Question 1

#include <iostream>

using namespace std;

class node

{

public:

int data;

node \*next;

void push(node \*&head, int d)

{

node \*n = new node;

n->data = d;

n->next = NULL;

if (head == NULL)

{

head = n;

}

else

{

node \*t = head;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

void print(node \*q)

{

node \*n = q;

while (n != NULL)

{

cout << n->data << " ";

n = n->next;

}

cout << "NULL" << endl;

}

};

int main()

{

int v;

cout << "enter the number of vertices : ";

cin >> v;

int e;

cout << "enter the number of edges : ";

cin >> e;

node \*\*head = new node \*[v];

for (int i = 0; i < v; i++)

{

head[i] = NULL;

}

for (int i = 0; i < e; i++)

{

cout << "enter the edge (a<->b) : ";

int a, b;

cin >> a >> b;

head[a]->push(head[a], b);

head[b]->push(head[b], a);

}

cout << "ADJ LIST: \n";

for (int i = 0; i < v; i++)

{

cout << i << " : ";

head[i]->print(head[i]);

}

}

Question 2 ( a)

#include<iostream>

using namespace std;

class node

{

public :

int data;

node \* next;

void push(node \* & h,int d)

{

node \* n=new node;

n->data=d;

n->next=NULL;

if(h==NULL)

{

h=n;

}

else

{

node \* t=h;

while(t->next!=NULL)

{

t=t->next;

}

t->next=n;

}

}

};

void dfs(int u,int visited[],node \*\* head)

{

visited[u]=1;

cout<<u<<" ";

node \* t=head[u];

while(t!=NULL)

{

if(!visited[t->data])

{

dfs(t->data,visited,head);

}

t=t->next;

}

}

int main()

{

int v;

cout<<"enter the number of vertices : ";

cin>>v;

int e;

cout<<"enter the numbr of edges : ";

cin>>e;

node \*\* head = new node \* [v];

int visited[v];

for(int i=0;i<v;i++)

{

head[i]=NULL;

visited[i]=0;

}

for(int i=0;i<e;i++)

{

int a,b;

cout<<"enter the edge (a<->b) : ";

cin>>a>>b;

head[a]->push(head[a],b);

head[b]->push(head[b],a);

}

cout<<"dfs traversal : \n";

for(int i=0;i<v;i++)

{

if(!visited[i])

{

dfs(i,visited,head);

}

}

}

Question 2 (b)

#include <iostream>

using namespace std;

class queue

{

public:

int data;

queue \*next;

queue(int d)

{

this->data = d;

this->next = NULL;

}

void push(queue \*&q, int d)

{

queue \*n = new queue(d);

if (q == NULL)

{

q = n;

}

else

{

queue \*t = q;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

void pop(queue \*&q)

{

if (q != NULL)

{

queue \*t = q;

q = q->next;

delete t;

return;

}

}

int top(queue \*q)

{

if (q != NULL)

{

return q->data;

}

return 0;

}

bool isempty(queue \*q)

{

return q == NULL ? true : false;

}

};

class node

{

public:

int data;

node \*next;

void push(node \*&q, int d)

{

node \*n = new node ;

n->data=d;

n->next=NULL;

if (q == NULL)

{

q = n;

}

else

{

node \*t = q;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

};

void bfs(int i,int visited[],node \*\* head)

{

queue \* q=NULL;

q->push(q,i);

while(!q->isempty(q))

{

int top=q->top(q);

q->pop(q);

cout<<top<<" ";

visited[top]=1;

node \* n=head[top];

while(n!=NULL)

{

if(!visited[n->data])

{

q->push(q,n->data);

}

n=n->next;

}

}

}

int main()

{

// queue \* q=NULL;

// for(int i=0;i<10;i++)

// {

// q->push(q,i+1);

// }

// while(!q->isempty(q))

// {

// cout<<q->top(q)<<" ";

// q->pop(q);

// }

int v;

cout<<"enter the number vertices : ";

cin>>v;

int e;

cout<<"enter the number of edges : ";

cin>>e;

node \*\* head =new node \* [v];

int visited[v];

for(int i=0;i<v;i++)

