Question 1

#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 \* d = q;

q=q->next;

delete d;

}

}

int top(queue \* & q)

{

if(q!=NULL)

{

return q->data;

}

return 0;

}

bool isempty(queue \* & q)

{

return q==NULL ? true : false;

}

};

int main()

{

int n;

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

cin>>n;

int mat[n][n];

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

{

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

{

mat[i][j]=0;

}

}

int e;

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

cin>>e;

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

{

int a,b;

cout<<"enter edge from a - b ";

cin>>a>>b;

mat[a][b]=1;

mat[b][a]=1;

}

int color[n];

int visited[n];

int max\_color=1;

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

{

color[i]=1;

visited[i]=0;

}

int m;

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

cin>>m;

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

{

if(visited[i]) continue;

visited[i]=1;

queue \* q=NULL;

q->push(q,i);

while(q->isempty(q))

{

int top = q->top(q);

q->pop(q);

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

{

if(mat[top][j])

{

if(color[top]==color[j])

{

color[j]++;

}

max\_color=max(max\_color,max (color[top],color[i]));

if(max\_color>m)

{

cout<<"\nnot possible to paint with m colors . ";

return 0;

}

if(!visited[j])

{

visited[j]=true;

q->push(q,j);

}

}

}

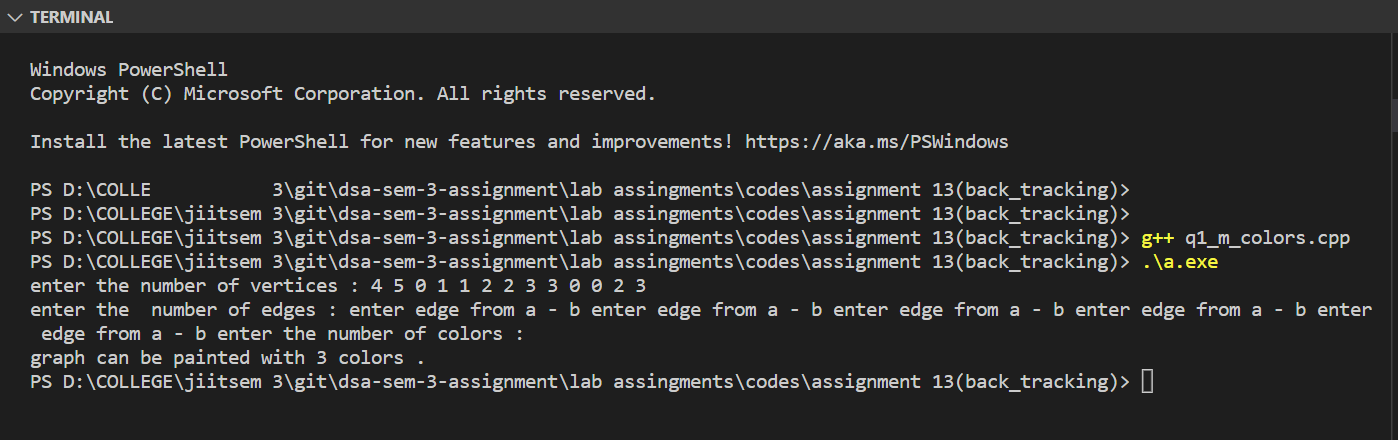
}

}

cout<<"\ngraph can be painted with "<<m<<" colors . ";

}

// 4 5 0 1 1 2 2 3 3 0 0 2 3



Question 2

#include <iostream>

// #include<limit.h>

using namespace std;

bool dfs(int nd, int t, int \*\*temp, int visited[], int n)

{

if (nd == t)

return true;

visited[nd] = 1;

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

{

if (temp[nd][i] && !visited[i])

{

if (dfs(i, t, temp, visited, n))

return true;

}

}

visited[nd] = 0;

return false;

}

int main()

{

int n;

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

cin >> n;

int e;

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

cin >> e;

int \*\*mat = new int \*[n];

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

{

mat[i] = new int[n];

}

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

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

mat[i][j] = 0;

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

{

int a, b;

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

cin >> a >> b;

mat[a][b] = 1;

mat[b][a] = 1;

}

int \*\*temp = new int \*[n];

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

{

temp[i] = new int[n];

}

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

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

temp[i][j] = 0;

int ans = 10000000;

for (int i = 1; i < n - 1; i++)

{

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

{

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

{

temp[l][k] = mat[l][k];

}

}

for (int j = i; j < n - 1; j++)

{

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

{

temp[j][k] = 0;

temp[k][j] = 0;

}

// for (int k = 0; k < n; k++)

// {

// for (int l = 0; l < n; l++)

// {

// cout<<temp[k][l]<<" ";

// }

// cout<<endl;

// }

int visited[n];

