Vildan Hakanaj

Cois 3020H

Assignment #1

Winter 2019

Professor Brian Patrick

January 28, 2019

Project Description

The type of graph I used.

For this project I decided to go with the adjacency list implementation of the graph.
The reason was that I found the lists were a bit easier to edit and manipulate when it came to adding removing items from them. Basically the adjacency lists is implemented as follows
We have 3 classes for this algorithm

1. Vertex

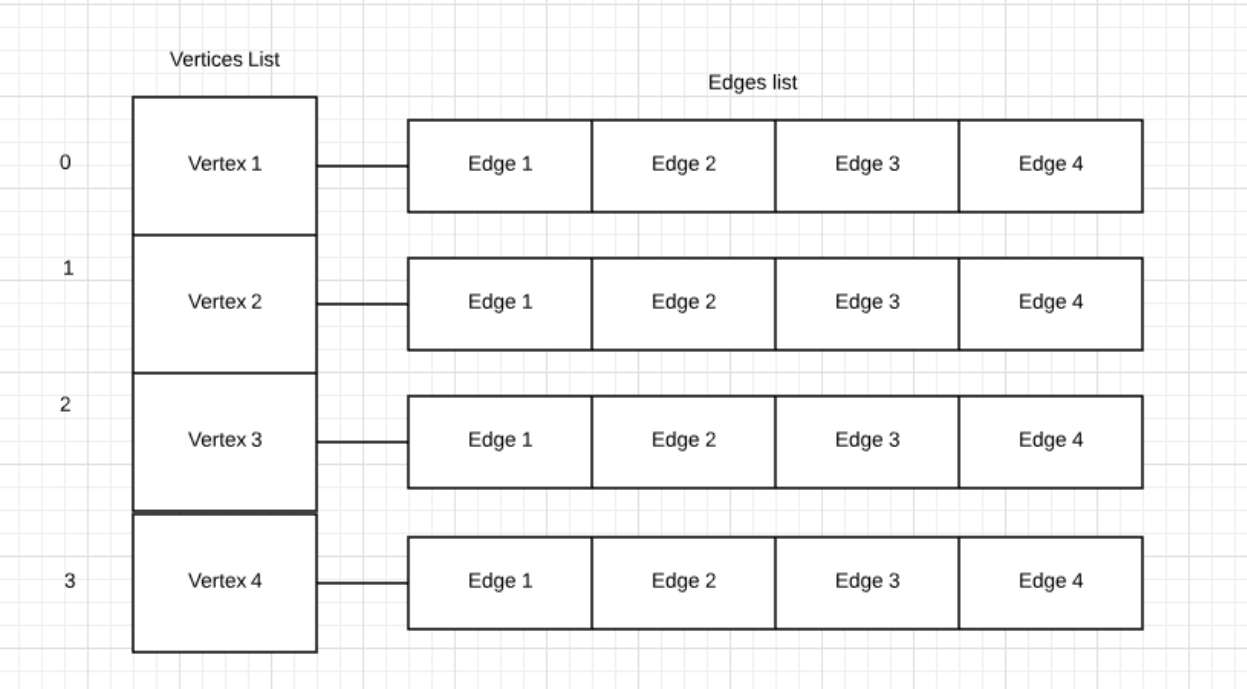
   The vertex class will contain all the information of a stations. Station name, and a list of edges that is pointing to other vertices.
   In addition to the base properties I have included some extra fields to help me with some of the operations.
   I have added fields like discovery, parent, lowlink which will come in handy when finding the critical paths.
   I could have utilized arrays to store those information only when doing the articulation points algorithm but i thought it would be a bit nicer and easier to read.
   The vertex class has three methods that assist in the main classes like subway and menu class

2. Edge

   The edge class is a representation of the link between two stations
   It contains the adjacent vertex and the color of the link

3. Graph (Subway)

   The Subway class is the main class that will execute, implement all the methods required from the project
   Its properties are
   A list of all vertices currently inserted in the graph.
   This class can insert station, insert link between two station, find articulation points, find shortest path , print the graph and more other helpful methods.



The Graph will contain a List of vertices. Each Vertex will contain a list of Edges that it has , as seen in the picture above.
The Edge will contain the adjacent vertex and the weight of the edge. In our case the weight is just a color of the line.
The vertex will also contain the vertex name.
Each vertex in the graph list contains a connection to the other vertices.