{

visited[i]=0;

head[i]=NULL;

}

for(int i=0;i<e;i++)

{

cout<<"enter the edge (a<->b) : ";

int a,b;

cin>>a>>b;

head[a]->push(head[a],b);

head[b]->push(head[b],a);

}

cout<<"BFS TRAVERSAL : \n";

int k=1;

cout<<"subgraph "<<k++<<" : ";

bfs(0,visited,head);

for(int i=1;i<v;i++)

{

if(!visited[i])

{

cout<<"\nsubgraph "<<k++<<" : ";

bfs(i,visited,head);

}

}

}

Question 2 ( C )

#include<iostream>

using namespace std;

class node

{

public:

node\* next;

int data;

void push(node \* & h,int d)

{

node \* n=new node;

n->data=d;

n->next=NULL;

if(h==NULL)

{

h=n;

}

else

{

node\* t=h;

while(t->next!=NULL)

{

t=t->next;

}

t->next=n;

}

}

};

int main()

{

int v1;

cout<<"enter the number of nodes of graph 1 : ";

cin>>v1;

int e1;

cout<<"enter the number of edges of graph 1 : ";

cin>>e1;

node \*\* head1= new node \*[v1];

int in1[v1];

for(int i=0;i<v1;i++)

{

head1[i]=NULL;

in1[i]=0;

}

for(int i=0;i<e1;i++)

{

cout<<"enter the edge (a->b) : " ;

int a,b;

cin>>a>>b;

head1[a]->push(head1[a],b);

in1[b]++;

}

for(int i=0;i<v1-1;i++)

{

for(int j=1;j<v1;j++ )

{

if(in1[j]<in1[i])

{

int t=in1[j];

in1[j]=in1[i];

in1[i]=t;

}

}

}

int v2;

cout<<"enter the number of nodes of graph 1 : ";

cin>>v2;

int e2;

cout<<"enter the number of edges of graph 1 : ";

cin>>e2;

node \*\* head2= new node \*[v2];

int in2[v2];

for(int i=0;i<v2;i++)

{

head2[i]=NULL;

in2[i]=0;

}

for(int i=0;i<e2;i++)

{

cout<<"enter the edge (a->b) : " ;

int a,b;

cin>>a>>b;

head2[a]->push(head2[a],b);

in2[b]++;

}

for(int i=0;i<v2-1;i++)

{

for(int j=1;j<v2;j++ )

{

if(in2[j]<in2[i])

{

int t=in2[j];

in2[j]=in2[i];

in2[i]=t;

}

}

}

if(v1==v2 && e1==e2)

{

for(int i=0;i<v1;i++)

{

if(in1[i]!=in2[i])

{

cout<<"non isomorphic .\n";

return 0;

}

}

cout<<"isomorphic .\n";

}

else

{

cout<<"non isomorphic \n";

}

}

Question 2 ( d)

#include<iostream>

using namespace std;

class node

{

public:

node \* next;

int data;

void push(node \* & h,int d)

{

node \* n =new node;

n->data=d;

n->next=NULL;

if(h==NULL)

{

h=n;

}

else

{

node \* t=h;

while(t->next!=NULL)

{

t=t->next;

}

t->next=n;

}

}

void pop(node \* & h)

{

if(h!=NULL)

{

if(h->next==NULL)

{

h=NULL;

return;

}

node \* t=h;

while(t->next->next!=NULL)

{

t=t->next;

}

node\* to = t->next;

t->next=NULL;

delete to;

}

}

int size(node \* h)

{

int c=0;

node \* t=h;

while(t!=NULL)

{

c++;

t=t->next;

}

return c;

}

};

bool path(int u,int v,int visited[], node \* p ,node \*\* head)

{

visited[u]=1;

p->push(p,u);

if(p->size(p)==v)

{

cout<<"HAMILTONIAN PATH : ";

node \* t=p;

while(t!=NULL)

{

cout<<t->data<<" ";

t=t->next;

}

return true;

}

node \* t =head[u];

while(t!=NULL)

{

if(!visited[t->data])

{

if(path(t->data,v,visited,p,head)) return true;

