This is not a codebook

June 3, 2021

Contents

1 A Hello world

1.1 Aloha

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

/* compile command */
g++ -std=c++14 -O2 -Wall -Wextra test.cpp -o test
/* script */
#!/bin/bash
g++ -std=c++14 -O2 -Wall -Wextra $1
/* compile script*/
chmod +x build
/* execute */
build test.cpp

/* cin cout */
ios::sync_with_stdio(false);
cin.tie(0); // endl -> '\n'

/* INF */
#define INF 0x3f3f3f3f // int
#define INF 0x3f3f3f3f3f3f3f3f // long long

/* bit */
p(k) denotes the largest power of two that devides k
p(k) = k & -k;
```

2 B Useful

2.1 ExGCD

```
// O(log(min(a,b)))
/* ax + by = gcd(a,b) */

tuple<int,int,int> exgcd(int a, int b){
    if(b == 0) return {1,0,a};
    else{
        int x, y, g;
        tie(x, y, g) = gcd(b, a%b);
        return {y, x-(a/b)*y, g};
    }
}

/*
to calculate a / b = ans (% MOD)
=> find b^(-1), then a * b^(-1) = ans (% MOD)
-----------------------------------------------
to find b^(-1), there are two methods

1.  Fermats  Little Theorem
* MOD is a prime and b is not divisible by MOD
=> find b^(MOD-2) with Fast Power

2.  Bezouts  Theorem
* gcd(b,MOD) == 1
=> find x with exgcd(b,MOD)
*/
```

2.2 Fast Power

```
// O(log exp)
// MOD

ll pw(ll x, ll y){
    ll ans = 1;
    while(y){
        if(y&1) ans *= x;
        x *= x;
        y >>= 1;
    }
    return ans;
}
```

2.3 GCD

```
// O(log(min(a,b)))

ll gcd(ll a, ll b){
    return b == 0? a : gcd(b,a%b);
}
```

2.4 LCM

```
// O(log(min(a,b)))

ll lcm(ll a, ll b){
    return a*b / gcd(a,b);
}
```

2.5 Prime

```
#define MAX_SIZE 1000000 //1e6

bool is_prime[MAX_SIZE];
vector<ll> primes;

void prime(){
    fill(is_prime, is_prime+MAX_SIZE, true);
    is_prime[0] = is_prime[1] = false;
    for(ll i = 2; i < MAX_SIZE; i++){
        if(is_prime[i]){
            primes.push_back(i);
            for(ll j = i*i; j < MAX_SIZE; j+=i){
                is_prime[j] = false;
            }
        }
    }
}
```

3 C Graph

3.1 BFS and DFS

3.1.1 BFS

```
// O(M+N)
// keep parent to find path
int bfs(int s,int t){
    fill(dis, dis+MAX_N, -1);
    queue<int> q;
    dis[s] = 0;
    q.push(s);
    while(!q.empty()){
        int now = q.front();q.pop();
        for(int u:adj[now]){
            if(dis[u] != -1) continue;
            dis[u] = dis[now] + 1;
            q.push(u);
        }
    }
    return dis[t];
}
```

3.1.2 DFS-Path

```
void dfs_path(int now){
    path.push_back(now);
    vis[now] = 1;
    for(auto u:v[now]){
        if(vis[t]) return;
        if(!vis[u]) dfs_path(u);
    }
    if(!vis[t]) path.pop_back();
}
```

3.1.3 DFS

```
// O(M+N)
// Cycle Detection : a neighbor has been visited and not the
     parent of current node
// Bipartiteness Check : no adjacent nodes with the same
    color
void dfs(int now){
    vis[now] = 1;
    for(auto x:adj[now]){
        if(!vis[x]) dfs(x);
    }
}
```

3.2 DAG

3.2.1 Successor Graph

```
// O(nlogu) for build, u is MAX_STEP
// O(logk) for go

void init(){
    for(int i = 1; i <= n; i++){
        cin >> succ[0][i];
    }
}

void build(){
    for(int i = 1; i < 35; i++){ // i <= logu
        for(int j = 1; j <= n; j++){
            succ[i][j] = succ[i-1][succ[i-1][j]];
        }
    }
}

int go(int now, int k){
    int x = 0;
    while(k != 0){
        if(k&1) now = succ[x][now];
        k >>= 1;
        x++;
    }
    return now;
}
```

3.2.2 Topological Sorting

