DIJKSTRA'S ALGORITHM FOR DETERMINING SHORTEST PATH

Ram Lakhan Sah,
Lecturer,
Dept. of Mathematics,
RRM Campus (TU), Janakpurdham,
Dhanusha, Province No. 2.

Abstract: The shortest path problem exists in a variety of areas. There are several different algorithms that find a shortest path between two vertices in a weighted graph. A well-known shortest path algorithm is Dijsktra's, also called "label algorithm" which was proposed by a Dutch mathematician Edsger Dijkstra in 1959AD. In this paper, some basic concepts of graph theory and the famous Dijsktra's algorithm are presented along with an application.

***Index terms*- Weight, label, trail, path, algorithm.**

1. INTRODUCTION

The problems of shortest path occur in our real life problems as well as in different areas like optimizations, engineering and research, social and economics. The problem may involve minimization of cost, distance, length, traffics, air routes, railway networks, pipelines etc. They can be put in mathematical model of weighted graph and can be solved with the help of mathematical procedures as well as computer programming. There are different methods like: Breadth First Search [6], Depth First Search [6], Bidirectional Search [5], Bellman-Ford algorithm [7], Greedy algorithms etc of solving shortest path problem. An algorithm is a systematic procedure in form of sequence of steps to obtain the solution of problem. Dijkstra's Algorithm is one of the methods of determining shortest path in a weighted undirected graph which was developed by the Dutch mathematician Edsger Dijkstra [4] in 1959AD.

2. CONCEPTS OF GRAPH

The "Graph Theory" is a modern branch of Discrete Mathematics which was published in 1736 AD by a Swiss Mathematician Leonard Euler first time. So Euler is known as father of Graph Theory. He developed the theory to solve a popular problem known as "Konigsberge Bridge Problem" of Russia. This theory was systematically developed in 1936 AD after 200 years of Euler's pulication. Later, Kirchhoff and Cayley added subsequent discoveries in this area.

2.1 Definitions: ([6])

A *graph* G = (V, E) consists of a nonempty finite set V whose elements are called vertices (points or nodes) and a set E whose elements are called edges (arcs or lines) which connects two vertices in V. The set V is called *vertex-set* and set E is called *edge-set*. An edge e connecting vertices u and v in V is denoted by e = {u, v} or u-v.

The number of elements in V is called *order* of graph G and denoted by $|V|$. Similarly, the number of elements in E is called *size* of graph G and denoted by $|E|$.

2.2 Definitions: ([1])

Two or more edges e_1, e_2, e_3 are called *parallel or multiple edges* of graph G if they have same end points. An edge e of graph G, connecting a vertex v to itself is called a *loop*. Two or more edges are called *adjacent* if they have common end points. Similarly, two vertices u and v are called *adjacent* if there is an edge e = {u, v} connecting them.

A graph G is called a *simple graph* if it has no multiple edges and loops. And, G is called *multigraph* (multiple graph) if it has parallel edges but no loops. A graph G is called *pseudograph* if it has parallel edges and loops.

2.3 Definitions: ([3])

A *walk W*, connecting vertices u and v in a graph G is a finite alternating sequence of vertices and edges in the form $u = u_0, e_1, u_1, e_2, u_2, e_3, u_3, e_4, u_4, e_5, u_5, \dots, u_{n-1}, e_n, u_n = v$. The walk W is called *closed walk* is u = v. In a walk, an edge or a vertex may repeat.

The number of edges in a walk is called its *length*. A walk is called a *trial* if all edges in it are distinct and a trial is called a *circuit* if it is closed.

A walk W is called a *path* if all vertices in it are distinct. A path is called a *cycle* if it is closed. Obviously, all edges in a path are also distinct and only first and last vertices are repeated.

2.4 Definitions: ([3])

A graph G is called *weighted graph* if each edge e in G is assigned a definite non-negative real number denoted by $w(e)$, called weight of e. The non-negative real number may be length, cost, time, distance, fuel etc.

2.5 Definitions: ([3])

A graph G is said to be connected if there is path connecting any two distinct vertices in G. Otherwise, the graph G is said to be disconnected. If a graph G is not connected, then its connected sub-graphs are called components of G.

2.6 Definitions: ([2])

A circuit in a connected graph G is called Eulerian circuit if it contains all the edges of graph and the graph is called Eulerian graph if such an Eulerian circuit exists in G.

A path in a connected graph G is called Hamiltonian path if it contains all vertices in G (not necessarily all edges) and the graph is called Hamiltonian graph if such Hamiltonian path exists in G.

2.7 Definitions: ([6])

The degree of a vertex *v* in a graph G is the number of edges incident from *v* and denoted by *deg(v)*. A loop always contributes degree 2 to a vertex. The vertex *v* is called even or odd according as *deg(v)* is even or odd.

