Longest Path Algorithm Project Guide

A guide to the download, use, and code of this project.

# Introduction

This project is a demonstration of the routing algorithm that will be used in the routing protocol. The project was created in Eclipse using C++ (C++11 and above) and tested on a linux machine set up with the same tools as the OpenWRT Build System machine.

This guide will walk through the steps to downloading this project from GitHub, how to build and run the project correctly, and a summary of the classes and functions within the code.

# Download

1. Go to <https://github.com/crumbj/Scenic-Routing>
2. Click the green Code ↓ button
3. Either select Download Zip or Copy the SSH line. Enter the following line into the terminal where you want to download the folder to.

* git clone [git@github.com](mailto:git@github.com):crumbj/Scenic-Routing.git

1. The project will be located in the Routing Algorithm Project folder within the downloaded project.

# How to Use

Open a terminal and change into the directory /Scenic-Routing/Routing Algorithm Project/Debug

This is the folder that contains the Makefile used for building the project with the right dependencies.

Enter the following commands to run the project:

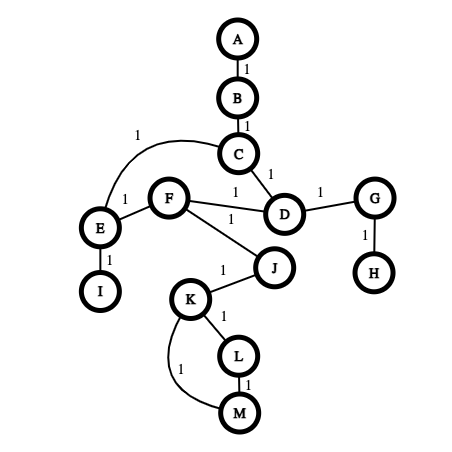
make clean

make all

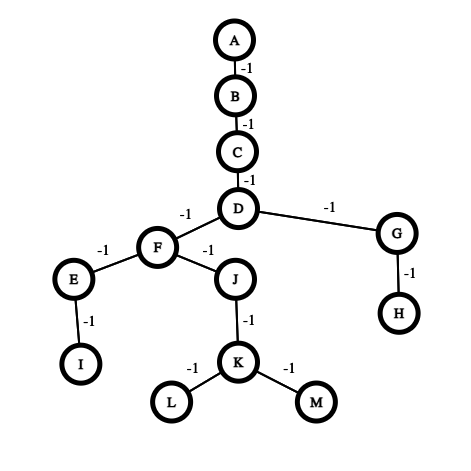
./longestPath

This program will ask you to enter two nodes from the graph in the file “colorpoints.txt”

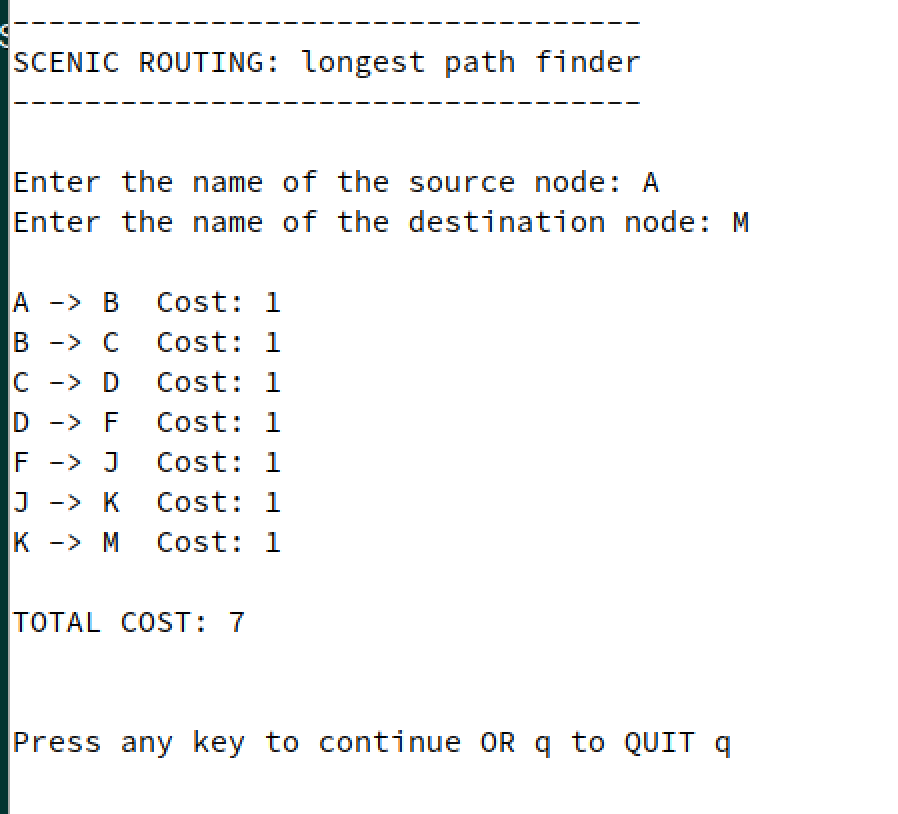
The graph contained in “colorpoints.txt” looks like:



The resulting maximum spanning tree created from the text file looks like:



Pick any two nodes on the graph, and the program will print out each edge and weight in the path. Here is an example of the output for this graph:



If you want to test other graphs, edit the contents of “colorpoints.txt”. This algorithm works for undirected graphs with simple cycles. If there are too many loops that include the same nodes, not every loop will be used (in undirected graphs there can be an infinite number of loops, so it's not possible to get them all.)

The graph files must be set up in the following way:

number of nodes

the name of each node on a separate line

number of edges

each edge in the format -> nodeName1 nodeName2 edgeWeight

# Code Explanation

### Quick Definitions:

Node - a vertex object that includes a node ID number and name.

Edge - an object containing a left Node, right Node, and weight

Graph - an object that contains a list of all nodes and a list of all edges in a graph along with an adjacency matrix in the form of a map using the node name as a key and list of edges containing the node name and the adjacent node as the value.

### Algorithm Classes:

#### Graph Class

Within the Graph Class there is a function called breadthFirstSearch that takes in two node IDS from the graph and uses the Breadth First Search Algorithm to find the path between these nodes, returning a vector of the names of the nodes in the path.

#### Kruskal Class

The Kruskal Class performs Kruskal’s algorithm on a graph to produce a Maximum Spanning Tree through the useKruskal function. Here are the following functions used within useKruskal :

1. negateAllWeights - negates all of the weights of the edges in the original graph in order for Kruskal’s algorithm to create a maximum spanning tree (MST).
2. sortEdgesLowToHigh - all edges within the graph must be sorted in descending order because the edge weights are now negative, so the algorithm chooses the edges with the largest original weights to add to the MST first.
3. findSet - part of the UNION-FIND algorithm. divides the nodes into clusters recursively and allows us to check if a node is in the same set as another and decide whether to exclude it from the MST.
4. makeUnion - part of the UNION-FIND algorithm. if the two nodes are in the same set, make a union of their sets and replace each of their original sets to be the new union

#### GraphColoring Class

The GraphColoring Class splits the Graph Coloring algorithm into two important functions.

1. dfsCycle - this function recursively calls depth first search on the graph to color nodes and detect cycles. The nodes found within cycles are marked to be examined in the next function
2. createCycleList - this function pushes marked nodes into the correct cycle vector in the cycle adjacency list, which is a vector of cycles, each cycle represented by a vector of node names.

Once these functions have been run, a list of the cycles from the graph has been created to access later.