Seive theroum

1. **for** (**int** i = 0; i <= 1e7; i++)
2. prime[i] = **true**;
4. **for** (**int** p = 2; p \* p <= 1e7; p++) {
5. **if** (prime[p]) {
6. **for** (**int** i = p \* p; i <= 1e7; i += p)
7. prime[i] = **false**;
8. }
9. }
10. vector<**int**> vec;
11. **for** (**int** i = 2; i <= sqrt(1e7); i++)
13. **if** (prime[i]) {
14. vec.push\_back(i);
15. }
16. Minimum spinning Tree- prim algorithm
17. vector<vector<pair<**int**,**int**>>> adj;
18. **int** vis[(**int**) 1e5 + 5];
19. **int** par[(**int**) 1e5 + 5];
20. **int** cost =0 ;
21. **void** MST(**int** node) {
22. priority\_queue<pair<**int**, **int**>> pq;
23. pq.push({0, node});
24. **while** (!pq.empty()) {
25. auto currNode = pq.top();
26. pq.pop();
27. **if**(vis[currNode.second])
28. {
29. **continue**;
30. }
31. vis[currNode.second] = 1;
32. cost +=currNode.first\*-1 ;
33. // cout<<currNode.second<<endl ;
34. **for** (auto child :adj[currNode.second]) {
35. **if** (!vis[child.first]) {
36. pq.push({-1\*child.second,child.first}) ;
37. par[child.first]=currNode.second ;
38. cout<<child.first<<" "<<currNode.second <<endl ;
39. }
40. }
41. }
42. }
44. int32\_t main() {
45. iso;
46. **int** n ,m  ;
47. cin>> n>>m;
48. adj.resize(n+1) ;
49. **for**(**int** i=0 ;i<m ;i++)
50. {
51. **int** u ,v ,c ;
52. cin>> u>>v>>c ;
53. adj[u].pb({v,c}) ;
54. adj[v].pb({u,c}) ;
55. }
56. MST(1) ;
57. **int** pt =1 ;
58. cout<<cost<<endl ;

**Merge Sort Tree(Number of numbers less than a number)**

#include <bits/stdc++.h>

#define FAST ios\_base::sync\_with\_stdio(false); cin.tie(0);

#define file freopen("john.in","r", stdin) ; freopen("john.out", "w", stdout) ;

#define int long long

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

using namespace std ;

const int maxn = 1e5+10;

class Bit

{

public:

vector<int> arr ;

vector<vector<int>> tree ;

Bit (vector<int> v)

{

arr.resize(maxn) ; tree.resize(4\*maxn) ;

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

}

void build(int node, int l, int r) {

if(l == r) {

tree[node].push\_back(arr[l]);

return;

}

int mid = (l + r) >> 1,

left = node << 1, right = left|1;

build(left, l, mid);

build(right, mid+1, r);

merge(all(tree[left]), all(tree[right]),

back\_inserter(tree[node]));

}

int query(int node, int l, int r, int i, int j, int k) {

if(i > r || l > j) return 0;

if(i <= l && r <= j) {

return lower\_bound(all(tree[node]), k)

- tree[node].begin();

}

int mid = (l + r) >> 1,

left = node << 1, right = left|1;

return query(left, l, mid, i, j, k) +

query(right, mid+1, r, i, j, k);

}

};

int32\_t main()

{

FAST file

int n ;

cin>>n ;

Bit tree2 (red2); tree2.build(1,0,red2.size()-1) ;

ans+=tree2.query(1,0,red2.size()-1,0,i,red2[i]) ;

}

**Segment Tree**

#include <iostream>

using namespace std;

int seg\_tree[(int)1e5+5] ;

int arr[(int)1e5+5] ;

int lazy[(int)1e5] ;

int n ;

int N =1 ;

void Fix\_arr()

{

while(N<n)

{

N\*=2 ;

}

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

{

arr[i]=0 ;

}

}

int propagate(int node,int l ,int r)

{

int left\_child =node\*2 ;

int right\_child =node\*2+1 ;

int mid = (l+r)>>1 ;

seg\_tree[left\_child]+=(mid-l+1)\*lazy[node] ;

seg\_tree[right\_child]+=(r-mid)\*lazy[node] ;

lazy[left\_child]+=lazy[node] ;

lazy[right\_child]+=lazy[node] ;

lazy[node]= 0 ;

