PROJECT OF DISCRETE STRUCTURE

SUBMITTED BY:

PARTICIPANT 01: **SRISTI MITRA**

ROLL NO. : **2K19/CO/389 (A6)**

PARTICIPANT 02: **ZISHNENDU SARKER**

ROLL NO. : **2K19/CO/450 (A6)**

SUBMITTED TO:

AJAY KUMAR SIR

DISCRETE MATHEMATICS

TITLE OF THE PROJECT: Determining the shortest route in Google Maps by the concept of Graph Theory (Dijkstra Algorithm) of Discrete Structures.

ACKNOWLEDGMENT : I would like to express my deepest appreciation towards all the resources that has provided me the possibility to make progress in our report. A special gratitude I give to our Discrete Structure teacher Ajay Kumar Sir, whose stimulating suggestions and encouragement helped to coordinate in writing this project. Since saving time is one of the main priorities now-a-days , finding the shortest route will be most effective towards this issue. So, we planned to work on the algorithm to make that theory paper to find the shortest way using Dijkstra algorithm . We were inspired from our subject teacher who taught us graph theory in the class . According to the knowledge we gained in the class and from some online resources , we are able to finished this project paper . We are also grateful that the project enhance our knowledge towards graph theory and some related topic in Discrete Structure .

CERTIFICATE:

INDEX :

***ABSTRACT*:** In mathematics, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines).

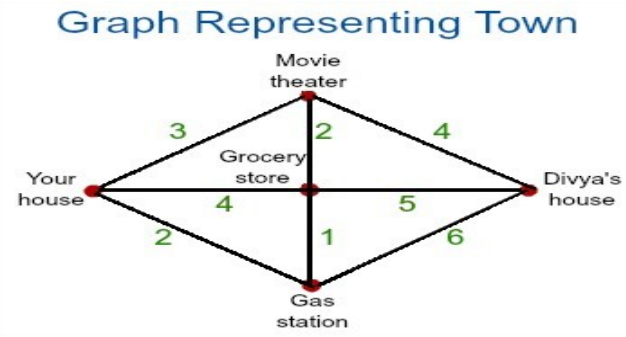
Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, for example, road networks. It was constructed by computer scientist Edsger W. Dijkstra in 1956 and published three years later. In Dijkstra's algorithm, we generate shortest path tree with given source as root. Here, it is certainly maintained two sets, one set contains the vertices included in 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 which is in the set and has a minimum distance from source.

We will generate that the core algorithm of Google Maps is Dijkstra's algorithm which is again related to graph theory.

***INTRODUCTION*:** The main aim of the project is to determine the shortest route in Google Maps by concept of graph theory.

Graph is a data structure which is used extensively in our real-life.

1. **Social Network**: Each user is represented as a node and all their activities, suggestion and friend list are represented as an edge between the nodes.
2. **Google Maps:** Various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find shortest path between two nodes.



3. **Recommendations on e-commerce websites:** The "Recommendations for you" section on various e-commerce websites uses graph theory to recommend items of similar type to user's choice.
4. Graph theory is a large field in mathematics and also used to study molecules in chemistry and physics.

CONCEPT USED:

GRAPH THEORY:

We can say that a graph will consists of two finite set which is let V and E. Here, each element of V is called a vertex (plural vertices). The elements of E which is called edges are unordered pairs of vertices. For instance, the set V can be {a,b,c,d,e,f,g,h}, and E might be {(a,d),(a,e), (b,c),(b,e),(b,g),(c,f),(d,f),(d,g),(g,h)}.