}

t=t->next;

}

visited[u]=0;

p->pop(p);

return false;

}

int main()

{

int v;

cout<<"enter the number of vertices : ";

cin>>v;

int e;

cout<<"enter the number of edges : ";

cin>>e;

node \*\* head = new node \* [v];

int visited[v];

for(int i=0;i<v;i++){

head[i]=NULL;

visited[i]=0;

}

for(int i=0;i<e;i++)

{

cout<<"enter edge (a<->b) : ";

int a,b;

cin>>a>>b;

head[a]->push(head[a],b);

head[b]->push(head[b],a);

}

node \* p=NULL;

bool k=false;

for(int i=0;i<v;i++)

{

if(!visited[i])

{

if(path(i,v,visited,p,head)) k=true;

}

}

if(!k)

{

cout<<"no hamiltonian path . ";

}

}

Question 3 (a)

#include <iostream>

using namespace std;

class node

{

public:

int data;

node \*next;

void push(node \*&h, int d)

{

node \*n = new node;

n->data = d;

n->next = NULL;

if (h == NULL)

{

h = n;

}

else

{

node \*t = h;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

int size(node \* & h)

{

int c=0;

node \* t=h;

while(t!=NULL)

{

c++;

t=t->next;

}

return c;

}

};

int main()

{

int v;

cout<<"enter the vertices : ";

cin>>v;

int e;

cout<<"enter the edges : ";

cin>>e;

node \*\* head =new node \* [v];

int in[v];

int out[v];

for(int i=0;i<v;i++)

{

in[i]=0;

out[i]=0;

head[i]=NULL;

}

for(int i=0;i<e;i++)

{

int a,b;

cout<<"enter edges a<->b : ";

cin>>a>>b;

head[a]->push(head[a],b);

head[b]->push(head[b],a);

}

for(int i=0;i<v;i++)

{

out[i]=head[i]->size(head[i]);

node \* t=head[i];

while(t!=NULL)

{

in[t->data]++;

t=t->next;

}

}

for(int i=0;i<v;i++)

{

cout<<i<<" out degree : "<<out[i]<<" in degree : "<<in[i]<<endl;

}

int k=0;

for(int i=0;i<v;i++)

{

if(in[k]<in[i]) i=k;

}

cout<<"max inorder degree of the node : "<<k;

cout<<endl;

int nd;

cout<<"enter the node whoes indegree has to be find : ";

cin>>nd;

cout<<"indegee of "<<nd<<" is : "<<in[nd];

}

Question 3 (b) cycle

#include<iostream>

using namespace std;

class node

{

public:

node \* next;

int data;

void push(node \* &h,int d)

{

node \* n = new node;

n->next=NULL;

n->data=d;

if(h==NULL)

{

h=n;

}

else

{

node \* t= h;

while(t->next!=NULL)

{

t=t->next;

}

t->next=n;

}

}

};

void solve(int u,bool & c,int visited[],int parent[],node \*\* adj)

{

visited[u]=1;

node \* t=adj[u];

while(t!=NULL)

{

if(!visited[t->data])

{

parent[t->data]=u;

solve(t->data,c,visited,parent,adj);

}

else

{

if(parent[u]!=t->data)

{

c=true;

return;

}

}

t=t->next;

}

}

int main()

{

int v;

cout<<"enter the number of vertices : ";

cin>>v;

int e;

cout<<"enter the number of edges : ";

cin>>e;

int parent[v];

int visited[v];

node \*\* head= new node \*[v];

for(int i=0;i<v;i++)

{

parent[i]=-1;

visited[i]=0;

head[i]=NULL;

}

for(int i=0;i<e;i++)

{

cout<<"enter the edge (a<->b) : " ;

int a,b;

cin>>a>>b;

head[a]->push(head[a],b);

head[b]->push(head[b],a);

}

bool k=false;

for(int i=0;i<v;i++)

{

if(!visited[i])

{

solve(i,k,visited,parent,head);

}

}

if(k)

{

cout<<"cycle is present";

}

else cout<<"cycle is not present";

