**2SAT 求方案，CF 题目 Gym B题，三个团。**

/\* http://codeforces.com/gym/100268 \*/

#pragma comment(linker, "/STACK:36777216")

#include <map>

#include <set>

#include <cmath>

#include <ctime>

#include <queue>

#include <stack>

#include <cstdio>

#include <string>

#include <vector>

#include <iomanip>

#include <cassert>

#include <cstdlib>

#include <cstring>

#include <iostream>

#include <algorithm>

#define lc(x) (x << 1)

#define rc(x) (lc(x) + 1)

#define lowbit(x) (x & (-x))

#define PI (acos(-1))

#define lowbit(x) (x & (-x))

#define LL long long

#define DB double

#define ULL unsigned long long

#define PII pair<int, int>

#define PLL pair<LL, LL>

#define PB push\_back

#define MP make\_pair

using namespace std;

const int N = 2005;

const int INF = 1e9 + 7;

const int MOD = 1e9 + 7;

const DB EPS = 1e-8;

const ULL MAGIC = 79;

struct Edge{

int v, next;

}e[2 \* N \* N];

int first[4 \* N], ecnt;

void Addedge(int u, int v){

e[++ ecnt].next = first[u], first[u] = ecnt, e[ecnt].v = v;

}

int n, dfn[4 \* N], low[4 \* N], scc\_id[4 \* N], ins[4 \* N], ts, scc\_cnt;

stack <int> stk;

void Dfs(int u){

stk.push(u);

dfn[u] = low[u] = ++ ts;

ins[u] = 1;

for(int i = first[u]; i != -1; i = e[i].next){

int v = e[i].v;

if(!dfn[v])

Dfs(v), low[u] = min(low[u], low[v]);

else if(ins[v])

low[u] = min(low[u], dfn[v]);

}

if(dfn[u] == low[u]){

scc\_cnt ++;

while(1){

int x = stk.top(); stk.pop();

scc\_id[x] = scc\_cnt;

ins[x] = false;

if(x == u) break;

}

}

}

void Tarjan(){

ts = 0;

for(int i = 0; i < 2 \* n; i ++)

if(!dfn[i]) Dfs(i);

}

vector <int> block[2 \* N]; int c[2 \* N];

bool \_2Sat(vector <bool> &res){

Tarjan();

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

if(scc\_id[2 \* i] == scc\_id[2 \* i + 1])

return false;

for(int i = 1; i <= scc\_cnt; i ++)

block[i].clear(), c[i] = -1;

for(int i = 0; i < 2 \* n; i ++) block[scc\_id[i]].PB(i);

for(int i = 1; i <= scc\_cnt; i ++){

c[i] = 1;

for(int j = 0; j < block[i].size(); j ++){

int u = block[i][j];

for(int k = first[u]; k != -1; k = e[k].next){

int v = e[k].v;

if(c[scc\_id[v]] == 0) c[i] = 0;

}

if(c[scc\_id[u ^ 1]] == 1) c[i] = 0;

}

}

res.resize(n);

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

res[i] = c[scc\_id[2 \* i]];

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

// cout << res[i] << ' '; cout << endl;

return true;

}

int m, sta[2 \* N], d[N][N], u, v;

vector <bool> res;

int main(){

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

cin >> n >> m;

memset(sta, -1, sizeof(sta));

memset(first, -1, sizeof(first)), ecnt = -1;

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

scanf("%d%d", &u, &v);

u --, v --;

if(u == 0) sta[v] = 0;

if(v == 0) sta[u] = 0;

if(u == 1) sta[v] = 1;

if(v == 1) sta[u] = 1;

d[u][v] = d[v][u] = true;

}

sta[0] = 0, sta[1] = 1;

Addedge(2 \* 0 + 1, 2 \* 0);

Addedge(2 \* 1 + 1, 2 \* 1);

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

if(sta[i] == -1) Addedge(2 \* i, 2 \* i + 1);

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

if(d[i][j] == false){

if((sta[i] == 0 && sta[j] == 0) || (sta[i] == 1 && sta[j] == 1)){

// cout << i << ' ' << j << endl;

Addedge(2 \* i, 2 \* j + 1);

Addedge(2 \* i + 1, 2 \* j);

Addedge(2 \* j, 2 \* i + 1);

Addedge(2 \* j + 1, 2 \* i);

} else{

Addedge(2 \* i + 1, 2 \* j);

Addedge(2 \* j + 1, 2 \* i);

}

}

}

}

// cout << "!";

if(!\_2Sat(res)) puts("impossible");

else{

vector <int> v[2];

