CS 456 : Advanced Algorithms
Programming Assignment #03
Total Points: 150

Assigned Date : Tuesday, November 12, 2019
Due Date : Tuesday, December 03, 2019 @ 09:29:59 a.m. **(firm deadline)**

Overview

For your third **programming assignment**, you will implement three versions of a **Max Flow Algorithm** – Ford-Fulkerson, Dinic's Algorithm, and Generic Push-Relabel – and empirically validate their asymptotic runtime behavior using **computer generated results**. While the Ford-Fulkerson and Generic Push-Relabel algorithms are discussed in your CLRS book, Dinic's algorithm is not. The paper by the author found at `https://www.cs.bgu.ac.il/~dinitz/Papers/Dinitz_alg.pdf` should be helpful as well as several videos found on youtube made by Tim Roughgarden. Specifically, you are expected to think about and address the following questions:

a. What are the input conditions that most affect the different algorithms?
b. What conditions favor or disfavor a particular algorithm being chosen over another?
c. In particular, one algorithm has a different theoretical runtime than you are used to seeing, one where the output is reflected in the bound. Why is this and what affect does it play?
d. At what size graphs do the algorithms begin to show the difference that their individual complexities should demonstrate?
e. How does the measured run time correspond to the theoretical complexity analysis?

Note: You are expected to explicitly answer the above questions in your report.

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.
- 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 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.

- For input, prompt user for 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 test your implementations on large graphs and/or 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 colon (:) separate `vertex:edge` tuple list.
- Treat the first vertex read from file as the source and the last vertex as the sink.
- After a file is read, it is important that the same graph is ran by all three algorithms. This will be help to get a better understanding of the differences in the algorithms.
- For output, generate a file named **[inputFileName]Out.txt**. The top three lines should include the times of the three algorithms followed by the maximum flow discovered.
- Be sure to test enough different size files to accurately graph behavior.
- Total points: [**150 points**]

Deliverables

The due date of this assignment is **Tuesday, December 03, 2019 @ 09:29:59 a.m.** A dropbox will be opened for submission on Moodle before the due date. Note that PR03 has a **firm deadline**, which means the typical 48-hour late submission period is not available.

A complete solution comprises of:

- A report that includes the followings:
  - Motivation and background of experiment. Explain possible uses for these algorithms and your learning objectives of this 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. [**12 points**]
  - Testing Plan and Test Results including explanations of the sample input files used to test your algorithms. In addition, any code turned in as a deliverable will only run one time on one particular graph designated by an input file. This would be cumbersome when testing multiple graphs and having multiple runs per graph to determine timing. How did you design your code for easy manipulation to minimize the testing time? [**12 points**]
  - Graphs displaying the timing results of your algorithms. Be sure to discuss the complexity of the algorithms and how closely your program follows the theoretical complexities. A minimum of the following graphs are required:
    - A theoretical-actual comparison graph of the timing results of the **Ford-Fulkerson Algorithm**. [**4 points**]
    - A theoretical-actual comparison graph of the timing results of the **Dinic's Algorithm**. [**4 points**]
    - Another theoretical-actual comparison graph of the timing results of the **Generic Push-Relabel Algorithm**. [**4 points**]
    - A graph comparing the run times of the three algorithms. [**8 points**]
  - Justification of your observations. You must be able to justify and/or argue the empirical asymptotic behavior you are observing. Be sure to discuss what graph sizes and properties begin to display asymptotic behavior, for which algorithms do the different graphs run

best, which graph properties cause worst case, and any graphs that produced irregular results. Note: You must explicitly answer (and appropriately justify) the questions listed in the Overview section. [**10 points**]
  - Conclusion and problems encountered/key insights. Explain in your own words what you learned from this experiment. [**5 points**]

- [**86 points**] A compressed tarball of the directory containing your source code and **(Makefile. Important:** Be sure to test your code on a Linux machine. No exceptions! 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_pr3.tar.gz source/`. Obviously, change the name to your last name.

Tentative Grading Rubric

- Styling: Easy to read, properly indented, etc..., [8 points]
- Input specifications [8 points]
- Output Specifications [10 points]
- Algorithmic (i.e. empirical) Correctness [60 points]
  - Ford-Fulkerson Search [20 points]
  - Dinic's Algorithm [20 points]
  - Generic Push-Relabel Algorithm [20 points]
- Report Specifications[64 points]

Sample input file

in format: *Vertex< >Adj.-Vertex:Edge-Weight< >Adj.-Vertex:Edge-Weight* $\cdots$
small.txt
0 1:6 2:7
1 2:1 3:3 4:4
2 3:2 5:5
3 4:3 5:2
4 6:7
5 4:2 6:4
6

The format follows that of the graph generator from PR01, which you can use to generate large graphs.

Sample output file

smallOut.txt
Ford-Fulkerson: .0256s
Dinic's Algorithm: .0212s
General Push-Relabel: .0202s

Max Flow: 11