

Contents

1 Setting	
1.1 Header	1
2 String	
2.1 KMP	1
2.2 Aho Chorasick	1
2.3 Suffix array	2
2.4 Manacher's algorithm	3
2.5 Z algorithm	3
3 Graph & Flow	
3.1 Dinic	3
3.2 Bipartite matching (simple)	4
3.3 MCMF	4
3.4 Articulation Point	5
3.5 2SAT & answer recover	5
3.6 SCC	6
4 Query	
4.1 HLD	6
4.2 Centroid decomposition	7
4.3 Mo's algorithm	8
4.4 Mo's algorithm on tree	8
4.5 Parallel binary search	10
5 Geometry	
5.1 Closest pair	10
5.2 Convex hull	11
5.3 Rotating Calipers	12
6 Miscellaneous	
6.1 Grundy number	12
6.2 Hungarian	12
6.3 Convex Hull trick	13

1 Setting

1.1 Header

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <iostream>
#include <memory.h>
#include <math.h>
#include <assert.h>
#include <queue>
#include <map>
#include <set>
#include <string>
#include <algorithm>
#include <functional>
#include <vector>
#include <stack>

using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> Pi;
typedef pair<ll, ll> Pll;

#define Fi first
#define Se second
#define pb(x) push_back(x)
#define sz(x) (int)x.size()
#define rep(i, n) for(int i=0; i<n; i++)
#define repp(i, n) for(int i=1; i<=n; i++)
#define all(x) x.begin(), x.end()

#define INF 987654321
#define IINF 987654321987654321
```

2 String

2.1 KMP

```
vector<int> preprocess(string p){
    int m = p.size();
    vector<int> fail(m);
    fail[0] = 0; int j = 0;
    for(int i=1; i<m; i++){
        while(j>0&&p[i]!=p[j]) j = fail[j-1];
        if( p[i] == p[j] ){
            fail[i] = j+1; j++;
        }else{
            fail[i] = 0;
        }
    }
    return fail;
}

vector<int> kmp(string s, string p){
    auto fail = preprocess(p);
    vector<int> ans; int n = s.size(), m = p.size();
    int j = 0;
```

```

for(int i=0;i<n;i++){
    while(j>0 && s[i]!=p[j]) j = fail[j-1];
    if( s[i] == p[j] ){
        if( j == m-1 ){
            ans.pb(i-m+1); j = fail[j];
        }else{
            j++;
        }
    }
}
return ans;
}

```

2.2 Aho Chorasick

```

struct AhoCorasick{
    struct Node{
        int fail;
        vector<int> output;
        int children[26];

        Node(){
            for(int i=0;i<26;i++) children[i] = -1;
            fail = -1;
        }
    };

    vector<Node> trie;
    int new_node(){
        Node x;
        trie.push_back(x);
        return (int)trie.size()-1;
    }

    void add(int node, string &s, int idx, int string_num){
        //cout << node << " " << idx << endl;
        if( idx == s.size() ){
            trie[node].output.push_back(string_num);
            return;
        }
        int c = s[idx] - 'a';
        if( trie[node].children[c] == -1 ){
            int next = new_node();
            trie[node].children[c] = next;
        }

        add(trie[node].children[c], s, idx+1, string_num);
    }

    void build(vector<string> v){
        int root = new_node();
        for(int i=0;i<v.size();i++){
            add(root,v[i],0,i);
        }
    }
}

```

```

queue<int> q;
q.push(root); trie[root].fail = root;
while( !q.empty() ){
    int cur = q.front(); q.pop();
    for(int i=0;i<26;i++){
        int next = trie[cur].children[i];
        if( next == -1 ) continue;

        // build fail
        if( cur == root ){
            trie[next].fail = root;
        }
        else{
            int x = trie[cur].fail;
            while( x != root && trie[x].children[i] == -1 ) x = trie[x].fail;
            if( trie[x].children[i] != -1 ) x = trie[x].children[i];
            trie[next].fail = x;
        }
        // build output
        int f = trie[next].fail;
        for(auto e : trie[f].output) trie[next].output.push_back(e);
        q.push(next);
    }
}

vector<Pi> find(string s){
    int n = (int) s.size();
    int cur = 0, root = 0;
    vector<Pi> ans;
    for(int i=0;i<n;i++){
        int c = s[i] - 'a';
        while( cur != root && trie[cur].children[c] == -1 ) cur = trie[cur].fail;
        if( trie[cur].children[c] != -1 ) cur = trie[cur].children[c];

        for(auto e : trie[cur].output){
            ans.push_back({e,i});
        }
    }
    return ans;
}
};

```

2.3 Suffix array

