This is not a codebook

September 30, 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<ll,ll,ll> exgcd(ll a, ll b){
    if(b == 0) return {1,0,a};
    else{
        ll x, y, g;
        tie(x, y, g) = exgcd(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 Articulation points

```
int n, m, order = 1, cnt, dfn[200005], up[200005], rt_child;

void init(){
    for(int i = 1; i <= n; i++){
        up[i] = INF;
    }
}

void dfs(int now){
    up[now] = dfn[now] = order++;
    for(auto x:adj[now]){
        if(x == p) continue;
        if(dfn[x] == 0){
            dfs(x,now);
            up[now] = min(up[now],up[x]);

            /* points*/
            if(now == 1) rt_child++;
            else if(up[x] >= dfn[now]) ans[now] = 1;

            /* bridges */
            if(up[x] > dfn[now]) ans.push_back({now,x});
```

```
        }
        else up[now] = min(up[now],dfn[x]);
    }
}

int main(){
    dfs(1,0);
    if(rt_child > 1) ans[1] = 1;
}
```

3.2 BFS and DFS

3.2.1 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];
}
```

3.2.2 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();
}
```

3.2.3 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);
    }
}
```

3.3 DAG

3.3.1 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;
}
```

3.3.2 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 << ' ';
        }
    }
}
```

3.4 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);
}
```

3.5 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
}
```

3.6 Max Flows and Min Cuts

3.6.1 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});
            }
        }
    }
}
```

3.7 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;
}
```

3.8 SCC

3.8.1 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;
}
```

3.8.2 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);
    }
}
```

3.9 Shortest Path

3.9.1 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;
}
```

3.9.2 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;
        dist = pq.top().first;
        pq.pop();
        if(dist > dis[a]) continue;
        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});
            }
        }
    }
}
```

3.9.3 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 Heavy-Light Decomposition

```
// O(log^2(N)) for one hld_qry

int p[N], dep[N], head[N], heavy[N], seg_arr[N], pos[N], seg
    [4*N];

int dfs(int now, int par){
    int sub = 1;
    p[now] = par;
    dep[now] = dep[par]+1;
    pair<int,int> mx_ch = {0,0};
    for(auto x:adj[now]){
        if(x == par) continue;
        int ch_sub = dfs(x,now);
        mx_ch = max(mx_ch,{ch_sub,x});
        sub += ch_sub;
    }
    heavy[now] = mx_ch.second;
    return sub;
}

int cur_pos = 1;
void hld(int now, int hd){
    seg_arr[cur_pos] = arr[now];
    pos[now] = cur_pos++;
    head[now] = hd;
    for(auto x:adj[now]){
        if(x != p[now]) continue;
        if(x == heavy[now]) hld(x,hd);
        else hld(x,x);
    }
}

int hld_qry(int a, int b){
    int ans = 0;
    while(head[a] != head[b]){
        if(dep[head[a]] > dep[head[b]]) swap(a,b);
        ans = max(ans,qry(pos[head[b]],pos[b],1,n,1));
        b = p[head[b]];
    }
    if(dep[a] > dep[b]) swap(a,b); // a is lca
    ans = max(ans,qry(pos[a],pos[b],1,n,1));
    return ans;
}
```

4.2 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)];
}
```

4.3 Tree Center

```
void init(){
    for(int i = 1; i <= n; i++){
        if(degree[i] == 1|| degree[i] == 0){
            leaf[0].push_back(i);
        }
    }
}

void treeCenters(){
    init();
    int t = 1;
    bool add = 1;
    while(add){
        add = 0;
        int now = t%2, pre = !(t%2);
        leaf[now].clear();
        for(auto z:leaf[pre]){
            for(auto x:adj[z]){
                degree[x]--;
                if(degree[x] == 1){
                    leaf[now].push_back(x);
                    add = 1;
                }
            }
        }
        t++;
    }
}// leaf[t%2] are centers of tree
```

4.4 Tree Centroid

```
// each subtree has at most floor(n/2) nodes

void find_centroid(int now, int p){
    for(auto x:adj[now]){
        if(x == p) continue;
        if(sub[x] > n/2) return find_centroid(x,now);
    }
    return now;
}
```

5 E Range Queries

5.1 BIT

5.1.1 1D-BIT

```
// O(logn) for update
// O(logn) for sum

void update(ll num, int pos){
    int k = pos;
    while(k <= n){
        BIT[k] += num - arr[pos];
        k += k & -k;
    }
    arr[pos] = num;
}

ll sum(int k){ //sum[1,k]
    ll res = 0;
    while(k > 0){
        res += BIT[k];
        k -= k & -k;
    }
    return res;
}
```

5.1.2 2D-BIT

```
void upd(int x, int y, ll dif){
    for(int i = x; i <= n; i+=i&-i){
        for(int j = y; j <= n; j+=j&-j){
            BIT[i][j] += dif;
        }
    }
}

int sum(int x, int y){
    int res = 0;
    for(int i = x; i >= 1; i-=i&-i){
        for(int j = y; j >= 1; j-=j&-j){
            res += BIT[i][j];
        }
    }
    return res;
}
```

5.2 Mo's algorithm

```
//O(qlogq+(q+n)sqrt(n))

struct Query{
    int left, right, idx;
};

Query qry[MX_q];
int block = sqrt(n);
int cnt[MX_N], pos[MX_N], ans[MX_N];
map<int,int> num2idx; // same number to same idx

bool cmp(Query &a, Query &b){
    if(a.left / block != b.left / block){
        return a.left / block < b.left / block;
    }
    return a.right < b.right;
}

void reindex(){
    int ptr = 1;
    for(int i = 1; i <= n; i++){
        if(!num2idx[arr[i]]) num2idx[arr[i]] = ptr++;
        arr[i] = num2idx[arr[i]];
    }
}

void Mo(){
```

```
    sort(qry,qry+q,cmp);
    reindex();
    int tmp_ans = 0, l = 0, r = 0;
    for(int i = 0; i < q; i++){
        Query x = qry[i];
        while(l < x.left){
            if(!--cnt[arr[l++]]) tmp_ans--;
        }
        while(x.left < l){
            if(!cnt[arr[--l]]++) tmp_ans++;
        }
        while(r < x.right){
            if(!cnt[arr[++r]]++) tmp_ans++;
        }
        while(x.right < r){
            if(!--cnt[arr[r--]]) tmp_ans--;
        }
        ans[x.idx] = tmp_ans;
    }
}
```