}

void update\_point(int i ,int value)

{

int node = i+N-1 ;

arr[node] +=value ;

seg\_tree[node]+=value ;

while(node>1)

{

node/=2 ;

int left\_child =node\*2 ;

int right\_child =node\*2+1 ;

seg\_tree[node]=seg\_tree[left\_child]+seg\_tree[right\_child] ;

}

return ;

}

void Build(int node,int l,int r)

{

if(l==r)

{

seg\_tree[node]=arr[l] ;

return ;

}

int mid =(l+r)>>1 ;

int left\_child = 2\*node ;

int right\_child =2\*node+1 ;

Build(left\_child,l,mid) ;

Build(right\_child,mid+1,r) ;

seg\_tree[node]=seg\_tree[left\_child]+seg\_tree[right\_child] ;

}

int query(int node,int l,int r,int x,int y)

{

if(l>=x&&r<=y)

{

return seg\_tree[node] ;

}

if(x>r||y<l)

{

return 0 ;

}

propagate(node,l,r) ;

int mid =(l+r)>>1 ;

int left\_child = 2\*node ;

int right\_child =2\*node+1 ;

int Left\_value=query(left\_child,l,mid,x,y) ;

int Right\_value = query(right\_child,mid+1,r,x,y) ;

return Left\_value+Right\_value ;

}

void update\_range(int node ,int l,int r,int x,int y,int value )

{

if(l>=x&&r<=y)

{

lazy[node]+=value ;

seg\_tree[node]+=(r-l+1)\*value ;

return ;

}

if(x>r||y<l)

{

return ;

}

propagate(node,l,r) ;

int left\_child =node\*2 ;

int right\_child =node\*2+1 ;

int mid =(l+r)>>1 ;

update\_range(left\_child ,l,mid,x,y,value) ;

update\_range(right\_child ,mid+1,r,x,y,value) ;

seg\_tree[node]=seg\_tree[left\_child]+seg\_tree[right\_child];

}

int main()

{

cin>>n ;

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

{

cin>>arr[i] ;

}

Fix\_arr() ;

Build(1,1,N) ;

int k ;

cin>> k ;

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

{

int l,r ,z,point;

cin>> l>>r>>z>>point ;

cout<< query(1,1,N,l,r)<<endl ;

update\_point(point,z) ;

cout<< query(1,1,N,l,r)<<endl ;

}

}

Floyed warshall algorithm Get all sssp pairs

memset(dis, '?', sizeof dis);

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

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

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

if (dis[i][k] == INT\_MAX || dis[k][j] == INT\_MAX)

{

continue;

}

dis[i][j] = min(dis[i][j], dis[i][k] + dis[j][k]);

}

}

}

Flating Tree (convert Tree to Array)

int n;

int starting [(int)1e5+5] ;

int ending [(int)1e5+5] ;

vector<vector<int>>adj ;

int FAT[(int)1e5+5] ;

int timer =1 ;

void dfs\_flating(int node,int par)

{

FAT[timer]=node ;

starting[node]=timer ;

timer++ ;

for(auto child:adj[node])

{

if(par!=child)

{

dfs\_flating(child,node) ;

}

}

ending[node]=timer ;

FAT[timer]=node ;

timer++ ;

}

int32\_t main() {

iso;

cin>> n ;

adj.resize(n+5) ;

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

{

int u ,v ;

cin>>u>>v ;

adj[u].push\_back(v) ;

adj[v].push\_back(u) ;

}

dfs\_flating(1,-1) ;

*/\**

*FAT is the new tree , if we wanna to know a subtree of a node , we can traverse from starting time of it*

*from FAT to ending time of it from FAT*

*Starting and ending are start and end time of each node*

*array start index > Node*

*\*/*

}

FastPower With Mod Inverse

long long fast\_power(long long a, long long b) {

long long res = 1;

while (b > 0) {

if (b & 1) {

res = res \* a;

res%=MOD;

}

a = a \* a;

a%=MOD ;

b >>= 1;

}

return res%=MOD;

}

DSU(Disjoint Set)

struct dsu {

int par[N], rank[N];

void init() {

iota(par, par + N, 0);

memset(rank, 0, sizeof rank);

}

int root(int u) {

if (par[u] == u) {

return u;

}

return par[u] = root(par[u]);