```

// Make sure to add !, #, $, %, & at the end of input string
class SuffixArray{
public:
    int n;
    string s;
    vector<int> rank, temprank, sa, tempsa, c;
    vector<int> lcp;
}

```

```

SuffixArray(string _s){
    n = _s.size(); s = _s;
    rank.resize(n); temprank.resize(n); sa.resize(n); tempsa.resize(n);
    lcp.resize(n);
    constructSA();
    constructLCP();
}

void countingSort(int k){
    int sum = 0, maxi = max(270, n); //ASCII 256
    c.clear(); c.resize(maxi+10);
    for(auto& e : c) e = 0;
    for(int i=0; i<n; i++) c[ i+k<n ? rank[i+k] : 0 ] ++;
    for(int i=0; i<maxi; i++){
        int t = c[i]; c[i] = sum; sum += t;
    }
    for(int i=0; i<n; i++) tempsa[ c[ sa[i]+k < n ? rank[sa[i]+k] : 0 ] ++
    ] = sa[i];
    for(int i=0; i<n; i++) sa[i] = tempsa[i];
}

void constructSA(){
    for(int i=0; i<n; i++) rank[i] = s[i];
    for(int i=0; i<n; i++) sa[i] = i;
    for(int k=1; k<n; k<=1){
        countingSort(k);
        countingSort(0);
        int r = 0;
        temprank[sa[0]] = 0;
        for(int i=1; i<n; i++){
            temprank[sa[i]] = (rank[sa[i]] == rank[sa[i-1]] && rank[sa[i]+
            k] == rank[sa[i-1]+k] ) ? r : ++r;
        }
        for(int i=0; i<n; i++) rank[i] = temprank[i];
        if( rank[sa[n-1]] == n-1 ) break;
    }
}

// lcp Implementation from
// http://m.blog.naver.com/dark__nebula/220419358547
void constructLCP(){
    int h = 0;
    for(int i=0; i<n; i++){
        if( rank[i] ){
            int j = sa[rank[i]-1];
            while( s[i+h] == s[j+h] ) h++;
            lcp[rank[i]] = h;
        }
        if( h > 0 ) h--;
    }
}

};

```

2.4 Manacher's algorithm

```

// finds radius of longest palindrome centered at s[i]
// If you also want to find even-length paidnromes, use dummy characters
// baab -> #b#a#a#b#
vector<int> ManacherAlgorithm(string s){
    int n = (int) s.size();
    int p = -1, r = -1;
    vector<int> A(n);
    for(int i=0; i<n; i++){
        if( r < i ){
            A[i] = 0;
            int j = 0;
            while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ]
            ) A[i]++;
            A[i]--;
        }
        else{
            A[i] = min( A[2*p - i] , r-i );
            while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ]
            ) A[i]++;
            A[i]--;
        }

        // update r
        if( r < i + A[i] ){
            r = i + A[i];
            p = i;
        }
    }
    return A;
}

```

2.5 Z algorithm

```

// Calculates LCP[i] for all 0 <= i < n
vector<int> Zalgorithm(string s){
    int l=0, r=0;
    int n = (int) s.size();
    vector<int> Z(n);
    Z[0] = n;
    for(int i=1; i<n; i++){
        // reset and calculate again
        if( i > r ){
            l = r = i;
            while( r<n && s[r] == s[r-l] ) r++;
            r--;
            Z[i] = r-l+1;
        }

        // extend [l,r]
        else{
            int k = i-l;
            // not enough matching at position k

```

```

        if( Z[k] < r-i+1 ) Z[i] = Z[k];
        // enough matching. extend [l,r]
        else{
            l = i;
            while( r<n && s[r] == s[r-l] ) r++;
            r--;
            Z[i] = r-l+1;
        }
    }
}
return Z;
};

```

3 Graph & Flow

3.1 Dinic

```

struct MaxFlowDinic{
    struct Edge{
        // next, inv, residual
        int to, inv; ll res;
    };

    int n;
    vector<vector<Edge>> graph;

    vector<int> lev,work;

    void init(int x){
        n = x+10;
        graph.resize(x+10);
        lev.resize(n); work.resize(n);
    }

    void make_edge(int s, int e, ll cap, ll caprev = 0){
        Edge forward = {e, (int)graph[e].size(), cap};
        Edge backward = {s, (int)graph[s].size(), caprev};
        graph[s].push_back(forward);
        graph[e].push_back(backward);
    }

    bool bfs(int source, int sink){
        queue<int> q;
        for(auto& e : lev) e = -1;
        lev[source] = 0; q.push(source);
        while(!q.empty()){
            int cur = q.front(); q.pop();
            for(auto e : graph[cur]){
                if(lev[e.to]==-1 && e.res > 0){
                    lev[e.to] = lev[cur]+1;
                    q.push(e.to);
                }
            }
        }
    }

```