Now, we can say that together V and E are a graph G. It has natural visual representations.
Graphs are discrete structures consisting of vertices and edges that connect these vertices. There are different kinds of graphs, depending on whether edges have directions, whether multiple edges can connect the same pair of vertices, and whether loops are allowed. Problems in almost every conceivable discipline can be solved using graph models. We will give examples to illustrate how graphs are used as models in a variety of areas. For instance, we will show how graphs are used to represent the competition of different species in an ecological niche, how graphs are used to represent who influences whom in an organization, and how graphs are used to represent the outcomes of round-robin tournaments. We will describe how graphs can be used to model acquaintanceship between people, collaboration between researchers, telephone calls between telephone numbers, and links between websites. We will show how graphs can be used to model road maps and the assignment of jobs to employees of an organization. Using graph models, we can determine whether it is

possible to walk down all the streets in a city without going down a street twice, and we can find the number of colors needed to color the regions of a map. Graphs can be used to determine whether a circuit can be implemented on a planar circuit board. We can distinguish between two chemical compounds with the same molecular formula but different structures using graphs. We can determine whether two computers are connected by a communications link using graph models of computer networks. Graphs with weights assigned to their edges can be used to solve problems such as finding the shortest path between two cities in a transportation network. We can also use graphs to schedule exams and assign channels to television stations. This project paper will introduce the basic concepts of Dijkstra algorithm of graph theory . To solve the wide variety of problems that can be studied using graphs, we will introduce graph theory through Dijkstra's algorithm. We will also study the time complexity of these algorithms.

SET THEOREM:

A set is an unordered collection of objects, called elements or members of the set. A set is said to contain its elements. We write $a \in A$ to denote that a is an element of the set A. It is common for sets to be denoted using uppercase letters. Lowercase letters are usually used to denote elements of sets. There are several ways to describe a set. One way is to list all the members of a set, when this is possible. We use a notation where all members of the set are listed between braces. For example, the notation {a, b, c, d} represents the set with the four elements a, b, c, and d. This way of describing a set is known as the roster method.

The concept of a function is extremely important in discrete mathematics. A function assigns to each element of a first set exactly one element of a second set, where the two sets are not necessarily distinct. Functions play important roles throughout discrete mathematics. They are used to represent the computational complexity of algorithms, to study the size of sets, to count objects, and in a myriad of other ways. Useful structures such as sequences and strings are special types of functions. Furthermore, we will introduce some important types of sequences and we will show how to define the terms of a sequence using earlier terms.

TIME COMPLEXITY:

Time complexity is a concept in computer science that deals with the quantification of the amount of time taken by a set of code or algorithm to process or run as a function of the amount of input.
In other words, time complexity is essentially efficiency, or how long a program function takes to process a given input.
Time complexity is simply a measure of the time it takes for a function or expression to complete its task, as well as the name of the process to measure that time. It can be applied to almost any algorithm or function but is more useful for recursive functions.
Time complexity is expressed typically in the "big O notation," but there are other notations. This is a mathematical representation of the upper limit of the scaling factor for an algorithm and is written as O(Nn), with "N" being the number of inputs and "n" being the number of looping expressions.

We used the concept of Dijkstra Algorithm which we will discuss in the theory part .

Theory of Dijkstra Algorithm :

As our main objective of the project is to express the process how google maps find the shortest route, lets know details about Google Maps, Google Maps is a web mapping service developed by Google. It offers satellite imagery, street maps, and 360° panoramic views of streets, real-time traffic conditions, and route planning for traveling by foot, car, bicycle, or public transportation; and it works as Google Maps Update Schedules. The satellite data on Google Maps is typically between 1 to 3 years old.
Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. It says that a problem should be solved in stages by taking one step at a time and considering one input at a time to get an optimal solution. There are some predefined which we need to follow for an

optimal solution. So, it can be said that, the problems where choosing locally optimal also leads to global solution are best fit for Greedy. Dijkstra's algorithm is one of the algorithms that, we can use to find shortest distances or minimum costs depending on what is represented in a graph . It does not work on the graph with negative widths (or edges). This is used for single source shortest path problems . From a weighted graph we have to find shortest path from some starting vertex to all of the vertices. Finding shortest path is both minimization and optimization problem. Optimization method can be solved as greedy method. It is a **Greedy Algorithm** because it always relay on local optimum . Dijkstra algorithm gives a procedure for getting a optimal solution that is minimal result or we can say shortest path. Mostly this algorithm works for directed graph . But it can be work also for non-directed graph but for that we need to convert the non-directed graph to the directed graph.

The steps to this algorithm are as follows:

Step 1: Start at the ending vertex by marking it with a distance of 0, because it's 0 units from the end. Call this vertex your current vertex, and put a circle around it indicating as such. And let the distance of all the vertices as infinity initially .

