***STRING***

***KMP***

#include <stdio.h>

#include <string.h>

const int MAX = 1e6 + 10;

int n, m, f[MAX];

char s[MAX], t[MAX];

void constructFSM() {

f[0] = -1;

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

int j = f[i - 1];

while (j >= 0 && t[j] != t[i - 1])

j = f[j];

f[i] = j + 1;

}

}

int main() {

//scanf("%s %s", s, t);

strcpy(s, "aaababa");

strcpy(t, "aba");

n = strlen(s);

m = strlen(t);

constructFSM();

int j = 0;

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

while (j >= 0 && t[j] != s[i - 1])

j = f[j];

++j;

if (j == m) {

printf("%d ", i - 1);

j = f[j];

}

}

return 0;

}

***AHO CORASICK***

#include <iostream>

#include <algorithm>

#include <string>

#include <queue>

using namespace std;

#include <memory.h>

int n, m, res, pos, u, v;

string s, kt;

struct Aho\_Corasick {

int cnt, label;

Aho\_Corasick \*child[26], \*par, \*link;

Aho\_Corasick() : cnt(0), label(-1), par(NULL), link(NULL) {

memset(child, NULL, sizeof(child));

}

};

Aho\_Corasick \*root = new Aho\_Corasick;

void addString(const string& s) {

Aho\_Corasick \*cur = root;

for (int i = 0; i < s.length(); ++i) {

if (cur->child[s[i] - 'a'] == NULL) {

(cur->child[s[i] - 'a'] = new Aho\_Corasick)->par = cur;

cur->child[s[i] - 'a']->label = s[i] - 'a';

}

cur = cur->child[s[i] - 'a'];

}

++cur->cnt;

}

void constructFSM() {

queue <Aho\_Corasick\*> que;

que.push(root);

while (!que.empty()) {

Aho\_Corasick \*u = que.front();

que.pop();

if (u != root) {

Aho\_Corasick \*cur = u->par->link;

while (cur != NULL && cur->child[u->label] == NULL)

cur = cur->link;

if (cur == NULL)

u->link = root;

else

u->link = cur->child[u->label];

u->cnt += u->link->cnt;

}

for (int i = 0; i < 26; ++i)

if (u->child[i] != NULL)

que.push(u->child[i]);

}

}

int get(const string& s, const int& lef, const int& rig) {

int res = 0;

Aho\_Corasick \*cur = root;

for (int i = lef; i < rig; ++i) {

while (cur != NULL && cur->child[s[i] - 'a'] == NULL)

cur = cur->link;

if (cur == NULL)

cur = root;

else

cur = cur->child[s[i] - 'a'];

res += cur->cnt;

}

return res;

}

void erase(Aho\_Corasick \*&root) {

if (root == NULL)

return;

for (int i = 0; i < 26; ++i)

if (root->child[i] != NULL)

erase(root->child[i]);

delete root;

root = NULL;

}

int main() {

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

cin >> n >> m;

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

cin >> s;

addString(s);

}

constructFSM();

cin >> s;

res = get(s, 0, s.length());

cout << res << endl;

while (m--) {

cin >> pos >> kt;

--pos;

u = max(0, pos - 100);

v = min(pos + 100, (int)s.length());

res -= get(s, u, v);

s[pos] = kt[0];

res += get(s, u, v);

cout << res << endl;

}

erase(root);

return 0;

}

***MANACHER***

#include <stdio.h>

#include <string.h>

#define min(a, b) (a < b ? a : b)

#define max(a, b) (a > b ? a : b)

const int MAX = 1e6 + 10;

int n, pal[MAX \* 2];

char s[MAX];

void constructManacher() {

int m = n << 1;

for (int i = 0, j = 0, k;

i < m;

i += k, j = max(j - k, 0)) {

for (;

i >= j && i + j + 1 < m &&

s[(i - j) / 2] == s[(i + j + 1) / 2];

++j);

pal[i] = j;

for (k = 1;

i >= k &&

pal[i] >= k &&

pal[i - k] != pal[i] - k;

++k)

pal[i + k] = min(pal[i - k], pal[i] - k);

}

}

int main() {

//scanf("%s", s);

strcpy(s, "abaabaaba");

n = strlen(s);

constructManacher();

//print length odd

for (int i = 0; i < (n << 1); i += 2)

printf("%d ", pal[i]);

printf("\n");

//print length even

for (int i = 1; i < (n << 1); i += 2)

printf("%d ", pal[i]);

return 0;

}

***SUFFIX ARRAY***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <algorithm>

using namespace std;

//Output:

// SA = The suffix array. Contains the n suffixes of txt sorted in lexicographical order.

// Each suffix is represented as a single integer (the SAition of txt where it starts).

// iSA = The inverse of the suffix array. iSA[i] = the index of the suffix txt[i..n)

// in the SA array. (In other words, SA[i] = k <==> iSA[k] = i)

// With this array, you can compare two suffixes in O(1): Suffix txt[i..n) is smaller

// than txt[j..n) if and only if iSA[i] < iSA[j]

const int MAX = 100010;

char txt[MAX]; //input

int iSA[MAX], SA[MAX]; //output

int cnt[MAX], ne[MAX]; //internal

bool bh[MAX], b2h[MAX];

bool smaller\_first\_char(int a, int b) {

return txt[a] < txt[b];

}

void suffixSort(int n) {

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

SA[i] = i;

sort(SA, SA + n, smaller\_first\_char);

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

bh[i] = i == 0 || txt[SA[i]] != txt[SA[i - 1]];

b2h[i] = false;

}

for (int h = 1; h < n; h <<= 1) {

int buckets = 0;

for (int i = 0, j; i < n; i = j) {

j = i + 1;

while (j < n && !bh[j]) j++;

ne[i] = j;

buckets++;

}

if (buckets == n) break;

for (int i = 0; i < n; i = ne[i]) {

cnt[i] = 0;

for (int j = i; j < ne[i]; ++j)

iSA[SA[j]] = i;

}

cnt[iSA[n - h]]++;

b2h[iSA[n - h]] = true;

for (int i = 0; i < n; i = ne[i]) {

for (int j = i; j < ne[i]; ++j) {

int s = SA[j] - h;

if (s >= 0) {

int head = iSA[s];

iSA[s] = head + cnt[head]++;

b2h[iSA[s]] = true;

}

}

for (int j = i; j < ne[i]; ++j) {

int s = SA[j] - h;

if (s >= 0 && b2h[iSA[s]])

for (int k = iSA[s] + 1; !bh[k] && b2h[k]; k++)

b2h[k] = false;

}

}

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

SA[iSA[i]] = i;

bh[i] |= b2h[i];

