Delhi Technological University

Submission of report for 1st review of MTE innovative project for Data Structure(SE 201)

A locality management system that uses graph data structure to organize, manage and know the relationship of components(houses, shops etc) inside a locality.

Submitted by,
Shuvam Shiwakoti
2K19/SE/124
Group partner - N/A (working alone)

Topic

A locality management system that uses graph data structure to organize, manage and know the relationship of components(houses, shops etc) inside a locality.

Proposal Overview

As mentioned in the proposal of this project, the basic purpose of this project is to develop a locality management system that allows the user to add multiple components(namely houses and shops) into a locality where the locality will act as a graph and the different components act as vertices of the graph.

This project is being built with programming language C++. The data structures being used in the project are graph and arrays and vectors.
It is also to be noted that in the proposal it was mentioned that there would be use of linked list and stack data structures as well. However that is not achieved so far and use of these data structures will depend on whether their need arises but as of now all features of the program are successfully implemented using graphs, arrays and vectors.

Objectives

The objectives of this project are restated here so that they can be compared with the progress so far.
There are mainly 3 objectives for this project.

1. To understand and implement complex data structures like graph, linked lists and stack into a program and understand how they work together.
2. To understand the real world implementation of such data structures and how they can impact and help in improving our day to day activities.
3. To improve, sharpen a get a in depth knowledge about data structures in general and learn to use them properly and efficiently with C++ programming language.

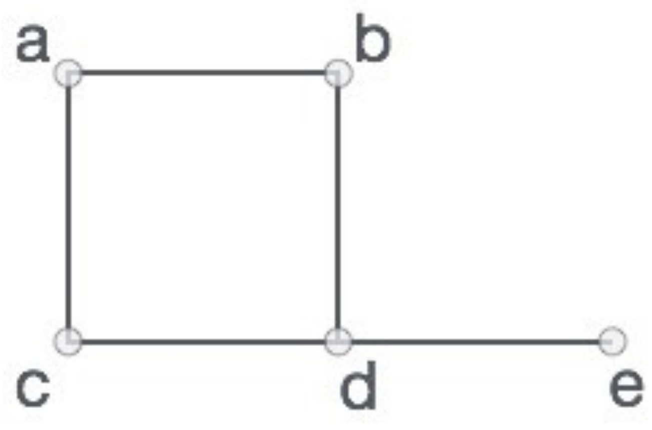
Progress Report

As mentioned above, the main objectives for the project involve a user interactive locality management system that uses graph data structure to define relation between different vertices in the graph.

Introduction to Graph data structure

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.
Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices.

In the diagram below,
V = {a, b, c, d, e}



E = {ab, ac, bd, cd, de}

An edge in a graph can either be **weighted** or **unweighted**.

- Weighted edges - Graph with edges that have a certain values or different values for other edges.
- Unweighted edges - Graph with edges that don't have separate values. All edges are considered as the same.

A **path** in a graph is the sequence of edges between the two vertices. In above example path from *a* to *e* can be '*abde*'.

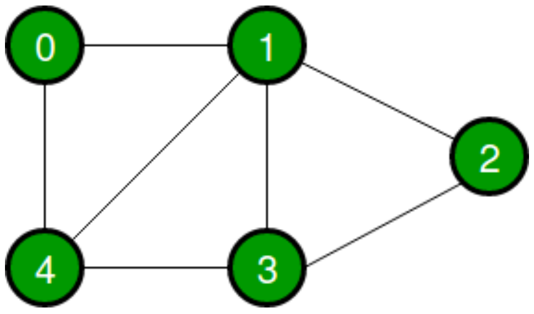
In a program a graph can be represented in mainly three forms :-

1. Adjacency matrix
2. Adjacency list
3. Compact list

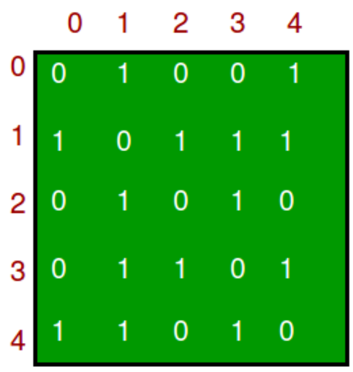
Here we shall discuss the Adjacency matrix representation of graph only as that is the one used in this project.