5.3 Segment tree

5.3.1 Lazy Propagation

```
struct SEG{
    int l, r;
    ll sum, add, setto;
};

int act; // 1(add) 2(set) 3(qry)
SEG seg[900000];

void build(int l, int r, int id){
    seg[id].l = l; seg[id].r = r;
    if(l == r){
        seg[id].sum = arr[l];
        return;
    }
    int m = (l+r)/2;
    build(l,m,id*2);
    build(m+1,r,id*2+1);
    seg[id].sum = seg[id*2].sum + seg[id*2+1].sum;
}

void upd(int id, ll ad_val, ll st_val){
    int len = seg[id].r - seg[id].l + 1;
    if(st_val){
        seg[id].sum = st_val*len;
        seg[id].setto = st_val;
        seg[id].add = 0;
    }
    seg[id].sum += ad_val*len;
    seg[id].add += ad_val;
}

ll Act(int id, ll ad_val, ll st_val){ //(1,0,0)
    int l = seg[id].l, r = seg[id].r;
    if(r < ql || qr < l){
        upd(id,ad_val,st_val);
        return 0;
    }
    if(ql <= l && r <= qr){
        if(act == 1) upd(id,val+ad_val,st_val);
        else if(act == 2) upd(id,0,val);
        else upd(id,ad_val,st_val);
        return seg[id].sum;
    }
    if(!st_val) st_val = seg[id].setto;
    else seg[id].add = 0;
    ll res = Act(id*2,seg[id].add+ad_val,st_val) + Act(id
        *2+1,seg[id].add+ad_val,st_val);
    seg[id].sum = seg[id*2].sum + seg[id*2+1].sum;
    seg[id].add = seg[id].setto = 0;
    return res;
}
```

5.3.2 Persistent segment tree

```
#define L first
#define R second
// k(version) starts at 0

struct SEG{
    ll sum;
    int l_ver, r_ver;
};

pair<int,int> rg[900000];
vector<SEG> seg[900000];
SEG tmp;

void build(int l, int r, int id){
    rg[id].L = l; rg[id].R = r;
    tmp.l_ver = 0; tmp.r_ver = 0;
    seg[id].push_back(tmp);
    if(l == r){
        seg[id][0].sum = arr[l];
        return;
    }
    int m = (l+r)/2;
    build(l,m,id*2);
    build(m+1,r,id*2+1);
    seg[id][0].sum = seg[id*2][0].sum + seg[id*2+1][0].sum;
}

void upd(int id, int k){
    int l = rg[id].L, r = rg[id].R;
    if(l == r){
        tmp.sum = val;
        seg[id].push_back(tmp);
        return;
    }
    int m = (l+r)/2;
    if(pos <= m){
        upd(id*2, seg[id][k].l_ver);
        tmp.l_ver = seg[id*2].size()-1;
        tmp.r_ver = seg[id][k].r_ver;
    }
    else{
        upd(id*2+1, seg[id][k].r_ver);
        tmp.l_ver = seg[id][k].l_ver;
        tmp.r_ver = seg[id*2+1].size()-1;
    }
    tmp.sum = seg[id*2][tmp.l_ver].sum + seg[id*2+1][tmp.
        r_ver].sum;
    if(id == 1) seg[id][k] = tmp;
    else seg[id].push_back(tmp);
}

ll qry(int id, int k){
    int l = rg[id].L, r = rg[id].R;
    if(r < ql || qr < l) return 0;
    if(ql <= l && r <= qr) return seg[id][k].sum;
    return qry(id*2,seg[id][k].l_ver) + qry(id*2+1,seg[id][k
        ].r_ver);
}

// copy k as the latest version
// seg[1].push_back(seg[1][k]);
```

5.3.3 Segment tree

```
// O(n) for build
// O(logn) for update
// O(logn) for query
```

```
void build(int id, int l, int r){
    if(l == r){
        seg[id] = arr[l];
        return;
    }
    int m = (l+r)/2;
    build(id*2,l,m);
    build(id*2+1,m+1,r);
    seg[id] = min(seg[id*2],seg[id*2+1]);
}

void upd(int num, int pos, int id, int l, int r){
    if(l == r){
        seg[id] = num;
        return;
    }
    int m = (l+r)/2;
    if(pos <= m) upd(num,pos,id*2,l,m);
    else upd(num,pos,id*2+1,m+1,r);
    seg[id] = min(seg[id*2],seg[id*2+1]);
}


int query(int ql, int qr, int id, int l, int r){
    int m = (l+r)/2;
    if(r < ql || qr < l) return INF;
    else if(ql <= l && r <= qr) return seg[id];
    else return min(query(ql,qr,id*2,l,m),query(ql,qr,id*2+1,
        m+1,r));
}
```

5.4 Sparse table

```
// O(nlogn) for build
// O(1) for query

void build(){
    for(int i = 1; i <= n; i++){
        cin >> sp[0][i];
    }
    for(int i = 1; (1<<i) <= n; i++){
        for(int j = 1; j+(1<<i)-1 <= n; j++){
            sp[i][j] = min(sp[i-1][j],sp[i-1][j+(1<<(i-1))]);
        }
    }
}

ll query(int l, int r){
    int k = (int)log2(r-l+1);
    return min(sp[k][l],sp[k][r-(1<<k)+1]);
}
```

