**Dijkstra:**

#include<bits/stdc++.h>

using namespace std;

struct Edge

{

int to, cost;

Edge(int \_to, int \_cost)

{

to=\_to;

cost=\_cost;

}

};

bool operator<(Edge a, Edge b)

{

return a.cost > b.cost;

}

map<int,vector<Edge> >arr;

int dijkstra(int node, int from, int to)

{

priority\_queue<Edge>pq;

map<int,int>dist;

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

dist[i]=INT\_MAX;

dist[from]=0;

pq.push(Edge(from,0)); // farmgate - 0

while(!pq.empty())

{

Edge top=pq.top(); // tsc - 10

pq.pop();

if(dist[top.to]!=top.cost)

continue;

for(int i=0; i<arr[top.to].size(); i++)

{

Edge x=arr[top.to][i]; // karjon - 7

if(dist[top.to]+x.cost<dist[x.to])

//10+7<25

{

dist[x.to]=dist[top.to]+x.cost;

pq.push(Edge(x.to,dist[x.to]));

}

}

}

return dist[to];

}

int main()

{

freopen("dijkstra.txt", "r", stdin);

int cas, node, edge, from, to;

int st, ed, cost, ans;

cin >> cas;

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

{

arr.clear();

cin >> node >> edge >> from >> to;

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

{

cin >> st >> ed >> cost;

arr[st].push\_back(Edge(ed,cost));

}

ans=dijkstra(node,from,to);

printf("From %d to %d = ",from,to);

if(ans!=INT\_MAX)

cout << ans << endl;

else

cout << "Impossible" << endl;

}

return 0;

}

**Floyd\_Warshall:**

#include<bits/stdc++.h>

using namespace std;

int arr[107][107];

void floyd\_warshall(int n)

{

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

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

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

if(arr[i][j]>arr[i][k]+arr[k][j])

arr[i][j]=arr[i][k]+arr[k][j];

}

int main()

{

int test, n, m, a, b;

cin >> test;

for(int t=1; t<=test; t++)

{

cin >> n >> m;

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

{

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

{

if(i==j)

arr[i][j]=0;

else

arr[i][j]=INT\_MAX;

}

}

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

{

cin >> a >> b;

arr[a][b]=1;

arr[b][a]=1;

}

floyd\_warshall(n);

cin >> a >> b;

printf("Shortest path between a and b is:

%d\n",arr[a][b]);

}

return 0;

}

**Disjoint\_Set:**

#include<bits/stdc++.h>

using namespace std;

int parent[50];

int rnk[50];

int member[50];

void initializeSet(int n)

{

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

{

parent[i]=i;

rnk[i]=0;

member[i]=1;

}

}

int findParent(int x)

{

if(x==parent[x])

return x;

else

return parent[x]=findParent(parent[x]);

}

void unionSet(int x, int y)

{

int px=findParent(x);

int py=findParent(y);

if(px==py)

return;

else if(rnk[px]>rnk[py])

{

parent[py]=px;

member[px]+=member[py];

}

else if(rnk[px]<rnk[py])

{

parent[px]=py;

member[py]+=member[px];

}

else

{

parent[py]=px;

rnk[px]++;

member[px]+=member[py];

}

}

void print(int n)

{

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

printf("Node: %d Parent: %d Rank: %d Member: %d\n",i,findParent(i),rnk[i],member[i]);

cout << endl;

}

int main()

{

int n=5;

initializeSet(n);

unionSet(0,0);

unionSet(0,2);

unionSet(1,3);

unionSet(4,0);

unionSet(2,3);

unionSet(1,2);

print(n);

return 0;

}

**Prim:**

#include<bits/stdc++.h>

#define MX 100

using namespace std;

struct Edge

{

int to, cost;

Edge(int \_to, int \_cost)

{

to=\_to;

cost=\_cost;

}

};

bool operator<(Edge a, Edge b)

{

return a.cost>b.cost;

}

vector<Edge>arr[MX];

int prim(int from, int node)

{

priority\_queue<Edge>pq;

bool taken[MX];

int cost[MX];

int x, y, sum=0;

int temp1=INT\_MAX, temp2;

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

{

taken[i]=false;

cost[i]=INT\_MAX;

}

cost[from]=0;

pq.push(Edge(from,cost[from]));

while(!pq.empty())

{

Edge top=pq.top();

pq.pop();

if(taken[top.to])

continue;

taken[top.to]=true;

sum+=top.cost;

for(int i=0; i<arr[top.to].size(); i++)

{

x=arr[top.to][i].to;

y=arr[top.to][i].cost;

if(taken[x])

continue;

if(cost[x]>y)

{

if(temp1>cost[x]) // for 2nd MST

{ // for 2nd MST

temp1=cost[x]; // for 2nd MST

temp2=y; // for 2nd MST

} // for 2nd MST

cost[x]=y;

pq.push(Edge(x,cost[x]));

}

else if(cost[x]<y) // for 2nd MST

{ // for 2nd MST

if(temp1>y) // for 2nd MST

{ // for 2nd MST

temp1=y; // for 2nd MST

temp2=cost[x]; // for 2nd MST

} // for 2nd MST

} // for 2nd MST

}

}

cout << "1st MST: " << sum << endl;

cout << "2nd MST: " << sum-temp2+temp1 << endl;

}

int main()

{

int node, edge, a, b, c;

cin >> node >> edge;

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

{

cin >> a >> b >> c;

arr[a].push\_back(Edge(b,c));

arr[b].push\_back(Edge(a,c));

}

prim(0,node);

return 0;

}

**STL\_List:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int t;

string s;

char c;

cin >> t;

while(t--)

{

list<char>arr;

list<char>::iterator it=arr.begin();

cin >> s;

for(int i=0; s[i]; i++)

{

c=s[i];

if(c!='<' && c!='>' && c!='-')

arr.insert(it,c);

if(c=='<' && it!=arr.begin())

it--;

if(c=='>' && it!=arr.end())

it++;

if(c=='-' && it!=arr.begin())

{

it--;

it=arr.erase(it);

}

}

for(it=arr.begin(); it!=arr.end(); it++)

cout << \*it;

cout << endl;

}

return 0;

}