Adjacency matrix

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.



The adjacency matrix for the above example graph is:



$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0
\end{array}$$

On a weighted graph we can simply add the weight of each edge in their respective position in the matrix instead of adding 1.

Different algorithms and techniques have been developed for analyzing these graphs. Some of these are

1. BFS
2. DFS
3. Kruskal's Algorithm
4. Prim's Min cost spanning tree algorithm
5. Dijkstra's Shortest Path algorithm

For understanding and implementing graph I was able to successfully learn about the above algorithms and for this project I shall be using the Dijkstra's Shortest Path algorithm to find the shortest path between two vertices in the graph.

Dijkstra's Shortest Path algorithm

Given a graph and a source vertex, the Dijkstra's algorithm can give the shortest path from source to all other vertices.

Data Structures used :-

We maintain 3 arrays

1. sptSet[ ] - To track which vertices are visited.
2. dist[ ] - To track the minimum distance from the source to all other vertices.
3. parent[ ] - To store the preceding vertex of each vertex in the shortest path.

For the sake of simplicity, these arrays will be referred to as ss[ ] , d[ ] , p[ ]

Initial Steps

1. Initialize all items of the ss[ ] to false ie no vertices are visited.
2. Initialize all the items of d to INT_MAX[ ] except for the source vertex so that it is picked first.
3. Initialize all items of p[ ] to -1.

Repeating steps

1. Pick a vertex which is not included in ss[ ] and has the minimum distance value in d[ ]
2. Set the ss[ ] of that vertex to true (variable 'u')
3. Traverse through every single vertex(using variable 'v')

Here check

• If vertex is not included in ss[ ],

- If there exists a path between u & v ,
- If d[u] != INT_MAX ,
- If the total weight of path from source to v through u is smaller than current value of d[v](which is the previously calculated distance from source to v)

If all of these conditions hold true,
4. Change d[v] to d[v] + graph[u][v].
5. Make p[v] = u . This means we reach v vertex through u vertex in the shortest path from source.
6. After this all the steps are repeating and they are repeated for each vertex.
7. In each repetition a new vertex is chosen which is not in ss[ ] and has the smallest value in d[ ]

At the end d[ ] gives the shortest distance from source to all other vertices.
To get shortest path from source to all other, we use recursion over p[ ]

Printing path

To print path, we pass the p[ ] array and the vertex whose shortest path from source is to be found, to a function printPath( ) which uses recursion over p[ ] to get the path for the passed vertex.

1. Iterate over p[ ] (variable j)
2. Simply return if the p[j] value of the passed vertex is -1, ie path doesn't exist. This typically happens for finding the path form source to source as the source position in p[ ] always has value -1.
3. If other positive value exists, printPath( ) is called recursively by passing p[ ] and the value at position j in p[ ] . Here we are passing the p[ ] array and the parent of the current vertex. Use this recursion as a head recursion and in returning print the values of j . These values will give the path from source to the passed vertex.

Time complexity - Time Complexity of this implementation is O(V^2) while using the Adjacency matrix representation.

Program Brief

So far, in this program user can enter the number of *landmarks* and then also enter the relation between each *landmarks.*
As mentioned above, these *landmarks* are the vertices and the relation between these *landmarks* is whether these *landmarks* are directly connected and if they are what is the distance between them (edge weight).
The user will be allowed to choose whether the landmarks are connected and the distance between the connected landmark.
Here the graph is a Non- directed graph ie if two *landmarks* are connected either can be visited from one. With that being said the distance form either to other is same and the user has to enter this distance value only once.

To store the edge values as an Adjacency matrix representation of graph a 2D vector array is used.
This to done to create 2D array dynamically by taking size input from the user.

Other important thing to be noted is that the 0 index elements are not considered for representing the graph.
This holds true for all other arrays in the program as well.
This is simply done for the shake of keeping the indexing simple and easy to work with.

