**Author: Yusuf Reza Hasnat**

**Debug file:**

#include <bits/stdc++.h>

#define int long long

#define float long double

using namespace std;

void \_print(int t) {cerr << t;}

void \_print(char t) {cerr << t;}

void \_print(float t) {cerr << t;}

void \_print(string t) {cerr << t;}

void \_\_print(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>

void \_\_print(const pair<T, V> &x) {cerr << '{'; \_\_print(x.first); cerr << ','; \_\_print(x.second); cerr << '}';}

template<typename T>

void \_\_print(const T &x) {int f = 0; cerr << '{'; for (auto &i: x) cerr << (f++ ? "," : ""), \_\_print(i); cerr << "}";}

void \_print() {cerr << "";}

template <typename T, typename... V>

void \_print(T t, V... v) {\_\_print(t); if (sizeof...(v)) cerr << ", "; \_print(v...);}

template <class T, class V> void \_print(pair <T, V> p);

template <class T> void \_print(vector <T> v);

template <class T> void \_print(set <T> v);

template <class T, class V> void \_print(map <T, V> v);

template <class T> void \_print(multiset <T> v);

template <class T, class V> void \_print(pair <T, V> p) {cerr << "{"; \_print(p.first); cerr << ","; \_print(p.second); cerr << "}";}

template <class T> void \_print(vector <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << " ]";}

template <class T> void \_print(set <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << " ]";}

template <class T> void \_print(multiset <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << " ]";}

template <class T, class V> void \_print(map <T, V> v) {cerr << "[ "; for (auto i : v) {\_print(i); cerr << " ";} cerr << " ]";}

#define dbg(x...) cerr << "[ " << #x << " ] = [ "; \_print(x) ; cerr << " ]" << endl;

**Template:**

//!-----------------------------------!//

//! YUSUF REZA HASNAT !//

//!-----------------------------------!//

#pragma GCC optimize("O3")

#include<bits/stdc++.h>

#ifndef ONLINE\_JUDGE

#include "debug.h"

#else

#define dbg(x...)

#define dbgc(x...)

#endif

using namespace std;

#define int long long

#define float long double

#define vf(v) (v).begin(), (v).end()

#define vr(v) (v).rbegin(), (v).rend()

#define endl "\n"

#define YUSUF ios\_base::sync\_with\_stdio(false),

#define REZA cin.tie(NULL),

#define HASNAT cout.tie(NULL)

int mod = 1000000007;

int inf = 1e18;

void solve(){

int n,m,q,x;

string s;

vector<int>v;

}

int32\_t main(){

YUSUF REZA HASNAT;

int t = 1;

//cin >> t;

for (int i = 1; i <= t; i++){

// case(i)

solve();

}

return 0;

}

**1 Number Theory**

**1.1: Prime number under 100**

// there are 25 numbers

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

**1.12: Decimal to Binary Converstion:**

string Binary=bitset<8>(num).to\_string();

**1.2: Check a number prime or not?**

bool prime(int n){

    if (n<2) return false;

    if (n<=3) return true;

    if (!(n%2) || !(n%3)) return false;

    for (int i=5; i\*i<=n; i+=6)

        if (!(n%i) || !(n%(i+2)))

return false;

    return true;

}

**1.3: Prime factorization**

// smallest prime factor of a number.

int factor(int n){

    if (n%2==0)

        return 2;

    for (int a=3; a<=sqrt(n); a++){

        if (n % a==0)

            return a;

    }

    return n;

}

// complete factorization

int r;

while (n>1){

r = factor(n);

    printf("%d", r);

    n /= r;

}

**1.4: Divisor Count**

int maxVal = 1e6 + 1;

vector<int> countDivisor(maxVal, 0);

void countingDivisor(){

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

for(int j= i; j<maxVal;j+= i)

countDivisor[j]++;

}

**1.5: Leap year**

bool isLeap(int n){

    if (n%100==0){

        if (n%400==0) return true;

      else return false;

