Mục lục

[1. code.cpp .1](#_Toc119110109)

[2. Dijkstra 2](#_Toc119110110)

[3. Hash 3](#_Toc119110111)

[4. LCA 4](#_Toc119110112)

[5. Number theory 5](#_Toc119110113)

[6. Prim 7](#_Toc119110114)

[7. Lazy SegTree 8](#_Toc119110115)

[8. Tarjan 10](#_Toc119110116)

[9. Z func 11](#_Toc119110117)

[10. KMP 11](#_Toc119110118)

[11. Suffix array 11](#_Toc119110119)

[12. Convex hull and some geometry 12](#_Toc119110120)

[13. Find bridge 12](#_Toc119110121)

[14. Find cut points 13](#_Toc119110122)

[15. Ternary search 14](#_Toc119110123)

[16. Sweep line – closest point 14](#_Toc119110124)

[17. Big prime test 15](#_Toc119110125)

[18. Testing 16](#_Toc119110126)

[19. HLD 17](#_Toc119110127)

[20. Theorems 18](#_Toc119110128)

21. FastIO ………………………………………………………………………..……18

22. Graph………………………………………………………………………..……..21

23. Segment Tree…………………………………………………………………..23

24. Prime Sieve………………………………………………………………………24

25. Hash………………………………………………………………………………24

26. DisjointSet Union………………………………………………………………..25

1. code.cpp

#include <bits/stdc++.h>

using namespace std;

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

#define ordered\_set tree<pair<int, int>, null\_type, less<pair<int, int>>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>

#pragma GCC optimize("O3,unroll-loops")

#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

inline void debugLocal() {

if (!fopen("input.txt", "r"))

return;

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(0); cout.tie(0);

debugLocal();

return 0;

}

1. Dijkstra

const long long INF = 1e18;

int n, m;

vector<vector<pair<int, long long>>> c;

vector<long long> dis, trace;

void dijkstra(int st) {

priority\_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> pq;

trace.assign(n + 1, -1);

dis.assign(n + 1, INF);

dis[st] = 0, trace[st] = st;

pq.push(make\_pair(0, st));

while (pq.size()) {

int u = pq.top().second;

long long du = pq.top().first;

pq.pop();

if (du != dis[u])

continue;

for (int i = 0; i < c[u].size(); i++) {

int v = c[u][i].first;

long long uv = c[u][i].second;

if (dis[v] > du + uv) {

dis[v] = du + uv, trace[v] = u;

pq.push(make\_pair(dis[v], v));

}

}

}

}

vector<int> getPath(int fi) {

int u = fi;

vector<int> path;

path.push\_back(u);

while (u != trace[u] && trace[u] != -1) {

u = trace[u];

path.push\_back(u);

}

reverse(path.begin(), path.end());

return path;

}

int main() {

int u, v, w;

cin >> n >> m;

c.resize(n + 1);

while (m--) {

cin >> u >> v >> w;

c[u].push\_back({v, w});

c[v].push\_back({u, w});

}

dijkstra(1);

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

cout << dis[i] << ": ";

vector<int> path = getPath(i);

for (int j = 0; j < path.size(); j++)

cout << path[j] << ' ';

cout << endl;

}

return 0;

}

1. Hash

const int base = 311, maxN = 1e6 + 5;

const long long m1 = 1e9 + 7, m2 = 1e9 + 9;

long long pow\_base1[maxN], pow\_base2[maxN];

void buildPow(int n) {

pow\_base1[0] = pow\_base2[0] = 1;

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

pow\_base1[i] = (pow\_base1[i - 1] \* base) % m1;

pow\_base2[i] = (pow\_base2[i - 1] \* base) % m2;

}

}

void buildHash(const string &s, vector<long long> &pf\_hash1, vector<long long> &pf\_hash2) {

pf\_hash1[0] = pf\_hash2[0] = 0;

for (int i = 1; i <= s.length(); i++) {

pf\_hash1[i] = (pf\_hash1[i - 1] \* base + s[i - 1]) % m1;

pf\_hash2[i] = (pf\_hash2[i - 1] \* base + s[i - 1]) % m2;

}

}

//index from 1

pair<long long, long long> getHash(int l, int r, vector<long long> &pf\_hash1, vector<long long> &pf\_hash2) {

pair<long long, long long> ans;

ans.first = (pf\_hash1[r] - pf\_hash1[l - 1] \* pow\_base1[r - l + 1] + m1 \* m1) % m1;

ans.second = (pf\_hash2[r] - pf\_hash2[l - 1] \* pow\_base2[r - l + 1] + m2 \* m2) % m2;

return ans;

}

int main() {

string s1, s2;

cin >> s1;

cin >> s2;

buildPow(max(s1.length(), s2.length()) + 1);

vector<long long> pf\_hashs11(s1.length() + 1), pf\_hashs12(s1.length() + 1), pf\_hashs21(s2.length() + 1), pf\_hashs22(s2.length() + 1);

buildHash(s1, pf\_hashs11, pf\_hashs12);

buildHash(s2, pf\_hashs21, pf\_hashs22);

for (int i = 1; i <= s1.length(); i++) {

int j = i + s2.length() - 1;

if (j > s1.length())

break;

pair<long long, long long> p1 = getHash(i, j, pf\_hashs11, pf\_hashs12), p2 = getHash(1, s2.length(), pf\_hashs21, pf\_hashs22);

