// PADDED SEGMENT TREE, WITH LAZY PROPAGATION

template<typename T>

struct segment\_tree {

//INGAT: kalau pake 1-based index, buat array source nya 1-based index, bukan segment tree nya.

vector<T> tree, lz;

int sz = 1;

bool lazy\_propagation = false;

segment\_tree(vector<T>&ar, bool lp = false){

lazy\_propagation = lp;

int n = ar.size();

while(sz < n) sz \*= 2;

tree.resize(2 \* sz);

if(lazy\_propagation){

lz.resize(2 \* sz);

}

build(ar);

}

// TO-DO: fill this build function

void build(vector<T>&ar){

int n = ar.size();

// isi leaf nya

for(int i = 0; i < n; i++) tree[sz + i] = ar[i];

// isi internal nodes

for(int i = sz - 1; i >= 1; i--) tree[i] = merge(tree[2 \* i], tree[2 \* i + 1]);

}

// TO-DO: merge behaviour between two nodes

T merge(T a, T b){

return a, b;

}

// TO-DO: change the default return value of out of range

T range\_query(int ql, int qr, int t, int tl, int tr){

if(lazy\_propagation && lz[t]){ // check kalau ada update di lz

tree[t] += lz[t];

if(tl != tr){

lz[2 \* t] += lz[t] / 2;

lz[2 \* t + 1] += lz[t] / 2;

}

lz[t] = 0;

}

if(ql <= tl && tr <= qr) return tree[t];

if(tl > qr || ql > tr) return 0;// default outside value

int mid = (tl + tr) / 2;

return merge(range\_query(ql, qr, 2 \* t, tl, mid) , range\_query(ql, qr, 2 \* t + 1, mid + 1, tr));

}

T range\_query(int ql, int qr){

return range\_query(ql, qr, 1, 0, sz - 1);

}

// TO-DO: updatenya increase apa change?

void point\_update(int idx, T new\_val){

idx += sz;

tree[idx] = new\_val;

for(idx /= 2; idx >= 1; idx /= 2){

tree[idx] = merge(tree[2 \* idx], tree[2 \* idx + 1]);

}

}

// TO-DO: LAZY PROPAGATION, increase range atau update range?

void range\_update(int ql, int qr, int val, int t, int tl, int tr){

if(lz[t]){ // check kalau ada update

tree[t] += lz[t];

if(tl != tr){

lz[2 \* t] += lz[t] / 2;

lz[2 \* t + 1] += lz[t] / 2;

}

lz[t] = 0;

}

if(ql <= tl && tr <= qr){

tree[t] += val \* (tr - tl + 1); // increase range

if(tl != tr){

lz[2 \* t] += val \* (tr - tl + 1) / 2;

lz[2 \* t + 1] += val \* (tr - tl + 1) / 2;

}

return ;

}

if(qr < tl || ql > tr) return ;

int mid = (tl + tr) / 2;

range\_update(ql, qr, val, 2 \* t, tl, mid);

range\_update(ql, qr, val, 2 \* t + 1, mid + 1, tr);

tree[t] = merge(tree[2 \* t] , tree[2 \* t + 1]);

}

// implement lazy\_propagation

void range\_update(int ql, int qr, int val){

if(!lazy\_propagation){

cout << "ERROR: MUST IMPLEMENT LAZY PROPAGATION\n"; return ;

}

range\_update(ql, qr, val, 1, 0, sz - 1);

}

void \_print(){

for(int i = 1; i < 2 \* sz; i++){

cout << i << ": " << tree[i] << '\n';

}

}

};

// DSU

struct DSU{

int n;

vector<int> par, sz;

int mx\_size = 1, comp;

DSU(int \_n){

n = \_n;

par.resize(n + 1);

sz.resize(n + 1);

comp = \_n;

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

par[i]=i;

sz[i] = 1;

}

}

int findRep(int a){

if(a == par[a]) return a;

sz[par[a]]=sz[findRep(par[a])];

return par[a] = findRep(par[a]);

}

bool same(int a, int b){

return findRep(a) == findRep(b);