2.8 Theorems: ([8])

(i) A connected graph G is Eulerian iff each vertex in G is of even degree.

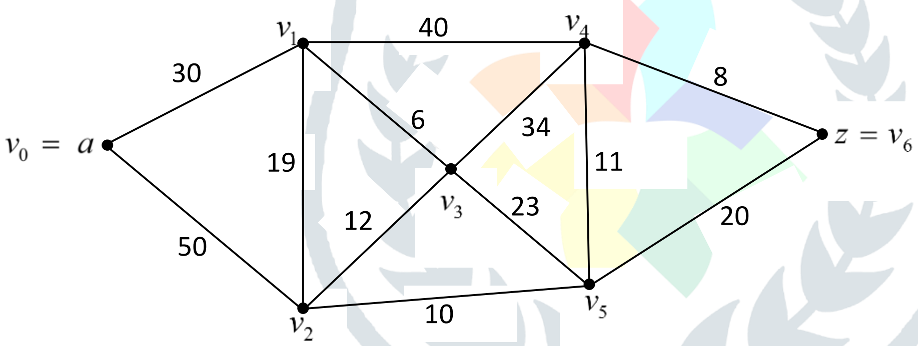
(ii) If G is a graph with n vertices such that for any two nonadjacent vertices u and v, deg(u) + deg(v)$\geq n$ then G is Hamiltonian.

(iii) If G is a graph with $p(\geq 3)$vertices such that $\deg(v) \geq {}^{p}/_{2}$ for every vertex v of G then G is Hamiltonian.

3. A SHORTEST PATH PROBLEM

3.1 Example:

Determine the shortest path from *a* to *z* from the following weighted graph:



Solution: *(We solve the problem using Dijkstra's algorithm which is stated in 3.2)*

STEP-1: Set vertex $a = v_0$ and assign to this vertex the permanent label 0. Assign every other vertex a temporary label ∞.

Vertex	$a = v_0$	v_1	v_2	v_3	v_4	v_5	$z = v_6$
Label	**0**	∞	∞	∞	∞	∞	∞

Which means that shortest path from a to a is of length 0. Here, $S = \{a\}$.

STEP-2: We denote label of vertex v_j by $L(v_j)$. The label of a is $L(a) = 0$. We compute label of each unlabeled adjacent vertex v_j of a by $L(v_j) = \min\{L(v_j), L(v_i) + w(v_i, v_j)\}$

In the graph vertices v_1 and v_2 are adjacent to labeled vertex a. Thus,

$$L(v_1) = \min\{L(v_1), L(a) + w(a, v_1)\} \qquad \text{(taking } v_i = v_0 = a\text{)}$$
$$= \min\{\infty, 0 + 30\} = 30$$
$$L(v_2) = \min\{L(v_2), L(a) + w(a, v_2)\}$$
$$= \min\{\infty, 0 + 50\} = 50$$

Vertex	$a = v_0$	v_1	v_2	v_3	v_4	v_5	$z = v_6$
Label	**0**	∞ **30**	∞ 50	∞ ∞	∞ ∞	∞ ∞	∞ ∞

The table shows the shortest distance from a to v_1 is 30 and from a to v_2 is 50. We choose smaller one, i.e. for v_1. Here, $S = \{a, v_1\}$.

STEP 3: We want to find label of adjacent vertices of v_1, which are v_2, v_3 and v_4. For this we carry out as above.

$$L(v_2) = \min\{L(v_2), L(v_1) + w(v_1, v_2)\}$$
$$= \min\{50, 30 + 19\} = 49$$
$$L(v_3) = \min\{L(v_3), L(v_1) + w(v_1, v_3)\}$$
$$= \min\{\infty, 30 + 6\} = 36$$
$$L(v_4) = \min\{L(v_4), L(v_1) + w(v_1, v_4)\}$$
$$= \min\{\infty, 30 + 40\} = 70$$

Vertex	$a = v_0$	v_1	v_2	v_3	v_4	v_5	$z = v_6$
Label	**0**	∞	∞	∞	∞	∞	∞
		30	50	∞	∞	∞	∞
			49	**36**	70	∞	∞

The table shows the shortest distance from a to v_2 is 49, from a to v_3 is 50 and from a to v_4 is 70. We choose smaller one, i.e. for v_3. Here, $S = \{a, v_1, v_3\}$.

Similarly, we find again new labels of v_2, v_4 and v_5 which are adjacent labels of v_3 and tabulate in following table.

Vertex	$a = v_0$	v_1	v_2	v_3	v_4	v_5	$z = v_6$
Label	**0**	∞	∞	∞	∞	∞	∞
		30	50	∞	∞	∞	∞
			49	**36**	70	∞	∞
			48		70	59	∞

Here, $S = \{a, v_1, v_3, v_2\}$.