//cout << p1.first << '|' << p1.second << ' ' << p2.first << '|' << p2.second << endl;

if (p1 == p2)

cout << i << ' ';

}

cout << endl;

return 0;

}

1. LCA

//2^max\_h >= n

const int max\_h = 20;

vector<vector<int>> c;

vector<vector<int>> up;

vector<int> depth;

void buildBL(int u, int par) {

up[u][0] = par;

for (int i = 1; i < max\_h; i++)

up[u][i] = up[up[u][i - 1]][i - 1];

for (int v : c[u]) {

if (v == par)

continue;

depth[v] = depth[u] + 1;

buildBL(v, u);

}

}

int upByK(int u, int k) {

bitset<max\_h> bs(k);

for (int i = 0; i < max\_h; i++) {

if (bs[i])

u = up[u][i];

}

return u;

}

int getLCA(int u, int v) {

if (depth[u] < depth[v])

swap(u, v);

int depth\_diff = depth[u] - depth[v];

u = upByK(u, depth\_diff);

for (int i = max\_h - 1; i >= 0; i--) {

if (up[u][i] != up[v][i]) {

u = up[u][i];

v = up[v][i];

}

}

if (u != v)

u = up[u][0];

return u;

}

//make long long if need

int dist(int u, int v) {

int lca = getLCA(u, v);

return depth[u] + depth[v] - 2 \* depth[lca];

}

int main() {

int n, q, a, b;

cin >> n >> q;

c.resize(n);

depth.assign(n, 0);

up.assign(n, vector<int>(max\_h));

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

cin >> a;

c[i].push\_back(a);

c[a].push\_back(i);

}

buildBL(0, 0);

while (q--) {

cin >> a >> b;

cout << getLCA(a, b) << '\n';

}

return 0;

}

1. Number theory

long long modExpo(long long x, long long n, long long M){

if(n == 0)

return 1;

else if(n % 2 == 0)

return modExpo((x \* x) % M, n / 2, M);

else

return (x \* modExpo((x \* x) % M, (n - 1) / 2, M)) % M;

}

int d,x,y;

void extEuclid(int A, int B) {

if(B == 0) {

d = A;

x = 1;

y = 0;

} else {

extEuclid(B, A % B);

int temp = x;

x = y;

y = temp - (A / B) \* y;

}

}

//gcd(A, M) == 1

int modInvEE(int A, int M) {

extEuclid(A, M);

return (x % M + M) % M;

}

//isPrime(M) == true

int modInvFE(int A,int M) {

return modExpo(A,M-2,M);

}

void sieve(int n, vector<bool> &isPrime) {

isPrime.assign(n + 1, true);

isPrime[0] = false;

isPrime[1] = false;

for(int i = 2; i \* i <= n; ++i) {

if(isPrime[i] == true) {

for(int j = i \* i; j <= n; j += i)

isPrime[j] = false;

}

}

}

void facSieve(int n, vector<int> &minPrime) {

minPrime.assign(n + 1, 0);

for (int i = 2; i \* i <= n; ++i) {

if (minPrime[i] == 0) {

for (int j = i \* i; j <= n; j += i) {

if (minPrime[j] == 0) {

minPrime[j] = i;

}

}

}

}

for (int i = 2; i <= n; ++i) {

if (minPrime[i] == 0) {

minPrime[i] = i;

}

}

}

vector<int> factorize(int n, vector<int> &minPrime) {

vector<int> res;

while (n != 1) {

res.push\_back(minPrime[n]);

n /= minPrime[n];

}

return res;

}

void rangeSieve(int L, int R, vector<bool> &isPrime) {

isPrime.assign(R - L + 1, true);

for (long long i = 2; i \* i <= R; ++i) {

for (long long j = max(i \* i, (L + i - 1) / i \* i); j <= R; j += i) {

isPrime[j - L] = false;

}

}

if (1 >= L) {

isPrime[1 - L] = false;

}

}

int xorOneToN(int n) {

if (n % 4 == 0)

return n;

if (n % 4 == 1)

return 1;

if (n % 4 == 2)

return n + 1;

return 0;

}

int main() {

cout << modExpo(2, 1000, 1e9 + 7) << endl;

cout << modInvEE(2, 1e9 + 7) << endl;

cout << modInvFE(2, 1e9 + 7) << endl;

vector<bool> isPrime;

sieve(1e6, isPrime);

for (int i = 0; i <= 10; i++) {

if (isPrime[i])

cout << i << ' ';

}

cout << endl;

vector<int> minPrime, facs;

facSieve(1e6, minPrime);

facs = factorize(100, minPrime);

for (int i = 0; i < facs.size(); i++)

cout << facs[i] << ' ';

cout << endl;

vector<bool> rgSieve;

int L = 1e7, R = 1e7 + 1e4;

rangeSieve(L, R, rgSieve);

for (long long x = L; x <= R; ++x) {

if (rgSieve[x - L]) {

cout << x << ' ';

