CPSC 2120/2121

Lab 09

Dijkstra's Algorithm and Kruskal's Minimum Spanning Tree
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To implement graph algorithms in C++
- To use simple data structures to implement algorithms
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement an algorithm and understand its intricacies

Problem/Exercise

For this assignment, you will read summary map information from a file and implement two common graph algorithms to find paths between nodes: Dijkstra's Shortest-Path algorithm and Kruskal's Minimum Spanning Tree (MST) algorithm. You are allowed to use any data structures in the C++ STL for this assignment. Your final output will be in a single file, named graphalgs.h, which will compile without errors using the provided Makefile. In order to thoroughly test your program, you will also submit a single input file named lab09.txt which has map information and queries in the same format as the provided lab09.txt. It will be necessary to add additional test cases to the lab09.txt file, following the provided example.

The final output of Dijkstra's algorithm will be a list of the shortest distances from the starting node to every other node in the map. The final output of Kruskal's MST algorithm will be a minimum spanning tree of the input.

Additional Requirements

- Your graphalgs.h file must compile and run with unmodified versions of the provided lab09.txt and Makefile to produce the same output as our examples. If your class does not, then you'll need to fix it. After you get your class working, you should create several examples on your own to further test your program to make sure it works with any valid set of test cases.

- To help you understand the inner workings of this algorithm, you must create your own. Once your program works correctly, draw a map of the nodes in the provided input file (lab09.txt) and work through each technique manually.
- Output will be to stdout.
- Output for Dijkstra will include the technique (Dijkstra), the starting node, and for all remaining nodes, the shortest path distance to those nodes.
- Output for Kruskal will include the technique (Kruskal), a comma-separated list of edges included in the MST, and the total weight of the MST

Examples

Your C++ source code should be in a file called graphalgs.h, and it should be compiled into an executable called lab09.out using our provided Makefile. The output of graphalgs.h must look exactly like the examples below (ordering of results can be different) when run on the command line on our Unix machines.

./lab09.out
---Test 1---
Dijkstra (from a)
b: 1
e: 12
d: 6
g: 6
f: 4
c: 3

Kruskal
a, b, 1
f, g, 2
f, d, 2
a, c, 3
a, f, 4
d, e, 6
total weight = 18

---Test 2---
Dijkstra (from Seatle)
Miami: 5580

Minneapolis: 2661
Boston: 4935
Chicago: 3322
Los Angeles: 1935
New York: 4850
Washington DC: 4467
San Fransisco: 1306
Denver: 2161
Las Vegas: 2225
Dallas: 3419

Kruskal
Boston, New York, 338
Washington DC, New York, 383
Las Vegas, Los Angeles, 435
San Fransisco, Los Angeles, 629
Minneapolis, Chicago, 661
Chicago, Washington DC, 1145
Las Vegas, Denver, 1225
Denver, Dallas, 1258
Seatle, San Fransisco, 1306
Denver, Minneapolis, 1483
Washington DC, Miami, 1709
total weight = 10572

Source Code Requirements

- Put a comment at the top of your source code file with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation as needed to explain ambiguous, tricky, or important parts of your code.
- All source code must follow good programming style standards such as properly indenting source code; and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.
- Your program must read input from a file. This file will be a list of weighted edges in the format ***first, second, weight*** with one such entry per line until a line is only "#####"

- All graphs are assumed to be undirected
- For Dijkstra's algorithm the source node will be the first node given in the test.

Submission

Before the date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to handin under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (graphalgs.h)
2. Your testing file (lab09.txt)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading

If your class does not compile on our Unix machines or your assignment was not submitted on time, you will receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with our testing program(s)	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
Alternate testing file containing map and query data you created for testing which is distinct from that provided.	1 point

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different from the provided lab09.txt) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.