EC 504
Spring, 2022
HW 4 Software

Due Monday, 4/11/22, 8 pm

Note: This must be turned in by creating a HW4 folder in your directory in /projectnb/students/ec504 directory on the SCC cluster (scc1.bu.edu). Please turn in the output file and the source files for your code. Repeat what you did in previous homeworks.

1. (50 pts) **Finding Shortest Paths** In this problem, you will implement three versions of shortest path algorithms: a simple version of Dijkstra's algorithm using a list instead of a priority queue, a version of Dijkstra's algorithm that uses a binary heap as a priority queue (the binary heap from HW 2 will work fine here, or you can choose any other), and a version of Bellman-Ford algorithm that uses a work queue.

   I've provided two files: `Shortest_paths.cpp` is the main program with all the timing routines included. It reads a directed graph from a file. It loads the information into an array of vertices called Nodes. Associated with each node is a linked list of outgoing edges (arcs). The Node structure and the arc structure are defined in an include file `shortPath.h` , along with the three shortest path algorithms that need to be developed.

   A makefile is included. To compile your executable, simply type `make shortest` . Three different input graphs are provided: Graph1.txt, Graph2.txt, Graph3.txt. A fourth input graph, smallgraph.txt, is provided along with the output file smallgraph1.txt_out to help you with debugging.

   Note that the vertices in the input files are numbered from 1 to Nm. The Nodes[] data structure hence ignores the Nodes[0] element, so that the index in the array equals to the node id. The element Nodes[0] has no information, and should not be used in any of the algorithms.

   To run the program, simply type `./shortest smallgraph.txt` . This will generate the output smallgraph.txt_out which you can compare with the output file provided.

   You are to run the executable shortest against the three graph inputs Graph1.txt, Graph2.txt and Graph3.txt. You will turn in a copy of all your include files, and the three output files Graph1.txt_out, Graph2.txt_out and Graph3.txt_out.

   To assist you with this assignment, I've provided a working solution for the simple Dijkstra algorithm that uses an array. This should help you see how to use the structures that are created in the main program to write the other two shortest path algorithms. I've also included my copy of a binary heap solution for HW 2, in case your HW 2 solution didn't work well. Thus, you have to complete the other two algorithms in `shortPaths.h`.

   Note that you can compile and run the codes as they are and get the correct output for one algorithm. You can use this to verify that your other algorithms are also getting correct outputs, as all three algorithms should find the same shortest paths.

   All the needed files are provided in the SCC cluster. The files are in the directory

   `/projectnb/ec504/HWProblems/HW4_codes` .

   If you program your algorithms well, your BellmanFord and DijkstraHeap algorithms should run four orders of magnitude faster than the simple Dijkstra algorithm I've provided, for Graph3.txt that has 100,000 vertices. This is due to the $O(|V|^2)$ complexity versus the $O(|E|\log(|V|)$ complexity of Dijkstra's algorithm with a binary heap. The surprise is the Bellman Ford algorithm's speed, which shows that an average case for this algorithm can be very fast when you avoid unneeded computations.