}

    if (n%4==0) return true;

    else return false;

}

**1.4: Seive**

int prime[20000005];

void sieve(int n) {

for (int i = 2; i <= n; i++) {

prime[i] = 1;

}

for (int i = 4; i <= n; i += 2) {

prime[i] = 0;

}

for (int i = 3; i \* i <= n; i++) {

if (prime[i]) {

for (int j=i\*i;j<=n;j+=i\*2)

prime[j] = 0;

}

}

}

**1.6: Optimized Sieve(finds(n+1)th prime)** ///TC: O(log(logn))

vector<int> nth\_prime;

bitset<MX> visited;

void optimized\_prime(){

    nth\_prime.push\_back(2);

    for(int i=3; i<MX; i+=2){

            if(visited[i])

                continue;

            nth\_prime.push\_back(i);

            if(1ll\*i\*i > MX)

                continue;

            for(int j = i\*i; j< MX; j+= i+i)

                visited[j] = true;

    }

}

**1.6: BINARY EXPONENTIATION:(a^b)**

int binaryExp(int base, int power)////O(log2(power))

{

if (power == 0)

return 1;

int ans = binaryExp(base, power / 2);

if (power % 2 == 1)

return (((ans \* ans) % mod) \* base) % mod;

return (ans \* ans) % mod;

}

**1.7: BINARY EXPONENTIATION:(a^b^c)**

int binaryExp(int base, int power, int modulo){

int ans = 1;

while (power){

if (power % 2 == 1)

ans = (ans \* base) % modulo;

base = (base \* base) % modulo;

power /= 2;

}

return ans;

}

//function call from main function

binaryExp(a, binaryExp(b, c, mod-1), mod)

//it also works for a^b.

//this works faster than previous. Cause recursion need extra time.

//for bc we need to pass (mod-1) and for a^b^c we need to pass mod.

//mod=1e9 + 7

**1.9: Power:**

int x = (int)(pow(base, power) + 1e-18);

**1.8 Factorial mod**

//n! mod p

//Here P is mod value

//For binaryExp we call 1.6 function

int factmod (int n, int p)

{

    int res = 1;

    while (n > 1){

        res=(res\*binaryExp(p-1,n/p,p))%p;

        for (int i=2; i<=n%p; ++i)

            res=(res\*i) %p;

         n /= p;

    }

    return int (res % p);

}

**1.10: nCr:**

const int MOD = 1e9 + 7;

const int MAX = 2e5 + 5;

vector<int> fact(MAX), inv(MAX);

void factorial() {

fact[0] = 1;

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

fact[i] = (i\*fact[i - 1]) % MOD;

}

int bigmod(int a, int n, int M = MOD) {

int res = 1;

while (n) {

if (n & 1)

res = (res \* a) % M;

a = (a \* a) % M, n /= 2;

}

return res;

}

void inverse() {

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

inv[i] = bigmod(fact[i],MOD - 2);

}

int C(int a, int b) {

if (a < b or a < 0 or b < 0)

return 0;

int de = (inv[b] \* inv[a - b]) % MOD;

return (fact[a] \* de) % MOD;

}

// call factorial() and inverse() from main function

// end nCR

int ModInv(int a, int M) { // M is prime

return bigmod(a, M - 2, M);

}

**Algorithm**

**2.1: Next Greater Element:**

int output[1000005];

void nextGreaterElement(int x[], int n) {

stack<int> s;

s.push(0);

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

while (!s.empty() && x[s.top()]

<= x[i]) {

output[s.top()] = i;

s.pop();

}

s.push(i);

}

while (!s.empty()) {

output[s.top()] = -1;

s.pop();

}

}

**2.2: KMP Algorithm**//complexity O(n+m)

vector<int> createLPS(string pattern){

vector<int> lps(pattern.length());

int index = 0;

for(int i=1;i < pattern.length();){

if(pattern[index]== pattern[i]){

lps[i] = index + 1;

index++, i++;