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

if(sta[i] == 0 && res[i]) v[0].PB(i);

if(sta[i] == 1 && res[i]) v[1].PB(i);

}

sort(v[0].begin(), v[0].end());

sort(v[1].begin(), v[1].end());

for(int i = 0; i < v[0].size(); i ++) printf("%d%c", 1 + v[0][i], (i == v[0].size() - 1) ? '\n' : ' ');

for(int i = 0; i < v[1].size(); i ++) printf("%d%c", 1 + v[1][i], (i == v[1].size() - 1) ? '\n' : ' ');

}

}

**矩形周长并（扫描线）**

#pragma comment(linker, "/STACK:36777216")

#include <map>

#include <set>

#include <cmath>

#include <ctime>

#include <queue>

#include <stack>

#include <cstdio>

#include <string>

#include <vector>

#include <iomanip>

#include <cassert>

#include <cstdlib>

#include <cstring>

#include <iostream>

#include <algorithm>

#define lc(x) (x << 1)

#define rc(x) (lc(x) + 1)

#define lowbit(x) (x & (-x))

#define PI (acos(-1))

#define lowbit(x) (x & (-x))

#define LL long long

#define DB double

#define ULL unsigned long long

#define PII pair<int, int>

#define PLL pair<LL, LL>

#define PB push\_back

#define MP make\_pair

using namespace std;

const int N = 5005;

const int M = 50005;

const int MOD = 1e9 + 7;

const int INF = 1e9 + 7;

const DB EPS = 1e-8;

struct Seg{

int l, r, x, op;

Seg(){}

Seg(int \_l, int \_r, int \_x, int \_op){

l = \_l, r = \_r, x = \_x, op = \_op;

}

bool operator < (const Seg &A) const{

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

return op > A.op;

}

};

vector <Seg> seg;

vector <int> lsh;

struct Segtree{

// cnt means the number of times the segment was covered, len means the length of the covered segment

// ct means number of lines, cl & cr means the points of the seg has been covered or not

int l, r, cnt, len, ct;

bool cl, cr;

int mid(){

return l + r >> 1;

}

} tree[N << 2];

void Init\_tree(int l, int r, int x){

tree[x].l = l, tree[x].r = r;

tree[x].cnt = tree[x].len = 0;

tree[x].cl = tree[x].cr = tree[x].ct = 0;

if(l == r) return;

int mid = l + r >> 1;

Init\_tree(l, mid, lc(x));

Init\_tree(mid + 1, r, rc(x));

}

void Update(int x){

if(tree[x].cnt != 0){

tree[x].cl = tree[x].cr = 1, tree[x].ct = 2;

tree[x].len = lsh[tree[x].r + 1] - lsh[tree[x].l];

} else if(tree[x].l != tree[x].r){

tree[x].len = tree[lc(x)].len + tree[rc(x)].len;

tree[x].cl = tree[lc(x)].cl;

tree[x].cr = tree[rc(x)].cr;

tree[x].ct = tree[lc(x)].ct + tree[rc(x)].ct;

if(tree[lc(x)].cr && tree[rc(x)].cl) tree[x].ct -= 2;

} else{

tree[x].cl = tree[x].cr = tree[x].ct = 0;

tree[x].len = 0;

}

}

void Gao(int l, int r, int op, int x){

if(l <= tree[x].l && tree[x].r <= r){

tree[x].cnt += op;

Update(x);

return;

}

int mid = tree[x].mid();

if(l <= mid) Gao(l, r, op, lc(x));

if(r > mid) Gao(l, r, op, rc(x));

Update(x);

}

int n, cas; int a, b, c, d;

int main(){

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

while(scanf("%d", &n) == 1){

if(!n) break;

lsh.clear();

seg.clear();

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

scanf("%d%d%d%d", &a, &b, &c, &d);

seg.PB(Seg(b, d, a, 1));

seg.PB(Seg(b, d, c, -1));

lsh.PB(b), lsh.PB(d);

}

sort(lsh.begin(), lsh.end());

lsh.erase(unique(lsh.begin(), lsh.end()), lsh.end());

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

seg[i].l = lower\_bound(lsh.begin(), lsh.end(), seg[i].l) - lsh.begin();

seg[i].r = lower\_bound(lsh.begin(), lsh.end(), seg[i].r) - lsh.begin() - 1;

}

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

// cout << lsh[i] << ' '; cout << endl;

sort(seg.begin(), seg.end());

Init\_tree(0, lsh.size() - 1, 1);

int tot = 0, pre = 0;