After entering all the values the user will be presented with a menu which will enable the user to :-

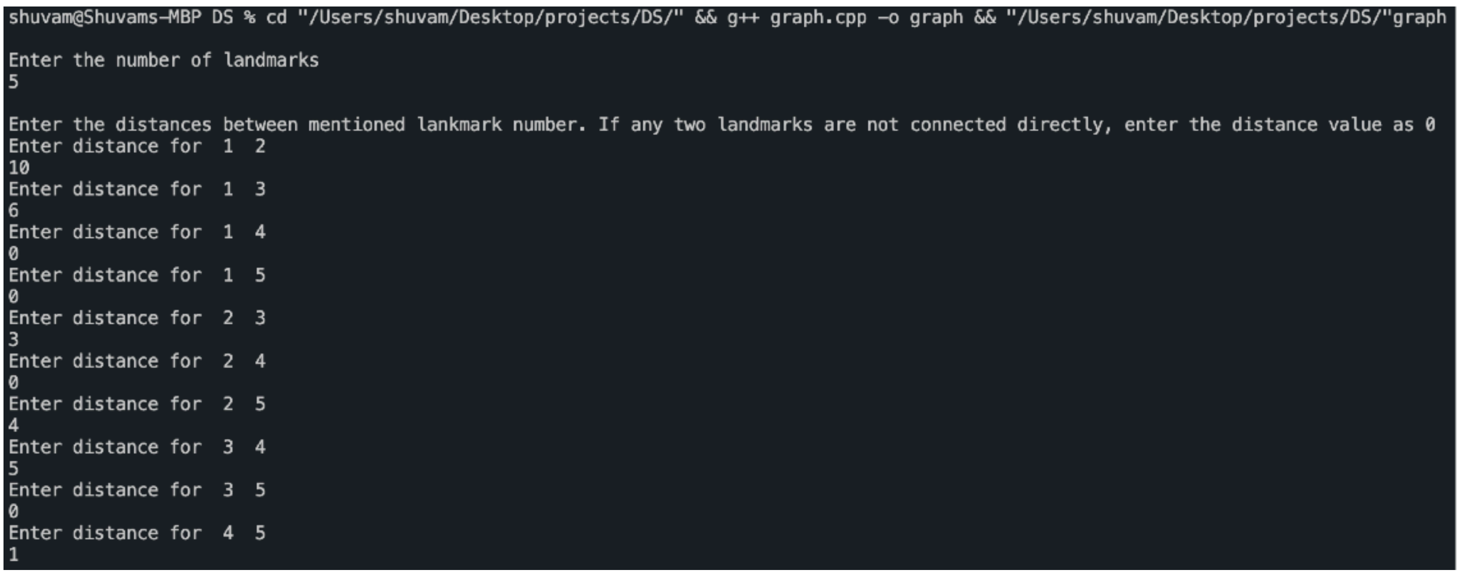
1. Get the shortest distance and path from a source *landmarks* to all other *landmarks*.
2. Get the shortest distance and path between any two *landmarks* only.
3. Get the shortest distance and path from each *landmark* or all other *landmarks.*

Since most of the time in this project was spent on learning the basics of graph data structure and implementing some algorithms, not a lot of work could be put on making the program interactive and organized.
However on the next phase of building the program some more features shall be added and the program will also be made much more interactive.

Test runs

Here are some snap shots of the test runs from the project completion so far.