```
// O(m+n)

void dfs(int now){
    if(cycle) return;
    vis[now] = 1; // processing
    for(auto x:adj[now]){
        if(vis[x] == 1) cycle = 1;
        if(!vis[x]) dfs(x);
    }
    vis[now] = 2; // processed
    order.push_back(now);
}

void Topological_sort(){
    for(int i = 1; i <= n && !cycle; i++){
        if(!vis[i]) dfs(i);
    }
    if(cycle){
        cout << "IMPOSSIBLE" << endl;
    }
    else{
        reverse(order.begin(),order.end());
        for(auto x:order){
            cout << x << ' ';
        }
    }
}
```

3.3 Disjoint Set

```
//O(alpha(N))

int p[MX_N], sz[MX_N]

void init(){
    for(int i = 0; i < MX_N; i++){
        p[i] = i;
        sz[i] = 1;
    }
}

int f(int x){
    if(p[x] == x) return x;
    return p[x] = f(p[x]);
}

void unite(int a, int b){
    a = f(a);
    b = f(b);
    if(sz[a] < sz[b]) swap(a,b);
    p[b] = a;
    sz[a] += sz[b];
}

bool same(int a, int b){
    return f(a) == f(b);
}
```

3.4 Eulerian Path not sure

```
/* undirected */
int a, b, id, degree[MX_N];
vector<pair<int,int>> adj[MX_N]; // b id
bool used[MX_M];
/* directed */
int a, b, out[MX_N], in[MX_N];
vector<int> adj[MX_N];

int s, t;
vector<int> path;

void init(){
    for(int i = 0 ; i < m; i++){
        cin >> a >> b;
        /* undirected*/
        adj[a].push_back({b,i});
        adj[b].push_back({a,i});
        degree[a]++; degree[b]++;
        /* directed */
        adj[a].push_back(b);
        out[a]++; in[b]++;
    }
}

bool is_able(){
    /* undirected */
```

```
    int cnt_odd = 0;
    for(int i = 1; i <= n; i++){
        if(degree[i] % 2) cnt_odd++, s = i;
        if(cnt_odd > 2) return 0;
    }
    return cnt_odd==0 || cnt_odd==2;
    // the former is also Eulerian circuit

    /* directed */
    int cnt_s = 0, cnt_t = 0;
    for(int i = 1; i <= n; i++){
        if(in[i] > out[i]+1 || out[i] > in[i]+1) return 0;
        if(out[i] == in[i]+1) cnt_s++, s = i;
        if(in[i] == out[i]+1) cnt_t++, t = i;
    }
    return (cnt_s==0 && cnt_t==0) || (cnt_s==1 && cnt_t==1);
    // the former is also Eulerian circuit
}

void dfs(int now){
    while(!adj[now].empty()){
        b = adj[now].back().first;
        id = adj[now].back().second; // undirected
        adj[now].pop_back();
        if(used[id]) continue; // undirected
        used[id] = 1; // undirected
        dfs(b);
    }
    path.push_back(now);
}

bool all(){
    for(int i = 1; i <= n; i++){
        if(!adj[i].empty()) return 0;
    }
    return 1;
}

bool Euler(){
    init();
    if(is_able()){
        dfs(s);
        if(all()) {reverse(path.begin(),path.end()); return
             1;}
        else return 0;// no Euler Path
    }
    else return 0;// no Euler Path
}
```

3.5 Max Flows and Min Cuts

3.5.1 Ford-Fulkerson

```
// O(?)

#define to first.first
#define cap first.second
#define rvsid second

vector<pair<pair<int,ll>,int>> adj[MAX_N];
vector<pair<int,int>> cuts;

void init(){
    adj[a].push_back({{b,w},adj[b].size()});
    /* undirected */
    adj[b].push_back({{a,w},adj[a].size()-1});
    /* directed */
    adj[b].push_back({{a,0},adj[a].size()-1});
}

ll dfs(int now, ll flow){
    if(now == t) return flow;
    vis[now] = 1;
    ll res;
    for(auto &x:adj[now]){ // reference!!
        if(vis[x.to] || x.cap == 0) continue;
        if(res = dfs(x.to,min(flow,x.cap))){
            x.cap -= res;
            adj[x.to][x.rvsid].cap += res;
            return res;
        }
    }
    return 0;
}

void max_flow(){
    ll res, ans = 0;
    while(res = dfs(s,INF)){
        ans += res;
        fill(vis,vis+n+1,0);
    }
    return ans;
}

void find_cuts(){ // last dfs s can reach i but not adj[i]
    for(int i = 1; i <= n; i++){
        if(vis[i]){
            for(auto x:adj[i]){
                if(!vis[x.to]) cuts.push_back({i,x.to});
            }
        }
    }
}
```

3.6 Minimum Spanning Tree

```
// O(mlogn) after sorting O(mlogm)
vector<pair<ll,pair<int,int>>> edge; // w a b
int cnt = 0; // exactly n-1 edges have to be added