}

Q3 s to d

#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

class queue

{

public:

queue \*next;

int data;

queue(int d)

{

this->data = d;

this->next = NULL;

}

void push(queue \*&h, int d)

{

queue \*n = new queue(d);

if (h == NULL)

{

h = n;

}

else

{

queue \*t = h;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

void pop(queue \*&h)

{

if (h != NULL)

{

queue \*t = h;

h = h->next;

delete t;

}

}

int top(queue \*&h)

{

if (h != NULL)

{

return h->data;

}

return 0;

}

bool isempty(queue \*&q)

{

return q == NULL ? true : false;

}

};

class node

{

public:

node \*next;

int data;

void push(node \*&h, int d)

{

node \*n = new node;

n->next = NULL;

n->data = d;

if (h == NULL)

{

h = n;

}

else

{

node \*t = h;

while (t->next != NULL)

{

t = t->next;

}

t->next = n;

}

}

};

int main()

{

int v;

cout << "enter the number of vertices : ";

cin >> v;

int e;

cout << "enter the number of edges : ";

cin >> e;

int parent[v];

int visited[v];

node \*\*head = new node \*[v];

for (int i = 0; i < v; i++)

{

parent[i] = -1;

visited[i] = 0;

head[i] = NULL;

}

for (int i = 0; i < e; i++)

{

cout << "enter the edge (a<->b) : ";

int a, b;

cin >> a >> b;

head[a]->push(head[a], b);

head[b]->push(head[b], a);

}

int s, d;

cout << "enter source and destination : ";

cin >> s >> d;

int number\_of\_path=0;

queue \*q = NULL;

q->push(q, s);

while (!q->isempty(q))

{

int t = q->top(q);

cout<<t<<endl;

q->pop(q);

// if(t!=d){

visited[t] = 1;

if(t==d)

{

break;

}

// }

// else

// {

// number\_of\_path++;

// continue;

// }

node \*l = head[t];

while (l != NULL)

{

if (!visited[l->data])

{

parent[l->data] = t;

q->push(q, l->data);

}

l = l->next;

}

}

int c = 0;

int dd = d;

while (dd != -1 && dd != s)

{

c++;

dd = parent[dd];

}

// cout<<dd;

if (dd == s)

{

dd = d;

while (dd != -1 && dd != s)

{

cout<<dd<<" <- ";

dd = parent[dd];

}

cout<<s<<"\npath lenght = "<<c;

cout<<"\nnumber of paths : "<<number\_of\_path;

}

else

{

cout<<"cannot reach from source to destination ";

}

}

// 8 9 0 1 0 4 0 5 1 2 2 3 4 3 5 6 6 7 7 3 0 3

Question 5 a

#include<iostream>

#include<limits.h>

using namespace std;

//implementation of dijkstra algorithm

class pr

{

public:

int key;

int value;

};

class priority\_queue

{

public:

pr \*arr;

int capacity;

int size;

priority\_queue(int c)

{

this->arr = new pr [c];

this->capacity = c;

this->size = 0;

}

void heapify(pr \* arr, int i, int n)

{

int smallest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[smallest].key > arr[l].key)

{

smallest = l;

}

if (r < n && arr[smallest].key > arr[r].key)

{

smallest = r;

}

if (smallest != i)

{

pr t = arr[i];

arr[i] = arr[smallest];

arr[smallest] = t;

heapify(arr, smallest, n);

}

}

void shift\_up(pr arr[], int i)

{

if (i>0)

{

if (arr[i].key < arr[i / 2].key)

{

pr t = arr[i];

arr[i] = arr[i / 2];

arr[i / 2] = t;

shift\_up(arr, i/2);

}

}

}

void insert( int key,int value)

{

if (size == capacity)

{

cout << "overflow\n";

return;

}

arr[size].key= key;

arr[size].value=value;

int i = size;

size++;

shift\_up(arr, i);

}

void del(int d)

{

if (size <= 0)

{

cout << "underflow\n";

return;

}

int i;

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

{

if (arr[i].key == d)

{

break;

}

}

if (i == size)

{

cout << "not found data in heap\n";

return;

