. Failure to do so will most certainly cost some points. Since the output format is simple and there is an example on the web, this should not be a problem. Additionally, if your code generates a large number of warnings during compilation, you may lose points, so try and eliminate compile-time warnings. Additionally, your code should be well documented. If something goes wrong during compile and grading, if the fix proves easy, the amount of points lost will be far less. As such, documentation makes fixing easier, so it is to your advantage to do so.

You will be provided with a stub Perl script called “stubby.pl”, which works like the grading Perl script. You can run this script to make sure that your project will work properly. Thus, it is expected that your project will run through the grading Perl script without problems. Pedantically, we assert it will cost you points if your code does not run on the Perl script correctly. You will be able to tell if your output is correct if stubby shows each of the lines from your program output is exactly the same as from the comparison file which shows what output you should be getting.

To run stubby, copy it to your code directory along with the input, output and comparison files. Be sure to be in your code directory then type “perl stubby.pl”. The script is loaded with all sorts of output and feedback, so you should be able to see what it is doing if for some reason it isn't working for you. If you don't understand the script and want to know more about Perl, go to <http://www.perl.org/>.

Small amounts of extra credit (not more than 10% total) are given for all sorts of things. So long as you meet the base homework requirements anything creative, fun, interesting or outright cool will most likely earn you extra credit. What is cool and earns you extra credit is somewhat subjective and bound to the whim of the grader.

### **Other information:**

For this assignment and many others you will be required to parse input and output files. To help you out, a file called “proj1.cpp” is in the homework 1 folder on den. This is an example file on how to open, read in and then dump a file back out to disk. You may use parts of this code for your homework if you wish.

To compile this file, try something like:

```
“g++ proj1.cpp -o proj1”
```

Then to run it, try something like:

```
“./proj1 matrix2.mat out2.mat”
```

This will read in the matrix file “matrix2.mat” and dump it back out to another file “out2.mat”.

### **Handing in the Assignment:**

Since the class is very large it is important to hand in your homework as directed. *The homework is due before class on the due date shown.* There are two parts of the homework, which you must hand in. These are:

- (A) Your code tar/gzipped in **one** file. Do not send the binaries. - This should just include your uncompiled code and a readme file called readme.txt. When I run tar/gzip to uncompress your file, it should uncompress into its own directory. To keep things uniform, do not use bzip2. I should then be able to run my stubby Perl script and grade your code. See below on how to tar/gzip your code.
- (B) Your written part on paper. *Iff* you are a DEN student, you may email your written part to cs561. Otherwise, you must submit a paper copy either in class or in a drop-box in front of my office in HNB (NOTE: the HNB building closes at 5:00pm). Late submissions will be noted. Be sure your homework is stapled together and is generally tidy. Electronic submissions of the written part from non-DEN students will not be accepted.

Be sure that your name is *clearly visible* on all material you hand in. To hand in your code you can compress it with tar/gzip with a command like:

```
“tar -czvf my.name.hw1.code.tgz my.name.hw1.code”
```

You might also try:

```
“tar -cvf my.name.hw1.code.tar my.name.hw1.code” then type “gzip my.name.hw1.code.tar”
```

This will compress the contents of the directory named “my.name.hw1.code” into a single file my.name.hw1.code.tgz. If you need more info on this, try “man tar”.

### **Electronic Submission of code:**

You need to submit the assignment electronically using the EXACT following command FROM YOUR ALUDRA ACCOUNT.

```
submit -user csci561 -tag hw1 my.name.hw1.code.tgz
```

Be sure and substitute “my.name” with your own name. The submit command will immediately respond with a SUCCEEDED if your submission of file "my.name.hw1.code.tgz" is successful. That will be your means to know that your homework has reached the right place. Your submissions will be time stamped, so we will know the exact time when you made the submission.

### **DEN students using electronic submission of the written part:**

For DEN students handing in the written part electronically, if you use Word files, please compress the file using .zip or .rar.. Don't worry about compressing Acrobat documents since they are already compressed. Thus, I should see two files if you are doing an all-electronic submission, one with a name like bob.bobbinopolis.hw1.written.zip and the other with a name like bob.bobbinopolis.hw1.code.tgz. If you send me a file with a name like hw1.zip, I will stick three pins in your voodoo doll. Of course this does not apply to your .cpp/.h files which should have names that work with the stubby Perl script. Also, be sure your name is on the documents you hand in since we print them up to grade them and it can be hard keeping documents matched. Please make sure all images and material for the written part are collated into one single file. Thus, *do not* send a word document and 30 image files, be sure and place them inside the word document itself.