Step 2: Identify all of the vertices that are connected to the current vertex with an edge. Calculate their distance to the end by adding the weight of the edge to the mark on the current vertex. Mark each of the vertices with their corresponding distance, but only change a vertex's mark if it's less than a previous mark. Each time we mark the starting vertex with a mark, keep track of the path that resulted in that mark.

Step 3: Label the current vertex as visited by putting an X over it. Once a vertex is visited, we won't look at it again.

Step 4: Relax all vertices adjacent to the current vertex .

Step 5: Of the vertices we just marked, find the one with the smallest mark, and make it our current vertex. Now, we can start again from step 2.

Step 6: Once we have labeled the beginning vertex as visited - stop. The distance of the shortest path is the mark of the starting vertex, and the shortest path is the path that resulted in that mark.

We can also start from the starting vertex but in that case we need to follow the shortest path mark from the end vertex to starting vertex to get the shortest path . The algorithm that is used to find the shortest path between our current position node and our destination position node is called Dijkstra's algorithm, and the article has a fairly simple explanation and graphics that illustrate the algorithm. Starting from the first node, we iterate through each neighbor and assign the neighbor a value, which represents the distance from our starting node to the new node. This distance is quantified by the weight of the edge that connects those 2 nodes. We then repeat this process for all the neighbors and record on each new node the value of the shortest distance from that node to the starting node.

There is an easy and short example of approach of Dijkstra algorithm :

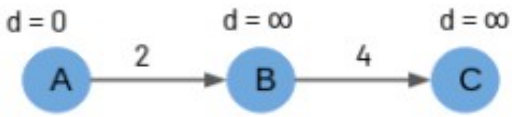


fig: 1 (a)

let, A is the staring vertex and we need to find the shortest path from A to B and C . Here, initially the distance of staring vertex A is 0 , B and C is ∞ . From A there is a direct path to B , so that we can update the distance of B from A is 2 unit , but there are no direct path to C , so we don't know the path and that's why this will remain ∞ .

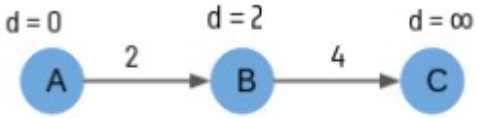


fig: 1 (b)

According to Dijkstra algorithm, we will choose the shortest path vertex B because the distance of the path of C is still ∞ . Because, Dijkstra always

says that ,when we select one of the shortest path , then check if there are any other short path available from that vertex to other vertex. If we check fig 1 (b) , we can see that shortest path with distance 2 unit is connected to A and B and From B to C it is connected to path 4 unit distance . But in fig 1 it starting vertex A has no direct path to C , that's why it counts ∞ for A. Now , as A and B is connected to distance 2 unit and B and C is connected to distance 4 unit , then the ∞ will change to 6 . That means there is a shortest path from A to C with distance 6 unit and it is not a direct path , it is going from A to C via B . That's how Dijkstra algorithm always select a vertex with the shortest path and then it will updated the shortest path to other vertices if possible and this updating is called as **Relaxation** .

How Relaxation works in Dijkstra Algorithm :

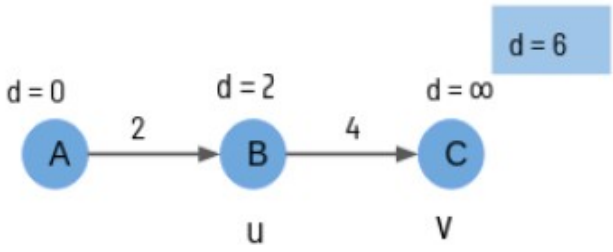


fig: 1 (c)

let , the vertex B as u and vertex C as v.
distance of u = 2 and distance of v =6 . [but initially it's infinity from starting vertex]
and distance 4 unit is the cost of an edge frame from u to v .
Relaxation method :