My implementations.

Articulation Points.

For this part of the assignments i used the Articulation points algorithm which is used in graph theory.
I used the depth first search to go through paths until there was no more nodes to go to or if all nodes were visited.
As you traverse you keep track of the time you discovered the nodes by giving them an integer of discovery time. Remember the vertex class is modified to contain the discovery time. After the depth first has reached the end and its time to go back, we keep track of the low link.
Low link basically is like a back edge. We set the current vertex low link to be the smallest low link between itself and the parent
.

```
// Check if there is a backedge from the adjacent vertex to something before
CurrentVertex.LowLink = Math.Min(CurrentVertex.LowLink, AdjVertex.LowLink);
```

After the nodes are assigned low links values and discovery times we then check if the vertex is a critical point. A vertex can be a Critical point in two cases.

```
//The vertex is root
if (CurrentVertex.Parent == null && children > 1)
{
    ArticulationPoints.Add(CurrentVertex);
}

//The vertex is not root
if (CurrentVertex.Parent != null && AdjVertex.LowLink >= CurrentVertex.Discovered)
{
    ArticulationPoints.Add(CurrentVertex);
}
```

1. The vertex is a root
    a. The vertex has no parent
    b. The vertex has more than one child
2. The vertex is not root
    a. The vertex discovery time is higher than its child low link
        i. This mean that there is no other way of reaching the child other than going through the current vertex which means if the nod is removed the graph gets broken therefore the vertex is a critical point

Shortest Path

To find the shortest path I basically used some ideas from the Dijkstra's algorithm for finding paths. Since Dijkstra's is typically used in a weighted graph and uses weights to travers pick the best path by selecting the smallest weighted link. In this case i decided to go with layers as i think its the best solution for this problem.
I used the breadth first search to go through the graph and visit the nodes.
This doesn't request to use recursion to restart the search if it gets interrupted as we need to start from a specific node and finish at the one we are trying to reach. In this method I keep track of the layers the node is in. Every time I insert something into the queue I give it a layer ++ and check if the nod is there and I also pass the parent from where the node came from. When the node is found I tell the user what layer was it found and return the node.
Then I use a print method to loop through the node.parent and print out the name of each node until forming the path.
Just like Dijkstra's I assume each layer is a weight of one and just add them as I move along. Ultimately this will always give me shortest path as it will be first path to reach that node.

Insert Station

When insert station is selected as an option the user will be asked to enter the name of the station.

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit
Enter a name for the station ==> A
```

The station can not be duplicate or the program will through an error.

```
Station A already exists

Press Any key to continue
```

After the user has inserted the Vertex correctly a success message will be prompted.

```
Just inserted station [ A ] into the graph

Press Any key to continue
```

In case you would want to remove the current graph you can just select the Clear Graph option on the menu and the graph will then be deleted and replaced with a new instance.

Insert Link

The insert link will ask the user to enter the appropriate names for the stations that they want to add a link and the color of the link. If the user enters invalid input or enters stations that don't exist or links that already exists the user will be prompted with the appropriate message.

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit

Enter a starting station: A

End a end station to : s

Enter a color for the line: w
Invalid input!
Press Any key to continue
```