### **What exactly is Stubby?**

*This is very important.* We use a Perl script to automate the grading process. Stubby is a Perl script we give to you so that you can check and make sure that your project conforms to specifications. That is, you can use Stubby to make sure that your assignment when handed in will run with our grading script. Thus, we have our own grading script *like* stubby, which we use to grade your project. By checking to make sure your project works with stubby, you make sure that your project will work with our automated grading script.

It is important to note that we will not use your version of stubby. We have our own script. As such, if you edit stubby to work with your code and not the other way around, this will not be very helpful. Additionally, since there are so many projects to grade, it is imperative that your program work with the grading Perl script. *If your program does not work with the grading Perl script you will lose points.*

**Problem 1 (20%):** Gung Ho Joe is a video game warrior. He is slashing his way through vile monsters when he comes upon the gratuitously tentacled Purple People Eater. He must act fast and dispatch the hideous foe before it splashes radioactive bacteria on him. There are a variety of items Joe can use to destroy this beast.

- Joe has an M-16 with a grenade launcher and two types of bullet ammunition, Teflon® tipped and explosive tipped.
- Joe is wearing a coat, which has inside it a very large .45 caliber gun, a rubber mallet and two Lakers tickets.
- Joe has slung over his shoulder some really cool particle cannon and a weird alien hand that as far as we can tell, shoots bees.
- Joe has a sack, which contains a hand grenade, a two-way radio, the galactic transponder and a small change purse, which has a nickel, two dimes and pocket lint inside.

Assume that any of these items can be used against the Purple People Eater. Also, assume that in order to use an item inside another *i.e.* coat or sack, Joe must first open it. As such, to use the nickel, Joe must first use the sack, then the change purse. Also, assume that the cost is the same for each edge in this problem.

(2 points each)

- Draw the basic search tree assuming Joe will try all of these items against the people eater.
- Given the tree you have drawn, list the order that the items will be used in a breadth first search.
- Given the tree you have drawn, list the order the items will be used in a depth first search.
- Give an example from the tree of a depth-limited search with limit  $L=2$  (the start node is at depth 0 and its children at depth 1).
- Would using an iterative deepening search be helpful in this example? Why or why not?
- Give the time and memory complexity for both depth first and breadth first search and give a simple derivation and explanation of how they are computed.
- If the Purple People Eater was a tollbooth operator in its past life and a dime will make it go away, will one of the search types be more optimal. Why or why not? Would your answer change if Joe needed to use the really cool particle cannon? Explain.
- How could you augment this problem to work with a Uniform-cost search? Intuitively, do you believe such an augmentation would make more sense?
- Does trying each item like this seem like a reasonable thing to do? In real life when would it be reasonable to do so?
- Give some intuitive examples and explanations as to how else you might try and select the correct item for Joe to use.

**Problem 2 (Programming 60%, Analysis 20%):** Homer Simpson is at the nuclear power plant. A plutonium rod has fallen on his head. He no longer remembers the way to Moe's Tavern. You will need to help Homer find the way so he can enjoy a frosty Duff™ beer after work.

To accomplish this, you will be given a list of street intersections, which list the streets that meet each other. You will be required to create a program in GNU C++ (g++) that finds the list of streets Homer must travel on in order to get to Moe's. You may either use the g++ version on Aludra or g++ versions 3.3 to 3.4.1 on Linux. Be sure and *clearly* note which one you used so I know which machine to compile it on.

Your program must take in an arbitrarily large list of intersections and find the correct route to take. It will then output a list of each street taken in order (see example file). An example **input** would be a list of intersections (edges):

```
Nuclear_Plant Burns_Ave
Burns_Ave West_Springfield_Bvld
Burns_Ave Ye_Olde_Fortran_St
Burns_Ave Krustofski_Way
Moes_Tavern Krustofski_Way
```

Your program would return the **output** of streets and other connecting locations traveled over including the starting and finishing states, in order of travel:

```
Nuclear_Plant
Burns_Ave
Krustofski_Way
Moes_Tavern
```

For the time and memory requirement file, **output** the time and memory requirements in the following format:

```
DFS 5 5
BFS 5 6
```

The first column is the time requirement while the second column is the memory requirement.