6 F Math

6.1 Miller Rabin

```
// srand( time(NULL) );

// for big integer multiplication
ll mult(ll x, ll y){
    ll ans = 0;
    while(y){
        if(y&1) ans += x;
        ans %= n;
        x += x;
        x %= n;
        y >>= 1;
    }
    return ans;
}

bool is_prime(ll x){
    if(x == 2) return 1;
    if(x == 1 || !(x & 1)) return 0;
    else{
        s = 0;
        d = x-1;
        while(!(d & 1)){
            s++;
            d >>= 1;
        }
        for(int t = 0; t < 10; t++){
            a = rand() % (x-1) + 1;
            tmp = pw(a,d);
            bool fg = 0;
            for(int i = 0; i < s; i++){
                if(tmp % x == x-1) fg = 1;
                if(i == 0 && tmp % x == 1) fg = 1;
                tmp = mult(tmp,tmp);
                tmp %= x;
            }
            if(!fg) return 0;
        }
        return 1;
    }
}
```

7 Z Others

7.1 Bipartite$_{M}atching_{A}nd_{M}aximum_{F}low$

7.1.1 Bipartite

```
#include<bits/stdc++.h>
using namespace std;
# define N 5000 // N has to be larger than n+m
int n, m, k; // n = left set size, m = right set ,size, k =
    number of edge
int match[N]; bool used[N]; vector<int> adj[N];
bool DFS(int x) {
    for(auto u : adj[x]) {
        if(used[u]) continue;
        used[u] = 1;
        int next = match[u];
        if(next == -1 || DFS(next)) {
            match[u] = x;
            return 1;
        }
    }
    return 0;
}

int Bipartite_match() {
    memset(match, -1, sizeof(match));
    int match_number = 0;
    for(int i = 1;i <= n;i++) {
        memset(used, 0, sizeof(used));

        match_number += DFS(i);
        //cout<<match_number<<endl;
    }
// and if(match[i] != -1) {i(right set) and ,match[i](left
    set) match}
    return match_number;
}
void init() {
    for(int i = 0;i < k;i++) {
        int a, b; scanf("%d%d", &a, &b);
        b = b + n; // this is an important part about
            debugging
        adj[a].push_back(b); // a in left set, b ,in right
            set
    }
}
```

7.1.2 Bipartite$_h opcroft_k arp$

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 100001
#define NIL 0
#define INF (1<<28)

vector< int > G[MAX];
int n, m, match[MAX], dist[MAX];
// n: number of nodes on left side, nodes are numbered 1 to
    n
// m: number of nodes on right side, nodes are numbered n+1
    to n+m
// G = NIL[0]    G1[G[1---n]]    G2[G[n+1---n+m]]

bool bfs() {
    int i, u, v, len;
    queue< int > Q;
    for(i=1; i<=n; i++) {
        if(match[i]==NIL) {
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = INF;
    }
    dist[NIL] = INF;
    while(!Q.empty()) {
        u = Q.front(); Q.pop();
        if(u!=NIL) {
            len = G[u].size();
            for(i=0; i<len; i++) {
                v = G[u][i];
                if(dist[match[v]]==INF) {
                    dist[match[v]] = dist[u] + 1;
                    Q.push(match[v]);
                }
            }
        }
    }
    return (dist[NIL]!=INF);
}

bool dfs(int u) {
    int i, v, len;
    if(u!=NIL) {
        len = G[u].size();
        for(i=0; i<len; i++) {
            v = G[u][i];
            if(dist[match[v]]==dist[u]+1) {
                if(dfs(match[v])) {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
            }
        }
        dist[u] = INF;
        return false;
    }
    return true;
}

int hopcroft_karp() {
    int matching = 0, i;
    // match[] is assumed NIL for all vertex in G
    while(bfs())
        for(i=1; i<=n; i++)
            if(match[i]==NIL && dfs(i))
                matching++;
    return matching;
}
int main ()
{
    // cause
    // n: number of nodes on left side, nodes are numbered 1
        to n
    // m: number of nodes on right side, nodes are numbered n
        +1 to n+m
    // when input m dots you need to assign it as (+n +b)
        nodes.
    int k,a,b;
    cin>>n>>m>>k;
    for(int i = 0; i < k; i++){
        cin>>a>>b;
        b = b + n; // girls
        G[a].push_back(b);
        G[b].push_back(a);
    }
    cout<<hopcroft_karp()<<endl;
}
```

7.1.3 Maxflow

```
#include<bits/stdc++.h>
using namespace std;
#define N 5050
#define INF 1e18
typedef long long ll;
int n,m,a,b,c;
int vis[5050]={0};
struct Edge {
    int to;
    ll cap;
    int rev; /*                    intindex
        */
    //Edge(){} /*             constructor */
    Edge(int _to, ll _cap, int _rev): to(_to), cap(_cap), rev
        (_rev) {}
};

vector<Edge> adj[N];

void add_edge(int from, int to, ll cap) {
    adj[from].push_back(Edge(to, cap, (int)adj[to].size()));
    /* from -> to, e.cap = cap */

    adj[to].push_back(Edge(from, 0, (int)adj[from].size() -
        1));
    /* to -> from, e.cap = 0 */
}


int s, t;
ll DFS(int now, ll flow) {
    if(now == t) return flow;
    vis[now] = 1;
    for(int i = 0;i < (int)adj[now].size();i++) {
        Edge &e = adj[now][i];
        if(e.cap > 0 && !vis[e.to]) {
            ll ret = DFS(e.to, min(flow, e.cap));
            if(ret > 0) {
                e.cap -= ret;
                adj[e.to][e.rev].cap += ret;
                return ret;
            }
        }
    }
    return 0;
}
ll max_flow() {
    ll ret = 0;
    ll tmp = 0;
    while((tmp = DFS(s, INF)) > 0) {
        ret += tmp;
        memset(vis, 0, sizeof(vis)); /*       */
    }
    return ret;
}
```

```
int main ()
{
    cin>>n>>m;
    for(int i = 0; i < m; i++){
        cin>>a>>b>>c;

        add_edge(a,b,c);
        add_edge(b,a,-c);
    }
    s=1, t=n;
    cout<<max_flow()<<endl;
}
```

7.2 DP

7.2.1 dp on tree

7.2.2 distance tree