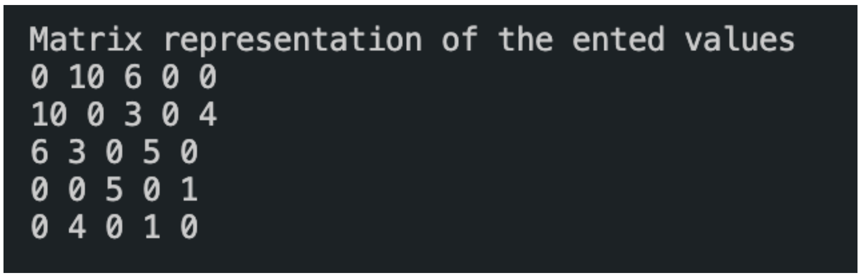
On first starting the program, the user is asked to enter the number of *landmarks* that are going to be included in the program.



```
shuvam@Shuvams-MBP DS % cd "/Users/shuvam/Desktop/projects/DS/" && g++ graph.cpp -o graph && "/Users/shuvam/Desktop/projects/DS/"graph

Enter the number of landmarks
5

Enter the distances between mentioned lankmark number. If any two landmarks are not connected directly, enter the distance value as 0
Enter distance for  1  2
10
Enter distance for  1  3
6
Enter distance for  1  4
0
Enter distance for  1  5
0
Enter distance for  2  3
3
Enter distance for  2  4
0
Enter distance for  2  5
4
Enter distance for  3  4
5
Enter distance for  3  5
0
Enter distance for  4  5
1
```

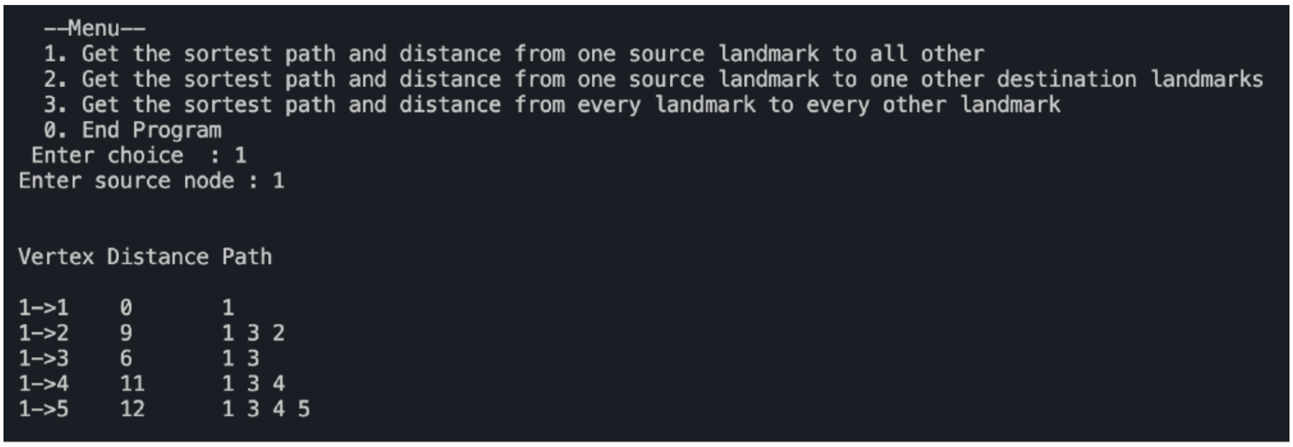
The *landmarks* are labeled from 1 to number entered by user.
On entering the number of *landmarks*, the user will be asked to define the relation between the *landmarks*.
Here the user can either enter a value 0 (saying no direct connection) or an integer/float value to declare it as the distance between the two *landmarks.*

After entering all the values the Adjacency matrix representation of the formed graph is presented.
This might not be of much use to the user but it is still done for reference and review purposes.



