CS 456 : Advanced Algorithms

Programming Assignment #03

Total Points: 150

Assigned Date	: Thursday, November 16, 2017
Due Date	: Thursday, December 07, 2017 @ 09:29:59 a.m. (**hard deadline**)

Overview

For this **programmming assignment**, you will implement two programs that will generate directions to go from one vertex to another vertex in a weighted graph. One program will use Floyd-Warshall and the other will use A* (pronounced A star). This video `https://www.youtube.com/watch?v=ySN5Wnu88nE&t=0s` gives an explanation of how A* works.

Each program should accept as a command line argument a file path, which is relative to the executable's location, which leads to a file that contains the graph information. Information on the graph and file format will be given below.

Once the program is launched, your program should prompt the user for a starting location and a desired destination. Then, your program should return as output to the user a set of points which is the path from start to destination, the total cost to get there, and how long it took to generate the path. Then repeat the process until the user closes the program. You should provide a clean exit option for the user.

The user query should be done as quickly as possible. To accomplish this, consider how both of the algorithms work when you write each of the programs in order to accomplish this.

The purpose of this project is to not only analyze the runtime of different algorithms on the same problem but to consider how an algorithm would be used in an actual real world application, to consider the differences in actual execution in order to achieve better results, and to consider the algorithms in the face of a problem that is persistent over time and could also change.

Bonus points are possible for insights and implementations that improve execution times compared to these. Think about the data you are working with and how it will affect possible solution algorithms. If you come up with a better way, implement it and show it is faster or explain why it is better.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- You may use any programming language of your choice. However, you **must** make sure that your code compiles and runs on a typical Linux machine.
- Absolutely **DO NOT** include executables with your submissions.
- You **MUST** submit a makefile. If your program does not need to be compiled, then have the makefile output instructions on how to execute your program instead.
- It is highly recommended that you test your program's compilation and execution on the home server, **home.cs.siue.edu**, before submitting.
- The report part of your solution must be produced using a word processor. I highly recommend LaTeX.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications.
- Graphs/plots should be properly labeled.
- Your final report must be in **PDF** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- Include only the things necessary to run your program in your tarball.

Deliverables

The due date of this assignment is **Thursday, December 07, 2017 @ 09:29:59 a.m.**. This is a **hard deadline**; there will be no late submission period. A dropbox will be opened for submission on Moodle before the due date.

Your submission should be your report and a compressed tarball named your last name comprised of the makefile to compile your program and your source code.

The tarball should be such that when it is decompressed it will create a directory that is named your last name. Your makefile should be in the base directory created.

The makefile should put your executables into this same location. Your executables should be called astar and fw.

For example:

Submission to moodle:

```
lastname.tar.gz lastname-report.pdf
```

Report

Your report should be structured as follows:

- Motivation and Introduction
- Flloyd-Warshall Program
  - Pseudo code for the program as a whole.
  - Justification for why you plan to implement the program in this manner.
  - Pseudo code of the Flloyd-Warshall algorithm.
  - A graph showing execution time. You should use enough data points to establish a clear pattern and large enough data sizes such that the dominating factor in execution time is data size.
  - Problems encountered. What problems did you encounter when trying to implement the program?
  - Observations and Insights. Here you should describe any observations you had about the results and any insights you had about the problem.
  - Conclusion. Here you should describe if the algorithm is a good fit to solve this kind of problem and provide reasoning to backup your answer. Take into consideration also what happens if the dataset is changed, i.e. a new vertex or edge is inserted, an existing vertex or edge is removed. How would this affect your implementation? What would you have to do in order to get to a correct working state? In what scenario is this algorithm a good fit? In what scenario is this algorithm a bad fit? What affect would negative weight edges have?
- A* Program
  - Pseudo code for the program as a whole.
  - Justification for why you plan to implement the program in this manner.
  - Pseudo code of the A* algorithm.
  - Details on the heuristic you used. Show that it is admissible.
  - A graph showing execution time. You should use enough data points to establish a clear pattern and large enough data sizes such that the dominating factor in execution time is data size.
  - Observations and Insights. Here you should describe any observations you had about the results and any insights you had about the problem.
  - Problems encountered.
  - Conclusion. Here you should describe if the algorithm is a good fit to solve this kind of problem and provide reasoning to backup your answer. Take into consideration also what happens if the dataset is changed, i.e. a new vertex or edge is inserted, an existing vertex or edge is removed. How would this affect your implementation? What would you have to do in order to get to a correct working state? In what scenario is this algorithm a good fit? In what scenario is this algorithm a bad fit? What affect would negative weight edges have?
- Report conclusion. Here you should offer up a comparison between the two implementations. Offer reasoning as to when you would use one over the other.

If you have more you want to say on this problem, feel free to include it. The above is only starting points to get you thinking on how to deal with the problem of considering what goes into choosing an algorithm to solve a problem in a larger sense than just perform a single action as fast as possible.

File Details

Graph Data File

This contains vertices and edges of the graph. The vertices are points in 2D space. The format will be name:x,y. It will be space delimited.

Then, the second line will be a list of edges in the form name1,name2:weight

For example:

```
1:0,0 2:1,3 3:10,30
1,2:10 2,3:5 1,3:1
```

This will create a graph with point 1 at (0,0), point 2 at (1,3), and point 3 at (10,30) with an edge from 1 to 2 with a weight of 10, an edge from 2 to 3 with a weight of 5, and an edge from 1 to 3 with a weight of 1.

Program Execution Example

Launching program

The program will be launched on a command line

```
astar <filepath to graph data>
```

The user should be prompted with something like the following

```
Enter desired starting point: 1
Enter desired destination: 3
For point 1 to point 3
The shortest path is 1 3
Number of points in path is 2
Total cost is 1
```

Tentative Grading Rubric

Program execution (50)

- Handles user input correctly. [**20 points**]
- Outputs correct results in a readable manner. [**20 points**]

Correct Submission Format [**10 points**]

For each program (50x2)

- Program pseudo code and justification [**15 points**]
- Algorithm pseudo code [**5 points**]
- Problems, Observations, Justifications [**10 points**]
- Conclusion [**20 points**]