}

pr t = arr[i];

arr[i] = arr[size - 1];

arr[size - 1] = t;

size--;

heapify(arr, i, size);

}

pr extract\_min()

{

int i = 0;

pr ans = arr[i];

pr t = arr[i];

arr[i] = arr[size - 1];

arr[size - 1] = t;

size--;

heapify(arr, i, size);

return ans;

}

bool isempty()

{

return size<=0 ? true: false;

}

};

class node

{

public:

int n;

int w;

node \* next;

void push(node \* & h ,int n ,int w)

{

node \* ne = new node ;

ne->n=n;

ne->w=w;

ne->next=NULL;

if(h==NULL)

{

h=ne;

}

else

{

node \* t=h;

while(t->next!=NULL) t=t->next;

t->next=ne;

}

}

};

int main()

{

int v;

cout<<"enter the number of vertices in a direct graph : ";

cin>>v;

int e;

cout<<"enter the number of edges in a direct graph : ";

cin>>e;

node \*\* head = new node \* [v];

for(int i=0;i<v;i++)

{

head[i]=NULL;

}

for(int i=0;i<e;i++)

{

cout<<"enter a->b and weight ";

int a,b,w;

cin>>a>>b>>w;

head[a]->push(head[a],b,w);

}

int s;

cout<<"enter the source node : ";

cin>>s;

int ans[v];

for(int i=0;i<v;i++)

{

ans[i]=INT\_MAX;

}

ans[s]=0;

priority\_queue \* q = new priority\_queue(1000);

q->insert(s,ans[s]);

while(!q->isempty())

{

pr top =q->extract\_min();

int w,a;

w=top.key;

a=top.value;

node \* t= head[a];

while(t!=NULL)

{

if(ans[t->n] > w+t->w)

{

ans[t->n]=w+t->w;

q->insert(t->n,ans[t->n]);

}

t=t->next;

}

}

cout<<"\nanswer : ";

for(int i=0;i<v;i++)

{

if(ans[i]!=INT\_MAX)

cout<<ans[i]<<" ";

else

{

cout<<char(236)<<" ";

}

}

}

Question 5 b

#include<iostream>

#include<algorithm>

using namespace std;

bool cmp (int a[],int b[])

{

return a[2]<b[2];

}

void make\_set(int parent [],int rank [],int n)

{

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

{

parent[i]=i;

rank[i]=0;

}

}

int findparent(int parent[],int node)

{

if(parent[node]==node)

{

return node;

}

return findparent(parent,parent[node]);

}

void unionset(int u,int v,int parent[],int rank[])

{

u=findparent(parent,u);

v=findparent(parent,v);

if(rank[u]<rank[v])

{

parent[u]=v;

}

else if(rank[v]<rank[u])

{

parent[v]=u;

}

else

{

parent[u]=v;

rank[u]++;

}

}

int min\_spanning\_tree(int \*\* edges, int n,int e)

{

sort(edges,edges+e,cmp);

//for(int i=0;i<e;i++)

// {

// cout<<edges[i][0]<<" "<<edges[i][1]<<" "<<edges[i][2]<<"\n";

//}

int rank[n];

int parent[n];

make\_set(parent,rank,n);

int min\_wt=0;

// for(int i=0;i<n;i++) cout<<parent[i]<<" "<<rank[i]<<"\n";

for(int i=0;i<e;i++)

{

int u=findparent(parent,edges[i][0]);

int v=findparent(parent,edges[i][1]);

int wt=edges[i][2];

if(u!=v)

{

unionset(u,v,parent,rank);

min\_wt+=wt;

}

// cout<<parent[u]<<" "<<rank[u]<<"\n";

// cout<<parent[v]<<" "<<rank[v]<<"\n";

// cout<<"cost : "<<min\_wt<<"\n";

}

return min\_wt;

}

int main()

{

int n;

cout<<"enter the number of node : ";

cin>>n;

int e;

cout<<"enter the number of edges : ";

cin>>e;

int \*\* edges = new int \* [e];

for(int i=0;i<e;i++) {edges[i]=new int [3]; }

for(int i=0;i<e;i++)