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

visited[i] = 0;

if (!dfs(0, n - 1, temp, visited, n))

{

//cout << j << " " << 0 << " " << i << endl;

ans = min(ans, j - i+1);

}

}

}

if(ans == 10000000)

{

cout<<"\ntheif will reach the destination . ";

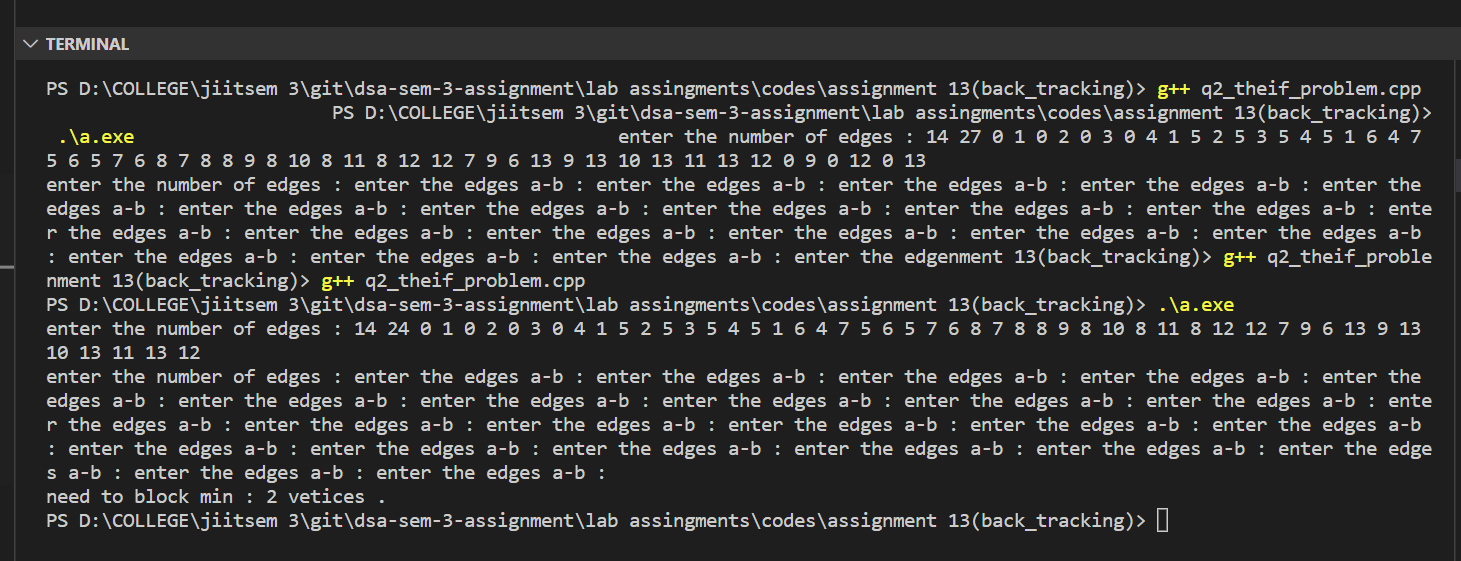
}

else

cout << "\nneed to block min : " << ans << " vetices . ";

} // 14 24 0 1 0 2 0 3 0 4 1 5 2 5 3 5 4 5 1 6 4 7 5 6 5 7 6 8 7 8 8 9 8 10 8 11 8 12 12 7 9 6 13 9 13 10 13 11 13 12

// 5 6 0 1 0 2 0 3 1 4 2 4 3 4



Question 3

#include <iostream>

#include <limits.h>

using namespace std;

class node

{

public:

int n;

int w;

node \*next;

void insert(node \*&head, int d, int w)

{

node \*ne = new node;

ne->n = d;

ne->w = w;

ne->next = NULL;

if (head == NULL)

{

head = ne;

}

else

{

node \*t = head;

while (t->next != NULL)

t = t->next;

t->next = ne;

}

}

};

void dfs(node \*\*head, int nd, int cost, int ct, int &ans, int visited[], int n)

{

if (ct >= n)

return;

if (nd != 0)

{

visited[nd] = 1;

}

node \*t = head[nd];

while (t)

{

if (t->n == 0)

{

if (ct == n - 1)

{

ans = min(ans, cost + t->w);

}

}

else if(!visited[t->n])

{

dfs(head, t->n, cost + t->w, ct+1, ans, visited, n);

}

t = t->next;

}

visited[nd] = 0;

}

int main()

{

int n;

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

cin >> n;

int e;

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

cin >> e;

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

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

{

head[i] = NULL;

}

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

{

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

int a, b, w;

cin >> a >> b >> w;

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

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

}

int visited[n];

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

{

visited[i] = 0;