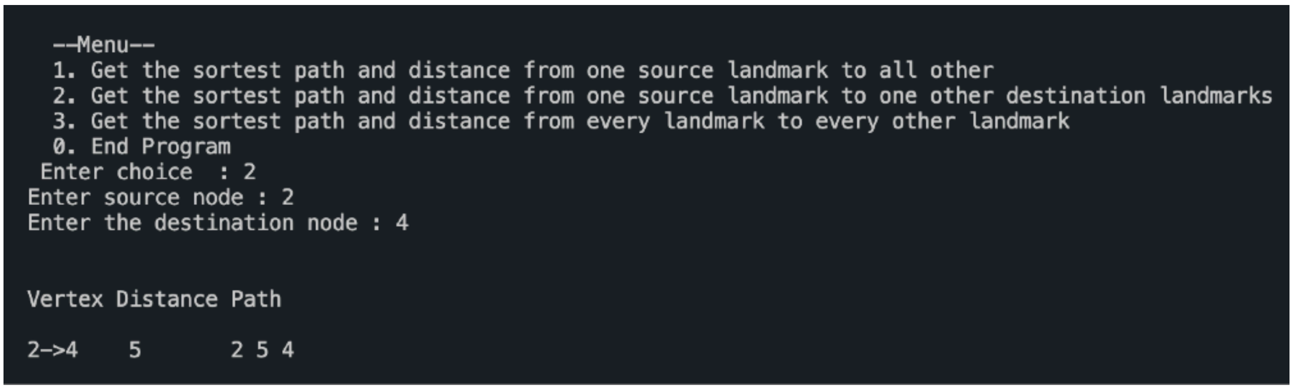
```
Matrix representation of the ented values
0 10 6 0 0
10 0 3 0 4
6 3 0 5 0
0 0 5 0 1
0 4 0 1 0
```

After displaying this matrix, the user is now presented with the menu in the program from where the user can choose from the services provided.
Here, if the user chooses the first option, the user is asked for a source *landmark* and as specified in the menu, the user receives the shortest distance and the path from the source *landmark* to all other *landmarks.*



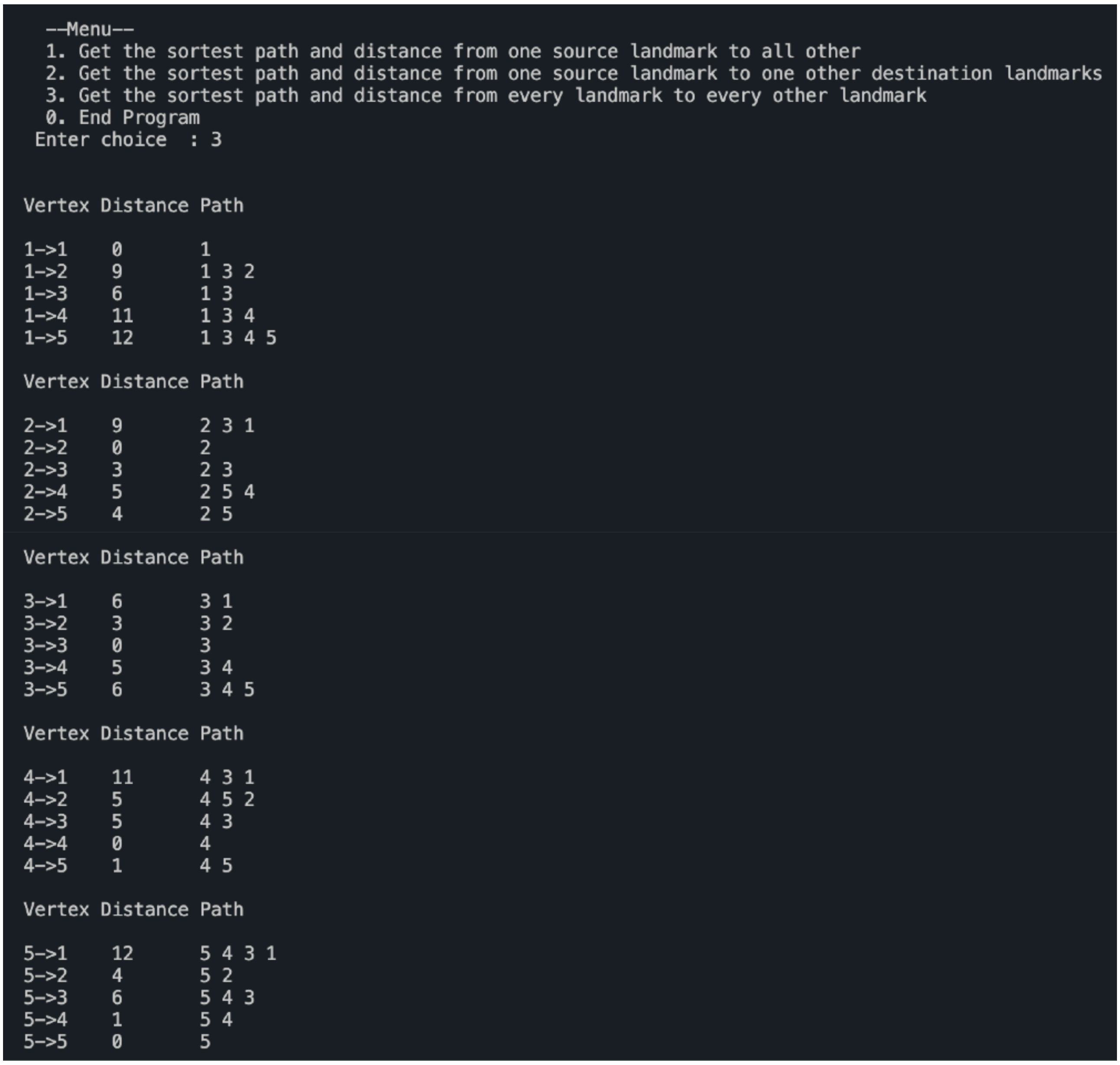
```
  --Menu--
  1. Get the sortest path and distance from one source landmark to all other
  2. Get the sortest path and distance from one source landmark to one other destination landmarks
  3. Get the sortest path and distance from every landmark to every other landmark
  0. End Program
 Enter choice  : 1
Enter source node : 1


Vertex Distance Path

1->1    0       1
1->2    9       1 3 2
1->3    6       1 3
1->4    11      1 3 4
1->5    12      1 3 4 5
```