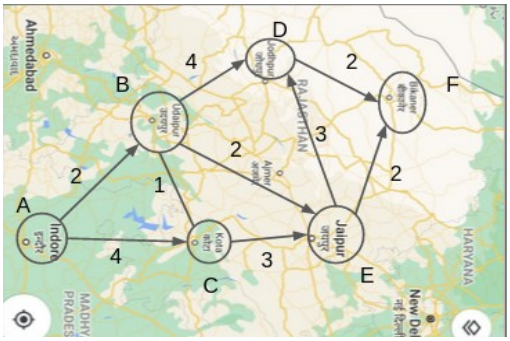
if (d[u] + c (u,v) < d [v])
d[v] = d[u] + c(u,v) ; …….(i)

Here , relaxation means is the distance of vertex u plus cost of an edge c(u,v) is less than distance of vertex v , then we will modify the distance of v to the distance of u plus cost of an edge c(u,v).
In fig 1(c) , the distance of u = 2 unit and cost of an edge c (u,v) = 4 unit , but the direct path to C from starting point is initially ∞ . So there , following equation (I),

if (2+4 < ∞)
d[v]=2+4
d[v]= 6

whenever we try to select a shortest path , always we try to relax other vertex's , that means we perform Relaxation .

Now with the example given below we will try to explain the relaxation concept and how the Dijkstra algorithm work :



Pic: Google maps

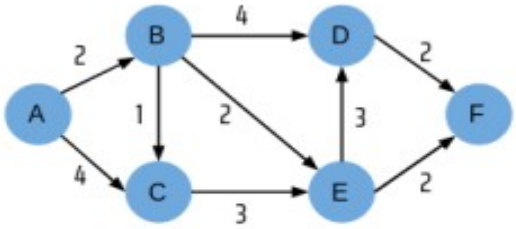


fig : 2 (a)

We create a sptset of shortest path tree that keeps track of vertices whose minimum distance from source is calculated and finalized . Let, A is the starting and F is the final vertex. So, we need to find the shortest path from A to F. Set the distance of source vertex as 0 and all other vertexes as infinity that is {0,∞,∞,∞,∞,∞} . To solve this, we have to select A as a source or starting vertex . So , it will be added in the sptset , that is the sptset will be {A} **.** We will also give the distances by considering just single edge. Label the distances above the vertices for initial state, from A to B distance is 2, and A to C distance is 4. But from A to D, E, F there is no direct edges. So, the distance will be infinity respect to A . This is the initial state of the graph.

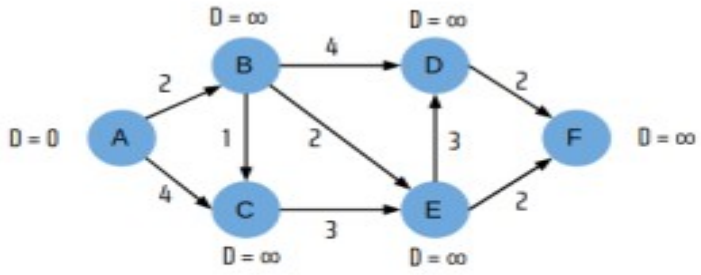


Fig : 2 (b)

1. Now, if we look at the adjacent vertices from starting vertex, we can see B and C are adjacent vertices and for A we have to relax B and C. So, first distances of A is O then, we follow the rules of Relaxing (equation 0), then distance of B from A=0+2 which is lesser than D = ∞ . So, we relax this distance and updates to 2. Again, from A to its adjacent vertex C the distance is = 0+4=4 which is lesser than infinity, so we have to relaxing the distance and we will get D=4 for C from A and rest of vertices D, E , F are not adjacent to A. So, there are no direct path from there, so they will remain infinity from A.

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A

Table : 2 (c)

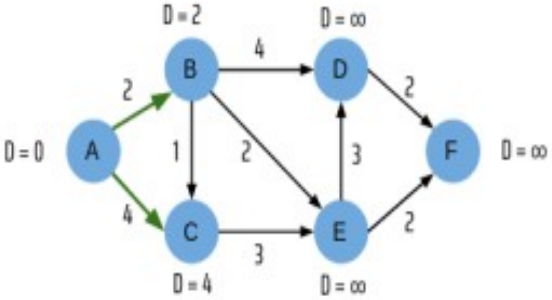


Fig : 2 (c)

2. After that we choose B as the next position or vertex from where we will start and also distance 2 will be fixed because that is the finalized shortest distance. So, B is our next current vertex and it will added to the sptset . So the sptset becomes {A,B} .

