A
PROJECT
REPORT
ON

FINDING SHORTEST PATH AND IT'S DISTANCE BETWEEN TWO LOCATIONS

Submitted in partial fulfilment of the requirement for the IV semester

of

BACHELOR OF

TECHNOLOGY

IN

DESIGN AND ANALYSIS OF ALGORITHM

Submitted By:

R.PIRANESH (RA2011032010068)

S.SHRUTHI (RA2011032010050)

JAWAHAR BASKARAN (RA2011032010018)

Under the supervision of

Dr.M.Shobana

(Asst. Professor)

DEPARTMENT OF COMPUTING AND TECHNOLOGY, SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY SESSION – 2022

18CSC204J- Design and Analysis of Algorithms

Mini Project

Record Work

Registration numbers: RA2011032010068, RA2011032010050, RA2011032010018

Names: R.Piranesh, S.Shruthi, Jawahar Baskaran.

Semester: 4th

Department: CTECH

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR -603 203

KANCHEEPURAM DISTRICT

BONAFIDE CERTIFICATE

Register No_____

Certified to be the bonafide record of work done by_____of _____

_____ , B.Tech Degree course in the Practical ***18CSC204J –***

DESIGN AND ANALYSIS OF ALGORITHMS *in* ***SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, Kattankulathur*** *during the academic year 2022-2023*

Lab Incharge

Date: **Year Co-ordinator**

Submitted for University Examination held in _____ _, ***SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, Kattankulathur.***

Date:

Examiner-1

Examiner-2

CERTIFICATE

This is to certify that the project entitled "**FINDING SHORTEST PATH AND IT'S DISTANCE BETWEEN TWO LOCATIONS.**" carried out by under my supervision at Department of Computing and Technology, SRM Institute of Technology, Kattankulathur, Chennai.

The work is original, as it has not been submitted earlier either in part or full for any purpose before.

Dr. M.Shobana
(Asst. Professor)

DECLARATION

I, hereby declare that the work presented in this dissertation entitled "**FINDING SHORTEST PATH**

AND IT'S DISTANCE BETWEEN TWO LOCATIONS" has been done by me and my team, and this dissertation embodies my own work.

Approved By:
Dr. M.Shobana

ACKNOWLEDGEMENTS

I would like to thank **Dr. M.Shobana** (Asst. Professor) who has been a great inspiration and who have provided sufficient background knowledge and understanding of this subject.

Our humble prostration goes to her, for providing all the necessary resources and environment, which have aided me to complete this project successfully.

We thank Dr. M.Shobana (Asst. Professor) who have been the great inspiration and who have provided Sufficient background knowledge and understanding of this subject. Our humble prostration goes to her, for providing all the necessary resources and environment, which have aided me to complete this project successfully.

PREFACE

This project report gives us the brief working of shortest path finding between two places using Dijkstra's algorithm in C++ language. Dijkstra's algorithm is a real-life application/scenario-based project. This project report gives us the detailed understanding of the working of the code as well as which kind of algorithm along with why is it that we are going only with that approach. A basic layman can also understand this report as it is very simple and user friendly, the code itself is very simple to understand.

ABSTRACT

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes,[6] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Introduction:

What is Dijkstra's Algorithm?

Given a graph and a source vertex in the graph, find the shortest paths and shortest distance from the source to all vertices in the given graph.

Dijkstra's Algorithm finds the shortest path between a given node (which is called the "source node") and all other nodes in a graph. This algorithm **uses the weights of the edges to find the path that minimizes the total distance (weight) between the source node and all other nodes.**

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with a given source as a root. We maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.

IMPLEMENTATION

ALGORITHM:

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty. 2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While *sptSet* doesn't include all vertices

….**a)** Pick a vertex u which is not there in *sptSet* and has a minimum distance value.

….**b)** Include u to *sptSet*.

….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

CODE:

```
#include <bits/stdc++.h> using

namespace std;


// Number of vertices in the graph

#define V 9


// A utility function to find the vertex with minimum

// distance value, from the set of vertices not yet included //

in shortest path tree


int minDistance(int dist[], bool sptSet[], unordered_map<int, string> mp) {

// Initialize min value int min =

INT_MAX, min_index;

for (int i = 0; i < V; i++) if (sptSet[i] ==

false && dist[i] <= min) min = dist[i],

min_index = i; return min_index;

}


// Function to print shortest path from source to j using

// parent array void printPath(int parent[], int j,

unordered_map<int, string> mp) {

// Base Case : If j is source if

(parent[j] == -1) return;

printPath(parent, parent[j], mp);

cout << "->"<< mp[j] ;

}
```

```
// A utility function to print the constructed distance
// array void printSolution(int dist[], int n, int parent[], unordered_map<int,
string> mp)
{      int src, dest;      cout<<"Enter the
source:";     cin>>src; cout<<"Enter the
destination:"; cin>>dest;      if(src>=n||
dest>=n || src<0 || dest<0)
   {
      cout<<"Invalid input!";
return;
   }
   system("clear");     cout<<"\n\n--------Shortest Path Discovered!----------\n-------------------------
------------------\n";
cout<<"Source: "<<mp[src]<<endl<<"Destination: "<<mp[dest]<<"\nDistance: "<<dist[dest]<<"\nPath:
"<<mp[src]; printPath(parent,
dest, mp);
}

// Function that implements Dijkstra's single source
// shortest path algorithm for a graph represented using
// adjacency matrix representation void dijkstra(int graph[V][V], int
src, unordered_map<int, string> mp) {
// The output array. dist[i] will hold the shortest
// distance from src to i int
dist[V];

// sptSet[i] will true if vertex i is included / in
// shortest path tree or shortest distance from src to i
// is finalized bool
sptSet[V] = { false }; //
```

