

Dividimos y No Conquistamos (D&!C)

Contents

1	Template	1
1.1	C++ Template	1
2	Data structures	2
2.1	BIT	2
2.2	DSU	2
2.3	Sparse Table	2
2.4	Segment tree	2
3	Dynamic Programming	4
3.1	Knapsack	4
3.2	LIS	4
3.3	Edit Distance	4
3.4	Kadane	5
4	Strings	5
4.1	Prefix Tree	5
4.2	HASHING	6
4.3	KMP	6
4.4	LPS	7
4.5	Z FUNCTION	7
5	Graph	8
5.1	Tarjan	8
5.2	Bellman Ford	8
5.3	SCC	8
5.4	Flow	8
5.5	Ford Fulkerson	10
6	Math	12
6.1	BINARY POW	12
6.2	CATALAN	12
6.3	COMBINATORICS	12
6.4	EUCLIDEAN EXTENDED	12
6.5	EULER TOTIENT	13
6.6	JOSEPHUS	13
6.7	MOBIUS	13

6.8	NTT	14
6.9	PRIME FACTORIZATION	15
6.10	SIEVE	15
6.11	fft	16
7	Geometry	18
7.1	CONVEX HULL	18
7.2	OPERATIONS	19
7.3	POLYGON AREA	19
7.4	RAY CASTING	19
8	Trees	20
8.1	Centroid	20
8.2	LCA	20

1 Template

1.1 C++ Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define TESTS
4 #define LOCAL
5
6 #define ll long long
7 #define ii pair<ll, ll>
8 #define F first
9 #define S second
10 #define forni(i, o, n) for (int i = o; i < n; i++)
11 #define forn(i, n) forn(i, 0, n)
12 #define pub push_back
13 #define popf pop_front
14
15 #ifdef LOCAL
16 #define DBG(x) cout << "[" << x << " ";
17 #else
18 #define DBG(x) 42
19 #endif
20
21
22 void solve(){
23
24 }
25
26
27 int main(){
28     ios::sync_with_stdio(0);cin.tie(0);
29     #ifdef LOCAL
30         freopen("in.txt","r", stdin);
31         freopen("out.txt","w", stdout);
32     #endif
33     int tt = 1;
34     #ifdef TESTS
35         cin >> tt;
36     #endif
37     while(tt--){solve();}
38 }

```

2 Data structures

2.1 BIT

```

1 #define LSOne(S) (S & -S)
2
3 struct BIT {
4     vector<int> B;
5     int n;
6     BIT(int n = 1): B(n + 1), n(n+1){}
7     BIT(vector<int> &v): B(v.size()+1), n(v.size()+1) {
8         for (int i = 1; i <= n; i++){
9             B[i] += v[i-1];
10            if (i + LSOne(i) <= n){
11                B[i + LSOne(i)] += B[i];
12            }
13        }
14    }
15    void update(int i, int x){
16        while (i <= n){
17            B[i] += x;
18            i += LSOne(i);
19        }
20    }
21    int sum(int i){
22        int res = 0;
23        while (i > 0){
24            res += B[i];
25            i -= LSOne(i);
26        }
27        return res;
28    }
29    int range_sum(int l, int r){
30        return sum(r) - sum(l - 1);
31    }
32 };

```

2.2 DSU

```

1 struct DSU {
2     vector<int> par, sz;
3     int n;
4     DSU(int n = 1): par(n), sz(n, 1), n(n) {

```

```

5     for (int i = 0; i < n; i++) par[i] = i;
6 }
7 int find(int a){
8     return a == par[a] ? a : par[a] = find(par[a]);
9 }
10 void join(int a, int b){
11     a=find(a);
12     b=find(b);
13     if (a != b){
14         if (sz[b] > sz[a]) swap(a,b);
15         par[b] = a;
16         sz[a] += sz[b];
17     }
18 }
19 };

```

2.3 Sparse Table

```

1 int log2_floor(unsigned long long i) {
2     return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
3 }
4
5 const int MAXN = 10;
6 int K = log2_floor(MAXN);
7 int st[K + 1][MAXN];
8
9 // Load Array to st[0][i]
10 std::copy(array.begin(), array.end(), st[0]);
11
12 // Build
13 for (int i = 1; (1 << i) <= n; i++){
14     for (int j = 0; j + (1 << (i - 1)) < n; j++){
15         st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i - 1))]);
16     }
17 }
18
19 // Query
20 int min_range(int l, int r){
21     int C = log2_floor(r - l + 1);
22     return min(st[C][l], st[C][r - (1 << C) + 1]);
23 }

```

2.4 Segment tree

```

1 struct Node {
2     long long sum = 0;
3     long long min_val = LLONG_MAX;
4     long long max_val = LLONG_MIN;
5     long long lazy = 0;
6
7     // Merge function to combine two nodes
8     void merge(const Node& left, const Node& right) {
9         sum = left.sum + right.sum;
10        min_val = min(left.min_val, right.min_val);
11        max_val = max(left.max_val, right.max_val);
12    }
13
14    // Update function for lazy propagation
15    void apply(int l, int r, long long value) {
16        sum += (r - l + 1) * value;
17        min_val += value;
18        max_val += value;
19        lazy += value;
20    }
21 };
22
23 struct SegTree {
24     int n;
25     vector<Node> tree;
26
27     SegTree(int n) : n(n) {
28         tree.resize(4 * n + 5);
29     }
30
31     SegTree(vector<int>& arr) : n(arr.size()) {
32         tree.resize(4 * n + 5);
33         build(arr, 0, 0, n-1);
34     }
35
36     // Push lazy value to children
37     void push(int id, int l, int r) {
38         if (tree[id].lazy == 0) return;
39
40         int mid = (l + r) >> 1;
41         tree[2*id + 1].apply(l, mid, tree[id].lazy);
42         tree[2*id + 2].apply(mid+1, r, tree[id].lazy);
43         tree[id].lazy = 0;

```