}

}

cout << endl;

return 0;

}

1. Prim

const int INF = 1e18;

int n, m;

vector<vector<pair<int, long long>>> c;

vector<long long> dis;

long long prim(int s) {

long long ret = 0;

priority\_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> q;

fill(dis.begin(), dis.end(), INF);

dis[s] = 0;

q.push({0, s});

while (!q.empty()) {

auto top = q.top(); q.pop();

long long curDis = top.first; int u = top.second;

if (curDis != dis[u]) continue;

ret += dis[u]; dis[u] = -INF;

for (auto &e : c[u]) {

int v = e.first; long long cc = e.second;

if (dis[v] > cc) {

dis[v] = cc;

q.push({dis[v], v});

}

}

}

return ret;

}

int main() {

ios\_base::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

cin >> n >> m;

c.clear();

c.resize(n + 1);

dis.assign(n + 1, INF);

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

int u, v, cc;

cin >> u >> v >> cc;

c[u].push\_back({v, cc});

c[v].push\_back({u, cc});

}

cout << prim(1) << '\n';

}

1. Lazy SegTree

struct Node {

int val, lazy;

Node(int dat, int lzy) {

val = dat;

lazy = lzy;

}

};

struct SegTree {

Node SKIP\_VALUE = Node(0, 0);

int ts;

vector<Node> ST;

SegTree(int tsize) {

ts = 1;

while (ts < tsize)

ts \*= 2;

ST.assign(2 \* ts + 2, SKIP\_VALUE);

}

Node mergeN(Node n1, Node n2) {

return Node(max(n1.val, n2.val), 0);

}

void pushDown(int id) {

int lz = ST[id].lazy;

ST[id \* 2].val += lz;

ST[id \* 2].lazy += lz;

ST[id \* 2 + 1].val += lz;

ST[id \* 2 + 1].lazy += lz;

ST[id].lazy = 0;

}

void build(int id, int l, int r, vector<int> &a, int n) {

if (l >= n) {

ST[id] = SKIP\_VALUE;

return;

}

if (l == r) {

ST[id] = Node(a[l], 0);

return;

}

int mid = (l + r) / 2;

build(id \* 2, l, mid, a, n);

build(id \* 2 + 1, mid + 1, r, a, n);

ST[id] = mergeN(ST[id \* 2], ST[id \* 2 + 1]);

}

void build(vector<int> &a, int n) {

build(1, 0, ts, a, n);

}

void update(int id, int l, int r, int u, int v, int val) {

if (v < l || r < u) {

return;

}

if (u <= l && r <= v) {

ST[id].val += val;

ST[id].lazy += val;

return ;

}

pushDown(id);

int mid = (l + r) / 2;

update(id \* 2, l, mid, u, v, val);

update(id \* 2 + 1, mid + 1, r, u, v, val);

ST[id] = mergeN(ST[id \* 2], ST[id \* 2 + 1]);

}

void update(int l, int r, int val) {

update(1, 0, ts, l, r, val);

}

Node get(int id, int l, int r, int u, int v) {

if (v < l || r < u) {

return SKIP\_VALUE;

}

if (u <= l && r <= v) {

return ST[id];

}

pushDown(id);

int mid = (l + r) / 2;

return mergeN(get(id \* 2, l, mid, u, v), get(id \* 2 + 1, mid + 1, r, u, v));

}

int get(int l, int r) {

Node rs = get(1, 0, ts, l, r);

return rs.val;

}

};

//Note: LazySegTree index 0

int main() {

int n, m, t, x, y, k;

cin >> n >> m;

vector<int> arr(n, 0);

SegTree ST = SegTree(n);

ST.build(arr, n);

while (m--) {

cin >> t;

if (t == 0) {

cin >> x >> y >> k;

ST.update(x - 1, y - 1, k);

} else {

cin >> x >> y;

cout << ST.get(x - 1, y - 1) << '\n';

}

}

return 0;

}

1. Tarjan

const int N = 100005;

const int oo = 1e9;

int n, m, Num[N], Low[N], cnt = 0;

vector<int> a[N];

stack<int> st;

int Count = 0;

void visit(int u) {

Low[u] = Num[u] = ++cnt;

st.push(u);

for (int v : a[u])

if (Num[v])

Low[u] = min(Low[u], Num[v]);

else {

visit(v);

Low[u] = min(Low[u], Low[v]);

}

if (Num[u] == Low[u]) { //found

Count++;

int v;

do {

v = st.top();

st.pop();

Num[v] = Low[v] = oo;

} while (v != u);

}

}

int main() {

scanf("%d%d", &n, &m);

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

int x, y;

scanf("%d%d", &x, &y);

a[x].push\_back(y);

}

for (int i = 1; i <= n; i++)

if (!Num[i]) visit(i);

cout << Count << endl;

}

1. Z func

vector<int> z\_function(string s) {

int n = (int) s.length();

vector<int> z(n);

for (int i = 1, l = 0, r = 0; i < n; ++i) {

if (i <= r)

z[i] = min (r - i + 1, z[i - l]);

while (i + z[i] < n && s[z[i]] == s[i + z[i]])