On choosing the second option, the user is asked for a source *landmark* and a destination *landmark*.
Once those are entered, shortest distance and path only between the source landmark to destination landmark is presented to the user.



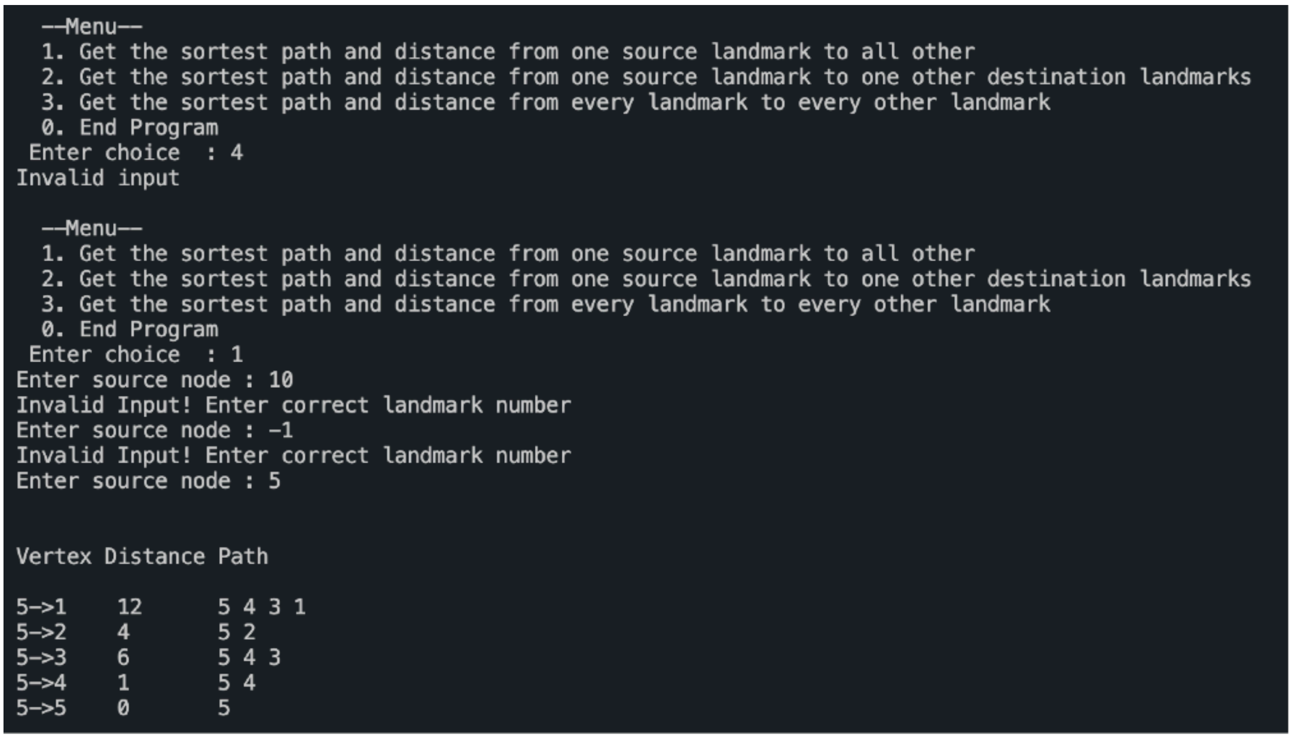
```
  --Menu--
  1. Get the sortest path and distance from one source landmark to all other
  2. Get the sortest path and distance from one source landmark to one other destination landmarks
  3. Get the sortest path and distance from every landmark to every other landmark
  0. End Program
 Enter choice  : 2
Enter source node : 2
Enter the destination node : 4


Vertex Distance Path

2->4    5       2 5 4
```