{

int a,b,w;

cout<<"enter the edge a - b and weight: ";

cin>>a>>b>>w;

edges[i][0]=a;

edges[i][1]=b;

edges[i][2]=w;

}

int cost = min\_spanning\_tree(edges,n+1,e);

cout<<"\nthe min spanning tree cost : "<<cost;

}

// 7 9 1 2 28 2 3 16 3 4 12 4 5 22 5 6 25 6 1 10 2 7 14 7 5 24 7 4 18

// 6 8 1 2 1 2 3 3 1 3 2 3 4 4 4 5 7 3 5 5 3 6 6 6 5 8

Question 6

#include<iostream>

#include<limits.h>

using namespace std;

//implementation of dijkstra algorithm

class pr

{

public:

int key;

int value;

};

class priority\_queue

{

public:

pr \*arr;

int capacity;

int size;

priority\_queue(int c)

{

this->arr = new pr [c];

this->capacity = c;

this->size = 0;

}

void heapify(pr \* arr, int i, int n)

{

int smallest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[smallest].key > arr[l].key)

{

smallest = l;

}

if (r < n && arr[smallest].key > arr[r].key)

{

smallest = r;

}

if (smallest != i)

{

pr t = arr[i];

arr[i] = arr[smallest];

arr[smallest] = t;

heapify(arr, smallest, n);

}

}

void shift\_up(pr arr[], int i)

{

if (i>0)

{

if (arr[i].key < arr[i / 2].key)

{

pr t = arr[i];

arr[i] = arr[i / 2];

arr[i / 2] = t;

shift\_up(arr, i/2);

}

}

}

void insert( int key,int value)

{

if (size == capacity)

{

cout << "overflow\n";

return;

}

arr[size].key= key;

arr[size].value=value;

int i = size;

size++;

shift\_up(arr, i);

}

void del(int d)

{

if (size <= 0)

{

cout << "underflow\n";

return;

}

int i;

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

{

if (arr[i].key == d)

{

break;

}

}

if (i == size)

{

cout << "not found data in heap\n";

return;

}

pr t = arr[i];

arr[i] = arr[size - 1];

arr[size - 1] = t;

size--;

heapify(arr, i, size);

}

pr extract\_min()

{

int i = 0;

pr ans = arr[i];

pr t = arr[i];

arr[i] = arr[size - 1];

arr[size - 1] = t;

size--;

heapify(arr, i, size);

return ans;

}

bool isempty()

{

return size<=0 ? true: false;

}

};

class node

{

public:

int n;

int w;

node \* next;

void push(node \* & h ,int n ,int w)

{

node \* ne = new node ;

ne->n=n;

ne->w=w;

ne->next=NULL;

if(h==NULL)

{

h=ne;

}

else

{

node \* t=h;

while(t->next!=NULL) t=t->next;

t->next=ne;

}

}

};

int main()

{

int v;

cout<<"enter the number of vertices in a direct graph : ";

cin>>v;

int e;

cout<<"enter the number of edges in a direct graph : ";

cin>>e;

node \*\* head = new node \* [v];

for(int i=0;i<v;i++)

{

head[i]=NULL;

}

for(int i=0;i<e;i++)

{

cout<<"enter a->b and weight ";

int a,b,w;

cin>>a>>b>>w;

head[a]->push(head[a],b,w);

}

int s;

cout<<"enter the source node : ";

cin>>s;

int ans[v];

for(int i=0;i<v;i++)

{

ans[i]=INT\_MAX;

}

ans[s]=0;

priority\_queue \* q = new priority\_queue(1000);

q->insert(s,ans[s]);

while(!q->isempty())

{

pr top =q->extract\_min();

int w,a;

w=top.key;

a=top.value;

node \* t= head[a];

while(t!=NULL)

{

if(ans[t->n] > w+t->w)

{

ans[t->n]=w+t->w;

q->insert(t->n,ans[t->n]);

}

t=t->next;

}

}

cout<<"\nanswer : ";

for(int i=0;i<v;i++)

{

if(ans[i]!=INT\_MAX)

cout<<ans[i]<<" ";

else

{

cout<<char(236)<<" ";

}

}

}