```
// find the sum of all path cost
// O(N)
void init(){
    v.clear();
    ans = 0;
    for(int i=0;i<=n;i++){
        si[i] = 0;
    }
}

void dfs(int node){
    if(si[node] !=0){return;}
    si[node] += 1;
    for(const int& e:v[node]){
        dfs(e);
        ans += weight[e] * (n-si[e]) * (si[e]) * 2;
        si[node] += si[e];
    }
}
```

7.2.3 knapsack$_{p}ath$

```
typedef long long int lld;
struct bag
{
    int w, v;// weight and value.
}o[1005]; // object
lld dp[1005][100005] = {0};
set<int> use;
set<int> :: iterator it1, it2;
cin>>n>>m; // n object and m weigt
for(int i = 0; i < n; i++)
    cin>>o[i].v>>o[i].w;
// initialize
for(int i = 1; i <= n; i++)
    for(int j = 0; j <= m; j++)
        if(o[i-1].w <= j) dp[i][j] = max(dp[i-1][j], dp[i-1][
            j-o[i-1].w] + o[i-1].v);
        else
            dp[i][j] = dp[i-1][j];

int rec_pos = m;
lld tmp = dp[n][rec_pos];
// the i means the line from down to top and means the
    object which is used if not continue.
// the below we can draw a DAG to debug
for(int i = n; i >= 0; i--){
    if(tmp == dp[i][rec_pos]) continue;
    rec_pos -= o[i].w;
    tmp = dp[i][rec_pos];
    use.insert(i);
}
it1 = use.begin();
it2 = use.end();
cout<<use.size()<<endl;
while(it1!=it2)
{
    cout<<*it1<<" ";
    it1++;
}
```

7.2.4 LCS

```
int l1 = s1.size(), l2 = s2.size();
for(int i = 1; i <= l1; i++){
    for(int j = 1; j <= l2; j++){
        if(s1[i-1] == s2[j-1]){
            dp[i][j] = dp[i-1][j-1] + 1;
        }
        else
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
}
```

7.2.5 LIS$_{l}ower_{b}ound$

```
fill(dp, dp+n+1, 1e15);
dp[0] = 0;
maxnum = 0;
for(int i = 0; i < n; i++){
    int index = lower_bound(dp, dp + maxnum+1, arr[i]) - dp;
    dp[index] = min(dp[index], arr[i]);
    if(index > maxnum) maxnum = index;
}
cout<<maxnum<<endl;
```

7.2.6 Throwing a Party

```
//giving a tree a choose the most node value sum that all
    nodes don't connect

// using DFS: ans = max(dp[0][1], dp[1][1]);
int r[1003];// node value
vector<vector<int>> v;
int dp[2][1003];

void init(int n){
    v.clear();
    for(int i=0;i<=n;i++){
        dp[0][i] = 0;// choose
        dp[1][i] = 0;
    }
}

void dfs(int node){
    dp[0][node] += r[node];// choose node
    for(const int& e:v[node]){
        dfs(e);
        dp[0][node] += dp[1][e];
        dp[1][node] += max(dp[1][e], dp[0][e]);
    }
}

// using BFS: ans = max(dp[0][1], dp[1][1]);
int in[1003], r[1003];
vector<vector<int>> v;
int dp[2][1003];
bool vis[1003];

void init(int n){
    v.clear();
    for(int i=0;i<=n;i++){
        in[i] = 0;
        dp[0][i] = 0;
```

```
        dp[1][i] = 0;
        vis[i] = 0;
    }
}

void BFS(int n){
    queue<int> q;
    for(int i=2;i<=n;i++){
        if(in[i] == 0){
            q.push(i);
        }
    }
    int node;
    dp[0][1] = r[1];
    while(!q.empty()){
        node = q.front();
        q.pop();
        vis[node] = true;
        dp[0][node] += r[node];
        for(const int& e:v[node]){
            if(vis[e]){continue;}
            --in[e];
            dp[0][e] += dp[1][node];//choose e
            dp[1][e] += max(dp[1][node], dp[0][node]);
            if(e != 1 and in[e] == 0){
                q.push(e);
            }
        }
    }
}
```

7.2.7 Recursion$_{T}ips$

7.3 Interval$_{A}citivity_{P}rob$

7.3.1 weight$_{a}ctivity$

```
typedef long long int lld;
lld n;
struct acti{
    lld s,t,w; // start, terminal, weight
} act[200005];
bool cmp(acti a, acti b){
    if(a.t == b.t)
        return a.s < b.s;
    return a.t < b.t;
}
lld dp[200005];
lld rec[200005];
lld bs(lld l, lld r, lld select_start);
int main()
{
    cin>>n;
    for(int i = 1; i <= n; i++)
        cin>>act[i].s>>act[i].t>>act[i].w;
    sort(act+1, act+n+1, cmp);
    act[0].s = act[0].t = act[0].w = 0;
    rec[0]=dp[0]=0;
    for(int i = 1; i <= n; i++){
        lld index=bs(-1, i-1, act[i].s);
        while(index > 0 && rec[index] >= act[i].s)
            index--;
        if(dp[i-1] > dp[index] + act[i].w){
            dp[i] = dp[i-1];
            rec[i] = rec[i-1];
        }else{
            dp[i] = dp[index] + act[i].w;
            rec[i] = act[i].t;
        }
    }
    cout<<dp[n]<<endl;
}

lld bs(lld l, lld r, lld select_start){
    lld mid;
    while(l < r-1)
    {
        mid = (l+r)/2;
        if(rec[mid] >= select_start)
            r = mid;
        else
            l = mid;
    }
    return r;
}
```

7.4 math

7.4.1 Binomial Coefficients

```
// build O(N), inverse O(log(MOD))

long long factorial[1000006];
const long long MOD = (1e9)+7;// should be a prime

void build(){
    factorial[0] = 1;
    for(int i=1;i<=1000000;i++){
        factorial[i] = factorial[i-1]*i % MOD;
    }
}

long long inverse(long long x){
    return exp(x, MOD-2);
}

long long binomial_coefficients(int n, int m){
    return factorial[n] * inverse(factorial[m])%MOD * inverse
        (factorial[n-m])%MOD;
}
```

7.4.2 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;
}
```