Finally on choosing the third option on the menu, the shortest distance and path from each vertex to all other vertex is displayed.



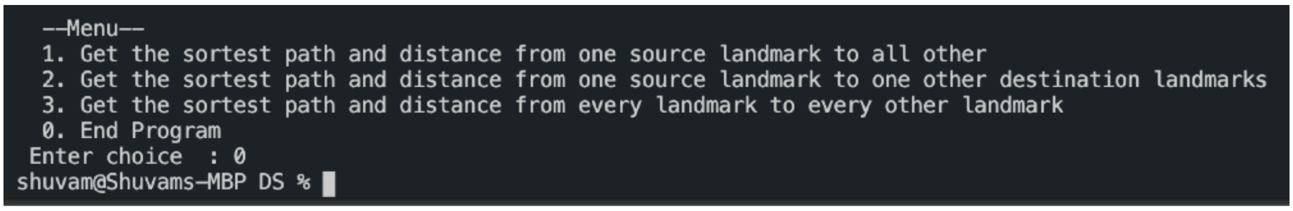
```
    --Menu--
    1. Get the sortest path and distance from one source landmark to all other
    2. Get the sortest path and distance from one source landmark to one other destination landmarks
    3. Get the sortest path and distance from every landmark to every other landmark
    0. End Program
   Enter choice  : 3


 Vertex Distance Path

 1->1    0       1
 1->2    9       1 3 2
 1->3    6       1 3
 1->4    11      1 3 4
 1->5    12      1 3 4 5

 Vertex Distance Path

 2->1    9       2 3 1
 2->2    0       2
 2->3    3       2 3
 2->4    5       2 5 4
 2->5    4       2 5

 Vertex Distance Path

 3->1    6       3 1
 3->2    3       3 2
 3->3    0       3
 3->4    5       3 4
 3->5    6       3 4 5

 Vertex Distance Path

 4->1    11      4 3 1
 4->2    5       4 5 2
 4->3    5       4 3
 4->4    0       4
 4->5    1       4 5

 Vertex Distance Path

 5->1    12      5 4 3 1
 5->2    4       5 2
 5->3    6       5 4 3
 5->4    1       5 4
 5->5    0       5
```

Different exceptions are also handled in the program. For instance, if a user gives an invalid input, an ‘invalid input’ message is displayed. Similarly, if the user enters a a non existing value of source or destination landmark, again similar message is displayed and the user is asked to enter the source and destination values again.



```
  --Menu--
  1. Get the sortest path and distance from one source landmark to all other
  2. Get the sortest path and distance from one source landmark to one other destination landmarks
  3. Get the sortest path and distance from every landmark to every other landmark
  0. End Program
 Enter choice  : 4
Invalid input

  --Menu--
  1. Get the sortest path and distance from one source landmark to all other
  2. Get the sortest path and distance from one source landmark to one other destination landmarks
  3. Get the sortest path and distance from every landmark to every other landmark
  0. End Program
 Enter choice  : 1
Enter source node : 10
Invalid Input! Enter correct landmark number
Enter source node : -1
Invalid Input! Enter correct landmark number
Enter source node : 5


Vertex Distance Path

5->1    12      5 4 3 1
5->2    4       5 2
5->3    6       5 4 3
5->4    1       5 4
5->5    0       5
```

To end/terminate the program, the user can simple enter 0 in the menu choice and the program will terminate normally.



```
  --Menu--
  1. Get the sortest path and distance from one source landmark to all other
  2. Get the sortest path and distance from one source landmark to one other destination landmarks
  3. Get the sortest path and distance from every landmark to every other landmark
  0. End Program
 Enter choice  : 0
shuvam@Shuvams-MBP DS % █
```

Objectives achieved

Since majority of the work on this project was dependent on learning graph data structure and implementing it in real world scenarios, given all the work that is done so far I can positively say that more than half of the work on this project has been completed.
I feel like I have developed a firm understanding of graph data structure and learned some very useful algorithms that are used in real world scenarios.

So with this it can be said that out of the three major objectives of this program all have been achieved to some extent. Now the next thing would be to build on the knowledge gained and add some more features to the program by implementing some more concepts and algorithms. Some of those will be discussed in the Project next phase section of this report.

Project next phase

Some of the work that will be done in the next phase of the project is discussed here along will that objectives that shall be achieved in this phase.

The major work in the next phase of this project will be giving the user the ability to add more *landmarks* once the program has begun.
At the moment I am not really sure about how to implement this feature but I plan to learn more about graph data structure, the Adjacency list representation of graph and Standard Template Library in C++ in the next phase of the project and hopefully with the knowledge gained, I will be able to implement this feature.

One major drawback or simply put an issue with this program is with the second option in the menu where the user gets to enter the source and destination landmarks and the information regarding the shortest distance and path is displayed.
The issue here is even though we only want the path and distance between two vertices the program runs the Dijkstra's algorithm all over again, scans through every *landmark* to get the desired result.

This doesn't seem very efficient to me. On trying to tackle this issue I came across some information on the web where this distance between two vertices can be achieved with a simple BFS of the graph. This method seems slightly more effective than Dijkstra's algorithm in this particular situation.
However I was not able to explore much on this idea but again this becomes another work for the next phase of the project.
Here I will try to explore the graph data structure even more and try to make the program more effective.

Other than the technical aspects, I will also be woking on making the program more user interactive by enhancing the menus, possibly adding more menus and making the program self explanatory on the user's end.

References

The reference list for the project remains pretty much the same as mentioned in the proposal of this project however they are restated here as well.

Websites like stackoverflow.com , geeksforgeeks.org , www.tutorialspoint.com and platforms like Youtube have been a major contributor in learning about the graph data structure and implementing different algorithms on it.
This project also involves the use of Standard Template Library(STL) to implement vectors, which too I learned from the above mentioned resources.
I also referred to our recommended course book Data Structures with C by Seymour Lipschutz for understanding the basics of and getting started with graphs.
I am also be referring to the notes I created during our lectures and the presentation notes provided by our course instructor for SE 201 Mr. Sanjay Kumar for understanding and implementing other data structures used in this project like arrays.

Conclusion

This project brings great opportunities for me as I am learning and implementing some very interesting and important concepts in data structures.
It also gives me opportunities to explore the real world scenarios of data structures.
So far it has been a very interesting journey working on this project and I am very excited about the things I will learn and implement in the next phase.

Also as mentioned in the proposal, it is to be noted that, this project will be build with main purposes being learning and demonstrating the the use of data structure in real world scenarios but the actual implementation of this program in the real world might be limited as in real world scenarios a lot of factors like user interface, user experience, better defined and organized inputs/outputs have to be considered which might not be possible with my current skill level.
However, as mentioned above, this projects is bringing me great opportunities on learning and innovating. With that being said this project could be a great starting point and hopefully enough to be within the scope of the MTE innovative project for Data Structure.