CS 456 : Advanced Algorithms
Programming Assignment #02

Total Points: 150

Assigned Date : Wednesday, March 16, 2016
Due Date : Wednesday, March 30, 2016 @ 02:59:59 p.m.

Overview

For your second programming assignment, you will implement three versions of **Dijkstra's Algorithm** – using an adjacency matrix and an unordered array, using an adjacency list and a min-heap, and using an adjacency list with a Fibonacci heap as discussed in Section 9.3 in the Levitin book – and empirically validate their asymptotic runtime behavior on various graph types using **computer generated results**. More specifically, you are expected to think about and address the following questions:

a. At what size of the input data n_0 does each version start to exhibit asymptotic behavior?
b. What type of input data (in this case, graphs) do you need to show the asymptotic behavior for each version of the algorithm?
c. How do you plan to generate the appropriate graphs?
d. How does the empirical (i.e., measured) runtime correspond to the theoretical complexity analysis (such as counting the number of basic operations as discussed in class)?
e. How to create your test driver so that it demonstrates the various versions of the algorithm?
f. How to create the algorithm class so that it will be extensible and reusable for future projects?

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- Make sure to test your program properly before your final submission.
- 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. A Makefile is mandatory. Please be sure to test your makefile.
- The report part of your solution must be produced using a word processor. Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. Be professional with your reports; properly label and title your graphs; properly caption and cross-reference your figures; make sure to include all sections/subsection mentioned below.
- Your final report should 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.

- It will be necessary to generate many graphs, so it would be a good idea to start early producing graphs. This can either be accomplished by creating your own generator that allows for variations of edges and vertexes. Also, datasets and generators can be found on online that model networks. `https://www-complexnetworks.lip6.fr/~latapy/FV/generation.html` and `http://snap.stanford.edu/data/` are two such examples. However, you will have to manipulate the datasets into the form needed for this assignment. The grader has found it easier to start a little early and just write his own generator.
- For input, prompt user for the input filename.
- An initial file will be given, **small.txt**. It will represent a small, directed graph. The given file will be used to test the accuracy of your program. It would be beneficial to test this file on your program beforehand. It is important that you develop your own input files to truly test the algorithms and obtain solid data.
- All input files should follow an adjacency list format that has each line containing the vertex name followed by a list of the adjacent vertices and their weights. Only integers and spaces will be in the files. See example below.
- Treat the first vertex read from file as the source.
- After a file is read, it is important that the same graph is ran by all three versions. This will be helpful in gaining a better understanding of how the choice of data structure, even for the same algorithm, influence the overall runtime behavior. It is important to note that the input file must be converted to a matrix and a list to properly run the three versions.
- For output, generate a file named **[inputFileName]Out.txt**. The top three lines should include the times of the three algorithms followed by the individual results of each algorithm separated by a header naming the algorithm.
- Dijkstra's algorithm produces a shortest path tree from the source to all other vertices; however, for the sake of brevity, it will only be necessary to print to the output file the path to the destination vertex whose path had the largest total weight of all found paths(if multiple paths have the highest total weight, print the path with the lowest numeric vertex as destination). A string of vertex separated by spaces. See example below.
- Be sure to test enough different size sets to accurately graph behavior.
- Total points: [**150 points**]

Deliverables

The due date of this assignment is **Wednesday, March 30, 2016 @ 02:59:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- [**56 points**] A report that includes the followings:
  - Motivation and background of the experiment [**5 points**].
  - Pseudocode of your algorithm appropriately annotated with the theoretical runtime analysis. It is advised to add a code walkthrough of the algorithms that explains why they have the time complexity that they have. [**5 points**].
  - Testing Plan and Test Results. Note that it is important to test multiple graphs of varying number of vertexes with same number of edges, as well as graphs with same number of vertexes but different edges. In essence, sparse vs dense graphs. A thorough discussion of your testing plan should include how you plan to demonstrate the characteristics of the algorithms and how to exploit them [**10 points**].
  - A correctness proof of the Dijkstra's algorithm [**5 points**].

  - Problems Encountered/Key insights [**5 points**].
  - Justification of your observations. You must be able to justify and/or argue the empirical asymptotic behavior you are observing [**13 points**].
  - Conclusion and performance comparisons [**13 points**].

- [**94 points**] A compressed tarball of the directory containing your source codes, Makefile, and a few samples of different graph types. Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name-pr2.tar.gz source/`. Obviously, change the name to your last name.

Tentative Grading Rubric

- Styling: Compiles, easy to read, properly indented, etc..., [7 points]
- Input specifications [8 points]
- Output Specifications [10 points]
- Algorithmic Correctness[69 points]
  - Dijkstra's Algorithm Adjacency Matrix O(V^2)[23 points]
  - Dijkstra's Algorithm Min-Heap O(V + E)(log V)[23 points]
  - Dijkstra's Algorithm Fibonnaci Heap O(Vlog V)[23 points]
- Report Specifications[56 points]

Sample input file

in format:
vertexname adj.vertex weight adj.vertex weight.....
small.txt

```
1 2 1 3 1
2 4 2
3 2 2 5 4
4 3 3 5 3
5 1 4
```

Sample output file

smallOut.txt

```
Adjacency Matrix: .0256s
Min-Heap: .0212s
Fibonnaci Heap: .0125s
Dijkstra Adjacency Matrix
1 3 5
Dijkstra Min-Heap
1 3 5
Dijkstra Fibonnaci Heap
1 3 5
```