// Kruskal
ll MST(){
    init(); // Union-Find init
    sort(edge.begin(),edge.end());
    for(int i = 0; i < m && cnt < n; i++){
        a = edge[i].second.first;
        b = edge[i].second.second;
        w = edge[i].first;
        if(same(a,b)) continue;
        cnt++;
        ans += w;
        unite(a,b);
    }
    return cnt==n-1? ans: INF;
}
```

3.7 SCC

3.7.1 Giant Pizza(2-SAT)

```
/*
(x1 || x2) && ... && (xi || xj)
build !x1 -> x2 , !x2 -> x1 ... !xi -> xj , !xj -> xi
*/

#include<bits/stdc++.h>
using namespace std;

#define F first
#define S second

int m, n, a, b, c, d, ans[100005], gp[100005][2], cnt;
char C, D;
vector<pair<int,int>> adj[100005][2], rvs[100005][2], order;
bool vis[100005][2];
```

```
void dfs(pair<int,int> now){
    vis[now.F][now.S] = 1;
    for(auto x:adj[now.F][now.S]){
        if(!vis[x.F][x.S]) dfs({x.F,x.S});
    }
    order.push_back({now.F,now.S});
}

void rvsdfs(pair<int,int> now){
    gp[now.F][now.S] = cnt;
    for(auto x:rvs[now.F][now.S]){
        if(!gp[x.F][x.S]) rvsdfs({x.F,x.S});
    }
}

void ansdfs(pair<int,int> now){
    //cout << now.F << ' ' << now.S << endl;
    vis[now.F][now.S] = 1;
    ans[now.F] = now.S;
    for(auto x:adj[now.F][now.S]){
        if(!vis[x.F][x.S]) ansdfs({x.F,x.S});
    }
}

void Kosaraju(){
    for(int i = 1; i <= n; i++){
        if(!vis[i][0]) dfs({i,0});
        if(!vis[i][1]) dfs({i,1});
    }
    for(int i = order.size()-1; i >= 0; i--){
        if(!gp[order[i].F][order[i].S]){
            cnt++;
            rvsdfs({order[i].F,order[i].S});
        }
    }
}

bool contradiction(){
    for(int i = 1; i <= n; i++){
        if(gp[i][0] != 0 && gp[i][0] == gp[i][1]) return 1;
    }
    return 0;
}

int main(){
    cin >> m >> n;
    for(int i = 0; i < m; i++){
        cin >> C >> a >> D >> b;
        if(C == '+') c = 1;
        else c = 0;
        if(D == '+') d = 1;
        else d = 0;
        adj[a][!c].push_back({b,d});
        adj[b][!d].push_back({a,c});
        rvs[b][d].push_back({a,!c});
        rvs[a][c].push_back({b,!d});
    }
    Kosaraju();
    if(contradiction()){
        cout << "IMPOSSIBLE" << endl;
    }
    else{
        for(int i = 1; i <= n; i++){
            vis[i][0] = vis[i][1] = 0;
        }
        for(int i = 1; i <= n; i++){
            if(!vis[i][0] && !vis[i][1]) ansdfs({i,0});
            if(ans[i] == 0) cout << "- ";
            else cout << "+ ";
        }
    }
    return 0;
}
```

3.7.2 Kosaraju

```
// O(m+n)
int id, gp[MX_N];
vector<int> adj[MX_N], rvsadj[MX_N], sccadj[MX_N], order;

void init(){
    adj[a].push_back(b);
    rvsadj[b].push_back(a);
}

void rvsdfs(int now){
    vis[now] = 1;
    for(auto x:rvsadj[now]){
        if(!vis[x]) rvsdfs(x);
    }
    order.push_back(now);
}

void dfs(int now){
    gp[now] = id;
    for(auto x:adj[now]){
        if(!gp[x]) dfs(x);
        else if(gp[x] != id) sccadj[id].push_back(gp[x]);
    }
}

void Kosaraju(){
    init();
    for(int i = 1; i <= n; i++){
        if(!vis[i]) rvsdfs(i);
    }
    reverse(order.begin(),order.end());
    for(auto x:order){
        if(!gp[x]) id++,dfs(x);
    }
}
```

3.8 Shortest Path

3.8.1 Bellman-Ford

```
//O(mn)
/* Detect Negative Cycles */
vector<tuple<int, int, ll>> edge; //a b w
ll dis[MX_N];

// negative cycles might not exit between s and t
// to check connection to start node, skip INF node
// to check connection to terminal node, DFS