```

44 }
45
46 void build(vector<int>& arr, int id, int l, int r) {
47     if (l == r) {
48         tree[id].sum = arr[l];
49         tree[id].min_val = arr[l];
50         tree[id].max_val = arr[l];
51         return;
52     }
53
54     int mid = (l + r) >> 1;
55     build(arr, 2*id + 1, l, mid);
56     build(arr, 2*id + 2, mid+1, r);
57     tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
58 }
59
60 // Range update with lazy propagation
61 void update(int id, int l, int r, int ql, int qr, long long val) {
62     if (ql > r || qr < l) return;
63
64     if (ql <= l && r <= qr) {
65         tree[id].apply(l, r, val);
66         return;
67     }
68
69     push(id, l, r);
70     int mid = (l + r) >> 1;
71     update(2*id + 1, l, mid, ql, qr, val);
72     update(2*id + 2, mid+1, r, ql, qr, val);
73     tree[id].merge(tree[2*id + 1], tree[2*id + 2]);
74 }
75
76 // Range query
77 Node query(int id, int l, int r, int ql, int qr) {
78     if (ql > r || qr < l) return Node();
79
80     if (ql <= l && r <= qr) {
81         return tree[id];
82     }
83
84     push(id, l, r);
85     int mid = (l + r) >> 1;
86     Node left = query(2*id + 1, l, mid, ql, qr);

```

```

87         Node right = query(2*id + 2, mid+1, r, ql, qr);
88
89         Node result;
90         result.merge(left, right);
91         return result;
92     }
93
94     // Public interface
95     void update(int l, int r, long long val) {
96         update(0, 0, n-1, l, r, val);
97     }
98
99     Node query(int l, int r) {
100         return query(0, 0, n-1, l, r);
101     }
102 };

```

3 Dynamic Programming

3.1 Knapsack

```

1  int knapsack(vector<int>& values, vector<int>& weights, int W) {
2      int n = values.size();
3      vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));
4
5      for(int i = 1; i <= n; i++) {
6          for(int w = 0; w <= W; w++) {
7              if(weights[i-1] <= w) {
8                  dp[i][w] = max(dp[i-1][w],
9                                dp[i-1][w-weights[i-1]] + values[i-1]);
10             } else {
11                 dp[i][w] = dp[i-1][w];
12             }
13         }
14     }
15     return dp[n][W];
16 }

```

3.2 LIS

```
1 vector<int> getLIS(vector<int>& arr) {
2     int n = arr.size();
3     vector<int> dp(n + 1, INT_MAX); // dp[i] = smallest value that ends
    an LIS of length i
```

```

4   vector<int> len(n);           // Length of LIS ending at each
   position
5   dp[0] = INT_MIN;
6
7   for(int i = 0; i < n; i++) {
8       int j = upper_bound(dp.begin(), dp.end(), arr[i]) - dp.begin();
9       dp[j] = arr[i];
10      len[i] = j;
11  }
12
13  // Find maxLen and reconstruct sequence
14  int maxLen = 0;
15  for(int i = n-1; i >= 0; i--) maxLen = max(maxLen, len[i]);
16
17  vector<int> lis;
18  for(int i = n-1, currLen = maxLen; i >= 0; i--) {
19      if(len[i] == currLen) {
20          lis.push_back(arr[i]);
21          currLen--;
22      }
23  }
24  reverse(lis.begin(), lis.end());
25  return lis;
26 }

```

3.3 Edit Distance

```

1  //3. Edit Distance - O(n*m)
2  int editDistance(string& s1, string& s2) {
3      int n = s1.length(), m = s2.length();
4      vector<vector<int>> dp(n + 1, vector<int>(m + 1));
5
6
7      // Base cases
8      for(int i = 0; i <= n; i++) dp[i][0] = i;
9      for(int j = 0; j <= m; j++) dp[0][j] = j;
10
11     for(int i = 1; i <= n; i++) {
12         for(int j = 1; j <= m; j++) {
13             if(s1[i-1] == s2[j-1]) {
14                 dp[i][j] = dp[i-1][j-1];
15             } else {
16                 dp[i][j] = 1 + min({dp[i-1][j], // deletion

```

```

17         dp[i][j-1], // insertion
18         dp[i-1][j-1]}); // replacement
19     }
20 }
21 }
22 return dp[n][m];
23 }

```

3.4 Kadane

```

1  pair<int, pair<int,int>> kadane(vector<int>& arr) {
2      int maxSoFar = arr[0], maxEndingHere = arr[0];
3      int start = 0, end = 0, s = 0;
4
5      for(int i = 1; i < arr.size(); i++) {
6          if(maxEndingHere + arr[i] < arr[i]) {
7              maxEndingHere = arr[i];
8              s = i;
9          } else {
10             maxEndingHere += arr[i];
11         }
12
13         if(maxEndingHere > maxSoFar) {
14             maxSoFar = maxEndingHere;
15             start = s;
16             end = i;
17         }
18     }
19     return {maxSoFar, {start, end}}; // max, l, r
20 }

```

4 Strings

4.1 Prefix Tree

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  struct TrieNodeStruct {
6      TrieNodeStruct* children[26];
7      bool isEndOfWord;
8

```

```

9     TrieNodeStruct() {
10         isEndOfWord = false;
11         for(int i = 0; i < 26; i++) {
12             children[i] = nullptr;
13         }
14     }
15 };

16
17 struct TrieStruct {
18     TrieNodeStruct* root;
19
20     TrieStruct() {
21         root = new TrieNodeStruct();
22     }
23
24     void insert(string word) {
25         TrieNodeStruct* current = root;
26         for(char c : word) {
27             int index = c - 'a';
28             if(current->children[index] == nullptr) {
29                 current->children[index] = new TrieNodeStruct();
30             }
31             current = current->children[index];
32         }
33         current->isEndOfWord = true;
34     }
35
36     bool search(string word) {
37         TrieNodeStruct* current = root;
38         for(char c : word) {
39             int index = c - 'a';
40             if(current->children[index] == nullptr) {
41                 return false;
42             }
43             current = current->children[index];
44         }
45         return current->isEndOfWord;
46     }
47
48     bool startsWithDirect(string prefix) {
49         TrieNodeStruct* current = root;
50         for(char c : prefix) {
51             int index = c - 'a';

```

```

52         if(current->children[index] == nullptr) {
53             return false;
54         }
55         current = current->children[index];
56     }
57     return true;
58 }
59 };

