ICT Tool: - 'C' Language Program for Dijkstra's Algorithm

P Kolhe , P P Kolhe ,M H Tharkar,R M Dharaskar
Dr BSKKV Dapoli, Dapoli, Maharashtra, India.

Abstract – In mathematics, Dijkstra's Shortest Path Algorithm is a popular algorithm for finding the shortest path between different nodes in a graph. Dijkstra's algorithm finds the solution for the single source shortest path problems only when all the edge-weights are non-negative on a weighted, directed graph. In other words, the graph is weighted and directed with the first two integers being the number of vertices and edges that must be followed by pairs of vertices having an edge between them.

In the era of Information Communication Technology (ICT) .The ICT programming technique, it is easier task. One of the very popular programs in C programming is Dijkstra's algorithm. This paper discuss Dijkstra's algorithm in C language, source code and methods with outputs. The source codes of program for Dijkstra's algorithm in C programming are to be compiled. Running them on Turbo C or available version and other platforms might require a few modifications to the code.

Index Terms – Dijkstra's algorithm, vertices and edges ICT, C Lang., Turbo C.

INTRODUCTION TO DIJKSTRA'S ALGORITHM

One of the very popular programs in C programming is Dijkstra's algorithm, whereas a program in C can carry out the operations with short, simple and understandable codes. Djikstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the *source*) to a destination. It turns out that one can find the shortest paths from a given source to *all* points in a graph in the same time, hence this problem is sometimes called the single-source shortest paths problem.

DIJKSTRA'S ALGORITHM IS DEFINED AS

Dijkstra's algorithm finds the length of an *optimal* path between two vertices in a graph. (*Optimal* can mean *shortest* or *cheapest* or *fastest* or optimal in some other sense: it depends on how you choose to label the edges of the graph.) The algorithm works as follows:

1. Choose the source vertex
2. Define a set S of vertices, and initialise it to emptyset. As the algorithm progresses, the set S will store those vertices to which a shortest path has been found.
3. Label the source vertex with 0, and insert it into S.
4. Consider each vertex not in S connected by an edge from the newly inserted vertex. Label the vertex not in S with the label of the newly inserted vertex + the length of the edge.
    - But if the vertex not in S was already labelled, its new label will be min(label of newly inserted vertex + length of edge, old label)
5. Pick a vertex not in S with the smallest label, and add it to S.
6. Repeat from step 4, until the destination vertex is in S or there are no labelled vertices not in S.

METHOD FOR DIJKSTRA'S ALGORITHM

Step 1 : Source node is initialized and can be indicated as filled circle.

Step 2 : Initial path cost to neighbouring nodes (adjacent nodes) or link cost is computed and these nodes are relabelled considering source node.

Step 3 : Examine all adjacent nodes and find smallest label, make it permanent.

Step 4 : The smallest label is now working node, then Step 2 and Step 3 are repeated till the destination node is reached.