   B has 3 adjacent vertex C, D, E. Start from C, distance from B=distance from B + distance of C=2+1=3. There, distance of C from A is 4 but distance of C from B is 3 which is lesser than 4. So, relaxing the distance of C from A to B and distance of C will be 3.The distance of D from B will be =2+4=6 which is lesser than infinity, distance will be set as 6. The distance of E from B will be=2+2=4 which is lesser than infinity so the distance will be set as 4. Distance of F will remain as infinity according to A. Here the

distance of C from B is the shortest and finalized which is 3 unit. [by looking at the results and table]

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A

Table : 2(d)

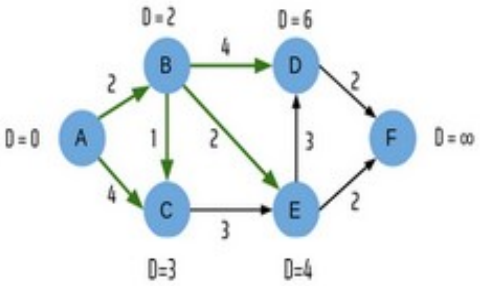


Fig : 2 (d)

3. Now C is the nearest vertex of B . So , C is the next current vertex and and sptset will become {A,B,C} .C has only 1 adjacent vertex which is E. After relaxing we can see the distance of E is from C is 3+3=6 unit. But the distance of E from B is 4 unit which one is shortest. So, it will be updated as 6, it will remain 4 unit, and others will remain same.

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A

Table : 2 (e)

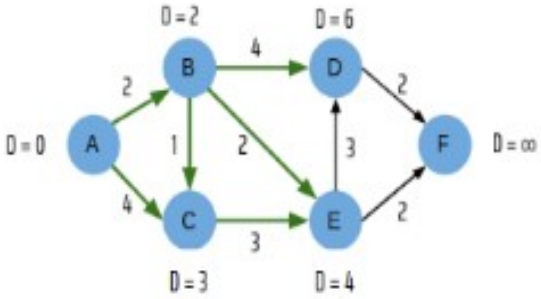


Fig : 2(e)

4. Now, here our goal is to reach F and F is the adjacent vertex of D and E. From our staring vertex A, we can see after relaxing and updating the distance, the distance of D is 6 but distance of E is 4 unit. Here 4 unit is lesser than 6 unit. So, we will take E as the next current vertex

from where will reach F. And then E will added to the sptset and it becomes {A,B,C,E}.

There are 2 adjacent vertices of E, these are D and F. Distance of D from B is 6 but distance of D from E is 4+3=7 which is not lesser than 6. So, the distance and path of D will remain same from B.

The next adjacent vertex is F which distance is initially infinity from A. but from E the distance of F will be 4+2=6, which is lesser than infinity. So, the distance will be set as 6 and the rest will remain same.

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E

Table : 2 (f)

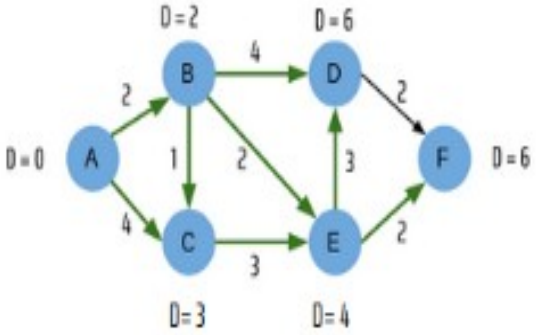


Fig : 2(f)

5. The next greatest value of the table is 6, the distance of D and from D its adjacent vertex is F. After that , vertex D adds in the sptset and sptset will become {A,B,C,E,D} . The distance of F from D is=6+2=8 which is not less than distance 6 unit from E. So, it will not be updated as 8 unit. It will remain 6 unit and path will from E. And rest value will remain same.