++z[i];

if (i + z[i] - 1 > r)

l = i, r = i + z[i] - 1;

}

return z;

}

1. KMP

vector<int> prefix\_function(string s) {

int n = (int)s.length();

vector<int> pi(n);

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

int j = pi[i-1];

while (j > 0 && s[i] != s[j])

j = pi[j-1];

if (s[i] == s[j])

j++;

pi[i] = j;

}

return pi;

}

1. Suffix array

// sorted suffix, ans[i] = x => suffix start at x

vector<int> sort\_cyclic\_shifts(string const& s) {

int n = s.size();

const int alphabet = 256;

vector<int> p(n), c(n), cnt(max(alphabet, n), 0);

for (int i = 0; i < n; i++)

cnt[s[i]]++;

for (int i = 1; i < alphabet; i++)

cnt[i] += cnt[i-1];

for (int i = 0; i < n; i++)

p[--cnt[s[i]]] = i;

c[p[0]] = 0;

int classes = 1;

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

if (s[p[i]] != s[p[i-1]])

classes++;

c[p[i]] = classes - 1;

}

1. Convex hull and some geometry

//Convex Hull

const int N = 20000;

struct Point {

long long x, y;

bool operator<(const Point &v) const { return x == v.x ? y < v.y : x < v.x; }

long long cross(const Point &p, const Point &q) const { return (p.x - x) \* (q.y - y) - (p.y - y) \* (q.x - x); }

} p[N], poly[N];

int n;

void enter() {

scanf("%d", &n);

for (int i = 0; i < n; ++i)

scanf("%lld%lld", &p[i].x, &p[i].y);

}

long long size(Point poly[], int k) {

long long S = (poly[k - 1].x - poly[0].x) \* (poly[k - 1].y + poly[0].y);

for (int i = 1; i < k; ++i)

S += (poly[i - 1].x - poly[i].x) \* (poly[i - 1].y + poly[i].y);

return S;

printf("%lld\n", S);

}

void solve() {

sort(p, p + n);

int k = 0;

for (int i = 0; i < n; ++i) {

while (k >= 2 && poly[k - 2].cross(poly[k - 1], p[i]) <= 0) --k;

poly[k++] = p[i];

}

for (int i = n - 2, t = k + 1; i >= 0; --i) {

while (k >= t && poly[k - 2].cross(poly[k - 1], p[i]) <= 0) --k;

poly[k++] = p[i];

}

printf("%lld\n", size(poly, k));

}

//get vector point1->point2 (to check inline without error)

pair<long long, long long> getVector(pair<long long, long long> p1, pair<long long, long long> p2) {

return {p1.first - p2.first, p1.second - p2.second};

}

//check inline point1 point2 point3 with vector point1->point2

bool isInLine(pair<long long, long long> p1, pair<long long, long long> p2, pair<long long, long long> vec) {

return ((p1.first - p2.first) \* vec.second) == ((p1.second - p2.second) \* vec.first);

}

1. Find bridge

int n; // number of nodes

vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;

vector<int> tin, low;

int timer;

void dfs(int v, int p = -1) {

visited[v] = true;

tin[v] = low[v] = timer++;

for (int to : adj[v]) {

if (to == p) continue;

if (visited[to]) {

low[v] = min(low[v], tin[to]);

} else {

dfs(to, v);

low[v] = min(low[v], low[to]);

if (low[to] > tin[v])

IS\_BRIDGE(v, to);

}

}

}

void find\_bridges() {

timer = 0;

visited.assign(n, false);

tin.assign(n, -1);

low.assign(n, -1);

for (int i = 0; i < n; ++i) {

if (!visited[i])

dfs(i);

}

}

1. Find cut points

int n; // number of nodes

vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;

vector<int> tin, low;

int timer;

void dfs(int v, int p = -1) {

visited[v] = true;

tin[v] = low[v] = timer++;

int children=0;

for (int to : adj[v]) {

if (to == p) continue;

if (visited[to]) {

low[v] = min(low[v], tin[to]);

} else {

dfs(to, v);

low[v] = min(low[v], low[to]);

if (low[to] >= tin[v] && p!=-1)

IS\_CUTPOINT(v);

++children;

}

}

if(p == -1 && children > 1)

IS\_CUTPOINT(v);

}

void find\_cutpoints() {

timer = 0;

visited.assign(n, false);

tin.assign(n, -1);

low.assign(n, -1);

for (int i = 0; i < n; ++i) {

if (!visited[i])

dfs (i);

}

}

1. Ternary search

double max\_f(double left, double right) {

int N\_ITER = 100;

for (int i = 0; i < N\_ITER; i++) {

double x1 = left + (right - left) / 3.0;

double x2 = right - (right - left) / 3.0;

if (f(x1) > f(x2)) right = x2;

else left = x1;

}

return f(left);

}

1. Sweep line – closest point

#define ll long long

struct Point{

ll x, y;

int id;

bool operator < (const Point& other) {

if (x != other.x) return x < other.x;

return y < other.y;

