**Task 9: Greedy Technique – Dijkstra’s algorithm**

**Aim:**

Create to c program to find shortest path using dijkstra’s algorithm

**Algorithm**:

Step 1 : Create a set shortPath to store vertices that come in the way of the shortest path tree.

Step 2 : Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.

Step 3 : Loop until all vertices of the graph are in the shortPath.

Step 4 : Take a new vertex that is not visited and is nearest.

Step 5 : Add this vertex to shortPath.

Step 6 : For all adjacent vertices of this vertex update distances. Now check every adjacent vertex of V, if sum of distance of u and weight of edge is less the update it

**Program:**

#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);

if(w<0)

{

printf("Dijkstra’s Algorithm cannot work for a weighted connected graph

with negative weights.");

exit(0);

}

else

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];

}

Sample Input/output:

**Test case 1:**

Enter no. of vertices:4

Enter the adjacency matrix:

0 1 2 0

1 0 7 0

2 7 0 3

0 0 3 0

Enter the starting node:0

Distance of node1=1

Path=1<-0

Distance of node2=2

Path=2<-0

Distance of node3=5

Path=3<-2<-0

**Testcase 2:**

Enter no. of vertices:3

Enter the adjacency matrix:

0 1 -1

1 0 1

-1 1 0

Enter the starting node:0

Dijkstra’s Algorithm cannot work for a weighted connected graph with negative weights

**Result:**

Thus the dijikstra’s algorithm was executed successfully.

**Task 10-Greedy Technique – Topological Sort**

**Aim:**

Implement a c program for topological ordering

**Algorithm:**

Step 1:Store each vertex’s In-Degree in an array D

Step 2. Initialize queue with all “in-degree=0” vertices

Step 3. While there are vertices remaining in the queue:

              (a) Dequeue and output a vertex

              (b) Reduce In-Degree of all vertices adjacent to it by 1

              (c) Enqueue any of these vertices whose In-Degree became zero

Step 4. If all vertices are output then success, otherwise there is a cycle.

**Program:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int n;

int adj[MAX][MAX];

void create\_graph();

int queue[MAX], front = -1,rear = -1;

void insert\_queue(int v);

int delete\_queue();

int isEmpty\_queue();

int indegree(int v);

int main()

{

int i,v,count,topo\_order[MAX],indeg[MAX];

create\_graph();

for(i=0;i<n;i++)

{

indeg[i] = indegree(i);

if( indeg[i] == 0 )

insert\_queue(i);

}

count = 0;

while( !isEmpty\_queue( ) && count < n )

{

v = delete\_queue();

topo\_order[++count] = v;

for(i=0; i<n; i++)

{

if(adj[v][i] == 1)

{

adj[v][i] = 0;

indeg[i] = indeg[i]-1;

if(indeg[i] == 0)

insert\_queue(i);

}

}

}

if( count < n )

{

printf("\nNo topological ordering possible, graph contains cycle\n");

exit(1);

}

printf("\nVertices in topological order are :\n");

for(i=1; i<=count; i++)

printf( "%d ",topo\_order[i] );

printf("\n");

return 0;

}

void insert\_queue(int vertex)

{

if (rear == MAX-1)

printf("\nQueue Overflow\n");

else

{

if (front == -1) /\*If queue is initially empty \*/

front = 0;

rear = rear+1;

queue[rear] = vertex ;

}

}

int isEmpty\_queue()

{

if(front == -1 || front > rear )

return 1;

else

return 0;

}

int delete\_queue()

{

int del\_item;

if (front == -1 || front > rear)

{

printf("\nQueue Underflow\n");

exit(1);

}

else

{

del\_item = queue[front];

front = front+1;

return del\_item;

}

}

int indegree(int v)

{

int i,in\_deg = 0;

for(i=0; i<n; i++)

if(adj[i][v] == 1)

in\_deg++;

return in\_deg;

}

void create\_graph()

{

int i,max\_edges,origin,destin;

printf("\nEnter number of vertices : ");

scanf("%d",&n);

max\_edges = n\*(n-1);

for(i=1; i<=max\_edges; i++)

{

printf("\nEnter edge %d(-1 -1 to quit): ",i);

scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))

break;

if( origin >= n || destin >= n || origin<0 || destin<0)

{

printf("\nInvalid edge!\n");

i--;

}

else

adj[origin][destin] = 1;

}

}

**Sample Input/Output:**

**Testcase 1:**

Enter number of vertices : 4

Enter edge 1(-1 -1 to quit): 0 1

Enter edge 2(-1 -1 to quit): 0 2

Enter edge 3(-1 -1 to quit): 2 3

Enter edge 4(-1 -1 to quit): -1 -1

Vertices in topological order are : 0 1 2 3

**Result:**

Thus the c program for topological ordering was executed.