Dividimos y No Conquistamos (D&!C)

Contents

1 Template

1.1 C++ Template

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

#define LOCAL
#define L(i, j, n) for (int i = (j); i < (int)n; i ++)
#define LI(i, j, n) for (int i = (j); i <= (int)n; i ++)
#define R(i, j, n) for (int i = (j); i > (int)n; i --)
#define RI(i, j, n) for (int i = (j); i >= (int)n; i --)
#define SZ(x) int((x).size())
#define ALL(x) begin(x),end(x)
#define IS_IN(x, v) ((x).find(v) != (x).end())
#define vec vector
#define pb push_back

using ll = long long;
using ld = long double;
using pii = pair<int, int>;
using pil = pair<int, ll>;
using pli = pair<ll, int>;
using pll = pair<ll, ll>;


const int N = (int)2e5+5;
const int MOD = (int)1e9 + 7;
const int oo = (int)1e9;

void solve()
{
}


int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    int TT = 1;
    //cin >> TT;
    while (TT--)
    {
        solve();
    }
}
```

1.2 Fast Python

```python
import os, sys, io
finput = io.BytesIO(os.read(0, os.fstat(0).st_size)).readline
fprint = sys.stdout.write
```

1.3 Policy Based

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<typename Key, typename Val=null_type>
using indexed_set = tree<Key, Val, less<Key>, rb_tree_tag,
                         tree_order_statistics_node_update>;
// indexed_set<char> s;
// char val = *s.find_by_order(0); // acceso por indice
// int idx = s.order_of_key('a'); // busca indice del valor
template<class Key,class Val=null_type>using htable=gp_hash_table<Key,
    Val>;
// como unordered_map (o unordered_set si Val es vacio), pero sin metodo
     count
```

2 Search

2.1 Ternary

```cpp
// Minimo de `f` en `(l,r)`.
template<class Fun>ll ternary(Fun f, ll l, ll r) {
  for (ll d = r-l; d > 2; d = r-l) {
    ll a = l + d/3, b = r - d/3;
    if (f(a) > f(b)) l = a; else r = b;
  }
  return l + 1;
}
// para error < EPS, usar iters=log((r-l)/EPS)/log(1.618)
template<class Fun>double golden(Fun f, double l, double r, int iters){
  double const ratio = (3-sqrt(5))/2;
  double x1=l+(r-l)*ratio, x2=r-(r-l)*ratio, f1=f(x1), f2=f(x2);
  while (iters--) {
    if (f1 > f2) l=x1, x1=x2, f1=f2, x2=r-(r-l)*ratio, f2=f(x2);
    else         r=x2, x2=x1, f2=f1, x1=l+(r-l)*ratio, f1=f(x1);
  }
```

```
  return (l+r)/2;
}
```

3 Data structures

3.1 BIT

```
#define LSOne(S) (S & -S)

struct BIT {
    vector<int> B;
    int n;
    BIT(int n = 1): B(n + 1), n(n+1){}
    BIT(vector<int> &v): B(v.size()+1), n(v.size()+1) {
        for (int i = 1; i <= n; i ++){
            B[i] += v[i-1];
            if (i + LSOne(i) <= n){
                B[i + LSOne(i)] += B[i];
            }
        }
    }
    void update(int i, int x){
        while (i <= n){
            B[i] += x;
            i += LSOne(i);
        }
    }
    int sum(int i){
        int res = 0;
        while (i > 0){
            res += B[i];
            i -= LSOne(i);
        }
        return res;
    }
    int range_sum(int l, int r){
        return sum(r) - sum(l - 1);
    }
};
```

3.2 DSU

```
struct DSU {
    vector<int> par, sz;
    int n;
    DSU(int n = 1): par(n), sz(n, 1), n(n) {
        for (int i = 0;i < n; i ++) par[i] = i;
    }
    int find(int a){
        return a == par[a] ? a : par[a] = find(par[a]);
    }
    void join(int a, int b){
        a=find(a);
        b=find(b);
        if (a != b){
            if (sz[b] > sz[a]) swap(a,b);
            par[b] = a;
            sz[a] += sz[b];
        }
    }
};
```

3.3 Sparse Table

```
int log2_floor(unsigned long long i) {
    return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}

const int MAXN = 10;
int K = log2_floor(MAXN);
int st[K + 1][MAXN];

// Load Array to st[0][i]
std::copy(array.begin(), array.end(), st[0]);

// Build
for (int i = 1; (1 << i) <= n; i ++){
    for (int j = 0; j + (1 << (i - 1)) < n; j ++){
        st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i - 1))]);
    }
}

// Query
int min_range(int l, int r){
    int C = log2_floor(r - l + 1);
    return min(st[C][l], st[C][r - (1 << C) + 1]);
```

```
}
```

3.4 Segment tree

```
struct Node {
    long long sum = 0;
    long long min_val = LLONG_MAX;
    long long max_val = LLONG_MIN;
    long long lazy = 0;

    // Merge function to combine two nodes
    void merge(const Node& left, const Node& right) {
        sum = left.sum + right.sum;
        min_val = min(left.min_val, right.min_val);
        max_val = max(left.max_val, right.max_val);
    }

    // Update function for lazy propagation
    void apply(int l, int r, long long value) {
        sum += (r - l + 1) * value;
        min_val += value;
        max_val += value;
        lazy += value;
    }
};

struct SegTree {
    int n;
    vector<Node> tree;

    SegTree(int n) : n(n) {
        tree.resize(4 * n + 5);
    }

    SegTree(vector<int>& arr) : n(arr.size()) {
        tree.resize(4 * n + 5);
        build(arr, 0, 0, n-1);
    }

    // Push lazy value to children
    void push(int id, int l, int r) {
        if (tree[id].lazy == 0) return;

        int mid = (l + r) >> 1;
        tree[2*id + 1].apply(l, mid, tree[id].lazy);
        tree[2*id + 2].apply(mid+1, r, tree[id].lazy);
        tree[id].lazy = 0;
    }

    void build(vector<int>& arr, int id, int l, int r) {
        if (l == r) {
            tree[id].sum = arr[l];
            tree[id].min_val = arr[l];
            tree[id].max_val = arr[l];
            return;
        }

        int mid = (l + r) >> 1;
        build(arr, 2*id + 1, l, mid);
        build(arr, 2*id + 2, mid+1, r);
        tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
    }

    // Range update with lazy propagation
    void update(int id, int l, int r, int ql, int qr, long long val) {
        if (ql > r || qr < l) return;

        if (ql <= l && r <= qr) {
            tree[id].apply(l, r, val);
            return;
        }

        push(id, l, r);
        int mid = (l + r) >> 1;
        update(2*id + 1, l, mid, ql, qr, val);
        update(2*id + 2, mid+1, r, ql, qr, val);
        tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
    }

    // Range query
    Node query(int id, int l, int r, int ql, int qr) {
        if (ql > r || qr < l) return Node();

        if (ql <= l && r <= qr) {
            return tree[id];
        }
```

```
        push(id, l, r);
        int mid = (l + r) >> 1;
        Node left = query(2*id + 1, l, mid, ql, qr);
        Node right = query(2*id + 2, mid+1, r, ql, qr);

        Node result;
        result.merge(left, right);
        return result;
    }

    // Public interface
    void update(int l, int r, long long val) {
        update(0, 0, n-1, l, r, val);
    }

    Node query(int l, int r) {
        return query(0, 0, n-1, l, r);
    }
};
```

4 Dynamic Programming

4.1 Knapsack

```
int knapsack(vector<int>& values, vector<int>& weights, int W) {
    int n = values.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for(int i = 1; i <= n; i++) {
        for(int w = 0; w <= W; w++) {
            if(weights[i-1] <= w) {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w-weights[i-1]] + values[i-1]);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][W];
}
```

4.2 LIS

```
vector<int> getLIS(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n + 1, INT_MAX);  // dp[i] = smallest value that ends
        an LIS of length i
    vector<int> len(n);              // Length of LIS ending at each
        position
    dp[0] = INT_MIN;

    for(int i = 0; i < n; i++) {
        int j = upper_bound(dp.begin(), dp.end(), arr[i]) - dp.begin();
        dp[j] = arr[i];
        len[i] = j;
    }

    // Find maxLen and reconstruct sequence
    int maxLen = 0;
    for(int i = n-1; i >= 0; i--) maxLen = max(maxLen, len[i]);

    vector<int> lis;
    for(int i = n-1, currLen = maxLen; i >= 0; i--) {
        if(len[i] == currLen) {
            lis.push_back(arr[i]);
            currLen--;
        }
    }
    reverse(lis.begin(), lis.end());
    return lis;
}
```

4.3 Edit Distance

```

//3. Edit Distance - O(n*m)
int editDistance(string& s1, string& s2) {
    int n = s1.length(), m = s2.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1));

    // Base cases
    for(int i = 0; i <= n; i++) dp[i][0] = i;
    for(int j = 0; j <= m; j++) dp[0][j] = j;

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

```
            if(s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1];
            } else {
                dp[i][j] = 1 + min({dp[i-1][j],    // deletion
                                    dp[i][j-1],    // insertion
                                    dp[i-1][j-1]}); // replacement
            }
        }
    }
    return dp[n][m];
}
```

4.4 Kadane

```
pair<int, pair<int,int>> kadane(vector<int>& arr) {
    int maxSoFar = arr[0], maxEndingHere = arr[0];
    int start = 0, end = 0, s = 0;

    for(int i = 1; i < arr.size(); i++) {
        if(maxEndingHere + arr[i] < arr[i]) {
            maxEndingHere = arr[i];
            s = i;
        } else {
            maxEndingHere += arr[i];
        }

        if(maxEndingHere > maxSoFar) {
            maxSoFar = maxEndingHere;
            start = s;
            end = i;
        }
    }
    return {maxSoFar, {start, end}}; // max, l, r
}
```

5 Strings

5.1 Prefix Trie

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

using namespace std;

struct TrieNodeStruct {
    TrieNodeStruct* children[26];
    bool isEndOfWord;

    TrieNodeStruct() {
        isEndOfWord = false;
        for(int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};

struct TrieStruct {
    TrieNodeStruct* root;

    TrieStruct() {
        root = new TrieNodeStruct();
    }

    void insert(string word) {
        TrieNodeStruct* current = root;
        for(char c : word) {
            int index = c - 'a';
            if(current->children[index] == nullptr) {
                current->children[index] = new TrieNodeStruct();
            }
            current = current->children[index];
        }
        current->isEndOfWord = true;
    }
};
```

5.2 Hashing

```
static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
static constexpr ll b = 500'000'000;
struct StrHash { // Hash polinomial con exponentes decrecientes.
  vector<ll> hs[2], bs[2];
  StrHash(string const& s) {
    int n = SZ(s);
    L(k, 0, 2) {
      hs[k].resize(n+1), bs[k].resize(n+1, 1);
      L(i, 1, n) {
```

```
        hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
        bs[k][i+1] =  bs[k][i] * b         % ms[k];
      }
    }
  }
  ll get(int idx, int len) const { // Hashes en `s[idx, idx+len)`.
    ll h[2];
    L(k, 0, 2) {
      h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
      if (h[k] < 0) h[k] += ms[k];
    }
    return (h[0] << 32) | h[1];
  }
};

pll union_hash(vector<pll> hs, vector<ll> lens){ //use arrays makes it
    slower
  ll len = 0;
  for(int i = hs.size()-1; i > 0; i--){
    len += lens[i];
    pll& [l1, l2] = hs[i];
    pll& [r1, r2] = hs[i-1];
    l1 = ((l1 * binpow(b, len, ms[0])) % ms[0] + r1) % ms[0];
    l2 = ((l2 * binpow(b, len, ms[1])) % ms[1] + r2) % ms[1];
  }

  return hs[0];
}
```

5.3 KMP

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

vector<int> kmp(string pat, string sec){ //geeks4geeks implementation
    with some changes
  int m = pat.length();
  int n = sec.length();
  cout << m << "␣" << n << endl;

  vector<int> lps = getLps(pat);
  vector<int> res;

  int i = 0;
  int j = 0;

  while((n - i) >= (m - j)){
    if(pat[j] == sec[i]){
      i++;
      j++;
    }
    if(j == m){
      res.push_back(i - j);
      j = lps[j - 1];
    }
    else{
      if(i < n && pat[j] != sec[i]){
        if(j != 0) j = lps[ j - 1 ];
        else i = i + 1;
      }
    }
  }

  return res;
}
```

5.4 LPS

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

vector<int> getLps(string pat){ //geek4geeks implementatio with some
    changes
  vector<int> lps(pat.length(), 0);
  int len = 0;
  int i = 1;
  lps[0] = 0;
  while(i < pat.length()){
    if(pat[i] == pat[len]){
      len++;
      lps[i] = len;
      i++;
    }
    else //pat[i] != pat[len]
    {
      lps[i] = 0;
```

```
      i++;
    }
  }

  return lps;
}
```

5.5 Z-FUNCTION

```
template<class Char=char>vector<int> zfun(const basic_string<Char>& w) {
  int n = SZ(w), l = 0, r = 0; vector<int> z(n);
  z[0] = w.length();
  L(i, 1, n) {
    if (i <= r) {z[i] = min(r - i + 1, z[i - l]);}
    while (i + z[i] < n && w[z[i]] == w[i + z[i]]) {++z[i];}
    if (i + z[i] - 1 > r) {l = i, r = i + z[i] - 1;}
  }
  return z;
}
```

6 Graph

6.1 Tarjan

```
const int N = 10;

vector<int> G[N];
vector<int> dfs_low(N, -1), dfs_num(N, -1), ap(N, 0); // ap for
    Articulation Points
int dfs_count = 0;
int root = -1; // For AP

void dfs(int u, int p = -1){
    dfs_low[u]=dfs_num[u]=dfs_count++;
    int child = 0;
    for (int v: G[u]){
        if (v == p) continue;
        if (dfs_num[v] == -1){
            child ++;
            dfs(v, u);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
            if (dfs_low[v] > dfs_num[u]){
              // Bridge from u -> v
                cout << "Bridge " << u << " -> " << v << "\n";
            }
            if (dfs_low[v] >= dfs_num[u]) {
                // u is AP
                ap[u] = 1;
            }
        } else dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
    if (u == root){
        ap[u] = child > 1;
    }
}
```

6.2 Bellman Ford

```
struct Edge {
    int a, b, cost;
};

int n, m, v;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n, INF);
    d[v] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (Edge e : edges)
            if (d[e.a] < INF)
                d[e.b] = min(d[e.b], d[e.a] + e.cost);
}
```

6.3 SCC

```
vector<int> dfs_num(N, -1), dfs_low(N, -1), visited(N);
int dfs_count = 0;
int numSCC = 0;
stack<int> st;
void dfs(int u){
  dfs_low[u]=dfs_num[u]=dfs_count++;
  st.push(u);
  visited[u] = 1;
  for(int v: G[u]) {
```

```
    if (dfs_num[v] == -1) dfs(v);
    if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }
  if (dfs_num[u] == dfs_low[u]){
    numSCC ++;
    while(1){
      int v = st.top(); st.pop();
      visited[v] = 0;
      if (u == v) break;
    }
  }
}
```

6.4 Bipartite Matching Hopcroft-Karp

```
int L_S, R_S;
vec<int> G[S_MX]; // S_MX (Maxima cantidad de nodos de un lado)
int mat[S_MX];  // matching [0,L_S) -> [0,R_S)
int inv[S_MX];  // matching [0,R_S) -> [0,L_S)
int hopkarp() {
    fill(mat,mat+L_S,-1);
    fill(inv,inv+R_S,-1);
    int size = 0;
    vector<int> d(L_S);
    auto bfs = [&] {
        bool aug = false;
        queue<int> q;
        L(u, 0, L_S) if (mat[u] < 0) q.push(u); else d[u] = -1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : G[u]) {
                if (inv[v] < 0) aug = true;
                else if (d[inv[v]] < 0) d[inv[v]] = d[u] + 1, q.push(inv
                    [v]);
            }
        }
        return aug;
    };
    auto dfs = [&](auto&& me, int u) -> bool {
        for (auto v : G[u]) if (inv[v] < 0) {
            mat[u] = v, inv[v] = u;
            return true;
        }
        for (auto v : G[u]) if (d[inv[v]] > d[u] && me(me,inv[v])) {
            mat[u] = v, inv[v] = u;
            return true;
        }
        d[u] = 0;
        return false;
    };
    while (bfs()) L(u, 0, L_S) if (mat[u] < 0) size += dfs(dfs,u);
    return size;
}
```

6.5 Konig Theorem Min V.Cover

```
vec<int> cover[2]; // if cover[i][j] = 1 -> node i, j is part of cover
int konig() {
    cover[0].assign(L_S,true); // L_S left size
    cover[1].assign(R_S,false); // R_S right size
    int size = hopkarp();  // alternativamente, tambien funciona con
        Kuhn
    auto dfs = [&](auto&& me, int u) -> void {
        cover[0][u] = false;
        for (auto v : g[u]) if (!cover[1][v]) {
            cover[1][v] = true;
            me(me,inv[v]);
        }
    };
    L(u,0,L_S) if (mat[u] < 0) dfs(dfs,u);
    return size;
}
```

6.6 Hungarian

```
using vi = vec<int>;
using vd = vec<ld>;
const ld INF = 1e100;    // Para max asignacion, INF = 0, y negar costos
bool zero(ld x) {return fabs(x) < 1e-9;}  // Para int/ll: return x==0;
vec<pii> ans; // Guarda las aristas usadas en el matching: [0..n)x[0..m)
struct Hungarian{
  int n; vec<vd> cs; vi vL, vR;
  Hungarian(int N, int M) : n(max(N,M)), cs(n,vd(n)), vL(n), vR(n){
    L(x, 0, N) L(y, 0, M) cs[x][y] = INF;
  }
  void set(int x, int y, ld c) { cs[x][y] = c; }
```

```
  ld assign(){
    int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
    L(i, 0, n) u[i] = *min_element(ALL(cs[i]));
    L(j, 0, n){
      v[j] = cs[0][j]-u[0];
      L(i, 1, n) v[j] = min(v[j], cs[i][j] - u[i]);
    }
    vL = vR = vi(n, -1);
    L(i,0, n) L(j, 0, n) if(vR[j] == -1 and zero(cs[i][j] - u[i] - v[j])
        ){
      vL[i] = j; vR[j] = i; mat++; break;
    }
    for(; mat < n; mat ++){
      int s = 0, j = 0, i;
      while(vL[s] != -1) s++;
      fill(ALL(dad), -1); fill(ALL(sn), 0);
      L(k, 0, n) ds[k] = cs[s][k]-u[s]-v[k];
      while(true){
        j = -1;
        L(k, 0, n) if(!sn[k] and (j == -1 or ds[k] < ds[j])) j = k;
        sn[j] = 1; i = vR[j];
        if(i == -1) break;
        L(k, 0, n) if(!sn[k]){
          auto new_ds = ds[j] + cs[i][k] - u[i]-v[k];
          if(ds[k] > new_ds) ds[k]=new_ds, dad[k]=j;
        }
      }
      L(k, 0, n) if(k!=j and sn[k]){
        auto w = ds[k]-ds[j]; v[k] += w, u[vR[k]] -= w;
      }
      u[s] += ds[j];
      while(dad[j] >= 0){ int d = dad[j]; vR[j] = vR[d]; vL[vR[j]] = j;
          j = d; }
      vR[j] = s; vL[s] = j;
    }
    ld value = 0; L(i, 0, n) value += cs[i][vL[i]], ans.pb({i, vL[i]});
    return value;
  }
};
```

6.7 Flow

```
// Complexity (V * V * E);
```

```
struct Dinic {
    struct Edge {
        int to, rev;
        long long cap, flow;
        Edge(int to, int rev, long long cap) :
            to(to), rev(rev), cap(cap), flow(0) {}
    };

    vector<vector<Edge>> g;
    vector<int> level, ptr;
    queue<int> q;
    int n, source, sink;
    const long long INF = 1e18;

    Dinic(int n, int s, int t) : n(n), source(s), sink(t) {
        g.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int from, int to, long long cap) {
        g[from].emplace_back(to, g[to].size(), cap);
        g[to].emplace_back(from, g[from].size()-1, 0);  // Reverse edge
    }

    bool bfs() {
        while(!q.empty()) {
            q.pop();
        }
        fill(level.begin(), level.end(), -1);

        q.push(source);
        level[source] = 0;

        while(!q.empty() && level[sink] == -1) {
            int v = q.front();
            q.pop();

            for(const Edge& e : g[v]) {
                if(level[e.to] == -1 && e.flow < e.cap) {
                    level[e.to] = level[v] + 1;
                    q.push(e.to);
                }
```

```
            }
        }
        return level[sink] != -1;
    }

    long long dfs(int v, long long pushed) {
        if(v == sink || pushed == 0) return pushed;

        for(int& i = ptr[v]; i < (int)g[v].size(); i++) {
            Edge& e = g[v][i];

            if(level[e.to] != level[v] + 1 || e.flow >= e.cap) continue;

            long long flow = dfs(e.to, min(pushed, e.cap - e.flow));
            if(flow == 0) continue;

            e.flow += flow;
            g[e.to][e.rev].flow -= flow;
            return flow;
        }
        return 0;
    }

    long long max_flow() {
        long long flow = 0;

        while(bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
            while(long long pushed = dfs(source, INF)) {
                flow += pushed;
            }
        }
        return flow;
    }

    // Get the actual flow passing through each edge
    vector<vector<long long>> get_flow() {
        vector<vector<long long>> flow(n, vector<long long>(n, 0));
        for(int v = 0; v < n; v++) {
            for(const Edge& e : g[v]) {
                if(e.cap > 0) {  // Only original edges, not residual
                    flow[v][e.to] = e.flow;
                }
            }
        }
        return flow;
    }

    // Find minimum cut
    vector<bool> min_cut() {
        vector<bool> reachable(n, false);
        queue<int> q;
        q.push(source);
        reachable[source] = true;

        while(!q.empty()) {
            int v = q.front();
            q.pop();

            for(const Edge& e : g[v]) {
                if(!reachable[e.to] && e.flow < e.cap) {
                    reachable[e.to] = true;
                    q.push(e.to);
                }
            }
        }
        return reachable;
    }
};

// Example usage:
/*
int main() {
    // Example: 6 vertices, source = 0, sink = 5
    int n = 6;
    Dinic flow(n, 0, 5);

    // Add edges: (from, to, capacity)
    flow.add_edge(0, 1, 16);
    flow.add_edge(0, 2, 13);
    flow.add_edge(1, 2, 10);
    flow.add_edge(1, 3, 12);
    flow.add_edge(2, 1, 4);
    flow.add_edge(2, 4, 14);
    flow.add_edge(3, 2, 9);
    flow.add_edge(3, 5, 20);
```

```
    flow.add_edge(4, 3, 7);
    flow.add_edge(4, 5, 4);

    // Calculate maximum flow
    long long max_flow = flow.max_flow();
    cout << "Maximum flow: " << max_flow << "\n";

    // Get minimum cut
    vector<bool> cut = flow.min_cut();
    cout << "Vertices on source side of min cut: ";
    for(int i = 0; i < n; i++) {
        if(cut[i]) cout << i << " ";
    }
    cout << "\n";

    // Get flow through each edge
    auto flow_matrix = flow.get_flow();
    cout << "Flow matrix:\n";
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(flow_matrix[i][j] > 0) {
                cout << i << " -> " << j << ": " << flow_matrix[i][j] <<
                    "\n";
            }
        }
    }

    return 0;
}
*/
```

6.8 Ford Fulkerson

```
#define ll long long
const ll INF = (ll)4e18;
struct Edge {
    int from, to;
    ll cap, flow;
    Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow
        (0) {}
};

struct MaxFlow {
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<int> level, ptr;
    int n;
    queue<int> q;

    MaxFlow(int n) : n(n) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int from, int to, ll cap) {
        edges.emplace_back(from, to, cap);
        edges.emplace_back(to, from, 0);
        adj[from].push_back(edges.size() - 2);
        adj[to].push_back(edges.size() - 1);
    }

    bool bfs(int s, int t) {
        while(!q.empty()) q.pop();
        fill(level.begin(), level.end(), -1);

        q.push(s);
        level[s] = 0;

        while(!q.empty() && level[t] == -1) {
            int v = q.front();
            q.pop();

            for(int id : adj[v]) {
                if(level[edges[id].to] == -1 && edges[id].cap - edges[id
                    ].flow > 0) {
                    level[edges[id].to] = level[v] + 1;
                    q.push(edges[id].to);
                }
            }
        }
        return level[t] != -1;
    }

    ll dfs(int v, int t, ll pushed) {
        if(v == t || pushed == 0)
```

```
            return pushed;

        for(; ptr[v] < (int)adj[v].size(); ptr[v]++) {
            int id = adj[v][ptr[v]];
            int u = edges[id].to;

            if(level[u] != level[v] + 1) continue;

            ll tr = dfs(u, t, min(pushed, edges[id].cap - edges[id].flow
                ));
            if(tr > 0) {
                edges[id].flow += tr;
                edges[id ^ 1].flow -= tr;
                return tr;
            }
        }
        return 0;
    }

    ll max_flow(int s, int t) {
        ll flow = 0;
        while(bfs(s, t)) {
            fill(ptr.begin(), ptr.end(), 0);
            while(ll pushed = dfs(s, t, LLONG_MAX)) {
                flow += pushed;
            }
        }
        return flow;
    }

    vector<ll> get_flows() {
        vector<ll> flows;
        for(int i = 0; i < edges.size(); i += 2) {
            flows.push_back(edges[i].flow);
        }
        return flows;
    }
};
```

7 Math

7.1 Euclidean Extended

```
ll extendedGCD(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll gcd = extendedGCD(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}

bool findSolutionWithConstraints(ll a, ll b, ll c, ll x_min, ll y_min,
    ll &x, ll &y) {
    ll g = extendedGCD(a, b, x, y);

    if (c % g != 0) return false;

    x *= c / g;
    y *= c / g;

    // Ajustamos las variables a/g y b/g para mover las soluciones
    a /= g;
    b /= g;

    if (x < x_min) {
        ll k = (x_min - x + b - 1) / b;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (x > x_min) {
        ll k = (x - x_min) / b;
        x -= k * b;
        y += k * a;
    }

    if (y < y_min) {
        ll k = (y_min - y + a - 1) / a;  // Redondeo hacia arriba
        x += k * b;
        y -= k * a;
    } else if (y > y_min) {
        ll k = (y - y_min) / a;
        x -= k * b;
```

```
        y += k * a;
    }

    return x >= x_min && y >= y_min;
}
```

7.2 Euler Totient

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


vector<ll> compute_totients(ll n) {
    vector<ll> phi(n + 1);
    for (ll i = 0; i <= n; i++) {
        phi[i] = i;
    }

    for (ll i = 2; i <= n; i++) {
        if (phi[i] == i) { // i es primo
            for (ll j = i; j <= n; j += i) {
                phi[j] = phi[j] * (i - 1) / i;
            }
        }
    }

    return phi;
}
```

7.3 Josephus

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

typedef long long ll;

ll josephus_iterative(ll n, ll k) {
    ll result = 0;
    for (ll i = 2; i <= n; ++i) {
        result = (result + k) % i;
    }
    return result;
}


ll josephus_recursive(ll n, ll k) {

    if (n == 1)
        return 0;

    return (josephus_recursive(n - 1, k) + k) % n;
}


ll josephus_power_of_2(ll n) {

    ll power = 1;
    while (power <= n) {
        power <<= 1;
    }
    power >>= 1;


    return 2 * (n - power);
}
```

7.4 Mobius

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


vector<ll> compute_mobius(ll n) {
    vector<ll> mu(n + 1, 1);
    vector<bool> is_prime(n + 1, true);

    for (ll i = 2; i <= n; i++) {
        if (is_prime[i]) { // i es un primo
            for (ll j = i; j <= n; j += i) {
                mu[j] *= -1; // Multiplicamos por -1 para cada primo
                is_prime[j] = false;
            }
            for (ll j = i * i; j <= n; j += i * i) {
                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
                    en 0
```

```
            }
        }
    }

    return mu;
}


ll mobius(ll x) {
    ll count = 0;
    for (ll i = 2; i * i <= x; i++) {
        if (x % (i * i) == 0)
            return 0;
        if (x % i == 0) {
            count++;
            x /= i;
        }
    }

    if (x > 1) count++;

    return (count % 2 == 0) ? 1 : -1;
}
```

7.5 NTT

```
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
typedef long long ll;
const ll mod = 998244353;
const ll root = 31;
const ll root_1 = inverse(root, mod);
const ll root_pw = 1 << 23;

ll inverse(ll a, ll m) {
    ll res = 1, exp = m - 2;
    while (exp) {
        if (exp % 2 == 1) res = (1LL * res * a) % m;
        a = (1LL * a * a) % m;
        exp /= 2;
    }
    return res;
}

void ntt(vector<ll> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = inverse(n, mod);
        for (auto & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
    vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    ll n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
```

```
    fa.resize(n);
    fb.resize(n);

    ntt(fa, false);
    ntt(fb, false);
    for (ll i = 0; i < n; i++)
        fa[i] = (fa[i] * fb[i]) % mod;
    ntt(fa, true);

    vector<ll> result(n);
    for (ll i = 0; i < n; i++)
        result[i] = fa[i];
    return result;
}
```

7.6 FFT

```
typedef long long ll;
typedef complex<double> C;
typedef vector<double> vd;
typedef vector<ll> vll;
const double PI = acos(-1);

void fft(vector<C>& a) {
    int n = a.size(), L = 31 - __builtin_clz(n);
    static vector<C> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0, PI / k);
        for (int i = k; i < 2 * k; i++)
            rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vector<int> rev(n);
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) /
        2;
    for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) for (int j = 0; j < k; j++) {
            auto x = (double*)&rt[j + k], y = (double*)&a[i + j + k];
            C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
}

vll multiply(const vll& a, const vll& b) {
    if (a.empty() || b.empty()) return {};
    vd fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int L = 32 - __builtin_clz(fa.size() + fb.size() - 1), n = 1 << L;
    vector<C> in(n), out(n);

    for (int i = 0; i < a.size(); i++) in[i] = C(fa[i], 0);
    for (int i = 0; i < b.size(); i++) in[i].imag(fb[i]);

    fft(in);
    for (C& x : in) x *= x;
    for (int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
          // Corregido aqui
    fft(out);

    vll res(a.size() + b.size() - 1);
    for (int i = 0; i < res.size(); i++) {
        res[i] = llround(imag(out[i]) / (4 * n));
    }
    return res;
}
```

7.7 Rho

```
//RECOMENDADO USAR UNSIGNED LONG LONG
ll mulmod(ll a, ll b, ll m) { return ll(__int128(a) * b % m); }

ll expmod(ll b, ll e, ll m) {
  ll ret = 1;
  while (e) {
    if (e%2) ret = mulmod(ret, b, m);
    b = mulmod(b, b, m);
    e /= 2;
  }
  return ret;
}
bool miller(ll n) {
    if (n < 2) return false;
    for (ll p : {2, 3, 5, 7, 11, 13, 17, 19}) {
        if (n % p == 0)
```

```
            return (n == p);
        }
        if (n < 529) return true; // 23^2 = 529
        int s = 0;
        ll d = n - 1;
        while ((d & 1) == 0) {
            d >>= 1;
            s++;
        }
        auto witness = [&](ll a) {
            ll x = expmod(a % n, d, n);
            if (x == 1 || x == n - 1) return false;
            for (int i = 1; i < s; i++) {
                x = mulmod(x, x, n);
                if (x == n - 1) return false;
            }
            return true; // "true" => 'a' es testigo => n es compuesto
        };
        // Bases para 64 bits
        for (ll b : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
            if (b % n == 0) return true;
            if (witness(b)) return false;
        }
        return true; // "probable primo"
};
ll rho(ll n) {
  if(n % 2 == 0) return 2;
  ll x = 2, y = 2, d = 1;
  ll c = rand() % n + 1;
  while(d == 1) {
    x = (mulmod(x, x, n) + c) % n;
    y = (mulmod(y, y, n) + c) % n;
    y = (mulmod(y, y, n) + c) % n;
    d = __gcd(x - y, n);
  }
  return d == n ? rho(n) : d;
}
void fact(ll n, map<ll, int>& F) { // Agrega los factores de n en F
  if (n == 1) return;
  if (miller(n)) {F[n]++; return;}
  ll q = rho(n); fact(q, F); fact(n / q, F);
}
```

7.8 Simpson

```
ld simpsonRule(function<ld(ld)> f, ld a, ld b, int n) {
    // Asegurarse de que n sea par
    if (n % 2 != 0) {
        n++;
    }
    ld h = (b - a) / n;
    ld s = f(a) + f(b);

    // Suma de terminos interiores con los factores apropiados
    for (int i = 1; i < n; i++) {
        ld x = a + i * h;
        s += (i % 2 == 1 ? 4.0L : 2.0L) * f(x);
    }
    // Multiplica por h/3
    return (h / 3.0L) * s;
}
// Ejemplo: integrar la funcion x^2 entre 0 y 3
auto f = [&](ld x){ return x * x; };
ld a = 0.0L, b = 3.0L;
int n = 1000; // numero de subintervalos
ld resultado = simpsonRule(f, a, b, n);
```

8 Geometry

8.1 Convex Hull

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

typedef long long ll;
typedef pair<ll, ll> Point;

ll cross_product(Point O, Point A, Point B) {
    return (A.first - O.first) * (B.second - O.second) - (A.second - O.
        second) * (B.first - O.first);
}

vector<Point> convex_hull(vector<Point>& points) {
    sort(points.begin(), points.end());
```

```
    points.erase(unique(points.begin(), points.end()), points.end());
    vector<Point> hull;

    // Parte inferior
    for (const auto& p : points) {
        while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], p) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != p) {
            hull.push_back(p);
        }
    }

    // Parte superior
    int t = hull.size() + 1;
    for (int i = points.size() - 1; i >= 0; --i) {
        while (hull.size() >= t && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], points[i]) < 0)
            hull.pop_back();
        if (hull.empty() || hull.back() != points[i]) {
            hull.push_back(points[i]);
        }
    }

    hull.pop_back();
    return hull;
}
```

8.2 Operations

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

typedef long long ll;


ll cross_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * y2 - y1 * x2;
}


double distancia(pair<ll, ll> P1, pair<ll, ll> P2) {
    return sqrt((P2.first - P1.first) * (P2.first - P1.first) +
                (P2.second - P1.second) * (P2.second - P1.second));
}


ll dot_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
    ll x1 = P2.first - P1.first;
    ll y1 = P2.second - P1.second;
    ll x2 = P3.first - P1.first;
    ll y2 = P3.second - P1.second;
    return x1 * x2 + y1 * y2;
}
```

8.3 Polygon Area

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

typedef long long ll;
typedef pair<ll, ll> Point;


double polygon_area(const vector<Point>& polygon) {
    ll area = 0;
    int n = polygon.size();
    for (int i = 0; i < n; ++i) {
        ll j = (i + 1) % n;
        area += (polygon[i].first * polygon[j].second - polygon[i].
            second * polygon[j].first);
    }
    return abs(area) / 2.0;
}
```

8.4 Ray Casting

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

```

```cpp
typedef long long ll;
typedef pair<ll, ll> Point;


bool is_point_in_polygon(const vector<Point>& polygon, Point p) {
    bool inside = false;
    int n = polygon.size();
    for (int i = 0, j = n - 1; i < n; j = i++) {
        if ((polygon[i].second > p.second) != (polygon[j].second > p.second) &&
            p.first < (polygon[j].first - polygon[i].first) * (p.second - polygon[i].second) /
                  (polygon[j].second - polygon[i].second) + polygon[i].first) {
            inside = !inside;
        }
    }
    return inside;
}
```

9 Trees

9.1 Centroid

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

#define L(i, j, n) for (int i = (j); i < int(n); i ++)
#define ii pair<int, int>
const int inf = 1e9;
const int N = 1e5;

vector<int> G[N];
int ct[N];
set<ii> dist[N];
int up[N][18];
int colors[N];
int depth[N];
int sz[N];
bool removed[N];
int n, root, L;

int getSize(int u, int p){
    int szi = 1;
    for(int v: G[u]){
        if (p == v || removed[v]) continue;
        szi += getSize(v, u);
    }
    return sz[u] = szi;
}

int centroid(int u, int tree_size, int p){
    for (int v: G[u]){
        if (v == p || removed[v]) continue;
        if (sz[v] * 2 > tree_size) return centroid(v, tree_size, u);
    }
    return u;
}

void build(int node, int tree_size, int p)
{
    getSize(node, - 1);
    int cen = centroid(node, tree_size, -1);
    removed[cen] = 1;
    ct[cen] = p;
    if (p == -1) root = cen;

    if (tree_size == 1) return;

    for (int v: G[cen]){
        if (removed[v]) continue;
        build(v, sz[v], cen);
    }

}

void update(int v){
    int u = v;
    while(v != -1){
        dist[v].insert(distance(u, v), v);
        v = par[v];
    }
    return res;
}

int query(int v){
```

```
    int u = v;
    int res = INT_MAX;
    while(v != -1){
        res = min(res, distance(u, v), dist[v].begin()->first); //
            Minimun
        v = par[v];
    }
    return res;
}
```

9.2 LCA

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

#define L(i, j, n) for (int i = (j); i < int(n); i ++)
#define ii pair<int, int>
const int inf = 1e9;
const int N = 1e5;

vector<int> G[N], ct[N];
set<ii> dist[N];
int up[N][18];
int colors[N];
int depth[N];
int sz[N];
bool removed[N];
int n, root, L;

int getSize(int u, int p){
    int szi = 1;
    for(int v: G[u]){
        if (p == v || removed[v]) continue;
        szi += getSize(v, u);
    }
    return sz[u] = szi;
}

int centroid(int u, int tree_size, int p){
    for (int v: G[u]){
        if (v == p || removed[v]) continue;
        if (sz[v] * 2 > tree_size) return centroid(v, tree_size, u);
    }
    return u;
}

void build(int node, int tree_size, int p)
{
    getSize(node, - 1);
    int cen = centroid(node, tree_size, -1);
    removed[cen] = 1;
    if (p != -1){
        ct[cen].push_back(p);
    } else root = cen;

    if (tree_size == 1) return;

    for (int v: G[cen]){
        if (removed[v]) continue;
        build(v, sz[v], cen);
    }

}

void dfs(int u, int p){
    up[u][0] = p;
    for (int i = 1; i <= L; i ++){
        if (up[u][i-1] != -1) up[u][i] = up[up[u][i-1]][i-1];
        else up[u][i] = -1;
    }
    for (int v: G[u]){
        if (v == p) continue;
        depth[v] = depth[u] + 1;
        dfs(v, u);
    }
}

int LCA(int u, int v){
    if (depth[u] < depth[v]) swap(u, v);
    for (int i = L; i >= 0; i --){
        if (up[u][i] != -1 && depth[up[u][i]] >= depth[v]){
            u = up[u][i];
        }
    }
    if (u == v) return u;
    for (int i = L; i>= 0; i --){
```

```
        if (up[u][i] != up[v][i] && up[u][i] != -1 && up[v][i] != -1){
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}

int dis(int u, int v){
    int cmm = LCA(u, v);
    // cout << u << " " << v << " " << cmm << "\n";
    return depth[u] + depth[v] - (2 * depth[cmm]);
}

void uup(int u, int node){
    dist[u].insert({dis(u, node), node});
    for (int v: ct[u])
        uup(v, node);
}

void update(int node){
    dist[node].insert({0, node});
    for (int v: ct[node])
        uup(v, node);
}

int qup(int u, int node){
    int mn = dis(node, u) + dist[u].begin()->first;
    for (int v: ct[u]) mn = min(mn, qup(v, node));
    return mn;
}

int query(int node){
    int mn = dist[node].begin()->first;
    for (int v: ct[node]) mn = min(mn, qup(v, node));
    return mn;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    int m; cin >> n >> m;
    L = log2(n);
    L(i, 1, n){
        int u, v; cin >> u >> v;
        u --; v --;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    L(i, 0, n){
        dist[i].insert({inf, i});
    }
    build(0, n, -1);
    L(i, 0, L + 1) up[root][i] = -1;
    run(root, -1);
    update(0);
    L(_q, 0, m){
        int op, node; cin >> op >> node;
        if (op == 2){
            cout << query(node-1) << '\n';
        } else {
            update(node-1); // Log Log
        }
    }
}
```