}

};

struct cmp{

bool operator () (const Point& a, const Point& b) const {

if (a.y != b.y) return a.y < b.y;

return a.x < b.x;

}

};

int n;

vector<Point> points; // Vector chứa tất cả các điểm

set<Point, cmp> T;

ll squared\_dist(Point a, Point b) { // Nhận vào hai điểm, trả vể

// bình phương khoảng cách giữa hai điểm

return (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);

}

signed main() {

ios\_base::sync\_with\_stdio(false); cin.tie(NULL);

cin >> n;

for (int i = 0; i < n; i++) {

ll x, y;

cin >> x >> y;

points.push\_back({x, y, i});

}

ll squared\_d = squared\_dist(points[0], points[1]); // Lưu bình phương của d

int res\_id1 = 0, res\_id2 = 1;

sort(points.begin(), points.end()); // Sắp xếp các điểm theo hoành độ

for (auto p : points) {

ll x = p.x, y = p.y;

int id = p.id;

ll d = sqrt(squared\_d);

Point cur = {-1000001, y - d, id};

while (1) { // Tìm tất cả các điểm có tung độ trong khoảng [y - d, y + d]

auto it = T.upper\_bound(cur);

if (it == T.end()) break;

cur = \*it;

if (cur.y > y + d) break; // Dừng lại nếu điểm có tung độ lớn hơn y + d

if (cur.x < x - d) {

T.erase(it);

continue;

} // Xóa điểm nếu điểm này có hoành độ bé hơn x - d

if (squared\_dist(p, cur) < squared\_d) {

squared\_d = squared\_dist(p, cur);

res\_id1 = id; res\_id2 = cur.id;

} // Gán đáp án mới nếu tìm được d nhỏ hơn

}

T.insert(p); // Thêm điểm hiện tại vào T

}

if (res\_id1 > res\_id2) swap(res\_id1, res\_id2);

cout << res\_id1 << " " << res\_id2 << " ";

cout << fixed << setprecision(6) << sqrt(squared\_d);

}

1. Big prime test

ll nt[1000010];

ll nhan(ll a, ll b, ll mod)

{

if (b == 0) return 0%mod;

if (b == 1) return a%mod;

ll g = nhan(a,b/2,mod);

if (b%2) return ((g+g)%mod+a)%mod;

return (g+g)%mod;

}

ll modpow(ll a, ll b, ll mod)

{

if (b == 0) return 1%mod;

if (b == 1) return a%mod;

ll g = modpow(a,b/2,mod);

if (b%2) return nhan(nhan(g,g,mod),a,mod);

return nhan(g,g,mod);

}

bool MillerRabin(ll n, ll seed)

{

ll k = 0;

if (n < 2) return false;

if (n == 2) return true;

if (!(n & 1)) return false;

ll m = n - 1;

while (!(m & 1)) m >>= 1, k++;

ll a = seed;

a = modpow(a, m, n);

if (a == 1 || a == n - 1) return true;

for (ll j = 0; j < k - 1; j++)

{

a = modpow(a, 2, n);

if (a == 1) return false;

if (a == n - 1) return true;

}

return false;

}

void Sieve()

{

FOR(i,2,1000000)

if (nt[i] == 0) {

nt[i] = 1;

for (ll j = i\*i; j <= 1000000; j += i) nt[j] = -1;

}

}

bool PrimalityTest(ll n)

{

if (n <= 1000000) return nt[n] == 1;

else return MillerRabin(n, 2) && MillerRabin(n, 13) && MillerRabin(n, 23) && MillerRabin(n, 1662803);

}

int main()

{

ll t, n, k;

ios::sync\_with\_stdio(0);

//freopen("inp.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

Sieve();

cin >> t;

while (t--) {

cin >> n >> k;

if (n < 2\*k) cout << "No\n";

else if (n == 2\*k) cout << "Yes\n";

else if (k == 1) cout << (PrimalityTest(n) ? "Yes" : "No") << endl;

else if (n % 2 == 0) cout << "Yes\n";

else {

if (n == 5) cout << "Yes\n";

else if (k == 2) cout << (PrimalityTest(n-2) ? "Yes" : "No") << endl;

else cout << "Yes\n";

}

}

return 0;

}

1. Testing

// inp.txt (file in), out.txt (file out bai lam), ans.txt (file out trau)

#include <bits/stdc++.h>

using namespace std;

// So test kiem tra

const int NTEST = 100;

// Ham random nay sinh random so trong pham vi long long, so sinh ra >= l và <= h.

long long Rand(long long l, long long h)

{

return l + ((long long)rand() \* (RAND\_MAX + 1) \* (RAND\_MAX + 1) \* (RAND\_MAX + 1) +

(long long)rand() \* (RAND\_MAX + 1) \* (RAND\_MAX + 1) +

(long long)rand() \* (RAND\_MAX + 1) +

rand()) % (h - l + 1);

}

int main()

{

srand(time(NULL));

for(int iTest = 1; iTest <= NTEST; iTest++)

{

ofstream inp("inp.txt");

// Code sinh test

// inp << n << "\n";

//for(int i = 1; i <= n; i++)