```

    }
    return lev[sink] != -1;
}

ll dfs(int cur, int sink, ll flow){
    if( cur == sink ) return flow;
    for(int &i = work[cur]; i < (int)graph[cur].size(); i++){
        Edge &e = graph[cur][i];
        if( e.res == 0 || lev[e.to] != lev[cur]+1 ) continue;
        ll df = dfs(e.to, sink, min(flow, e.res) );
        if( df > 0 ){
            e.res -= df;
            graph[e.to][e.inv].res += df;
            return df;
        }
    }
    return 0;
}

ll solve( int source, int sink ){
    ll ans = 0;
    while( bfs(source, sink) ){
        for(auto& e : work) e = 0;
        while( true ){
            ll flow = dfs(source,sink,54321987654321LL);
            if( flow == 0 ) break;
            ans += flow;
        }
    }
    return ans;
}
};

```

3.2 Bipartite matching (simple)

```

int yx[5000], xy[5000];
bool vis[5000];
vector<int> E[5000];
int dfs(int x){
    vis[x] = true;
    for(auto e : E[x]){
        if( yx[e] == -1 || (vis[yx[e]] == false && dfs(yx[e])) ){
            yx[e] = x;
            xy[e] = e;
            return 1;
        }
    }
    return 0;
}

int main(){
    memset(yx,-1,sizeof yx);

```

```

int ans = 0;
rep(i,N){
    memset(vis,0,sizeof vis);
    ans += dfs(i);
}
cout << ans;
}

```

3.3 MCMF

```

struct MCMF{
    struct edge{
        int to, inv, cap, flow, cost;
        int res(){
            return cap - flow;
        }
    };

    vector<vector<edge>> graph;
    vector<int> pv, pe;
    vector<int> dist, inq;

    void init(int x){
        graph.resize(x+10);
        for(auto& e : graph) e.resize(x+10);
        pv.resize(x+10); pe.resize(x+10);
        dist.resize(x+10);
        inq.resize(x+10);
    }

    void make_edge(int from, int to, int cap, int cost){
        //printf("%d -> %d | cost = %d\n",from,to,cost);
        edge forward = {to, (int)graph[to].size(), cap, 0, cost};
        edge backward = {from, (int)graph[from].size(), 0, 0, -cost};
        graph[from].push_back(forward);
        graph[to].push_back(backward);
    }

    int solve(int source, int sink){
        int ans = 0;
        int totalflow = 0;
        while(true){
            for(auto& e : dist) e = INF;
            for(auto& e : inq) e = 0;
            queue<int> q;
            q.push(source); inq[source] = 1;
            dist[source] = 0;

            while(!q.empty()){
                int cur = q.front(); q.pop();
                inq[cur] = 0;
                for(int i=0;i<(int)graph[cur].size();i++){
                    auto& e = graph[cur][i];
                    if( e.res() > 0 && dist[e.to] > dist[cur] + e.cost ){

```

```

                        dist[e.to] = dist[cur] + e.cost;
                        pv[e.to] = cur; pe[e.to] = i;
                        if( inq[e.to] == 0 ){
                            q.push(e.to); inq[e.to] = 1;
                        }
                    }
                }
            }

            if( dist[sink] == INF ) break;

            // add this limit when we don't require maxflow
            //if( dist[sink] > 0 ) break;

            int mnflow = INF;
            for( int v = sink; v != source; v = pv[v] ){
                mnflow = min( mnflow, graph[pv[v]][pe[v]].res() );
            }

            for( int v = sink; v != source; v = pv[v] ){
                int tmp = graph[pv[v]][pe[v]].inv;
                graph[pv[v]][pe[v]].flow += mnflow;
                graph[v][tmp].flow -= mnflow;
            }
            totalflow += mnflow;
            ans += dist[sink] * mnflow;
        }
        return ans;
    }
};

```

3.4 Articulation Point

3.5 2SAT & answer recover