```

4.2 HASHING

```

1 struct StrHash { // Hash polinomial con exponentes decrecientes.
2     static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
3     static constexpr ll b = 500'000'000;
4     vector<ll> hs[2], bs[2];
5     StrHash(string const& s) {
6         int n = sz(s);
7         forn(k, 2) {
8             hs[k].resize(n+1), bs[k].resize(n+1, 1);
9             forn(i, n) {
10                 hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
11                 bs[k][i+1] = bs[k][i] * b % ms[k];
12             }
13         }
14     }
15     ll get(int idx, int len) const { // Hashes en 's[idx, idx+len)'.
16         ll h[2];
17         forn(k, 2) {
18             h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
19             if (h[k] < 0) h[k] += ms[k];
20         }
21         return (h[0] << 32) | h[1];
22     }
23 };

24
25 pll union_hash(pll l, pll r, ll len_r){ //pll = pair<ll,ll>
26     l.first = ((l.first * binpow(b, len_r, ms[0])) % ms[0] + r.first) % ms[0];
27     l.second = ((l.second * binpow(b, len_r, ms[1])) % ms[1] + r.second) % ms[1];
28
29     return l;
30 }

```

4.3 KMP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<int> kmp(string pat, string sec){ //geeks4geeks implementation
    with some changes
5     int m = pat.length();
6     int n = sec.length();
7     cout << m << " " << n << endl;
8
9     vector<int> lps = getLps(pat);
10    vector<int> res;
11
12    int i = 0;
13    int j = 0;
14
15    while((n - i) >= (m - j)){
16        if(pat[j] == sec[i]){
17            i++;
18            j++;
19        }
20        if(j == m){
21            res.push_back(i - j);
22            j = lps[j - 1];
23        }
24        else{
25            if(i < n && pat[j] != sec[i]){
26                if(j != 0) j = lps[j - 1];
27                else i = i + 1;
28            }
29        }
30    }
31
32    return res;
33 }
34
35 vector<int> getLps(string pat){ //geek4geeks implementatio with some
    changes
36     vector<int> lps(pat.length(), 0);
37     int len = 0;
38     int i = 1;
39     lps[0] = 0;

```

```

40     while(i < pat.length()){
41         if(pat[i] == pat[len]){
42             len++;
43             lps[i] = len;
44             i++;
45         }
46         else //pat[i] != pat[len]
47         {
48             lps[i] = 0;
49             i++;
50         }
51     }
52
53     return lps;
54 }

```

4.4 LPS

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<int> getLps(string pat){ //geek4geeks implementatio with some
    changes
5     vector<int> lps(pat.length(), 0);
6     int len = 0;
7     int i = 1;
8     lps[0] = 0;
9     while(i < pat.length()){
10         if(pat[i] == pat[len]){
11             len++;
12             lps[i] = len;
13             i++;
14         }
15         else //pat[i] != pat[len]
16         {
17             lps[i] = 0;
18             i++;
19         }
20     }
21
22     return lps;
23 }

```

4.5 Z FUNCTION

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  vector<int> z_function(string s) {
6      int n = s.length();
7      vector<int> z(n, 0);
8
9      z[0] = n;
10
11     int l = 0, r = 0;
12
13     for(int i = 1; i < n; i++) {
14         if(i <= r) {
15             z[i] = min(r - i + 1, z[i - l]);
16         }
17
18         while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
19             z[i]++;
20         }
21
22         if(i + z[i] - 1 > r) {
23             l = i;
24             r = i + z[i] - 1;
25         }
26     }
27
28     return z;
29 }
30
31 vector<int> find_pattern(string text, string pattern) {
32
33     string s = pattern + "$" + text;
34     vector<int> z = z_function(s);
35     vector<int> result;
36
37
38     for(int i = pattern.length() + 1; i < s.length(); i++) {
39         if(z[i] == pattern.length()) {
40             result.push_back(i - pattern.length() - 1);
41         }
42     }
43 }

```

```

42     }
43 }
44
45     return result;
46 }

```

5 Graph

5.1 Tarjan

```

1  const int N = 10;
2
3  vector<int> G[N];
4  vector<int> dfs_low(N, -1), dfs_num(N, -1), ap(N, 0); // ap for
5  Articulation Points
6  int dfs_count = 0;
7  int root = -1; // For AP
8
9  void dfs(int u, int p = -1){
10     dfs_low[u]=dfs_num[u]=dfs_count++;
11     int child = 0;
12     for (int v: G[u]){
13         if (v == p) continue;
14         if (dfs_num[v] == -1){
15             child ++;
16             dfs(v, u);
17             dfs_low[u] = min(dfs_low[u], dfs_low[v]);
18             if (dfs_low[v] > dfs_num[u]){
19                 // Bridge from u -> v
20                 cout << "Bridge_ " << u << " -> " << v << "\n";
21             }
22             if (dfs_low[v] >= dfs_num[u]) {
23                 // u is AP
24                 ap[u] = 1;
25             }
26         } else dfs_low[u] = min(dfs_low[u], dfs_num[v]);
27     }
28     if (u == root){
29         ap[u] = child > 1;
30     }
31 }

```

5.2 Bellman Ford


```

1 struct Edge {
2     int a, b, cost;
3 };
4
5 int n, m, v;
6 vector<Edge> edges;
7 const int INF = 1000000000;
8
9 void solve()
10 {
11     vector<int> d(n, INF);
12     d[v] = 0;
13     for (int i = 0; i < n - 1; ++i)
14         for (Edge e : edges)
15             if (d[e.a] < INF)
16                 d[e.b] = min(d[e.b], d[e.a] + e.cost);
17 }

```

5.3 SCC

```

1 vector<int> dfs_num(N, -1), dfs_low(N, -1), visited(N);
2 int dfs_count = 0;
3 int numSCC = 0;
4 stack<int> st;
5 void dfs(int u){
6     dfs_low[u]=dfs_num[u]=dfs_count++;
7     st.push(u);
8     visited[u] = 1;
9     for(int v: G[u]) {
10         if (dfs_num[v] == -1) dfs(v);
11         if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
12     }
13     if (dfs_num[u] == dfs_low[u]){
14         numSCC ++;
15         while(1){
16             int v = st.top(); st.pop();
17             visited[v] = 0;
18             if (u == v) break;
19         }
20     }
21 }

```

5.4 Flow