}

}

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

iSA[SA[i]] = i;

}

int lcp[MAX];

// lcp[i] = length of the longest common prefix of suffix SA[i] and suffix SA[i-1]

// lcp[0] = 0

void getlcp(int n)

{

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

iSA[SA[i]] = i;

lcp[0] = 0;

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

if (iSA[i] > 0){

int j = SA[iSA[i] - 1];

while (i + h < n && j + h < n && txt[i + h] == txt[j + h])

h++;

lcp[iSA[i]] = h;

if (h > 0)

h--;

}

}

int main(){

int len;

scanf("%s", txt);

len = strlen(txt);

suffixSort(len);

getlcp(len);

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

{

printf("%d %d\n", SA[i], lcp[i]);

printf("%s\n", txt + SA[i]);

}

return 0;

}

***GRAPH***

//O(E)

***HOPCROFT–KARP***

#include <algorithm>

#include <iostream>

using namespace std;

const int MAXN1 = 50000;

const int MAXN2 = 50000;

const int MAXM = 150000;

int n1, n2, edges, last[MAXN1], pre[MAXM], head[MAXM];

int matching[MAXN2], dist[MAXN1], Q[MAXN1];

bool used[MAXN1], vis[MAXN1];

void init(int \_n1, int \_n2) {

n1 = \_n1;

n2 = \_n2;

edges = 0;

fill(last, last + n1, -1);

}

void addEdge(int u, int v) {

head[edges] = v;

pre[edges] = last[u];

last[u] = edges++;

}

void bfs() {

fill(dist, dist + n1, -1);

int sizeQ = 0;

for (int u = 0; u < n1; ++u) {

if (!used[u]) {

Q[sizeQ++] = u;

dist[u] = 0;

}

}

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

int u1 = Q[i];

for (int e = last[u1]; e >= 0; e = pre[e]) {

int u2 = matching[head[e]];

if (u2 >= 0 && dist[u2] < 0) {

dist[u2] = dist[u1] + 1;

Q[sizeQ++] = u2;

}

}

}

}

bool dfs(int u1) {

vis[u1] = true;

for (int e = last[u1]; e >= 0; e = pre[e]) {

int v = head[e];

int u2 = matching[v];

if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)) {

matching[v] = u1;

used[u1] = true;

return true;

}

}

return false;

}

int maxMatching() {

fill(used, used + n1, false);

fill(matching, matching + n2, -1);

for (int res = 0;;) {

bfs();

fill(vis, vis + n1, false);

int f = 0;

for (int u = 0; u < n1; ++u)

if (!used[u] && dfs(u))

++f;

if (!f)

return res;

res += f;

}

}

int main() {

init(2, 2);

addEdge(0, 0);

addEdge(0, 1);

addEdge(1, 1);

cout << (2 == maxMatching()) << endl;

}

***HUNGRARIAN***

#include <algorithm>

#include <vector>

using namespace std;

struct Hungary {

const int oo = 1e9;

int n, m, \*\*cost, \*fx, \*fy, \*matx, \*which, \*dis;

bool \*used;

Hungary(int n, int m) : n(n), m(m) {

which = new int[m + 1];

dis = new int[m + 1];

fx = new int[n + 1];

memset(fx, 0, (n + 1) \* sizeof(int));

fy = new int[m + 1];

memset(fy, 0, (m + 1) \* sizeof(int));

used = new bool[m + 1];

memset(used, false, (m + 1) \* sizeof(bool));

matx = new int[m + 1];

memset(matx, 0, (m + 1) \* sizeof(int));

cost = new int\*[n + 1];

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

cost[i] = new int[m + 1];

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

cost[i][j] = oo;

}

}

void add(int x, int y, int c){

cost[x][y] = min(cost[x][y], c);

}

int mincost() {

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

int u = 0; matx[0] = x;

for (int y = 0; y <= m; ++y) {

dis[y] = oo + 1;

used[y] = false;

}

do {

used[u] = true;

int x0 = matx[u], delta = oo + 1, v;

for (int y = 1; y <= m; ++y)

if (!used[y]) {

int curdis = cost[x0][y] - fx[x0] - fy[y];

if (curdis < dis[y]) {

dis[y] = curdis;

which[y] = u;

}

if (dis[y] < delta) {

delta = dis[y];

v = y;

}

}

for (int y = 0; y <= m; ++y)

if (used[y]) {

fx[matx[y]] += delta;

fy[y] -= delta;

}

else

dis[y] -= delta;

u = v;

} while (matx[u] != 0);

do {

int v = which[u];

matx[u] = matx[v];

u = v;

} while (u);

}

int res = 0;

for (int y = 1; y <= m; ++y)

if (cost[matx[y]][y] < oo)

res += cost[matx[y]][y];

return res;

}

};

***MINCOST MATCHING***

//////////////////////////////////////////////////////////////////////

// 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.

//////////////////////////////////////////////////////////////////////

#include <algorithm>

#include <cstdio>

#include <cmath>

#include <vector>

using namespace std;

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;

}

***GENERAL MATCH***

#include <bits/stdc++.h>

using namespace std;

#define maxn 411

bool a[maxn][maxn];

int match[maxn], b[maxn], T[maxn];

bool inQ[maxn], inPath[maxn], mark[maxn];

int n, start, finish;

queue<int> q;

void init() { memset(match, 0, sizeof(match)); }

void initBFS() {

while (!q.empty()) q.pop();

q.push(start);

memset(inQ, 0, sizeof(inQ));

inQ[start] = true;

memset(T, 0, sizeof(T));

for (int i = 1; i <= n; ++i) b[i] = i;

finish = 0;

}

void push(int v) { q.push(v); inQ[v] = true; }

int LCA(int u, int v) {

memset(inPath, 0, sizeof(inPath));

while (true) {

u = b[u];

inPath[u] = true;

if (u == start) break;

u = T[match[u]];

}

while (true) {

v = b[v];

if (inPath[v]) break;

v = T[match[v]];

}

return v;

}

void resetTrace(int x, int lca) {

int v = x;

while (b[v] != lca) {

int u = match[v];

mark[b[v]] = mark[b[u]] = true;

v = T[u];

if (b[v] != lca) T[v] = u;

}

}

void shrink(int u, int v) {

memset(mark, 0, sizeof(mark));

int lca = LCA(u, v);

resetTrace(u, lca); resetTrace(v, lca);

if (b[u] != lca) T[u] = v;

if (b[v] != lca) T[v] = u;

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

if (mark[b[i]]) b[i] = lca;

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

if (!inQ[i] && b[i] == lca) push(i);

}

void findPath() {

initBFS();

while (!q.empty()) {

int u = q.front(); q.pop();