7.4.3 CRT

```
typedef __int128 lld;
lld inv(lld a, lld m); // a mod m
lld n;
lld T[5], d[5];
lld crt();
int main()
{
    n = read();
    for(int i = 0; i < n; i++){
        T[i]=read(), d[i]=read();

    }
    print(crt());
    cout<<'\n';

}
lld inv(lld a, lld m) // a mod m
{
    lld m0 = m, t, q;
    lld x0 = 0, x1 = 1;
    if (m == 1)
        return 0;
    // Apply extended Euclid Algorithm
    while (a > 1) {
        // q is quotient
        q = a / m;
        t = m;
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if (x1 < 0)
        x1 += m0;
    return x1;
}
lld crt()
{
    lld sum=1;
    for(int i = 0; i < n; i++){
        sum *= T[i];
    }
    lld x=0;
    for(int i = 0; i < n; i++){
        x += (d[i]%sum) * inv(sum/T[i], T[i])%sum * (sum/T[i
            ])%sum;
        x %= sum;
    }
    x = x % sum;
    return x ;
}
```

7.4.4 decomposition

```
// Pollard-Rho algorithm
// O(N^(1/4))
// https://iter01.com/550538.html
// do factorize decomposition

vector<long long> ans;// mutiply all number in ans is n and
     all number in ans is prime
// remeber to initialize ans

long long mul(long long a, long long b, long long mod){//
     fast mutiply
    // try not to overflow when doing: long long * long long
         % long long
 long long sum = 0;
 a %= mod, b %= mod;
 while(b > 0){
  if(b & 1){
   sum = (sum + a) % mod;
  }
  a = (a + a) % mod;
  b /= 2;
 }
 return sum;
}

long long f(long long x, long long c, long long mod){
 return (mul(x, x, mod) + c) % mod;
}

void factor(long long n){
 if(is_prime(n)){
  ans.push_back(n);
  return;
 }
 long long c, x, y, d = 1;
 y = x = rand() % n;
 while(d == 1 or d == n){
  c = rand() % n;
  d = __gcd(c, n);
  if(d != 1 and d != n){break;}
  bool first = true;
  while(x != y or first){
   x = f(x, c, n);
```

```
   y = f(f(y, c, n), c, n);
   d = __gcd(abs(x - y), n);
   if(d != 1 and d != n){
    break;
   }
   if(x == y){d = __gcd(x, n);}
   first = false;
  }
 }
 factor(d), factor(n/d);
}
```

7.4.5 diceTurning

```
// how to represent the turning dice?
// O(1)

struct DICE{
 int t, d, f, r, b, l;// someone use bad naming XD
 bool operator == (const DICE& right)const{// maybe
      different
  if(t != right.t) return false;
  if(d != right.d) return false;
  if(f != right.f) return false;
  if(r != right.r) return false;
  if(b != right.b) return false;
  if(l != right.l) return false;
  return true;
 }
 bool operator < (const DICE& right)const{// check it
      carefully
  if(t != right.t) return t < right.t;
  if(f != right.f) return f < right.f;
  if(r != right.r) return r < right.t;
  if(b != right.b) return b < right.b;
  if(l != right.l) return l < right.l;
  return d < right.d;
 }

};

DICE turn(DICE dice, string way){
 if(way == "north"){
  swap(dice.d, dice.f);
  swap(dice.f, dice.t);
  swap(dice.t, dice.b);
 }
 else if(way == "east"){
  swap(dice.d, dice.r);
  swap(dice.t, dice.r);
  swap(dice.t, dice.l);
 }else if(way == "south"){
  swap(dice.t, dice.b);
  swap(dice.t, dice.f);
  swap(dice.f, dice.d);
 }else if(way == "west"){
  swap(dice.t, dice.l);
  swap(dice.t, dice.r);
  swap(dice.r, dice.d);
 }
 return dice;
}
```

7.4.6 doInsertect

```
// consider two segment is collision or not
// O(1)
// use the concept of vector in math
// there are also some special case too

struct Point{
 int x, y;
};

// checks if point q lies on line segment p-r
bool onSegment(Point p, Point q, Point r)
{
 if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
   q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
  return true;

 return false;
}

// To find orientation of ordered triplet (p, q, r).
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
 // reference from http://www.dcs.gla.ac.uk/~pat/52233/
      slides/Geometry1x1.pdf
 int val = (q.y - p.y) * (r.x - q.x)-(q.x - p.x) * (r.y - q.
      y);

 if (val == 0) return 0; // colinear

 return (val > 0)? 1: 2; // clock or counterclock wise
}

// line segment p1-q1 and line segent p2-q2 have
     intersection point or not.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
 //Find four orientations needed for general and special
      case
 int o1 = orientation(p1, q1, p2);
 int o2 = orientation(p1, q1, q2);
 int o3 = orientation(p2, q2, p1);
 int o4 = orientation(p2, q2, q1);

 // General case- line segment A crosses line segment B,
      looks like shape X.
 if (o1 != o2 && o3 != o4)
  return true;

 // Special Cases-one end point of line segment A(p1-q1 or
      p2-q2) lie s on line segment B(p2-q2 or p1-q1), looks
      like shape T.
 if (o1 == 0 && onSegment(p1, p2, q1)) return true;

 if (o2 == 0 && onSegment(p1, q2, q1)) return true;

 if (o3 == 0 && onSegment(p2, p1, q2)) return true;

 if (o4 == 0 && onSegment(p2, q1, q2)) return true;

 return false;
}
```

