CS 456 : Advanced Algorithms

Programming Assignment #02

Total Points: 150

Assigned Date : Wednesday, February 28, 2018
Due Date : Wednesday, March 21, 2018 @ 02:59:59 p.m.

Overview

For your second programming assignment, you will tackle the **All-Pairs Shortest Path Problem** (APSP) using two different approaches – **Floyd-Warshall** Algorithm and the **Johnson's** Algorithm – and empirically validate their asymptotic runtime behavior on various graph types using **computer generated results**. **Note**: Johnson's Algorithm (CLRS pgs. 700 – 706) uses **Bellman-Ford** Algorithm (to detect negative-weight cycles), and **Dijkstra's** Algorithm as subroutines. As discussed in class, the runtime of Dijkstra varies based on the underlaying data structure being used. For this assignment, you are expected to implement two variants for Dijkstra – **Fibonnacci Heap** (CLRS Ch. 19), and **Min-Priority Heap** (CLRS Sec. 6.5).
By the time you complete this assignment, you are expected to have implemented a total of five algorithms – Floyd-Warshall, Bellman-Ford, Dijkstra+Fibonnacci, Dijkstra+Min-Heap, and Johnson's. You are expected specifically to think about and address the following questions during your testing:

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 asymptotic behaviors for same input?
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!!**.
- You may use any programming language of your choice out of the four major languages – `C`, `C++`, `Java`, `Python`. 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. The report must be in **.PDF** format. No exceptions.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications.Graphs/plots should be properly labeled.
- 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. Absolutely do not include executables of any format with your submission.
- Include any additional instructions in a README file if needed.

Logistics

Given the nature of the assignment, it will be necessary to generate many different types of large graphs for testing; 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 vertices. 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. Note: You might have to manipulate the datasets found online into the form needed for this assignment.

- For input, prompt user for the input filename (along with the path, if not under `pwd`).
- All input files should follow an adjacency list format that has each line containing the vertex name followed by a list of the adjacent vertex:weights tuple. `e.g. A B:3 C:4` means A is connected to B and C with cost/weight 3 and 4 respectively. Only alpha-numerals, spaces, and : allowed in input files. The sample input below depicts a non-negative directed graph.
- It is important that the same input graph is run on all algorithm variants you implement for proper comparison. Also make sure to run the same input file multiple times and take the average to eliminate statistical outliers.
- For output, generate a file named **[inputFileName]Out.txt**. The top three lines of this output file should include the empirically measured runtimes for all your implementations followed by the individual results of each algorithm separated by a header naming the algorithm.
- Be sure to test enough different size inputs to accurately graph empirical behavior.

Important: It is important to test your implementations against varies types of graphs to fully appreciate the effectiveness of different algorithms and their utility. As such, ensure to test your implementations against sparse vs. dense graphs, directed vs. undirected graphs, as well as non-negative vs. negative weighted graphs.

Deliverables

The due date of this assignment is **Wednesday, March 21, 2018 @ 02:59:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- [**120 points**] A report that includes the following sections:

  - [**5 points**] Motivation and background of the experiment .
  - [**90 points**] For each of the three implementation variants – Floyd-Warshall, Fibonacci-based Johnson, Min-Heap-based Johnson, in their own sections, per variant
    - ◇ [**5 points**] Appropriately annotated Pseudocode with 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.
    - ◇ [**8 points**] Correctness proof.
    - ◇ [**12 points**] Test results. This should naturally include graph(s) showing empirical execution time in comparison to the corresponding theoretical runtime. You should ensure to use sufficiently large input to establish a runtime behavior pattern.
    - ◇ [**5 points**] Observations and insights. You must be able to justify and/or argue the empirical asymptotic behavior you are observing.
  - [**25 points**] Overall conclusions and performance comparisons. This includes comparison of different algorithmic approaches to the problem against each other, comparing and contrasting runtime behavior, as well as a discussion on the impact of data structures where applicable. This section should also explain the graphs that were displayed in the test results section

- [**30 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.

Sample input file

in format:
vertexname adj.vertex weight adj.vertex weight.....
small.txt
1 3:6 4:3
2 1:3
3 4:2
4 2:1 3:1
5 2:4 4:2

Sample output file

smallOut.txt
Floyd-Warshall: .0256s
Johnson Min-Heap: .0212s
Johnson Fibonnaci Heap: .0225s

Final Solution:
0 4 4 3 -1
3 0 7 6 -1
6 2 0 2 -1
4 1 1 0 -1
6 3 3 2 0