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

if (T[v] == 0 && a[u][v] && b[u] != b[v]) {

if (match[v] == 0){

T[v] = u;

finish = v;

return;

}

if (v == start ||

T[match[v]] != 0) shrink(u, v);

else {

T[v] = u;

push(match[v]);

}

}

}

}

void enlarge() {

do {

int v = T[finish];

int next = match[v];

match[v] = finish;

match[finish] = v;

finish = next;

} while (finish);

}

int solve() {

memset(match, 0, sizeof(match));

memset(b, 0, sizeof(b));

memset(T, 0, sizeof(T));

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

if (match[u] == 0) {

start = u;

findPath();

if (finish) enlarge();

}

//

int ret = 0;

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

if (match[u] > u)

++ret; //cout << u - match[u]

return ret;

}

***MAX FLOW PUSH RELABEL***

// Adjacency list implementation of FIFO push relabel maximum flow

// with the gap relabeling heuristic. This implementation is

// significantly faster than straight Ford-Fulkerson. It solves

// random problems with 10000 vertices and 1000000 edges in a few

// seconds, though it is possible to construct test cases that

// achieve the worst-case.

//

// Running time:

// O(|V|^3)

//

// INPUT:

// - graph, constructed using AddEdge()

// - source

// - sink

//

// OUTPUT:

// - maximum flow value

// - To obtain the actual flow values, look at all edges with

// capacity > 0 (zero capacity edges are residual edges).

#include <cmath>

#include <vector>

#include <iostream>

#include <queue>

using namespace std;

typedef long long LL;

struct Edge {

int from, to, cap, flow, index;

Edge(int from, int to, int cap, int flow, int index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

struct PushRelabel {

int N;

vector<vector<Edge> > G;

vector<LL> excess;

vector<int> dist, active, count;

queue<int> Q;

PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N), count(2 \* N) {}

void AddEdge(int from, int to, int cap) {

G[from].push\_back(Edge(from, to, cap, 0, G[to].size()));

if (from == to) G[from].back().index++;

G[to].push\_back(Edge(to, from, 0, 0, G[from].size() - 1));

}

void Enqueue(int v) {

if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }

}

void Push(Edge &e) {

int amt = int(min(excess[e.from], LL(e.cap - e.flow)));

if (dist[e.from] <= dist[e.to] || amt == 0) return;

e.flow += amt;

G[e.to][e.index].flow -= amt;

excess[e.to] += amt;

excess[e.from] -= amt;

Enqueue(e.to);

}

void Gap(int k) {

for (int v = 0; v < N; v++) {

if (dist[v] < k) continue;

count[dist[v]]--;

dist[v] = max(dist[v], N + 1);

count[dist[v]]++;

Enqueue(v);

}

}

void Relabel(int v) {

count[dist[v]]--;

dist[v] = 2 \* N;

for (int i = 0; i < G[v].size(); i++)

if (G[v][i].cap - G[v][i].flow > 0)

dist[v] = min(dist[v], dist[G[v][i].to] + 1);

count[dist[v]]++;

Enqueue(v);

}

void Discharge(int v) {

for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);

if (excess[v] > 0) {

if (count[dist[v]] == 1)

Gap(dist[v]);

else

Relabel(v);

}

}

LL GetMaxFlow(int s, int t) {

count[0] = N - 1;

count[N] = 1;

dist[s] = N;

active[s] = active[t] = true;

for (int i = 0; i < G[s].size(); i++) {

excess[s] += G[s][i].cap;

Push(G[s][i]);

}

while (!Q.empty()) {

int v = Q.front();

Q.pop();

active[v] = false;

Discharge(v);

}

LL totflow = 0;

for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;

return totflow;

}

};

***MINCOST MAXFLOW (SUPPORT NEGATIVE EDGE)***

#include <bits/stdc++.h>

#define fi(a,b,c) for(int a=b; a<=c; a++)

#define fd(a,b,c) for(int a=b; a>=c; a--)

#define fii(a,b,c) for(int a=b; a<c; a++)

#define pb push\_back

#define mp make\_pair

#define ft first

#define sc second

#define reset(a,b) memset(a,b, sizeof a)

using namespace std;

typedef long long ll;

typedef pair <int, int> pii;

typedef unsigned int ui;

// MAX EDGE NUMBER (M)

const int MAXM = 70000;

// MAX NODE NUMBER (N)

const int MAXN = 600;

// INFINITE VALUE

const int inf = 0x3fffffff;

int L, N; int K;

struct edges{

int to, next, cap, flow, cost;

} edge[MAXM];

struct nodes{

int dis, pre, head;

bool visit;

} node[MAXN];

void init(int n){

N = n;

L = 0;

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

node[i].head = -1;

}

void add\_edge(int x, int y, int cap, int cost){

edge[L].to = y;

edge[L].cap = cap;

edge[L].cost = cost;

edge[L].flow = 0;

edge[L].next = node[x].head;

node[x].head = L++;

edge[L].to = x;

edge[L].cap = 0;

edge[L].cost = -cost;

edge[L].flow = 0;

edge[L].next = node[y].head;

node[y].head = L++;

}

bool spfa(int s, int t){

queue <int> q;

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

node[i].dis = 0x3fffffff;

node[i].pre = -1;

node[i].visit = 0;

}

node[s].dis = 0;

node[s].visit = 1;

q.push(s);

while (!q.empty()){

int u = q.front();

node[u].visit = 0;

for (int i = node[u].head; i != -1; i = edge[i].next){

int v = edge[i].to;

if (edge[i].cap>edge[i].flow &&

node[v].dis>node[u].dis + edge[i].cost){

node[v].dis = node[u].dis + edge[i].cost;

node[v].pre = i;

if (!node[v].visit){

node[v].visit = 1;

q.push(v);

}

}

}

q.pop();

}

if (node[t].pre == -1)

return 0;

else

return 1;

}

int mcmf(int s, int t, int &cost){

int flow = 0;

while (spfa(s, t)){

int max = inf;

for (int i = node[t].pre; i != -1; i = node[edge[i ^ 1].to].pre)

if (max>edge[i].cap - edge[i].flow)

max = edge[i].cap - edge[i].flow;

for (int i = node[t].pre; i != -1; i = node[edge[i ^ 1].to].pre){

edge[i].flow += max;

edge[i ^ 1].flow -= max;

cost += edge[i].cost\*max;

}

flow += max;

}

return flow;

}

string s, p;

int x, n, m;

int main() {

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

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

cin >> n >> s;

cin >> m;

init(n + 3);

fi(k, 1, m) {

cin >> p >> x;

int np = p.length();

fii(i, 0, n) {

bool ok = 1;

fii(j, 0, np) if (s[i + j] != p[j]) {

ok = 0;

break;