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E

Table : 2 (g)

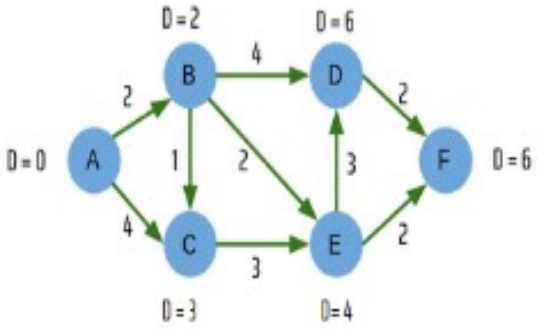


Fig : 2(g)

6. Our final vertex is F which is our goal position and from there are no further vertices. So, we will just keep all the value of distances and the path remain same as step 6 and in the sptset F will be added and sptset will become {A,B,C,E,D,F} .

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E
F	0^A	2^A	3^B	6^B	4^B	6^E

Table : 2 (h)

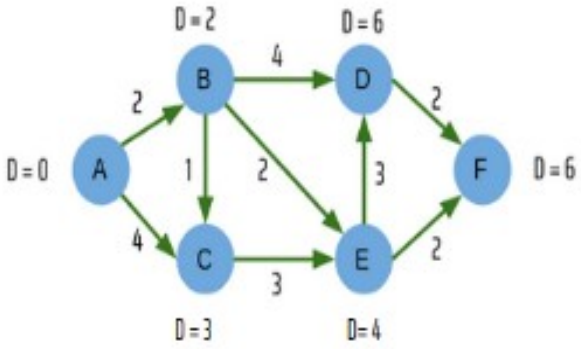


Fig : 2 (h)

7. Finally, we can see that in the table, our goal position in the graph F is finalized with the other vertices relaxed and updated. If we go from the last row we can see, F is connected with distance 6 with E, E is connected through the path of B with distances 4, B is commercial through the shortest path A with distance 2. So, our directed path from source on starting vertex A to F will be,

A—=>B—>E >F

So, the distance between both (A to F) vertices on nodes is= 2+2+2+6 unit. Which is the shortest distance among all the possibilities.

Q<=V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E
F	0^A	2^A	3^B	6^B	4^B	6^E

Table : 2(i)

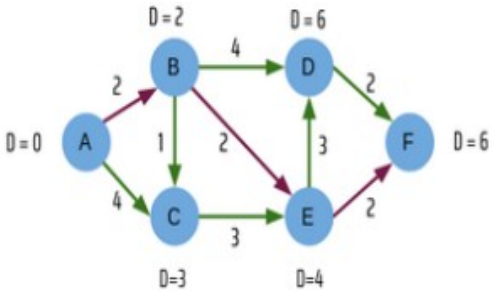
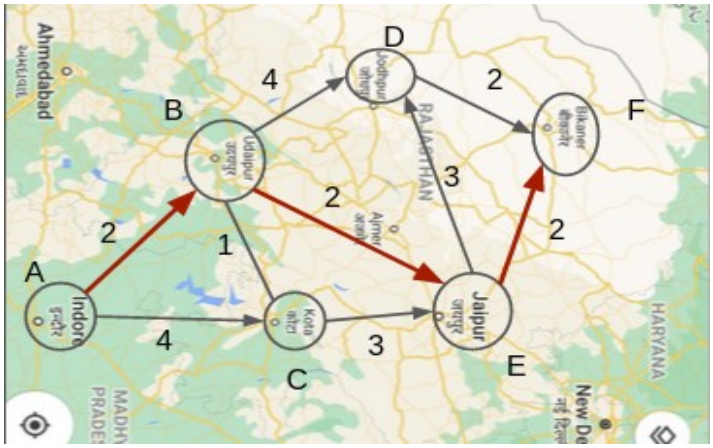


Fig : 2 (i)



So from here , we can say that , Dijkstra Algorithm gives us the shortest path from starting node to every single node. By following the previous vertex of any node, back up to the start, we can retrace the shortest path. So, to find the shortest path from A to F, we start from F and trace back to the starting node. It is a easy method to find the shortest path .

CODE:

In below there is the Code for Dijkstra algorithm for finding shortest path between starting vertex to other vertices . Here we show the code for out

example and graph . So , to find shortest path for any other graph , user need to input their graphs in main() of the program .