```

1 // Complexity (V * V * E);
2 struct Dinic {
3     struct Edge {
4         int to, rev;
5         long long cap, flow;
6         Edge(int to, int rev, long long cap) :
7             to(to), rev(rev), cap(cap), flow(0) {}
8     };
9
10    vector<vector<Edge>> g;
11    vector<int> level, ptr;
12    queue<int> q;
13    int n, source, sink;
14    const long long INF = 1e18;
15
16    Dinic(int n, int s, int t) : n(n), source(s), sink(t) {
17        g.resize(n);
18        level.resize(n);
19        ptr.resize(n);
20    }
21
22    void add_edge(int from, int to, long long cap) {
23        g[from].emplace_back(to, g[to].size(), cap);
24        g[to].emplace_back(from, g[from].size()-1, 0); // Reverse edge
25    }
26
27    bool bfs() {
28        while(!q.empty()) {
29            q.pop();
30        }
31        fill(level.begin(), level.end(), -1);
32
33        q.push(source);
34        level[source] = 0;
35
36        while(!q.empty() && level[sink] == -1) {
37            int v = q.front();
38            q.pop();
39
40            for(const Edge& e : g[v]) {
41                if(level[e.to] == -1 && e.flow < e.cap) {
42                    level[e.to] = level[v] + 1;
43                    q.push(e.to);
44                }
45            }
46        }
47    }
48
49    long long flow() {
50        long long total_flow = 0;
51        while(bfs()) {
52            vector<int> p(n, -1);
53            p[source] = source;
54            int v = sink;
55            while(v != source) {
56                v = p[v];
57                for(Edge& e : g[v]) {
58                    if(e.to == source || e.to == v || e.flow == e.cap) continue;
59                    e.flow++;
60                    total_flow++;
61                    p[e.to] = v;
62                }
63            }
64        }
65        return total_flow;
66    }
67 }

```

```

44     }
45     }
46 }
47     return level[sink] != -1;
48 }
49
50 long long dfs(int v, long long pushed) {
51     if(v == sink || pushed == 0) return pushed;
52
53     for(int& i = ptr[v]; i < (int)g[v].size(); i++) {
54         Edge& e = g[v][i];
55
56         if(level[e.to] != level[v] + 1 || e.flow >= e.cap) continue;
57
58         long long flow = dfs(e.to, min(pushed, e.cap - e.flow));
59         if(flow == 0) continue;
60
61         e.flow += flow;
62         g[e.to][e.rev].flow -= flow;
63         return flow;
64     }
65     return 0;
66 }
67
68 long long max_flow() {
69     long long flow = 0;
70
71     while(bfs()) {
72         fill(ptr.begin(), ptr.end(), 0);
73         while(long long pushed = dfs(source, INF)) {
74             flow += pushed;
75         }
76     }
77     return flow;
78 }
79
80 // Get the actual flow passing through each edge
81 vector<vector<long long>> get_flow() {
82     vector<vector<long long>> flow(n, vector<long long>(n, 0));
83     for(int v = 0; v < n; v++) {
84         for(const Edge& e : g[v]) {
85             if(e.cap > 0) { // Only original edges, not residual
86                 flow[v][e.to] = e.flow;

```

```

87     }
88     }
89 }
90     return flow;
91 }
92
93 // Find minimum cut
94 vector<bool> min_cut() {
95     vector<bool> reachable(n, false);
96     queue<int> q;
97     q.push(source);
98     reachable[source] = true;
99
100     while(!q.empty()) {
101         int v = q.front();
102         q.pop();
103
104         for(const Edge& e : g[v]) {
105             if(!reachable[e.to] && e.flow < e.cap) {
106                 reachable[e.to] = true;
107                 q.push(e.to);
108             }
109         }
110     }
111     return reachable;
112 }
113 };
114
115 // Example usage:
116 /*
117 int main() {
118     // Example: 6 vertices, source = 0, sink = 5
119     int n = 6;
120     Dinic flow(n, 0, 5);
121
122     // Add edges: (from, to, capacity)
123     flow.add_edge(0, 1, 16);
124     flow.add_edge(0, 2, 13);
125     flow.add_edge(1, 2, 10);
126     flow.add_edge(1, 3, 12);
127     flow.add_edge(2, 1, 4);
128     flow.add_edge(2, 4, 14);
129     flow.add_edge(3, 2, 9);

```

```

130 flow.add_edge(3, 5, 20);
131 flow.add_edge(4, 3, 7);
132 flow.add_edge(4, 5, 4);
133
134 // Calculate maximum flow
135 long long max_flow = flow.max_flow();
136 cout << "Maximum flow: " << max_flow << "\n";
137
138 // Get minimum cut
139 vector<bool> cut = flow.min_cut();
140 cout << "Vertices on source side of min cut: ";
141 for(int i = 0; i < n; i++) {
142     if(cut[i]) cout << i << " ";
143 }
144 cout << "\n";
145
146 // Get flow through each edge
147 auto flow_matrix = flow.get_flow();
148 cout << "Flow matrix:\n";
149 for(int i = 0; i < n; i++) {
150     for(int j = 0; j < n; j++) {
151         if(flow_matrix[i][j] > 0) {
152             cout << i << " -> " << j << ": " << flow_matrix[i][j] <<
153                 "\n";
154         }
155     }
156 }
157 return 0;
158 }
159 */

```

5.5 Ford Fulkerson

```

1 #define ll long long
2 const ll INF = (1ll)4e18;
3 struct Edge {
4     int from, to;
5     ll cap, flow;
6     Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow
7         (0) {}
8 };

```

```

9 struct MaxFlow {
10     vector<Edge> edges;
11     vector<vector<int>>> adj;
12     vector<int> level, ptr;
13     int n;
14     queue<int> q;
15
16     MaxFlow(int n) : n(n) {
17         adj.resize(n);
18         level.resize(n);
19         ptr.resize(n);
20     }
21
22     void add_edge(int from, int to, ll cap) {
23         edges.emplace_back(from, to, cap);
24         edges.emplace_back(to, from, 0);
25         adj[from].push_back(edges.size() - 2);
26         adj[to].push_back(edges.size() - 1);
27     }
28
29     bool bfs(int s, int t) {
30         while(!q.empty()) q.pop();
31         fill(level.begin(), level.end(), -1);
32
33         q.push(s);
34         level[s] = 0;
35
36         while(!q.empty() && level[t] == -1) {
37             int v = q.front();
38             q.pop();
39
40             for(int id : adj[v]) {
41                 if(level[edges[id].to] == -1 && edges[id].cap - edges[id]
42                     ].flow > 0) {
43                     level[edges[id].to] = level[v] + 1;
44                     q.push(edges[id].to);
45                 }
46             }
47             return level[t] != -1;
48         }
49
50         ll dfs(int v, int t, ll pushed) {

```