}

void make\_set(int u, int v) {

u = root(u);

v = root(v);

if (u == v) {

return;

}

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

swap(u, v);

}

par[v] = u;

if (rank[u] == rank[v]) {

rank[u]++;

}

}} DSU;

DP\_Trace(Build OutPut)

void trace(int index,int rem)

{

if(index==n)

{

return ;

}

int take =0 ;

int leave =0 ;

if(rem>=vv[index])

{

take =vec[index].second+dp(index+1,rem-vv[index]) ;

}

leave =dp(index+1,rem) ;

if(take==dp(index,rem))

{

tra.push\_back({vec[index].first,vec[index].second}) ;

trace(index+1,rem-vv[index]);

}

else

trace(index+1,rem) ;

}

Dijkstra (shortest Path)

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

vector<pair<int, int>> adj[N];

long long dis[N];

int par[N];

void dijkstra(int src) {

memset(dis, '?', sizeof dis);

memset(par, -1, sizeof par);

dis[src] = 0;

priority\_queue<pair<long long, int>> pq;

pq.push({0, src});

while (pq.size()) {

long long d = -pq.top().first;

int u = pq.top().second;

pq.pop();

if (dis[u] != d) continue;

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

int v = adj[u][i].first, c = adj[u][i].second;

if (dis[u] + c < dis[v]) {

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

par[v] = u;

pq.push({-(dis[u] + c), v});

}

}

}

}

void go(int u) {

if (u == -1) return;

go(par[u]);

printf("%d ", u + 1);

}

int main() {

//freopen("meciul.in", "r", stdin);

//freopen("meciul.out", "w", stdout);

int n, m, src, dest;

scanf("%d%d", &n, &m);

src = 0, dest = n - 1;

for (int i = 0; i < n; ++i) adj[i].clear();

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

int u, v, c;

scanf("%d%d%d", &u, &v, &c);

--u, --v;

adj[u].push\_back({v, c});

adj[v].push\_back({u, c});

}

dijkstra(src);

if (dis[dest] == dis[n]) puts("-1");

else go(n - 1);

return 0;

}

Connected Component

vector<vector<int>>vec ;

int vis[sz] ;

int low [sz] ;

int id [sz] ;

int in\_stack[sz] ;

stack<int>ss ;

int scc ;

int timer ;

vector<int>ans ;

vector<int>temp ;

void dfs(int node)

{

vis[node]=1 ;

//cout<<node<<" " ;

in\_stack[node]=1 ;

ss.push(node) ;

low[node]=id[node]=timer++ ;

for(auto child:vec[node])

{

if(!vis[child])

{

dfs(child) ;

if(in\_stack[child]==1)

{

low[node]=min(low[node],low[child]) ;

}

}

else if (vis[child]==1&&in\_stack[child]==1)

{

low[node]=min(low[node],id[child]) ;

}

}

if(low[node]==id[node])

{

int x ;

scc++ ;

while(1)

{

x =ss.top() ;

temp.push\_back(x) ;

// cout<<x<<" "<<scc<<endl ;

ss.pop() ;

in\_stack[x]=0 ;

if(node==x)

{

break ;

}

} } }

Bridge Detection

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int n ,m;

vector<vector<int>>vec(1e5+5) ;

int vis[(int)1e5+5] ;

int in[(int)1e5+5] ;

int low[(int)1e5+5] ;

int cnt =0 ;

bool flag=true ;

vector<pair<int,int>>ans ;

void dfs(int node , int par)

{

vis[node]=1 ;

low[node]=in[node]=cnt++ ;

for(auto child :vec[node])

{

if(child==par)

{

continue ;

}

if(!vis[child])

{

dfs(child,node) ;

if(low[child]>in[node])

{

cout<<"there is a bridge"<<endl ;

flag=false ;

return ;

}

low[node]=min(low[child],low[node]) ;

}

else

{

low[node]=min(low[node],in[child]) ;

}

}

}

Accelration Point Detection

vector<vector<int>>vec(1e5+5) ;

int vis[(int)1e5];

int id [(int)1e5+5] ;

int low[(int)1e5+5] ;

int arrc[(int)1e5+5] ;

int timer ;

int arc\_point ;

void dfs(int node ,int par)

{

int children =0 ;

vis[node]=1 ;

low[node]=id[node]=timer++ ;

for(auto child :vec[node])