//return whether negative cycles exist
bool Bellman_Ford(int s = 1, int t = n){
    fill(dis, dis+n+1, INF);
    dis[s] = 0;
    for(int i = 0; i < n-1; i++){
        for(auto e: edge){
            tie(a, b, w) = e;
            //if(dis[a] == INF) continue;
            dis[b] = min(dis[b], dis[a]+w);
        }
    }
    for(auto e: edge){
        tie(a, b, w) = e;
        //if(dis[a] == INF) continue;
        if(dis[a]+w < dis[b]) return 1; // or DFS(b) and vis[
            t];
    }
    return 0;
}
```

3.8.2 Dijkstra

```
// O(n + mlogm)
/* Only Non-negative weights*/

void Dijkstra(int s){
    priority_queue<pli,vector<pli>,greater<pli>> pq;
    fill(dis,dis+n+1,INF);
    dis[s] = 0;
    pq.push({0,s});
    while(!pq.empty()){
        a = pq.top().second;
        pq.pop();
        if(processed[a]) continue;
        processed[a] = 1;
        for(auto x:adj[a]){
            b = x.first;
            w = x.second;
            if(dis[a]+w < dis[b]){
                dis[b] = dis[a] + w;
                pq.push({dis[b],b});
            }
        }
    }
}
```

3.8.3 Floyd-Warshall

```
// O(n^3)

void init(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i != j) dis[i][j] = INF;
        }
    }
}

void Floyd_Warshall(){
    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                dis[i][j] = min(dis[i][j], dis[i][k]+dis[k][j
                    ]);
            }
        }
    }
}
```

4 D Tree

4.1 LCA

```
// O(logn) after build O(nlogn)

int p[logMX_N][MX_N], dep[MX_N];

void dfs(int now, int par, int level){
    p[0][now] = par;
    dep[now] = level;
    for(auto x:adj[now]){
        if(x == par) continue;
        dfs(x,now,level+1);
    }
}

void init(){
    dfs(r,0,0);
    build(); // build p[][] as successor graph
}

int lca(int a, int b){
    if(dep[a] > dep[b]) swap(a,b);
    b = go(b,dep[b]-dep[a]);
    if(a == b) return a;
    for(int i = logMX_N-1; i >= 0; i--){
        if(p[i][a] != p[i][b]){
            a = p[i][a];
            b = p[i][b];
        }
    }
    return p[0][a];
}

int dis(int a, int b){
    return dep[a] + dep[b] - 2*dep[lca(a,b)];
}
```

5 Z Others

5.1 graph

5.1.1 Bellman-Ford

```
#include <tuple>

tuple<int, int, int> edge[10005];
long long dis[1003];

void bellman_Ford(){
    for(int i=0;i<=n;i++)
        dis[i] = INF;
    dis[1] = 0;
    for(int i =0;i<n;i++){
        for(int j = 0;j<m;j++){
            tie(a, b, w) = edge[j];
            dis[b] = min(dis[b], dis[a]+w);
        }
    }
    // do one more times to found negative cycles
    // negative cycles might not connect to start
}
```

5.1.2 BFS

```
//O(N+M) untested
#include<bits/stdc++.h>
using namespace std;
#define MaxSize 2010

vector< vector<int> > adj;
int dis[MaxSize];
//remember to initial graph vis

int BFS(int s,int t){
    queue<int> q;
    dis[s] = 0;
    q.push(s);
    int keep;
    while(!q.empty()){
        keep = q.front();q.pop();
        for(int u:adj[keep]){
            if(dis[u] != -1) continue;
            dis[u] = dis[keep] + 1;
            q.push(u);
        }
    }
    return dis[t];
}

int main(){
    int n,m;
    cin>>n>>m;
    adj.resize(n+5);
    int a,b;
```

```
    for(int i = 0;i<m ;i++){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    for(int i = 0;i<=n;i++){
        dis[i] = -1;
    }
    cin>>a>>b;
    cout<<BFS(a, b)<<endl;
    return 0;
}
```

5.1.3 BFS$_{path}$

```
//O(M+N)

vector< vector<int> > adj;

void BFS_path(){
    queue<int> q;
    q.push(1); //q.push(start)
    while(!q.empty()){
        keep = q.front();
        q.pop();
        if(keep == n){// keep == end
            flag = true;
        }
        for(int i=0;i<adj[keep].size();i++){
            if(vis[adj[keep][i]] == 0){
                vis[adj[keep][i]] = keep;
                //
                q.push(adj[keep][i]);
            }
        }
    }
    keep = n;//
    vector<int> v;//
    while(keep != 1){
        v.push_back(keep);
        keep = vis[keep];
    }
    v.push_back(1); // v
}
```

5.1.4 DFS

```
// O(M+N) untested

vector< vector<int> > adj;
bool vis[MaxSize];

//remember to initial adj vis
void dfs(int x){
    vis[x] = true;
    for(int e:adj[x]){
        if(!vis[e]) dfs(e);
    }
}
```

5.1.5 DFS$_{path}$