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit

Enter a starting station: A

End a end station to : B

Enter a color for the line: red
```

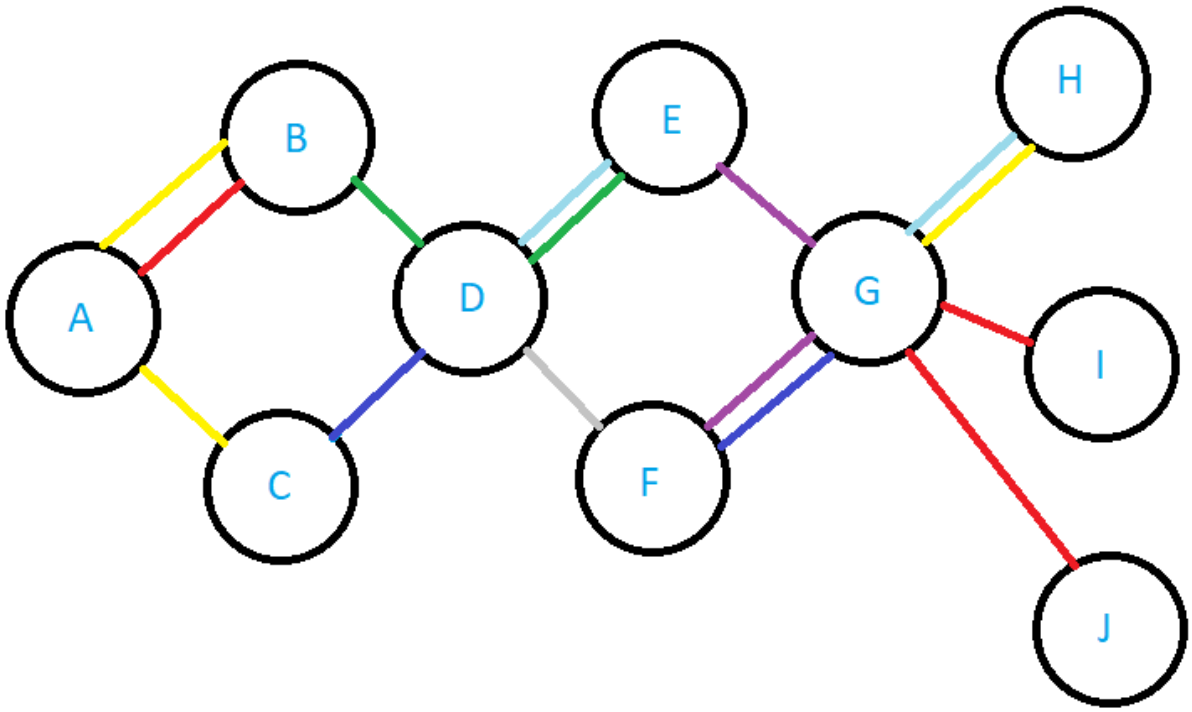
You can check if the link was successful you can use the print graph option which will print the graph and the links with the appropriate colors.

```
[ A ]---->[ B ]
[ B ]---->[ A ]
Press Any key to continue
```

2

Test Cases

Test case 1



Articulation points test.

This is the first test option. In this test case are two articulation points.
If we remove D or G the graph breaks in two parts making this two points critical.
If you run the critical point on the second test we get the following result.

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit

[ D ] Is an articulation point
[ G ] Is an articulation point
Press Any key to continue
```

If we can check in the graph and the result gotten from the program we see that it tell use the correct articulation points .
In this case the articulation points are in the middle of the graph so its not the case where the AP is the root.

Testing the short path:
Shortest Path cases : [A, J], [J, A], [D, J]

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit
Enter the the starting station: A
Enter the endstation: J
```

The user will be asked to enter the start station and the end station then the algorithm will start.
If the user enters the wrong station or there is no path the program will present an error message

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit
Enter the the starting station: A
Enter the endstation: P
One of the stations doesn't exists

No path was found
Press Any key to continue
```

Here the user entered a station that doesn't exists.

```
The shortest path from station [ A ] to station [ J ] is:

Take line

[ A ]---->[ B ]

Or You can take line

[ A ]---->[ B ]

Take line

[ B ]---->[ D ]

Take line

[ D ]---->[ E ]

Or You can take line

[ D ]---->[ E ]

Take line

[ E ]---->[ G ]

Take line

[ G ]---->[ J ]
Arrived
Press Any key to continue
```

Here we see that the user has entered the correct input and the algorithm ran successfully. The program recognizes if there is more than one line from one station to the other so it notifies the user the choices it has. Example you can take the Red line from A to B or you can take the line Yellow From A to B. When we find the path we notify the user that it has arrived.