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

Gao(seg[i].l, seg[i].r, seg[i].op, 1);

if(i != seg.size() - 1) tot += tree[1].ct \* (seg[i + 1].x - seg[i].x);

tot += abs(tree[1].len - pre);

pre = tree[1].len;

// cout << seg[i].x << ' ' << seg[i + 1].x << ' ' << tree[1].len << ' ' << tree[1].ct << ' ' << seg[i].l << ' ' << seg[i].r << ' ' <<seg[i].op << endl;

}

printf("%d\n", tot);

}

}

**求模 N 的斐波那契循环节**

#include <iostream>

#include <string.h>

#include <algorithm>

#include <stdio.h>

#include <math.h>

using namespace std;

typedef unsigned long long LL;

const int M = 2;

struct Matrix

{

LL m[M][M];

};

Matrix A;

Matrix I = {1,0,0,1};

Matrix multi(Matrix a,Matrix b,LL MOD)

{

Matrix c;

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

{

for(int j=0; j<M; j++)

{

c.m[i][j] = 0;

for(int k=0; k<M; k++)

c.m[i][j] = (c.m[i][j]%MOD + (a.m[i][k]%MOD)\*(b.m[k][j]%MOD)%MOD)%MOD;

c.m[i][j] %= MOD;

}

}

return c;

}

Matrix power(Matrix a,LL k,LL MOD)

{

Matrix ans = I,p = a;

while(k)

{

if(k & 1)

{

ans = multi(ans,p,MOD);

k--;

}

k >>= 1;

p = multi(p,p,MOD);

}

return ans;

}

LL gcd(LL a,LL b)

{

return b? gcd(b,a%b):a;

}

const int N = 400005;

const int NN = 5005;

LL num[NN],pri[NN];

LL fac[NN];

int cnt,c;

bool prime[N];

int p[N];

int k;

void isprime()

{

k = 0;

memset(prime,true,sizeof(prime));

for(int i=2; i<N; i++)

{

if(prime[i])

{

p[k++] = i;

for(int j=i+i; j<N; j+=i)

prime[j] = false;

}

}

}

LL quick\_mod(LL a,LL b,LL m)

{

LL ans = 1;

a %= m;

while(b)

{

if(b & 1)

{

ans = ans \* a % m;

b--;

}

b >>= 1;

a = a \* a % m;

}

return ans;

}

LL legendre(LL a,LL p)

{

if(quick\_mod(a,(p-1)>>1,p)==1) return 1;

else return -1;

}

void Solve(LL n,LL pri[],LL num[])

{

cnt = 0;

LL t = (LL)sqrt(1.0\*n);

for(int i=0; p[i]<=t; i++)

{

if(n%p[i]==0)

{

int a = 0;

pri[cnt] = p[i];

while(n%p[i]==0)

{

a++;

n /= p[i];

}

num[cnt] = a;

cnt++;

}

}

if(n > 1)

{

pri[cnt] = n;

num[cnt] = 1;

cnt++;

}

}

void Work(LL n)

{

c = 0;

LL t = (LL)sqrt(1.0\*n);

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

{

if(n % i == 0)

{

if(i \* i == n) fac[c++] = i;

else

{

fac[c++] = i;

fac[c++] = n / i;

}

}

}

}

LL find\_loop(LL n)

{

Solve(n,pri,num);

LL ans=1;

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

{

LL record=1;

if(pri[i]==2)

record=3;

else if(pri[i]==3)

record=8;

else if(pri[i]==5)

record=20;

else

{

if(legendre(5,pri[i])==1)

Work(pri[i]-1);

else

Work(2\*(pri[i]+1));

sort(fac,fac+c);

for(int k=0; k<c; k++)

{

Matrix a = power(A,fac[k]-1,pri[i]);

LL x = (a.m[0][0]%pri[i]+a.m[0][1]%pri[i])%pri[i];

LL y = (a.m[1][0]%pri[i]+a.m[1][1]%pri[i])%pri[i];

if(x==1 && y==0)

{

record = fac[k];

break;

}

}

}

for(int k=1; k<num[i]; k++)

record \*= pri[i];

ans = ans/gcd(ans,record)\*record;

}

return ans;

}

void Init()

{

A.m[0][0] = 1;

A.m[0][1] = 1;

A.m[1][0] = 1;

A.m[1][1] = 0;

}

int main()

{

LL n;