```
// O(M+N) untested

vector< vector<int> > adj;
bool vis[MaxSize];
vector<int> path;//path[0] = start

// remember to initial adj vis path
void dfs_path(int x){
    if(x == terminal){
        cout<<path<<endl;
        return;
    }
    if(vis[x]) return;
    vis[x] = true;
    for(int e:adj[x]){
        path.push_back(e);
        dfs(e);
        path.pop_back(e);
    }
}
```

5.1.6 dijkstra's$_{algorithm}$

```
//O(N + MlogM)
//N: number of nodes
//M: number of edges
typedef pair<long long, int> plli;
#define INF 0x3f3f3f3f3f3f3f3f

vector< vector<pair<int, int> >> adj;
int path[100005]; // void print_path(int s, int t);
long long dis[100005];
bool vis[100005];

void dijkstra(int n, int start){
    for(int i=0; i<=n;i++){
        dis[i] = INF;
        vis[i] = false;
    }
    dis[start] = 0;
    // path[start] = -1;

    priority_queue<plli, vector<plli>, greater<plli> > pq;
    pq.push({0, start});
    long long w;
    int node, b;
    while(!pq.empty()){
        node = pq.top().second;
        pq.pop();
        if(vis[node]){continue;}
        vis[node] = true;
        for(auto u:adj[node]){
            b = u.first, w = u.second;
            if(dis[b] > dis[node] + w){
                dis[b] = dis[node] + w;
                // path[b] = node;
                pq.push(dis[b], b);
            }
        }
    }
}
```

5.1.7 dijkstra's$_{algorithm_is_it_correct}$

```
//O(N + MlogM)
//N: number of nodes
//M: number of edges
typedef pair<long long, int> plli;
#define INF 0x3f3f3f3f3f3f3f3f

vector< vector<pair<int, int> >> adj;
int path[100005];
long long dis[100005];

void dijkstra(int n, int start){
    for(int i=0; i<=n;i++){
        dis[i] = INF;
    }
    priority_queue<plli, vector<plli>, greater<plli> > pq;
    pq.push({0, start});
    long long distance, w;
    int node, b;
```

```
    while(!pq.empty()){
        node = pq.top().second;
        distance = pq.top().first;
        pq.pop();
        if(dis[node] != INF){continue;}
        dis[node] = distance;
        for(auto u:adj[node]){
            b = u.first, w = u.second;
            if(dis[b] > dis[node] + w){
                dis[b] = dis[node] + w;
                pq.push(b, dis[node] + w);
            }
        }
    }
}
```

5.1.8 djset

```
//O(alpha(N))

int djset[100005];
int treesize[100005];

void build(int n){
    for(int i=0;i<=n;i++){
        djset[i] = i;
        treesize[i] = 1;
    }
}

int findBoss(int x){
    if(djset[x]== x){
        return x;
    }
    return djset[x]=findBoss(djset[x]);
}

void combine(int a,int b){
    a = findBoss(a);
    b = findBoss(b);
    if(a == b) return;
    int temp;
    if(treesize[a] < treesize[b]){
        temp = a;
        a = b;
        b = temp;
    }
    djset[b] = a;
    treesize[a] += treesize[b];
}

bool same(int a,int b){
 return findBoss(a)==findBoss(b);
}
```

5.1.9 Floyd$_{W}arshall$

```
//O(N*N*N)
// find the shortest path for each pair
// tested
long long dis[510][510];

void init(int n){
    for(int i=0;i<=n;i++){
        for(int j=0;j<=n;j++){
            dis[i][j] = INF;
        }
        dis[i][i] = 0;
    }
}

void Floyd(int n){
    for(int i=1;i<=n;i++){
        for(int j=1; j<=n;j++){
            for(int k=1;k<=n;k++){
                dis[j][k] = min(dis[j][k], dis[j][i] + dis[i][
                    k]);
            }
        }
    }
}
```

5.2 math

5.2.1 CONVEXHULL

