An Interim Report submitted

on

**NAVIGATION SYSTEM**

for

DATA STRUCTURES & ALGORITHM (UCS406)

(Second Year)

Submitted by

MADHAV GOYAL(101610050)

MAHESH SINGLA (101610051)

MANSHI TOMAR(101610053)

Group: COE-13

Submitted to

COMPUTER SCIENCE & ENGINEERING DEPARTMENT



PATIALA, PUNJAB-147004

INDIA

(JAN-JUN 2018)

**ACKNOWLEDGEMENT**

It is a matter of great pleasure to acknowledge our debt to our revered teacher and Supervisor

Dr. Ashish Aggarwal, Assistant Professor, Computer Science & Engineering Department, Thapar Insitute of Engineering & Technology. It is because of his priceless intellectual guidance, innovative and constructive idea for having us complete independence, affectionate encouragement to put our desire and thought, which paved the way for the successful completion of this work. It is indeed our privilege to work under him.

Last, but not the least, we thank God for giving us strength to overcome difficulty, which crossed our way to be a horizon.

**TABLE OF CONTENTS**

# **S. No. Title Page No.**

1 PROBLEM FORMULATION 4

2 ANALYSIS OF PROBLEM 5

a) Algorithmic Technique

b) Code Elaboration

c) Psedo Code/Algorithm

3 RESULTS 9

4 CONCLUSION 10

5 REFERENCES 11

**1. PROBLEM FORMULATION**

The main objective of this project is to implement an algorithm to show the working of navigation system. A map is shown in the front-end from which we will choose a source and a destination.

In this we can choose a source but we have a choice for destination , either we can choose destination or we can choose category of the destination.(eg:-source to destination :- dominos to Ajanta hospital and source to destination category :- dominos to hospital)

After choosing this we had to travel through many locations to reach our final destination. . There can be many paths from source to destination, but we choose the minimum optimal path to travel the destination so that it can take minimum distance to reach the destination. The distance travelled by the object travelling from the source to destination is calculated. The location which are in between the path should be shown. So that we can get the info about the whole path. After we pass each midway location distance is updated and final calculated distance with path is printed.

1. Firstly we take path from source to destination.

EASY DAY (Source) , ANAND HOSPITAL(Destination)

Path :-{ Ajanta Hospital, Mocha ,LifeplusHospital, Anand Hospital}

Distance is :- 19

1. Secondly we take path from source to category destination.

EASY DAY(Source), Hosiptal(Destination Category)

Path:-{Ajanta Hospital}

Distance is :- 5

Here you can the difference in the selection of destination by two different ways which is self-explanatory.

**2. ANALYSIS OF THE PROBLEM**

For this project, we’ll need to use a data structure that’s similar (but not identical) to a dijkstra algo.

We will use a graph data structure to search for the shortest path between our source and destination and source to category. We will calculate the shortest path and update it to show on front end.

The problem looks like:-

1. Treat all the locations as source which is updated after every move. Should pass once through a location.
2. We had to choose the best algorithm so that it can take infinite values in input for locations.
3. We had to choose the which path travelled by the object is the optimal shortest path.
4. Calculate distance such that each time it shows the minimum distance between adjacent two locations.
5. Wile travelling from source to category destination the object should always choose the destination which should have minimum distance from the locations available in that category.
6. Object should not pass a single location more than one time.

**2 (a) Algorithmic Technique :-**

To implement this project one should have proper understanding of following topics:-

* Structure
* Arrays
* Graphs
* Stacks
* Dikstra algorithm
* Concept of Asymptotic Notation

1. Big O Notation
2. Big Θ Notation
3. Big Ω Notation

**2(b) Code Elaboration :-**

1. In the project, adjacency matrix is used to implement the graph.
2. Nodes in the graph represent the locations. Information regarding each node is stored in an array called vertex of structure node datatype.
3. User will select a source(its current location). Binary search is used to match the index of node having its value equal to source given by user. Let say- vertex[i] has the value equal to source, then src=i. Command line arguments would be used for the same.
4. User will now select a destination location. Binary search is used to match the index of node having its value equal to destination given by user. Let say- vertex[j] has the value equal to destination, then des=j.
5. Now function dij(graph[NODES][NODES], vertex[NODES], src, des) is called.
6. In dij() function, three arrays are defined- minDist[NODES](to hold the minimum distance of each node from source), parent[NODES](to hold the index of parent node for each node), stree[NODES](to hold the record whether the node is visited or not).
7. All the values in minDist[NODES] are initialized with zero.
8. All the values in stree[NODES] are initialized with FALSE (to indicate no node is visited yet).
9. As each node is visited, its minimum distance is updated in minDist[NODES]. Dikstra is implemented to update the minDist[].
10. Say minimum distance of node 3 from source is 6 then minDist[3]=6.
11. Minimum distance corresponding to the given destination(stored in minDist[NODES]) is printed.
12. Now stack is used to print the complete path.

**Pseudocode for Dijkstra**

|  |  |
| --- | --- |
| 1: | **function** Dijkstra(Graph[NODES][NODES], vertex[NODES], src, des): |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | minDist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | parent[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := minDist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < minDist[v] | // Relax (u,v) |
| 13: | minDist[v] := alt |  |
| 14: | parent[v] := u |  |
| 15: | **return** previous[ ] |  |

**3. RESULTS**

1. When asource and corresponding destination is give, its minimum distance is printed.
2. Path, which the minimum distance is printed.

**4.CONCLUSION**

1. We implemented Graphs using adjacency matrix.
2. Graph implementation has O(|v|^2) time and space complexity.
3. We implemented Dijkstra’s Algorithm.
4. Overall implementation has O(|v|^2 +|v|+|v|) which is same as O(|v|^2).

**5. REFERENCES**

* <http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html>
* https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/