Second path. Path case [J, A]

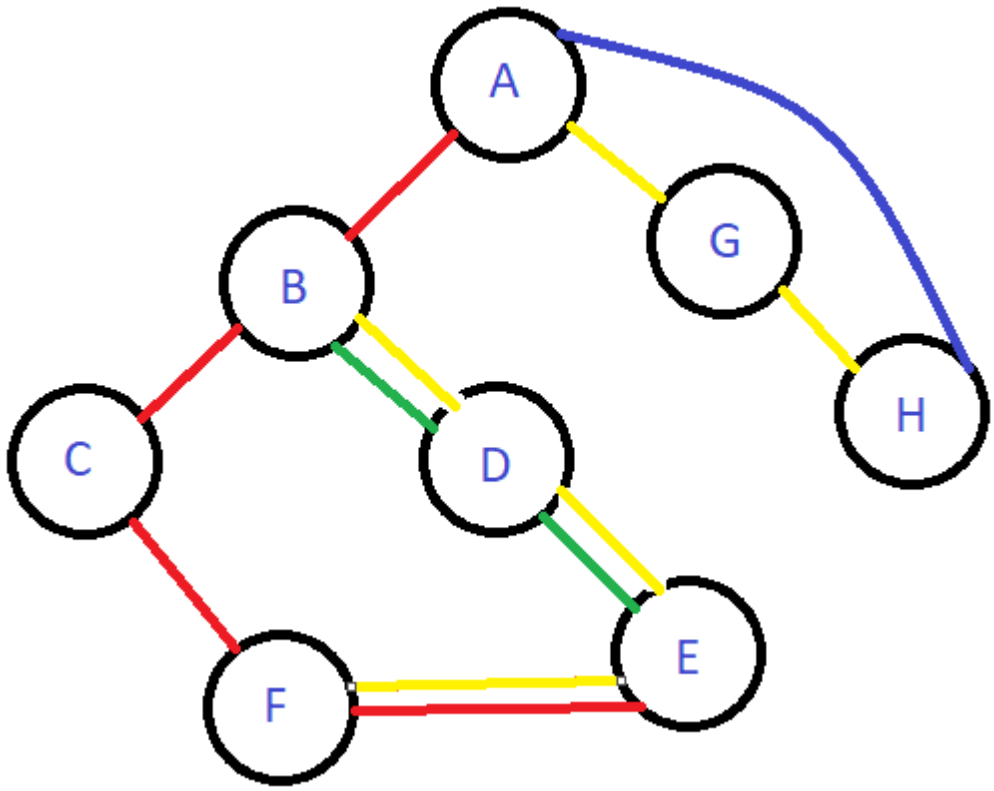
```
Enter the the starting station: J
Enter the endstation: A

The shortest path from station [ J ] to station [ A ] is:


Take line

[ J ]----->[ G ]

Take line

[ G ]----->[ E ]

Take line

[ E ]----->[ D ]

Or You can take line

[ E ]----->[ D ]

Take line

[ D ]----->[ B ]

Take line

[ B ]----->[ A ]

Or You can take line

[ B ]----->[ A ]
Arrived
Press Any key to continue
```

As seen in the picture the path goes from J to A and giving the alternative line if possible.

Third path Case: [D, J]

```
The shortest path from station [ D ] to station [ J ] is:


Take line

[ D ]----->[ E ]

Or You can take line

[ D ]----->[ E ]

Take line

[ E ]----->[ G ]

Take line

[ G ]----->[ J ]
Arrived
Press Any key to continue
```

Test Case 2



In this test case we see that we have 2 articulation points. One of them being A where now A is the root and will see if the program can recognize that as a articulation point. Also B is an AP both of this nodes remove will cause the graph to break.

The articulation point test.

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit

[ A ] Is an articulation point
[ B ] Is an articulation point
Press Any key to continue
```

As we can see the program recognized that A and B are articulation points.
G is excluded as a AP because H has a back edge pointing back to the root.

Shortest Path.

In this case we can really test if the program shows us the shortest path. As we can see the are two ways to reach to F station from A
Option 1: [A, B, C, F] --> This is the shortest path
Option 2: [A, B, D, E, F]

Example 1
Path from [A, F]

```
Enter the the starting station: A
Enter the endstation: F

The shortest path from station [ A ] to station [ F ] is:


Take line

[ A ]----->[ B ]

Take line

[ B ]----->[ C ]