```
#include <bits/stdc++.h>
using namespace std;
// the same angle problem.
// reference: https://www.youtube.com/watch?v=B2AJoQSZf4M
typedef long long int lld;
lld n;
struct point
{
   lld x,y,id;
}p[100006];
stack<point>dots;
lld smallest_id = 0;
point next_to_top();
bool cmp (point a, point b);
lld count_clockwise(int id, point top, point top_next);
int main ()
{
    cin>>n;
    for(int i = 0; i < n; i++){
        cin>>p[i].x>>p[i].y;
        p[i].id = i+1;
        if(p[smallest_id].y == p[i].y){ //find the lowest y-
            coordinate and leftmost point, called P0
            if(p[smallest_id].x > p[i].x)
                smallest_id = i;
        }
        else if(p[smallest_id].y > p[i].y)
            smallest_id = i;
    }
    swap(p[0], p[smallest_id]); // p[0] is the lowest point
        and the leftmost of the same y coordinate.
    sort(p+1, p+n, cmp);
    // we have to do something to keep the farthest distance
        point.
    // if we sort the same angle by distance, and the longest
        distance be the next

    for(int i = 0; i < n; i++){
        while(dots.size() >= 2 && count_clockwise(i, dots.top
            (), next_to_top()) <= 0) // when count_clockwise
            == 0, we can replace the longer distance point
            to the array.
            dots.pop();
        dots.push(p[i]);
    }
    cout<<dots.size()+1<<endl; // all the vertex and the
        start point itself.
    cout<<p[0].id<<" ";
    while(!dots.empty())
    {
        cout<<dots.top().id<<" ";
        dots.pop();
    }
    cout<<endl;
}
bool cmp (point a, point b)
{
    lld x1 = a.x - p[0].x;
    lld y1 = a.y - p[0].y;
    lld x2 = b.x - p[0].x;
```

```
    lld y2 = b.y - p[0].y;
    lld z = x1*y2-x2*y1;
    if(z == 0){
        return x1*x1 + y1*y1 < x2*x2 + y2*y2;
    }
    return z > 0;
}
point next_to_top()
{
    point tmp = dots.top();
    dots.pop();
    point top_next = dots.top();
    dots.push(tmp);
    return top_next;
}
lld count_clockwise(int id, point top, point top_next) //
     actually we do the cross product XD.
{
    lld x1 = top.x - p[id].x;
    lld y1 = top.y - p[id].y;
    lld x2 = top.x - top_next.x;
    lld y2 = top.y - top_next.y;
    return x1*y2 - x2*y1;
}
```

5.2.2 Exgcd

```
//O(logN)
//find ax + by = gcd(a, b); use in find inverse in modular

//two way to find s*t %m = 1
//     1.if m is a prime and gcd(s, m) == 1 --> Fermats
     Little Theorem
//            If p is prime and a is an integer not
     divisible by p, then a^(p-1)%p = 1
//            find a^(p-2) with fast exponotial
<<<<<<< HEAD
//     2.if gcd(s, m) == 1 -->Bezouts Theorem
=======
//     2.if gcd(s, m) == 1 -->Bezouts Theorem
>>>>>>> a09791fe2f6f9a9dc666dc0f749beae5d65b5098
//            If a and b are positive integers,
//            then there exist integers s and t such that
     gcd(a,b) = sa + tb.
//            make a = s, b = m, then t = x;


int ex_gcd(long long a, long long b, long long &x, long long
      &y){
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }
    long long d = ex_gcd(b, a%b, x, y);
    long long temp = y;
    y = x - y*(a/b);
    x = temp;
    return d;
}
```

5.3 tree

5.3.1 little$_s pan_t ree_d jset$

```
//two way to find little span tree
//     1.Kruskal AKA disjion set:
//          choose the two nodes are not connected and with
     the shortest edge
//     2.Prim:
//          choose the node which is closest to the tree and
     add it in the tree
#include<iostream>
#include<algorithm>
using namespace std;

// the data structure of disjion set
int djset[100005];

struct Edge{
    int s, t, w;
};

Edge edges[200005];

bool cmp(Edge a, Edge b){
    if(a.w != b.w) return a.w < b.w;
    if(a.s != b.s) return a.s < b.s;
    return a.t < b.t;
}

long long way1(int n, int m){
    sort(edges, edges + m, cmp);
    build(n+5);
    long long sum = 0;
    for(int i = 0;i<m;i++){
        if(!same(edges[i].s, edges[i].t)){
            combine(edges[i].s, edges[i].t);
            sum += edges[i].w;
        }
    }
    bool flag = false;
    for(int i =1;i<n;i++){
        if(!same(0, i)){
            flag = true;
            break;
        }
    }
    if(flag) return -1;
    return sum;
}
```

5.3.2 little$_s pan_t ree_p rim$

```
#include<iostream>
#include<queue>
using namespace std;

vector< vector<pair<int, int>> > adj;
vector<bool> vis;

long long Prim(int n){
    // n: number of nodes
    vis.resize(n+5);
    for(int i =0;i<n;i++){
        vis[i] = false;
    }
    priority_queue<pair<int,int>, vector<pair<int, int>>,
         greater<pair<int, int>>> pq;
    pq.push({0, 0});
    int wei, node;
    long long sum = 0;
    while(!pq.empty()){
        wei = pq.top().first;
        node = pq.top().second;
        pq.pop();
        if(vis[node]) continue;
        sum += wei;
        vis[node] = true;
        for(auto e:adj[node]){
            if(!vis[e.first]){
                pq.push({e.second, e.first});
            }
        }
    }
    bool flag = false;
    for(int i =0;i<n;i++){
```