}

if (ok) add\_edge(i + 1, i + np + 1, 1, -x);

}

}

cin >> x;

add\_edge(0, 1, x, 0);

add\_edge(n, n + 1, x, 0);

fi(i, 1, n - 1) add\_edge(i, i + 1, x, 0);

int cost = 0;

mcmf(0, n + 1, cost);

cout << -cost;

return 0;

}

***HEAVY LIGHT***

//http://codeforces.com/contest/696/problem/E

#include <stdio.h>

#include <algorithm>

#include <vector>

using namespace std;

typedef long long ll;

typedef pair <ll, int> pli;

const ll oo = 1e18;

const int MAX = 1e5 + 1;

int n, m, q, u, v, k, type, par[MAX], dep[MAX], sz[MAX], head[MAX], tail[MAX], ind[MAX], obj[MAX], nodeRes;

pli it[MAX \* 5];

ll add[MAX \* 5];

vector <int> adj[MAX], w[MAX], res;

void getSz(const int& u, const int& par = 0) {

::par[u] = par;

sz[u] = 1;

for (vector <int>::iterator it = adj[u].begin(); it != adj[u].end(); ++it)

if (\*it != par) {

dep[\*it] = dep[u] + 1;

getSz(\*it, u);

sz[u] += sz[\*it];

}

}

void buildHVL(const int& u, const int& head, const int& par = 0) {

static int cnt = 0;

obj[ind[u] = ++cnt] = u;

::head[u] = head;

int nextPoint = 0;

for (vector <int>::iterator it = adj[u].begin(); it != adj[u].end(); ++it)

if (\*it != par && sz[nextPoint] < sz[\*it])

nextPoint = \*it;

if (nextPoint != 0)

buildHVL(nextPoint, head, u);

for (vector <int>::iterator it = adj[u].begin(); it != adj[u].end(); ++it)

if (\*it != par && \*it != nextPoint)

buildHVL(\*it, \*it, u);

tail[u] = cnt;

}

int LCA(int u, int v) {

while (head[u] != head[v])

if (dep[head[v]] > dep[head[u]])

v = par[head[v]];

else

u = par[head[u]];

return (dep[u] > dep[v] ? v : u);

}

void initIT(const int& lef = 1, const int& rig = n, const int& k = 1) {

if (lef == rig) {

it[k] = (w[obj[lef]].empty() ? pli(oo, 0) : pli(w[obj[lef]].back(), obj[lef]));

return;

}

int mid = (lef + rig) >> 1;

initIT(lef, mid, k << 1);

initIT(mid + 1, rig, (k << 1) + 1);

it[k] = min(it[k << 1], it[(k << 1) + 1]);

}

void erase(const int& lef, const int& rig, const int& pos, const int& k = 1) {

if (lef == rig) {

w[obj[lef]].pop\_back();

it[k] = (w[obj[lef]].empty() ? pli(oo, 0) : pli(w[obj[lef]].back(), obj[lef]));

if (it[k].second != 0)

it[k].first += add[k];

return;

}

int mid = (lef + rig) >> 1;

if (pos <= mid)

erase(lef, mid, pos, k << 1);

else

erase(mid + 1, rig, pos, (k << 1) + 1);

it[k] = min(it[k << 1], it[(k << 1) + 1]);

if (it[k].second != 0)

it[k].first += add[k];

}

pli getIT(const int& lef, const int& rig, const int& u, const int& v, const int& k = 1) {

if (lef == u && rig == v)

return it[k];

int mid = (lef + rig) >> 1;

pli res;

if (v <= mid)

res = getIT(lef, mid, u, v, k << 1);

else

if (u > mid)

res = getIT(mid + 1, rig, u, v, (k << 1) + 1);

else

res = min(getIT(lef, mid, u, mid, k << 1), getIT(mid + 1, rig, mid + 1, v, (k << 1) + 1));

if (res.second != 0)

res.first += add[k];

return res;

}

ll getRes(int u, int v) {

pli res(oo, 0);

int parCommon = LCA(u, v);

while (head[u] != head[parCommon]) {

res = min(res, getIT(1, n, ind[head[u]], ind[u]));

u = par[head[u]];

}

if (ind[u] >= ind[parCommon])

res = min(res, getIT(1, n, ind[parCommon], ind[u]));

while (head[v] != head[parCommon]) {

res = min(res, getIT(1, n, ind[head[v]], ind[v]));

v = par[head[v]];

}

if (ind[v] >= ind[parCommon])

res = min(res, getIT(1, n, ind[parCommon], ind[v]));

if (res.second != 0) {

int pos = w[res.second].back();

erase(1, n, ind[res.second]);

return pos;

}

return -1;

}

void upIT(const int& lef, const int& rig, const int& u, const int& v, const int& k = 1) {

if (lef == u && rig == v) {

it[k].first += ::k;

add[k] += ::k;

return;

}

int mid = (lef + rig) >> 1;

if (v <= mid)

upIT(lef, mid, u, v, k << 1);

else

if (u > mid)

upIT(mid + 1, rig, u, v, (k << 1) + 1);

else {

upIT(lef, mid, u, mid, k << 1);

upIT(mid + 1, rig, mid + 1, v, (k << 1) + 1);

}

it[k] = min(it[k << 1], it[(k << 1) + 1]);

it[k].first += add[k];

}

int main() {

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

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

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

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

adj[u].push\_back(v);

adj[v].push\_back(u);

}

getSz(1);

buildHVL(1, 1);

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

scanf("%d", &u);

w[u].push\_back(i);

}

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

sort(w[i].begin(), w[i].end());

reverse(w[i].begin(), w[i].end());

}

initIT();

while (q--) {

scanf("%d", &type);

if (type == 1) {

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

res.clear();

while (k-- && (nodeRes = getRes(u, v)) != -1)

res.push\_back(nodeRes);

printf("%d", res.size());

for (vector <int>::iterator it = res.begin(); it != res.end(); ++it)

printf(" %d", \*it);

printf("\n");

}

else {

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

upIT(1, n, ind[u], tail[u]);

}

}

return 0;

}

***TWOSAT***

#include <stdio.h>

#include <algorithm>

#include <vector>

#include <stack>

#include <memory.h>

using namespace std;

const int MAX = (8000 << 1) + 1;

int n, nn, m, u, v, cnt, low[MAX], num[MAX],

cntCpn, cpn[MAX], cache[MAX], cntCache, res[MAX];

bool fr[MAX], avail[MAX];

vector <int> adj[MAX], adjTmp[MAX],

adjDag[2][MAX], detail[MAX];

stack <int> st;

int negative(const int& u) {

return (u > nn ? u - nn : u + nn);