```

51     if(v == t || pushed == 0)
52         return pushed;
53
54     for(; ptr[v] < (int)adj[v].size(); ptr[v]++) {
55         int id = adj[v][ptr[v]];
56         int u = edges[id].to;
57
58         if(level[u] != level[v] + 1) continue;
59
60         ll tr = dfs(u, t, min(pushed, edges[id].cap - edges[id].flow
61             ));
62         if(tr > 0) {
63             edges[id].flow += tr;
64             edges[id ^ 1].flow -= tr;
65             return tr;
66         }
67     }
68     return 0;
69 }
70
71 ll max_flow(int s, int t) {
72     ll flow = 0;
73     while(bfs(s, t)) {
74         fill(ptr.begin(), ptr.end(), 0);
75         while(ll pushed = dfs(s, t, LLONG_MAX)) {
76             flow += pushed;
77         }
78     }
79     return flow;
80 }
81
82 vector<ll> get_flows() {
83     vector<ll> flows;
84     for(int i = 0; i < edges.size(); i += 2) {
85         flows.push_back(edges[i].flow);
86     }
87     return flows;
88 }

```

6 Math

6.1 BINARY POW

```

1  #include <iostream>
2  using namespace std;
3
4  typedef long long ll;
5  ll mod = 1e9+7;
6
7  ll binary_pow(ll base, ll exp) {
8      ll result = 1;
9      base %= mod;
10     while (exp > 0) {
11         if (exp % 2 == 1) {
12             result = (result * base) % mod;
13         }
14         base = (base * base) % mod;
15         exp /= 2;
16     }
17
18     return result;
19 }

```

6.2 CATALAN

```

1  ll catalan(ll n) {
2      if (n == 0) return 1;
3      ll catalan_num = (fact[2 * n] * inv_fact[n] % MOD) * inv_fact[n + 1]
4          % MOD;
5      return catalan_num;
6  }

```

6.3 COMBINATORICS

```

1  vector<ll> fact, inv_fact;
2  void precompute_factorials(ll n, ll mod) {
3      fact.resize(n + 1);
4      inv_fact.resize(n + 1);
5      fact[0] = inv_fact[0] = 1;
6      for (ll i = 1; i <= n; i++) {
7          fact[i] = (fact[i - 1] * i) % mod;
8      }
9      inv_fact[n] = mod_inverse(fact[n], mod);

```

```

10     for (ll i = n - 1; i >= 1; i--) {
11         inv_fact[i] = (inv_fact[i + 1] * (i + 1)) % mod;
12     }
13 }
14
15 ll binomial_coefficient(ll n, ll k, ll mod) {
16     if (k > n) return 0;
17     return (fact[n] * inv_fact[k] % mod) * inv_fact[n - k] % mod;
18 }

```

6.4 EUCLIDEAN EXTENDED

```

1 ll extendedGCD(ll a, ll b, ll &x, ll &y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     ll x1, y1;
8     ll gcd = extendedGCD(b, a % b, x1, y1);
9     x = y1;
10    y = x1 - (a / b) * y1;
11    return gcd;
12 }
13
14 bool findSolutionWithConstraints(ll a, ll b, ll c, ll x_min, ll y_min,
15     ll &x, ll &y) {
16     ll g = extendedGCD(a, b, x, y);
17
18     if (c % g != 0) return false;
19
20     x *= c / g;
21     y *= c / g;
22
23     // Ajustamos las variables a/g y b/g para mover las soluciones
24     a /= g;
25     b /= g;
26
27     if (x < x_min) {
28         ll k = (x_min - x + b - 1) / b; // Redondeo hacia arriba
29         x += k * b;
30         y -= k * a;
31     } else if (x > x_min) {

```

```

31         ll k = (x - x_min) / b;
32         x -= k * b;
33         y += k * a;
34     }
35
36     if (y < y_min) {
37         ll k = (y_min - y + a - 1) / a; // Redondeo hacia arriba
38         x += k * b;
39         y -= k * a;
40     } else if (y > y_min) {
41         ll k = (y - y_min) / a;
42         x -= k * b;
43         y += k * a;
44     }
45
46     return x >= x_min && y >= y_min;
47 }

```

6.5 EULER TOTIENT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5
6 vector<ll> compute_totients(ll n) {
7     vector<ll> phi(n + 1);
8     for (ll i = 0; i <= n; i++) {
9         phi[i] = i;
10    }
11
12    for (ll i = 2; i <= n; i++) {
13        if (phi[i] == i) { // i es primo
14            for (ll j = i; j <= n; j += i) {
15                phi[j] = phi[j] * (i - 1) / i;
16            }
17        }
18    }
19
20    return phi;
21 }

```

6.6 JOSEPHUS

```

1 #include <iostream>
2 using namespace std;
3
4 typedef long long ll;
5
6 ll josephus_iterative(ll n, ll k) {
7     ll result = 0;
8     for (ll i = 2; i <= n; ++i) {
9         result = (result + k) % i;
10    }
11    return result;
12 }
13
14
15 ll josephus_recursive(ll n, ll k) {
16
17     if (n == 1)
18         return 0;
19
20     return (josephus_recursive(n - 1, k) + k) % n;
21 }
22
23
24 ll josephus_power_of_2(ll n) {
25
26     ll power = 1;
27     while (power <= n) {
28         power <<= 1;
29     }
30     power >>= 1;
31
32
33     return 2 * (n - power);
34 }

```

6.7 MOBIUS

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5
6 vector<ll> compute_mobius(ll n) {

```

```

7     vector<ll> mu(n + 1, 1);
8     vector<bool> is_prime(n + 1, true);
9
10    for (ll i = 2; i <= n; i++) {
11        if (is_prime[i]) { // i es un primo
12            for (ll j = i; j <= n; j += i) {
13                mu[j] *= -1; // Multiplicamos por -1 para cada primo
14                is_prime[j] = false;
15            }
16            for (ll j = i * i; j <= n; j += i * i) {
17                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
18                           // en 0
19            }
20        }
21    }
22    return mu;
23 }
24
25
26 ll mobius(ll x) {
27     ll count = 0;
28     for (ll i = 2; i * i <= x; i++) {
29         if (x % (i * i) == 0)
30             return 0;
31         if (x % i == 0) {
32             count++;
33             x /= i;
34         }
35     }
36
37     if (x > 1) count++;
38
39     return (count % 2 == 0) ? 1 : -1;
40 }

```

6.8 NTT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using cd = complex<double>;
4 typedef long long ll;
5 const ll mod = 998244353;

