# This is not a codebook

June 2, 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

void dfs(int now){
    vis[now] = true;
    for(auto u:adj[now]){
        if(!vis[u]) dfs(u);
    }
}
```

## 3.2 Disjoint Set

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

int boss[MX_N], sz[MX_N]

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

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

void combine(int a, int b){
    a = findBoss(a);
    b = findBoss(b);
    if(sz[a] < sz[b]) swap(a,b);
    boss[b] = a;
    sz[a] += sz[b];
}

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

## 3.3 Shortest Path

### 3.3.1 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;
}
```

### 3.3.2 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});
            }
        }
    }
}
```

### 3.3.3 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 Z Others

## 4.1 graph

### 4.1.1 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
}
```

### 4.1.2 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;
}
```

### 4.1.3 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
}
```

### 4.1.4 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);
    }
}
```

### 4.1.5 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);
    }
}
```

### 4.1.6 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(b, dis[node] + w);
            }
        }
    }
}
```

### 4.1.7 dijkstra's$_{a}lgorith_{i}s_{i}t_{c}orrect$

```
//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);
            }
        }
    }
}
```

### 4.1.8 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);
}
```

### 4.1.9 Floyd$_{W}arshall$

```
//O(N*N*N)
// find the shortest path for each pair

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

## 4.2 math

### 4.2.1 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
//     2.if gcd(s, m) == 0 -->Bezouts Theorem
//            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;
}
```

## 4.3 tree

### 4.3.1 little$_{s}pan_{t}ree$

```
//two way to find little span tree
//     1.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

int way1(){
    sort(edge.begin(), edge.end());
    int sum = 0;
    for(auto e:edge){
        if(!same(e.start, e.terminal)){
            combine(e.start, e.terminal);
            sum += e.weight;
        }
    }
    return sum;
}

int Prim(){
    priority_queue<plli, vector<plli>, greater<plli> > pq;
    pq.push({0, start});
    while(!pq.empty()){
        node = pq.second;
        if(vis[node]){continue;}
        sum += pq.first;
        for(auto u:adj[node]){
            if(!vis[u]){
                pq.push({});
            }
        }
    }
}
```

### 4.3.2 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);
}
```

### 4.3.3 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?
*/
```

### 4.3.4 tree$_{b}fs$

```
#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;
}
```

### 4.3.5 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 : 0-0)
    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;
}
```

### 4.3.6 tree$_diameter_buttom_up$

```
#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 : 0-0)
    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;
}
```