Project A3

Due Date: March 29 (Note change from syllabus)

Purpose

There are several objectives for this project. One is to write a program using an algorithm covered in class; that is, figuring out the details of an algorithm that has been described in more general terms. In addition, the algorithm will have a major modification to the original. Finally, you will use some real-world data which can be of significant size.

Problem

Depth-first search is used to solve many problems, such as finding paths (like through a maze), checking for graph connectivity, topological sorting (when do you put that watch on??), and looking for the best move in games. In this project, you will implement a depth-first search using a graph in the form of intersections (vertices) and roads (edges) from the METAL project.

Input

The program should first prompt for a filename. The file will contain highway data in the "collapsed" TMG format as described in class and on the METAL web site. The program should then prompt for a vertex number at which to start the depth-first search.

Output

The program should display all of the edges in the spanning tree, as shown in the METAL HDX visualization. For each edge, the "place," the number of hops, and "arrive from" should be displayed one one line, such that the columns are aligned. The "place" is the current visited vertex, and "arrive from" is the previous vertex which completes this edge. You can compare your output to the that shown in METAL.

Specifics

- Follow the general DFS algorithm discussed in class **except** do not use recursion! This means you will have to store information on a stack. You can use the C++ STL stack implementation for this purpose.
- As usual, follow good OOP practices. There is some starting code that you **must** follow. This is available on the course web page. The `a3.h` file is shown below:

```
class Vertex {
   string label;   // info for vertex
   double x;       // x-coordinate (longitude)
   double y;       // y-coordinate (latitude)

public:
   // inline constructors
   Vertex(){};   // default
   Vertex (string slabel, double xcoord, double ycoord) {
      this->label = slabel;
      this->x = xcoord;
      this->y = ycoord;
   }

   // getters
   double getX() { return x; }
   double getY() { return y; }
   string getLabel() { return label; }

   // other method declarations go here
};

class Edge : public Vertex {
   Vertex start;   // "starting" vertex of edge
   Vertex end;     // "ending" vertex of edge

public:
   // inline constructors
   Edge () {};     // default
   Edge (Vertex one, Vertex two) {
      start = one;
      end = two;
   }

   // getters
   Vertex getStart() { return start; }
   Vertex getEnd() { return end; }

   // other method declarations go here
};

class DFS : public Edge {
   // declarations here, including, perhaps, private methods

public:
   void readTMG ();   // method to read TMG file
   void runDFS ();    // run DFS
};
```

Here is `a3main.cpp`:

```
#include "a3.h"

void DFS :: readTMG () {
   // your code here
}

void DFS :: runDFS () {
   // your code here
}

int main() {
   DFS graph;

   graph.readTMG();
   graph.runDFS();

   return 0;
}
```

- You must use the above code as written, but you can add any additional code as described in the comments. You can also add methods and even additional classes, if desired.
- The `Vertex` and `Edge` classes should include only methods that are applicable to **general graph problems**. That is, these classes should not have any code specific to DFS. We will reuse this code for future projects which should be added easily to the code you write for this program.
- Choose the next incident vertex according to its number in the file, lowest first.
- You should do an error check on the input filename. If the filename does not exist, the program should display an error message and exit gracefully.
- You do not have to do any error checking of the data contained in the file.
- As before, include an introductory comment (name, description, etc.) at the top of your `.h` file.
- Remember to adequately comment your classes, methods, and all variables.

Notes

- Be sure you are reading the TMG data file correctly before continuing! No use trying to do the DFS if you you can't get the data properly. Proceed one step at a time!
- Start with small files to make sure your algorithm works properly. Then make sure your program still works with large files (e.g., no crashes!).

- Compare your output to the output produced by METAL.
- Zip your code together using the same naming convention as before, as in `gousieA3.zip`. **Zip only your code! I do not want either of the `__MACOSX` or `cmake-build-debug` folders!** Figure out how to zip up only your `.cpp` and `.h` files, not the entire CLion (or other) folder!
- A printed version of your source code is due in class on March 30th. Write/print and **sign** the Wheaton Honor Code Pledge on what you turn in: "I have abided by the Wheaton College Honor Code in this work."

It is not length of life, but depth of life.
– Ralph Waldo Emerson