}

void visit1(const int& u, const int& dad) {

st.push(u);

low[u] = num[u] = ++cnt;

avail[u] = false;

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

if (fr[adj[u][i]]) {

if (avail[adj[u][i]]) {

visit1(adj[u][i], u);

low[u] = min(low[u], low[adj[u][i]]);

}

else

low[u] = min(low[u], num[adj[u][i]]);

}

else

if (cpn[adj[u][i]])

adjTmp[u].push\_back(cpn[adj[u][i]]);

if (low[u] >= num[u]) {

++cntCpn;

if (dad)

adjTmp[dad].push\_back(cntCpn);

do {

cpn[v = st.top()] = cntCpn;

fr[v] = false;

detail[cntCpn].push\_back(v);

for (int i = 0; i < adjTmp[v].size(); ++i)

adjDag[0][cntCpn].push\_back(adjTmp[v][i]);

st.pop();

} while (v != u);

}

}

void visit2(const int& u) {

fr[u] = false;

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

if (fr[adjDag[0][u][i]])

visit2(adjDag[0][u][i]);

cache[cntCache++] = u;

}

int main() {

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

nn = n;

memset(fr + 1, false, n <<= 1);

memset(avail + 1, false, n);

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

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

if (u < 0)

u = negative(abs(u));

if (v < 0)

v = negative(abs(v));

avail[u] = fr[u] = true;

avail[negative(u)] = fr[negative(u)] = true;

avail[v] = fr[v] = true;

avail[negative(v)] = fr[negative(v)] = true;

adj[negative(u)].push\_back(v);

adj[negative(v)].push\_back(u);

}

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

if (fr[i])

visit1(i, 0);

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

if (cpn[i] && cpn[i] == cpn[negative(i)]) {

printf("NO");

return 0;

}

memset(fr + 1, true, cntCpn);

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

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

adjDag[1][adjDag[0][i][j]].push\_back(i);

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

if (adjDag[1][i].size() == 0)

visit2(i);

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

if (res[u = cache[i]] == 0) {

res[u] = 1;

for (int j = 0; j < detail[u].size(); ++j)

res[cpn[negative(detail[u][j])]] = -1;

}

else

for (int j = 0; j < adjDag[1][u].size(); ++j)

res[adjDag[1][u][j]] = -1;

m = 0;

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

if (res[cpn[i]] == 1)

++m;

printf("YES\n%d\n", m);

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

if (res[cpn[i]] == 1)

printf("%d ", i);

return 0;

}

***MDST***

#include <bits/stdc++.h>

#define fi(a,b,c) for(int a=b; a<=c; a++)

#define fd(a,b,c) for(int a=b; a>=c; a--)

#define reset(a, b) memset(a, b, sizeof a)

using namespace std;

const int NMAX = 1000;

const int oo = 100000;

bool dd[NMAX]; // danh dau nhung vi tri da xet

bool pass[NMAX]; // danh dau lai kiem tra chu trinh

int pa[NMAX]; // canh pa[u] -> u la canh nho nhat toi u

int hd[NMAX];

int n, m, g[NMAX][NMAX], top;

bool more; // kiem tra ton tai duong di co chu trinh de thay doi duong di ,

//neu ko co chu trinh thi duong di day thuoc cay khung

int sum; // gia tri cay khung

void combine(int id){

int top = 0; int from, i, j, k;

for (; id != 0 && !pass[id]; id = pa[id]){

hd[top++] = id; // id = pa[id] ->>> neu pass[id ] == 1 -> ton tai chu trinh

pass[id] = 1;

}

cout << "hd ";

fi(kk, 0, top - 1) cout << hd[kk] << ' ';

cout << endl;

for (from = 0; from < top && hd[from] != id; from++);

cout << "from" << from << endl;

if (from == top) return; // id = 0 , ko ton tai bat ki chu trinh nao

more = 1; // ton tai chu trinh

cout << "kiem tra" << id << endl;

for (i = from; i< top; i++){

sum += g[pa[hd[i]]][hd[i]]; // neu no nqam trong chu trinh

cout << "add sum" << ' ' << i << ' ' << sum << endl;

if (i != from){

dd[hd[i]] = 1;

for (j = 1; j <= n; j++)

if (!dd[j])

if (g[id][j] > g[hd[i]][j])

g[id][j] = g[hd[i]][j];

}

}

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

if (!dd[i] && i != id){

for (j = from; j< top; j++){ // j da duoc dd => j luon luon khac i

k = hd[j];

if (g[i][id] > g[i][k] - g[pa[k]][k]) // luong duong vi g[pa[k]][k] laf canh toi k nho nhat

g[i][id] = g[i][k] - g[pa[k]][k]; // cap nhat lai gia tri canh

// day la buoc toi uu hoa

}

}

}

void MDST(int root){

sum = 0;

more = 1;

while (more){

more = 0;

reset(pa, 0);

fi(i, 1, n)

if (i != root && !dd[i]){

int k = 0; g[k][i] = oo;

fi(j, 1, n)

if (!dd[k] && i != j)

if (g[j][i] < g[k][i])

k = j;

pa[i] = k; // xac dinh canh ngan nhat toi i

}

fi(i, 1, n) cout << pa[i] << ' ';

cout << endl;

reset(pass, 0);

fi(i, 1, n)

if (i != root && !dd[i] && !pass[i])

combine(i);

}

fi(i, 1, n) if (i != root && !dd[i])

sum += g[pa[i]][i];

}

int main(){

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

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

cin >> n >> m;

fi(i, 1, n){

fi(j, 1, n) g[i][j] = oo;

g[i][i] = 0;

}

fi(i, 1, m){

int u, v, val;

cin >> u >> v >> val;

g[u][v] = val;

}

MDST(1);

cout << sum;

return 0;

}

***DATA STRUCTURE***

***IT (SET SEGMENTS)***

#include <stdio.h>

#include <memory.h>

#include <algorithm>

using namespace std;

typedef long long ll;

const int MAX = 2e5;

int n, m, it[MAX \* 5];

struct NodeData {

ll s, e, v, d;

};

NodeData data[MAX];

ll cal(const int& id, const int& pos) {

return data[id].v +

data[id].d \* (pos - data[id].s);

}

void down(const int& lef, const int& rig,

const int& k, const int& id) {

if (it[k] == -1) {

it[k] = id;

return;

}

if (cal(id, lef) <= cal(it[k], lef) &&

cal(id, rig) <= cal(it[k], rig))

return;

if (cal(id, lef) >= cal(it[k], lef) &&

cal(id, rig) >= cal(it[k], rig)) {

it[k] = id;

return;

}

int mid = (lef + rig) >> 1;

if (cal(id, lef) <= cal(it[k], lef) &&

cal(id, mid) <= cal(it[k], mid))

down(mid + 1, rig, (k << 1) + 1, id);

else

if (cal(id, lef) >= cal(it[k], lef) &&

cal(id, mid) >= cal(it[k], mid)) {

down(mid + 1, rig, (k << 1) + 1, it[k]);

it[k] = id;

}

else

if (cal(id, mid + 1) <= cal(it[k], mid + 1) &&

cal(id, rig) <= cal(it[k], rig))

down(lef, mid, k << 1, id);

else

if (cal(id, mid + 1) >= cal(it[k], mid + 1) &&

cal(id, rig) >= cal(it[k], rig)) {

down(lef, mid, k << 1, it[k]);

it[k] = id;

}

}

void update(const int& lef, const int& rig,

const int& u, const int& v,

const int& k, const int& id) {

if (lef == u && rig == v) {

down(lef, rig, k, id);

return;

}

int mid = (lef + rig) >> 1;

if (v <= mid)

update(lef, mid, u, v, k << 1, id);

else

if (u > mid)

update(mid + 1, rig, u, v, (k << 1) + 1, id);

else {

update(lef, mid, u, mid, k << 1, id);

update(mid + 1, rig, mid + 1, v, (k << 1) + 1, id);

}

}

ll get(const int& lef, const int& rig,

const int& k, const int& pos) {

ll val = (it[k] == -1 ? -1 : cal(it[k], pos));

if (lef == rig)

return val;

int id;

int mid = (lef + rig) >> 1;

if (pos <= mid)

return max(val, get(lef, mid, k << 1, pos));

else

return max(val, get(mid + 1, rig, (k << 1) + 1, pos));

}

int main() {

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

memset(it, -1, n \* 5 \* sizeof(int));

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

scanf("%lld %lld %lld %lld",

&data[i].s, &data[i].e,

&data[i].v, &data[i].d);

update(1, n, data[i].s, data[i].e, 1, i);

}

ll val;

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

val = get(1, n, 1, i);

printf("%lld\n", (val == -1 ? 0 : val));

}

return 0;

}

***AVL***

#include <stdio.h>

#include <algorithm>

using namespace std;

const int oo = 2e9;

class NodeAVL {

int val, mx, h, sz;

NodeAVL \*lef, \*rig;

NodeAVL(const int& val) :

val(val), mx(val), sz(1),

h(1), lef(NULL), rig(NULL) {}

static int getHeight(NodeAVL \*&r) {

return r == NULL ? 0 : r->h;

}

static int getSize(NodeAVL \*&r) {

return r == NULL ? 0 : r->sz;

}

static int getMx(NodeAVL \*&r) {

return r == NULL ? -oo : r->mx;

}

static void update(NodeAVL \*&r) {

if (r == NULL)

return;

r->h = max(getHeight(r->lef),

getHeight(r->rig)

) + 1;

r->sz = getSize(r->lef) +

getSize(r->rig) + 1;

r->mx = max(r->val,

max(getMx(r->lef),

getMx(r->rig)));

}

static void LL(NodeAVL \*&r) {

NodeAVL \*node = r->lef;

r->lef = node->rig;

node->rig = r;

r = node;

update(r->rig);

update(r);

}

static void RR(NodeAVL \*&r) {

NodeAVL \*node = r->rig;

r->rig = node->lef;

node->lef = r;

r = node;

update(r->lef);

update(r);

}

static void LR(NodeAVL \*&r) {

RR(r->lef);

LL(r);

}

static void RL(NodeAVL \*&r) {

LL(r->rig);

RR(r);

}

static void balance(NodeAVL \*&r) {

if (r == NULL)

return;

int bal = getHeight(r->lef) - getHeight(r->rig);

if (bal == 2) {

NodeAVL \*node = r->lef;

if (getHeight(node->lef) -

getHeight(node->rig) == -1)

LR(r);

else

LL(r);

}

if (bal == -2) {

NodeAVL \*node = r->rig;

if (getHeight(node->lef) -

getHeight(node->rig) == 1)

RL(r);

else

RR(r);

}

}

static void change(NodeAVL \*&r, NodeAVL \*& q) {

if (r->rig)

change(r->rig, q);

else {

q->val = r->val;

q = r;

r = r->lef;

}

update(r);

balance(r);

}

public:

static void remove(NodeAVL \*&r, const int& pos) {

if (r == NULL)

return;

int szTmp = getSize(r->lef) + 1;

if (pos < szTmp)

remove(r->lef, pos);

else

if (pos > szTmp)

remove(r->rig, pos - szTmp);

else {

NodeAVL \*q = r;

if (q->lef == NULL)

r = r->rig;

else

if (q->rig == NULL)

r = r->lef;

else

change(r->lef, q);

delete q;

}

update(r);

balance(r);

}

static int getValue(NodeAVL \*&r, const int& pos) {

int szTmp = getSize(r->lef) + 1;

if (pos < szTmp)

return getValue(r->lef, pos);

else

if (pos > szTmp)

return getValue(r->rig, pos - szTmp);

else

return r->val;

}

static void insert(NodeAVL \*&r,

const int& pos, const int& val) {

if (r == NULL)

r = new NodeAVL(val);

else {

int szTmp = getSize(r->lef) + 1;

if (pos <= szTmp)

insert(r->lef, pos, val);

else

insert(r->rig, pos - szTmp, val);

update(r);

balance(r);

}

}

static void update(NodeAVL \*&r,

const int& pos, const int& val) {

if (r == NULL)

return;

int szTmp = getSize(r->lef);

if (pos <= szTmp)

update(r->lef, pos, val);

else

if (pos > szTmp + 1)

update(r->rig, pos - szTmp - 1, val);

else

r->val = val;

update(r);

}

static int getMx(NodeAVL \*&r,

const int& u, const int& v,

const int& lef = 1, int rig = -1) {

if (rig == -1)

rig = getSize(r);

if (r == NULL || u > v)

return -oo;

if (u == lef && v == rig)

return r->mx;

int szTmp = getSize(r->lef) + 1;

if (v < szTmp)

return getMx(r->lef, u, v, lef, szTmp);

if (u > szTmp)

return getMx(r->rig, u - szTmp, v - szTmp, 1, rig - szTmp);

int res = max(

getMx(r->lef, u, szTmp, lef, szTmp),

getMx(r->rig, 1, v - szTmp, 1, rig - szTmp)

);

return max(res, r->val);

}

static void erase(NodeAVL \*&root) {

if (root == NULL)

return;

erase(root->lef);

erase(root->rig);

delete root;

}

};

NodeAVL \*root = NULL;

***TREAP (HAVE REVERSE OPERATION)***

#include <bits/stdc++.h>

using namespace std;

const int oo = 1e9;

class Treap {

int key, sz, val;

bool rev;

Treap(const int& key, const int& val) :

key(key), sz(1), val(val), rev(false) {

child[0] = child[1] = NULL;

}

Treap \*child[2];

static int getSz(Treap \*&root) {

return root == NULL ? 0 : root->sz;

}

static int getKey(Treap \*&root) {

return root == NULL ? oo : root->key;

}

static void update(Treap \*&root) {

if (root == NULL)

return;

root->sz = getSz(root->child[0]) +

getSz(root->child[1]) + 1;

}

static void down(Treap \*&root) {

if (root->rev) {

swap(root->child[0], root->child[1]);

if (root->child[0] != NULL)

root->child[0]->rev ^= 1;

if (root->child[1] != NULL)

root->child[1]->rev ^= 1;

root->rev = false;

}

}

static void rotate(Treap \*&root, const int& dir) {

if (root == NULL)

return;

Treap \*cache = root->child[dir];

down(cache);

root->child[dir] = cache->child[dir ^ 1];

cache->child[dir ^ 1] = root;

update(root);

update(cache);

root = cache;

}

static void erase\_this(Treap \*&root) {

if (root == NULL)

return;

down(root);

if (root->child[0] == NULL &&

root->child[1] == NULL) {

delete root;

root = NULL;

return;

}

int dir = (getKey(root->child[0]) <

getKey(root->child[1]) ? 0 : 1);

rotate(root, dir);

erase\_this(root->child[dir ^ 1]);

update(root);

}

static void split(Treap \*root,

Treap \*&A, Treap \*&B, const int& pos) {

if (root == NULL)

return;

insert(root, pos + 1, 0, -1);

A = root->child[0];

B = root->child[1];

delete root;

}

static void merge(Treap \*&root,

Treap \*A, Treap \*B) {

root = A;

insert(root, getSz(A) + 1, 0, -1);

root->child[1] = B;

update(root);

erase(root, getSz(A) + 1);

}

public:

static void insert(Treap \*&root, const int& pos,

const int& val, const int& key) {

if (root == NULL) {

root = new Treap(key, val);

return;

}

down(root);

int szLef = getSz(root->child[0]);

int dir = (pos > szLef + 1 ? 1 : 0);

insert(root->child[dir],

pos - (dir == 0 ? 0 : szLef + 1),

val, key);

if (root->key > root->child[dir]->key)

rotate(root, dir);

update(root);

}

static int findVal(Treap \*&root, const int& pos) {

if (root == NULL)

return -1;

down(root);

int szLef = getSz(root->child[0]);

if (pos == szLef + 1)

return root->val;

if (pos > szLef + 1)

return

findVal(root->child[1], pos - szLef - 1);

return findVal(root->child[0], pos);

}

static void erase(Treap \*&root, const int& pos) {

if (root == NULL)

return;

down(root);

int szLef = getSz(root->child[0]);

if (pos == szLef + 1)

erase\_this(root);

else

if (pos > szLef + 1)

erase(root->child[1],

pos - szLef - 1);

else

erase(root->child[0], pos);

update(root);

}

static void reverse(Treap \*&root,

const int& lef, const int& rig) {

Treap \*A = NULL, \*B = NULL, \*C = NULL;

split(root, A, B, lef - 1);

split(B, B, C, rig - lef + 1);

B->rev = true;

merge(root, A, B);

merge(root, root, C);

}

static void clear(Treap \*&root) {

if (root == NULL)

return;

clear(root->child[0]);

clear(root->child[1]);

delete root;

root = NULL;

}

};

Treap \*root = NULL;

***FFT***

#include <algorithm>

#include <vector>

#include <complex>

using namespace std;

typedef complex<double> base;

const double PI = acos(-1);

void fft(vector<base> & a, bool invert) {

int n = a.size();

if (n == 1)

return;

vector<base> a0(n / 2), a1(n / 2);

for (int i = 0, j = 0; i < n; i += 2, ++j) {

a0[j] = a[i];

a1[j] = a[i + 1];

}

fft(a0, invert);

fft(a1, invert);

double ang = 2 \* PI / n \* (invert ? -1 : 1);

base w(1), wn(cos(ang), sin(ang));

for (int i = 0; i < n / 2; ++i) {

a[i] = a0[i] + w \* a1[i];

a[i + n / 2] = a0[i] - w \* a1[i];

if (invert)

a[i] /= 2, a[i + n / 2] /= 2;

w \*= wn;

}

}

void multiply(const vector<int>& a, const vector<int>& b, vector<int>& res) {

vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());

size\_t n = 1;

while (n < max(a.size(), b.size())) n <<= 1;

n <<= 1;

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

fft(fa, false), fft(fb, false);

for (size\_t i = 0; i < n; ++i)

fa[i] \*= fb[i];

fft(fa, true);

res.resize(n);

for (size\_t i = 0; i < n; ++i)

res[i] = int(fa[i].real() + 0.5);

res.resize(a.size() + b.size() - 1);

}

***NUMERICAL ALGORITHM***

***RABIN MILLER***

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair <ll, ll> pll;

map <ll, ll> prm;

ll t[3] = { 2, 7, 61 };

ll mod\_pow(ll a, ll b, ll n) {

ll res = 1;

while (b) {

if (b & 1)

res = (res \* a) % n;

a = (a \* a) % n;

b >>= 1;

}

return res;

}

pll thed(ll n) {

ll cnt = 0;

while (!(n & 1)) {

n >>= 1;

++cnt;

}

return pll(n, cnt);

}

int isprime(ll n) {

if (n == 1)

return 0;

else

if (n == 2)

return 1;

else

if (n == 3)

return 1;

else

if (n == 5)

return 1;

else

if (n == 7)

return 1;

else

if (n == 61)

return 1;

else {

ll a, x, d, flag;

int s;

pll ast = thed(n - 1);

d = ast.first;

s = ast.second;

for (int i = 0; i < 3; ++i) {

a = t[i];

x = mod\_pow(a, d, n);

if (x == 1 || x == n - 1)

continue;

else {

flag = 1;

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

x = (x \* x) % n;

if (x == 1)

return 0;

if (x == n - 1) {

flag = 0;

break;

}

}

if (flag)

return 0;

}

}

return 1;

}

}

int main() {

int lef, rig, res = 0;

cin >> lef >> rig;

for (int i = lef; i <= rig; ++i)

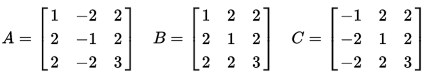
res += isprime(i);

cout << res;

return 0;

}

***Tree of primitive Pythagorean triples***



***Extended Euclidean algorithm***

typedef long long ll;

ll ExtendedGcd(ll a, ll b, ll &x, ll &y) {

if (a % b == 0) {

x = 0;

y = 1;

return b;

}

ll newx, newy;

ll ret = ExtendedGcd(b, a % b, newx, newy);

x = newy;

y = newx - newy \* (a / b);

return ret;

}

//Trả về một nghiệm(x0

//, y0) của pt ax + by = 1 với a>0, b>0, gcd(a, b) = 1.Khi đó

//nghiệm tổng quát của pt là x = x0 + kb, y = y0−ka

***GEOMETRY***

x' = x\*cos(phi) - y\*sin(phi)

y' = x\*sin(phi) + y\*cos(phi)

Phép xoay góc phi quanh tâm là O

***GEOMETRY (HAVE CONVEX\_HULL)***

#include <math.h>

#include <stdlib.h>

#include <algorithm>

#include <vector>

const double PI = acos(-1);

const double EP = 0.000000001;

template <class T>

T sqr(T x) {

return x \* x;

}

struct Point;

struct Vector;

struct Segment;

struct Line;

struct Polygon;

struct Point {

double x, y;

Point() : x(0), y(0) {}

Point(const double& x, const double& y) : x(x), y(y) {}

bool operator == (const Point& point) const {

return fabs(x - point.x) < EP && fabs(y - point.y) < EP;

}

double disToPoint\_square(const Point& other) const {

return sqr(x - other.x) + sqr(y - other.y);

}

double disToPoint(const Point& other) const {

return sqrt(disToPoint\_square(other));

}

double disToLine(const Line& line) const;

};

struct Vector {

double x, y;

Vector() : x(0), y(0) {}

Vector(const double& x, const double& y) : x(x), y(y) {}

Vector(const Point& a, const Point& b) : x(b.x - a.x), y(b.y - a.y) {}

Vector(const Segment& seg);

bool operator == (const Vector& point) const {

return fabs(x - point.x) < EP && fabs(y - point.y) < EP;

}

Vector operator \* (const double& cof) const {

return Vector(x \* cof, y \* cof);

}

Vector& operator \*= (const double& cof) {

return \*this = Vector(x \* cof, y \* cof);

}

double length() const {

return sqrt(sqr(x) + sqr(y));

}

double cosToVector(const Vector& other) const {

return (x \* other.x + y \* other.y) / (length() \* other.length());

}

int isDirect(const Vector& other) const {

double cosValue = cosToVector(other);

if (fabs(cosValue - 1) < EP)

return 1;

if (fabs(cosValue + 1) < EP)

return -1;

return 0;

}

};

struct Segment {

Point a, b;

Segment() : a(Point(0, 0)), b(Point(0, 0)) {}

Segment(const Point& a, const Point& b) : a(a), b(b) {}

bool hasPoint(const Point& point) const {

if (point == a || point == b)

return true;

Vector vecA(point, a);

Vector vecB(point, b);

return vecA.isDirect(vecB) == -1;

}

bool crossSegment(const Segment& other) const;

};

Vector::Vector(const Segment& seg) : x(seg.b.x - seg.a.x), y(seg.b.y - seg.a.y) {}

struct Line {

double a, b, c;

Line(const double& a, const double& b, const double& c) : a(a), b(b), c(c) {}

Line(const Point& a, const Point& b) {

Vector vec1(a, b);

Vector vec2(-vec1.y, vec1.x);

\*this = Line(vec2.x, vec2.y, -a.x \* vec2.x - a.y \* vec2.y);

}

Line(const Segment& seg) : Line(seg.a, seg.b) {}

double valueWithPoint(const Point& point) const {

return a \* point.x + b \* point.y + c;

}

bool hasPoint(const Point& point) const {

return fabs(valueWithPoint(point)) < EP;

}

bool crossLine(const Line& other) const {

double det = a \* other.b - other.a \* b;

return det != 0;

}

Point commonPointWithLine(const Line& other) const {

double det = a \* other.b - other.a \* b;

return Point(-(other.b \* c - b \* other.c) / det, -(a \* other.c - other.a \* c) / det);

}

};

double Point::disToLine(const Line& line) const {

return fabs(line.a \* x + line.b \* y + line.c) / sqrt(sqr(line.a) + sqr(line.b));;

}

bool Segment::crossSegment(const Segment& other) const {

Vector vecA(\*this);

Vector vecB(other);

if (vecA.isDirect(vecB) != 0)

return hasPoint(other.a) || hasPoint(other.b) || other.hasPoint(a) || other.hasPoint(b);

Line lineA(\*this);

Line lineB(other);

return lineA.valueWithPoint(other.a) \* lineA.valueWithPoint(other.b) <= 0 &&

lineB.valueWithPoint(a) \* lineB.valueWithPoint(b) <= 0;

}

struct Polygon {

std::vector <Point> setPoint;

Polygon() {

setPoint.clear();

}

Polygon(const int& sz, Point \*const& points) {

setPoint.clear();

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

setPoint.push\_back(points[i]);

}

Polygon(std::vector <Point> points) {

setPoint = points;

}

void push(const Point& point) {

setPoint.push\_back(point);

}

static double cross(const Point &O, const Point &A, const Point &B) {

return (double)(A.x - O.x) \* (B.y - O.y) - (double)(A.y - O.y) \* (B.x - O.x);

}

static bool cmp(const Point& a, const Point& b) {

return a.x < b.x || (fabs(a.x - b.x) < EP && a.y < b.y);

}

void sortClockWise() {

int n = setPoint.size(), k = 0;

std::vector <Point> H(2 \* n);

// Sort points lexicographically

std::sort(setPoint.begin(), setPoint.end(), cmp);

// Build lower hull

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

while (k >= 2 && cross(H[k - 2], H[k - 1], setPoint[i]) <= 0) k--;

H[k++] = setPoint[i];

}

// Build upper hull

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

while (k >= t && cross(H[k - 2], H[k - 1], setPoint[i]) <= 0) k--;

H[k++] = setPoint[i];

}

H.resize(k - 1);

std::reverse(H.begin(), H.end());

setPoint = H;

}

bool hasPoint(const Point& point) {

int i, j;

bool res = false;

for (i = 0, j = setPoint.size() - 1; i < setPoint.size(); j = i++)

if (((setPoint[i].y > point.y) != (setPoint[j].y > point.y)) &&

(point.x < (setPoint[j].x - setPoint[i].x) \* (point.y - setPoint[i].y) / (setPoint[j].y - setPoint[i].y) + setPoint[i].x))

res = !res;

return res;

}

};