//{

// inp << MINV + rand() % (MAXV - MINV + 1) << " ";

//}

inp.close();

system("main.exe");

system("trau.exe");

if(system("fc out.txt ans.txt") != 0)

{

cout << "Test " << iTest << ": WRONG!\n";

return 0;

}

cout << "Test " << iTest << ": CORRECT!\n";

}

return 0;

}

1. HLD

vector<int> parent, depth, heavy, head, pos;

int cur\_pos;

int dfs(int v, vector<vector<int>> const& adj) {

int size = 1;

int max\_c\_size = 0;

for (int c : adj[v]) {

if (c != parent[v]) {

parent[c] = v, depth[c] = depth[v] + 1;

int c\_size = dfs(c, adj);

size += c\_size;

if (c\_size > max\_c\_size)

max\_c\_size = c\_size, heavy[v] = c;

}

}

return size;

}

void decompose(int v, int h, vector<vector<int>> const& adj) {

head[v] = h, pos[v] = cur\_pos++;

if (heavy[v] != -1)

decompose(heavy[v], h, adj);

for (int c : adj[v]) {

if (c != parent[v] && c != heavy[v])

decompose(c, c, adj);

}

}

void init(vector<vector<int>> const& adj) {

int n = adj.size();

parent = vector<int>(n);

depth = vector<int>(n);

heavy = vector<int>(n, -1);

head = vector<int>(n);

pos = vector<int>(n);

cur\_pos = 0;

dfs(0, adj);

decompose(0, 0, adj);

}

int query(int a, int b) {

int res = 0;

for (; head[a] != head[b]; b = parent[head[b]]) {

if (depth[head[a]] > depth[head[b]])

swap(a, b);

int cur\_heavy\_path\_max = segment\_tree\_query(pos[head[b]], pos[b]);

res = max(res, cur\_heavy\_path\_max);

}

if (depth[a] > depth[b])

swap(a, b);

int last\_heavy\_path\_max = segment\_tree\_query(pos[a], pos[b]);

res = max(res, last\_heavy\_path\_max);

return res;

}

1. Theorems

* Số ước của n=p1^x\*p2\*y.. với p1, p2.. là ước nguyên tố của n => số ước bằng (x+1)\*(y+1)..
* Từ 0 tới n có xấp xỉ n / ln(n) số nguyên tố
* Cách tính số lượng số nguyên tố cùng nhau với n và nhỏ hơn bằng n: n = p1^x\*p2^y.. => đáp án là: p1^(x-1)\*(p1-1)\*p2^(y-1)\*(p2-1)..
* Nim game: Có n hộp, mỗi hộp chứa 1 số lượng sỏi, người chơi lần lượt bốc sỏi trong những hộp còn sỏi, ai không đi được thì thua. VD có n hộp với số lượng sỏi x1,x2..xn. Gọi s = x1^x2^..^xn. Nếu s = 0 thì thua ngược lại thắng.
* Chia x viên sỏi vào y hộp => (y+x-1)C(y-1)
* Số lương dãy ngược đúng độ dài 2\*n: C(n) = (2\*nCn)/(n+1). Số lương cây nhị phân có n nút là C(n), số lương cây n nút là C(n-1)

1. Fast IO:

**import** io,os

input = io.**BytesIO**(os.read(0,os.fstat(0).st\_size)).readline

**import** sys

input = sys.stdin.readline

sys.stdout.write(str(n) + **"\n"**)

sys.stdout.write(**" "**.**join**(map(str,list)) + **"\n"**)

from queue import PriorityQueue

from collections import deque,OrderedDict,defaultdict,Counter

from heapq import heappop, heappush, heapify

import sys

import time

import os

from io import BytesIO, IOBase

import math

from random import randint

from itertools import compress

1. Graph

class Graph:

    INF=10\*\*9

    #Searching Algorithms: DFS, BFS

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

        self.count=defaultdict(lambda:0)

    def addEdge(self,u,v):

        self.graph[u].append(v)

        self.graph[v].append(u)

    # Algorithms for finding the shortest path: Dijsktra

    # def \_\_init\_\_(self,vertices):

    #     self.distance=[self.INF]\*vertices

    #     self.edges=defaultdict(dict)

    # def addEdge(self,u,v,w):

    #     self.edges[u][v]=w

    #Minimum Spanning Trees: Kruskal's Algorithm

    # def \_\_init\_\_(self, vertices):

    #     self.vertices=vertices

    #     self.graph=[]

    # def addEdge(self,node1,node2,weight):

    #     self.graph.append([node1,node2,weight])

    # 1.Handling A Disconnected Graph:

    def DFS(self,numVertex):

        visited=defaultdict(lambda:0)

        path=[]

        for vertex in range(1,numVertex+1):

            if visited[vertex]==0:

                self.DFSUtil(vertex,visited,path)

        return path

    # 2.DFS from a vertex

    # def DFS(self,vertex):

    #     path=[]

    #     stack=[vertex]

    #     visited=defaultdict(lambda:0)

    #     while len(stack):

    #         u=stack.pop()

    #         if visited[u]==0:

    #             path.append(u)

    #             visited[u]=1

    #             for neighbour in self.graph[u]:

    #                 stack.append(neighbour)

    #     return path

    # 1. Handling A Disconnected Graph:

    def BFS(self,numVertex):

        visited=defaultdict(lambda:0)

        queue=[]

        for i in range(1,numVertex+1):

            if visited[i]==0:

                queue.append(i)

                visited[i]=1

                while queue:

                    u=queue.pop(0)

                    print(u,end=' ')

                    for v in self.graph[u]:

                        if visited[v]==0:

                            queue.append(v)

                            visited[v]=1

    # 2. BFS from a vertex

    # def BFS(self,vertex):

    #     path=[]

    #     queue=[vertex]

    #     visited=defaultdict(lambda:0)

    #     while len(queue):

    #         u=queue.pop(0)

    #         if visited[u]==0:

    #             path.append(u)

    #             visited[u]=1

    #             for neighbour in self.graph[u]:

    #                 queue.append(neighbour)

    #     return path

    def connectedComponentsBFS(self,numVertex):

        visited=defaultdict(lambda:0)

        stack=[]

        path=[]

        for i in range(1,numVertex+1):

            if visited[i]==0:

                stack.append(i)

                visited[i]=1

                temp=[]

                while stack:

                    u=stack.pop()

                    temp.append(u)

                    for v in self.graph[u]:

                        if visited[v]==0:

                            stack.append(v)

                            visited[v]=1

                path.append(temp)

        return path

    def Dijsktra(self,S):

        self.distance[S]=0

        queue=[(0,S)]

        trace=defaultdict(lambda:-1)

        while queue:

            cost,vertex=heappop(queue)

            for neighbour, weight in self.edges[vertex].items():

                if cost+weight<self.distance[neighbour]:

                    self.distance[neighbour]=cost+weight

                    heappush(queue, (cost + weight, neighbour))

                    trace[neighbour]=vertex

        #Trả về đường đi từ đỉnh S đến u nào đó

        #return trace

        # if u!=S and trace[u]==-1:

        #     return -1 #Không có đường đi

        # else:

        #     path=[]

        #     while u!=-1:

        #         path.append(u)

        #         u=trace[u]

        #     path.reverse()

        #     return path

        return self.distance

    def FloydWarshall(self,vertices):

        for k in range(1,vertices+1):

            for i in range(1,vertices+1):

                for j in range(1,vertices+1):

                   self.distance[i][j]=min(self.distance[i][j],self.edges[i][k]+self.edges[k][j])

        return self.distance

1. Segment Tree

1//

class SegmentTree:

def \_\_init\_\_(self,mang,n):

self.a=mang

self.N=n

self.tree=[0]\*(4\*self.N)

self.INF\_MIN=-10\*\*9

def showTree(self):

print(self.tree)

def buildTree(self,id,l,r):

#Chú ý khi khởi tạo t.buildTree(id,l,r), id luôn phải bằng 1

#l=0, r=n-1 hoặc ta phải thêm một phần tử bất kỳ vào đẩu mảng a rồi l=1, r=n

if l==r:

self.tree[id]=self.a[l];

return

m=(l+r)//2;

self.buildTree(id\*2,l,m)

self.buildTree(id\*2+1,m+1,r)

self.tree[id]=max(self.tree[id\*2],self.tree[id\*2+1])

def updateTree(self, id, l, r, u, v, val): #u,v là đoạn cần cập nhật

if u>r or v<l:

return

if l==r:

self.tree[id]+=val

return

m=(l+r)//2

self.updateTree(id\*2,l,m,u,v,val)

self.updateTree(id\*2+1,m+1,r,u,v,val)

self.tree[id]=max(self.tree[id\*2],self.tree[id\*2+1])

# def updateTree(self,id,l,r,i,v):

# if i<l or i>r:

# return

# if l==r:

# self.tree[id]=i

# return

# m=(l+r)//2

# self.updateTree(id\*2,l,m,i,v)

# self.updateTree(id\*2+1,m+1,r,i,v)

# self.tree[id]=max(self.tree[id\*2],self.tree[id\*2+1])

def getValue(self,id,l,r,u,v): #u,v là đoạn lấy giá trị lớn nhất

if u>r or v<l:

#Đoạn [u, v] không giao với đoạn [l, r], ta bỏ qua đoạn này

return self.INF\_MIN

if l==r:

return self.tree[id]

m=(l+r)//2

return max(self.getValue(id\*2,l,m,u,v),self.getValue(id\*2+1,m+1,r,u,v))

2//

class SegmentTree:

    def \_\_init\_\_(self, mang, n):

        self.a = mang

        self.N = n

        self.tree = [0]\*(4\*self.N)

        self.INF\_MIN = -10\*\*9

    def showTree(self):

        print(self.tree)

    def buildTree(self, id, l, r):