}

else{

if (index != 0)

index = lps[index - 1];

else

lps[i] = index, i++;

}

}

return lps;

}

int kmp(string text, string pattern)

{

int cnt\_of\_match = 0;

vector<int> lps = createLPS(pattern);

debug(lps);

int i = 0, j = 0;

// i -> text, j -> pattern

while (i < text.length()){

if (text[i] == pattern[j])

i++, j++;

else{

if (j != 0)

j = lps[j - 1];

else

i++;

}

if (j == pattern.length()){

cnt\_of\_match++;

// the index where match found -> (i - pattern.length());

j = lps[j - 1];

}

}

return cnt\_of\_match;

}

**2.3: SEGMENT TREE:**

vector<int> v(2\*1e5 +5),seg(4\*1e5 + 5);

void build(int ti, int low, int high)

{

if (high == low){

seg[ti] = v[low];

return;

}

int mid = (low + high) / 2;

build(2 \* ti + 1, low, mid);

build(2 \* ti + 2, mid + 1, high);

seg[ti] = seg[2\*ti+1]+seg[ti\*2+2];

}

//tree left, tree right, query left, query right, index

int findValue(int ti, int tl, int tr, int ql, int qr)

{

if (tl > qr or tr < ql)

return 0;(sum, xor)

// return INT\_MAX;(min)

// return INT\_MIN;(max)

if (tl >= ql and tr <= qr)

return seg[ti];

int mid = (tl + tr) / 2;

int l=findValue(tl,mid,ql,qr,2\*ti+1);

int r = findValue(mid + 1, tr, ql,

qr, 2 \* ti + 2);

return l + r;(sum)

// return min(l,r);

// return max(l,r);

}

void update(int ti, int low, int high, int id, int val){

if (id > high or id < low)

return;

if (id == high and high == low){

seg[ti] = val;

return;

}

int mid = (low + high) / 2;

update(2 \* ti + 1,low, mid, id, val);

update(2 \* ti+2,mid+1,high, id, val);

seg[ti] = (seg[2\*ti+1]+seg[ti\*2+2]);

}

**2.4: DSU**

#For every i, set parent[i]=i and size[i]=1

int find\_set(int x){

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

return parent[x]=find\_set(parent[x]);

}

void Union(int x, int y){

x=find\_set(x);y=find\_set(y);

if(x==y) return;

if(Size[x]>Size[y]) swap(x,y);

parent[x]=y;

Size[y]+=Size[x];

}

**2.5 SEGMENT TREE LAZY**

const int N = 1e5 + 100;

int tree[N << 2], lz[N << 2];

void propagate(int u, int st, int en)

{

if (!lz[u]) return;

tree[u] += lz[u] \* (en - st + 1);

if (st != en)

{

lz[2 \* u] += lz[u];

lz[2 \* u + 1] += lz[u];

}

lz[u] = 0;

}

void update(int u, int st, int en, int l, int r, int x)

{

propagate(u, st, en);

if (r < st or en < l) return;

else if (st >= l and en <= r)

{

lz[u] += x;

propagate(u, st, en);

}

else

{

int mid = (st + en) >> 1;

update(2 \* u, st, mid, l, r, x);

update(2 \* u + 1, mid + 1, en, l, r, x);

tree[u] = tree[2 \* u] + tree[2 \* u + 1];

}

}

int query(int u, int st, int en, int l, int r)

{

propagate(u, st, en);

if (r < st or en < l) return 0;

else if (st >= l and en <= r) return tree[u];

else

{

int mid = (st + en) >> 1;

int left = query(2 \* u, st, mid, l, r);

int right = query(2 \* u + 1, mid + 1, en, l, r);

return left + right;

}

}

**2.5: SET BALANCING:**

void balance(multiset<int> right, multiset<int> &left){

while (true){

int st = right.size();

int sl = left.size();

if (st == sl || st == sl + 1)

break;

if (st < sl)

right.insert(\*left.begin()), left.erase(left.begin());

else

left.insert(\*right.rbegin()), right.erase(right.rbegin());