```
        if(!vis[i]) {
            flag = true;
        }
    }
    return (flag ? -1:sum);
}

int main(){
    int n,m;
    while(cin>>n>>m){
        adj.clear();
        adj.resize(n+5);
        int s, t, w;
        for(int i=0;i<m;i++){
            cin>>s>>t>>w;
            adj[s].push_back({t,w});
            adj[t].push_back({s,w});
        }
        cout<<Prim(n)<<endl;
    }
    return 0;
}
```

5.3.3 LowCommonAnesctorWay1

```
// find the deepest subtree's root
// which contains two node
// the distance of two node = deep[a] + deep[b] - 2*deep[c]
// two way to solve the problem
// O(height)
#define maxSize 30004
#define root 1
int deep[maxSize];
int parentid[maxSize];
vector< vector<int> > adj;

void init_aka_DFS(int node){
    // before call it make root's deep = 1
    // and call DFS(root)
    for(int u:adj[node]){
        deep[u] = deep[node]+1;
        parentid[u] = node;
        DFS(u);
    }
}

int way1(int a, int b){
    while(deep[a] > deep[b]){
        a = parentid[a];
    }
    while(deep[a] < deep[b]){
        b = parentid[b];
    }
    while(a != b){
        a = parentid[a];
        b = parentid[b];
    }
    return a;
}
```

5.3.4 segment$_t$ree

```
#include<iostream>
using namespace std;
#define MaxSize 200005
#define EdgeStatuation 1000000009

int a[MaxSize];

struct Node{
    int left, right;
    int val;
}tree[4*MaxSize];

int pull(int x, int y){
    //think of divide and conquer
    return min(x, y);
}
// root : idx = 1
void build(int idx, int L, int R){
    tree[idx].left = L;
    tree[idx].right = R;
    if(L == R){
        tree[idx].val = a[L];
        return;
    }
    int M = (L + R)/2;
    build(idx*2, L, M);
    build(idx*2+1, M+1, R);
    tree[idx].val = pull(tree[idx*2].val, tree[idx*2+1].val);
}

int query(int idx, int qL, int qR){
    if(tree[idx].right < qL || tree[idx].left > qR){
        return EdgeStatuation;
    }
    if(tree[idx].left >= qL && tree[idx].right <= qR){
        return tree[idx].val;
    }
    return pull(query(idx*2, qL, qR), query(idx*2+1, qL, qR))
        ;
}

void update(int idx, int pos, int modify){
    if(tree[idx].right < pos || tree[idx].left > pos){
        return;
    }
    if(tree[idx].right == pos && tree[idx].left == pos){
        tree[idx].val = modify;
        return;
    }
    update(idx*2, pos, modify);
    update(idx*2+1, pos, modify);
    tree[idx].val = pull(tree[idx*2].val, tree[idx*2+1].val);
}
```

5.3.5 segment$_t$ree$_a$nother

```
#include<iostream>
using namespace std;
#define maxSize 200005

// different segment tree

const int edge_situation = 0;
struct Node{
    int L, R;
    long long val;
}tree[4*maxSize];

int a[maxSize];

void build(int idx, int L, int R){
    tree[idx].L = L;
    tree[idx].R = R;
    if(L == R){
        tree[idx].val = a[L];
        return ;
    }
    int M = (L+R)/2;
    build(idx*2, L, M);
    build(idx*2+1, M+1, R);
}

long long query(int idx, int pos){
    if(tree[idx].L > pos || tree[idx].R < pos){
        return edge_situation;
```

```
    }
    if(tree[idx].L == pos && tree[idx].R == pos){
        return tree[idx].val;
    }
    int M = (tree[idx].L+ tree[idx].R)/2;
    if(pos <= M){
        return tree[idx].val + query(idx*2, pos);
    }
    return tree[idx].val + query(idx*2+1, pos);
}

void update(int idx, int uL, int uR, int modify){
    if(tree[idx].R < uL || tree[idx].L > uR){
        return;
    }
    if(tree[idx].L >= uL && tree[idx].R <= uR){
        tree[idx].val += modify;
        return;
    }
    update(idx*2, uL, uR, modify);
    update(idx*2+1, uL, uR, modify);
}

int main(){
     int n, q;
     while(cin>>n>>q){
           for(int i=0;i<n;i++){
               cin>>a[i];
           }
           build(1, 0, n);
           int k, x, y, u;
           while(q--){
               cin>>k;
               if(k == 1){
                   cin>>x>>y>>u;
                   update(1, x-1, y-1, u);
               }
               else{
                   cin>>k;
                   cout<<query(1, k-1)<<'\n';
               }
           }
     }
    return 0;
}