C PROGRAM FOR DIJKSTRA'S ALGORITHM

```
/* C program to implement Dijkstra's algorithm */

#include<stdio.h>

#include<math.h>

#include<conio.h>

#define inf 9999

main()

{

        int a[size][size],i,j,n,v1,v2,lcost;

        int dij(int[i][j],int,int,int);

        printf("Enter the number of vertex : ");

        scanf("%d",&n);
```

```
		printf("Enter a weighted matrix(with weights) as input :n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			printf("Enter the value of a[%d][%d] : ",i,j);
			scanf("%d",&a[i][j]);
		}
	}
	printf("The entered matrix is:n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			printf("%dt",a[i][j]);
			printf("n");
	}
	printf("Enter starting vertex v");
	scanf("%d",&v1);
	printf("Enter ending vertex v");
	scanf("%d",&v2);
	/*Check for validity of input vertices*/
	if(v1<0||v1>n-1||v2<0||v2>n-1)
	{
		printf("!!!!!ERROR!!!!!n");
		printf("!!!!!Invalid vertex given!!!!!");
		return;
	}
	printf("Shortest path between v%d & v%d : ",v1,v2);
	lcost=dij(a,n,v1,v2);
	printf("Shortest cost between v%d & v%d : ",v1,v2);
	printf("%d",lcost);/*Print the shortest cost*/

getch();
}
int dij(int a[size][size],int n,int v1,int v2)
{
	int length[size],set[size],path[size],i,j,s,z,tmp,temp[size],c=0,f=0;
	s=v1;
	z=v2;
	int srch_min(int[],int[],int);
	for(i=0;i<n;i++)
		set[i]=0;
		for(i=0;i<n;i++)
		{
			if(a[s][i]==0)/*There is no direct edge between vertices s and i*/
			{
				length[i]=inf;
				path[i]=0;/*Empty path*/
			}
			else
			{
				length[i]=a[s][i];
				path[i]=s;/*s is immediate predecessor of i*/
			}
		}
		set[s]=1;/*s is included in the set*/
		length[s]=0;/*s is implicitly enumerated with length as 0*/
		while(set[z]!=1)/*Iteration will be considered until final vertex z belongs to s*/
		{
			j=srch_min(length,set,n);/*Select a vertex j with minimum label such that it is not included in the set[]*/
			set[j]=1;/*Vertex j is included in the set[]*/
			for(i=0;i<n;i++)
			{
				if(set[i]!=1)
```

```
                        {
                                if(a[i][j]!=0)
                                {

if(length[j]+a[i][j]<length[i])/*When exsisting label is not minimum only then replacement is done*/
                                        {

length[i]=length[j]+a[i][j];

path[i]=j;
                                        }
                                }
                        }
                }
                j=0;
                i=z;
                while(i!=s)
                {
                        tmp=path[i];
                        temp[j]=tmp;
                        i=tmp;
                        j++;
                        c++;
                }
                for(j=c-1;j>=0;j--)
                {
                        printf("v%d->",temp[j]);/*Print the shortest path*/
                        if(temp[j]==z)
                                f=1;
                }
                if(f!=1)
                        printf("v%d",z);
                printf("n");
        return length[z];
}
/*This function will return a vertex with minimum label such that it is not included in set[]*/
int srch_min(int length[],int set[],int n)
{
        int min,i,min_index;
        min=99999,min_index;
        for(i=1;i<n;i++)
        {
                if(set[i]!=1)
                {
                        if(length[i]<min)
                        {
                                min=length[i];
                                min_index=i;
                        }
                }
        }
        return min_index;
}
```

OUTPUT OF DIJKSTRA'S ALGORITHM:

```
Enter the number of vertex : 5
Enter a weighted matrix(with weights) as input :

Enter the value of a[0][0] : 0
Enter the value of a[0][1] : 2
Enter the value of a[0][2] : 1
Enter the value of a[0][3] : 5
Enter the value of a[0][4] : 0
Enter the value of a[1][0] : 2
Enter the value of a[1][1] : 0
Enter the value of a[1][2] : 0
Enter the value of a[1][3] : 1
Enter the value of a[1][4] : 3
Enter the value of a[2][0] : 1
Enter the value of a[2][1] : 0
Enter the value of a[2][2] : 0
Enter the value of a[2][3] : 1
Enter the value of a[2][4] : 0
Enter the value of a[3][0] : 5
Enter the value of a[3][1] : 1
Enter the value of a[3][2] : 1
Enter the value of a[3][3] : 0
```

```
Enter the value of a[3][4] : 2
Enter the value of a[4][0] : 0
Enter the value of a[4][1] : 3
Enter the value of a[4][2] : 0
Enter the value of a[4][3] : 2
Enter the value of a[4][4] : 0
The entered matrix is:

0 2 1 5 0
2 0 0 1 3
1 0 0 1 0
5 1 1 0 2
0 3 0 2 0

Enter starting vertex v0
Enter ending vertex v4
Shortest path between v0 & v4 : v0->v2->v3->v4
Shortest cost between v0 & v4 : 4
```

REFERENCES

[1] Dial, Robert B. (1969). "Algorithm 360: Shortest-path forest with topological ordering [H]". Communications of the ACM. 12 (11): 632–633..

[2] Fredman, Michael Lawrence; Tarjan, Robert E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346..

[3] Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the Association for Computing Machinery. 34 (3): 596–615.

[4] Zhan, F. Benjamin; Noon, Charles E. (February 1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks". Transportation Science. 32 (1): 65–73.

[5] Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, Jr., S. R.; Petry, R. M.; Seitz, R. N. (1957). Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems. Cleveland, Ohio: Case Institute of Technology.

[6] Knuth, D.E. (1977). "A Generalization of Dijkstra's Algorithm". Information Processing Letters. 6 (1): 1–5

[7] Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). "Faster Algorithms for the Shortest Path Problem". Journal of Association for

[8] Thorup, Mikkel (1999). "Undirected single-source shortest paths with positive integer weights in linear time". Journal of the ACM. **46** (3): 362–394.