}

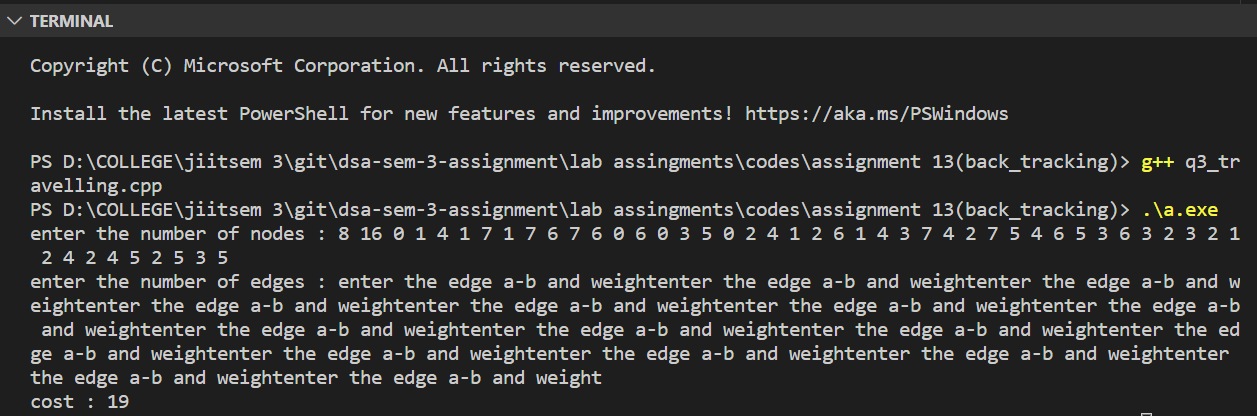
int a = INT\_MAX;

dfs(head, 0, 0, 0, a, visited, n);

cout<<"\ncost : "<<a;

}

// 8 16 0 1 4 1 7 1 7 6 7 6 0 6 0 3 5 0 2 4 1 2 6 1 4 3 7 4 2 7 5 4 6 5 3 6 3 2 3 2 1 2 4 2 4 5 2 5 3 5



Question 4

#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 \* d = q;

q=q->next;

delete d;

}

}

int top(queue \* & q)

{

if(q!=NULL)

{

return q->data;

}

return 0;

}

bool isempty(queue \* & q)

{

return q==NULL ? true : false;

}

};

bool bfs(int \*\* rmat , int s,int t,int parent[],int n)

{

bool visited[n];

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

{

visited[i]=0;

queue \* q=NULL;

q->push(q,s);

visited[s]=1;

parent[s]=-1;

while(!q->isempty(q))

{

int u = q->top(q);

q->pop(q);

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

{

if(visited[i]==false && rmat[u][i]>0)

{

if(i==t)

{

parent[i]=u;

return true;

}

q->push(q,i);

parent[i]=u;

visited[i]=true;

}

}

}

}

return false;

}

int fordFulkerson(int \*\* mat,int s,int t,int n)

{

int u, v;

int \*\* rmat = new int \* [n];

for(int i=0;i<n;i++) rmat[i]=new int [n];

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

{

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

{

rmat[i][j]=mat[i][j];

}

}

int parent[n];

int max\_flow = 0;

while (bfs(rmat, s, t, parent,n)) {

int path\_flow = 1000000;

for (v = t; v != s; v = parent[v]) {

u = parent[v];

path\_flow = min(path\_flow, rmat[u][v]);

}

for (v = t; v != s; v = parent[v]) {

u = parent[v];

rmat[u][v] -= path\_flow;

rmat[v][u] += path\_flow;

}

max\_flow += path\_flow;

}

return max\_flow;

}

int main()

{

int n;

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

cin>>n;

int \*\* mat = new int \*[n];

for(int i=0;i<n;i++) mat[i]=new int[n];

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

{

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

{

mat[i][j]=0;

}

}

int e;

cout<<"enter number of edges : ";

cin>>e;

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

{

cout<<"enter the a -> b and capacity ";

int a,b,c;

cin>>a>>b>>c;

mat[a][b]=c;

}

int s,t;

cout<<"\nenter the starting and target point : ";

cin>>s>>t;

int m = fordFulkerson(mat,s,t,n);

int a ,b;

cout<<"\nenter the add capacity edge a ->b ";

cin>>a>>b;

mat[a][b]++;

cout<<"\nthe new flow (after add capacity) : "<<fordFulkerson(mat,s,t,n);

cout<<"\nenter the reduce capacity edge a ->b ";

cin>>a>>b;

mat[a][b]--;

cout<<"\nthe new flow (after reduce capacity) : "<<fordFulkerson(mat,s,t,n);

}

// 6 10 0 1 16 0 2 13 1 3 12 1 2 10 2 1 4 2 4 14 3 2 9 4 3 7 3 5 20 4 5 4 0 5