/*
question:
https://cses.fi/problemset/task/1651/

Given an array of n integers,
your task is to process q queries of the following types:
    1:increase each value in range [a,b] by u
    2:what is the value at position k?
*/
```

5.3.6 tree_{bfs}

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;

vector< vector<int> > adj;
vector<int> dis;
vector<int> parent;
int n;//n nodes

void init(void){
    for(int i = 0;i<=n;i++){
        dis[i] = -1;
        parent[i] = -1;
    }
}

// find diameter use twice BFS
// BFS return farthest node from start point

int BFS(int start){
    queue<int> q;
    init();
    int now = start;
    q.push(start);
    dis[start] = 0;
    while(!q.empty()){
        now = q.front();
        q.pop();
        for(int u:adj[now]){
            if(dis[u] == -1){
                dis[u] = dis[now] + 1;
                parent[u] = now;
                q.push(u);
            }
        }
    }
    return now;
}

int main(){
    int a, b;
    cin>>n;
    adj.resize(n+5);
    dis.resize(n+5);
    parent.resize(n+5);
    int m = n;
    while(m-- > 1){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    int P = BFS(BFS(1));
    //find diameter
    //cout<<dis[P]<<endl;
    //find center;
    int diameter = dis[P];
    for(int i = 0;i< diameter/2;i++){
        P = parent[P];
    }
    if(diameter %2 && parent[P] < P ){
        P = parent[P];
    }
    cout<<P<<endl;
    return 0;
}
```

5.3.7 $\text{tree}_{center_{bottom_{up}}}$

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;

vector< vector<int> > adj;
int edgecnt[200005];
int dis[200005];

int tree_center(int n){
    //found the longest
    //shortest path from two points on tree
    if(n == 1) {
        // only the one node
        // center = 1
        return 1;
    }
    for(int i = 0;i<=n;i++){
```

```
        //initial
        dis[i] = -1;
    }
    queue<int> q;
    for(int i =1;i<=n; i++){
        edgecnt[i] = adj[i].size();
        if(edgecnt[i] == 1){
            // find leaves
            dis[i] = 0;
            q.push(i);
        }
    }
    int last = 1;
    bool flag = false;
    while(!q.empty()){
        last = q.front();
        q.pop();
        for(int u:adj[last]){
            //remove the node for every node
            //connected to the remove node
            edgecnt[u] -= 1;
            if(edgecnt[u] == 1){
                dis[u] = dis[last] +1;
                q.push(u);
            }
        }
    }
    //inspect the node connected to last
    //for same dis[] (case : O-O)
    for(int u:adj[last]){
        if(dis[u] == dis[last]){
            // two center change when question diverse
            last = min(last, u);
            flag = true;
            break;
        }
    }
    return last;
}

int main(){
    int n;
    int a, b;
    cin>>n;
    adj.resize(n+5);
    int m = n;
    while(m-- > 1){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    cout<<tree_center(n)<<endl;
    return 0;
}
```

5.3.8 tree$_d$iameter$_b$uttom$_u$p

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;

vector< vector<int> > adj;
int edgecnt[200005];
int dis[200005];

int tree_diameter(int n){
    //found the longest
    //shortest path from two points on tree
    if(n == 1) {
        // only the one node
        //diameter = 0
        return 0;
    }
    for(int i = 0;i<=n;i++){
        //initial
        dis[i] = -1;
    }
    queue<int> q;
    for(int i =1;i<=n; i++){
        edgecnt[i] = adj[i].size();
        if(edgecnt[i] == 1){
            // find leaves
            dis[i] = 0;
            q.push(i);
        }
    }
    int last = 1;
    bool flag = false;
    while(!q.empty()){
        last = q.front();
        q.pop();
        for(int u:adj[last]){
            //remove the node for every node
            //connected to the remove node
            edgecnt[u] -= 1;
            if(edgecnt[u] == 1){
                dis[u] = dis[last] +1;
                q.push(u);
            }
        }
    }
    //inspect the node connected to last
    //for same dis[] (case : O-O)
    for(int u:adj[last]){
        if(dis[u] == dis[last]){
            flag = true;
            break;
        }
    }
    if(flag) return 2 *dis[last] +1;
    return 2*dis[last];
}

int main(){
    int n;
    int a, b;
    cin>>n;
    adj.resize(n+5);
    int m = n;
    while(m-- > 1){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    cout<<tree_diameter(n)<<endl;
    return 0;
}
```