7.4.7 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;
}
```

7.4.8 Quadratic$_{C}$ongruence$_{E}$quation

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

long long exp(long long base, long long deg, long long mod){
    base = base % mod;
    long long sum = 1;
    while(deg > 0){
        if(deg & 1){
            sum = (sum * base) % mod;
        }
        base = (base * base) % mod;
        deg >>= 1;
    }
    return sum;
}

struct Complix{
    long long r, v, w, p;
    Complix(long long _r, long long _v, long long _w, long
         long _p):r(_r), v(_v), w(_w), p(_p){};
    Complix operator*(const Complix& right){
        long long _r = r * right.r % p + v * right.v % p * w
             % p;
        long long _v = v * right.r % p + r * right.v % p;
        _r %= p, _v %= p;
        return Complix(_r, _v, w, p);
    }
};

long long inverse(long long x, long long mod){
    return exp(x, mod-2, mod);
}

long long exp(Complix base, long long deg){
    Complix sum = Complix(1, 0, base.w, base.p);
    while(deg > 0){
        if(deg & 1){
            sum = (sum * base);
        }
        base = (base * base);
        deg >>= 1;
    }
    return sum.r;
}

int main(){
    srand(time(NULL));
    int T;
    cin>>T;
    long long a, b, d, p;
    while(T--){
        cin>>a>>b>>d>>p;
        if(d == 0){
            cout<<(p-b)%p<<'\n';
            continue;
        }
        if(p == 2){
            //a == 1
            cout<< ((b+d) & 1)<<'\n';
            continue;
        }
        d = d * inverse(a, p) % p;
        if(exp(d, (p-1)/2, p) == p-1 || exp(d, (p-1)/2, p) ==
             -1){
            cout<<-1<<'\n';
            continue;
        }
        long long _a, _w;
        bool flag = false;
        while(!flag){
            _a = (rand() % p + p) % p;
            _w = (_a*_a - d) % p;
            if( or exp(_w, p/2, p) == -1){
                flag = true;
            }
        }
        Complix c = Complix(_a, 1, _w, p);
        long long solution = exp(c, (p+1)/2);
        long long ans1 = ((solution - b)%p + p) % p;
        long long ans2 = ((-solution - b)%p + p) % p;
        if(ans1 == ans2){cout<<ans1<<endl;continue;}
        cout<<min(ans1, ans2)<<" "<<max(ans1, ans2)<<endl;
    }
    return 0;
}
```

7.4.9 random$_{p}rime$

```
// Miller-Rabin
// https://www.cnblogs.com/RioTian/p/13927952.html
// O(K*logN) k is modifiable just consider the correctness
// use random algorithm to check a number is prime or not

long long exp(__int128 base, long long deg, long long mod){
    __int128 sum = 1;// use __int128 or fast_multipy to avoid
         overflow
    while(deg > 0){
        if(deg & 1){
            sum = sum * base % mod;
        }
        base = base * base % mod;
        deg /= 2;
    }
    return sum;
}

bool is_prime(long long n){
    if(n == 2){return true;}
    if(n == 1 or n %2 == 0){return false;}
    long long s=0, d=n-1;
    while((d & 1) == 0){
        s++, d/=2;
    }
    int k = 100;
    long long a;
    while(k--){
        a = rand() % n;
        if(a == 0){continue;}
        bool flag = false;
        if(exp(a, d, n) == 1){flag = true;}
        for(int i=0;i<s;i++){
            if(exp(a, (1LL<<i)*d, n) == n-1){
                flag = true;
            }
        }
        if(!flag){return false;}
```

```
    }
    return true;
}
```

7.5 STL

7.5.1 mapAndSet

```
#include<map> // header file of map
#include<set> // header file of set

//
map<int, int> mp;
set<int> s;
//          :   keyoperator   <

// usage

// add thing in it
    mp[x] = y;
    s.insert(x);
// access thing
    y = mp[x];
// check x exist or not
    mp.find(x) != mp.end();
    s.find(x) != s.end();
// check empty
    mp.empty();
    s.empty();
// check size
    mp.size();
    s.size();
// binary search
    map<int, int>::iterator it = mp.lower_bound(x);
    set<int>::iterator it2 = s.lower_bound(x);
    if(it == mp.end()){}//not found
    *it// the first element y >= x
    // upper_bound: first element y > x
```

7.5.2 others

```
#include<algorithm>// fill, sort, __gcd
#include <cstring> // memset

//fill
vector<int> v(5);
fill(v.begin(), v.end(), -1);
fill(a, a+n, 0);

//memset
void* memset( void* dest, int ch, std::size_t count );
int a[20];
std::memset(a, 0, sizeof a);

// sort

bool cmp(int a, int b){
    // should not put '=' in return, '=' will causing RE
    if(a %2 != b % 2){
        return a % 2 > b % 2;
    }
    return a > b;
}
sort(a, a+n, cmp);
lower_bound(a, a+n, x);// first element y >= x
upper_bound(a, a+n, x);// first element y > x
```

7.5.3 queueAndStack

```
#include<queue> // header file of queue and priority_queue
#include<stack> // herder file of stack


//
queue<int> q;

priority_queue<int> pq;// max heap
priority_queue<int, vector<int>, greater<int>> pq;// min
     heap
//priority_queue                    operator < and don't
     forget to add const
struct Point{
    int a, b;
    bool operator < (const Point& right)const{
        if(a != right.a)  return a < right.a;
        return b < b.right.b;
    }
}
priority_queue<Point> pq;

stack<int> st;

// usage

// add thing in it
    q.push(x);
    pq.push(x);
    st.push(x);
// access thing
    x = q.front();
    x = pq.top();
    x = st.top();
// take the thing out of it
    q.pop();
    pq.pop();
    st.pop();
// check empty
    q.empty();
// check size
    q.size();
```

7.5.4 stringstream

```
#include<sstream>// header file of stringstream

stringstream ss;

void init(){// initialize stringstream
    ss.str("");
    ss.clear();
}

void usage(){
    // just use it like cin, cout
    ss<<t;// put t into ss
    ss>>t;// put a token of value to t
}
```