Similarly, we level rest all the vertices and get the following table.

Vertex	$a = v_0$	v_1	v_2	v_3	v_4	v_5	$z = v_6$
Label	**0**	∞	∞	∞	∞	∞	∞
		30	50	∞	∞	∞	∞
			49	**36**	70	∞	∞
			48		70	59	∞
					70	**58**	∞
					69		78
							77

Here, $S = \{a, v_1, v_3, v_2, v_5, v_4, z\}$.

The shortest path in the given graph from a to z is $a = v_0 - v_1 - v_3 - v_2 - v_5 - v_4 - v_6$ = z of length 77.

3.2 Pseudo Code for Dijkstra's Algorithm: ([6])

Procedure: *Dijkstra (G: weighted connected simple graph, with all weights positive)*
{G has vertices $a = v_0, v_1, v_2, v_3, \ldots, v_n = z$ *and weights* $w(v_i, v_j)$
where $w(v_i, v_j) = \infty$ *if* $\{v_i, v_j\}$ *is not an edge in G}*
for $i := 1$ ***to*** n
$L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set}
while $z \notin S$
begin
u : = a vertex not in S with L(u) minimal
$S := S \cup \{u\}$
for *all vertices v not in S*
$if\, L(u) + w(u, v) < L(v)\, then\, L(v) := L(u) + w(u, v)$
{this adds a vertex to S with minimal label and updates the labels of vertices not in S}
end $\{L(z) = length\ of\ a\ shortest\ path from\ a\ to\ z\}$

3.3 Dijkstra's Algorithm in C source code:

```
/* Dijkstra's Algorithm in C */
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
```

```
#define N 6
int dijkstra(int cost[][N], int source, int target);
int main()
{
   int cost[N][N],i,j,w,ch,co;
   int source, target,x,y;
   printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");
   for(i=1;i< N;i++)
   for(j=1;j< N;j++)
   cost[i][j] = IN;
   for(x=1;x< N;x++)
   {
      for(y=x+1;y< N;y++)
      {
         printf("Enter the weight of the path between nodes %d and %d: ",x,y);
         scanf("%d",&w);
         cost [x][y] = cost[y][x] = w;
      }
      printf("\n");
   }
   printf("\nEnter the source:");
   scanf("%d", &source);
   printf("\nEnter the target");
   scanf("%d", &target);
   co = dijsktra(cost,source,target);
   printf("\nThe Shortest Path: %d",co);
}
int dijsktra(int cost[][N],int source,int target)
{
   int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
   char path[N];
   for(i=1;i< N;i++)
   {
      dist[i] = IN;
      prev[i] = -1;
   }
   start = source;
   selected[start]=1;
   dist[start] = 0;
   while(selected[target] ==0)
   {
      min = IN;
      m = 0;
      for(i=1;i< N;i++)
      {
         d = dist[start] +cost[start][i];
         if(d< dist[i]&&selected[i]==0)
         {
            dist[i] = d;
            prev[i] = start;
         }
         if(min>dist[i] && selected[i]==0)
         {
            min = dist[i];
            m = i;
         }
      }
      start = m;
      selected[start] = 1;
   }
   start = target;
   j = 0;
   while(start != -1)
   {
      path[j++] = start+65;
      start = prev[start];
   }
   path[j]='\0';
```

```
    strrev(path);
    printf("%s", path);
    return dist[target];
}
```

4. CONCLUSION

In this paper, we have first presented some related concepts of Graph theory and discussed the famous Dijkstra's Algorithm with its steps along with an example. The algorithm can be used to solve problems after converting in mathematical model of shortest distance in many areas in which such types of situation occurs.

REFERENCES

[1] Lin, W. 2009. Analysis of shortest path algorithm in ITS. *Technology and Economy in areas of Communications, 4.*

[2] Maharjan, H.B. and Sharma, L.N. 2008. An Introduction to Graph Theory. *Paluwa Prakashan, Kathmandu Nepal.*

[3] Maskey, S.M. 2008. First Course in Graph Theory. *Ratna Pustak Bhandar, Kathmandu, Nepal.*

[4] Mott, J.L. and Kandel, A. et al 2008. Discrete Mathematics for Computer Scientists and Mathematicians. *Prentice Hall of India*.

[5] Ping, L. 2008. An algorithm of shortest path based on Dijkstra algorithm. *Journal of value Engineering*, 5.

[6] Rosen, K.H. 2011. Discrete Mathematics and its Applications. *McGraw Hill Education Pvt Ltd, India.*

[7] Shu-Xi, W. 2012. The improved Dijkstra's shortest path algorithm and its application. *SciVersa ScienceDirect, 29: 1186-1190.*

[8] West, D.B. 2009. Introduction to Graph Theory. *Prentice Hall of India.*