}

}

void insert\_in\_set(multiset<int> &right, multiset<int> &left, int value)

{

if (right.emptleft())

right.insert(value);

else

{

auto it = right.end();

it--;

if (value < \*it)

right.insert(value);

else

left.insert(value);

}

}

**3: Dynamic Programming**

**3.1: 0/1 Knapsack problems- O(n\*w)**

**//Top Down Approch**

int dp[1001][100001];

int items[2][1001]; //0,i->wt, 1,i->value

int knapsack(int i, int wt) {

if (i < 0 || wt <= 0) return 0;

if (dp[i][wt] !=-1) return dp[i][wt];

if (items[0][i] <= wt){

return dp[i][wt] = max(items[1][i] + knapsack(i - 1, wt - items[0][i]),knapsack(i - 1, wt));

}

else

return dp[i][wt] = knapsack(i - 1, wt);

}

**//Bottom Up**

int knapsack(int capacity, int ind){

for(int i=1;i<=ind;i++){

for(int c=1;c<=capacity;c++){

if(weight[i]>c){

mem[i][c]=mem[i-1][c];

}

else{

int k1=mem[i-1][c];

int k2=value[i]+mem[i-1][c-weight[i]];

mem[i][c]=max(k1,k2);

}

}

}

int max\_profit=mem[ind][capacity];

return max\_profit;

}

**3.2: Complete Knapsack problems**

int f[1000]= {0};

int n=0, m=0;

int main(void){

    cin >> n >> m;

    for (int i=1; i<=n; i++){

        int price=0, value=0;

        cin >> price >> value;

        for (int j=price; j<=m; j++)

            if (f[j-price]+value>f[j])

                f[j]=f[j-price]+value;

    }

    cout << f[m] << endl;

    return 0;

}

**3.3: Longest common subsequence (LCS)-O(n\*m)**

int dp[N][N], mark[N][N];

**//TOP DOWN**

int LCS(int i, int j, string a,string b){

if (i == 0 or j == 0)

return 0;

if (dp[i][j] != -1)

return dp[i][j];

if (a[i - 1] == b[j - 1]) {

mark[i][j] = 1;

return dp[i][j] = LCS(i - 1, j –

1, a, b) + 1;

}

else {

int left = LCS(i - 1, j, a, b),

right = LCS(i, j - 1, a, b);

left > right ?

mark[i][j] = 2 : mark[i][j] = 3;

return dp[i][j]=max(left, right);

}

}

**//BOTTOM UP**

int LCS(string a, string b) {

int n = a.size(), m = b.size();

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

for (int j = 1; j <= m; j++) {

if (a[i - 1] == b[j - 1])

dp[i][j]=1+dp[i-1][j-1],

mark[i][j] = 1;

else {

dp[i-1][j]>dp[i][j - 1] ?

mark[i][j]=2 : mark[i][j]=3;

dp[i][j]=max(dp[i-1][j],

dp[i][j - 1]);

}

}

}

}

// ans store in dp[n][m]

**//Lexicographically smallest LCS**

string dp[105][105]; //intial with “.”;

string LCS(int i, int j) {

if (i <= 0 or j <= 0)

return dp[i][j] = "";

if (dp[i][j] != ".")

return dp[i][j];

if (a[i - 1] == b[j - 1])

return dp[i][j] = LCS(i-1,j-1)

+ a[i - 1];

else {

string iP = LCS(i - 1, j);

string jP = LCS(i, j - 1);

if (iP.size() > jP.size())

return dp[i][j] = iP;

if (iP.size() < jP.size())

return dp[i][j] = jP;

if (iP < jP)

return dp[i][j] = iP;

return dp[i][j] = jP;

}

}

**// LCS string**

void print\_lcs(int i, int j, string a, string b) {

if (i == 0 or j == 0)

return;

if (mark[i][j] == 1)

print\_lcs(i - 1, j - 1, a, b),

cout << a[i - 1];

if (mark[i][j] == 2)

print\_lcs(i - 1, j, a, b);

if (mark[i][j] == 3)

print\_lcs(i, j - 1, a, b);