```

#define MAX_V 20010
int V,M;

vector<int> Edge[MAX_V];
vector<int> rEdge[MAX_V];
vector<int> vs;

bool vis[MAX_V];
int cmp[MAX_V];
set<int> printSet[MAX_V];

void addEdge(int from, int to){
    Edge[from].push_back(to);
    rEdge[to].push_back(from);
}

void dfs(int v){

```

```

    vis[v] = true;
    for (int i = 0; i < Edge[v].size(); i++){
        if (!vis[Edge[v][i]]) dfs(Edge[v][i]);
    }
    vs.push_back(v);
}

void rdfs(int v, int k){
    vis[v] = true;
    cmp[v] = k;
    printSet[k].insert(v);
    for (int i = 0; i < rEdge[v].size(); i++){
        if (!vis[rEdge[v][i]]) rdfs(rEdge[v][i], k);
    }
}

bool cmp1(set<int>& a, set<int>& b) {
    return *a.begin() < *b.begin();
}

int main()
{
    //freopen("in.txt", "r", stdin);
    geti(V); geti(M);
    int cnt = 0;
    while (M--){
        int a, b;
        scanf("%d%d", &a, &b);
        if (a > 0 && b > 0 ){
            addEdge(a + V, b);
            addEdge(b + V, a);
        }
        else if (a > 0 && b < 0){
            b = -b;
            addEdge(a + V, b + V);
            addEdge(b, a);
        }
        else if (a < 0 && b > 0){
            a = -a;
            addEdge(a, b);
            addEdge(b + V, a + V);
        }
        else{
            a = -a; b = -b;
            addEdge(a, b + V);
            addEdge(b, a + V);
        }
    }
}

```

```

memset(vis, false, sizeof(vis));
for (int i = 1; i <= 2*V; i++){
    if (!vis[i]) dfs(i);
}

memset(vis, false, sizeof(vis));
int k = 0;
for (int i = vs.size()-1; i >= 0 ; i--){
    if (!vis[vs[i]]) rdfs(vs[i],k++);
}

for (int i = 1; i <= V; i++){
    if (cmp[i] == cmp[V + i]){
        printf("0\n");
        return 0;
    }
}
printf("1\n");

for (int i = 1; i <= V; i++){
    if (cmp[i] > cmp[V + i]){
        printf("1 ");
    }
    else printf("0 ");
}
}

```

3.6 SCC

4 Query

4.1 HLD

```

// 1-index
#define L(x) ((x)<<1)
#define R(x) (((x)<<1)+1)

const int MAXN = 100050;
const int LOGN = 17;

vector<int> adj[MAXN];
int st[6 * MAXN], sub[MAXN], pa[MAXN];
int idx[MAXN], head[MAXN], pos[MAXN], rev[MAXN];
int sz, cnt;

void init(int n) {
    fill(st, st + 6*n, INF);
    fill(head, head + n, -1);
}

void dfs(int x, int p) {

```

```

sub[x] = 1;
for(auto c : adj[x]) {
    if(c != p) {
        pa[c] = x;
        dfs(c, x);
        sub[x] += sub[c];
    }
}
}

void update(int x, int id = 1, int l = 0, int r = sz) {
    if(x < l || x >= r) return;
    if(r - l <= 1) {
        if(st[id] == INF)
            st[id] = l;
        else
            st[id] = INF;
        return;
    }
    int mid = (l + r) >> 1;
    update(x, L(id), l, mid);
    update(x, R(id), mid, r);
    st[id] = min(st[L(id)], st[R(id)]);
}

int query(int x, int y, int id = 1, int l = 0, int r = sz) {
    if(y <= l || r <= x) return INF;
    if(x <= l && r <= y) return st[id];
    int mid = (l + r) >> 1;
    return min(query(x, y, L(id), l, mid), query(x, y, R(id), mid, r));
}

void HLD(int x, int p) {
    if(head[cnt] == -1)
        head[cnt] = x;
    idx[x] = cnt;
    pos[x] = sz;
    rev[sz] = x;
    sz++;

    int cindex = -1;
    for(int i = 0; i < adj[x].size(); i++) {
        if(adj[x][i] != p)
            if(cindex == -1 || sub[adj[x][cindex]] < sub[adj[x][i]])
                cindex = i;
    }
    if(cindex != -1)
        HLD(adj[x][cindex], x);
    for(int i = 0; i < adj[x].size(); i++) {
        if(adj[x][i] != p && i != cindex) {
            cnt++;
            HLD(adj[x][i], x);
        }
    }
}

```