The code :-

```
#include <limits.h>
#include <stdio.h>
#define V 6

int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)

      if (sptSet[v] == false && dist[v] <= min)

         min = dist[v], min_index = v;

   return min_index;
}

void printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");

   for (int i = 0; i < V; i++)
   {
      printf("%d \t\t %d\n", i , dist[i]);
   }
}

char convert(int i)
{
   return (i%26) + 'A';
}
```

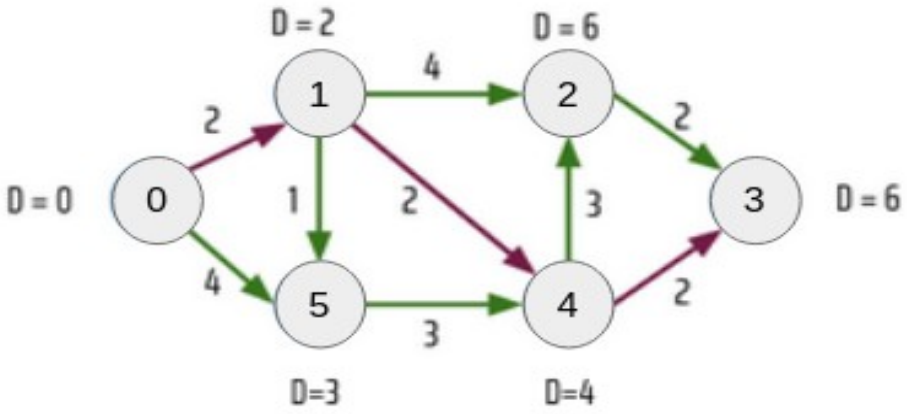
```
void dijkstra(int graph[V][V], int src)
{
   int dist[V];
   bool sptSet[V];

   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
      dist[src] = 0;

   for (int count = 0; count < V - 1; count++)
 {
      int u = minDistance(dist, sptSet);
      sptSet[u] = true;

      for (int v = 0; v < V; v++)

         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
            && dist[u] + graph[u][v] < dist[v])

            dist[v] = dist[u] + graph[u][v];
   }

   printSolution(dist);
}

int main()
{
   int graph[V][V] = { { 0, 2, 0, 0, 0, 4 },
                       { 2, 0, 4, 0, 2, 1 },
                       { 0, 4, 0, 2, 3, 0 },
                       { 0, 0, 0, 2, 2, 0 },
                       { 0, 2, 3, 2, 0, 3 },
                       { 4, 1, 0, 0, 3, 0 },
                     };

   dijkstra(graph, 0);
```

```
    return 0;
}
```

OUTPUT :

According to our graph we have vertices A, B, C, D, E, F , but in the code we are showing those vertices as integer value, from 0 to 5 where ,

A will be represented by 0
B will be represented by 1
D will be represented by 2
F will be represented by 3
E will be represented by 4
C will be represented by 5



