**Math Tools**

**read in octal and convert into binary:**

python:

print("{0:b}".format(int(str(input()), 8)))

java:

int b = in.nextInt(8);

baseConversion(String number, int sBase, int dBase)

{

// Parse the number with source radix

// and return in specified radix(base)

return Integer.toString(

Integer.parseInt(number, sBase), dBase);

}

**Count number of digits in base b:**

(int)floor(1 + log10((double) val) / log10((double)b))

**n-th root:**

Math.pow((double)a, 1.0 / (double)n)

**if N = a^i \* b^j \* c^k**

**sum of divisors of N = (a^(i - 1) - 1) \* (b^(j - 1) - 1) \* (c^(k - 1) - 1) / ((a - 1) \* (b - 1) \* (c - 1))**

**Count number of coprime numbers to N:**

Using Euler's Phi function:

Euler(N) = N \* product(1 - 1/pi), where pi is a prime factor of N

**solve linear diophantine equation:**

integer solution to 25x + 18y = 839

compute gcd(25, 18) = 1

extended euclidean alg: 25 \* -5 + 18 \* 7 = 1

multiply both side by 839: 25 \* -4195 + 18 \* 5873 = 839

-4195 + 18n >= 0, 5873 - 25n >= 0

233.05 <= n <= 234.92, n = 234

n-th fibonacci number = (a^n - (-a)^(-n)) / sqrt(5), where a = (1 + sqrt(5)) / 2

or [1 1] ^ n.

[1 0]

**catalan numbers: Cat(n + 1) = (4n + 2) / (n + 2) \* Cat(n)**

(distinct binary trees with n vertices)

(number of expressions containing n pairs of parentheses)

Compute number of paths of length L in a graph (graph stored in adjacency matrix M): (M^L)[0][1]

**given p = a + b, q = a \* b, and n, compute a^n + b^n**

X = a^n + b^n = (a + b) \* (a ^ (n - 1) + b ^ (n - 1)) - (a \* b) \* (a ^ (n - 2) + b ^ (n - 2))

recurrence: p \* Xn-1 - q \* Xn = Xn+2, p \* Xn - q \* Xn-1 = Xn+1

[p -q] \* [Xn+1] = [Xn+2]

[1 0] [Xn] [Xn+1]

so [Xn+1] = [p -q]^n \* [X1]

[Xn] [1 0] [X0]

**Compute C(n + 1, k + 1) (n large k small) = sum p from k to n: C(p, n) = 2^(n - 1) - sum p from 1 to k - 1:** C(p, n) with matrix multiplication

**suffix tree:**

Longest repeated substring in O(n): path with longest characters to a node that have two children or more.

Longest common substring in O(n): a tree that added suffices of 2 strings, add a different character in the end of each string, find

path with longest characters to a node that have two children with different ending characters.

**the least significant 1 bit:**

x & -x

**n-queen problem: accelerated with 3 bitmasks.**

basic idea: iterate column by column.

For each column j:

bit mask rw: i-th bit is 1 if row i is attacked already (thus attacking (i, j) too)

bit mask ld: i-th bit is 1 if some leftward diagonal attacks (i, j)

... rd: .... rightward ...

code:

int ans = 0, OK = (1<<n) - 1;

void backtrack(int rw, int ld, int rd){

if(rw == OK){

//all n rows filled, since on each column there is at most one queen, this means that we've filled all n queens.

ans++;

return;

}

int pos = OK & (~(rw | ld | rd)); //i-th bit of pos is true if (i, j) not attacked

while(pos){

int p = pos & -pos;

pos -= p;

backtrack(rw | p, (ld | p) << 1, (rd | p) >> 1);

}

}

int main(){

backtrack(0, 0, 0);

}

**Rounding:**

**PrintWriter out = new PrintWriter(System.out, true);**

**out.printf("%.3f\n", lo);**

**Tower of Hanoi:**

n disks, 3 poles A, B, C.

Solution:

move disk 1 to n - 1 to B with C as intermediate.

move disk n to c

move disk 1 to n - 1 to C with A as intermediate.

Can be solved with memoization.

Optimal Moves: 2^n - 1.

**De Brujin Sequence of (n, alphabet set A):**

Shortest CIRCULAR string containing every string of length n with letters in A.

Example De Brujin(3, {a, b}) = "aaababbb", length 8 = 2 ^ 3.

Generally, De Brujin(n, A s.t. |A| = k) has length k ^ n (which is also the minimum possible length).

Generating a De Brujin sequence: using a De Brujin Graph (Directed Graph, one vertex for each possible string of length n with letters in A, and an edge (s1, s2) if s2 = s1[1..n - 1] + c, c in A, the edge has label c)

n^k vertices, n^(k+1) edges, each vertex has k incoming and k outgoing edges.

Now perform eulerian tour on the De Brujin graph (n - 1, A), and concatenate the edge letters.

Number of De Brujin sequence: (k!)^(k^(n - 1)) / (k^n)

For k = 2, it's 2^(2^(n - 1) - n)

**FFT:**

Poly mult is the same as convolution of two sequences: s-th term of f \* g = sum\_{j + k = s}(fj \* gk)

Example: How many ways to get y = Aj + Bk, given array A, B.

let fi = freq(i in A), gi = freq(i in B)

Take (f \* g)y as answer.

All Dot products:

A, B array, len(A) >= len(B), compute sum of dot(B, substring(A))

Convolution of A and B, take the centers of terms in the convolution and get the answers.

Example: A = 5 7 2 1 3 6, B = 2 1 3 4, convolution = 20, 43, 34, 27, 31, 38, 23, 12, 12. 27, 31, 38 are what we want.

String Matching: detertine how many times B appears in A as a substring.

fj = e^(i \* 2pi \* Aj / n)

gj = e^(-i \* 2pi \* Bj / n)

if the corresponding j-th term in convolution of f \* g (refer to above) is equal to length of B, then string matches.

Wildward String Matching: B can have '\_' elements:

set gj to 0 if B[j] = '\_'

corresponding term in convolution should be len(B) - B's wildcard counts

All Distances:

Given bitstring A, for all k, determine how many j s.t. Aj = A(j+k) = 1

Just do all dot products of A and itself.

**Pollard Rho:**

Quicker factorization (probable factorization), time complexity around O(n ^ 1/4).

Text

Description automatically generated

**Lucas Theorem:**

Text, letter

Description automatically generated

Text, letter

Description automatically generated

**Additional Formulas:**

Text

Description automatically generated

Text, letter

Description automatically generated

**Sprague Grundy:**

In [combinatorial game theory](https://en.wikipedia.org/wiki/Combinatorial_game_theory), an **impartial game** is a [game](https://en.wikipedia.org/wiki/Mathematical_game) in which the allowable moves depend only on the position and not on which of the two players is currently moving, and where the payoffs are symmetric. In other words, the only difference between player 1 and player 2 is that player 1 goes first.  [Go](https://en.wikipedia.org/wiki/Go_(board_game)) and [chess](https://en.wikipedia.org/wiki/Chess) are not impartial, as each player can only place or move pieces of their own color. Games such as [poker](https://en.wikipedia.org/wiki/Poker), [dice](https://en.wikipedia.org/wiki/Dice) or [dominos](https://en.wikipedia.org/wiki/Dominoes) are not impartial games as they rely on chance.

Any impartial game with perfect information and final “win” / “lose” and finite number of moves can be reduce to NIM.

Sprague Grundy Theorem:

Nim-value(n) = mex(xor-sum(nim-value(mi))), where {mi1, … mij} is a state n results in after 1 move.

Say if taking one move from n = 5, would result in 2 states: {2, 3}, and {4}, then nim-value(5) = mex(nim-value(2) xor nim-value(3), nim-value(4))

**DP Optimization:**

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

Diagram, text

Description automatically generated

**Gaussian Elimination:**

Text

Description automatically generated

**Graph Tools:**

**HierHolzer's to find Eulerian Path**

void dfsEulerianPath(map<string, vector<string>>& graph, string node, vector<string>& path) {

// iterate adjacency list backwards to make use of vector.pop\_back();

while (!graph[node].empty()) {

string v = graph[node].back();

if (!graph[node].empty())

graph[node].pop\_back();

dfsEulerianPath(graph, v, path);

}

// Hierholzer's algorithm: add node when backtrack from node;

path.push\_back(node);

}

**For some graphs shortest s-t distance problems, if too many edges, can try meet in middle.**

**Max-Flow Tips:**

Try to model each inequality x1 + x2 + x3 … + xn <= y1 + y2 + … + ym as n edges each with capacity xi into one vertex and m edges each with capacity yi out of the edge. (But you have to guarantee that in the context of the problem, the flow into this vertex = the flow out of it).

Create directed edge (a, b) as the flow from a (a vertex representing some structure in the problem statement) to b (some other structure). (say you have n taps, m buckets, each tap can deliver at most c capacity to each bucket, and … (some other problem constraints), then you create n vertices as taps, m vertices as buckets, and n \* m edges of capacity c from each vertex to each bucket).

**Min-Cut: after max flow, run bfs from source, all reachable vertices with the residual graph would be in component S, all the rest in S'.**

**Multi-source / sink: create super source ss (connecting to all sources with infinite capacity), same as st.**

**Number of edge disjoint paths from s to t: give vertex flow 1 (which can be reduced to max flow by turning one vertex to 2 new vertices), and give each edge flow 1 (preserve the original graph).**

**What algorithms to use for Max Cardinality Matching?**

if(bipartite){

if(greedy-able){

greedy bipartite matching

}

else if(weighted){

Hungarian Alg

}

else{

Dinic's

}

}

else if(graph small){

DP with Bitmask

}

else if(weighted){

Weighted MCM

}

else{

Edmond's

}

**Heavy Light Decomposition:**

Solve path queries in O(log(N)) (log(N) paths could solve each query)

Decomposing trees into many paths.

An edge (a,b) in the tree is heavy is size(b) >= size(a) / 2, ignore the none heavy edges.

Any paths in the original tree is included in log(N) heavy chains (since light chain decreases the size of the range more than half).

If we identify (a,b) as heavy if b has the largest subtree under a, the properties still remain.

\*: usually if the queries can be solved in log(N) on linked list (line graph), then it can be done log(N) with Heavy Light Decomposition.

Example:

want to perform 2 queries:

1. add a b k: add to each vertex on path (a,b) by value k

2. sum a b: sum values of vertices on path (a,b)

get LCA(a, b) = x, then solve(x, a) and solve(x, b), create a Segment Tree for each heavy chain, then solve(x, a) involves at most O(log(N)) (alternating between heavy chains and light edges) updates, time complexity per query is log^2(N). (Also record the two ends of each heavy chain to enable quick skip)

Text, letter

Description automatically generated

Give a timestamp for all nodes

Text

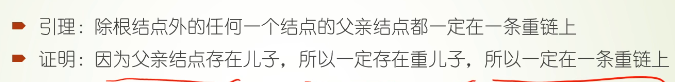
Description automatically generated

A picture containing text, receipt

Description automatically generated

Timeline

Description automatically generated with low confidence



Chart

Description automatically generated

Text

Description automatically generated

**Tree Isomorphism:**

Determining if 2 graphs are isomorphic (exists an f(V1) = V2 s.t. if (a, b) is an edge in G1, (f(a), f(b)) is an edge in G2).

O(N) time complexity.

