

# CPSC 2120/2121

## Lab 05

### Breadth-First and Depth-First Search

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 a graph search 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 lab, you will read summary map information from a file and implement two algorithms to find paths between cities: a Breadth-First Search (BFS) on a graph, and a Depth-First Search (DFS) on a graph. 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 `worldmap.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 `lab05.txt` which has map information and queries in the same format as the provided `lab05.txt`. It will be necessary to add additional test cases to the `lab05.txt` file, following the provided example.

## Additional Requirements

- Your `worldmap.h` file must compile and run with unmodified versions of the provided `lab05.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 cities in the provided input file (`lab05.txt`) and show the final path taken by each technique (BFS and DFS).

## Examples

Your C++ source code should be in a file called `worldmap.h`, and it should be compiled into an executable called `lab05.out` using our provided Makefile. The output of `worldmap.h` must look exactly like the examples below when run on the command line on our Unix machines.

```
./lab05.out
```

```
BFS: Chicago, NYC, DC, Raleigh, Norfolk; distance traveled: 16
```

```
DFS: Chicago, NYC, DC, Raleigh, Clemson, Norfolk; distance traveled: 26
```

```
BFS: Clemson, Raleigh, DC, NYC, Norfolk; distance traveled: 20
```

```
DFS: Clemson, Raleigh, DC, NYC, Chicago, Norfolk; distance traveled: 26
```

```
BFS: DC, NYC, Raleigh, Norfolk, Chicago; distance traveled: 16
```

```
DFS: DC, NYC, Chicago; distance traveled: 9
```

## 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 in the format *city1, city2, distance*, with one such entry per line until a line is only `#####`
  - Following `#####`, there will be pairs of *source, destination* queries in the format, *city1, city2*
- For each query, print a breadth-first and depth-first traversal of the graph, as well as the total distance traveled to find the destination.
- In addition, for each traversal print out each city that is visited. For example
  - BFS: Boston, Philadelphia, Washington DC, New York, Baltimore, Ottawa, Virginia Beach; distance traveled: 54
  - DFS: Boston, Washington DC, Virginia Beach; distance traveled: 25

## 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 (worldmap.h)
2. Your testing file (lab05.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 lab05.txt) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.