7.6 String$_{manipulation}$

7.6.1 KMP$_{Tutorial}$

```
int nexts[50005]={0};
vector<int>ans;
void getNext(string s) {
    int i,j;
    j=0;
    nexts[0]=0;
    for(i=1;i<s.size();i++){
        while(j>0 && s[i] != s[j])
            j = nexts[j-1];
        if(s[i]==s[j])
            j++;
```

```
        nexts[i]=j;
    }
    for(int i = 0; i < s.size(); i++)
        cout<<i<<" ";
    cout<<endl;
    for(int i = 0; i < s.size(); i++)
        cout<<nexts[i]<<" ";
    cout<<endl;
}
// h means the text n means the pattern
int get_pos(string h, string n){
    ans.clear();
    int j=0;
    for(int i = 0; i < h.size(); i++){
        if(h[i]==n[j]){
            j++;

            if(j == n.size()){
               //return i-n.size()+1;
               ans.push_back(i-n.size()+1);
               j=j-1;
               while(j > 0 && h[i] != n[j])
                 j = nexts[j-1];
            }

        }else{
            while(j > 0 && h[i] != n[j])
                j = nexts[j-1];
            if(h[i]==n[j])
                j++;
        }

    }

    return -1;
}

int main ()
{
    string h="aaaaaaaaaaa"; // THE TEXT
    string n="aa"; // THE PATTERN
    getNext(n);
    get_pos(h,n);
    for(int i = 0; i < ans.size(); i++)
        cout<<ans[i]<<" ";
    cout<<endl;
}
```

7.6.2 Stringstream

```
stringstream ss;
ss.str("");
ss.clear();
getline(cin, s);
ss<<s;
ss>>(string); // stoi can only use for 214748367 as max
     value.
ss>>(long long int ); // if use ss>>(long long int), it will
      be really strong.
```

7.6.3 String$_{Function}$

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    string s = "abcd";
    // to upper - 32 ASCII to lower + 32 ASCII
    s.substr(0, 0) // ""
    s.substr(0, 1) // a
    s.substr(0, 2) // ab
    s.erase(2,2) // start position (including), from the
         start position count number.
    s.pop_back(); // delete the last element
    s.size();
    s.empty(); // return 1 if empty
    reverse(s.begin() + 1, s.begin()+2); // s.begin() + i, s.
         begin()+j;
    // int convert to string
    string num = to_string(5);
    // string convert to int
    int number = stoi("123");

}
```

7.7 tree

7.7.1 articulation point and bridge

```
// O(N)

vector<vector<int>> v;
int D[100005], L[100005];
int time_stamp = 0;
vector<int> ans_AP;
vector<pair<int, int>>ans_bridge;

void DFS(int node, int par){
    D[node] = L[node] = ++time_stamp;
    int child_cnt = 0;
    bool is_AP = false;
    for(const int& e:v[node]){
        if(e == par){continue;}
        if(D[e] == 0){
            child_cnt++;
            DFS(e, node);
            if(D[node] <= L[e]){is_AP = true;}// point
            if(D[node] < L[e]){ans_bridge.push_back({e, node
                 });}// bridge
            L[node] = min(L[node], L[e]);
        }else{
            // back edge
            L[node] = min(D[e], L[node]);
        }
    }
    if(par == 0){is_AP = child_cnt >=2;}// root is special
         case
    if(is_AP){
        ans_AP.push_back(node);
    }
}
```

7.7.2 little$_{span}$tree$_{d}$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;
}
```

7.7.3 $\text{little}_{s}pan_{t}ree_{p}rim$

```
#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;
}
```

7.7.4 LowCommonAncestorSuc

```
// use successor graph's data structure to find LCA
// O(NlogN)

vector<vector<int>> v;
int suc[32][200005];
int deep[200005];

void dfs(int node, int par){
    suc[0][node] = node;
    suc[1][node] = par;
    deep[node] = deep[par]+1;
    for(int u:v[node]){
        if(u != par){
            dfs(u, node);
        }
    }
}

void build(int n){
    for(int i = 2; i<32;i++){
        for(int j = 1;j<=n;j++){
            suc[i][j] = suc[i-1][suc[i-1][j]];
        }
    }
}

int jump(int node, int k){
    int i = 1;
    while(k > 0){
        if(k & 1){
            node = suc[i][node];
        }
        k = k>>1, i++;
    }
    return node;
}

int find_LCA(int node1, int node2){
    if(deep[node1] < deep[node2]){
        swap(node1, node2);
    }
    node1 = jump(node1, deep[node1] - deep[node2]);
    for(int i = 31;i>0;i--){
        if(suc[i][node1] != suc[i][node2]){
            node1 = suc[i][node1];
            node2 = suc[i][node2];
        }
    }
    if(node1 != node2){
            node1 = suc[1][node1];
    }
    return node1;
}

int find_distance(int node1, int node2){
    int lca = find_LCA(node1, node2);
    return deep[node1] + deep[node2] - 2*deep[lca];
}
```

7.7.5 LowCommonAnesctorRMQ

```
#include<iostream>
#include<vector>
using namespace std;
#define INF 100000008
#define maxSize 200005
#define root 1

//can be improve with sparse table

int deep[maxSize];
int visidx[maxSize];
vector<int> way;
vector< vector<int> > adj;
struct Node{
    int L, R;
    int val;
    int id;
}tree[maxSize*8];

void build(int idx, int L, int R){
    tree[idx].L = L;
    tree[idx].R = R;
    if(L == R){
        tree[idx].val = deep[way[L]];
        tree[idx].id = L;
        return;
    }
    int mid = (L + R)/2;
    build(idx*2, L, mid);
    build(idx*2+1, mid+1, R);
    if(tree[idx*2].val > tree[idx*2+1].val){
        tree[idx].val = tree[idx*2+1].val;
        tree[idx].id = tree[idx*2+1].id;
    }else{
        tree[idx].val = tree[idx*2].val;
        tree[idx].id = tree[idx*2].id;
    }
}

int query(int idx, int qL, int qR){
    //cout<<idx<<" "<<tree[idx].L<<" "<<tree[idx].R<<" "<<
        tree[idx].id<<endl;
    if(qL > tree[idx].R || qR < tree[idx].L){return -1;}
    else if(tree[idx].L >= qL && tree[idx].R <= qR){
        return tree[idx].id;
    }
    int id1 = query(idx*2, qL, qR);
    int id2 = query(idx*2+1, qL, qR);
    if(id1 == -1){
        return id2;
    }else if(id2 == -1){
        return id1;
    }else{
        if(deep[way[id1]] < deep[way[id2]]){
            return id1;
        }else{
            return id2;
        }
    }
}

void DFS(int node){
    visidx[node] = way.size();
    way.push_back(node);
    for(int u:adj[node]){
        deep[u] = deep[node]+1;
        DFS(u);
        way.push_back(node);
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, q;
    cin>>n>>q;
    adj.resize(n+5);
    int x;
    for(int i=2;i<=n;i++){
        cin>>x;
        adj[x].push_back(i);
    }
    deep[root] = 1;
    DFS(root);
    build(1, 0, way.size()-1);
    int a, b, sum;
    for(int i=0;i<q;i++){
        cin>>a>>b;
        if(visidx[a] > visidx[b]) swap(a, b);
        int ans = query(1, visidx[a], visidx[b]);
        //" "<<deep[a] + deep[b] - 2*deep[way[ans]]
        cout<<way[ans]<<'\n';
    }
    return 0;
}
```