{

if(par==child)

continue ;

if(!vis[child])

{

dfs(child,node) ;

low[node]=min(low[node],low[child]) ;

if(id[node]<=low[child]&&par!=-1)

{

arrc[node]=1 ;

}

children++ ;

}

else

{

low[node]=min(low[node],id[child]) ;

}

}

if(par==-1&&children>1)

{

arrc[node]=1 ;

}

}

Max Flow

vector<vector<int>> adj;

int precalc[(int)1e3][(int)1e3] ;

int vis[(int) 1e3 + 5];

int pathlen;

int path[(int) 1e3 + 5];

int n;

int get\_path(int CurrNode, int EndNode, int len, int CurrMin) {

vis[CurrNode] = 1;

path[len] = CurrNode;

if (EndNode == CurrNode) {

pathlen = len + 1;

return CurrMin;

}

int ret = 0;

for (auto child:adj[CurrNode]) {

if (!vis[child]&&precalc[CurrNode][child]>0) {

ret = get\_path(child, EndNode, len + 1, min(CurrMin, precalc[CurrNode][child]));

}

if (ret > 0) {

break;

}

}

return ret;

}

int max\_flow(int src, int think) {

int total = 0;

while (true) {

memset(vis,0,sizeof vis) ;

pathlen =0 ;

int newflow= get\_path(src,think ,0,INT\_MAX) ;

if(!newflow)

{

break ;

}

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

{

int to =path[i] ;

int from=path [i-1] ;

precalc[from][to]-=newflow ;

precalc[to][from]+=newflow ;

}

total +=newflow ;

}

return total ;}

Trenary Search

long double l =0 ;

long double r =1e7 ;

for(int i=0 ;i<300 ;i++)

{

long double mid1 =(l\*2+r)/3.0 ;

long double mid2 = (l+r\*2)/3.0 ;

if(better(mid1)<better(mid2))

{

r=mid2;

}

else

{

l=mid1 ;

}

}

cout<<l<<endl ;

**Lowest\_Common\_Ancestor**

**#include <iostream>**

**#include<bits/stdc++.h>**

**using namespace std;**

**int LCA[(int) 2e5 + 5][33];**

**vector<vector<int>> adj;**

**int n;**

**int vis[(int) 2e5 + 5];**

**int level[(int) 2e5 + 5];**

**void dfs(int node, int par) {**

**LCA[node][0] = par;**

**for (auto child:adj[node]) {**

**if (par != child) {**

**dfs(child, node);**

**}**

**}**

**}**

**void bfs(int node) {**

**queue<pair<int, int>> q;**

**q.push({node, 0});**

**while (!q.empty()) {**

**auto pt = q.front();**

**q.pop();**

**level[pt.first] = pt.second;**

**// cout<<pt.first<<" "<<pt.second<<endl ;**

**vis[pt.first] = 1;**

**for (auto child:adj[pt.first]) {**

**if (!vis[child]) {**

**q.push({child, pt.second + 1});**

**}**

**}**

**}**

**}**

**void init() {**

**memset(LCA, -1, sizeof LCA);**

**dfs(1, -1);**

**memset(vis, 0, sizeof vis);**

**bfs(1);**

**for (int i = 1; i <= 32; i++) {**

**for (int j = 1; j <= n; j++) {**

**if (LCA[j][i - 1] != -1) {**

**int par = LCA[j][i- 1];**

**LCA[j][i] = LCA[par][i- 1];**

**}**

**}**

**}**

**}**

**int get\_lca(int u, int v) {**

**if (level[u] < level[v]) {**

**swap(u, v);**

**}**

**int d = level[u] - level[v];**

**while (d) {**

**int i = log2(d);**

**u = LCA[u][i];**

**d -= (1 << i);**

**}**

**if (u == v) {**

**return u;**

**}**

**for (int i = 32; i >= 0; i--) {**

**if (LCA[u][i] != -1 && (LCA[u][i] != LCA[v][i])) {**

**u = LCA[u][i];**

**v = LCA[v][i];**

**}**

**}**

**return LCA[u][0];**

**}**

**int main() {**

**cin >> n;**

**adj.resize(n + 2);**

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

**int u, v;**

**cin >> u >> v;**

**adj[u].push\_back(v);**

**adj[v].push\_back(u);**

**}**

**init();**

**}**