```

```

6  const ll root = 31;
7  const ll root_1 = inverse(root, mod);
8  const ll root_pw = 1 << 23;
9
10 ll inverse(ll a, ll m) {
11     ll res = 1, exp = m - 2;
12     while (exp) {
13         if (exp % 2 == 1) res = (1LL * res * a) % m;
14         a = (1LL * a * a) % m;
15         exp /= 2;
16     }
17     return res;
18 }
19
20 void ntt(vector<ll> & a, bool invert) {
21     int n = a.size();
22
23     for (int i = 1, j = 0; i < n; i++) {
24         int bit = n >> 1;
25         for (; j & bit; bit >>= 1)
26             j ^= bit;
27         j ^= bit;
28
29         if (i < j)
30             swap(a[i], a[j]);
31     }
32
33     for (int len = 2; len <= n; len <= 1) {
34         int wlen = invert ? root_1 : root;
35         for (int i = len; i < root_pw; i <= 1)
36             wlen = (int)(1LL * wlen * wlen % mod);
37
38         for (int i = 0; i < n; i += len) {
39             int w = 1;
40             for (int j = 0; j < len / 2; j++) {
41                 int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
42                 a[i+j] = u + v < mod ? u + v : u + v - mod;
43                 a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
44                 w = (int)(1LL * w * wlen % mod);
45             }
46         }
47     }
48 }

```

```

49     if (invert) {
50         int n_1 = inverse(n, mod);
51         for (auto & x : a)
52             x = (int)(1LL * x * n_1 % mod);
53     }
54 }
55
56 vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
57     vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
58     ll n = 1;
59     while (n < a.size() + b.size())
60         n <= 1;
61     fa.resize(n);
62     fb.resize(n);
63
64     ntt(fa, false);
65     ntt(fb, false);
66     for (ll i = 0; i < n; i++)
67         fa[i] = (fa[i] * fb[i]) % mod;
68     ntt(fa, true);
69
70     vector<ll> result(n);
71     for (ll i = 0; i < n; i++)
72         result[i] = fa[i];
73     return result;
74 }

```

6.9 PRIME FACTORIZATION

```

1  vector<pair<ll, ll>> prime_factors(ll n) {
2      vector<pair<ll, ll>> factors;
3      for (ll i = 2; i * i <= n; i++) {
4          if (n % i == 0) {
5              ll count = 0;
6              while (n % i == 0) {
7                  n /= i;
8                  count++;
9              }
10             factors.push_back({i, count});
11         }
12     }
13     if (n > 1) factors.push_back({n, 1});
14 }

```

```

15     return factors;
16 }
17
18
19 vector<ll> divisors(ll n) {
20     vector<ll> divs;
21     for (ll i = 1; i * i <= n; i++) {
22         if (n % i == 0) {
23             divs.push_back(i);
24             if (i != n / i) { // Evita duplicar si n es un cuadrado
25                 perfecto
26                 divs.push_back(n / i);
27             }
28         }
29     }
30     sort(divs.begin(), divs.end()); // Ordena los divisores en orden
31                                     ascendente
32     return divs;
33 }

```

6.10 SIEVE

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 typedef long long ll;
6
7 vector<ll> sieve_of_eratosthenes(ll n) {
8
9     vector<ll> primes;
10    vector<ll> primoRel(n,0);
11    for(int i = 2; i < n; i++){
12        if(!primoRel[i]){
13            primes.push_back(i);
14            for(int j = i*i; j < n ;j+=i){
15                primoRel[j] = i;
16            }
17        }
18    }
19
20    return primes;
21 }

```

6.11 fft

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using cd = complex<double>;
4 typedef long long ll;
5 const double PI = acos(-1);
6
7 void fft(vector<cd> &a, bool invert) {
8     ll n = a.size();
9     if (n == 1)
10         return;
11     vector<cd> a0(n / 2), a1(n / 2);
12     for (ll i = 0; 2 * i < n; i++) {
13         a0[i] = a[2 * i];
14         a1[i] = a[2 * i + 1];
15     }
16     fft(a0, invert);
17     fft(a1, invert);
18     double ang = 2 * PI / n * (invert ? -1 : 1);
19     cd w(1), wn(cos(ang), sin(ang));
20     for (ll i = 0; 2 * i < n; i++) {
21         a[i] = a0[i] + w * a1[i];
22         a[i + n / 2] = a0[i] - w * a1[i];
23         if (invert) {
24             a[i] /= 2;
25             a[i + n / 2] /= 2;
26         }
27         w *= wn;
28     }
29 }
30
31 vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
32     vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
33     ll n = 1;
34     while (n < a.size() + b.size())
35         n <= 1;
36     fa.resize(n);
37     fb.resize(n);
38
39     fft(fa, false);
40     fft(fb, false);
41     for (ll i = 0; i < n; i++)

```



```

42     fa[i] *= fb[i];
43     fft(fa, true);
44
45     vector<ll> result(n);
46     for (ll i = 0; i < n; i++)
47         result[i] = round(fa[i].real());
48     return result;
49 }
50
51 //Exponenciacion binommial-----
52
53
54 vector<ll> binomial_exponentiation(const vector<ll> &a, int exp) {
55     vector<ll> result = {1};
56     vector<ll> base = a;
57
58     while (exp > 0) {
59         if (exp % 2 == 1) {
60             result = multiply(result, base);
61         }
62         base = multiply(base, base);
63         exp /= 2;
64     }
65
66     while (result.size() > 1 && result.back() == 0) {
67         result.pop_back();
68     }
69
70     return result;
71 }
72
73 //FFT PRECISO -----
74
75 #define ll long long
76 using namespace std;
77 const double pi = acos(-1);
78
79 typedef long double ld;
80 typedef complex<ld> cd;
81 const ld PI = acos(-1);
82
83 void fft(vector<cd>& a, bool invert) {
84     int n = a.size();

```

```

85
86
87     for (int i = 1, j = 0; i < n; ++i) {
88         int bit = n >> 1;
89         for (; j & bit; bit >>= 1)
90             j ^= bit;
91         j ^= bit;
92         if (i < j)
93             swap(a[i], a[j]);
94     }
95
96     // Cooley-Tukey FFT
97     for (int len = 2; len <= n; len <= 1) {
98         ld ang = 2 * PI / len * (invert ? -1 : 1);
99         cd wlen(cosl(ang), sinl(ang));
100         for (int i = 0; i < n; i += len) {
101             cd w(1);
102             int len2 = len >> 1;
103             for (int j = 0; j < len2; ++j) {
104                 cd u = a[i + j];
105                 cd v = a[i + j + len2] * w;
106                 a[i + j] = u + v;
107                 a[i + j + len2] = u - v;
108                 w *= wlen;
109             }
110         }
111     }
112
113     if (invert) {
114         for (cd & x : a)
115             x /= n;
116     }
117 }
118
119
120 vector<ll> multiply(const vector<ll>& a, const vector<ll>& b) {
121     const ll BASE = 1e6;
122
123     int n = 1;
124     while(n < (int)(a.size() + b.size()))
125         n <= 1;
126
127     vector<cd> al(n), ah(n), bl(n), bh(n);

```

```

128     for (size_t i = 0; i < a.size(); ++i) {
129         al[i] = a[i] % BASE;
130         ah[i] = a[i] / BASE;
131     }
132     for (size_t i = 0; i < b.size(); ++i) {
133         bl[i] = b[i] % BASE;
134         bh[i] = b[i] / BASE;
135     }
136
137     fft(al, false);
138     fft(ah, false);
139     fft(bl, false);
140     fft(bh, false);
141
142     vector<cd> lx(n), lh(n), hl(n), hh(n);
143     for (int i = 0; i < n; ++i) {
144         lx[i] = al[i] * bl[i];
145         lh[i] = al[i] * bh[i];
146         hl[i] = ah[i] * bl[i];
147         hh[i] = ah[i] * bh[i];
148     }
149
150     fft(lx, true);
151     fft(lh, true);
152     fft(hl, true);
153     fft(hh, true);
154
155     vector<ll> result(n);
156     for (int i = 0; i < n; ++i) {
157         ll temp_ll = llround(lx[i].real());
158         ll temp_lh = llround(lh[i].real());
159         ll temp_hl = llround(hl[i].real());
160         ll temp_hh = llround(hh[i].real());
161
162         result[i] = temp_ll +
163             ((temp_lh + temp_hl) * BASE) +
164             (temp_hh * BASE * BASE);
165     }
166
167     return result;
168 }
169
170 // mejor version

```

```

171 -----
172 typedef long long ll;
173 typedef complex<double> C;
174 typedef vector<double> vd;
175 typedef vector<ll> vll;
176 const double PI = acos(-1);
177
178 void fft(vector<C>& a) {
179     int n = a.size(), L = 31 - __builtin_clz(n);
180     static vector<C> R(2, 1);
181     static vector<C> rt(2, 1);
182     for (static int k = 2; k < n; k *= 2) {
183         R.resize(n); rt.resize(n);
184         auto x = polar(1.0, PI / k);
185         for (int i = k; i < 2 * k; i++)
186             rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
187     }
188     vector<int> rev(n);
189     for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) /
190         2;
191     for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
192     for (int k = 1; k < n; k *= 2)
193         for (int i = 0; i < n; i += 2 * k) for (int j = 0; j < k; j++) {
194             auto x = (double*)&rt[j + k], y = (double*)&a[i + j + k];
195             C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
196             a[i + j + k] = a[i + j] - z;
197             a[i + j] += z;
198         }
199 }
200
201 vll multiply(const vll& a, const vll& b) {
202     if (a.empty() || b.empty()) return {};
203     vd fa(a.begin(), a.end()), fb(b.begin(), b.end());
204     int L = 32 - __builtin_clz(fa.size() + fb.size() - 1), n = 1 << L;
205     vector<C> in(n), out(n);
206
207     for (int i = 0; i < a.size(); i++) in[i] = C(fa[i], 0);
208     for (int i = 0; i < b.size(); i++) in[i].imag(fb[i]);
209
210     fft(in);
211     for (C& x : in) x *= x;

```

```

211     for (int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
        // Corregido aqui
212     fft(out);
213
214     vll res(a.size() + b.size() - 1);
215     for (int i = 0; i < res.size(); i++) {
216         res[i] = llround(imag(out[i]) / (4 * n));
217     }
218     return res;
219 }

```

7 Geometry

7.1 CONVEX HULL

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  typedef long long ll;
7  typedef pair<ll, ll> Point;
8
9  ll cross_product(Point O, Point A, Point B) {
10     return (A.first - O.first) * (B.second - O.second) - (A.second - O.
        second) * (B.first - O.first);
11 }
12
13 vector<Point> convex_hull(vector<Point>& points) {
14     sort(points.begin(), points.end());
15     points.erase(unique(points.begin(), points.end()), points.end());
16     vector<Point> hull;
17
18     // Parte inferior
19     for (const auto& p : points) {
20         while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], p) < 0)
21             hull.pop_back();
22         if (hull.empty() || hull.back() != p) {
23             hull.push_back(p);
24         }
25     }
26 }

```

```

27     // Parte superior
28     int t = hull.size() + 1;
29     for (int i = points.size() - 1; i >= 0; --i) {
30         while (hull.size() >= t && cross_product(hull[hull.size() - 2],
            hull[hull.size() - 1], points[i]) < 0)
31             hull.pop_back();
32         if (hull.empty() || hull.back() != points[i]) {
33             hull.push_back(points[i]);
34         }
35     }
36
37     hull.pop_back();
38     return hull;
39 }

```

7.2 OPERATIONS

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6
7  ll cross_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
8     ll x1 = P2.first - P1.first;
9     ll y1 = P2.second - P1.second;
10    ll x2 = P3.first - P1.first;
11    ll y2 = P3.second - P1.second;
12    return x1 * y2 - y1 * x2;
13 }
14
15
16 double distancia(pair<ll, ll> P1, pair<ll, ll> P2) {
17     return sqrt((P2.first - P1.first) * (P2.first - P1.first) +
18         (P2.second - P1.second) * (P2.second - P1.second));
19 }
20
21
22 ll dot_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
23     ll x1 = P2.first - P1.first;
24     ll y1 = P2.second - P1.second;
25     ll x2 = P3.first - P1.first;
26     ll y2 = P3.second - P1.second;

```

```

27     return x1 * x2 + y1 * y2;
28 }

```

7.3 POLYGON AREA

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  using namespace std;
5
6  typedef long long ll;
7  typedef pair<ll, ll> Point;
8
9
10 double polygon_area(const vector<Point>& polygon) {
11     ll area = 0;
12     int n = polygon.size();
13     for (int i = 0; i < n; ++i) {
14         ll j = (i + 1) % n;
15         area += (polygon[i].first * polygon[j].second - polygon[i].
16                 second * polygon[j].first);
17     }
18     return abs(area) / 2.0;
19 }

```

7.4 RAY CASTING

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  typedef long long ll;
6  typedef pair<ll, ll> Point;
7
8
9  bool is_point_in_polygon(const vector<Point>& polygon, Point p) {
10     bool inside = false;
11     int n = polygon.size();
12     for (int i = 0, j = n - 1; i < n; j = i++) {
13         if ((polygon[i].second > p.second) != (polygon[j].second > p.
14             second) &&
15             p.first < (polygon[j].first - polygon[i].first) * (p.second
16                 - polygon[i].second) /