7.7.6 LowCommonAnsectorDjset

```
// use the concept of union and find to find LCA
// outline algorithm
// O(NlogN)

#define INF 100000008
#define maxSize 200005
#define root 1

vector< vector<int> > adj;
vector<vector<pair<int, int> > > query;
int vis[200005];// structure like disjoinset
int ans[200005];// the ans of the Nth query

void init(int n){
    adj.clear();
    query.clear();
    for(int i=1;i<=n;i++){
        vis[i] = i;
        ans[i] = 0;
    }
    adj.resize(n+5);
    query.resize(n+5);
}

int find_r(int x){
    if(x == vis[x]){return x;}
    return vis[x] = find_r(vis[x]);
}

void DFS(int node, int par){
    for(int u:adj[node]){
        DFS(u, node);
    }
    for(auto e:query[node]){
        if(ans[e.second] == 0 and vis[e.first] != e.first){
            ans[e.second] = find_r(e.first);
        }else if(e.first == node){
            ans[e.second] = node;
        }
    }
    vis[node] = par;
}

void query_input(int q){
    for(int i=0, a, b;i<q;i++){
        cin>>a>>b;
        query[a].push_back({b, i});
        query[b].push_back({a, i});
```

```
    }
}

// start with DFS(1, 0);
```

7.7.7 subtreeQueries

```
/*
You are given a rooted tree consisting of n nodes.
The nodes are numbered 1,2,,n, and node 1 is the root.
Each node has a value.

Your task is to process following types of queries:
    1. change the value of node s to x
    2. calculate the sum of values in the subtree of node s
*/

vector<vector<int>> v;
int val[200005];
int node_id[200005];
int sub_size[200005];
int id_value[200005];
int id = 0;

int dfs(int node, int par){
    node_id[node] = id;
    id_value[id] = val[node];
    id++;
    int ts = 1;
    for(int u:v[node]){
        if(u != par){
            ts += dfs(u, node);
        }
    }
    sub_size[node] = ts;
    return ts;
}

struct Node{
    int L, R;
    long long sum;
}tree[4*200005];

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

long long query(int idx, int qL, int qR){
    if(tree[idx].L > qR || tree[idx].R < qL){
        return 0;
    }
    if(tree[idx].L >= qL && tree[idx].R <= qR){
        return tree[idx].sum;
    }
    return query(idx*2, qL, qR) + query(idx*2+1, qL, qR);
}

void update(int idx, int pos, int mod){
    if(tree[idx].L > pos || tree[idx].R < pos){return;}
    if(tree[idx].L == pos && tree[idx].R == pos){
        id_value[pos] = mod;
        tree[idx].sum = mod;
        return;
    }
    update(idx*2, pos, mod);
    update(idx*2+1, pos, mod);
    tree[idx].sum = tree[idx*2].sum + tree[idx*2+1].sum;
}

int main(){
    int n, q;
    cin>>n>>q;
    v.resize(n+5);
    for(int i=1;i<=n;i++){
        cin>>val[i];
    }
    int x, y;
    for(int i=1;i<n;i++){
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    dfs(1, 0);
    build(1, 0, id);
    while(q--){
        cin>>x>>y;
        if(x == 1){
            cin>>x;
            update(1, node_id[y], x);
        }else{
            cout<<query(1, node_id[y], node_id[y] + sub_size[
                y]-1)<<endl;
        }
    }
    return 0;
}
```

7.7.8 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;
}
```

7.7.9 tree$_c$enter$_b$uttom$_u$p

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

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

int tree_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;
}
```

7.7.10 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;
}
```

7.8 useful

7.8.1 build

```
#!/bin/bash

g++ -Wall -O2 -std=c++14 -static -pipe $1 -Wextra && ./a.out
```

7.8.2 intro

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

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    return 0;
}

// useful function swap
// all STL type could swap in O(N)
swap(x, y)
```

7.8.3 sizeint128

```
typedef __int128 lld;
__int128 read();
void print(__int128 x);
int main()
{
    lld n;
    n = read();
    lld x=read();
    print(x); // you have to cout '\n' yourself
    cout<<'\n';
    print(x);
}
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

7.8.4 vimrc

```
set nu
set relativenumber
set autoindent
set smartindent
set cindent
set backspace=2
set confirm
set mouse=a
set tabstop=4
set softtabstop=4
set smarttab
set shiftwidth=4
set hlsearch
set cursorline
set cursorcolumn
set showmatch
set ruler
syntax on
set expandtab " for python code "
inoremap ( ()<Esc>i
inoremap () ()<Esc>i
inoremap [ []<Esc>i
inoremap {<CR> {<CR>}<Esc>O
inoremap " ""<Esc>i
colorscheme ron

" if accidentlt Ctrl + s, then use Ctrl + q to solve
```