        # Chú ý khi khởi tạo t.buildTree(id,l,r), id luôn phải bằng 1

        # l=0, r=n-1 hoặc ta phải thêm một phần tử bất kỳ vào đẩu mảng a rồi l=1, r=n

        if l == r:

            self.tree[id] = self.a[l]

            return

        m = (l+r)//2

        self.buildTree(id\*2, l, m)

        self.buildTree(id\*2+1, m+1, r)

        self.tree[id] = max(self.tree[id\*2], self.tree[id\*2+1])

    def updateTree(self, id, l, r, u, v, val):  # u,v là đoạn cần cập nhật

        if u > r or v < l:

            return

        if l == r:

            self.tree[id] += val

            return

        m = (l+r)//2

        self.updateTree(id\*2, l, m, u, v, val)

        self.updateTree(id\*2+1, m+1, r, u, v, val)

        self.tree[id] = max(self.tree[id\*2], self.tree[id\*2+1])

    # def updateTree(self,id,l,r,i,v):

    #     if i<l or i>r:

    #         return

    #     if l==r:

    #         self.tree[id]=i

    #         return

    #     m=(l+r)//2

    #     self.updateTree(id\*2,l,m,i,v)

    #     self.updateTree(id\*2+1,m+1,r,i,v)

    #     self.tree[id]=max(self.tree[id\*2],self.tree[id\*2+1])

    def getValue(self, id, l, r, u, v):  # u,v là đoạn lấy giá trị lớn nhất

        if u > r or v < l:

            # Đoạn [u, v] không giao với đoạn [l, r], ta bỏ qua đoạn này

            return self.INF\_MIN

        if l == r:

            return self.tree[id]

        m = (l+r)//2

        return max(self.getValue(id\*2, l, m, u, v), self.getValue(id\*2+1, m+1, r, u, v))

1. Sieve

def rwh\_primes1v1(n):

    """ Returns  a list of primes < n for n > 2 """

    sieve = bytearray([True]) \* (n//2)

    for i in range(3,int(n\*\*0.5)+1,2):

        if sieve[i//2]:

            sieve[i\*i//2::i] = bytearray((n-i\*i-1)//(2\*i)+1)

    return [2,\*compress(range(3,n,2), sieve[1:])]

1. Hash

class Hash:

def \_\_init\_\_(self):

self.MOD=10\*\*9+7

self.base=31

self.pow=[0]\*10\*\*6

self.pow[0]=1

for i in range(1,10\*\*6):

self.pow[i]=self.pow[i-1]\*self.base%self.MOD

def Hash(self,s):

hash=[0]\*(len(s)+1)

for i in range(1,len(s)+1):

hash[i]=(hash[i-1]\*self.base + ord(s[i-1])-97+1 ) % self.MOD

return hash

def getHash(self,i,j,hash):

return (hash[j] - hash[i-1]\*self.pow[j-i+1]) % self.MOD

def subOccurrences(self,string,subString):

hash\_string=self.Hash(string)

hash\_subString=self.Hash(subString)[-1]

occurrences=[]

for i in range(1,len(string)-len(subString)+2):

if hash\_subString==self.getHash(i,i+len(subString)-1,hash\_string):

occurrences.append(i)

return occurrences

1. DisjointSet

class DisjSet:

def \_\_init\_\_(self, n):

self.rank = [1] \* (n+1)

self.parent = [i for i in range(n+1)]

def find(self, x):

if (self.parent[x] != x):

self.parent[x] = self.find(self.parent[x])

return self.parent[x]

def Union(self, x, y):

xroot = self.find(x)

yroot = self.find(y)

if xroot == yroot:

return

if self.rank[xroot] < self.rank[yroot]:

self.parent[xroot] = yroot

elif self.rank[xroot] > self.rank[yroot]:

self.parent[yroot] = xroot

else:

self.parent[yroot] = xroot

self.rank[xroot] = self.rank[xroot] + 1

1. Persistent Segment Tree

struct Node {

int left, right; // ID of left child & right child

long long ln; // Max value of node

Node() {}

Node(long long ln, int left, int right) : ln(ln), left(left), right(right) {}

} it[11000111]; // Each node has a position in this array, called ID

int nNode;

int ver[MN]; // ID of root in each version

// Update max value of a node

inline void refine(int cur) {

it[cur].ln = max(it[it[cur].left].ln, it[it[cur].right].ln);

}

// Update a range, and return new ID of node

int update(int l, int r, int u, int x, int oldId) {

if (l == r) {

++nNode;

it[nNode] = Node(x, 0, 0);

return nNode;

}

int mid = (l + r) >> 1;

int cur = ++nNode;

if (u <= mid) {

it[cur].left = update(l, mid, u, x, it[oldId].left);

it[cur].right = it[oldId].right;

refine(cur);

}

else {

it[cur].left = it[oldId].left;

it[cur].right = update(mid+1, r, u, x, it[oldId].right);

refine(cur);

}

return cur;

}

// Get max of range. Same as usual IT

int get(int nodeId, int l, int r, int u, int v) {

if (v < l || r < u) return -1;

if (u <= l && r <= v) return it[nodeId].ln;

int mid = (l + r) >> 1;

return max(get(it[nodeId].left, l, mid, u, v), get(it[nodeId].right, mid+1, r, u, v));

}

// When update:

++nVer;

ver[nVer] = update(1, n, u, x, ver[nVer-1]);

// When query:

res = get(ver[t], 1, n, u, v);