Take line

[ C ]----->[ F ]
Arrived
Press Any key to continue
```

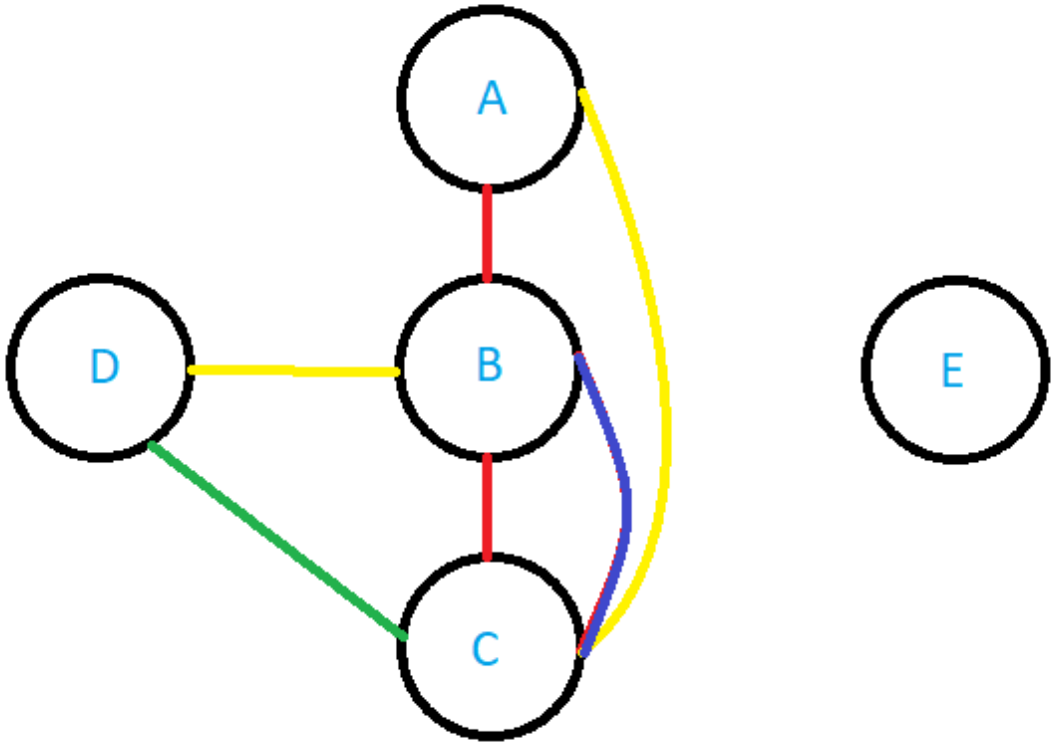
We can see that the program chooses the shortest path from the possible paths.

Example 2
Path from [A to H]

```
Enter the the starting station: A
Enter the endstation: H

The shortest path from station [ A ] to station [ H ] is:


Take line

[ A ]----->[ H ]
Arrived
Press Any key to continue
```

We can see that the shortest path tells use to go from A to H instead going through G.

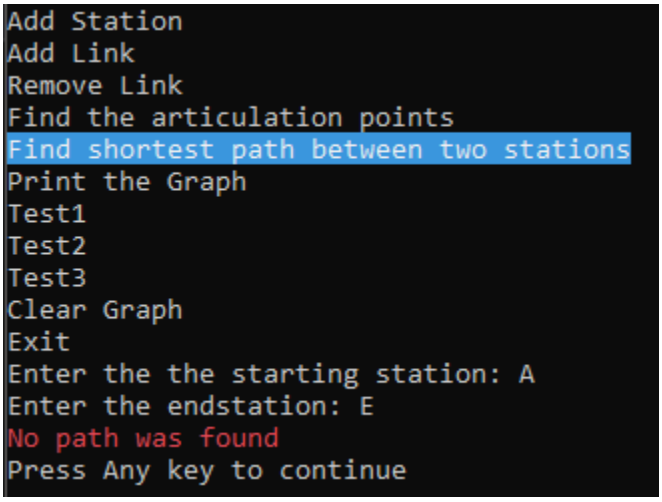
Test 3.



In this case we see that there is no Articulation points and also notice that there is no path going to E. Lets see how the program handles this.

```
Add Station
Add Link
Remove Link
Find the articulation points
Find shortest path between two stations
Print the Graph
Test1
Test2
Test3
Clear Graph
Exit

There is not articulation point in this graph
Press Any key to continue
```

In this case there is no articulation point so the program lets us now by throwing an error.



In this graph we have Vertex E that has no connection to any other vertex so that means that we cant reach that vertex from anywhere. The program throws this error to the user notifying us that there is no path to the specified vertex.