}

void join(int a, int b){

if(same(a, b)) return ;

a = findRep(a);

b = findRep(b);

mx\_size = max(mx\_size, sz[a] + sz[b]);

sz[a] += sz[b];

par[b] = a;

comp--;

}

};

// LCA

const int N = 1e4+5, LOG = 20;

int n, m;

vector<int> adj[N];

vector<vector<int>> anc(N, vector<int>(LOG));

vector<int> depth(N);

int get\_lca(int a, int b){

if(depth[a] > depth[b]) swap(a, b); // buat b lebih dalam

// samakan depth dari a dan b, b lebih dalam

int k = depth[b] - depth[a];

for(int j = LOG - 1; j >= 0; j--){

if(k >= (1 << j)){

k -= (1<<j);

b = anc[b][j];

}

}

// depth sama dan node sama

if(a == b) return a;

// depth sama node beda

for(int j = LOG - 1; j >= 0; j--){

if(anc[a][j] != anc[b][j]){

a = anc[a][j];

b = anc[b][j];

}

}

return anc[a][0];

}

void fill\_anc(int now, int par){

anc[now][0] = par;

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

anc[now][i] = anc[anc[now][i-1]][i-1];

}

depth[now] = depth[par] + 1;

for(int nxt : adj[now]){

if(nxt == par) continue;

fill\_anc(nxt, now);

}

}

int main(){

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

cin >> n;

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

int cnt = 0;

cin >> cnt;

while(cnt--){

int v; cin >> v;

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

}

}

fill\_anc(0, 0);

int q; cin >> q;

while(q--){

int a, b;

cin >> a >> b;

cout << get\_lca(a, b) << '\n';

}

}

// PREFIX SUM 2D

struct prefsum\_2d{

int n, m;

vector<vector<int>> pf;

void init(vector<vector<int>>&ar, int \_n, int \_m){

n = \_n;

m = \_m;

pf = ar;

}

void build(){

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

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

pf[i][j] += pf[i-1][j] + pf[i][j-1] - pf[i-1][j-1];

}

}

}

int get\_sum(int r1, int c1, int r2, int c2){

if(r1 <= 0) r1=1;

if(r2 <= 0) r2=1;

if(r1 > n) r1=n;

if(r2 > n) r2=n;

if(c1 <= 0) c1=1;

if(c2 <= 0) c2=1;

if(c1 > m) c1=m;

if(c2 > m) c2=m;

return pf[r2][c2] - pf[r1-1][c2] - pf[r2][c1-1] + pf[r1-1][c1-1];

}

};

// MATRIX OPERATION

vector<vector<int>> mul(vector<vector<int>>left,vector<vector<int>>right){

vector<vector<int>> res;

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

res.push\_back({});

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

int sum = 0;

for(int k = 0; k < left[0].size(); k++){

sum += left[i][k] \* right[k][j];

sum = (sum + MOD) % MOD;

}

res.back().push\_back(sum);

}

}

return res;

}

vector<vector<int>> mpow(vector<vector<int>> a, int b){

vector<vector<int>> res = { {1, 0}, {0, 1} };

while(b){

if(b % 2) res = mul(res, a);

a = mul(a, a);

b /= 2;

}

return res;

}

// SQUARE ROOT DECOMPOSITION

struct square\_root\_decomposition{

vector<int> ar;

vector<vector<int>> s;

int sq, n;

square\_root\_decomposition(vector<int>&A){

n = A.size();

sq = sqrt(n) + 1;

ar = A;

s.resize(sq);

}

void build(){

// setiap blok isi nya sorted array

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

s[i / sq].push\_back(ar[i]);

}

for(vector<int>&v : s) if(v.size()){

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

}

}

int qry(int l, int r){ // TO-DO: sesuaikan tipe query

int res = 0;

for(auto&v : s){

int sz = v.size();

int lb = sz;

for(int J = 1 << 9; J; J /= 2){

if(lb - J >= 0 && v[lb - J] >= l) lb -= J;

}

int ub = -1;

for(int J = 1 << 9; J; J /= 2){

if(ub + J < sz && v[ub + J] <= r) ub += J;

}

res += max(0ll, ub - lb + 1);

}

return res;

}

void upd(int idx, int new\_val){// idx use 0 based index

int old\_val = ar[idx];

ar[idx] = new\_val;

int block = idx / sq;

// hapus old\_val di s[block], ganti ke new\_val

int sz = s[block].size();

int pos = -1;

for(int J = 1 << 9; J; J /= 2){

if(pos + J < sz && s[block][pos + J] <= old\_val) pos+=J;

}

s[block][pos] = new\_val;

// urutkan lagi array nya

int j = pos;

while(j > 0 && s[block][j-1] > s[block][j]){

swap(s[block][j-1] , s[block][j]);

j--;

}

j = pos;

while(j + 1 < sz && s[block][j] > s[block][j+1]){

swap(s[block][j] , s[block][j+1]);

j++;

}

}

void print(){

for(auto v : s){

for(int i : v){

cout << i << ' ';

}

cout << '\n';

}

cout << '\n';

}

};

void doumo\_same\_desu(){

int n, q;

cin >> n >> q;

vector<int> ar(n);

for(int i = 0; i < n; i++) cin >> ar[i];

square\_root\_decomposition srd(ar);

srd.build();

// srd.print();

while(q--){

char t;

int a, b;

cin >> t >> a >> b;

if(t == '?'){ // count number of ppl with salary a .. b

cout << srd.qry(a, b) << '\n';

}

else { // ganti gaji orang ke a menjadi b

srd.upd(a - 1, b);

}

}

}

=========================================================

// MILLER RABIN

using u64 = uint64\_t;

using u128 = \_\_uint128\_t;

u64 binpower(u64 base, u64 e, u64 mod) {

u64 result = 1;

base %= mod;

while (e) {

if (e & 1) result = (u128)result \* base % mod;

base = (u128)base \* base % mod;

e >>= 1;

}

return result;

}

bool check\_composite(u64 n, u64 a, u64 d, int s) {

u64 x = binpower(a, d, n);

if (x == 1 || x == n - 1) return false;

for (int r = 1; r < s; r++) {

x = (u128)x \* x % n;

if (x == n - 1) return false;

}

return true;

};

// returns true if n is probably prime, else returns false.

bool MillerRabin(u64 n, int iter=5) {

if (n < 4) return n == 2 || n == 3;

int s = 0;

u64 d = n - 1;

while ((d & 1) == 0) {

d >>= 1;

s++;

}

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

int a = 2 + rand() % (n - 3);

if (check\_composite(n, a, d, s))

return false;

}

return true;

}

// RABIN KARP

struct RabinKarp{

static const int MOD = 1e9 + 9;

int PP = 53;

vector<int> H, P; // uses 1-base index

string S;

void init(string s, int \_PP = 53){

S = " " + s;

PP = \_PP;

int sz = s.size();

H = vector<int>(sz + 1);

P = vector<int>(sz + 1);

P[0]=1;

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

P[i] = (P[i-1] \* PP) % MOD;

}

// calculate hash table for S

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

H[i] = (H[i-1] + (S[i]-'a'+1) \* P[i]) % MOD;

}

}

int get\_hash(int l, int r){ // remember to use 1-based index

int res = (H[r]-H[l-1]+MOD)%MOD;

res \*= inv(P[l], MOD);

res %= MOD;

return res;

}

};

// FLOYD WARSHALL

void doumo\_same\_desu(){

cin >> n >> m >> q;

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

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

if(i == j) adj[i][j] = 0;

else adj[i][j] = INF;

}

}

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

int u, v, w;

cin >> u >> v >> w;

adj[u][v] = min(adj[u][v], w);

adj[v][u] = min(adj[v][u], w);

}

// floyd-warshall

for(int i = 1; i <= n; i++){ // node i jadi jembatan

// cek semua pasangan

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

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

adj[j][k] = min(adj[j][k], adj[j][i] + adj[i][k]);

}

}

}

while(q--){

int u, v;

cin >> u >> v;

cout << (adj[u][v] >= INF ? -1 : adj[u][v]) << '\n';

}

}

// BELLMAN FORD

const int N = 5005, INF = 1e15;

pair<int,int> edge[N]; // first-> awal, second->tujuan

int n, m, dist[N], cost[N];

vector<int> adj[N];

bool vis[N];

void dfs(int now){

if(vis[now]) return ;

vis[now] = 1;

for(int j : adj[now]) dfs(j);

}

void doumo\_same\_desu(){

cin >> n >> m;

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

cin >> edge[i].first >> edge[i].second >> cost[i];

adj[edge[i].second].push\_back(edge[i].first);

}

dfs(n);

for(int i = 1; i <= n; i++) dist[i] = -INF;

dist[1] = 0;

bool changed;

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

changed = false;

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

auto[u, v] = edge[j];

if(dist[u] == -INF) continue;

if(dist[v] < dist[u] + cost[j]){

dist[v] = dist[u] + cost[j];

changed |= vis[v];

// changed = true;

}

}

}

if(changed){

cout << -1 << "\n";

return ;

}

cout << dist[n] << '\n';

}

=========================================================

// BASE TEMPLATE

#include<bits/stdc++.h>

using namespace std;

#define int long long

#define Int \_\_int128\_t

#define bpc(x) \_\_builtin\_popcountll(x)

#define msb(x) (63-\_\_builtin\_clzll(x))

#ifdef DEBUG

#define dbg(x) cout<<"["<< #x <<"] : "<<(x)<<endl;

#else

#define dbg(x)

#endif

int bpow(int a, int b, long long mod=LLONG\_MAX){

int res=1;while(b){if(b%2)res=res\*a%mod;a=a\*a%mod;b/=2;}return res;

}

int inv(int a, int mod=1e9+7){ return bpow(a, mod-2, mod); }

signed main(){

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

cout << fixed << setprecision(5);

}

// KNAPSACK RECURSIVE RESULT BACKTRACK

const int N = 2005;

int w[N], h[N], k, n, dp[N][N];

int f(int i, int bag){

if(i < 0) return 0;

if(dp[i][bag] != -1) return dp[i][bag];

int res = f(i - 1, bag);

if(bag >= w[i]) res = max(res, f(i - 1, bag - w[i]) + h[i]);

return dp[i][bag] = res;

}

void solve(){

cin >> n >> k;

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

cin >> w[i] >> h[i];

}

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

f(k - 1, n);

int x = k - 1, y = n;

vector<int> res;

while(x >= 0 && y >= 0){

if(y >= w[x] && f(x - 1, y - w[x]) + h[x] == dp[x][y]){

res.push\_back(x);

y -= w[x];

}

x--;

}

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

for(int i : res) cout << i + 1 << " ";

}

// KNAPSACK ITERATIVE RESULT BACKTRACK

int w[2100], h[2100], dp[2100][2100];

void solve(){

int bag, n;

cin >> bag >> n;

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

cin >> w[i] >> h[i];

}

for(int i = 0; i < 2100; i++)memset(dp[i], 0, sizeof(dp[i]));

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

for(int j = 0; j <= bag; j++){

int res = dp[i-1][j];

if(j >= w[i]){

res = max(res, dp[i-1][j-w[i]] + h[i]);

}

dp[i][j] = res;

}

}

int min\_w = bag;

for(int i = bag - 1; i >= 0; i--){

if(dp[n][bag] == dp[n][i]) min\_w = i;

}

vector<int> c;

for(int i = n; i > 0 && min\_w > 0; i--){

if(dp[i][min\_w] != dp[i-1][min\_w]){

min\_w -= w[i];

c.push\_back(i);

}

}

for(int i = c.size() -1; i >= 0; i--) cout << c[i] << '\n';

}

// POINT LOCATION TEST

void solve(){

pair<int, int> a, b, c;

cin >> a.first >> a.second >> b.first >> b.second >> c.first >> c.second;

pair<int, int> ab = {b.first - a.first, b.second - a.second};

pair<int, int> ac = {c.first - a.first, c.second - a.second};

int res = ab.first \* ac.second - ac.first \* ab.second;

if(res == 0){

cout << "TOUCH" << '\n';

return ;

}

cout << (res > 0 ? "LEFT" : "RIGHT") << '\n';

}

=========================================================

// LINE SEGMENT INTERSECTION

struct P{

int x, y;

void read(){

cin >> x >> y;

}

P operator -(P b){

return P{x - b.x, y - b.y};

}

int operator \*(P b){

return x \* b.y - y \* b.x;

}

int triangle(P a, P b){

return (a - \*this) \* (b - \*this);

}

};

void solve(){

vector<P> arr(4);

for(int i = 0; i < 4; i++) arr[i].read();

if((arr[1] - arr[0]) \* (arr[3] - arr[2]) == 0){

if(arr[0].triangle(arr[1], arr[2]) != 0){

cout << "NO" << '\n'; return ;

}

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

if(max(arr[0].x, arr[1].x) < min(arr[2].x, arr[3].x) ||

max(arr[0].y, arr[1].y) < min(arr[2].y, arr[3].y)){

cout << "NO" << '\n'; return ;

}

swap(arr[0], arr[2]);

swap(arr[1], arr[3]);

}

cout << "YES" << '\n';

return ;

}

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

int A = (arr[1] - arr[0]) \* (arr[2] - arr[0]);

int B = (arr[1] - arr[0]) \* (arr[3] - arr[0]);

if((A < 0 && B < 0) || (A > 0 && B > 0)){

cout << "NO" << '\n';

return ;

}

swap(arr[0], arr[2]);

swap(arr[1], arr[3]);

}

cout << "YES" << "\n";

}

// POLYGON AREA

const int mxN = 1e3 + 5;

struct T{

int x, y;

} arr[mxN];

int n;

void solve(){

cin >> n;

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

cin >> arr[i].x >> arr[i].y;

}

int res = 0;

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

res += (arr[i].x\*arr[(i+1)%n].y - arr[i].y\*arr[(i+1)%n].x);

}

cout << abs(res) << '\n';

}

=========================================================

// CONVEX HULL

const int INF = 1e15;

struct pt{

int x, y;

pt(){}

pt(int \_x, int \_y){

x=\_x; y=\_y;

}

void read(){

cin >> x >> y;

}

void dbug(){

cout << x << ' ' << y << '\n';

}

bool same(pt other){

return x == other.x && y == other.y;

}

};

vector<pt> ar;

int n;

int cross\_product(pt p, pt q, pt r){

return (r.y - p.y) \* (q.x - p.x) - (q.y - p.y) \* (r.x - p.x);

}

bool belok\_kiri(pt p, pt q, pt r){

return (r.y - p.y) \* (q.x - p.x) >=

(q.y - p.y) \* (r.x - p.x);

}

float distance(pt a, pt b){

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

}

void doumo\_same\_desu(){

cin >> n;

ar.resize(n);

pt leftmost = pt(INF, INF);

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

ar[i].read();

if(ar[i].y < leftmost.y) leftmost = ar[i];

else if(ar[i].y == leftmost.y && ar[i].x < leftmost.x) leftmost = ar[i];

}

{

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

if(ar[i].x == leftmost.x && ar[i].y==leftmost.y){

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

ar[j] = ar[j+1];

}

ar.pop\_back();

break;

}

}

}

sort(ar.begin(), ar.end(), [&](pt a, pt b){

if((a.y-leftmost.y)\*(b.x-leftmost.x)==(b.y-leftmost.y)\*(a.x-leftmost.x)){ // collinear

return distance(leftmost, a) > distance(leftmost, b);

}

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

});

ar.push\_back(leftmost);

// cout << "---\n";

// for(pt p : ar) p.dbug();

// cout << "---\n";

vector<pt> ans = {leftmost, ar[0]};

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

pt prv = ans[ans.size()-2], now = ans[ans.size()-1], nxt = ar[i];

if(belok\_kiri(prv, now, nxt)){

ans.push\_back(nxt);

}

else {

ans.pop\_back();

i--;

}

}

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

if(ar[i].same(leftmost) || ar[i].same(ans[1])) continue;

if(cross\_product(leftmost, ar[i], ans[1]) == 0) ans.push\_back(ar[i]);

}

cout << ans.size() -1 << '\n';

leftmost.dbug();

for(int i = 1; i < (int)ans.size();i++){

if(ans[i].same(leftmost)) continue;

ans[i].dbug();

}

}