```

}

int queryTree(int v) {
    if(v == 0) {
        int ans = query(pos[0], pos[0] + 1);
        if(ans == INF)
            return -1;
        else
            return 1;
    }
    int vchain, ans = INF;
    while(1) {
        vchain = idx[v];
        if(idx[v] == 0) {
            ans = min(ans, query(pos[0], pos[v]+1));
            break;
        }
        ans = min(ans, query(pos[head[vchain]], pos[v]+1));
        v = pa[head[vchain]];
    }
    if(ans == INF)
        return -1;
    else
        return rev[ans] + 1;
}

void updateTree(int v) {
    update(pos[v]);
}

int main() {
    int n, q;
    geti(n, q);
    for(int i = 1; i < n; i++) {
        int u, v;
        geti(u, v);
        u--; v--;
        adj[u].pb(v);
        adj[v].pb(u);
    }

    init(n);
    dfs(0, -1);
    HLD(0, -1);

    while(q--) {
        int type, x;
        geti(type, x);
        x--;
        if(type == 0) {
            updateTree(x);
        } else {
            printf("%d\n", queryTree(x));
        }
    }
}

```

```
}
```

4.2 Centroid decomposition

```
int n;
set<int> adj[MAXN];
int sub[MAXN], dep[MAXN];

void dfsSubtree(int node, int pnode) {
    sub[node] = 1;
    for(auto cnode : adj[node]) {
        if(cnode != pnode) {
            dfsSubtree(cnode, node);
            sub[node] += sub[cnode];
        }
    }
}

int findCentroid(int node, int pnode, int size) {
    for(auto cnode : adj[node]) {
        if(cnode != pnode && sub[cnode] > size / 2)
            return findCentroid(cnode, node, size);
    }
    return node;
}

bool decompose(int node, int depth) {
    bool result = true;
    if(depth >= 26) {
        return false;
    }
    dfsSubtree(node, -1);
    int ctr = findCentroid(node, -1, sub[node]);
    dep[ctr] = depth;
    for(auto cnode : adj[ctr]) {
        adj[cnode].erase(ctr);
        result &= decompose(cnode, depth + 1);
    }
    adj[ctr].clear();
    return result;
}

int main() {
    geti(n);
    rep(i, n-1) {
        int u, v;
        geti(u, v);
        adj[u].insert(v);
        adj[v].insert(u);
    }
    if(decompose(1, 0)) {
        repp(i, n) printf("%c ", dep[i] + 'A');
    } else {
        cout << "Impossible!";
    }
}
```

```
}
```

4.3 Mo's algorithm

```
int N,M,K,tc;
ll c[1000005];
ll p[1000005]; int Bsize;
typedef struct query{
    int l,r,n; ll ans;
} query;
bool cmp(query& a, query& b){
    if( a.l/Bsize == b.l/Bsize ) return a.r < b.r;
    else return a.l/Bsize < b.l/Bsize;
}
bool cmp2(query&a, query& b ){ return a.n < b.n; }
int main(void)
{
    geti(N,M); rep(i,N) scanf("%lld",p+i);
    Bsize = (int) sqrt(1.0*N);
    vector<query> q;
    rep(i,M){
        int a,b; geti(a,b); a--;b--;
        q.push_back({a,b,i});
    }

    sort(all(q),cmp);
    int l=0, r=-1; ll sum = 0;

    for(int i=0;i<q.size();i++){
        query& e = q[i];
        int ql = e.l, qr = e.r;
        while( r < qr ){
            r++;
            sum += p[r]*(2*c[p[r]]+1); c[p[r]]++;
        }
        while( r > qr ){
            sum += p[r]*(1-2*c[p[r]]); c[p[r]]--;
            r--;
        }
        while( l < ql ){
            sum += p[l]*(1-2*c[p[l]]); c[p[l]]--;
            l++;
        }
        while( l > ql ){
            l--;
            sum += p[l]*(2*c[p[l]]+1); c[p[l]]++;
        }
        e.ans = sum;
    }

    sort(all(q),cmp2);
    for(auto e : q ){
        printf("%lld\n",e.ans);
    }
}
```


4.4 Mo's algorithm on tree

```
int N;
int g[MAXN];
int f[MAXN];
int pa[LOGV][MAXV]; int level[MAXN];
int ST[MAXN], EN[MAXN], arr[MAXN*3];
int tt = 0;
vector<int> E[MAXN];

void dfs_build(int x, int p, int lev){
    pa[0][x] = p;
    level[x] = lev;
    ST[x] = ++tt; arr[tt] = x;
    for(auto e : E[x]) if(e!=p){
        dfs_build(e,x,lev+1);
    }
    EN[x] = ++tt; arr[tt] = x;
}

void lca_build(){
    for(int k=1;k<LOGV;k++){
        repp(i,N){
            if( pa[k-1][i] != -1 ) pa[k][i] = pa[k-1][pa[k-1][i]];
            else pa[k][i] = -1;
        }
    }
}