Init();

isprime();

while(cin>>n)

cout<<find\_loop(n)<<endl;

return 0;}

**Query On The Subtree 叉姐版非递归树分治**

#include <algorithm>

#include <cassert>

#include <cstdio>

#include <cstring>

#include <climits>

#include <map>

#include <queue>

#include <vector>

const int N = 100000;

const int D = 40;

int n, q, queue[N], parent[N], size[N], balance[N], depth[N], weight[N];

bool visited[N];

std::vector <int> tree[N];

struct Entry {

Entry(int ancestor, int distance, int subtree) : ancestor(ancestor), distance(distance), subtree(subtree) {}

int ancestor, distance, subtree;

};

std::vector <Entry> entries[N];

struct BinaryIndexedTree {

void allocate(int size) {

n = size;

count = pool + pool\_size;

std::fill(count, count + n, 0);

pool\_size += size;

}

void add(int k, int v) {

for (; k < n; k += ~k & k + 1) {

count[k] += v;

}

}

int query(int k) {

k = std::min(k, n - 1);

int result = 0;

for (; k >= 0; k -= ~k & k + 1) {

result += count[k];

}

return result;

}

int n;

int\* count;

static int pool\_size;

static int pool[N \* D];

};

int BinaryIndexedTree::pool\_size;

int BinaryIndexedTree::pool[N \* D];

std::map <int, BinaryIndexedTree> bits[N];

void divide(int root)

{

parent[root] = -1;

size[root] = balance[root] = 0;

int tail = 0;

queue[tail ++] = root;

for (int head = 0; head < tail; ++ head) {

int u = queue[head];

for (int \_ = 0; \_ < (int)tree[u].size(); ++ \_) {

int v = tree[u][\_];

if (v != parent[u] && !visited[v]) {

parent[v] = u;

size[v] = balance[v] = 0;

queue[tail ++] = v;

}

}

}

for (int i = tail - 1; i >= 1; -- i) {

int u = queue[i];

size[u] ++;

size[parent[u]] += size[u];

balance[parent[u]] = std::max(balance[parent[u]], size[u]);

}

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

int u = queue[i];

balance[u] = std::max(balance[u], size[root] - size[u]);

}

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

int u = queue[i];

if (balance[u] < balance[root]) {

root = u;

}

}

bits[root][-1].allocate(tail);

bits[root][-1].add(0, weight[root]);

visited[root] = true;

entries[root].push\_back(Entry(root, 0, -1));

for (int \_ = 0; \_ < (int)tree[root].size(); ++ \_) {

int vv = tree[root][\_];

if (!visited[vv]) {

tail = 0;

queue[tail ++] = vv;

parent[vv] = root;

depth[vv] = 1;

for (int head = 0; head < tail; ++ head) {

int u = queue[head];

entries[u].push\_back(Entry(root, depth[u], vv));

for (int \_\_ = 0; \_\_ < (int)tree[u].size(); ++ \_\_) {

int v = tree[u][\_\_];

if (v != parent[u] && !visited[v]) {

parent[v] = u;

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

queue[tail ++] = v;

}

}

}

bits[root][vv].allocate(tail + 1);

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

int u = queue[i];

bits[root][-1].add(depth[u], weight[u]);

bits[root][vv].add(depth[u], weight[u]);

}

divide(vv);

}

}

}

int main()

{

while (scanf("%d%d", &n, &q) == 2) {

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

scanf("%d", weight + i);

}

std::fill(tree, tree + n, std::vector <int>());

for (int i = 0; i < n - 1; ++ i) {

int a, b;

scanf("%d%d", &a, &b);

a --;

b --;

tree[a].push\_back(b);

tree[b].push\_back(a);

}

memset(visited, 0, sizeof(visited));

std::fill(entries, entries + n, std::vector <Entry>());

BinaryIndexedTree::pool\_size = 0;

std::fill(bits, bits + n, std::map <int, BinaryIndexedTree>());

divide(0);

while (q --) {

int u, d;

char buffer[2];

scanf("%s%d%d", buffer, &u, &d);

u --;

if (\*buffer == '!') {

d -= weight[u];

for (int \_ = 0; \_ < (int)entries[u].size(); ++ \_) {

const Entry &entry = entries[u][\_];

bits[entry.ancestor][-1].add(entry.distance, d);

if (~entry.subtree) {

bits[entry.ancestor][entry.subtree].add(entry.distance, d);

}

}

weight[u] += d;

} else {

int result = 0;

for (int \_ = 0; \_ < (int)entries[u].size(); ++ \_) {

const Entry &entry = entries[u][\_];

if (entry.distance <= d) {

result += bits[entry.ancestor][-1].query(d - entry.distance);

if (~entry.subtree) {

result -= bits[entry.ancestor][entry.subtree].query(d - entry.distance);

}

}

}

printf("%d\n", result);

}

}

}

return 0;

}