Project 3 – Graphs

Overview

This project is to create a graph. A graph is a collection of vertices and edges that connect pairs of vertices. Graphs can be used to represent many real-world problems. The goal of this project is to build a graph and allow a user to explore that graph. Like other projects, this will build upon previous programming techniques used.

Requirements

You will read in a set of edges from a text file and construct a graph, represented internally using the adjacency list data structure. Once the graph is constructed, let the user pick a starting vertex and display the list of vertices they may travel to. They enter the number of the vertex, move to that vertex, and the process repeats.

In addition to this, you must also have display functionality that displays the entire graph. Nothing fancy required, just display a list of vertices, a colon, and then all the vertices adjacent to that vertex.

Sample Output

This is just an example. Feel free to display the output and menus however you would like.

```
Loading graph from “smallGraph.txt”.
Please enter the vertex to start at: 1

Currently at vertex: 1. Adjacent vertices: 5, 6, 10
What would you like to do? (M)ove (P)rint Graph (S)earch?: M
Please enter the vertex: 10

Currently at vertex: 10. Adjacent vertices: 1, 6, 2
What would you like to do? (M)ove (P)rint Graph (S)earch?: S
Please enter the value to search for: 8
Vertex 8 was found in this graph.

Currently at vertex: 10. Adjacent vertices: 1, 6, 2
What would you like to do? (M)ove (P)rint Graph (S)earch?: P
1: 5, 6, 10
2: 10, 4
3: 7
4: 2, 8, 7
(and so forth…)
```

Programming Languages

You may use any programming language as long as you confirm it with me first. This course will be centered on C++ and Java, so those are my recommendations.

Dataset

Three different graph datasets are provided. The first value in these text files will be the number of vertices. You can use the number of vertices to dynamically declare the size of your adjacency list. After that, each line contains an edge, which is really just two vertices separated by a space.

Hints

The adjacency list data structure is really an array of linked lists. You may be able to reuse Linked List code if you have it. Make sure to reject invalid moves if attempted. It may be helpful to manually draw out the graph. If you're using C++, learn how to use the STL vector class. There are other STL class that may be useful, ask me if you'd like details.

Extra Credit

If you're looking for an extra challenge, here are some extra features you may add for extra points. Once you have basic graph functionality there are a ton of features you could add.

- Implement a search method. This will let the user enter a value, your search algorithm will search the graph for it, and then report if the value was found or not on the graph. You may implement this using Breadth First Search (BFS) or Depth First Search (DFS).
- When doing your search, display the resulting path used to find the result and the length of the path (if result is found).
- Add needed functionality to turn the graph into a directional graph. Assume the edges in the text files you're reading in are only one way (this may be easier than you'd think).
- Implement both depth first and breadth first searching.
- Add a check to determine if the graph is fully connected and display if it is or not.

Grading

The assignment is worth 100 points. If your code doesn't compile, there is an immediate 30-point penalty. You will be graded on the following criteria:

Header comment block with name, class, date, project	10
Graph is read from input and constructed using adjacency lists	30
Graph traversal	30
Graph print feature	20
Overall organization and quality of code	10
Total	**100**

Submitting your work

Your work should be submitted as an archive (.zip, .7z, .rar) on Moodle. This archive should include all of your source code files. Please do not submit your entire project folder.

This project is due Wednesday, April 15th at 11:55pm.