int lca(int x, int y){
    if( level[x] < level[y] ) swap(x,y);
    int diff = level[x] - level[y];
    for(int k=0;k<LOGV;k++){
        if( diff & (1<<k) ) x = pa[k][x];
    }
    if( x == y ) return x;
    for(int k=LOGV-1;k>=0;k--){
        if( pa[k][x] != pa[k][y] ){
            x = pa[k][x]; y = pa[k][y];
        }
    }
    return pa[0][x];
}

int Bsize;
struct query{
    int l,r,n;
};
bool cmp1(query& a, query& b){
    if( a.l/Bsize == b.l/Bsize ) return a.r < b.r;
    else return a.l/Bsize < b.l/Bsize;
};
bool cmp2(query&a, query& b ){ return a.n < b.n; }

ll ans[100500];
ll cnt[2][200500];
```

```
int vis[100500];
ll sum = 0;

void update(int x, int type){
    // add node to range
    if( type == 1 ){
        sum += cnt[g[x]^1][f[x]];
        cnt[g[x]][f[x]]++;
    }
    // remove node from range
    if( type == 0 ){
        sum -= cnt[g[x]^1][f[x]];
        cnt[g[x]][f[x]]--;
    }
}

int main(void){
    geti(N);
    repp(i,N) geti(g[i]);
    repp(i,N) geti(f[i]);

    set<int> flist;
    map<int,int> fmp;
    repp(i,N) flist.insert(f[i]);
    int tmp = 1;
    for(auto e: flist) fmp[e] = tmp++;
    repp(i,N) f[i] = fmp[f[i]];

    repp(i,N-1){
        int a,b; geti(a,b);
        E[a].pb(b); E[b].pb(a);
    }
    tt = 0;
    dfs_build(1,-1,0);
    lca_build();
    Bsize = (int) sqrt(1.0*tt);

    int Q; geti(Q);
    vector<query> v;
    repp(q,Q){
        int a,b; geti(a,b);
        if( ST[a] > ST[b] ) swap(a,b);
        int l = lca(a,b);
        if( a == l || b == l ){
            v.push_back({ST[a],ST[b],q});
        }
        else{
            v.push_back({EN[a],ST[b],q});
        }
    }

    sort(all(v),cmp1);
    int l=1, r=0;
    for(int i=0;i<v.size();i++){
        query& e = v[i];
```

```

int ql = e.l, qr = e.r;
while( r < qr ){
    r++;
    int node = arr[r];
    vis[node]++;
    if( vis[node] == 1 ) update(node,1);
    if( vis[node] == 2 ) update(node,0);
}
while( r > qr ){
    int node = arr[r];
    vis[node]--;
    if( vis[node] == 0 ) update(node,0);
    if( vis[node] == 1 ) update(node,1);
    r--;
}
while( l < ql ){
    int node = arr[l];
    vis[node]--;
    if( vis[node] == 0 ) update(node,0);
    if( vis[node] == 1 ) update(node,1);
    l++;
}
while( l > ql ){
    l--;
    int node = arr[l];
    vis[node]++;
    if( vis[node] == 1 ) update(node,1);
    if( vis[node] == 2 ) update(node,0);
}

int u = arr[ql]; int v = arr[qr];
int l = lca(u,v);

if( u != l && v != l ){
    int node = l;
    vis[node]++;
    if( vis[node] == 1 ) update(node,1);
    if( vis[node] == 2 ) update(node,0);
}

ans[e.n] += sum;

if( u != l && v != l ){
    int node = l;
    vis[node]--;
    if( vis[node] == 0 ) update(node,0);
    if( vis[node] == 1 ) update(node,1);
}

}
repp(i,Q) printf("%lld\n",ans[i]);
}

```

4.5 Parallel binary search

```

int N,M,K,Q;

vector<Pi> edge[1000500];
int pa[MAXN]; int sz[MAXN];

// each query's answer
Pi ans[MAXN];
// each query's possible answer range for binary search
int low[MAXN], high[MAXN];
// focus[x] : list of query # where it's mid value is x
vector<int> focus[1000500];

int find(int x){
    if( x == pa[x] ) return x;
    return pa[x] = find(pa[x]);
}

int x[MAXN], y[MAXN];

void uni(int a, int b){
    a = find(a); b = find(b);
    if( a == b ) return;
    pa[a] = b;
    sz[b] += sz[a];
}

int main(void){
    //ios::sync_with_stdio(false);
    geti(N,M);
    int C = -1;
    repp(i,M){
        int a,b,c; geti(a,b,c);
        edge[c].push_back({a,b});
        C = max(C, c);
    }

    geti(Q);
    repp(i,Q){
        int a,b;
        geti(a,b); x[i] = a; y[i] = b;
        ans[i] = {INF,-1};
        // Initially, every query has answer in [0,C] range
        low[i] = 0; high[i] = C;
    }

    bool changed = true;
    while( changed ){
        changed = false;