}

**3.4: Longest increasing common sequence (LICS)**

int a[100]= {0};

int b[100]= {0};

int f[100]= {0};

int n=0, m=0;

int main(void){

    cin >> n;

    for (int i=1; i<=n; i++) cin >> a[i];

    cin >> m;

    for (int i=1; i<=m; i++) cin >> b[i];

    for (int i=1; i<=n; i++)

    {

        int k=0;

        for (int j=1; j<=m; j++)

        {

            if (a[i]>b[j] && f[j]>k)

k=f[j];

            else if (a[i]==b[j] &&

k+1>f[j]) f[j]=k+1;

        }

    }

    int ans=0;

    for (int i=1; i<=m; i++)

        if (f[i]>ans) ans=f[i];

    cout << ans << endl;

    return 0;

}

**3.5: Longest Increasing Subsequence (LIS)-O(n^2)**

int n=0;

int a[100]={0};

int f[100]= {0},x[100]={0};

int main(void){

    cin >> n;

    for (int i=1; i<=n; i++){

        cin >> a[i];

        x[i]=INT\_MAX;

    }

    f[0]=0;

    int ans=0;

    for(int i=1; i<=n; i++){

        int l=0, r=i;

        while (l+1<r){

            int m=(l+r)/2;

            if (x[m]<a[i]) l=m;

            else r=m;

// change to x[m]<=a[i]

for non-decreasing case

        }

        f[i]=l+1;

        x[l+1]=a[i];

        if (f[i]>ans) ans=f[i];

    }

    cout << ans << endl;

    return 0;

}

**3.6: MCM:**

const int N = 1005;

int d[N];

int dp[N][N], mark[N][N];

int MCM(int i, int j) {

if (i == j)

return dp[i][j] = 0;

if (dp[i][j] != -1)

return dp[i][j];

int mn = inf;

for (int k = i; k < j; k++) {

int x = mn;

mn = min(mn, MCM(i, k) + MCM(k +

1,j)+d[i-1]\* d[k] \* d[j]);

if (x != mn)

mark[i][j] = k;

}

return dp[i][j] = mn;

}

**4: Graph**

**4.1: BFS**

map<int, vector<int>> adj;

map<int, int> visited, parent, level, color;

void bfs(int start) {

queue<int> q;

q.push(start);

visited[start] = 1;

while (!q.empty()) {

int k = q.front();

q.pop();

for (auto child : adj[k]) {

if (!visited[child]) {

q.push(child);

visited[child] = 1;

visited[child] = 1;

level[child]=level[k]+1;

}

}

}

}

**4.2: DFS**

map<int, vector<int>> adj;

map<int, int> visited, parent, level, color;

void dfs(int start) {

visited[start] = 1;

for (auto child : adj[start]) {

if (!visited[child]) {

dfs(child);

}

}

visited[start] = 2;

}

**4.3: Bipartite Graph**

bool dfs(int start, int clr)

{

color[start]=clr,visited[start]= 1;

for (auto child : adj[start]){

if (!visited[child]){

if (!dfs(child, clr ^ 1))

return false;

}

else if (color[child] == color[start])

return false;

}

return true;

}

***4.4:* Knight Moves**

int X[8]={2,1,-1,-2,-2,-1,1,2};

int Y[8]={1,2,2,1,-1,-2,-2,-1};

**4.5: Rerooting**

map<int, vector<int>> m;

int dp[1000001], dp1[1000001],

sub[1000001], n;

void dfs(int x, int parent) {

dp[x] = 0;

sub[x] = 1;

for (int i = 0; i<m[x].size(); i++){

if (m[x][i] != parent) {

dfs(m[x][i], x);

sub[x] += sub[m[x][i]];

dp[x] += dp[m[x][i]] +

sub[m[x][i]];