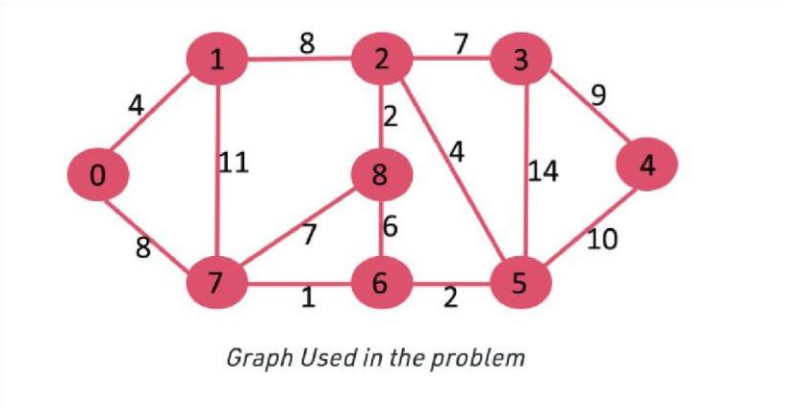
```
Parent array to store
shortest path tree int
parent[V] = { -1 };


// Initialize all distances as INFINITE
for (int i = 0; i < V; i++) dist[i] =
INT_MAX;


// Distance of source vertex from itself is always 0 dist[src]
= 0;


// Find shortest path for all vertices for (int
count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of
// vertices not yet processed. u is always equal to
// src in first iteration.
int u = minDistance(dist, sptSet, mp); //
Mark the picked vertex as processed
sptSet[u] = true;

// Update dist value of the adjacent vertices of the
// picked vertex.
for (int v = 0; v < V; v++)
// Update dist[v] only if is not in sptSet,
// there is an edge from u to v, and total
// weight of path from src to v through u is
// smaller than current value of dist[v] if
(!sptSet[v] && graph[u][v] && dist[u] +
graph[u][v] < dist[v]) { parent[v] = u;
dist[v] = dist[u] + graph[u][v];

}
```

```
}
// print the constructed distance array printSolution(dist,
V, parent, mp);
}

// Driver Code int
main()
{
// Let us create the example graph discussed above int
graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },
{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },
{ 0, 0, 7, 0, 9, 14, 14, 0, 0 },
{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },
{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },
{ 0, 0, 0, 14, 0, 2, 0, 1, 6 },
{ 8, 11, 0, 0, 0, 0, 1, 0, 7 }, { 0,
0, 2, 0, 0, 0, 6, 7, 0 } };
unordered_map<int, string> mp;
mp[8] = "shiva temple";
mp[7] = "arch-gate";     mp[6] =
"biotech";     mp[5] = "SRM
hospital";     mp[4] = "potheri";
mp[3] = "kc";     mp[1] = "tech-
park";     mp[0] = "ub";

for(auto x:mp)
{     cout<<x.first<<"
"<<x.second<<endl;
```

```
} dijkstra(graph, 0,mp); cout<<"\n-----------------------------------------\n"; int a;
cout<<"\n\n\n1. Explore more paths.\n2. Exit.\n\nPlease input your selection: ";
cin>>a; if(a==1)
dijkstra(graph, 0,mp);
if(a==2) return 0; else
cout<<"\nInvalid input!\n";
return 0;

}
```

INPUT:

For example, consider the below graph.