        // Clear variables
        rep(i,C+1) focus[i].clear();
        repp(i,N) pa[i] = i, sz[i] = 1;

        // Put each query into corresponding focus group
        repp(i,Q){
            if( low[i] > high[i] ) continue;

```

```

        focus[ (low[i] + high[i])/2 ].push_back(i);
    }

    // for every time 0~C
    for(int k=0;k<=C;k++){
        // perform action of that time
        for(auto e : edge[k]) uni(e.Fi,e.Se);

        // for each focus group
        // determine it's answer & next position
        for(auto e : focus[k]){
            changed = true;
            int a = x[e]; int b = y[e];
            if( find(a) == find(b) ){
                ans[e].Fi = min(ans[e].Fi, k);
                ans[e].Se = sz[find(a)];
                high[e] = k-1;
            }
            else{
                low[e] = k+1;
            }
        }
    }

    repp(i,Q){
        if( ans[i].Fi == INF ) printf("%d\n",-1);
        else printf("%d %d\n",ans[i].Fi, ans[i].Se);
    }
}

```

5 Geometry

5.1 Closest pair

```

int N,M,T,K,V;

typedef struct Point{
    int x,y;
    bool operator<(const Point& l) const{
        if( y == l.y ) return x < l.x;
        return y < l.y;
    }
    bool operator==(const Point& l) const{
        return (x==l.x)&&(y==l.y);
    }
} Point;

bool cmp(const Point& l, const Point& r){
    if(l.x == r.x ) return l.y < r.y;
    return l.x < r.x;
}

```

```

int dist(Point& l, Point& r ){
    return (l.x-r.x)*(l.x-r.x) + (l.y-r.y)*(l.y-r.y);
}

int main(void)
{
    geti(N); vector<Point> v(N);
    for(int i=0;i<N;i++){
        int x ,y; geti(x,y); v[i].x = x; v[i].y = y;
    }
    sort(all(v),cmp);
    int ans = dist(v[0],v[1]); int left = 0;
    set<Point> possible; possible.insert(v[0]); possible.insert(v[1]);

    for(int i=2;i<N;i++){
        while( (v[i].x - v[left].x)*(v[i].x - v[left].x ) > ans ){
            possible.erase(v[left]);
            left++;
        }
        int d = (int) sqrt(ans) + 1;
        auto bottom = possible.lower_bound({-100000,v[i].y-d});
        auto top = possible.upper_bound({100000,v[i].y+d});
        for(auto it = bottom; it != top; it++){
            Point cur = *it;
            if( dist(v[i],cur) < ans ) ans = dist(v[i],cur);
        }
        possible.insert(v[i]);
    }
    cout << ans;
}

```

5.2 Convex hull

```

int N,M,T,K,V;

ll ccw(ll x1 ,ll y1, ll x2, ll y2, ll x3, ll y3){
    ll ax = x2-x1; ll ay = y2-y1;
    ll bx = x3-x1; ll by = y3-y1;
    return (ax*by)-(bx*ay);
}

typedef struct Point{
    ll x,y,n;
} Point;

ll ccw(Point& a, Point& b, Point& c){
    return ccw(a.x,a.y,b.x,b.y,c.x,c.y);
}

bool cmp(const Point& a, const Point& b ){
    if( a.y == b.y ) return a.x > b.x;
    return a.y < b.y;
}

```

```

Point p[100500];

bool cmp_angle(const Point& a,const Point& b){
    if( ccw(0,0,a.x,a.y,b.x,b.y) == 0 ){
        return a.x*a.x+a.y*a.y < b.x*b.x+b.y*b.y;
    }
    return ccw(0,0,a.x,a.y,b.x,b.y) > 0;
}

int x[100500], y[100500];

int main(void)
{
    geti(N);
    repp(i,N) p[i].n = i;
    repp(i,N) scanf("%lld%lld",&p[i].x,&p[i].y);
    sort(p+1,p+N+1,cmp);
    for(int i=2;i<=N;i++){
        p[i].x -= p[1].x; p[i].y -= p[1].y;
    }

    p[1].x = p[1].y = 0;
    sort(p+1,p+N+1,cmp_angle);

    stack<Point> stk; stk.push(p[1]);
    for(int i=2;i<=N;i++){
        if( stk.size() <= 1 ){
            stk.push(p[i]); continue;
        }
        Point p3 = p[i];
        Point p2 = stk.top(); stk.pop();
        Point p1 = stk.top(); stk.pop();

        if( ccw(p1,p2,p3) > 0 ){
            stk.push(p1); stk.push(p2); stk.push(p3); continue;
        }else if( ccw(p1,p2,p3) <= 0 ){
            //p2 제거
            stk.push(p1); i--; continue;
        }
    }

    printf("%d",(int)stk.size());
}

```

5.3 Rotating Calipers

// TBD

6 Miscellaneous

6.1 Grundy number

```

map<set<int>,int> grundy;
map<ll,set<int>> mp;

int get_grundy(set<int> x){
    // base case
    if( sz(x) == 0 ) return 0;
    if( grundy.find(x) != grundy.end() ) return grundy[x];

    set<int> S;
    int res = 0;

    auto iter = x.end(); iter--;
    int mx = *iter;

    // transition : which k to select
    for(int i=1;i<=mx;i++){
        set<int> nxt;
        for(auto e : x){
            if( e < i ) nxt.insert(e);
            else if( e == i ) continue;
            else nxt.insert(e-i);
        }
        S.insert(get_grundy(nxt));
    }

    // find mex and return
    while( S.find(res) != S.end() ) res++;
    grundy[x] = res;
    return res;
}

int main(void){
    int n; geti(n);

    // Simple prime factorization
    rep(i,n){
        ll x; scanf("%lld",&x);
        for(ll i=2;i*i<=x;i++){
            if( x>0 && x%i == 0 ){
                int cnt = 0;
                while( x>0 && x%i == 0 ){
                    cnt++; x/= i;
                }
                mp[i].insert(cnt);
            }
        }
        if( x > 1 ){
            mp[x].insert(1);
        }
    }

    int res = 0;
    for(auto e : mp){
        res ^= get_grundy(e.Se);
    }
}

```

```

    if( res == 0 ) printf("Arpa");
    else printf("Mojtaba");
}

```

6.2 Hungarian

```

// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
// cost[i][j] = cost for pairing left node i with right node j
// Lmate[i] = index of right node that left node i pairs with
// Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
    int n = int(cost.size());

    // construct dual feasible solution
    VD u(n);
    VD v(n);
    for (int i = 0; i < n; i++) {
        u[i] = cost[i][0];
        for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
    }
    for (int j = 0; j < n; j++) {
        v[j] = cost[0][j] - u[0];
        for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
    }

    // construct primal solution satisfying complementary slackness
    Lmate = VI(n, -1);
    Rmate = VI(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (Rmate[j] != -1) continue;
            if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
                Lmate[i] = j;
                Rmate[j] = i;
                mated++;
                break;
            }
        }
    }
}

```

```

VD dist(n);
VI dad(n);
VI seen(n);

// repeat until primal solution is feasible
while (mated < n) {

    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
        dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {

        // find closest
        j = -1;
        for (int k = 0; k < n; k++) {
            if (seen[k]) continue;
            if (j == -1 || dist[k] < dist[j]) j = k;
        }
        seen[j] = 1;

        // termination condition
        if (Rmate[j] == -1) break;

        // relax neighbors
        const int i = Rmate[j];
        for (int k = 0; k < n; k++) {
            if (seen[k]) continue;
            const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
            if (dist[k] > new_dist) {
                dist[k] = new_dist;
                dad[k] = j;
            }
        }
    }

    // update dual variables
    for (int k = 0; k < n; k++) {
        if (k == j || !seen[k]) continue;
        const int i = Rmate[k];
        v[k] += dist[k] - dist[j];
        u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {

```

```

    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
}
Rmate[j] = s;
Lmate[s] = j;

mated++;
}

double value = 0;
for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

return value;
}

```

```

for(int i = 2; i <= n; i++) {
    dp[i] = find(a[i]);
    newLine(b[i], dp[i]);
}
cout << dp[n];
}

```

6.3 Convex Hull trick

```

ll a[MAXN], b[MAXN], dp[MAXN];
ll la[MAXN], lb[MAXN];
int sz, cur, n;

double cross(int x, int y) {
    return (double)(lb[x] - lb[y]) / (la[y] - la[x]);
}

void newLine(ll p, ll q) {
    la[sz] = p;
    lb[sz] = q;

    while(sz > 1 && cross(sz-1, sz-2) > cross(sz, sz-1)) {
        la[sz-1] = la[sz];
        lb[sz-1] = lb[sz];
        sz--;
    }
    sz++;
}

ll find(ll x) {
    while(cur+1 < sz && x > cross(cur, cur+1)) cur++;
    return la[cur] * x + lb[cur];
}

int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    for(int i = 1; i <= n; i++)
        cin >> b[i];

    dp[1] = 0;
    newLine(b[1], 0);
}

```