```

```

15         (polygon[j].second - polygon[i].second) + polygon[
16             i].first) {
17             inside = !inside;
18         }
19     }
20     return inside;
21 }

```

8 Trees

8.1 Centroid

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define L(i, j, n) for (int i = (j); i < int(n); i++)
5  #define ii pair<int, int>
6  const int inf = 1e9;
7  const int N = 1e5;
8
9  vector<int> G[N];
10 int ct[N];
11 set<ii> dist[N];
12 int up[N][18];
13 int colors[N];
14 int depth[N];
15 int sz[N];
16 bool removed[N];
17 int n, root, L;
18
19 int getSize(int u, int p){
20     int szi = 1;
21     for(int v: G[u]){
22         if (p == v || removed[v]) continue;
23         szi += getSize(v, u);
24     }
25     return sz[u] = szi;
26 }
27
28 int centroid(int u, int tree_size, int p){
29     for (int v: G[u]){
30         if (v == p || removed[v]) continue;
31         if (sz[v] * 2 > tree_size) return centroid(v, tree_size, u);

```

```

32     }
33     return u;
34 }
35
36 void build(int node, int tree_size, int p)
37 {
38     getSize(node, - 1);
39     int cen = centroid(node, tree_size, -1);
40     removed[cen] = 1;
41     ct[cen] = p;
42     if (p == -1) root = cen;
43
44     if (tree_size == 1) return;
45
46     for (int v: G[cen]){
47         if (removed[v]) continue;
48         build(v, sz[v], cen);
49     }
50 }
51
52 void update(int v){
53     int u = v;
54     while(v != -1){
55         dist[v].insert(distance(u, v), v);
56         v = par[v];
57     }
58     return res;
59 }
60
61 int query(int v){
62     int u = v;
63     int res = INT_MAX;
64     while(v != -1){
65         res = min(res, distance(u, v), dist[v].begin()->first); //
66             Minimum
67         v = par[v];
68     }
69     return res;
70 }

```

8.2 LCA

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define L(i, j, n) for (int i = (j); i < int(n); i ++)
5  #define ii pair<int, int>
6  const int inf = 1e9;
7  const int N = 1e5;
8
9  vector<int> G[N], ct[N];
10 set<ii> dist[N];
11 int up[N][18];
12 int colors[N];
13 int depth[N];
14 int sz[N];
15 bool removed[N];
16 int n, root, L;
17
18 int getSize(int u, int p){
19     int szi = 1;
20     for(int v: G[u]){
21         if (p == v || removed[v]) continue;
22         szi += getSize(v, u);
23     }
24     return sz[u] = szi;
25 }
26
27 int centroid(int u, int tree_size, int p){
28     for (int v: G[u]){
29         if (v == p || removed[v]) continue;
30         if (sz[v] * 2 > tree_size) return centroid(v, tree_size, u);
31     }
32     return u;
33 }
34
35 void build(int node, int tree_size, int p)
36 {
37     getSize(node, - 1);
38     int cen = centroid(node, tree_size, -1);
39     removed[cen] = 1;
40     if (p != -1){
41         ct[cen].push_back(p);
42     } else root = cen;
43 }

```

```

44     if (tree_size == 1) return;
45
46     for (int v: G[cen]){
47         if (removed[v]) continue;
48         build(v, sz[v], cen);
49     }
50
51 }
52
53 void dfs(int u, int p){
54     up[u][0] = p;
55     for (int i = 1; i <= L; i++){
56         if (up[u][i-1] != -1) up[u][i] = up[up[u][i-1]][i-1];
57         else up[u][i] = -1;
58     }
59     for (int v: G[u]){
60         if (v == p) continue;
61         depth[v] = depth[u] + 1;
62         dfs(v, u);
63     }
64 }
65
66 int LCA(int u, int v){
67     if (depth[u] < depth[v]) swap(u, v);
68     for (int i = L; i >= 0; i--){
69         if (up[u][i] != -1 && depth[up[u][i]] >= depth[v]){
70             u = up[u][i];
71         }
72     }
73     if (u == v) return u;
74     for (int i = L; i >= 0; i--){
75         if (up[u][i] != up[v][i] && up[u][i] != -1 && up[v][i] != -1){
76             u = up[u][i];
77             v = up[v][i];
78         }
79     }
80     return up[u][0];
81 }
82
83 int dis(int u, int v){
84     int cmm = LCA(u, v);
85     // cout << u << " " << v << " " << cmm << "\n";
86     return depth[u] + depth[v] - (2 * depth[cmm]);

```

```

87 }
88
89 void uup(int u, int node){
90     dist[u].insert({dis(u, node), node});
91     for (int v: ct[u])
92         uup(v, node);
93 }
94
95 void update(int node){
96     dist[node].insert({0, node});
97     for (int v: ct[node])
98         uup(v, node);
99 }
100
101 int qup(int u, int node){
102     int mn = dis(node, u) + dist[u].begin()->first;
103     for (int v: ct[u]) mn = min(mn, qup(v, node));
104     return mn;
105 }
106
107 int query(int node){
108     int mn = dist[node].begin()->first;
109     for (int v: ct[node]) mn = min(mn, qup(v, node));
110     return mn;
111 }
112
113 int main()
114 {
115     ios::sync_with_stdio(0);cin.tie(0);
116     int m; cin >> n >> m;
117     L = log2(n);
118     L(i, 1, n){
119         int u, v; cin >> u >> v;
120         u--; v--;
121         G[u].push_back(v);
122         G[v].push_back(u);
123     }
124     L(i, 0, n){
125         dist[i].insert({inf, i});
126     }
127     build(0, n, -1);
128     L(i, 0, L + 1) up[root][i] = -1;
129     run(root, -1);

```

```
130     update(0);
131     L(_q, 0, m){
132         int op, node; cin >> op >> node;
133         if (op == 2){
134             cout << query(node-1) << '\n';
135         } else {
136             update(node-1); // Log Log
137         }
138     }
139 }
```