}

}

}

void dfs1(int x, int parent, int carry) {

dp1[x] = dp[x] + carry;

sub[x] = 1;

for (int i = 0; i<m[x].size(); i++) {

if (m[x][i] != parent) {

int parent\_dp = dp1[x];

parent\_dp = dp[m[x][i]] +

sub[m[x][i]];

int parent\_sub = (n –

sub[m[x][i]]);

int new\_carry = parent\_dp +

parent\_sub;

dfs1(m[x][i], x, new\_carry);

}

}

}

int main() {

int x, y, n;

cin >> n;

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

cin >> x >> y;

m[x].pb(y), m[y].pb(x);

}

dfs(1, -1);

dfs1(1, -1, 0);

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

cout<<i+1 <<" "<<dp[i+1]<<"\n";

}

m.clear();

return 0;

}

**4.6: SPFA — shortest path O(VxE)**

int q[3001]= {0}; // queue for node

it d[1001]= {0}; // record shortest path

from start to ith node

bool f[1001]= {0};

int a[1001][1001]= {0}; // adjacency list

int w[1001][1001]= {0};//adjacency matrix

void SPFA(int v0);

int main(void) {

  int n=0, m=0;

  cin >> n >> m;

  for (int i=1; i<=m; i++){

   int x=0, y=0, z=0;

cin >> x >> y >> z;

// node x to node y has weight z

a[x][0]++;

  a[x][a[x][0]]=y;

   w[x][y]=z;

/\*// for undirected graph

  a[x][0]++;

a[y][a[y][0]]=x;

w[y][x]=z;

\*/

}

int s=0, e=0;

cin >> s >> e; // s: start, e: end

SPFA(s);

cout << d[e] << endl;

return 0;

}

void SPFA(int v0)

{

    int t,h,u,v;

    for (int i=0; i<1001; i++)

d[i]=INT\_MAX;

    for (int i=0; i<1001; i++)

f[i]=false;

    d[v0]=0, h=0, t=1, q[1]=v0;

    f[v0]=true;

    while (h!=t)

    {

        h++;

        if (h>3000) h=1;

        u=q[h];

        for (int j=1; j<=a[u][0]; j++)

        {

            v=a[u][j];

            if (d[u]+w[u][v]<d[v]) // change to > if calculating longest path

            {

                d[v]=d[u]+w[u][v];

                if (!f[v])

                {

                    t++;

                    if (t>3000) t=1;

                    q[t]=v;

                    f[v]=true;

                }

            }

        }

        f[u]=false;

    }

}

**4.7.1: Prim — MST by Hasnat**

typedef pair<int,pair<int,int>> pairUV;

map<int, bool> visited;

map<int, vector<pair<int, int>>> adj;

void Prims() {

int sum = 0, c = 0;

vector<pairUV> ans;

priority\_queue<pairUV, vector<pairUV>, greater<pairUV>> pq;

pq.push({0, {1, -1}});

while (!pq.empty()) {

pairUV k = pq.top();

pq.pop();

int u = k.second.first;

int v = k.second.second;

int wt = k.first;

if (visited[u])

continue;

sum += wt;

visited[u] = 1;

if (v != -1)

ans.pb({wt, {u, v}});

for (auto it : adj[u]) {

int adjNode = it.first;

int adjwt = it.second;

if (!visited[adjNode])

pq.push({adjwt, {adjNode, u}});

}

}

}

**4.7.2:Prim—MST by Rizu bhai O(ElogV)**

int d[1001]= {0};

bool v[1001]= {0};

int a[1001][1001]= {0};

int prim(int u, int n)

{

    int mst=0,k;

    for (int i=0; i<d.length; i++) d[i]=INT\_MAX;

    for (int i=0; i<v.length; i++) v[i]=false;

    d[u]=0;

    int i=u;

    while (i!=0)

    {

        v[i]=true;

        k=0;

        mst+=d[i];

        for (int j=1; j<=n; j++)

            if (!v[j])

