|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>   #define MAX 100  #define infinity 9999  #define NIL -1  #define TRUE 1  #define FALSE 0    int n;  int adj[MAX][MAX  int predecessor[MAX];  int pathLength[MAX];  int isPresent\_in\_queue[MAX];    int front,rear;  int queue[MAX];  void initialize\_queue( );  void insert\_queue(int u);  int delete\_queue();  int isEmpty\_queue();  void create\_graph( );  void findPath(int s, int v);  int BellmanFord(int s);    int main()  {  int flag,s,v;    create\_graph();    printf("\nEnter source vertex : ");  scanf("%d",&s);    flag = BellmanFord(s);    if(flag == -1)  {  printf("\nError : negative cycle in Graph\n");  exit(1);  }    while(1)  {  printf("\nEnter destination vertex(-1 to quit): ");  scanf("%d",&v);  if(v == -1)  break;  if(v < 0 || v >= n )  printf("\nThis vertex does not exist\n");  else if(v == s)  printf("\nSource and destination vertices are same\n");  else if( pathLength[v] == infinity )              printf("\nThere is no path from source to destination vertex\n");  else  findPath(s,v);  }  return 0;  }/\*End of main()\*/      void findPath(int s, int v )  {  int i,u;  int path[MAX];  int shortdist = 0;  int count = 0;  while( v != s )  {  count++;  path[count] = v;  u = predecessor[v];  shortdist += adj[u][v];  v = u;  }  count++;  path[count]=s;    printf("\nShortest Path is : ");  for(i=count; i>=1; i--)  printf("%d  ",path[i]);  printf("\n Shortest distance is : %d\n", shortdist);  }/    int BellmanFord(int s)  {  int k = 0,i,current;    for(i=0;i<n;i++)  {  predecessor[i] = NIL;  pathLength[i] = infinity;  isPresent\_in\_queue[i] = FALSE;  }    initialize\_queue( );  pathLength[s] = 0;  insert\_queue(s);      isPresent\_in\_queue[s] = TRUE;  while( !isEmpty\_queue( ) )  {  current = delete\_queue( );  isPresent\_in\_queue[current] = FALSE;  if(s == current)  k++;  if(k > n )  return -1;  for(i=0;i<n;i++)  {  if ( adj[current][i] != 0 )  if( pathLength[i] > pathLength[current] + adj[current][i] )  {  pathLength[i] = pathLength[current] + adj[current][i];  predecessor[i] = current;  if( !isPresent\_in\_queue[i] )  {  insert\_queue(i);  isPresent\_in\_queue[i]=TRUE;  }  }  }  }  return 1;  }    void initialize\_queue( )  {  int i;  for(i=0;i<MAX;i++)  queue[i] = 0;  rear = -1;front = -1;  }   int isEmpty\_queue()  {  if(front == -1 || front>rear )  return 1;  else  return 0;  }    void insert\_queue(int added\_item)  {  if (rear == MAX-1)  {  printf("\nQueue Overflow\n");  exit(1);  }  else  {  if (front == -1)  front = 0;  rear = rear+1;  queue[rear] = added\_item ;  }  }    int delete\_queue()  {  int d;  if (front == -1 || front > rear)  {  printf("\nQueue Underflow\n");  exit(1);  }  else  {  d = queue[front];  front=front+1;  }  return d;  }  void create\_graph()  {  int i,max\_edges,origin,destin, wt;    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;    printf("\nEnter weight for this edge : ");  scanf("%d",&wt);    if( origin >= n || destin >= n || origin<0 || destin<0)  {  printf("\nInvalid edge!\n");  i--;  }  else  adj[origin][destin] = wt;  }  } |