Name- Abhinav Kumar

PRN- 21070126006

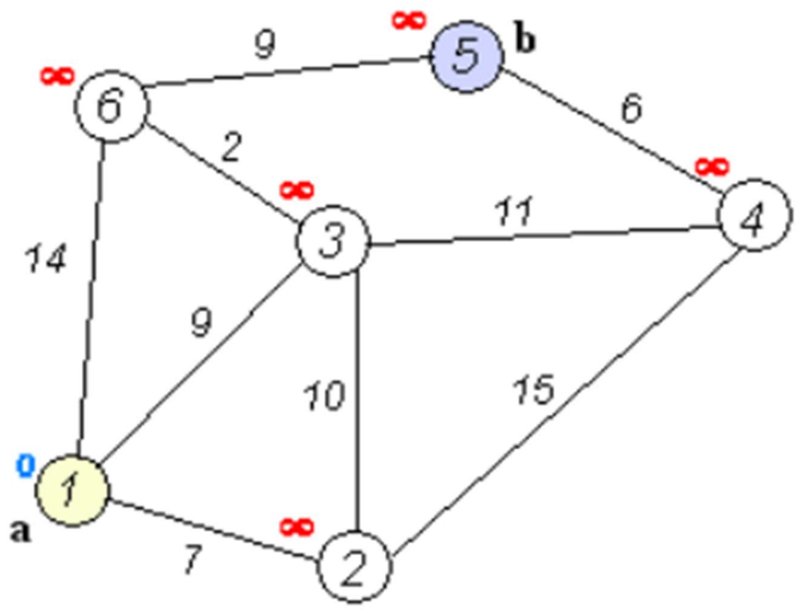
Branch- AIML- A1

Data Structural & Algorithms Experiential Learning Assignment

(Topic- Dijkstra's Algorithm)

Dijkstra Algorithm is additionally called single source most limited way calculation. It depends on ravenous procedure. The calculation keeps a rundown visited[ ] of vertices, whose briefest separation from the source is now known.

On the off chance that visited[1], approaches 1, the briefest distance of vertex I is as of now known. At first, visited[i] is set apart as, for source vertex. At each step, we mark visited[v] as 1. Vertex v is a vertex at most limited separation from the source vertex. At each step of the calculation, most brief distance of every vertex is put away in an exhibit distance[ ].



Algorithm:

1. Make cost network C[ ][ ] from nearness grid adj[ ][ ]. C[i][j] is the expense of going from vertex I to vertex j. In the event that there is no edge between vertices I and j, C[i][j] is vastness.

2. Array visited[ ] is introduced to nothing.

```
for(i=0;i<n;i++)
    visited[i]=0;
```

3. In the event that the vertex 0 is the source vertex, visited[0] is set apart as 1.

4. Make the distance network, by putting away the expense of vertices from vertex no. 0 to n-1 from the source vertex 0.

```
for(i=1;i<n;i++)
    distance[i]=cost[0][i];
```

At first, distance of source vertex is taken as 0. for example distance[0]=0;

5. for(i=1;i<n;i++)

- Pick a vertex w, to such an extent that distance[w] is least and visited[w] is 0. Mark visited[w] as 1.

- Recalculate the most limited distance of residual vertices from the source.

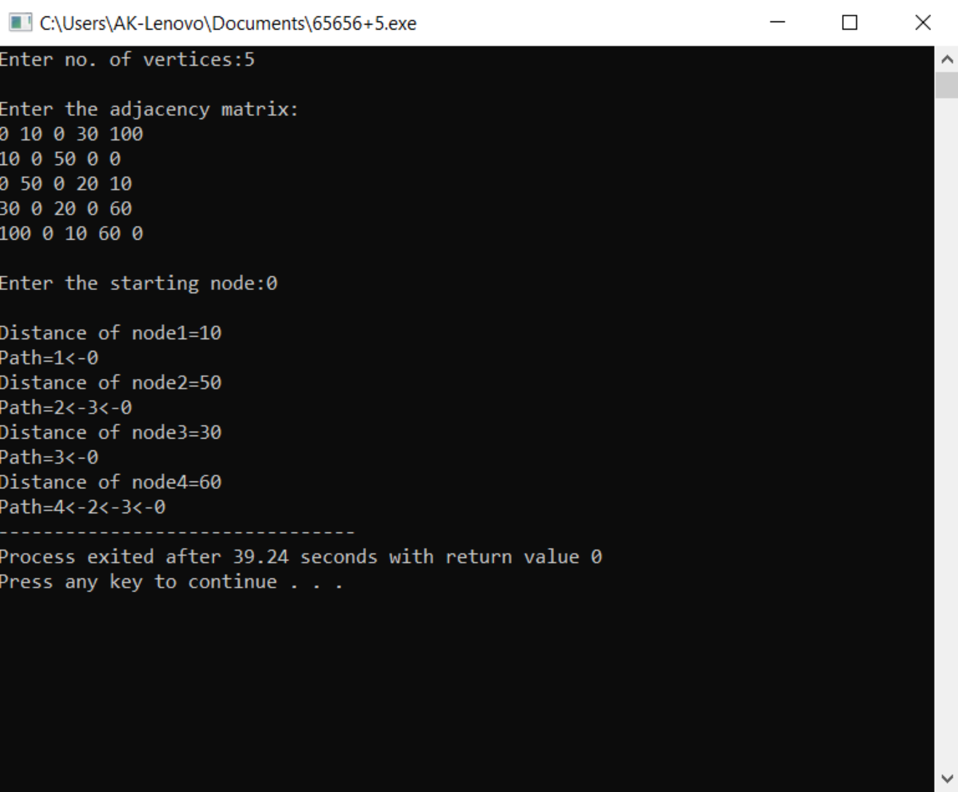
- Just, the vertices not set apart as 1 in exhibit visited[ ] ought to be considered for recalculation of distance. for example for every vertex v

```
if(visited[v]==0)
    distance[v]=min(distance[v],
    distance[w]+cost[w][v])
```

Time Complexity:

The program contains two nested loops each of which has a complexity of O(n). n is number of vertices. So the complexity of algorithm is $O(n^2)$.

```c
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```



```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
--------------------------------
Process exited after 39.24 seconds with return value 0
Press any key to continue . . .
```