            {

                if (a[i][j]<d[j]) d[j]=a[i][j];

                if (d[j]<d[k]) k=j;

            }

        i=k;

    }

    return mst;

}

int main(void)

{

    int n=0;

    cin >> n;

    for (int i=1; i<=n; i++)

    {

        int x=0, y=0, z=0;

        cin >> x >> y >> z;

        a[x][y]=z;

    }

    for (int i=1; i<=n; i++)

        for (int j=1; j<=n; j++)

            if (a[i][j]==0) a[i][j]=INT\_MAX;

    cout << prim(1,n) << endl;

}

**4.8: Kruskal**

int n, e;

class DSU {

int\* parent;

int\* \_size;

public:

DSU(int n) {

parent = new int[n + 1];

\_size = new int[n + 1];

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

parent[i] = i;

\_size[i] = 1;

}

}

int find\_set(int x) {

if (x == parent[x])

return x;

int y = find\_set(parent[x]);

parent[x] = y;

return y;

}

void Union(int x, int y) {

int rx = find\_set(x);

int ry = find\_set(y);

if (rx == ry)

return;

if (\_size[rx] <= \_size[ry]) {

parent[rx] = parent[ry];

\_size[ry] += \_size[rx];

}

else {

parent[ry] = parent[rx];

\_size[rx] += \_size[ry];

}

}

~DSU() {

delete parent;

delete \_size;

}

};

int Kruskal(pair<int, pair<int, int>> edges[]) {

DSU d(n);

sort(edges, edges + n + 1);

int weight = 0;

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

int w = edges[i].first;

int u = edges[i].second.first;

int v = edges[i].second.second;

if (d.find\_set(u) != d.find\_set(v)) {

weight += w;

d.Union(u, v);

}

}

return weight;

}

int main() {

cin >> n >> e;

pair<int, pair<int, int>> edges[e];

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

int u, v, w;

cin >> u >> v >> w;

edges[i].first = w;

edges[i].second.first = u;

edges[i].second.second = v;

}

int ans = Kruskal(edges);

cout << ans << "\n";

}

**4.9: Topological sort**

// Find any solution of topological sort.

int f[100]= {0}, ans[100]= {0};

int bool g[100][100]= {0}, v[100]= {0};

int n=0, m=0;

void dfs(int k)

{

    int i=0;

    v[k]=true;

    for (int i=1; i<=n; i++)

        if (g[k][i] && !v[i]) dfs(i);

    m++;

    ans[m]=k;

}

int main(void){

    cin >> n >> m;

    for (int i=1; i<=m; i++){

        int x=0, y=0;

        cin >> x >> y;

        g[y][x]=true;

    }

    m=0;

    for (int i=1; i<=n; i++)

        if (!v[i])

dfs(i);

    for (int i=1; i<=n; i++)

cout << ans[i] << endl;

    return 0;

}

**4.10: Dijkstra:**

map<int, vector<pair<int, int>>> m;

map<int, int> dist;

#define pairi pair<int, int>

void dijkstra(int src, int n) {

priority\_queue<pairi, vector<pairi>, greater<pairi>> pq;

pq.push({0, src});

dist[src] = 0;

vector<int> dis(n, inf);

dis[src] = 0;

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (int i = 0; i < m[u].size(); i++) {

int wt = m[u][i].second;

int v = m[u][i].first;

if (dis[v] > dis[u] + wt) {

dis[v] = dis[u] + wt;

pq.push({dis[v], v});

dist[v] = dis[u] + wt;

}

}

}

}

**4.11: Floyd-Warshall algorithm – shortest path of all pairs O(n^3)**

// map[i][j]=infinity at start

void floyd()

{

    for (int k=1; k<=n; k++)

        for (int i=1; i<=n; i++)

            for (int j=1; j<=n; j++)

                if (i!=j && j!=k && i!=k)

                    if (map[i][k]+map[k][j]<map[i][j])

      map[i][j]=map[i][k]+map[k][j];

}