You can assume that capitalization will be maintained and node names will not have any white spaces in them. Additionally, the very first row has the starting state, while the very last row has the ending state just in the same way as the example. So, with other test cases, you must be able to help Homer find his way to other places as well. After all, you never know, he might want to go home to his loving family.

Hint: It may be helpful to use the C++ member function 'compare' or the C function 'strcmp' function to transform the street names into states. These functions will compare two strings to let you know how similar they are. Thus, one way to approach the problem is to read the list of

streets and if a street name has not been read in yet, create a state for it. Thus, keep some sort of list for street names and which states they correspond to.

You must solve this problem using both **depth first** and **breadth first** search. Additionally, you should keep track of memory and time requirements. Time required is tracked by incrementing a counter for each node that is expanded. Memory required is tracked by storing the largest number of unexpanded nodes, which were stored in memory at any given time. These should be output by your computer program at the same time as the solution. Both your final solution and your time/memory requirement answer should be output to a file, you should output the time and memory requirements to a separate file. Also note that to simplify grading, your screen output should be sparse. So please remove any extra cout and printf statements from your program. Please keep in mind that the streets of Springfield can lead many places, including in circles. So you should be able to detect loops in this problem.

**Your program should output three files:**

If your input is example01.txt, your output files will be:

- (A) example01.depth.txt – this is your depth first search solution
- (B) example01.breadth.txt – this is your breadth first search solution
- (C) example01.requirements.txt – this is your computed requirements

You will be provided with at least one set of example solution files, which you can use to make sure your output, matches the standard.

**Thus your program should:**

- (A) Read in a list of names as detailed above and create associated states.
- (B) Find the route from the start to the ending states.
- (C) Write files with the solution routes, as detailed above.
- (D) Write the time and memory required to a file.
- (E) Maintain the format of the example file(s).

**What you should hand in:**

- (A) Your source code – Be sure that it works with the stubby Perl script.
- (B) Readme.txt – This is your readme file and should contain the usual readme stuff.
- (C) Aludra.txt *xor* Linux.txt – This is a blank file that just makes it clear where you developed your code.

**What you should NOT hand in:**

- (A) Binaries – We will compile your code, this takes up a whole lot of space to store if everyone includes their binaries.
- (B) Stubby.pl – We run our own Perl script, we will not run your copy at all.

- (C) Solution files – We will run your code to derive the solution files. We will not look at any solution files you hand in since you can just fabricate them.
- (D) A make file – Note that stubby does not work with a make file, so you should not create and use one.

If you have some problems with C++ or would just like a handy reference I recommend the following book:

*C++ Black Book*  
*Steve Holzner*  
ISBN: 1-57610-777-9

I'm not sure if it's at the bookstore. However, it is available on amazon.com for \$23.00. As far as C++ books go its rather comprehensive over all the basic C++ topics and has easy to read and understand examples.

**Analysis and questions:**

- (A) (5 points) Analyze the memory and time requirements for depth first and breadth first search over the example files.
  - a. Show the mean and standard deviation of the number of steps needed to execute depth and breadth first search (be sure to show your work).
  - b. Show the mean and standard deviation for the amount of memory required given the examples. You can represent this as the maximum number of nodes, which must be kept in memory at any one time for each example.
- (B) (5 points) From your analysis, which of the two methods yields superior results. Be sure and give a careful explanation as to how you reach your conclusion.
- (C) (5 points) What if we added more than one goal state to each graph and we asked you to find the shortest (optimal) solution, would that change the complexity of the problem? Back up you explanation with a simple derivation of the complexity requirements.
- (D) (5 points) What if in addition to adding more than one goal, we added a cost to each intersection (edge) such as the wait time at the traffic light. Would a breadth-first search still make sense? Explain.
- (E) (0 points) If Matt Goening stuffed all his money under a mattress, how large would his mattress need to be in order to cover all of his cash?