If two trees are rooted, create a bracket representation of the tree.

child[a] = b, c, d

child[c] = e

then b, d, e = (), c = (()), a = ((())()()) (c followed by b, d, sorted by length of brackets of each child)

Notice strings can be O(N) in length, instead we do rolling hash to transform each string into an int.

Unrooted, need to make sure 2 trees are rooted at the same vertex. So we select some vertex that is UNIQUE in every tree, like center or centroid.

Center: the distance to the farthest vertex is minimum, which is the center of the longest path in the tree.

**Finding Center: BFS on tree, find farthest vertex from any root, denote as v.**

BFS starting from v, denote the farthest vertex as u.

Center is in their middle.

If there are 2 centers, simply try each one, takes twice the time.

**Chinese Postman Problem:**

Shortest tour that visits every edge of a undirected weighted graph:

Want to add double edges to make the graph an even graph (thus can then perform Euler tour, answer = total edge weights)

Find all odd degree vertices (even number of them) and pair them up for the smallest total path length. Let n = number of odd degree vertices, create Kn, each edge weight of (a,b) is the weight of shortest path between a and b.

Now we solve minimum weight perfect matching.

Can be transformed to Hungarian Algorithm (Kuhn-Munkres) through some tweaking (reversing edge weights).

Min-Cost Flow: minimum cost to send some flow from s to t.

Every edge has capacity + unit cost to send one flow.

Replace augmenting path (shortest augmenting path) (O(E) BFS method) by general O(k \* E) algorithm Bellman-Ford-Moore (since now our shortest path can involve negative edges (residual edges have negative costs)).

Final complexity: O(V^2 \* E^2)

**DP Tools**

**Edit Distance of A[1..n] and B[1..m]: use dp[i][j] = edit distance of A[1..i] and B[1..j], compute by maxing 3 cases**

(Edit Distance: how many ops does it take to transform A to B? You are allowed 3 ops: deleting a character, adding a character, change a character)

**Bellman Ford:**

Text

Description automatically generated

**Sparse Table:**

Range Max Query:

RMQ(0,0) RMQ(1,1) RMQ(2,2) ... RMQ(6,6)

RMQ(0,1) RMQ(1,2) ... RMQ(5,6)

RMQ(0,3) RMQ(1,4) ...

Query RMQ(0,5) = Max(RMQ(0,3), RMQ(2,5))

So for a sparse table to work, need an overlapping range query property (ranges maxed over can overlap).

**Square Root Decomposition:**

Q queries of (L, R) (number of distinct integers in A[L..R]): separate A into sqrt(N) buckets, sort the intervals by left boundary.

**Geometry Tools**

**Rotate(alpha) = [cos(alpha) \* x - sin(alpha) \* y, sin(alpha) \* x + cos(theta) \* y]**

**line that passes through 2 points:**

pointsToLine(p1, p2){

if(abs(p1.x - p2.x) < EPS (like 1e-9)){

l = {1.0, 0.0, -p1.x};

}

else{

l = {-(double)(p1.y - p2.y) / (p1.x - p2.x), 1.0, -(double)(p1.y - p2.y) / (p1.x - p2.x) \* p1.x - p1.y)};

}

}

**line that passes through 1 point with slope m:**

{-m, 1.0, -((-m \* p.x) + p.y)};

**intersectOfLines**(line l1, line l2){

if(l1 and l2 parallel)

return null;

x = (l2.b \* l1.c - l1.b \* l2.c) / (l2.a \* l1.b - l1.a \* l2.b);

if(abs(l1.b) > EPS)

y = -(l1.a \* x + l1.c);

else

y = -(l2.a \* x + l2.c);

}

**compute angle aob** (point a, point o, point b){

vec oa = toVec(o, a), ob = toVec(o, b);

return acos(dot(oa, ob) / sqrt(norm(oa) \* norm(ob)));

}

**q, p, r makes a CCW or CW turn**

**double cross(vec a, vec b) {return a.x \* b.y - a.y \* b.x;}**

bool ccw(p, q, r) {return cross(toVec(p, q), toVec(p, r)) > EPS;}

bool collinear(p, q, r) {return abs(cross(toVec(p, q), toVec(p, r))) < EPS}

**double distToLine**(point p, point a, point b) (a, b defines a line){

vec ap = toVec(a, p), ab = toVec(a, b);

double u = dot(ap, ab) / norm(ab);

c = a (vec+) u \* ab;

return dist(p, c);

}

**double distToLineSegment**(point p, point a, point b){

vec ap = toVec(a, p), ab = toVec(a, b);

double u = dot(ap, ab) / norm(ab);

if(u < 0.0){

return dist(p, a);

}

else if(u > 1.0){

return dist(p, b);

}

else{

return distToLine(p, a, b);

}

}

**length of chord with central angle alpha and radius r**

= sqrt(2 \* r^2 \* (1 - cos(alpha)))

**Given 2 points on a circle (p1 and p2) and radius, give two possible centers c1 and c2 of the circle**

{

double d2 = (p1.x - p2.x)^2 + (p1.y - p2.y)^2;

double det = r \* r / d2 - 0.25;

if(det < EPS)

return false;

double h = sqrt(det);

c1 = {(p1.x + p2.x) / 2 + (p1.y - p2.y) \* h, (p1.y + p2.y) / 2 + (p2.x - p1.x) \* h}.

c2 = (c1 but reverse p1 and p2).

}

**size of triangle with 3 sides a, b, c:**

s = (a + b + c) / 2

A = sqrt(s \* (s - a) \* (s - b) \* (s - c))

**inscribed circle radius = Area / semi-perimeter**

**center of incircle = meeting point of triangle's angle bisectors**

**bool inCircle**(p1, p2, p3){

r = rInCircle(p1, p2, p3);

if(abs(r) < EPS) return false;

line l1, l2; //2 angle bisectors

double ratio = dist(p1, p2) / dist(p1, p3);

point p = p2 (vec+) toVec(p2, p3) \* (ratio / (1 + ratio)));

l1 = pointsToLine(p1, p);

ratio = dist(p2, p1) / dist(p2, p3);

p = p1 + toVec(p1, p3) \* (ratio / (1 + ratio));

l2 = pointsToLine(p2, p);

return intersectOfLines(l1, l2);

}

**Circumscribed circle with radius R = a \* b \* c / (4 \* A);**

**Center: meeting point of perpendicular bisectors.**

**law of cosines: c^2 = a^2 + b^2 - 2ab\*cos(gamma), gamma = acos((a^2 + b^2 - c^2) / (2ab));**

**law of sines: a / sin(alpha) = b / sin(beta) = c / sin(gamma) = 2R (R = radius of circumscribed circle)**

**Area of Polygon:**

double area(list of point P){

double res = 0.0;

for(int i = 0; i < P.size() - 1; ++i){

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

}

return res / 2.0;

}

**Checking polygon is convex? Just do CCW on each angle to see if they are all CCW or all CW.**

**Checking if a point pt is in the polygon: sum up the angles made by {P[i], pt, P[i + 1]}. If equals = 2\*pi, then yes, else, no.**

**int insidePolygon**(point pt, list of point P){

int n = P.size();

if(n <= 3)

return -1;

//check if pt on polygon

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

if(abs(dist(P[i], pt) + dist(pt, P[i + 1]) - dist(P[i], P[i + 1])) < EPS){

return 0;

}

}

double sum = 0.0;

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

if(ccw(pt, P[i], P[i + 1])){

sum += angle(P[i], pt, P[i + 1]);

}

else{

sum -= angle(P[i], pt, P[i + 1]);

}

}

return abs(sum) > M\_PI ? 1 : -1;

}

**Polygon representation: front and back are same point (wrap around)**

**Cutting Polygon with a line:**

**point lineIntersectSeg**(point p, point q, point A, point B){

double a = B.y - A.y, b = A.x - B.x, c = B.x \* A.y - A.x \* B.y;

double u = abs(a \* p.x + b \* p.y + c);

double v = abs(a \* q.x + b \* q.y + c);

return point((p.x \* v + q.x \* u) / (u + v), (p.y \* v + q.y \* u) / (u + v));

}

**list<point> cutPolygon**(point A, point B, list of point Q){

list<point> l = new ArrayList<>();

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

double left1 = cross(toVec(A, B), toVec(A, Q[i])), left2 = 0.0;

if(i != Q.size() - 1)

left2 = cross(toVec(A, B), toVec(A, Q[i + 1]));

//Q[i] on the left

if(left1 > -EPS)

l.add(Q[i]);

//crossesline AB

if(left1 \* left2 < -EPS)

l.add(lineIntersectSeg(Q[i], Q[i + 1], A, B));

}

if((!l.isEmpty()) && !(l.back() == l.front())){

l.add(l.front());

}

return l;

}

**convex hull: credit Leetcode**

public class Solution {

public int orientation(int[] p, int[] q, int[] r) {

return (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1]);

}

public int[][] outerTrees(int[][] points) {

Arrays.sort(points, new Comparator<int[]> () {

public int compare(int[] p, int[] q) {

return q[0] - p[0] == 0 ? q[1] - p[1] : q[0] - p[0];

}

});

Stack<int[]> hull = new Stack<>();

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

while (hull.size() >= 2 && orientation(hull.get(hull.size() - 2), hull.get(hull.size() - 1), points[i]) > 0)

hull.pop();

hull.push(points[i]);

}

hull.pop();

for (int i = points.length - 1; i >= 0; i--) {

while (hull.size() >= 2 && orientation(hull.get(hull.size() - 2), hull.get(hull.size() - 1), points[i]) > 0)

hull.pop();

hull.push(points[i]);

}

// remove redundant elements from the stack

HashSet<int[]> ret = new HashSet<>(hull);

return ret.toArray(new int[ret.size()][]);

}

}

**Greedy Tools**

**When sorting by order:**

**prove this:**

if an answer is feasible under an arbitrary order,

it would still be feasible if two elements are swapped under some condition

**Template**

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Solution {

static long MOD = 1000000007L;

public static int[] dir = {1, 0, -1, 0, 1, 1, -1, -1, 1};

public static class Union{

int[] p, size;

public Union(int n){

p = new int[n];

size = new int[n];

Arrays.fill(size, 1);

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

p[i] = i;

}

}

public int find(int n){

if(p[n] != n){

p[n] = find(p[n]);

}

return p[n];

}

public int union(int a, int b){

int fa = find(a), fb = find(b);

if(fa == fb){

return -1;

}

if(size[fa] > size[fb]){

p[fb] = fa;

size[fa] += size[fb];

return size[fa];

}

else{

p[fa] = fb;

size[fb] += size[fa];

return size[fb];

}

}

}

public static String read(){

try{

return sc.readLine();

}

catch(IOException e){

e.printStackTrace();

return "";

}

}

public static String[] read\_toks(){

return read().split(" ");

}

public static BufferedReader sc;

public static void main(String[] args){

sc = new BufferedReader(new InputStreamReader(System.in));

int tt = 1;

for(int qq = 0; qq < tt; ++qq){

int n = Integer.parseInt(read());

}

}

}

**2SAT:**

#include <bits/stdc++.h>

using namespace std;

int n;

vector<vector<int>> adj, adj\_t;

vector<bool> used;

vector<int> order, comp;

vector<bool> assignment;

void dfs1(int v) {

used[v] = true;

for (int u : adj[v]) {

if (!used[u])

dfs1(u);

