INFSCI 2591: Algorithm Design
Project 2: Optimization Algorithms
Due: October 11, 2016

The objectives of this project are: (a) to learn how to design and implement optimization algorithms (greedy and dynamic programming) and (b) to learn how to design and implement algorithms using different data structures.

Data Structures

In this project you must design and implement Dijkstra's algorithm and Floyd's algorithm to compute all-pair shortest paths in any given graph. You must implement 6 programs: 3 programs for Dijkstra's algorithm (each program with one of the 3 data structures below) and 3 programs for Floyd's algorithm (each program with one of the 3 data structures below).

1. An adjacency matrix using a two-dimensional array.
2. An adjacency list using a linked list.
3. An adjacency matrix using a one-dimensional array for storing only the elements of the lower triangle in the adjacency matrix.

Input

The input to your 6 programs must be connected, undirected, and weighted graphs. Your programs must be able to find shortest paths on two types of connected, undirected, and weighted graphs: complete graph (a graph with a link between every pair of nodes) and sparse graph (a graph with exactly n-1 links). To prepare graphs as inputs to your 6 programs, your project must include these additional programs (functions) as follows:

1. **Random**. A program to randomly generate a number between 1 and n. You can use a random number generator function in the libraries of your programming language or write your own function.
2. **Complete**. A program that can create an adjacency matrix for an undirected complete graph with any given size n.
3. **Sparse**. A program that can create an adjacency matrix for an undirected sparse graph with any given size n. The Sparse program must use the Random program (number 1 in this list) to determine all the links (n-1) in the sparse graph.
4. **Weight**. To include weights on the links of the undirected complete graphs (generated by the Complete program, number 2 in this list) and on the links of the undirected sparse graphs (generated by the Sparse program, number 3 in this list), use the Random program (number 1 in this list) to generate positive numbers as weights.

Test Cases

Your 6 programs (Dijkstra's and Floyd's, each with 3 different data structures) must be tested by using the test cases available on courseweb.

Performance and Memory

You must report on the performance and on the memory requirements of the 3 data structures. Performance is defined as the total time taken to construct the data structure and compute all-pair shortest paths. To measure performance, you must run each program (Dijkstra's and Floyd's) for different sizes of complete and sparse graphs (e.g., graphs of 100, 1000, 2000, 3000, 4000, 5000 and 6000 nodes). For each case (e.g., Dijkstra's algorithm, the first data structure listed under Data Structures above, and sparse graphs with 100, 1000, 2000, 3000, 5000, and 6000 nodes), you must plot the number of nodes on the X-axis and the time on the Y-axis. Note that you must prepare a total of 12 plots: 2 different algorithms, 3 different data structures, 2 different graph types with different sizes. For the memory requirements, you can derive the information based on the data structures you used in your programs or report a summary statistics provided by your programming language tools, if available.

Submission

Submit a report in a PDF file that contains: (a) the pseudocode for the two algorithms; (b) 12 performance plots; (c) a discussion on the memory requirements of each of the 3 data structures and (d) the outputs (results) of your 6 programs for the test cases. You must submit the source code for each program (a total of 6 source codes: 2 algorithms implemented in 3 different data structures) in a separate zip file. All submissions (PDF file and zip file) must be made through courseweb.

Total: 200 points

- Pseudocode for the two algorithms (5 points/algorithm): 10 points
- Six correct programs (source code) with the results for the test cases (15 points/source code): 90 points
- Readable (code documentation) codes (5 points/source code): 10 points
- Report on memory (for each data structure in each program): 30 points
- Report on time performance (12 plots): 60 points