| | |
|---|---|
| **Name** | Aditi Nilesh Bhutada |
| **UID no.** | 2021700009 |
| **Experiment No.** | 6 |

<table>
<tr><td>AIM:</td><td>Greedy-Approach single source shortest path- Dijkstra’s Algorithm</td></tr>
<tr><td colspan="2" align="center">PROGRAM</td></tr>
<tr><td>PROBLEM STATEMENT :</td><td>To find the shortest path from a source node</td></tr>
<tr><td>THEORY:</td><td>Dijkstra's Algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956. It is a single source shortest paths algorithm. It means that it finds the shortest paths from a single source vertex to all other vertices in a graph. It is a greedy algorithm and works for both directed and undirected, positively weighted graphs (a graph is called positively weighted if all of its edges have only positive weights).<br><br>In this algorithm each vertex will have two properties defined for it-<br><br>• Visited property:-<br>    • This property represents whether the vertex has been visited or not.<br>    • We are using this property so that we don't revisit a vertex.<br>    • A vertex is marked visited only after the shortest path to it has been found.<br>• Path property:-<br>    • This property stores the value of the current minimum path to the vertex. Current minimum path means the shortest way in which we have reached this vertex till now.<br>    • This property is updated whenever any neighbour of the vertex is visited.<br>    • The path property is important as it will store the final answer for each vertex.</td></tr>
<tr><td>ALGORITHM:</td><td>1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.<br><br>2. Array visited[ ] is initialized to zero.<br>    for(i=0;i<n;i++)<br>        visited[i]=0;<br><br>3. If the vertex 0 is the source vertex then visited[0] is marked as 1.<br><br>4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.<br>    for(i=1;i<n;i++)</td></tr>
</table>

distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

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

– Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

– Recalculate the shortest distance of remaining vertices from the source.

– Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

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

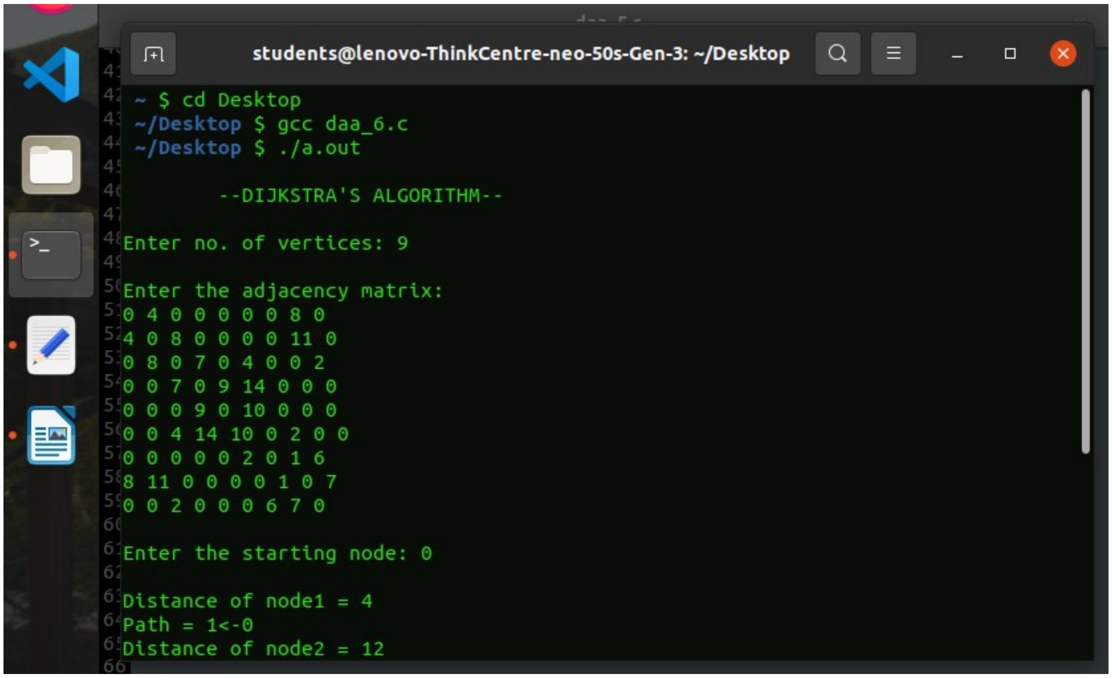
**PROGRAM:**

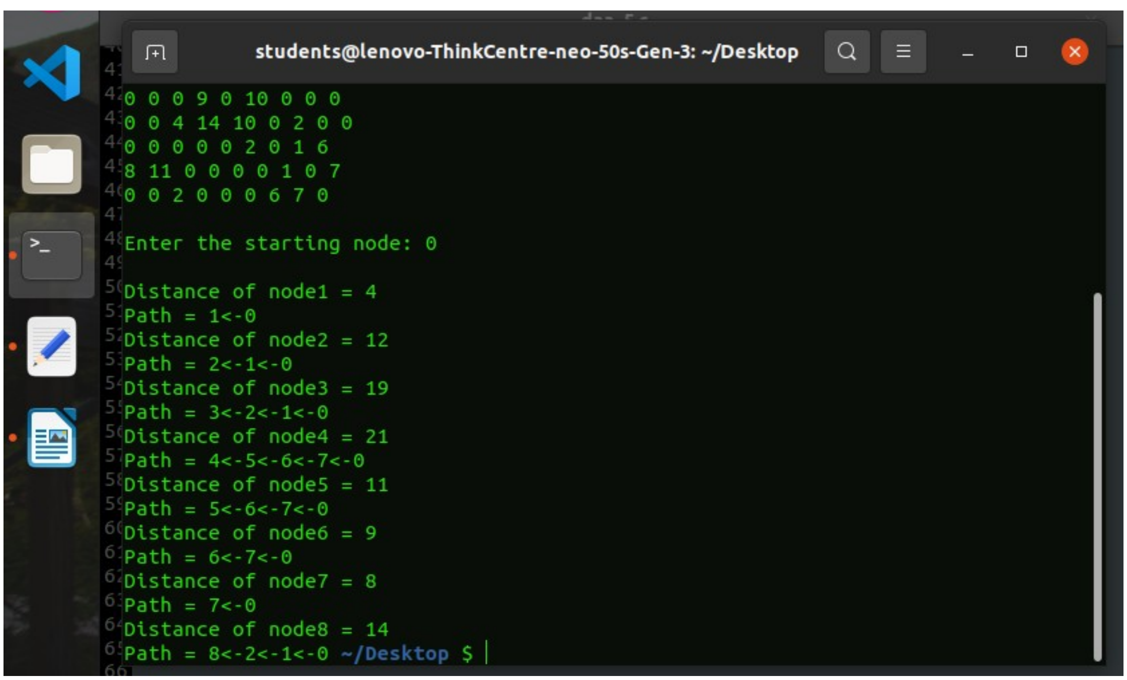
```
#include<stdio.h>
#include<stdlib.h>
#define INFINITY 9999
#define MAX 10

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);
}
}

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("\n\t--DIJKSTRA'S ALGORITHM--");
printf("\n\nEnter 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;
}
```

**OUTPUT:**



**RESULT ANALYSIS:**

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)

So the time complexity of dijkstra's algorithm using adjacency matrix representation comes out to be **O(V2)**

With this implementation, the time to visit each vertex becomes O(V+E) and the time required to process all the neighbours of a vertex becomes O(logV).

Time for visiting all vertices =O(V+E)

Time required for processing one vertex=O(logV)

Time required for visiting and processing all the vertices = O(V+E)*O(logV) = O((V+E)logV)

The space complexity in this case will also improve to **O(V)** as both the adjacency list and min-heap require **O(V)** space. So the total space complexity becomes

O(V)+O(V)=O(2V) = O(V)

| | |
|---|---|
| **CONCLUSION:** | In this experiment I uderstood about the greedy approach- dijkistra’s algorithm to find the shortest path from a single source vertex |