}

order.push\_back(v);

}

void dfs2(int v, int cl) {

comp[v] = cl;

for (int u : adj\_t[v]) {

if (comp[u] == -1)

dfs2(u, cl);

}

}

bool solve\_2SAT() {

order.clear();

used.assign(n, false);

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

if (!used[i])

dfs1(i);

}

comp.assign(n, -1);

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

int v = order[n - i - 1];

if (comp[v] == -1)

dfs2(v, j++);

}

assignment.assign(n / 2, false);

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

if (comp[i] == comp[i + 1])

return false;

assignment[i / 2] = comp[i] > comp[i + 1];

}

return true;

}

void add\_disjunction(int a, bool na, int b, bool nb) {

// na and nb signify whether a and b are to be negated

a = 2\*a ^ na;

b = 2\*b ^ nb;

int neg\_a = a ^ 1;

int neg\_b = b ^ 1;

adj[neg\_a].push\_back(b);

adj[neg\_b].push\_back(a);

adj\_t[b].push\_back(neg\_a);

adj\_t[a].push\_back(neg\_b);

}

**DP Convex Hull:**

#include <bits/stdc++.h>

using namespace std;

#define send {ios\_base::sync\_with\_stdio(false);}

#define help {cin.tie(NULL);}

#define f first

#define s second

#define getunique(v) {sort(v.begin(), v.end()); v.erase(unique(v.begin(), v.end()), v.end());}

typedef long long ll;

typedef long double lld;

typedef unsigned long long ull;

template<typename A> ostream& operator<<(ostream &cout, vector<A> const &v);

template<typename A, typename B> ostream& operator<<(ostream &cout, pair<A, B> const &p) { return cout << "(" << p.f << ", " << p.s << ")"; }

template<typename A> ostream& operator<<(ostream &cout, vector<A> const &v) {

cout << "["; for(int i = 0; i < v.size(); i++) {if (i) cout << ", "; cout << v[i];} return cout << "]";

}

template<typename A, typename B> istream& operator>>(istream& cin, pair<A, B> &p) {

cin >> p.first;

return cin >> p.second;

}

// mt19937\_64 rng(std::chrono::steady\_clock::now().time\_since\_epoch().count());

mt19937\_64 rng(61378913);

/\* usage - just do rng() \*/

void usaco(string filename) {

// #pragma message("be careful, freopen may be wrong")

freopen((filename + ".in").c\_str(), "r", stdin);

freopen((filename + ".out").c\_str(), "w", stdout);

}

// #include <atcoder/all>

// using namespace atcoder;

const lld pi = 3.14159265358979323846;

// const ll mod = 1000000007;

// const ll mod = 998244353;

// ll mod;

const ll is\_query = -(1LL<<62);

struct line {

ll m, b;

mutable function<const line\*()> succ;

bool operator<(const line& rhs) const {

if (rhs.b != is\_query) return m < rhs.m;

const line\* s = succ();

if (!s) return 0;

ll x = rhs.m;

return b - s->b < (s->m - m) \* x;

}

};

struct dynamic\_hull : public multiset<line> { // will maintain upper hull for maximum

const ll inf = LLONG\_MAX;

bool bad(iterator y) {

auto z = next(y);

if (y == begin()) {

if (z == end()) return 0;

return y->m == z->m && y->b <= z->b;

}

auto x = prev(y);

if (z == end()) return y->m == x->m && y->b <= x->b;

/\* compare two lines by slope, make sure denominator is not 0 \*/

ll v1 = (x->b - y->b);

if (y->m == x->m) v1 = x->b > y->b ? inf : -inf;

else v1 /= (y->m - x->m);

ll v2 = (y->b - z->b);

if (z->m == y->m) v2 = y->b > z->b ? inf : -inf;

else v2 /= (z->m - y->m);

return v1 >= v2;

}

void insert\_line(ll m, ll b) {

auto y = insert({ m, b });

y->succ = [=] { return next(y) == end() ? 0 : &\*next(y); };

if (bad(y)) { erase(y); return; }

while (next(y) != end() && bad(next(y))) erase(next(y));

while (y != begin() && bad(prev(y))) erase(prev(y));

}

ll eval(ll x) {

auto l = \*lower\_bound((line) { x, is\_query });

return l.m \* x + l.b;

}

};

ll n, m, k, q, l, r, x, y, z;

const ll template\_array\_size = 1e6 + 8432;

ll a[template\_array\_size];

ll b[template\_array\_size];

ll c[template\_array\_size];

string s, t;

ll ans = 0;

dynamic\_hull cht;

ll p[200005];

ll dp[200005];

void solve(int tc = 0) {

cin >> n;

for (ll i = 0; i < n; i++) cin >> a[i];

for (ll i = 0; i < n; i++) cin >> b[i];

ll r = 0;

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

r += a[i];

p[i] = r;

}

memset(dp, 0, sizeof(dp));

cht.insert\_line(0, 0);

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

dp[i] = p[i] \* b[i] - cht.eval(b[i]);

cht.insert\_line(p[i - 1], -dp[i]);

}

for (ll i = 0; i < n; i++) cout << dp[i] << " ";

}

**FFT Polymult:**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef vector<ll> vll;

typedef complex<double> cd;

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

int n = a.size();

if(n == 1)

return;

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

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

a0[i] = a[2 \* i];

a1[i] = a[2 \* i + 1];

}

fft(a0, invert);

fft(a1, invert);

double ang = 2 \* acos(-1.0) / n \* (invert ? -1 : 1);

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

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

}

}

vll mult(vll const& a, vll const& b){

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

int n = 1;

while(n < a.size() + b.size()){

n = n << 1;

}

fa.resize(n);

fb.resize(n);

vll c(n);

fft(fa, false);

fft(fb, false);

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

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

}

fft(fa, true);

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

c[i] = round(fa[i].real());

}

return c;

}

int main() {

int t;

cin >> t;

while(t--){

ll n, x, m;

cin >> n;

vll a, b;

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

scanf("%lld", &x);

a.push\_back(x);

}

cin >> m;

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

scanf("%lld", &x);

b.push\_back(x);

}

vll res = mult(a, b);

cout << n + m << endl;

cout << res[0];

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

printf(" %lld", res[i]);

}

cout << endl;

}

return 0;

}

**FFT:**

//Based on Solutions from VJudge, understood and re-implemented

#include <bits/stdc++.h>

using namespace std;

typedef complex<double> cd;

typedef long long ll;

typedef vector<ll> vll;

typedef vector<cd> vcd;

void fft(vcd &v, bool inverse){

int len = v.size();

if(len == 1){

return;

}

vcd ve(len / 2), vo(len / 2);

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

ve[i] = v[i \* 2];

vo[i] = v[i \* 2 + 1];

}

fft(ve, inverse);

fft(vo, inverse);

double angle = 2 \* acos(-1.0) \* (inverse \* -2 + 1) / (double)len;

cd a(1), omega(cos(angle), sin(angle));

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

double divide = inverse + 1;

v[i] = (ve[i] + a \* vo[i]) / divide;

v[i + len / 2] = (ve[i] - a \* vo[i]) / divide;

a \*= omega;

}

}

vll poly\_mult(vll const& v1, vll const& v2){

vcd c1(v1.begin(), v1.end()), c2(v2.begin(), v2.end());

int n = 1, tot = v1.size() + v2.size();

while(n < tot){

n <<= 1;

}

c1.resize(n);

c2.resize(n);

vll res(n);

fft(c1, false);

fft(c2, false);

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

c1[i] \*= c2[i];

}

fft(c1, true);

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

res[i] = round(c1[i].real());

}

return res;

}

int main(){

}

**AP:**

// A Java program to find articulation

// points in an undirected graph

import java.util.\*;

class Graph {

static int time;

static void addEdge(ArrayList<ArrayList<Integer> > adj, int u, int v)

{

adj.get(u).add(v);

adj.get(v).add(u);

}

static void APUtil(ArrayList<ArrayList<Integer> > adj, int u,

boolean visited[], int disc[], int low[],

int parent, boolean isAP[])

{

// Count of children in DFS Tree

int children = 0;

// Mark the current node as visited

visited[u] = true;

// Initialize discovery time and low value

disc[u] = low[u] = ++time;

// Go through all vertices adjacent to this

for (Integer v : adj.get(u)) {

// If v is not visited yet, then make it a child of u

// in DFS tree and recur for it

if (!visited[v]) {

children++;

APUtil(adj, v, visited, disc, low, u, isAP);

// Check if the subtree rooted with v has

// a connection to one of the ancestors of u

low[u] = Math.min(low[u], low[v]);

// If u is not root and low value of one of

// its child is more than discovery value of u.

if (parent != -1 && low[v] >= disc[u])

isAP[u] = true;

}

// Update low value of u for parent function calls.

else if (v != parent)

low[u] = Math.min(low[u], disc[v]);

}

// If u is root of DFS tree and has two or more children.

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

isAP[u] = true;

}

static void AP(ArrayList<ArrayList<Integer> > adj, int V)

{

boolean[] visited = new boolean[V];

int[] disc = new int[V];

int[] low = new int[V];

boolean[] isAP = new boolean[V];

int time = 0, par = -1;

// Adding this loop so that the

// code works even if we are given

// disconnected graph

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

if (visited[u] == false)

APUtil(adj, u, visited, disc, low, par, isAP);

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

if (isAP[u] == true)

System.out.print(u + " ");

System.out.println();

}

public static void main(String[] args)

{

// Creating first example graph

int V = 5;

ArrayList<ArrayList<Integer> > adj1 =

new ArrayList<ArrayList<Integer> >(V);

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

adj1.add(new ArrayList<Integer>());

addEdge(adj1, 1, 0);

addEdge(adj1, 0, 2);

addEdge(adj1, 2, 1);

addEdge(adj1, 0, 3);

addEdge(adj1, 3, 4);

System.out.println("Articulation points in first graph");

AP(adj1, V);

// Creating second example graph

V = 4;

ArrayList<ArrayList<Integer> > adj2 =

new ArrayList<ArrayList<Integer> >(V);

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

adj2.add(new ArrayList<Integer>());

addEdge(adj2, 0, 1);

addEdge(adj2, 1, 2);

addEdge(adj2, 2, 3);

System.out.println("Articulation points in second graph");

AP(adj2, V);

// Creating third example graph

V = 7;

ArrayList<ArrayList<Integer> > adj3 =

new ArrayList<ArrayList<Integer> >(V);

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

adj3.add(new ArrayList<Integer>());

addEdge(adj3, 0, 1);

addEdge(adj3, 1, 2);

addEdge(adj3, 2, 0);

addEdge(adj3, 1, 3);

addEdge(adj3, 1, 4);

addEdge(adj3, 1, 6);

addEdge(adj3, 3, 5);

addEdge(adj3, 4, 5);

System.out.println("Articulation points in third graph");

AP(adj3, V);

}

**BCC:**

//Based on https://leetcode.cn/problems/s5kipK/solution/by-tsreaper-z8by/

//Understood, translated to Java, and improved for scenarios with multiple connected components

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Solution {

private static long MOD = 1000000007L;

private static int INF = Integer.MAX\_VALUE / 2;

private static void print(long[] lwler){

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

System.out.print(lwler[i]+" ");

}

System.out.println();

}

private static void print(int[] lwler){

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

System.out.print(lwler[i]+" ");

}

System.out.println();

}

public static Stack<Integer> stack;

public static List<List<Integer>> dcc;

public static List<List<Integer>> edges;

public static int[] dfn, low;

public static boolean[] cut;

public static int n, st, time;

public static void tarjan(int cur){

dfn[cur] = ++time;

low[cur] = time;

int flag = 0;

stack.add(cur);

for(int ne : edges.get(cur)){

if(dfn[ne] == Integer.MAX\_VALUE / 2){

tarjan(ne);

low[cur] = Math.min(low[cur], low[ne]);

if(dfn[cur] <= low[ne]){

++flag;

int t;

if(cur != st || flag > 1){

cut[cur] = true;

}

dcc.add(new ArrayList<>());

do{

t = stack.get(stack.size() - 1);

stack.pop();

dcc.get(dcc.size() - 1).add(t);

} while(t != ne);

dcc.get(dcc.size() - 1).add(cur);

}

}

else{

low[cur] = Math.min(low[cur], dfn[ne]);

}

}

}

public static long minimumCost(int[] cost, int[][] roads) {

n = cost.length;

if(n == 1){

return cost[0];

}

stack = new Stack<>();

dcc = new ArrayList<>();

edges = new ArrayList<>();

st = 0;

time = 0;

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

edges.add(new ArrayList<>());

}

for(int[] r : roads){

edges.get(r[0]).add(r[1]);

edges.get(r[1]).add(r[0]);

}

dfn = new int[n];

low = new int[n];

cut = new boolean[n];

Arrays.fill(dfn, Integer.MAX\_VALUE / 2);

Arrays.fill(low, Integer.MAX\_VALUE / 2);

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

if(dfn[i] == Integer.MAX\_VALUE / 2){

st = i;

tarjan(i);

}

}

if(dcc.size() == 1){

int res = Integer.MAX\_VALUE;

for(int ele : cost)

res = Math.min(res, ele);

return (long)res;

}

List<Integer> leaves = new ArrayList<>();

for(List<Integer> l : dcc){

int cut\_cnt = 0, min = Integer.MAX\_VALUE;

for(int x : l){

if(cut[x])

++cut\_cnt;

else

min = Math.min(min, cost[x]);

}

if(cut\_cnt == 1)

leaves.add(min);

}

Collections.sort(leaves);

long res = 0L;

for(int i = 0; i < leaves.size() - 1; ++i){

res += leaves.get(i);

}

return res;

}

public static void main(String[] args){

Scanner in = new Scanner(new BufferedReader(new InputStreamReader(System.in)));

}

}

**Blossom:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class Solution

{

static int lca(int[] match, int[] base, int[] p, int a, int b)

{

boolean[] used = new boolean[match.length];

while (true)

{

a = base[a];

used[a] = true;

if (match[a] == -1)

break;

a = p[match[a]];

}

while (true)

{

b = base[b];

if (used[b])

return b;

b = p[match[b]];

}

}

static void markPath(int[] match, int[] base, boolean[] blossom, int[] p,

int v, int b, int children)

{

for (; base[v] != b; v = p[match[v]])

{

blossom[base[v]] = blossom[base[match[v]]] = true;

p[v] = children;

children = match[v];

}

}

static int findPath(List<Integer>[] graph, int[] match, int[] p, int root)

{

int n = graph.length;

boolean[] used = new boolean[n];

Arrays.fill(p, -1);

int[] base = new int[n];

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

base[i] = i;

used[root] = true;

int qh = 0;

int qt = 0;

int[] q = new int[n];

q[qt++] = root;

while (qh < qt)

{

int v = q[qh++];

for (int to : graph[v])

{

if (base[v] == base[to] || match[v] == to)

continue;

if (to == root || match[to] != -1 && p[match[to]] != -1)

{

int curbase = lca(match, base, p, v, to);

boolean[] blossom = new boolean[n];

markPath(match, base, blossom, p, v, curbase, to);

markPath(match, base, blossom, p, to, curbase, v);

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

if (blossom[base[i]])

{

base[i] = curbase;

if (!used[i])

{

used[i] = true;

q[qt++] = i;

}

}

}

else if (p[to] == -1)

{

p[to] = v;

if (match[to] == -1)

return to;

to = match[to];

used[to] = true;

q[qt++] = to;

}

}

}

return -1;

}

public static int maxMatching(List<Integer>[] graph)

{

int n = graph.length;

int[] match = new int[n];

Arrays.fill(match, -1);

int[] p = new int[n];

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

{

if (match[i] == -1)

{

int v = findPath(graph, match, p, i);

while (v != -1)

{

int pv = p[v];

int ppv = match[pv];

match[v] = pv;

match[pv] = v;

v = ppv;

}

}

}

int matches = 0;

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

if (match[i] != -1)

++matches;

return matches / 2;

}

@SuppressWarnings("unchecked")

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of vertices: ");

int v = sc.nextInt();

System.out.println("Enter the number of edges: ");

int e = sc.nextInt();

List<Integer>[] g = new List[v];

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

{

g[i] = new ArrayList<Integer>();

}

System.out.println("Enter all the edges: <from> <to>");

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

{

g[sc.nextInt()].add(sc.nextInt());

}

System.out.println("Maximum matching for the given graph is: "

+ maxMatching(g));

sc.close();

}

}

**Bridge Finding:**

// A Java program to find bridges in a given undirected graph

import java.io.\*;

import java.util.\*;

import java.util.LinkedList;

// This class represents a undirected graph using adjacency list

// representation

class Graph

{

private int V; // No. of vertices

// Array of lists for Adjacency List Representation

private LinkedList<Integer> adj[];

int time = 0;

static final int NIL = -1;

// Constructor

@SuppressWarnings("unchecked")Graph(int v)

{

V = v;

adj = new LinkedList[v];

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

adj[i] = new LinkedList();

}

// Function to add an edge into the graph

void addEdge(int v, int w)

{

adj[v].add(w); // Add w to v's list.

adj[w].add(v); //Add v to w's list

}

// A recursive function that finds and prints bridges

// using DFS traversal

// u --> The vertex to be visited next

// visited[] --> keeps track of visited vertices

// disc[] --> Stores discovery times of visited vertices

// parent[] --> Stores parent vertices in DFS tree

void bridgeUtil(int u, boolean visited[], int disc[],

int low[], int parent[])

{

// Mark the current node as visited

visited[u] = true;

// Initialize discovery time and low value

disc[u] = low[u] = ++time;

// Go through all vertices adjacent to this

Iterator<Integer> i = adj[u].iterator();

while (i.hasNext())

{

int v = i.next(); // v is current adjacent of u

// If v is not visited yet, then make it a child

// of u in DFS tree and recur for it.

// If v is not visited yet, then recur for it

if (!visited[v])

{

parent[v] = u;

bridgeUtil(v, visited, disc, low, parent);

// Check if the subtree rooted with v has a

// connection to one of the ancestors of u

low[u] = Math.min(low[u], low[v]);

// If the lowest vertex reachable from subtree

// under v is below u in DFS tree, then u-v is

// a bridge

if (low[v] > disc[u])

System.out.println(u+" "+v);

}

// Update low value of u for parent function calls.

else if (v != parent[u])

low[u] = Math.min(low[u], disc[v]);

}

}

// DFS based function to find all bridges. It uses recursive

// function bridgeUtil()

void bridge()

{

// Mark all the vertices as not visited

boolean visited[] = new boolean[V];

int disc[] = new int[V];

int low[] = new int[V];

int parent[] = new int[V];

// Initialize parent and visited, and ap(articulation point)

// arrays

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

{

parent[i] = NIL;

visited[i] = false;

}

// Call the recursive helper function to find Bridges

// in DFS tree rooted with vertex 'i'

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

if (visited[i] == false)

bridgeUtil(i, visited, disc, low, parent);

}

public static void main(String args[])

{

// Create graphs given in above diagrams

System.out.println("Bridges in first graph ");

Graph g1 = new Graph(5);

g1.addEdge(1, 0);

g1.addEdge(0, 2);

g1.addEdge(2, 1);

g1.addEdge(0, 3);

g1.addEdge(3, 4);

g1.bridge();

System.out.println();

System.out.println("Bridges in Second graph");

Graph g2 = new Graph(4);

g2.addEdge(0, 1);

g2.addEdge(1, 2);

g2.addEdge(2, 3);

g2.bridge();

System.out.println();

System.out.println("Bridges in Third graph ");

Graph g3 = new Graph(7);

g3.addEdge(0, 1);

g3.addEdge(1, 2);

g3.addEdge(2, 0);

g3.addEdge(1, 3);

g3.addEdge(1, 4);

g3.addEdge(1, 6);

g3.addEdge(3, 5);

g3.addEdge(4, 5);

g3.bridge();

}

}

**Dinics:**

// Based on https://github.com/williamfiset/Algorithms/tree/master/src/main/java/com/williamfiset/algorithms/graphtheory/networkflow/examples

// Understood and optimized / shortened for Competitive Programming

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Solution {

private static long MOD = 1000000007L;

private static void print(long[] lwler){

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

System.out.print(lwler[i]+" ");

}

System.out.println();

}

private static void print(int[] lwler){

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

System.out.print(lwler[i]+" ");

}

System.out.println();

}

private static class Edge{

public int from, to;

public long flow, capacity;

public Edge residual;

public Edge(int from, int to, long capacity){

this.from = from;

this.to = to;

this.capacity = capacity;

}

public void augment(long bottle){

flow += bottle;

residual.flow -= bottle;

}

}

private static class Dinics{

static final long INF = Long.MAX\_VALUE / 2;

int n, s, t;

public long maxFlow;

public List<Edge>[] graph;

private int[] level, next;

public Dinics(int n, int s, int t){

this.n = n;

this.s = s;

this.t = t;

level = new int[n];

next = new int[n];

graph = new ArrayList[n];

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

graph[i] = new ArrayList<Edge>();

}

}

public void addEdge(int from, int to, long capacity){

Edge e = new Edge(from, to, capacity);

Edge er = new Edge(to, from, 0);

e.residual = er;

er.residual = e;

graph[from].add(e);

graph[to].add(er);

}

private boolean bfs(){

Arrays.fill(level, -1);

Queue<Integer> q = new LinkedList<>();

q.add(s);

level[s] = 0;

while(!q.isEmpty()){

int cur = q.poll();

for(Edge e : graph[cur]){

long space = e.capacity - e.flow;

if(space > 0 && level[e.to] == -1){

level[e.to] = level[cur] + 1;

q.add(e.to);

}

}

}

return level[t] != -1;

}

private long dfs(int cur, long flow){

if(cur == t)

return flow;

int ns = graph[cur].size();

while(next[cur] < ns){

Edge e = graph[cur].get(next[cur]);

long space = e.capacity - e.flow;

if(space > 0 && level[e.to] == level[cur] + 1){

long bottle = dfs(e.to, Math.min(flow, space));

if(bottle > 0){

e.augment(bottle);

return bottle;

}

}

++next[cur];

}

return 0;

}

public void solve(){

while(bfs()){

Arrays.fill(next, 0);

long f = dfs(s, INF);

while(f != 0){

maxFlow += f;

f = dfs(s, INF);

}

}

}

}

public String tricoloring(int[] A){

int n = A.length;

int[] val = new int[3];

String[] colors = {"R","G","B"};

for(int i = 0; i < (int)Math.pow(3, n); ++i){

int tmp = i;

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

val[tmp % 3] += A[j];

tmp /= 3;

}

if(val[0] == val[1] && val[1] == val[2] && val[0] == val[2]){

tmp = i;

String S = "";

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

S += colors[tmp % 3];

tmp /= 3;

}

return S;

}

Arrays.fill(val, 0);

}

return "None";

}

public static void main(String[] args){

Scanner in = new Scanner(new BufferedReader(new InputStreamReader(System.in)));

}

}

**Heavy Light Decom:**

// For details, check Tools.wrrd in ../../Tools (Search for Heavy Light Chain in the file)

#include <iostream>

#include <ctime>

#include <cstdio>

#include <cctype>

namespace FastIO

{

char buf[1 << 21], buf2[1 << 21], a[20], \*p1 = buf, \*p2 = buf, hh = '\n';

int p, p3 = -1;

void read() {}

void print() {}

inline int getc()

{

return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ? EOF : \*p1++;

}

inline void flush()

{

fwrite(buf2, 1, p3 + 1, stdout), p3 = -1;

}

template <typename T, typename... T2>

inline void read(T &x, T2 &... oth)

{

int f = 0;

x = 0;

char ch = getc();

while (!isdigit(ch))

{

if (ch == '-')

f = 1;

ch = getc();

}

while (isdigit(ch))

{

x = x \* 10 + ch - 48;

ch = getc();

}

x = f ? -x : x;

read(oth...);

}

template <typename T, typename... T2>

inline void print(T x, T2... oth)

{

if (p3 > 1 << 20)

flush();

if (x < 0)

buf2[++p3] = 45, x = -x;

do

{

a[++p] = x % 10 + 48;

} while (x /= 10);

do

{

buf2[++p3] = a[p];

} while (--p);

buf2[++p3] = hh;

print(oth...);

}

} // namespace FastIO

#define read FastIO::read

#define print FastIO::print

//======================================

const int maxn = 1e5 + 5;

const int maxm = maxn \* 2;

struct E

{

int to, next;

} Edge[maxm];

int tot, Head[maxn];

inline void AddEdge(int u, int v)

{

Edge[tot] = (E){v, Head[u]};

Head[u] = tot++;

Edge[tot] = (E){u, Head[v]};

Head[v] = tot++;

}

int mod;

int v[maxn];

int fa[maxn], dep[maxn], siz[maxn], son[maxn];

void dfs1(int u, int f)

{

fa[u] = f;

dep[u] = dep[f] + 1;

siz[u] = 1;

int maxsize = -1;

for (int i = Head[u]; ~i; i = Edge[i].next)

{

int v = Edge[i].to;

if (v == f)

continue;

dfs1(v, u);

siz[u] += siz[v];

if (siz[v] > maxsize)

{

maxsize = siz[v];

son[u] = v;

}

}

}

int tim, dfn[maxn], top[maxn], w[maxn];

void dfs2(int u, int t)

{

dfn[u] = ++tim;

top[u] = t;

w[tim] = v[u];

if (!son[u])

return;

dfs2(son[u], t);

for (int i = Head[u]; ~i; i = Edge[i].next)

{

int v = Edge[i].to;

if (v == fa[u] || v == son[u])

continue;

dfs2(v, v);

}

}

struct Node

{

int l, r, f, val;

} sgt[maxn \* 4];

inline int ls(int k) { return k << 1; }

inline int rs(int k) { return k << 1 | 1; }

inline void pushup(int k) { sgt[k].val = (sgt[ls(k)].val + sgt[rs(k)].val) % mod; }

inline void pushdown(int k)

{

sgt[ls(k)].f += sgt[k].f;

sgt[rs(k)].f += sgt[k].f;

sgt[ls(k)].val += (sgt[ls(k)].r - sgt[ls(k)].l + 1) \* sgt[k].f % mod;

sgt[rs(k)].val += (sgt[rs(k)].r - sgt[rs(k)].l + 1) \* sgt[k].f % mod;

sgt[k].f = 0;

}

void build(int l, int r, int k = 1)

{

sgt[k].l = l, sgt[k].r = r;

if (l == r)

{

sgt[k].val = w[l] % mod;

return;

}

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

build(l, m, ls(k));

build(m + 1, r, rs(k));

pushup(k);

}

void modify(int x, int y, int z, int k = 1)

{

int l = sgt[k].l, r = sgt[k].r;

if (x <= l && y >= r)

{

sgt[k].f += z;

sgt[k].val += (r - l + 1) \* z;

sgt[k].val %= mod;

return;

}

if (sgt[k].f)

pushdown(k);

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

if (x <= m)

modify(x, y, z, ls(k));

if (y > m)

modify(x, y, z, rs(k));

pushup(k);

}

int query(int x, int y, int k = 1)

{

int l = sgt[k].l, r = sgt[k].r;

if (x <= l && y >= r)

return sgt[k].val;

if (sgt[k].f)

pushdown(k);

int sum = 0, m = (l + r) >> 1;

if (x <= m)

sum += query(x, y, ls(k));

if (y > m)

sum += query(x, y, rs(k));

return sum % mod;

}

void mchain(int x, int y, int z)

{

z %= mod;

while (top[x] != top[y])

{

if (dep[top[x]] < dep[top[y]])

std::swap(x, y);

modify(dfn[top[x]], dfn[x], z);

x = fa[top[x]];

}

if (dep[x] > dep[y])

std::swap(x, y);

modify(dfn[x], dfn[y], z);

}

int qchain(int x, int y)

{

int ret = 0;

while (top[x] != top[y])

{

if (dep[top[x]] < dep[top[y]])

std::swap(x, y);

ret += query(dfn[top[x]], dfn[x]);

x = fa[top[x]];

}

if (dep[x] > dep[y])

std::swap(x, y);

ret += query(dfn[x], dfn[y]);

return ret % mod;

}

inline void mson(int x, int z)

{

modify(dfn[x], dfn[x] + siz[x] - 1, z);

}

inline int qson(int x)

{

return query(dfn[x], dfn[x] + siz[x] - 1);

}

#include <cstring>

int main(int argc, char const \*argv[])

{

#ifndef ONLINE\_JUDGE

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

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

#endif

clock\_t c1 = clock();

//======================================

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

int n, m, r;

read(n, m, r, mod);

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

read(v[i]);

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

{

int u, v;

read(u, v);

AddEdge(u, v);

}

dfs1(r, r);

dfs2(r, r);

build(1, n);

while (m--)

{

int opt, x, y, z;

read(opt);

switch (opt)

{

case 1:

read(x, y, z);

mchain(x, y, z);

break;

case 2:

read(x, y);

print(qchain(x, y));

break;

case 3:

read(x, z);

mson(x, z);

break;

case 4:

read(x);

print(qson(x));

break;

}

}

//======================================

FastIO::flush();

std::cerr << "Time:" << clock() - c1 << "ms" << std::endl;

return 0;

}

**Hungarian:**

#include<bits/stdc++.h>

using namespace std;

const int N = 509;

/\* Complexity: O(n^3) but optimized

It finds minimum cost maximum matching.

For finding maximum cost maximum matching

add -cost and return -matching()

1-indexed \*/

struct Hungarian {

long long c[N][N], fx[N], fy[N], d[N];

int l[N], r[N], arg[N], trace[N];

queue<int> q;

int start, finish, n;

const long long inf = 1e18;

Hungarian() {}

Hungarian(int n1, int n2): n(max(n1, n2)) {

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

fy[i] = l[i] = r[i] = 0;

for (int j = 1; j <= n; ++j) c[i][j] = inf; // make it 0 for maximum cost matching (not necessarily with max count of matching)

}

}

void add\_edge(int u, int v, long long cost) {

c[u][v] = min(c[u][v], cost);

}

inline long long getC(int u, int v) {

return c[u][v] - fx[u] - fy[v];

}

void initBFS() {

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

q.push(start);

for (int i = 0; i <= n; ++i) trace[i] = 0;

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

d[v] = getC(start, v);

arg[v] = start;

}

finish = 0;

}

void findAugPath() {

while (!q.empty()) {

int u = q.front();

q.pop();

for (int v = 1; v <= n; ++v) if (!trace[v]) {

long long w = getC(u, v);

if (!w) {

trace[v] = u;

if (!r[v]) {

finish = v;

return;

}

q.push(r[v]);

}

if (d[v] > w) {

d[v] = w;

arg[v] = u;

}

}

}

}

void subX\_addY() {

long long delta = inf;

for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v] < delta) {

delta = d[v];

}

// Rotate

fx[start] += delta;

for (int v = 1; v <= n; ++v) if(trace[v]) {

int u = r[v];

fy[v] -= delta;

fx[u] += delta;

} else d[v] -= delta;

for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v]) {

trace[v] = arg[v];

if (!r[v]) {

finish = v;

return;

}

q.push(r[v]);

}

}

void Enlarge() {

do {

int u = trace[finish];

int nxt = l[u];

l[u] = finish;

r[finish] = u;

finish = nxt;

} while (finish);

}

long long maximum\_matching() {

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

fx[u] = c[u][1];

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

fx[u] = min(fx[u], c[u][v]);

}

}

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

fy[v] = c[1][v] - fx[1];

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

fy[v] = min(fy[v], c[u][v] - fx[u]);

}

}

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

start = u;

initBFS();

while (!finish) {

findAugPath();

if (!finish) subX\_addY();

}

Enlarge();

}

long long ans = 0;

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

if (c[i][l[i]] != inf) ans += c[i][l[i]];

else l[i] = 0;

}

return ans;

}

};

int32\_t main() {

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

cin.tie(0);

int n1, n2, m;

cin >> n1 >> n2 >> m;

Hungarian M(n1, n2);

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

int u, v, w;

cin >> u >> v >> w;

M.add\_edge(u, v, -w);

}

cout << -M.maximum\_matching() << '\n';

for (int i = 1; i <= n1; i++) cout << M.l[i] << ' ';

return 0;

}

**Min-Cost Flow:**

// This code uses new C++17 structured binding

// use this compiler setting "g++ -O2 -std=gnu++17 {cpp17file}"

// Disclaimer: This code is a hybrid between old CP1-2-3 implementation of

// Edmonds Karp's algorithm -- re-written in OOP fashion and the fast

// Dinic's algorithm implementation by

// https://github.com/jaehyunp/stanfordacm/blob/master/code/Dinic.cc

// This code is written in modern C++17 standard

// We replace BFS with SPFA

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef tuple<int, ll, ll, ll> edge;

typedef vector<int> vi;

typedef vector<ll> vll;

const ll INF = 1e18; // INF = 1e18, not 2^63-1 to avoid overflow

class min\_cost\_max\_flow {

private:

int V;

ll total\_cost;

vector<edge> EL;

vector<vi> AL;

vll d;

vi last, vis;

bool SPFA(int s, int t) { // SPFA to find augmenting path in residual graph

d.assign(V, INF); d[s] = 0; vis[s] = 1;

queue<int> q({s});

while (!q.empty()) {

int u = q.front(); q.pop(); vis[u] = 0;

for (auto &idx : AL[u]) { // explore neighbors of u

auto &[v, cap, flow, cost] = EL[idx]; // stored in EL[idx]

if ((cap-flow > 0) && (d[v] > d[u] + cost)) { // positive residual edge

d[v] = d[u]+cost;

if(!vis[v]) q.push(v), vis[v] = 1;

}

}

}

return d[t] != INF; // has an augmenting path

}

ll DFS(int u, int t, ll f = INF) { // traverse from s->t

if ((u == t) || (f == 0)) return f;

vis[u] = 1;

for (int &i = last[u]; i < (int)AL[u].size(); ++i) { // from last edge

auto &[v, cap, flow, cost] = EL[AL[u][i]];

if (!vis[v] && d[v] == d[u]+cost) { // in current layer graph

if (ll pushed = DFS(v, t, min(f, cap-flow))) {

total\_cost += pushed \* cost;

flow += pushed;

auto &[rv, rcap, rflow, rcost] = EL[AL[u][i]^1]; // back edge

rflow -= pushed;

vis[u] = 0;

return pushed;

}

}

}

vis[u] = 0;

return 0;

}

public:

min\_cost\_max\_flow(int initialV) : V(initialV), total\_cost(0) {

EL.clear();

AL.assign(V, vi());

vis.assign(V, 0);

}

// if you are adding a bidirectional edge u<->v with weight w into your

// flow graph, set directed = false (default value is directed = true)

void add\_edge(int u, int v, ll w, ll c, bool directed = true) {

if (u == v) return; // safeguard: no self loop

EL.emplace\_back(v, w, 0, c); // u->v, cap w, flow 0, cost c

AL[u].push\_back(EL.size()-1); // remember this index

EL.emplace\_back(u, 0, 0, -c); // back edge

AL[v].push\_back(EL.size()-1); // remember this index

if (!directed) add\_edge(v, u, w, c); // add again in reverse

}

pair<ll, ll> mcmf(int s, int t) {

ll mf = 0; // mf stands for max\_flow

while (SPFA(s, t)) { // an O(V^2\*E) algorithm

last.assign(V, 0); // important speedup

while (ll f = DFS(s, t)) // exhaust blocking flow

mf += f;

}

return {mf, total\_cost};

}

};

int main() {

int V, E, s, t; scanf("%d %d %d %d", &V, &E, &s, &t);

min\_cost\_max\_flow mf(V);

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

int u, v, w, c; scanf("%d %d %d %d", &u, &v, &w, &c);

mf.add\_edge(u, v, w, c); // default: directed edge

}

pair<ll, ll> res = mf.mcmf(s, t);

printf("%lld %lld\n", res.first, res.second);

return 0;

}

**O(1) LCA:**

// C++ program to demonstrate LCA of n-ary tree

// in constant time.

#include "bits/stdc++.h"

using namespace std;

#define sz 101

vector < int > adj[sz]; // stores the tree

vector < int > euler; // tracks the eulerwalk

vector < int > depthArr; // depth for each node corresponding

// to eulerwalk

int FAI[sz]; // stores first appearance index of every node

int level[sz]; // stores depth for all nodes in the tree

int ptr; // pointer to euler walk

int dp[sz][18]; // sparse table

int logn[sz]; // stores log values

int p2[20]; // stores power of 2

void buildSparseTable(int n)

{

// initializing sparse table

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

// filling base case values

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

dp[i-1][0] = (depthArr[i]>depthArr[i-1])?i-1:i;

// dp to fill sparse table

for (int l=1; l<15; l++)

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

if (dp[i][l-1]!=-1 and dp[i+p2[l-1]][l-1]!=-1)

dp[i][l] =

(depthArr[dp[i][l-1]]>depthArr[dp[i+p2[l-1]][l-1]])?

dp[i+p2[l-1]][l-1] : dp[i][l-1];

else

break;

}

int query(int l,int r)

{

int d = r-l;

int dx = logn[d];

if (l==r) return l;

if (depthArr[dp[l][dx]] > depthArr[dp[r-p2[dx]][dx]])

return dp[r-p2[dx]][dx];

else

return dp[l][dx];

}

void preprocess()

{

// memorizing powers of 2

p2[0] = 1;

for (int i=1; i<18; i++)

p2[i] = p2[i-1]\*2;

// memorizing all log(n) values

int val = 1,ptr=0;

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

{

logn[i] = ptr-1;

if (val==i)

{

val\*=2;

logn[i] = ptr;

ptr++;

}

}

}

/\*\*

\* Euler Walk ( preorder traversal)

\* converting tree to linear depthArray

\* Time Complexity : O(n)

\* \*/

void dfs(int cur,int prev,int dep)

{

// marking FAI for cur node

if (FAI[cur]==-1)

FAI[cur] = ptr;

level[cur] = dep;

// pushing root to euler walk

euler.push\_back(cur);

// incrementing euler walk pointer

ptr++;

for (auto x:adj[cur])

{

if (x != prev)

{

dfs(x,cur,dep+1);

// pushing cur again in backtrack

// of euler walk

euler.push\_back(cur);

// increment euler walk pointer

ptr++;

}

}

}

// Create Level depthArray corresponding

// to the Euler walk Array

void makeArr()

{

for (auto x : euler)

depthArr.push\_back(level[x]);

}

int LCA(int u,int v)

{

// trivial case

if (u==v)

return u;

if (FAI[u] > FAI[v])

swap(u,v);

// doing RMQ in the required range

return euler[query(FAI[u], FAI[v])];

}

void addEdge(int u,int v)

{

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

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

}

int main(int argc, char const \*argv[])

{

// constructing the described tree

int numberOfNodes = 8;

addEdge(1,2);

addEdge(1,3);

addEdge(2,4);

addEdge(2,5);

addEdge(2,6);

addEdge(3,7);

addEdge(3,8);

// performing required precalculations

preprocess();

// doing the Euler walk

ptr = 0;

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

dfs(1,0,0);

// creating depthArray corresponding to euler[]

makeArr();

// building sparse table

buildSparseTable(depthArr.size());

cout << "LCA(6,7) : " << LCA(6,7) << "\n";

cout << "LCA(6,4) : " << LCA(6,4) << "\n";

return 0;

}

**SCC:**

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Main {

static long MOD = 1000000007L;

public static void print(long[] aa){

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

System.out.print(aa[i] + "\t");

System.out.println();

}

public static void print(int[] aa){

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

System.out.print(aa[i] + "\t");

System.out.println();

}

public static class Pair{

int a;

int b;

int c;

public Pair(int a, int b, int c){

this.a = a;

this.b = b;

this.c = c;

}

}

public static int[] dir = {0,1,0,-1,0};

public static int[] dfn, low;

public static List<List<Integer>> scc;

public static int time;

public static HashSet<Integer>[] map;

public static Stack<Integer> stack;

public static boolean[] stkItem;

public static void tarjan(int cur){

//System.out.println(cur);

++time;

dfn[cur] = time;

low[cur] = time;

stack.add(cur);

stkItem[cur] = true;

//System.out.println(cur + " " + stack);

for(int ne : map[cur]){

if(dfn[ne] == Integer.MAX\_VALUE / 2){

tarjan(ne);

low[cur] = Math.min(low[ne], low[cur]);

}

else if(stkItem[ne]){

//TODO: check

low[cur] = Math.min(low[cur], dfn[ne]);

}

}

int t = 0;

//System.out.println(cur + " " + stack);

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

List<Integer> l = new ArrayList<>();

while(stack.get(stack.size() - 1) != cur){

t = stack.get(stack.size() - 1);

stkItem[t] = false;

l.add(t);

stack.pop();

}

stack.pop();

l.add(cur);

scc.add(l);

stkItem[cur] = false;

}

}

public static void main(String[] args){

Scanner in = new Scanner(new BufferedReader(new InputStreamReader(System.in)));

int n = in.nextInt(), m = in.nextInt();

map = new HashSet[n + 1];

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

map[i] = new HashSet<>();

}

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

map[in.nextInt()].add(in.nextInt());

}

stack = new Stack<>();

scc = new ArrayList<>();

dfn = new int[n + 1];

low = new int[n + 1];

stkItem = new boolean[n + 1];

Arrays.fill(dfn, Integer.MAX\_VALUE / 2);

Arrays.fill(low, Integer.MAX\_VALUE / 2);

time = 0;

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

if(dfn[i] == Integer.MAX\_VALUE / 2){

tarjan(i);

}

}

}

}

**DCC:**

#include <bits/stdc++.h>

using namespace std;

class Solution {

const int S = 0;

int n;

vector<vector<int>> e;

vector<bool> isCut;

vector<int> dfn, low;

int clk = 0;

stack<int> stk;

// 所有点双连通分量

vector<vector<int>> dcc;

void tarjan(int sn) {

dfn[sn] = low[sn] = ++clk;

stk.push(sn);

int flag = 0;

for (int fn : e[sn]) {

if (!dfn[fn]) {

tarjan(fn);

low[sn] = min(low[sn], low[fn]);

if (low[fn] >= dfn[sn]) {

flag++;

if (sn != S || flag > 1) isCut[sn] = true;

int t;

dcc.push\_back(vector<int>());

do {

t = stk.top(); stk.pop();

dcc.back().push\_back(t);

} while (t != fn);

dcc.back().push\_back(sn);

}

} else low[sn] = min(low[sn], dfn[fn]);

}

}

public:

long long minimumCost(vector<int>& cost, vector<vector<int>>& roads) {

n = cost.size();

if (n == 1) return cost[0];

e = vector<vector<int>>(n);

for (auto &r : roads) e[r[0]].push\_back(r[1]), e[r[1]].push\_back(r[0]);

isCut = vector<bool>(n);

dfn = low = vector<int>(n);

tarjan(S);

}

};

**Segment Tree Persistent Lazy:**

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5 + 9, mod = 998244353;

template <const int32\_t MOD>

struct modint {

int32\_t value;

modint() = default;

modint(int32\_t value\_) : value(value\_) {}

inline modint<MOD> operator + (modint<MOD> other) const { int32\_t c = this->value + other.value; return modint<MOD>(c >= MOD ? c - MOD : c); }

inline modint<MOD> operator - (modint<MOD> other) const { int32\_t c = this->value - other.value; return modint<MOD>(c < 0 ? c + MOD : c); }

inline modint<MOD> operator \* (modint<MOD> other) const { int32\_t c = (int64\_t)this->value \* other.value % MOD; return modint<MOD>(c < 0 ? c + MOD : c); }

inline modint<MOD> & operator += (modint<MOD> other) { this->value += other.value; if (this->value >= MOD) this->value -= MOD; return \*this; }

inline modint<MOD> & operator -= (modint<MOD> other) { this->value -= other.value; if (this->value < 0) this->value += MOD; return \*this; }

inline modint<MOD> & operator \*= (modint<MOD> other) { this->value = (int64\_t)this->value \* other.value % MOD; if (this->value < 0) this->value += MOD; return \*this; }

inline modint<MOD> operator - () const { return modint<MOD>(this->value ? MOD - this->value : 0); }

modint<MOD> pow(uint64\_t k) const { modint<MOD> x = \*this, y = 1; for (; k; k >>= 1) { if (k & 1) y \*= x; x \*= x; } return y; }

modint<MOD> inv() const { return pow(MOD - 2); } // MOD must be a prime

inline modint<MOD> operator / (modint<MOD> other) const { return \*this \* other.inv(); }

inline modint<MOD> operator /= (modint<MOD> other) { return \*this \*= other.inv(); }

inline bool operator == (modint<MOD> other) const { return value == other.value; }

inline bool operator != (modint<MOD> other) const { return value != other.value; }

inline bool operator < (modint<MOD> other) const { return value < other.value; }

inline bool operator > (modint<MOD> other) const { return value > other.value; }

};

template <int32\_t MOD> modint<MOD> operator \* (int64\_t value, modint<MOD> n) { return modint<MOD>(value) \* n; }

template <int32\_t MOD> modint<MOD> operator \* (int32\_t value, modint<MOD> n) { return modint<MOD>(value % MOD) \* n; }

template <int32\_t MOD> istream & operator >> (istream & in, modint<MOD> &n) { return in >> n.value; }

template <int32\_t MOD> ostream & operator << (ostream & out, modint<MOD> n) { return out << n.value; }

using mint = modint<mod>;

mint s[N], P = 37;

struct PST {

#define lc t[cur].l

#define rc t[cur].r

struct node {

int l = 0, r = 0, lazy = 0, p = 0;

mint val = 0;

} t[300 \* N];

int T = 0;

int build(int b, int e) {

int cur = ++T;

if(b == e) return cur;

int mid = b + e >> 1;

lc = build(b, mid);

rc = build(mid + 1, e);

return cur;

}

int push(int pre, int b, int e, int x = 0) {

int cur = ++T;

t[cur] = t[pre];

t[cur].lazy ^= x;

if(t[cur].lazy) {

t[cur].val = s[e] - s[b - 1] - t[cur].val;

t[cur].p ^= 1;

if(b != e) {

lc = ++T;

rc = ++T;

t[lc] = t[t[pre].l];

t[rc] = t[t[pre].r];

t[lc].lazy ^= 1, t[rc].lazy ^= 1;

}

}

t[cur].lazy = 0;

return cur;

}

int upd(int pre, int b, int e, int i, int j) {

int cur = push(pre, b, e);

if(b > j || e < i) return cur;

if(i <= b && e <= j) {

cur = push(cur, b, e, 1);

return cur;

}

int mid = b + e >> 1;

lc = upd(lc, b, mid, i, j);

rc = upd(rc, mid + 1, e, i, j);

t[cur].val = t[lc].val + t[rc].val;

return cur;

}

int cmp(int cur, int oth, int b, int e) {

cur = push(cur, b, e);

oth = push(oth, b, e);

if(t[cur].val == t[oth].val) return 0;

if(b == e) {

int x = t[cur].p, y = t[oth].p;

if(x == y) return 0;

else if(x > y) return 1;

else return 2;

}

int mid = b + e >> 1;

int p = cmp(lc, t[oth].l, b, mid);

if(p) return p;

return cmp(rc, t[oth].r, mid + 1, e);

}

void print(int cur, int b, int e) {

cur = push(cur, b, e);

if(b == e) {

cout << t[cur].p;

return;

}

int mid = b + e >> 1;

print(lc, b, mid);

print(rc, mid + 1, e);

}

} t;

int root[N];

int32\_t main() {

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

cin.tie(0);

for(int i = 1; i < N; i++) s[i] = s[i - 1] + P.pow(i);

int n, q;

cin >> n >> q;

root[0] = t.build(1, n);

int ans = 0;

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

int l, r;

cin >> l >> r;

root[i] = t.upd(root[i - 1], 1, n, l, r);

if(t.cmp(root[ans], root[i], 1, n) == 2) ans = i;

}

t.print(root[ans], 1, n);

return 0;

}

**Segment Tree Lazy:**

|  |
| --- |
| #include<bits/stdc++.h> |
|  | using namespace std; |
|  |  |
|  | const int N = 5e5 + 9; |
|  | int a[N]; |
|  | struct ST { |
|  | #define lc (n << 1) |
|  | #define rc ((n << 1) | 1) |
|  | long long t[4 \* N], lazy[4 \* N]; |
|  | ST() { |
|  | memset(t, 0, sizeof t); |
|  | memset(lazy, 0, sizeof lazy); |
|  | } |
|  | inline void push(int n, int b, int e) { |
|  | if (lazy[n] == 0) return; |
|  | t[n] = t[n] + lazy[n] \* (e - b + 1); |
|  | if (b != e) { |
|  | lazy[lc] = lazy[lc] + lazy[n]; |
|  | lazy[rc] = lazy[rc] + lazy[n]; |
|  | } |
|  | lazy[n] = 0; |
|  | } |
|  | inline long long combine(long long a,long long b) { |
|  | return a + b; |
|  | } |
|  | inline void pull(int n) { |
|  | t[n] = t[lc] + t[rc]; |
|  | } |
|  | void build(int n, int b, int e) { |
|  | lazy[n] = 0; |
|  | if (b == e) { |
|  | t[n] = a[b]; |
|  | return; |
|  | } |
|  | int mid = (b + e) >> 1; |
|  | build(lc, b, mid); |
|  | build(rc, mid + 1, e); |
|  | pull(n); |
|  | } |
|  | void upd(int n, int b, int e, int i, int j, long long v) { |
|  | push(n, b, e); |
|  | if (j < b || e < i) return; |
|  | if (i <= b && e <= j) { |
|  | lazy[n] = v; //set lazy |
|  | push(n, b, e); |
|  | return; |
|  | } |
|  | int mid = (b + e) >> 1; |
|  | upd(lc, b, mid, i, j, v); |
|  | upd(rc, mid + 1, e, i, j, v); |
|  | pull(n); |
|  | } |
|  | long long query(int n, int b, int e, int i, int j) { |
|  | push(n, b, e); |
|  | if (i > e || b > j) return 0; //return null |
|  | if (i <= b && e <= j) return t[n]; |
|  | int mid = (b + e) >> 1; |
|  | return combine(query(lc, b, mid, i, j), query(rc, mid + 1, e, i, j)); |
|  | } |
|  | }; |
|  | int32\_t main() { |
|  |  |
|  | } |

**All prime factors in O(n^(1/3)):**

N = input()

primes = array containing primes till 10^6

ans = 1

for all p in primes :

if p\*p\*p > N:

break

count = 1

while N divisible by p:

N = N/p

count = count + 1

ans = ans \* count

if N is prime:

ans = ans \* 2

else if N is square of a prime:

ans = ans \* 3

else if N != 1:

ans = ans \* 4

**Chinese Remainder Theorem:**

typedef long long ll;

typedef pair<int,int> pi;

typedef double db;

// In case of pairwise coprime remainders:

// Extended Euclidean Algorithm

// Input: a, b, x, y

// Output: gcd(a, b), with x & y being set so that ax + by = 1

ll Exgcd(ll a,ll b,ll &x,ll &y)

{

if(!b)

{

x=1;y=0;

return a;

}

ll d=Exgcd(b,a%b,x,y);

ll t=x;

x=y;

y=t-(a/b)\*y;

return d;

}

// Chinese Remainder Theorem Algorithm

// Input: 'a' for array of remainders, 'm' for array of coprime modulos, n is the length of the array

// Output: smallest positive ans that satisfies ans % m[i] = a[i], 0 <= i < n

ll CRT(ll \*a,ll \*m,int n)

{

ll M=1;

ll ans=0;

for(int i=0;i<n;i++) M\*=m[i];

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

{

ll x=0, y=0;

ll Mi=M/m[i];

Exgcd(Mi,m[i],x,y); //x=inverse(Mi)

ans=(ans+a[i]\*Mi%M\*x%M)%M;

}

if(ans<0) ans+=M;

return ans;

}

**Closest Pair:**

#include <cstdio>

#include <vector>

#include <algorithm>

#include <cfloat>

using namespace std;

#define \_x\_ first

#define \_y\_ second

using ll = double;

typedef pair<ll,ll> point;

/\*

Closest pair in a set of points

Divide & Conquer algo in O(N log N)

NB: there also exists a sweep line O(N log N) algo

and an expected O(N) randomized algo

\*/

int N;

vector<point> P;

#define SQR(X) ((X)\*(X))

ll dist(const point &A, const point &B) {

return SQR(A.\_x\_ - B.\_x\_) + SQR(A.\_y\_ - B.\_y\_);

}

ll min\_dist(int l, int r, point &p1, point &p2) {

if (l>=r) return DBL\_MAX;

int m = (l+r)/2;

ll D = min\_dist(l,m,p1,p2);

point p,q;

ll d = min\_dist(m+1,r,p,q);

if (d<D) { D = d; p1 = p; p2 = q; }

ll mx = P[m].\_x\_;

vector<point> Mid;

for (int i=l; i<=r; ++i)

if (SQR(abs(P[i].\_x\_ - mx)) < D)

Mid.push\_back(point(P[i].\_y\_,P[i].\_x\_));

sort(Mid.begin(),Mid.end()); // sorted by y

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

// trick: checking a sliding window of size 6 is enough here

for (int j=i+1; j<min(i+6,(int)Mid.size()); ++j) {

d = dist(Mid[i],Mid[j]);

if (d<D) {

D = d;

p1 = make\_pair(Mid[i].\_y\_,Mid[i].\_x\_);

p2 = make\_pair(Mid[j].\_y\_,Mid[j].\_x\_);

}

}

return D;

}

int main() {

while (true) {

scanf("%d", &N);

if (N<=0) break;

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

ll x,y;

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

P.push\_back(point(x,y));

}

sort(P.begin(),P.end()); // sorted by x

bool uniq = true;

point p,q;

for (int i=1; uniq && i<N; ++i)

if (P[i-1]==P[i]) {

p = P[i]; q = P[i]; uniq = false;

}

if (uniq) min\_dist(0,N-1,p,q);

printf("%.2f %.2f %.2f %.2f\n", p.\_x\_, p.\_y\_, q.\_x\_ , q.\_y\_);

// cleaning

P.clear();

}

return 0;

}

**Convex Hull:**

import java.util.\*;

public class Solution {

public int orientation(int[] p, int[] q, int[] r) {

return (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1]);

}

public int[][] outerTrees(int[][] points) {

Arrays.sort(points, new Comparator<int[]> () {

public int compare(int[] p, int[] q) {

return q[0] - p[0] == 0 ? q[1] - p[1] : q[0] - p[0];

}

});

Stack<int[]> hull = new Stack<>();

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

while (hull.size() >= 2 && orientation(hull.get(hull.size() - 2), hull.get(hull.size() - 1), points[i]) > 0)

hull.pop();

hull.push(points[i]);

}

hull.pop();

for (int i = points.length - 1; i >= 0; i--) {

while (hull.size() >= 2 && orientation(hull.get(hull.size() - 2), hull.get(hull.size() - 1), points[i]) > 0)

hull.pop();

hull.push(points[i]);

}

// remove redundant elements from the stack

HashSet<int[]> ret = new HashSet<>(hull);

return ret.toArray(new int[ret.size()][]);

}

}

**Extended Euclidean:**

Text

Description automatically generated

**Mod Inverse:**

ll extended\_euclidean(ll a,ll b,ll &x,ll &y)

{

if(!b)

{

x=1;y=0;

return a;

}

ll d=extended\_euclidean(b,a%b,x,y);

ll t=x;

x=y;

y=t-(a/b)\*y;

return d;

}

//inv for one number

int mod(int a, int m){

ll x, y;

ll g = extended\_euclidean(a, m, x, y);

if (g != 1) {

cout << "No solution!";

}

else {

x = (x % m + m) % m;

cout << x << endl;

}

}

//inv of all numbers in an array: Average O(1) run time for each

std::vector<int> invs(const std::vector<int> &a, int m) {

int n = a.size();

if (n == 0) return {};

std::vector<int> b(n);

int v = 1;

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

b[i] = v;

v = static\_cast<long long>(v) \* a[i] % m;

}

ll x, y;

extended\_euclidean(v, m, x, y);

x = (x % m + m) % m;

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

b[i] = static\_cast<long long>(x) \* b[i] % m;

x = static\_cast<long long>(x) \* a[i] % m;

}

return b;

}

**LIS:**

public int lengthOfLIS(int[] nums) {

int[] tails = new int[nums.length];

int size = 0;

for (int x : nums) {

int i = 0, j = size;

while (i != j) {

int m = (i + j) / 2;

if (tails[m] < x)

i = m + 1;

else

j = m;

}

tails[i] = x;

if (i == size) ++size;

}

return size;

}

**Manacher’s:**

class Solution {

private char[] preprocess(String s) {

char[] t = new char[s.length()\*2 + 3];

char[] ss = s.toCharArray();

t[0] = '$';

t[ss.length\*2 + 2] = '@';

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

t[2\*i + 1] = '#';

t[2\*i + 2] = ss[i];

}

t[ss.length\*2 + 1] = '#';

return t;

}

public int[] Manacher(String s){

char[] t = preprocess(s);

int[] p = new int[t.length];

int center = 0, right = 0;

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

int mirror = 2\*center - i;

if(right>i)

p[i]=Math.min(right-i,p[mirror]);

while(t[i+(1+p[i])]==t[i-(1+p[i])])

++p[i];

if(i+p[i]>right){

center=i;

right=i+p[i];

}

}

return p;

}

public String longestPalindrome(String s) {

int[] p = Manacher(s);

int length = 0; // length of longest palindromic substring

int center = 0; // center of longest palindromic substring

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

if (p[i] > length) {

length = p[i];

center = i;

}

}

return s.substring((center - 1 - length) / 2, (center - 1 + length) / 2);

}

}

**Trie:**

class TrieNode {

public char val;

public boolean isWord;

public TrieNode[] children = new TrieNode[26];

public TrieNode() {}

TrieNode(char c){

TrieNode node = new TrieNode();

node.val = c;

}

}

public class Trie {

private TrieNode root;

public Trie() {

root = new TrieNode();

root.val = ' ';

}

public void insert(String word) {

TrieNode ws = root;

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

char c = word.charAt(i);

if(ws.children[c - 'a'] == null){

ws.children[c - 'a'] = new TrieNode(c);

}

ws = ws.children[c - 'a'];

}

ws.isWord = true;

}

public boolean search(String word) {

TrieNode ws = root;

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

char c = word.charAt(i);

if(ws.children[c - 'a'] == null) return false;

ws = ws.children[c - 'a'];

}

return ws.isWord;

}

public boolean startsWith(String prefix) {

TrieNode ws = root;

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

char c = prefix.charAt(i);

if(ws.children[c - 'a'] == null) return false;

ws = ws.children[c - 'a'];

}

return true;

}

}

**AC\_AUTOMATON:**

// Meysam Aghighi

// Aho-Corasick's algorithm, find keywords' hits in a pattern

using namespace std;

#include <algorithm>

#include <iostream>

#include <cstring>

#include <cstdio>

#include <vector>

#include <queue>

#include <map>

#define iter(it,c) for (\_\_typeof((c).begin()) it = (c).begin(); it != (c).end(); ++it)

typedef vector<vector<int> > vvi;

map<string,int> all;

// Aho-Corasick from SuprDewd

struct aho\_corasick {

struct out\_node {

string keyword; out\_node \*next;

out\_node(string k, out\_node \*n) : keyword(k), next(n) { }

};

struct go\_node {

map<char, go\_node\*> next;

out\_node \*out; go\_node \*fail;

go\_node() { out = NULL; fail = NULL; }

};

go\_node \*go;

int N;

aho\_corasick(vector<string> keywords) {

N = keywords.size();

go = new go\_node();

iter(k, keywords) {

go\_node \*cur = go;

iter(c, \*k)

cur = cur->next.find(\*c) != cur->next.end() ? cur->next[\*c] :

(cur->next[\*c] = new go\_node());

cur->out = new out\_node(\*k, cur->out);

}

queue<go\_node\*> q;

iter(a, go->next) q.push(a->second);

while (!q.empty()) {

go\_node \*r = q.front(); q.pop();

iter(a, r->next) {

go\_node \*s = a->second;

q.push(s);

go\_node \*st = r->fail;

while (st && st->next.find(a->first) == st->next.end())

st = st->fail;

if (!st) st = go;

s->fail = st->next[a->first];

if (s->fail) {

if (!s->out) s->out = s->fail->out;

else {

out\_node\* out = s->out;

while (out->next) out = out->next;

out->next = s->fail->out;

}

}

}

}

}

vvi search(string s){

vvi ress(N);

go\_node \*cur = go;

iter(c, s) {

while (cur && cur->next.find(\*c) == cur->next.end())

cur = cur->fail;

if (!cur) cur = go;

cur = cur->next[\*c];

if (!cur) cur = go;

for (out\_node \*out = cur->out; out; out = out->next){

ress[all[out->keyword]].push\_back(c-s.begin()-out->keyword.size()+1);

}

}

return ress;

}

};

**KMP:**

import java.io.\*;

import java.util.\*;

public class KMP{

public static int[] prev;

public static int[] computeLPS(char[] s){

int[] lps = new int[s.length];

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

int j = lps[i - 1];

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

j = lps[j - 1];

}

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

++j;

}

lps[i] = j;

}

return lps;

}

public static void print(int[] arr){

for(int ele : arr){

System.out.print(ele + "\t");

}

System.out.println();

}

public static void main(String[] args){

Scanner in = new Scanner(System.in);

char[] txt = in.nextLine().toCharArray(), pat = in.nextLine().toCharArray();

int[] lps = computeLPS(pat);

int i = 0, j = 0;

while(i < txt.length){

if(txt[i] == pat[j]){

++i;

++j;

}

//matches

if(j == pat.length){

j = lps[j - 1];

}

else if(i < txt.length && txt[i] != pat[j]){

if(j != 0){

j = lps[j - 1];

}

else{

++i;

}

}

}

}

}

**Suffix Array:**

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e6 + 10;

int s[maxn],wa[maxn],wb[maxn],wss[maxn],wv[maxn],height[maxn],rnk[maxn],sa[maxn];

bool cmp(int \*r,int a,int b,int l){

return r[a] == r[b] && r[a + l] == r[b + l];

}

void build\_sa(int \*r,int \*sa,int n,int m){

int i,j,p,\*x = wa,\*y = wb;

for(i = 0; i < m; i++) wss[i] = 0;

for(i = 0; i < n; i++) wss[x[i] = r[i]]++;

for(i = 1; i < m; i++) wss[i] += wss[i - 1];

for(i = n - 1; i >= 0; i--) sa[-wss[x[i]]] = i;

for(j = 1,p = 1; p < n; j <<= 1,m = p){

for(p = 0,i = n - j; i < n; i++) y[p++] = i;

for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;

for(i = 0; i < m; i++) wss[i] = 0;

for(i = 0; i < n; i++) wv[i] = x[y[i]];

for(i = 0; i < n; i++) wss[wv[i]]++;

for(i = 1; i < m; i++) wss[i] += wss[i - 1];

for(i = n - 1; i >= 0; i--) sa[--wss[wv[i]]] = y[i];

swap(x,y);

for(i = 1,p = 1,x[sa[0]] = 0; i < n; i++){

x[sa[i]] = cmp(y,sa[i - 1],sa[i],j) ? p - 1: p++;

}

}

}

void get\_height(int \*r,int n){

int i,j,k = 0;

for(i = 1; i <= n; i++) rnk[sa[i]] = i;

for(i = 0; i < n; height[rnk[i++]] = k){

for(k ? k-- : 0, j = sa[rnk[i] - 1]; r[a + k] == r[b + k]; k++);

}

}

**Suffix\_Automaton:**

#include <bits/stdc++.h>

const int maxn = 1e6 + 5;

char str[maxn];

class SuffixAutomaton{

public:

struct node{

int mx;

node \*fail,\*ch[26];

node(int a = 0):mx(a),fail(NULL){}

~node(){ delete fail;}

} \*start,\*last,\_pool[maxn<<2],\*\_cur;

vector<node\*> topo;

void init(){

\_cur = \_pool;

start = last = new (\_cur++) node;

}

SuffixAutomaton(){ init();}

static inline int idx(char c){ return c - 'a';}

void extend(char cha){

int c = idx(cha);

node \*u = new (\_cur++) node(last->mx + 1), \*v = last;

for(; v && !v->ch[c]; v = v->fail) v->ch[c] = u;

if(!v){

u->fail = start;

}

else if(v->ch[c]->mx == v->mx + 1){

u->fail = v->ch[c];

}

else {

node \*n = new (\_cur++) node(v->mx + 1), \*o = v->ch[c];

memcpy(n->ch,o->ch,sizeof(o->ch));

n->fail = o->fail;

o->fail = u->fail = n;

for(; v && v->ch[c] == o; v = v->fail) v->ch[c] = n;

}

last = u;

}

void extend(char \*s){

int n = strlen(s);

for(int i = 0; i < n; i++) extend(s[i]);

}

void toposort(){

static int buc[maxn<<2];

int mx = 0;

for(node \*cur = \_pool; cur != \_cur; cur++){

buc[cur->mx]++;

mx = max(mx,buc[cur->mx]);

}

for(int i = 1; i <= mx; i++) buc[i] += buc[i - 1];

topo.resize(\_cur - \_pool);

for(node \*cur = \_pool; cur != \_cur; cur++){

topo[--buc[cur->mx]] = cur;

}

fill(buc,buc + 1+ mx,0);

}

} \*sa;

**SegTree:**

import java.io.\*;

import java.util.\*;

public class SegTree {

long[] tree;

int n;

public SegTree(int[] nums) {

if (nums.length > 0) {

n = nums.length;

tree = new long[n \* 4];

//buildTree(nums);

}

}

private void buildTree(int[] nums) {

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

tree[i] = (long)nums[j];

for (int i = n - 1; i > 0; --i)

tree[i] = tree[i \* 2] + tree[i \* 2 + 1];

}

void update(int pos, long val) {

pos += n;

tree[pos] = val;

while (pos > 0) {

int left = pos;

int right = pos;

if (pos % 2 == 0) {

right = pos + 1;

} else {

left = pos - 1;

}

// parent is updated after child is updated

tree[pos / 2] = tree[left] + tree[right];

pos /= 2;

}

}

public long sumRange(int l, int r) {

// get leaf with value 'l'

l += n;

// get leaf with value 'r'

r += n;

long sum = 0;

while (l <= r) {

if ((l % 2) == 1) {

sum += tree[l];

l++;

}

if ((r % 2) == 0) {

sum += tree[r];

r--;

}

l /= 2;

r /= 2;

}

return sum;

}

public void print(){

for(int i = 0; i < n \* 4; ++i){

System.out.print(tree[i]+",");

}

System.out.println();

}

}