Output from code :-

```
Vertex          Distance from Source
0               0
1               2
2               6
3               6
4               4
5               3

Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

ADVANTAGES OF DIJKSTRA ALGORITHM:

1. This algorithm has the ability to find the shortest way from one node to every other node within the same graph data structure.

2. Rather than finding the shortest way of two specific node , this algorithm works to find shortest path to every single reachable node without changing the graph

3. This algorithm runs until all the nodes have been visited

4. We can reuse the algorithm . We can can use this algorithm without running the algorithm again and again unless the graph data changed in any way .

5. If there is a change we can return the new graph and ensure the most updated shortest path of the data structure.

6. If there are any blocked path or undergoing works on any path , this algorithm will find the shortest path while avoiding any edges with larger weight .

DRAWBACK OF DIJKSTRA ALGORITHM:

Can't determine negative weighted edge, because distances cannot be considered as negative. It can work some cases, may not work in some. Lets look at the example that given below :

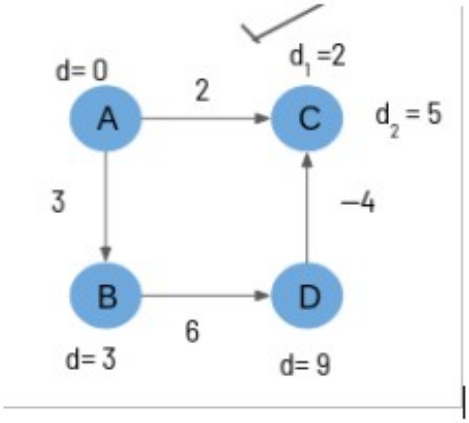


Fig : 3 (a)

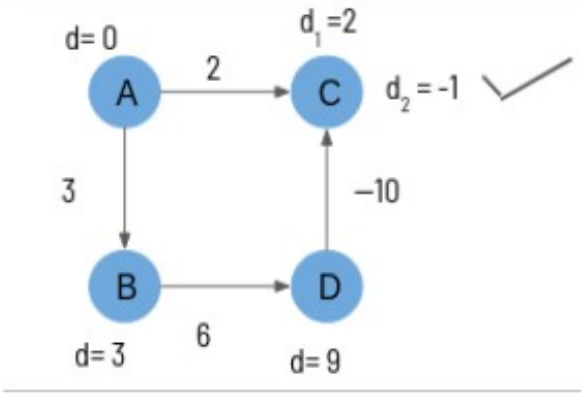


Fig : 3 (b)

Let, starting vertex is A and we need to find the shortest way for all other vertices , that is for B , C , D . but in two figures given above , for negative weighted edges , there will be a problem :

1. At fig : 3 (a) , it is visible that , from A to B and C has direct path . So , the distance will be 2 (d_1) , and 3 for C and B respectively . D is a adjacent vertex of only B , so the distance will be 3 + 6 = 9 . Now , C is also a adjacent vertex of D and is negatively weighted which is -4 and from D the distance is d_2= 5 but this is larger than d_1 = 2 . So , the distance will be 2 for C .

2. At fig : 3 (b) , it is almost same as fig : 3(a) . It is visible that , from A to B and C has direct path . So , the distance will be 2 (d_1) , and 3 for C and B respectively . D is a adjacent vertex of only B , so the distance will be 3

+ 6 = 9 . Now , C is also a adjacent vertex of D and is negatively weighted which is -10 and from D the distance is d_2= -1 but this is smaller than d_1 = 2 . So , the distance will be -1 for C . But we made a hurry for choosing the distance of C . But from graph of fig : 3(b) it is clear that , the distance of -1 will take more time and length is greater from direct A to C . But mathematically it is showing the shortest distance .

this type of problem can be happened by Dijkstra algorithm .

Time Complexity :

Dijkstra Algorithm find the shortest path to all the vertices .
Let, there are **n** number of vertices . And it is Relaxing when it is finding the shortest path .

So , n = |V|

And all the vertices relaxing its adjacent vertices . Let , one vertex is connected with all the vertices that is n number of vertices , so all n vertices are relaxing which will be the at most number can be relaxed. When all the vertices are connected , then it will be a complete graph , and for a complete graph,

it will take n * n times which is = n^2

So, n * n = |V| * |V|

here , 1st n is for the vertices and 2nd n is for the vertices are relaxed .
This is the wort-case time of Dijkstra Algorithm , where maximum time can be order of n^2
So,we can write it as $\theta (| V^2 |)$ or $\theta (| n^2 |)$.

CONCLUSION:

So, we can conclude here as Dijkstra algorithm being the basic and one of the most effective ways to find shortest route which inspired us to develop a project on this topic. It is saving time by avoiding unnecessary increment of roaming on streets and giving information how to reach the destination in minimum time as possible. We have mentioned the whole

process and the algorithm behind it that is also providing us with the knowledge how google maps or other navigation system works.

REFERENCES *:*

https://www.geeksforgeeks.org/mathematics-graph-theory-basics-set-1/
https://blogs.cornell.edu/info2040/2015/09/21/how-does-google-maps-work/
http://www.theijes.com/papers/vol8-issue10/Series-1/G0810014047.pdf
https://www.geeksforgeeks.org/greedy-algorithms/
https://www.youtube.com/watch?v=xkw49OfVDsk