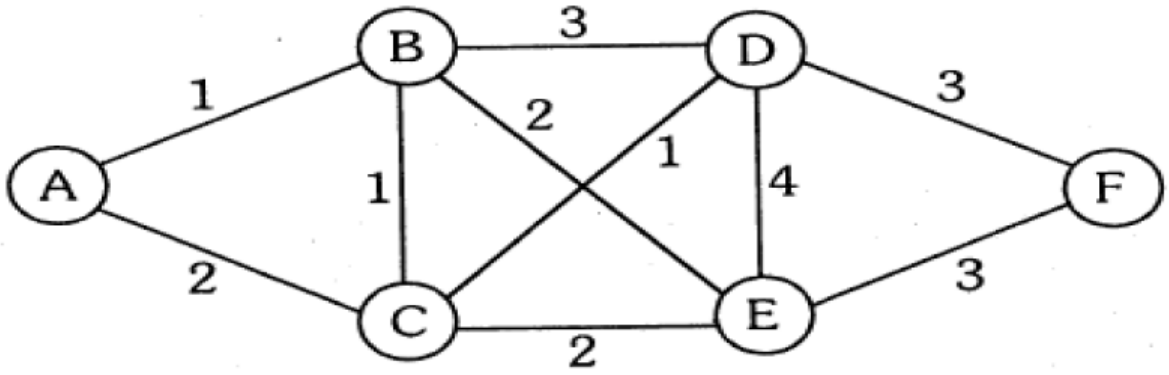
Name: Khushi Nitinkumar Patel

PRN: 2020BTECS00037

Batch: T5

Assignment no 8: Greedy Method

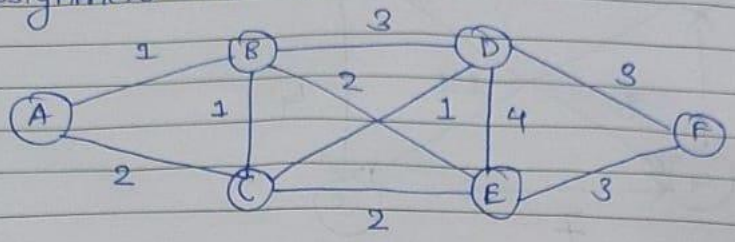
1) From a given vertex in a weighted connected graph, implement shortest path finding Dijkstra's algorithm.



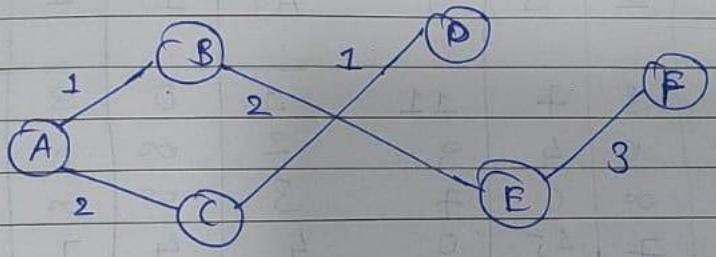
Q) Show that Dijkstra's algorithm doesn't work for graphs with negative weight edges
Q) Modify the Dijkstra's algorithm to find shortest path.

2020BTECS00037
Khushi Nitinkumar Patel.
Assignment no - 8.

1)



source	Destination				
A	B	C	D	E	F
	∞	∞	∞	∞	∞
A, B	(1)	2	∞	∞	∞
A, B, C	(1)	(2)	4	3	∞
A, B, C, D	(1)	(2)	(3)	3	∞
A, B, C, D, E	(1)	(2)	(3)	(3)	6
A, B, C, D, E, F	(1)	(2)	(3)	(3)	(6)

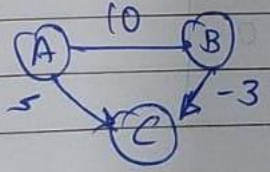


Q.

→ Negative edge problem in Dijkstra's algorithm

Case 1: works for negative edge.

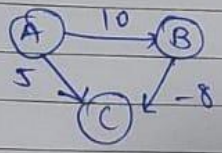
eg:



Source	Destination.	
	B	C
A	∞	∞
A	10	(5)
A.C	(10)	(5)
A.C.B	(10)	(5)

→ Here, negative edge doesn't affect the shortest path from source to destination.
~~If we~~ As A to C get 5 weight.
If we consider negative edge, then A to C will be 7, so as it is not considered, Dijstra's algorithm worked for -ve edge.

Case 2: Non-works for -ve edge.



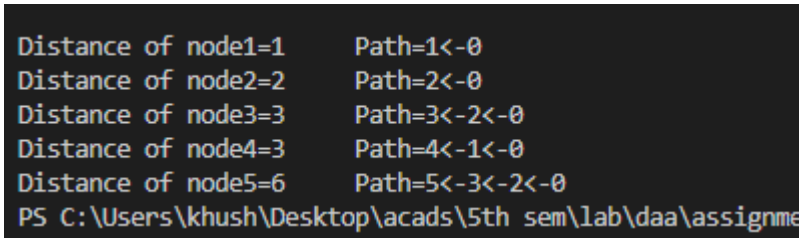
Source	Destinations.	
	B	C
A	∞	∞
A	10	(5)
A,B	(10)	(5)
A,C,B	(10)	(5)

Here, if we consider -ve edge then A to C will be 2 but as it already recived node we can't change its value, so node having small value we choose large so as per dijkstara's algorithm it never looks back again, so here it fails.

```
#include<iostream>
#include<stdio.h>
using namespace std;
#define INFINITY 9999
#define max 6
void dijkstra(int G[max][max],int n,int startnode);
int main() {
int G[max][max]={{0,1,2,0,0,0},{1,0,1,3,2,0},{2,1,0,1,2,0},{0,3,1,0,4,3},{0,0,2,4,
0,3},{0,0,0,3,3,0}};
int n=6; int u=0; dijkstra(G,n,u);
 }
 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;

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];
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;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
  { mindistance=distance[i]; nextnode=i; } 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++; }
for(i=0;i<n;i++)
if(i!=startnode)
  { cout<<"\nDistance of node"<<i<<"="<<distance[i];
  cout<<"\tPath="<<i; j=i;
do { j=pred[j];
  cout<<"<-"<<j; }
while(j!=startnode); } }
```

OUTPUT


Distance of node1=1 Path=1<-0
Distance of node2=2 Path=2<-0
Distance of node3=3 Path=3<-2<-0
Distance of node4=3 Path=4<-1<-0
Distance of node5=6 Path=5<-3<-2<-0
PS C:\Users\khush\Desktop\acads\5th sem\lab\daa\assignme