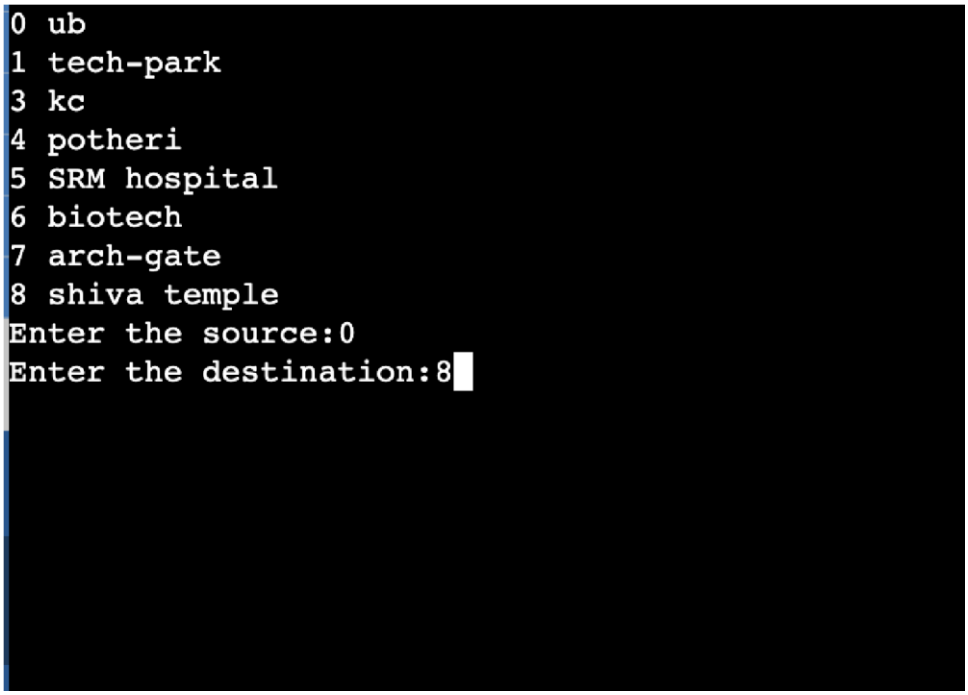


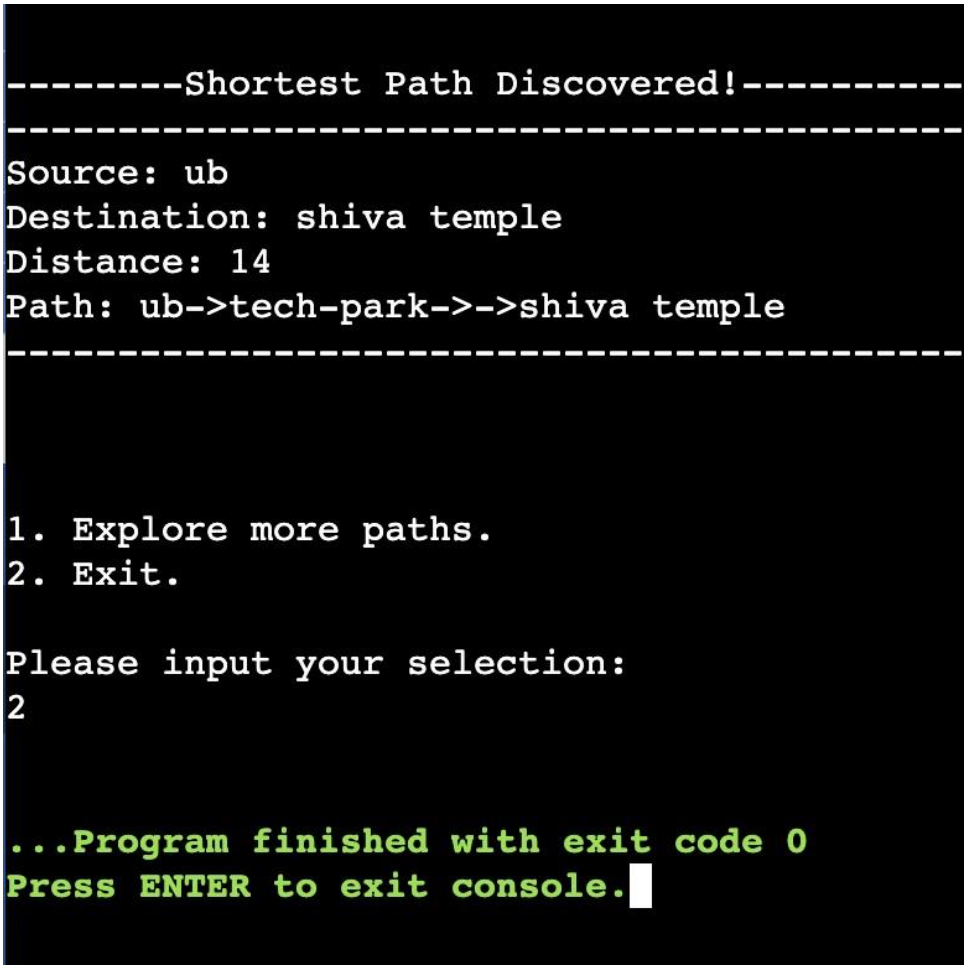
Graph Used in the problem

Source = 0

Destination = 8

Output:





ANALYSIS – TIME AND SPACE COMPLEXITY

Time complexity:

- Time for visiting all vertices =O(V)
- Time required for processing one vertex=O(V)
- Time required for visiting and processing all the vertices = O(V)*O(V) = O(V^2)
- Time taken for printing the path = O(2^V)
- Over all time complexity = O(2^V) **Space complexity:**

- Overall space complexity: O(V)*O(V) + O(V) = O(V^2)

Some more applications:

1) It is used in Google Maps

2) It is used in finding Shortest Path.

3) It is used in geographical Maps

4) To find locations of Map which refers to vertices of graph.

5) Distance between the location refers to edges.

6) It is used in IP routing to find Open shortest Path First.

7) It is used in the telephone network.

CONCLUSION:

Dijkstra's algorithm can be used to calculate the shortest path between a single node to all other nodes and a single source node to a single destination node by stopping the algorithm once the shortest distance is achieved for the destination node.

REFERENCES:

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm