

Contents

1 Misc	1	4.3 Algebra	14
1.1 Contest	1	4.3.1 Formal Power Series	14
1.1.1 Makefile	1	4.4 Theorems	15
1.2 How Did We Get Here?	1	4.4.1 Kirchhoff's Theorem	15
1.2.1 Fast I/O	1	4.4.2 Tutte's Matrix	15
1.3 Tools	2	4.4.3 Cayley's Formula	15
1.3.1 Floating Point Binary Search	2	4.4.4 Erdős–Gallai Theorem	15
1.3.2 SplitMix64	2	4.4.5 Burnside's Lemma	15
1.3.3 <random>	2	5 Numeric	15
1.4 Algorithms	2	5.1 Barrett Reduction	15
1.4.1 Bit Hacks	2	5.2 Long Long Multiplication	15
1.4.2 Aliens Trick	2	5.3 Fast Fourier Transform	15
1.4.3 Hilbert Curve	2	5.4 Fast Walsh-Hadamard Transform	16
1.4.4 Infinite Grid Knight Distance	2	5.5 Subset Convolution	16
1.4.5 Palindrome Subsequence	2	5.6 Linear Recurrences	16
1.4.6 Longest Increasing Subsequence	3	5.6.1 Berlekamp-Massey Algorithm	16
1.4.7 Mo's Algorithm on Tree	3	5.6.2 Linear Recurrence Calculation	16
2 Data Structures	3	5.7 Matrices	16
2.1 Fenwick Tree	3	5.7.1 Determinant	16
2.2 Segment Tree (SIMPLE)	3	5.7.2 Inverse	16
2.3 Lazy Segment Tree (SIMPLE)	3	5.7.3 Characteristic Polynomial	17
2.4 Binary Lifting (1 based)	4	5.7.4 Solve Linear Equation	17
2.5 DSU	4	5.8 Polynomial Interpolation	17
2.6 SparseTable	4	5.9 Simplex Algorithm	18
2.7 EulerTour	4	6 Geometry	18
2.8 Heavy-Light Decomposition	5	6.1 Point	18
3 Graph	5	6.1.1 Quarternion	19
3.1 Modeling	5	6.1.2 Spherical Coordinates	19
3.2 Matching/Flows	6	6.2 Segments	19
3.2.1 Dinic's Algorithm	6	6.3 Convex Hull	19
3.2.2 Minimum Cost Flow	6	6.3.1 3D Hull	19
3.2.3 Gomory-Hu Tree	6	6.4 Angular Sort	20
3.2.4 Global Minimum Cut	6	6.5 Convex Polygon Minkowski Sum	20
3.2.5 Bipartite Minimum Cover	7	6.6 Point In Polygon	20
3.2.6 Edmonds' Algorithm	7	6.6.1 Convex Version	20
3.2.7 Minimum Weight Matching	7	6.7 Closest Pair	20
3.2.8 Stable Marriage	8	6.8 Minimum Enclosing Circle	20
3.2.9 Kuhn-Munkres algorithm	8	6.9 Delaunay Triangulation	20
3.3 Shortest Path Faster Algorithm	9	6.9.1 Slower Version	21
3.4 Strongly Connected Components	9	6.10 Half Plane Intersection	21
3.4.1 2-Satisfiability	9	7 Strings	21
3.5 Biconnected Components	9	7.1 Knuth-Morris-Pratt Algorithm	21
3.5.1 Articulation Points	9	7.2 Aho-Corasick Automaton	22
3.5.2 Bridges	9	7.3 Suffix Array	22
3.6 Triconnected Components	9	7.4 Suffix Tree	22
3.7 Centroid Decomposition	10	7.5 Cocke-Younger-Kasami Algorithm	23
3.8 Minimum Mean Cycle	10	7.6 Z Value	23
3.9 Directed MST	10	7.7 Manacher's Algorithm	23
3.10 Maximum Clique	11	7.8 Minimum Rotation	23
3.11 Dominator Tree	11	7.9 Palindromic Tree	23
3.12 Manhattan Distance MST	11	8 Debug List	24
4 Math	12	1. Misc	
4.1 Number Theory	12	1.1. Contest	
4.1.1 Mod Struct	12	1.1.1. Makefile	
4.1.2 Miller-Rabin	12	1 .PRECIOUS: ./p%	
4.1.3 Linear Sieve	12	3 %: p%	
4.1.4 Get Factors and SPF Fucn	12	ulimit -s unlimited && ./p%	
4.1.5 Binary GCD	12	5 p%: p%.cpp	
4.1.6 Extended GCD	12	g++ -o \$@ \$< -std=c++17 -Wall -Wextra -Wshadow \	
4.1.7 Chinese Remainder Theorem	12	-fsanitize=address,undefined	
4.1.8 Baby-Step Giant-Step	13	1.2. How Did We Get Here?	
4.1.9 Pollard's Rho	13	1.2.1. Fast I/O	
4.1.10 Tonelli-Shanks Algorithm	13	1 // use Scanner in Simple tasks and if required then	
4.1.11 Chinese Sieve	13	3 // use fast io	
4.1.12 Rational Number Binary Search	13	5 import java.io.*;	
4.1.13 Farey Sequence	13	7 import java.util.*;	
4.2 Combinatorics	13		
4.2.1 Matroid Intersection	13		
4.2.2 De Bruijn Sequence	14		
4.2.3 Multinomial	14		

```

9 public class fast_io {
10     public static PrintWriter out =
11         new PrintWriter(new BufferedOutputStream(System.out));
12     static FASTIO in = new FASTIO();
13
14     public static void main(String[] args) throws IOException {
15         int cp = in.nextInt();
16         while (cp-- > 0) {
17             solve();
18         }
19         out.close();
20     }
21
22     static void solve() {
23     }
24
25     static class FASTIO {
26         BufferedReader br;
27         StringTokenizer st;
28
29         public FASTIO() {
30             br = new BufferedReader(
31                 new InputStreamReader(System.in)
32             );
33         }
34
35         String next() {
36             while (st == null || !st.hasMoreElements()) {
37                 try {
38                     st = new StringTokenizer(br.readLine());
39                 } catch (IOException e) {
40                     e.printStackTrace();
41                 }
42             }
43             return st.nextToken();
44         }
45
46         int nextInt() {
47             return Integer.parseInt(next());
48         }
49
50         long nextLong() {
51             return Long.parseLong(next());
52         }
53
54         double nextDouble() {
55             return Double.parseDouble(next());
56         }
57
58         String nextLine() {
59             String str = "";
60             try {
61                 st = null;
62                 str = br.readLine();
63             } catch (IOException e) {
64                 e.printStackTrace();
65             }
66             return str;
67         }
68     }
69 }
70
71 }

```

1.3. Tools

1.3.1. Floating Point Binary Search

```

1 union di {
2     double d;
3     ull i;
4 };
5 bool check(double);
6 // binary search in [L, R) with relative error 2^-eps
7 double binary_search(double L, double R, int eps) {
8     di l = {L}, r = {R}, m;
9     while (r.i - l.i > 1LL << (52 - eps)) {
10         m.i = (l.i + r.i) >> 1;
11         if (check(m.d)) r = m;
12         else l = m;
13     }
14     return l.d;
15 }

```

1.3.2. SplitMix64

```

1 using ull = unsigned long long;
2 inline ull splitmix64(ull x) {
3     // change to `static ull x = SEED;` for DRBG
4     ull z = (x += 0x9E3779B97F4A7C15);
5     z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
6     z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
7     return z ^ (z >> 31);
8 }

```

1.3.3. <random>

```

1 import java.util.Random;
2
3 class random {
4     static final Random rng = new Random();
5
6     static int randInt(int l, int r) {
7         return l + rng.nextInt(r - l + 1);
8     }
9
10    static long randLong(long l, long r) {
11        return l + (Math.abs(rng.nextLong()) % (r - l + 1));
12    }
13    // use inside the main
14    // int a = randInt(1, 10);
15    // long b = randLong(100, 1000);
16 }

```

1.4. Algorithms

1.4.1. Bit Hacks

```

1 // next permutation of x as a bit sequence
2 ull next_bits_permutation(ull x) {
3     ull c = __builtin_ctzll(x), r = x + (1ULL << c);
4     return (r ^ x) >> (c + 2) | r;
5 }
6 // iterate over all (proper) subsets of bitset s
7 void subsets(ull s) {
8     for (ull x = s; x; x--) { --x &= s; /* do stuff */ }
9 }

```

1.4.2. Aliens Trick

```

1 // min dp[i] value and its i (smallest one)
2 pll get_dp(int cost);
3 ll aliens(int k, int l, int r) {
4     while (l != r) {
5         int m = (l + r) / 2;
6         auto [f, s] = get_dp(m);
7         if (s == k) return f - m * k;
8         if (s < k) r = m;
9         else l = m + 1;
10    }
11    return get_dp(l).first - l * k;
12 }

```

1.4.3. Hilbert Curve

```

1 ll hilbert(ll n, int x, int y) {
2     ll res = 0;
3     for (ll s = n; s /= 2; ) {
4         int rx = !!((x & s)), ry = !!((y & s));
5         res += s * s * ((3 * rx) ^ ry);
6         if (ry == 0) {
7             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
8             swap(x, y);
9         }
10    }
11    return res;
12 }

```

1.4.4. Infinite Grid Knight Distance

```

1 ll get_dist(ll dx, ll dy) {
2     if (++(dx = abs(dx)) > ++(dy = abs(dy))) swap(dx, dy);
3     if (dx == 1 && dy == 2) return 3;
4     if (dx == 3 && dy == 3) return 4;
5     ll lb = max(dy / 2, (dx + dy) / 3);
6     return ((dx ^ dy ^ lb) & 1) ? ++lb : lb;
7 }

```

1.4.5. Palindrome Subsequence

```

1 public class palSubsequence {
2     public static void main(String[] args) {
3         solve();
4     }
5     public static void solve() {
6         for (int gap = 0; gap < n; gap++) {
7             for (int i = 0, j = gap; j < n; i++, j++) {
8                 if (gap == 0) {
9                     // single char is a palindrome
10                    dp[i][j] = 1;
11                } else if (gap == 1) {
12                    // if both char are same then 3 else 2
13                    if (s.charAt(i) == s.charAt(j)) {
14                        dp[i][j] = 3;
15                    } else {
16                        dp[i][j] = 2;
17                    }
18                }
19            }
20        }
21    }
22 }

```

```

19     } else {
20         // the we have two cases
21         if (s.charAt(i) == s.charAt(j)) {
22             dp[i][j] = dp[i][j - 1] + dp[i + 1][j] + 1;
23         } else {
24             dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i + 1][j - 1];
25         }
26     }
27 }
28 // println(dp[0][n - 1]);
29 }
30 }
31 }

```

1.4.6. Longest Increasing Subsequence

```

1 import java.util.*;
2 public class lis {
3     public static void main(String[] args) {
4         // int[] arr = new int[n];
5         List<Long> dp = new ArrayList<>();
6         for (long x : arr) {
7             // Find the position to replace or extend
8             int pos = Collections.binarySearch(dp, x);
9             if (pos < 0) {
10                 pos = -(pos + 1); // If not found, get insertion point
11                 // If pos is within dp, replace the element
12                 if (pos < dp.size()) {
13                     dp.set(pos, x);
14                 } else {
15                     // Else, extend the subsequence
16                     dp.add(x);
17                 }
18             }
19             // out.println(dp.size()); length of LIS
20         }
21     }
22 }
23 }

```

1.4.7. Mo's Algorithm on Tree

```

1 void MoAlgoOnTree() {
2     Dfs(0, -1);
3     vector<int> euler(tk);
4     for (int i = 0; i < n; ++i) {
5         euler[tin[i]] = i;
6         euler[tout[i]] = i;
7     }
8     vector<int> l(q), r(q), qr(q), sp(q, -1);
9     for (int i = 0; i < q; ++i) {
10         if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
11         int z = GetLCA(u[i], v[i]);
12         sp[i] = z[i];
13         if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
14         else l[i] = tout[u[i]], r[i] = tin[v[i]];
15         qr[i] = i;
16     }
17     sort(qr.begin(), qr.end(), [&](int i, int j) {
18         if (l[i] / KB == l[j] / KB) return r[i] < r[j];
19         return l[i] / KB < l[j] / KB;
20     });
21     vector<bool> used(n);
22     // Add(v): add/remove v to/from the path based on used[v]
23     for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
24         while (tl < l[qr[i]]) Add(euler[tl++]);
25         while (tl > l[qr[i]]) Add(euler[--tl]);
26         while (tr > r[qr[i]]) Add(euler[tr--]);
27         while (tr < r[qr[i]]) Add(euler[++tr]);
28         // add/remove LCA(u, v) if necessary
29     }
30 }

```

2. Data Structures

2.1. Fenwick Tree

```

1 public class FT {
2     static int[] fTree;
3     public static void main(String[] args) {
4         // int[] arr = new int[n + 1]; // 1-based
5         // preProcess(arr);
6     }
7     // 1-based indexing
8     static void preProcess(int[] arr) {
9         int n = arr.length - 1;
10         fTree = new int[n + 1];
11         for (int i = 1; i <= n; i++) {
12             update(i, arr[i]);
13         }
14     }
15 }

```

```

17 static int query(int l, int r) {
18     return prefixSum(r) - prefixSum(l - 1);
19 }
20 static int prefixSum(int idx) {
21     int sum = 0;
22     while (idx > 0) {
23         sum += fTree[idx];
24         idx -= (idx & -idx);
25     }
26     return sum;
27 }
28 static void update(int idx, int delta) {
29     while (idx < fTree.length) {
30         fTree[idx] += delta;
31         idx += (idx & -idx);
32     }
33 }

```

2.2. Segment Tree (SIMPLE)

```

1 public class SegTreeSimple { }
2 class SegmentTree {
3     private int[] tree; private int n;
4     public SegmentTree(int[] arr) {
5         this.n = arr.length; this.tree = new int[4 * n];
6         build(arr, 0, 0, n - 1);
7     }
8     private void build(int[] arr, int node, int start, int end) {
9         if (start == end) {
10             tree[node] = arr[start]; return;
11         }
12         int mid = (start + end) / 2;
13         build(arr, 2 * node + 1, start, mid);
14         build(arr, 2 * node + 2, mid + 1, end);
15         tree[node] = tree[2 * node + 1] + tree[2 * node + 2];
16     }
17     public void update(int index, int value) {
18         update(0, 0, n - 1, index, value);
19     }
20     private void update(int node, int st, int en, int id, int val) {
21         if (st == en) {
22             tree[node] = val; return;
23         }
24         int mid = (st + en) / 2;
25         if (id <= mid) {
26             update(2 * node + 1, st, mid, id, val);
27         } else {
28             update(2 * node + 2, mid + 1, en, id, val);
29         }
30         tree[node] = tree[2 * node + 1] + tree[2 * node + 2];
31     }
32     public int query(int left, int right) {
33         return query(0, 0, n - 1, left, right);
34     }
35     private int KthOne(int node, int start, int end, int k) {
36         if (start == end) return start;
37         int leftCount = tree[2 * node + 1];
38         if (k < leftCount) {
39             return KthOne(2 * node + 1, start, (start + end) / 2, k);
40         } else {
41             return KthOne(2 * node + 2, (start + end) / 2 + 1, end, k - leftCount);
42         }
43     }
44     public int findKthOne(int k) {
45         return KthOne(0, 0, n - 1, k);
46     }
47     private int query(int node, int start, int end, int l, int r) {
48         if (r < start || l > end) return 0; // outside
49         if (l <= start && end <= r) return tree[node]; // inside
50         int mid = (start + end) / 2;
51         int leftSum = query(2 * node + 1, start, mid, l, r);
52         int rightSum = query(2 * node + 2, mid + 1, end, l, r);
53         return leftSum + rightSum;
54     }
55 }

```

2.3. Lazy Segment Tree (SIMPLE)

```

1 import java.util.*;
2 public class LazySimple {
3     private int n;
4     private long[] st;
5     private long[] lazy;
6     public void init(int _n) {
7         this.n = _n;
8         st = new long[4 * n];
9         lazy = new long[4 * n];
10     }
11     private long combine(long a, long b) {
12         return a + b;
13     }
14     private void push(int start, int end, int node) {

```

```

15     if (lazy[node] != 0) {
16         st[node] += (end - start + 1) * lazy[node];
17         if (start != end) {
18             lazy[2 * node + 1] += lazy[node];
19             lazy[2 * node + 2] += lazy[node];
20         }
21         lazy[node] = 0;
22     }
23 }
24 private void build(int start, int end, int node, long[] v) {
25     if (start == end) {
26         st[node] = v[start]; return;
27     }
28     int mid = (start + end) / 2;
29     build(start, mid, 2 * node + 1, v);
30     build(mid + 1, end, 2 * node + 2, v);
31     st[node] = combine(st[2 * node + 1], st[2 * node + 2]);
32 }
33 private long query(int start, int end, int l, int r, int node) {
34     push(start, end, node);
35     if (start > r || end < l) return 0;
36     if (start >= l && end <= r) return st[node];
37     int mid = (start + end) / 2;
38     long q1 = query(start, mid, l, r, 2 * node + 1);
39     long q2 = query(mid + 1, end, l, r, 2 * node + 2);
40     return combine(q1, q2);
41 }
42 private void update(int sta, int en, int node, int l,
43     int r, long val) {
44     push(sta, en, node);
45     if (sta > r || en < l) return;
46     if (sta >= l && en <= r) {
47         lazy[node] = val;
48         push(sta, en, node); return;
49     }
50     int mid = (sta + en) / 2;
51     update(sta, mid, 2 * node + 1, l, r, val);
52     update(mid + 1, en, 2 * node + 2, l, r, val);
53     st[node] = combine(st[2 * node + 1], st[2 * node + 2]);
54 }
55 public void build(long[] v) {
56     build(0, n - 1, 0, v);
57 }
58 public long query(int l, int r) {
59     return query(0, n - 1, l, r, 0);
60 }
61 public void update(int l, int r, long x) {
62     update(0, n - 1, 0, l, r, x);
63 }
64 }
65

```

2.4. Binary Lifting (1 based)

```

1 import java.io.*;
2 import java.util.*;
3 /*
4  * parent[node][i] = parent[parent[node][i - 1]][i - 1];
5  * This means that the 2i th parent of the node is
6  * 2i - 1 th parent of the node ka 2i-1 th parent
7  */
8 public class BinaryLifting {
9     private static final int MAX_LOG = 20;
10    private static void solve() {
11        int[][] par = new int[n + 1][MAX_LOG];
12        dfs(1, 0, adj, par);
13    }
14    private static void dfs(int node, int parent,
15        List<List<Integer>> adj, int[][] par) {
16        par[node][0] = parent;
17        for (int j = 1; j < MAX_LOG; j++) {
18            par[node][j] = par[par[node][j - 1]][j - 1];
19        }
20        for (int adjNode : adj.get(node)) {
21            if (adjNode != parent)
22                dfs(adjNode, node, adj, par);
23        }
24    }
25    static int Kthparent(int node, int k, int[][] par) {
26        for (int i = MAX_LOG - 1; i >= 0; i--) {
27            if (((1 << i) & k) != 0) {
28                node = par[node][i];
29                if (node == 0) return 0;
30            }
31        }
32        return node;
33    }
34 }

```

2.5. DSU

```

1 public class DSU {
2     private int[] parent, rank, size;

```

```

3     int component;
4     public DSU(int n) {
5         parent = new int[n];
6         rank = new int[n];
7         size = new int[n]; //
8         for (int i = 0; i < n; i++) {
9             parent[i] = i;
10            size[i] = 1; //
11        }
12        component = n;
13    }
14    public int find(int x) {
15        if (parent[x] != x)
16            parent[x] = find(parent[x]);
17        return parent[x];
18    }
19    public boolean union(int u, int v) {
20        int rootU = find(u);
21        int rootV = find(v);
22        if (rootU == rootV)
23            return false;
24        component--;
25        if (rank[rootU] > rank[rootV]) {
26            parent[rootV] = rootU;
27            size[rootU] += size[rootV]; //
28        } else if (rank[rootU] < rank[rootV]) {
29            parent[rootU] = rootV;
30            size[rootV] += size[rootU]; //
31        } else {
32            parent[rootV] = rootU;
33            rank[rootU]++;
34            size[rootU] += size[rootV]; //
35        }
36        return true;
37    }
38    public int getComp() {
39        return component;
40    }
41    public int getSize(int x) {
42        return size[find(x)];
43    }
44 }
45

```

2.6. SparseTable

```

1 public class SparseTable {
2     int[][] st;
3     int[] log;
4     public SparseTable(int[] arr) {
5         int n = arr.length;
6         int K = 32 - Integer.numberOfLeadingZeros(n);
7         st = new int[n][K];
8         log = new int[n + 1];
9         log[1] = 0;
10        for (int i = 2; i <= n; i++) {
11            log[i] = log[i / 2] + 1;
12        }
13        for (int i = 0; i < n; i++) {
14            st[i][0] = arr[i];
15        }
16        for (int j = 1; j < K; j++) {
17            for (int i = 0; i + (1 << j) <= n; i++) {
18                st[i][j] = Math.min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
19            }
20        }
21    }
22    public int query(int l, int r) {
23        int len = r - l + 1;
24        int j = log[len];
25        return Math.min(st[l][j], st[r - (1 << j) + 1][j]);
26    }
27 }

```

2.7. EulerTour

```

1 import java.io.*;
2 import java.util.*;
3 public class euler_tour {
4     static List<Integer>[] adj;
5     static int time = 0;
6
7     public static void main(String[] args) throws IOException {
8         int t = 1;
9         while (t-- > 0) {
10             solve();
11         }
12         // out.close();
13     }
14
15     static void solve() {
16         long[] euler = new long[2 * n];

```

```

17  int[] inTime = new int[n + 1];
18  int[] outTime = new int[n + 1];
19
20  dfs(1, -1, inTime, outTime);
21
22  for (int i = 1; i <= n; i++) {
23      euler[inTime[i]] = v[i - 1];
24      euler[outTime[i]] = -v[i - 1];
25  }
26
27  SegTree seg = new SegTree();
28  seg.init(2 * n); // Euler array size
29  seg.build(0, 2 * n - 1, 0, euler);
30
31  while (q-- > 0) {
32      int type = in.nextInt();
33      if (type == 1) {
34          int s = in.nextInt();
35          long x = in.nextLong();
36          seg.update(0, 2 * n - 1, inTime[s], 0, x);
37          seg.update(0, 2 * n - 1, outTime[s], 0, -x);
38      } else {
39          int s = in.nextInt();
40          out.println(seg.query(0, 2 * n - 1, 0, inTime[s], 0));
41      }
42  }
43 }
44
45 private static void dfs(int node, int parent, int[] inTime, int[] outTime) {
46     inTime[node] = time++;
47     for (int adjNode : adj[node]) {
48         if (adjNode != parent) {
49             dfs(adjNode, node, inTime, outTime);
50         }
51     }
52     outTime[node] = time++;
53 }

```

2.8. Heavy-Light Decomposition

```

1  template <bool VALS_EDGES> struct HLD {
2      int N, tim = 0;
3      vector<vi> adj;
4      vi par, siz, depth, rt, pos;
5      Node *tree;
6      HLD(vector<vi> adj_)
7          : N(siz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
8            depth(N), rt(N), pos(N), tree(new Node(0, N)) {
9          dfsSz(0);
10         dfsHld(0);
11     }
12     void dfsSz(int v) {
13         if (par[v] != -1)
14             adj[v].erase(find(all(adj[v]), par[v]));
15         for (int &u : adj[v]) {
16             par[u] = v, depth[u] = depth[v] + 1;
17             dfsSz(u);
18             siz[v] += siz[u];
19             if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
20         }
21     }
22     void dfsHld(int v) {
23         pos[v] = tim++;
24         for (int u : adj[v]) {
25             rt[u] = (u == adj[v][0] ? rt[v] : u);
26             dfsHld(u);
27         }
28     }
29     template <class B> void process(int u, int v, B op) {
30         for (; rt[u] != rt[v]; v = par[rt[v]]) {
31             if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
32             op(pos[rt[v]], pos[v] + 1);
33         }
34         if (depth[u] > depth[v]) swap(u, v);
35         op(pos[u] + VALS_EDGES, pos[v] + 1);
36     }
37     void modifyPath(int u, int v, int val) {
38         process(u, v, [&](int l, int r) { tree->add(l, r, val); });
39     }
40     int queryPath(int u, int v) { // Modify depending on problem
41         int res = -1e9;
42         process(u, v, [&](int l, int r) {
43             res = max(res, tree->query(l, r));
44         });
45         return res;
46     }
47     int querySubtree(int v) { // modifySubtree is similar
48         return tree->query(pos[v] + VALS_EDGES,
49                             pos[v] + siz[v]);
50     }
51 }

```

```

55 }
56 };

```

3. Graph

3.1. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
- Flow solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 - Construct super source S and sink T
 - For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 - For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 - For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 - For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 - Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v$, $v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 - For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

- Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
- Create edge (x, y) with capacity c_{xy} .
- Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2. Matching/Flows

3.2.1. Dinic's Algorithm

```

1 struct Dinic {
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     static constexpr int MAXN = 1000, MAXF = 1e9;
6     vector<edge> v[MAXN];
7     int top[MAXN], deep[MAXN], side[MAXN], s, t;
8     void make_edge(int s, int t, int cap) {
9         v[s].push_back({t, cap, 0, (int)v[t].size()});
10        v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
11    }
12    int dfs(int a, int flow) {
13        if (a == t || !flow) return flow;
14        for (int &i = top[a]; i < v[a].size(); i++) {
15            edge &e = v[a][i];
16            if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
17                int x = dfs(e.to, min(e.cap - e.flow, flow));
18                if (x) {
19                    e.flow += x, v[e.to][e.rev].flow -= x;
20                    return x;
21                }
22            }
23        }
24        deep[a] = -1;
25        return 0;
26    }
27    bool bfs() {
28        queue<int> q;
29        fill_n(deep, MAXN, 0);
30        q.push(s), deep[s] = 1;
31        int tmp;
32        while (!q.empty()) {
33            tmp = q.front(), q.pop();
34            for (edge e : v[tmp])
35                if (!deep[e.to] && e.cap != e.flow)
36                    deep[e.to] = deep[tmp] + 1, q.push(e.to);
37        }
38        return deep[t];
39    }
40    int max_flow(int _s, int _t) {
41        s = _s, t = _t;
42        int flow = 0, tflow;
43        while (bfs()) {
44            fill_n(top, MAXN, 0);
45            while ((tflow = dfs(s, MAXF))) flow += tflow;
46        }
47        return flow;
48    }
49    void reset() {
50        fill_n(side, MAXN, 0);
51        for (auto &i : v) i.clear();
52    }
53 };

```

3.2.2. Minimum Cost Flow

```

1 struct MCF {
2     struct edge {
3         ll to, from, cap, flow, cost, rev;
4     } *fromE[MAXN];
5     vector<edge> v[MAXN];
6     ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
7     void make_edge(int s, int t, ll cap, ll cost) {
8         if (!cap) return;
9         v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
10        v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
11    }
12    bitset<MAXN> vis;
13    void dijkstra() {
14        vis.reset();
15        __gnu_pbds::priority_queue<pair<ll, int>> q;
16        vector<decltype(q)::point_iterator> its(n);
17        q.push({0LL, s});
18        while (!q.empty()) {
19            int now = q.top().second;
20            q.pop();
21            if (vis[now]) continue;
22            vis[now] = 1;
23            ll ndis = dis[now] + pi[now];
24            for (edge &e : v[now]) {
25                if (e.flow == e.cap || vis[e.to]) continue;
26                if (dis[e.to] > ndis + e.cost - pi[e.to]) {
27                    dis[e.to] = ndis + e.cost - pi[e.to];
28                    flows[e.to] = min(flows[now], e.cap - e.flow);
29                    fromE[e.to] = &e;
30                    if (its[e.to] == q.end())
31                        its[e.to] = q.push({-dis[e.to], e.to});
32                    else q.modify(its[e.to], {-dis[e.to], e.to});
33                }
34            }
35        }
36    }
37 }

```

```

35 }
36 }
37 bool AP(ll &flow) {
38     fill_n(dis, n, INF);
39     fromE[s] = 0;
40     dis[s] = 0;
41     flows[s] = flowlim - flow;
42     dijkstra();
43     if (dis[t] == INF) return false;
44     flow += flows[t];
45     for (edge *e = fromE[t]; e; e = fromE[e->from]) {
46         e->flow += flows[t];
47         v[e->to][e->rev].flow -= flows[t];
48     }
49     for (int i = 0; i < n; i++)
50         pi[i] = min(pi[i] + dis[i], INF);
51     return true;
52 }
53 pll solve(int _s, int _t, ll _flowlim = INF) {
54     s = _s, t = _t, flowlim = _flowlim;
55     pll re;
56     while (re.F != flowlim && AP(re.F)) {
57         for (int i = 0; i < n; i++)
58             for (edge &e : v[i])
59                 if (e.flow != 0) re.S += e.flow * e.cost;
60         re.S /= 2;
61         return re;
62     }
63     void init(int _n) {
64         n = _n;
65         fill_n(pi, n, 0);
66         for (int i = 0; i < n; i++) v[i].clear();
67     }
68     void setpi(int s) {
69         fill_n(pi, n, INF);
70         pi[s] = 0;
71         for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
72             flag = 0;
73             for (int i = 0; i < n; i++)
74                 if (pi[i] != INF)
75                     for (edge &e : v[i])
76                         if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
77                             pi[e.to] = tdis, flag = 1;
78         }
79     }
80 };

```

3.2.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```

1
2
3 int e[MAXN][MAXN];
4 int p[MAXN];
5 Dinic D; // original graph
6 void gomory_hu() {
7     fill(p, p + n, 0);
8     fill(e[0], e[n], INF);
9     for (int s = 1; s < n; s++) {
10        int t = p[s];
11        Dinic F = D;
12        int tmp = F.max_flow(s, t);
13        for (int i = 1; i < s; i++)
14            e[s][i] = e[i][s] = min(tmp, e[t][i]);
15        for (int i = s + 1; i <= n; i++)
16            if (p[i] == t && F.side[i]) p[i] = s;
17    }
18 }

```

3.2.4. Global Minimum Cut

```

1
2
3 // weights is an adjacency matrix, undirected
4 pair<int, vi> getMinCut(vector<vi> &weights) {
5     int N = sz(weights);
6     vi used(N), cut, best_cut;
7     int best_weight = -1;
8
9     for (int phase = N - 1; phase >= 0; phase--) {
10        vi w = weights[0], added = used;
11        int prev, k = 0;
12        rep(i, 0, phase) {
13            prev = k;
14            k = -1;
15            rep(j, 1, N) if (!added[j] &&
16                (k == -1 || w[j] > w[k])) k = j;
17            if (i == phase - 1) {
18                rep(j, 0, N) weights[prev][j] += weights[k][j];
19                rep(j, 0, N) weights[j][prev] = weights[prev][j];
20                used[k] = true;
21                cut.push_back(k);
22                if (best_weight == -1 || w[k] < best_weight) {

```

```

23     best_cut = cut;
24     best_weight = w[k];
25 }
26 } else {
27     rep(j, 0, N) w[j] += weights[k][j];
28     added[k] = true;
29 }
30 }
31 }
32 return {best_weight, best_cut};
33 }

```

3.2.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```

1
3 // maximum independent set = all vertices not covered
4 // x : [0, n), y : [0, m]
5 struct Bipartite_vertex_cover {
6     Dinic D;
7     int n, m, s, t, x[maxn], y[maxn];
8     void make_edge(int x, int y) { D.make_edge(x, y + n, 1); }
9     int matching() {
10         int re = D.max_flow(s, t);
11         for (int i = 0; i < n; i++)
12             for (Dinic::edge &e : D.v[i])
13                 if (e.to != s && e.flow == 1) {
14                     x[i] = e.to - n, y[e.to - n] = i;
15                     break;
16                 }
17         return re;
18     }
19     // init() and matching() before use
20     void solve(vector<int> &vx, vector<int> &vy) {
21         bitset<maxn * 2 + 10> vis;
22         queue<int> q;
23         for (int i = 0; i < n; i++)
24             if (x[i] == -1) q.push(i), vis[i] = 1;
25         while (!q.empty()) {
26             int now = q.front();
27             q.pop();
28             if (now < n) {
29                 for (Dinic::edge &e : D.v[now])
30                     if (e.to != s && e.to - n != x[now] && !vis[e.to])
31                         vis[e.to] = 1, q.push(e.to);
32             } else {
33                 if (!vis[y[now - n]])
34                     vis[y[now - n]] = 1, q.push(y[now - n]);
35             }
36         }
37         for (int i = 0; i < n; i++)
38             if (!vis[i]) vx.pb(i);
39         for (int i = 0; i < m; i++)
40             if (vis[i + n]) vy.pb(i);
41     }
42     void init(int _n, int _m) {
43         n = _n, m = _m, s = n + m, t = s + 1;
44         for (int i = 0; i < n; i++)
45             x[i] = -1, D.make_edge(s, i, 1);
46         for (int i = 0; i < m; i++)
47             y[i] = -1, D.make_edge(i + n, t, 1);
48     }
49 };

```

3.2.6. Edmonds' Algorithm

```

1
3 struct Edmonds {
4     int n, T;
5     vector<vector<int>>> g;
6     vector<int> pa, p, used, base;
7     Edmonds(int n)
8         : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
9           base(n) {}
10     void add(int a, int b) {
11         g[a].push_back(b);
12         g[b].push_back(a);
13     }
14     int getBase(int i) {
15         while (i != base[i])
16             base[i] = base[base[i]], i = base[i];
17         return i;
18     }
19     vector<int> toJoin;
20     void mark_path(int v, int x, int b, vector<int> &path) {
21         for (; getBase(v) != b; v = p[x]) {
22             p[v] = x, x = pa[v];
23             toJoin.push_back(v);
24             toJoin.push_back(x);
25             if (!used[x]) used[x] = ++T, path.push_back(x);
26         }
27     }
28 }

```

```

27 }
28 bool go(int v) {
29     for (int x : g[v]) {
30         int b, bv = getBase(v), bx = getBase(x);
31         if (bv == bx) {
32             continue;
33         } else if (used[x]) {
34             vector<int> path;
35             toJoin.clear();
36             if (used[bx] < used[bv])
37                 mark_path(v, x, b = bx, path);
38             else mark_path(x, v, b = bv, path);
39             for (int z : toJoin) base[getBase(z)] = b;
40             for (int z : path)
41                 if (go(z)) return 1;
42         } else if (p[x] == -1) {
43             p[x] = v;
44             if (pa[x] == -1) {
45                 for (int y : g[x]) {
46                     if (y != -1 && x == v)
47                         y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
48                     return 1;
49                 }
50             }
51             if (!used[pa[x]]) {
52                 used[pa[x]] = ++T;
53                 if (go(pa[x])) return 1;
54             }
55         }
56     }
57     return 0;
58 }
59 void init_dfs() {
60     for (int i = 0; i < n; i++)
61         used[i] = 0, p[i] = -1, base[i] = i;
62 }
63 bool dfs(int root) {
64     used[root] = ++T;
65     return go(root);
66 }
67 void match() {
68     int ans = 0;
69     for (int v = 0; v < n; v++)
70         for (int x : g[v])
71             if (pa[v] == -1 && pa[x] == -1) {
72                 pa[v] = x, pa[x] = v, ans++;
73                 break;
74             }
75     init_dfs();
76     for (int i = 0; i < n; i++)
77         if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
78     cout << ans * 2 << "\n";
79     for (int i = 0; i < n; i++)
80         if (pa[i] > i)
81             cout << i + 1 << " " << pa[i] + 1 << "\n";
82 };

```

3.2.7. Minimum Weight Matching

```

1 struct Graph {
2     static const int MAXN = 105;
3     int n, e[MAXN][MAXN];
4     int match[MAXN], d[MAXN], onstk[MAXN];
5     vector<int> stk;
6     void init(int _n) {
7         n = _n;
8         for (int i = 0; i < n; i++)
9             for (int j = 0; j < n; j++)
10                 // change to appropriate infinity
11                 // if not complete graph
12                 e[i][j] = 0;
13     }
14     void add_edge(int u, int v, int w) {
15         e[u][v] = e[v][u] = w;
16     }
17     bool SPFA(int u) {
18         if (onstk[u]) return true;
19         stk.push_back(u);
20         onstk[u] = 1;
21         for (int v = 0; v < n; v++) {
22             if (u != v && match[u] != v && !onstk[v]) {
23                 int m = match[v];
24                 if (d[m] > d[u] - e[v][m] + e[u][v]) {
25                     d[m] = d[u] - e[v][m] + e[u][v];
26                     onstk[v] = 1;
27                     stk.push_back(v);
28                     if (SPFA(m)) return true;
29                     stk.pop_back();
30                     onstk[v] = 0;
31                 }
32             }
33         }
34         onstk[u] = 0;
35         stk.pop_back();
36         return false;
37     }
38 }

```

```

37 }
38 int solve() {
39     for (int i = 0; i < n; i += 2) {
40         match[i] = i + 1;
41         match[i + 1] = i;
42     }
43     while (true) {
44         int found = 0;
45         for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
46         for (int i = 0; i < n; i++) {
47             stk.clear();
48             if (!onstk[i] && SPFA(i)) {
49                 found = 1;
50                 while (stk.size() >= 2) {
51                     int u = stk.back();
52                     stk.pop_back();
53                     int v = stk.back();
54                     stk.pop_back();
55                     match[u] = v;
56                     match[v] = u;
57                 }
58             }
59         }
60         if (!found) break;
61     }
62     int ret = 0;
63     for (int i = 0; i < n; i++) ret += e[i][match[i]];
64     ret /= 2;
65     return ret;
66 }
67 } graph;

```

3.2.8. Stable Marriage

```

1 // normal stable marriage problem
2 /* input:
3 3
4 Albert Laura Nancy Marcy
5 Brad Marcy Nancy Laura
6 Chuck Laura Marcy Nancy
7 Laura Chuck Albert Brad
8 Marcy Albert Chuck Brad
9 Nancy Brad Albert Chuck
10 */
11
12 using namespace std;
13 const int MAXN = 505;
14
15 int n;
16 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
17 int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
18 int current[MAXN]; // current[boy_id] = rank;
19 // boy_id will pursue current[boy_id] girl.
20 int girl_current[MAXN]; // girl[girl_id] = boy_id;
21
22 void initialize() {
23     for (int i = 0; i < n; i++) {
24         current[i] = 0;
25         girl_current[i] = n;
26         order[i][n] = n;
27     }
28 }
29
30 map<string, int> male, female;
31 string bname[MAXN], gname[MAXN];
32 int fit = 0;
33
34 void stable_marriage() {
35     queue<int> que;
36     for (int i = 0; i < n; i++) que.push(i);
37     while (!que.empty()) {
38         int boy_id = que.front();
39         que.pop();
40
41         int girl_id = favor[boy_id][current[boy_id]];
42         current[boy_id]++;
43
44         if (order[girl_id][boy_id] <
45             order[girl_id][girl_current[girl_id]]) {
46             if (girl_current[girl_id] < n)
47                 que.push(girl_current[girl_id]);
48             girl_current[girl_id] = boy_id;
49         } else {
50             que.push(boy_id);
51         }
52     }
53 }
54
55 int main() {
56     cin >> n;
57
58     for (int i = 0; i < n; i++) {

```

```

61     string p, t;
62     cin >> p;
63     male[p] = i;
64     bname[i] = p;
65     for (int j = 0; j < n; j++) {
66         cin >> t;
67         if (!female.count(t)) {
68             gname[fit] = t;
69             female[t] = fit++;
70         }
71         favor[i][j] = female[t];
72     }
73 }
74
75 for (int i = 0; i < n; i++) {
76     string p, t;
77     cin >> p;
78     for (int j = 0; j < n; j++) {
79         cin >> t;
80         order[female[p]][male[t]] = j;
81     }
82 }
83
84 initialize();
85 stable_marriage();
86
87 for (int i = 0; i < n; i++) {
88     cout << bname[i] << " "
89         << gname[favor[i][current[i] - 1]] << endl;
90 }
91 }

```

3.2.9. Kuhn-Munkres algorithm

```

1 // Maximum Weight Perfect Bipartite Matching
2 // Detect non-perfect-matching:
3 // 1. set all edge[i][j] as INF
4 // 2. if solve() >= INF, it is not perfect matching.
5
6 typedef long long ll;
7 struct KM {
8     static const int MAXN = 1050;
9     static const ll INF = 1LL << 60;
10     int n, match[MAXN], vx[MAXN], vy[MAXN];
11     ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
12     void init(int _n) {
13         n = _n;
14         for (int i = 0; i < n; i++)
15             for (int j = 0; j < n; j++) edge[i][j] = 0;
16     }
17     void add_edge(int x, int y, ll w) { edge[x][y] = w; }
18     bool DFS(int x) {
19         vx[x] = 1;
20         for (int y = 0; y < n; y++) {
21             if (vy[y]) continue;
22             if (lx[x] + ly[y] > edge[x][y]) {
23                 slack[y] =
24                     min(slack[y], lx[x] + ly[y] - edge[x][y]);
25             } else {
26                 vy[y] = 1;
27                 if (match[y] == -1 || DFS(match[y])) {
28                     match[y] = x;
29                     return true;
30                 }
31             }
32         }
33         return false;
34     }
35     ll solve() {
36         fill(match, match + n, -1);
37         fill(lx, lx + n, -INF);
38         fill(ly, ly + n, 0);
39         for (int i = 0; i < n; i++)
40             for (int j = 0; j < n; j++)
41                 lx[i] = max(lx[i], edge[i][j]);
42         for (int i = 0; i < n; i++) {
43             fill(slack, slack + n, INF);
44             while (true) {
45                 fill(vx, vx + n, 0);
46                 fill(vy, vy + n, 0);
47                 if (DFS(i)) break;
48                 ll d = INF;
49                 for (int j = 0; j < n; j++)
50                     if (!vy[j]) d = min(d, slack[j]);
51                 for (int j = 0; j < n; j++) {
52                     if (vx[j]) lx[j] -= d;
53                     if (vy[j]) ly[j] += d;
54                     else slack[j] -= d;
55                 }
56             }
57         }
58         ll res = 0;
59         for (int i = 0; i < n; i++) {

```

```

61     }
62     return res;
63 }
} graph;

```

3.3. Shortest Path Faster Algorithm

```

1 struct SPFA {
2     static const int maxn = 1010, INF = 1e9;
3     int dis[maxn];
4     bitset<maxn> inq, inneg;
5     queue<int> q, tq;
6     vector<pii> v[maxn];
7     void make_edge(int s, int t, int w) {
8         v[s].emplace_back(t, w);
9     }
10    void dfs(int a) {
11        inneg[a] = 1;
12        for (pii i : v[a])
13            if (!inneg[i.F]) dfs(i.F);
14    }
15    bool solve(int n, int s) { // true if have neg-cycle
16        for (int i = 0; i <= n; i++) dis[i] = INF;
17        dis[s] = 0, q.push(s);
18        for (int i = 0; i < n; i++) {
19            inq.reset();
20            int now;
21            while (!q.empty()) {
22                now = q.front(), q.pop();
23                for (pii &i : v[now]) {
24                    if (dis[i.F] > dis[now] + i.S) {
25                        dis[i.F] = dis[now] + i.S;
26                        if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
27                    }
28                }
29            }
30            q.swap(tq);
31        }
32        bool re = !q.empty();
33        inneg.reset();
34        while (!q.empty()) {
35            if (!inneg[q.front()]) dfs(q.front());
36            q.pop();
37        }
38        return re;
39    }
40    void reset(int n) {
41        for (int i = 0; i <= n; i++) v[i].clear();
42    }
43 };

```

3.4. Strongly Connected Components

```

1 struct TarjanScc {
2     int n, step;
3     vector<int> time, low, instk, stk;
4     vector<vector<int>> e, scc;
5     TarjanScc(int n_)
6         : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
7     void add_edge(int u, int v) { e[u].push_back(v); }
8     void dfs(int x) {
9         time[x] = low[x] = ++step;
10        stk.push_back(x);
11        instk[x] = 1;
12        for (int y : e[x])
13            if (!time[y]) {
14                dfs(y);
15                low[x] = min(low[x], low[y]);
16            } else if (instk[y]) {
17                low[x] = min(low[x], time[y]);
18            }
19        if (time[x] == low[x]) {
20            scc.emplace_back();
21            for (int y = -1; y != x; ) {
22                y = stk.back();
23                stk.pop_back();
24                instk[y] = 0;
25                scc.back().push_back(y);
26            }
27        }
28    }
29    void solve() {
30        for (int i = 0; i < n; i++)
31            if (!time[i]) dfs(i);
32        reverse(scc.begin(), scc.end());
33        // scc in topological order
34    }
35 };

```

3.4.1. 2-Satisfiability

Requires: Strongly Connected Components

```

1
2
3 // 1 based, vertex in SCC = MAXN * 2
4 // (not i) is i + n
5 struct two_SAT {
6     int n, ans[MAXN];
7     SCC S;
8     void imply(int a, int b) { S.make_edge(a, b); }
9     bool solve(int _n) {
10        n = _n;
11        S.solve(n * 2);
12        for (int i = 1; i <= n; i++) {
13            if (S.scc[i] == S.scc[i + n]) return false;
14            ans[i] = (S.scc[i] < S.scc[i + n]);
15        }
16        return true;
17    }
18    void init(int _n) {
19        n = _n;
20        fill_n(ans, n + 1, 0);
21        S.init(n * 2);
22    }
23 } SAT;

```

3.5. Biconnected Components

3.5.1. Articulation Points

```

1 void dfs(int x, int p) {
2     tin[x] = low[x] = ++t;
3     int ch = 0;
4     for (auto u : g[x])
5         if (u.first != p) {
6             if (!ins[u.second])
7                 st.push(u.second), ins[u.second] = true;
8             if (tin[u.first]) {
9                 low[x] = min(low[x], tin[u.first]);
10                continue;
11            }
12            ++ch;
13            dfs(u.first, x);
14            low[x] = min(low[x], low[u.first]);
15            if (low[u.first] >= tin[x]) {
16                cut[x] = true;
17                ++sz;
18                while (true) {
19                    int e = st.top();
20                    st.pop();
21                    bcc[e] = sz;
22                    if (e == u.second) break;
23                }
24            }
25        }
26    if (ch == 1 && p == -1) cut[x] = false;
27 }

```

3.5.2. Bridges

```

1 // if there are multi-edges, then they are not bridges
2 void dfs(int x, int p) {
3     tin[x] = low[x] = ++t;
4     st.push(x);
5     for (auto u : g[x])
6         if (u.first != p) {
7             if (tin[u.first]) {
8                 low[x] = min(low[x], tin[u.first]);
9                 continue;
10            }
11            dfs(u.first, x);
12            low[x] = min(low[x], low[u.first]);
13            if (low[u.first] == tin[u.first]) br[u.second] = true;
14        }
15    if (tin[x] == low[x]) {
16        ++sz;
17        while (st.size()) {
18            int u = st.top();
19            st.pop();
20            bcc[u] = sz;
21            if (u == x) break;
22        }
23    }
24 }

```

3.6. Triconnected Components

```

1
2
3 // requires a union-find data structure
4 struct ThreeEdgeCC {
5     int V, ind;
6     vector<int> id, pre, post, low, deg, path;
7 }

```

```

vector<vector<int>> components;
UnionFind uf;
template <class Graph>
void dfs(const Graph &G, int v, int prev) {
    pre[v] = ++ind;
    for (int w : G[v])
        if (w != v) {
            if (w == prev) {
                prev = -1;
                continue;
            }
            if (pre[w] != -1) {
                if (pre[w] < pre[v]) {
                    deg[v]++;
                    low[v] = min(low[v], pre[w]);
                } else {
                    deg[v]--;
                    int &u = path[v];
                    for (; u != -1 && pre[u] <= pre[w] &&
                        pre[w] <= post[u];) {
                        uf.join(v, u);
                        deg[v] += deg[u];
                        u = path[u];
                    }
                }
                continue;
            }
            dfs(G, w, v);
            if (path[w] == -1 && deg[w] <= 1) {
                deg[v] += deg[w];
                low[v] = min(low[v], low[w]);
                continue;
            }
            if (deg[w] == 0) w = path[w];
            if (low[v] > low[w]) {
                low[v] = min(low[v], low[w]);
                swap(w, path[v]);
            }
            for (; w != -1; w = path[w]) {
                uf.join(v, w);
                deg[v] += deg[w];
            }
        }
    post[v] = ind;
}
template <class Graph>
ThreeEdgeCC(const Graph &G)
: V(G.size()), ind(-1), id(V, -1), pre(V, -1),
  post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
  uf(V) {
    for (int v = 0; v < V; v++)
        if (pre[v] == -1) dfs(G, v, -1);
    components.reserve(uf.cnt);
    for (int v = 0; v < V; v++)
        if (uf.find(v) == v) {
            id[v] = components.size();
            components.emplace_back(1, v);
            components.back().reserve(uf.getSize(v));
        }
    for (int v = 0; v < V; v++)
        if (id[v] == -1)
            components[id[v] = id[uf.find(v)]] .push_back(v);
};

```

3.7. Centroid Decomposition

```

public class centroid_decomposition {
    // Find the size of the subtree under this node.
    public static int subtreeSize(int node, int par) {
        int res = 1;
        for (int next : adj[node]) {
            if (next == par) {
                continue;
            }
            res += subtreeSize(next, node);
        }
        return (subSize[node] = res);
    }

    // Find the centroid of the tree (the subtree with <= N/2 nodes)
    public static int getCentroid(int node, int par) {
        for (int next : adj[node]) {
            if (next == par) {
                continue;
            }
            // Keep searching for the centroid if there are subtrees with more
            // than N/2 nodes.
            if (subSize[next] * 2 > N) {
                return getCentroid(next, node);
            }
        }
        return node;
    }
}

```

```

}

```

3.8. Minimum Mean Cycle

```

1
3 // d[i][j] == 0 if {i,j} !in E
long long d[1003][1003], dp[1003][1003];
5
pair<long long, long long> MMWC() {
7     memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= n; ++i) dp[0][i] = 0;
9     for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
11             for (int k = 1; k <= n; ++k) {
                dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
13             }
        }
15     }
    long long au = 1ll << 31, ad = 1;
17     for (int i = 1; i <= n; ++i) {
        if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
19         long long u = 0, d = 1;
        for (int j = n - 1; j >= 0; --j) {
21             if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
                u = dp[n][i] - dp[j][i];
23             d = n - j;
            }
25         }
        if (u * ad < au * d) au = u, ad = d;
27     }
    long long g = __gcd(au, ad);
29     return make_pair(au / g, ad / g);
}

```

3.9. Directed MST

```

1 template <typename T> struct DMST {
    T g[maxn][maxn], fw[maxn];
3     int n, fr[maxn];
    bool vis[maxn], inc[maxn];
5     void clear() {
        for (int i = 0; i < maxn; ++i) {
7             for (int j = 0; j < maxn; ++j) g[i][j] = inf;
                vis[i] = inc[i] = false;
9         }
    }
11 void addedge(int u, int v, T w) {
    g[u][v] = min(g[u][v], w);
13 }
    T operator()(int root, int _n) {
15         n = _n;
        if (dfs(root) != n) return -1;
17         T ans = 0;
        while (true) {
19             for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
                for (int i = 1; i <= n; ++i)
21                 if (!inc[i]) {
                    for (int j = 1; j <= n; ++j) {
23                         if (!inc[j] && i != j && g[j][i] < fw[i]) {
                            fw[i] = g[j][i];
25                         fr[i] = j;
                        }
                    }
27                 }
            int x = -1;
29             for (int i = 1; i <= n; ++i)
                if (i != root && !inc[i]) {
31                 int j = i, c = 0;
                while (j != root && fr[j] != i && c <= n)
33                     ++c, j = fr[j];
                if (j == root || c > n) continue;
35                 else {
                    x = i;
37                     break;
                }
39             }
            if (!x) {
41                 for (int i = 1; i <= n; ++i)
                    if (i != root && !inc[i]) ans += fw[i];
                    return ans;
45             }
            int y = x;
47             for (int i = 1; i <= n; ++i) vis[i] = false;
            do {
49                 ans += fw[y];
                y = fr[y];
51                 vis[y] = inc[y] = true;
            } while (y != x);
            inc[x] = false;
53             for (int k = 1; k <= n; ++k)
                if (vis[k]) {
55                 for (int j = 1; j <= n; ++j)

```

```

57         if (!vis[j]) {
58             if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
59             if (g[j][k] < inf &&
60                 g[j][k] - fw[k] < g[j][x])
61                 g[j][x] = g[j][k] - fw[k];
62         }
63     }
64     return ans;
65 }
66 int dfs(int now) {
67     int r = 1;
68     vis[now] = true;
69     for (int i = 1; i <= n; ++i)
70         if (g[now][i] < inf && !vis[i]) r += dfs(i);
71     return r;
72 }
73 };

```

3.10. Maximum Clique

```

1 // source: KACTL
3 typedef vector<bitset<200>> vb;
4 struct MaxClique {
5     double limit = 0.025, pk = 0;
6     struct Vertex {
7         int i, d = 0;
8     };
9     typedef vector<Vertex> vv;
10    vb e;
11    vv V;
12    vector<vi> C;
13    vi qmax, q, S, old;
14    void init(vv &r) {
15        for (auto &v : r) v.d = 0;
16        for (auto &v : r)
17            for (auto j : r) v.d += e[v.i][j.i];
18        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
19        int mxD = r[0].d;
20        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
21    }
22    void expand(vv &R, int lev = 1) {
23        S[lev] += S[lev - 1] - old[lev];
24        old[lev] = S[lev - 1];
25        while (sz(R)) {
26            if (sz(q) + R.back().d <= sz(qmax)) return;
27            q.push_back(R.back().i);
28            vv T;
29            for (auto v : R)
30                if (e[R.back().i][v.i]) T.push_back({v.i});
31            if (sz(T)) {
32                if (S[lev]++ / ++pk < limit) init(T);
33                int j = 0, mxk = 1,
34                    mnk = max(sz(qmax) - sz(q) + 1, 1);
35                C[1].clear(), C[2].clear();
36                for (auto v : T) {
37                    int k = 1;
38                    auto f = [&](int i) { return e[v.i][i]; };
39                    while (any_of(all(C[k]), f)) k++;
40                    if (k > mxk) mxk = k, C[mxk + 1].clear();
41                    if (k < mnk) T[j++].i = v.i;
42                    C[k].push_back(v.i);
43                }
44                if (j > 0) T[j - 1].d = 0;
45                rep(k, mnk, mxk + 1) for (int i : C[k]) T[j++].i = i,
46                    T[j++].d = k;
47                expand(T, lev + 1);
48            } else if (sz(q) > sz(qmax)) qmax = q;
49            q.pop_back(), R.pop_back();
50        }
51    }
52    vi maxClique() {
53        init(V), expand(V);
54        return qmax;
55    }
56    MaxClique(vb conn)
57        : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
58        rep(i, 0, sz(e)) V.push_back({i});
59    }
60 };

```

3.11. Dominator Tree

```

1 // idom[n] is the unique node that strictly dominates n but
2 // does not strictly dominate any other node that strictly
3 // dominates n. idom[n] = 0 if n is entry or the entry
4 // cannot reach n.
5 struct DominatorTree {
6     static const int MAXN = 200010;
7     int n, s;
8     vector<int> g[MAXN], pred[MAXN];

```

```

9     vector<int> cov[MAXN];
10    int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
12    int sdom[MAXN], idom[MAXN];
13    int mom[MAXN], mn[MAXN];
14
15    inline bool cmp(int u, int v) { return dfn[u] < dfn[v]; }
16
17    int eval(int u) {
18        if (mom[u] == u) return u;
19        int res = eval(mom[u]);
20        if (cmp(sdom[mn[mom[u]]], sdom[mn[u]]))
21            mn[u] = mn[mom[u]];
22        return mom[u] = res;
23    }
24
25    void init(int _n, int _s) {
26        n = _n;
27        s = _s;
28        REP1(i, 1, n) {
29            g[i].clear();
30            pred[i].clear();
31            idom[i] = 0;
32        }
33    }
34    void add_edge(int u, int v) {
35        g[u].push_back(v);
36        pred[v].push_back(u);
37    }
38    void DFS(int u) {
39        ts++;
40        dfn[u] = ts;
41        nfd[ts] = u;
42        for (int v : g[u])
43            if (dfn[v] == 0) {
44                par[v] = u;
45                DFS(v);
46            }
47    }
48    void build() {
49        ts = 0;
50        REP1(i, 1, n) {
51            dfn[i] = nfd[i] = 0;
52            cov[i].clear();
53            mom[i] = mn[i] = sdom[i] = i;
54        }
55        DFS(s);
56        for (int i = ts; i >= 2; i--) {
57            int u = nfd[i];
58            if (u == 0) continue;
59            for (int v : pred[u])
60                if (dfn[v]) {
61                    eval(v);
62                    if (cmp(sdom[mn[v]], sdom[u]))
63                        sdom[u] = sdom[mn[v]];
64                }
65            cov[sdom[u]].push_back(u);
66            mom[u] = par[u];
67            for (int w : cov[par[u]]) {
68                eval(w);
69                if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
70                else idom[w] = par[u];
71            }
72            cov[par[u]].clear();
73        }
74        REP1(i, 2, ts) {
75            int u = nfd[i];
76            if (u == 0) continue;
77            if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
78        }
79    }
80    } dom;

```

3.12. Manhattan Distance MST

```

1 // returns [(dist, from, to), ...]
2 // then do normal mst afterwards
3 typedef Point<int> P;
4 vector<array<int, 3>> manhattanMST(vector<P> ps) {
5     vi id(sz(ps));
6     iota(all(id), 0);
7     vector<array<int, 3>> edges;
8     rep(k, 0, 4) {
9         sort(all(id), [&](int i, int j) {
10             return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
11         });
12         map<int, int> sweep;
13         for (int i : id) {
14             for (auto it = sweep.lower_bound(-ps[i].y);
15                 it != sweep.end(); sweep.erase(it++)) {
16                 int j = it->second;
17                 P d = ps[i] - ps[j];

```

```

21     if (d.y > d.x) break;
    edges.push_back({d.y + d.x, i, j});
23 }
    sweep[-ps[i].y] = i;
25 }
    for (P &p : ps)
        if (k & 1) p.x = -p.x;
27         else swap(p.x, p.y);
29 }
return edges;
}

```

```

    Arrays.fill(isPrime, true);
    isPrime[0] = false;
    isPrime[1] = false;
    for (int i = 2; (long) i * i < MAXN; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j < MAXN; j += i) {
                isPrime[j] = false;
            }
        }
    }
}

```

4. Math

4.1. Number Theory

4.1.1. Mod Struct

A list of safe primes: 26003, 27767, 28319, 28979, 29243, 29759, 30467, 910927547, 919012223, 947326223, 990669467, 1007939579, 1019126699, 929760389146037459, 975500632317046523, 989312547895528379

NTT prime p	$p - 1$	primitive root
65537	$1 \ll 16$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

Requires: Extended GCD

```

1
3 template <typename T> struct M {
    static T MOD; // change to constexpr if already known
    T v;
    M(T x = 0) {
        v = (-MOD <= x && x < MOD) ? x : x % MOD;
        if (v < 0) v += MOD;
    }
    explicit operator T() const { return v; }
    bool operator==(const M &b) const { return v == b.v; }
    bool operator!=(const M &b) const { return v != b.v; }
    M operator-() { return M(-v); }
    M operator+(M b) { return M(v + b.v); }
    M operator-(M b) { return M(v - b.v); }
    M operator*(M b) { return M((__int128)v * b.v % MOD); }
    M operator/(M b) { return *this * (b ^ (MOD - 2)); }
    // change above implementation to this if MOD is not prime
    M inv() {
        auto [p, _, g] = extgcd(v, MOD);
        return assert(g == 1), p;
    }
    friend M operator^(M a, ll b) {
        M ans(1);
        for (; b >= 1, a *= a)
            if (b & 1) ans *= a;
        return ans;
    }
    friend M &operator+=(M &a, M b) { return a = a + b; }
    friend M &operator-=(M &a, M b) { return a = a - b; }
    friend M &operator*=(M &a, M b) { return a = a * b; }
    friend M &operator/=(M &a, M b) { return a = a / b; }
};
using Mod = M<int>;
35 template <> int Mod::MOD = 1'000'000'007;
int &MOD = Mod::MOD;

```

4.1.2. Miller-Rabin

Requires: Mod Struct

```

1
3 // checks if Mod::MOD is prime
bool is_prime() {
    if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
    Mod A[] = {2, 7, 61}; // for int values (< 2^31)
    // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
    int s = __builtin_ctzll(MOD - 1), i;
    for (Mod a : A) {
        Mod x = a ^ (MOD >> s);
        for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
        if (i && x != -1) return 0;
    }
    return 1;
}

```

4.1.3. Linear Sieve

```

1 public class prime_sieve {
    static final int MAXN = 1_000_000;
    static boolean[] isPrime = new boolean[MAXN];
    public static void main(String[] args) { }
    static void sieve() {

```

4.1.4. Get Factors and SPF Fucn

```

1 import java.util.*;
3 public class allfactor {
    public static void main(String[] args) { }
    static int N = 100000;
    static int[] spf = new int[N + 1];
    // store the smallest prime factor of i in spf[i].
    static void spf() {
        for (int i = 2; i <= N; i++) {
            spf[i] = i;
        }
        // Sieve of Eratosthenes modified to find smallest prime factor
        for (int i = 2; i * i <= N; i++) {
            if (spf[i] == i) { // If i is prime
                for (int j = i * i; j <= N; j += i) {
                    if (spf[j] == j)
                        // Mark spf[j] with the smallest prime factor
                        spf[j] = i;
                }
            }
        }
    }
    static List<Integer> allFactors(int n) {
        List<Integer> fac = new ArrayList<>();
        fac.add(1);
        while (n > 1) {
            int p = spf[n];
            List<Integer> cur = new ArrayList<>();
            cur.add(1);
            while (n % p == 0) {
                n /= p;
                cur.add(cur.get(cur.size() - 1) * p);
            }
            List<Integer> next = new ArrayList<>();
            for (int x : fac)
                for (int y : cur)
                    next.add(x * y);
            fac = next;
        }
        return fac;
    }
}

```

4.1.5. Binary GCD

```

1 // returns the gcd of non-negative a, b
ull bin_gcd(ull a, ull b) {
    if (!a || !b) return a + b;
    int s = __builtin_ctzll(a | b);
    a >>= __builtin_ctzll(a);
    while (b) {
        if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);
        b >>= a;
    }
    return a << s;
}

```

4.1.6. Extended GCD

```

1 // returns (p, q, g): p * a + q * b == g == gcd(a, b)
// g is not guaranteed to be positive when a < 0 or b < 0
3 tuple<ll, ll, ll> extgcd(ll a, ll b) {
    ll s = 1, t = 0, u = 0, v = 1;
    while (b) {
        ll q = a / b;
        swap(a -= q * b, b);
        swap(s -= q * t, t);
        swap(u -= q * v, v);
    }
    return {s, u, a};
}

```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

1 // for  $0 \leq a < m$ ,  $0 \leq b < n$ , returns the smallest  $x \geq 0$ 
3 // such that  $x \% m == a$  and  $x \% n == b$ 
4 ll crt(ll a, ll m, ll b, ll n) {
5     if (n > m) swap(a, b), swap(m, n);
6     auto [x, y, g] = extgcd(m, n);
7     assert((a - b) % g == 0); // no solution
8     x = ((b - a) / g * x) % (n / g) * m + a;
9     return x < 0 ? x + m / g * n : x;
10 }

```

4.1.8. Baby-Step Giant-Step

Requires: Mod Struct

```

1 // returns  $x$  such that  $a^x = b$  where  $x \in [l, r]$ 
2 ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
3     int m = sqrt(r - l + 1), i;
4     unordered_map<ll, ll> tb;
5     Mod d = (a ^ l) / b;
6     for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
7         if (d == 1) return l + i;
8     else tb[(ll)d] = l + i;
9     Mod c = Mod(1) / (a ^ m);
10    for (i = 0, d = 1; i < m; i++, d *= c)
11        if (auto j = tb.find((ll)d); j != tb.end())
12            return j->second + i * m;
13    return assert(0), -1; // no solution
14 }

```

4.1.9. Pollard's Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
2 // n should be composite
3 ll pollard_rho(ll n) {
4     if (!(n & 1)) return 2;
5     while (1) {
6         ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7         for (int sz = 2; res == 1; sz *= 2) {
8             for (int i = 0; i < sz && res == 1; i++) {
9                 x = f(x, n);
10                res = __gcd(abs(x - y), n);
11            }
12            y = x;
13        }
14        if (res != 0 && res != n) return res;
15    }
16 }

```

4.1.10. Tonelli-Shanks Algorithm

Requires: Mod Struct

```

1 // int legendre(Mod a) {
2 //     if (a == 0) return 0;
3 //     return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
4 // }
5 Mod sqrt(Mod a) {
6     assert(legendre(a) != -1); // no solution
7     ll p = MOD, s = p - 1;
8     if (a == 0) return 0;
9     if (p == 2) return 1;
10    if (p % 4 == 3) return a ^ ((p + 1) / 4);
11    int r, m;
12    for (r = 0; !(s & 1); r++) s >>= 1;
13    Mod n = 2;
14    while (legendre(n) != -1) n += 1;
15    Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
16    while (b != 1) {
17        Mod t = b;
18        for (m = 0; t != 1; m++) t *= t;
19        Mod gs = g ^ (1LL << (r - m - 1));
20        g = gs * gs, x *= gs, b *= g, r = m;
21    }
22    return x;
23 }
24 // to get sqrt(X) modulo  $p^k$ , where  $p$  is an odd prime:
25 //  $c = x^2 \pmod{p}$ ,  $c = x^2 \pmod{p^k}$ ,  $q = p^{k-1}$ 
26 //  $x = x^q * c^{((p^k-2q+1)/2)} \pmod{p^k}$ 

```

4.1.11. Chinese Sieve

```

1 const ll N = 1000000;
2 // f, g, h multiplicative, h = f (dirichlet convolution) g
3 ll pre_g(ll n);
4 ll pre_h(ll n);
5 // preprocessed prefix sum of f
6 ll pre_f[N];
7 // prefix sum of multiplicative function f

```

```

1 ll solve_f(ll n) {
2     static unordered_map<ll, ll> m;
3     if (n < N) return pre_f[n];
4     if (m.count(n)) return m[n];
5     ll ans = pre_h(n);
6     for (ll l = 2, r; l <= n; l = r + 1) {
7         r = n / (n / l);
8         ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
9     }
10    return m[n] = ans;
11 }

```

4.1.12. Rational Number Binary Search

```

1 struct QQ {
2     ll p, q;
3     QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
4 };
5 bool pred(QQ);
6 // returns smallest  $p/q$  in  $[lo, hi]$  such that
7 //  $pred(p/q)$  is true, and  $0 \leq p, q \leq N$ 
8 QQ frac_bs(ll N) {
9     QQ lo{0, 1}, hi{1, 0};
10    if (pred(lo)) return lo;
11    assert(pred(hi));
12    bool dir = 1, L = 1, H = 1;
13    for (; L || H; dir = !dir) {
14        ll len = 0, step = 1;
15        for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
16            if (QQ mid = hi.go(lo, len + step);
17                mid.p > N || mid.q > N || dir ^ pred(mid))
18                t++;
19        else len += step;
20        swap(lo, hi = hi.go(lo, len));
21        (dir ? L : H) = !len;
22    }
23    return dir ? hi : lo;
24 }

```

4.1.13. Farey Sequence

```

1 // returns (e/f), where (a/b, c/d, e/f) are
2 // three consecutive terms in the order n farey sequence
3 // to start, call next_farey(n, 0, 1, 1, n)
4 pll next_farey(ll n, ll a, ll b, ll c, ll d) {
5     ll p = (n + b) / d;
6     return pll(p * c - a, p * d - b);
7 }

```

4.2. Combinatorics

4.2.1. Matroid Intersection

This template assumes 2 weighted matroids of the same type, and that removing an element is much more expensive than checking if one can be added. **Remember to change the implementation details.**

The ground set is $0, 1, \dots, n-1$, where element i has weight $w[i]$. For the unweighted version, remove weights and change BF/SPFA to BFS.

```

1 constexpr int N = 100;
2 constexpr int INF = 1e9;
3
4 struct Matroid { // represents an independent set
5     Matroid(bitset<N>); // initialize from an independent set
6     bool can_add(int); // if adding will break independence
7     Matroid remove(int); // removing from the set
8 };
9
10 auto matroid_intersection(int n, const vector<int> &w) {
11     bitset<N> S;
12     for (int sz = 1; sz <= n; sz++) {
13         Matroid M1(S), M2(S);
14
15         vector<vector<pii>> e(n + 2);
16         for (int j = 0; j < n; j++)
17             if (!S[j]) {
18                 if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
19                 if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
20             }
21         for (int i = 0; i < n; i++)
22             if (S[i]) {
23                 Matroid T1 = M1.remove(i), T2 = M2.remove(i);
24                 for (int j = 0; j < n; j++)
25                     if (!S[j]) {
26                         if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
27                         if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
28                     }
29             }
30
31         vector<pii> dis(n + 2, {INF, 0});
32         vector<int> prev(n + 2, -1);
33         dis[n] = {0, 0};
34     }
35 }

```

```

35 // change to SPFA for more speed, if necessary
36 bool upd = 1;
37 while (upd) {
38     upd = 0;
39     for (int u = 0; u < n + 2; u++)
40         for (auto [v, c] : e[u]) {
41             pii x(dis[u].first + c, dis[u].second + 1);
42             if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
43         }
44
45     if (dis[n + 1].first < INF)
46         for (int x = prev[n + 1]; x != n; x = prev[x])
47             S.flip(x);
48     else break;
49
50     // S is the max-weighted independent set with size sz
51 }
52 return S;
53 }

```

4.2.2. De Bruijn Sequence

```

1 int res[kN], aux[kN], a[kN], sz;
2 void Rec(int t, int p, int n, int k) {
3     if (t > n) {
4         if (n % p == 0)
5             for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
6         else {
7             aux[t] = aux[t - p];
8             Rec(t + 1, p, n, k);
9             for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
10                 Rec(t + 1, t, n, k);
11         }
12     }
13 int DeBruijn(int k, int n) {
14     // return cyclic string of length k^n such that every
15     // string of length n using k character appears as a
16     // substring.
17     if (k == 1) return res[0] = 0, 1;
18     fill(aux, aux + k * n, 0);
19     return sz = 0, Rec(1, 1, n, k), sz;
20
21     // dd jflkjs fjlk jlk
22 }

```

4.2.3. Multinomial

```

1
2 // ways to permute v[i]
3 ll multinomial(vi &v) {
4     ll c = 1, m = v.empty() ? 1 : v[0];
5     for (int i = 1; i < v.size(); i++)
6         for (int j = 0; j < v[i]; j++) c = c * ++m / (j + 1);
7     return c;
8 }

```

4.3. Algebra

4.3.1. Formal Power Series

```

1
2 template <typename mint>
3 struct FormalPowerSeries : vector<mint> {
4     using vector<mint>::vector;
5     using FPS = FormalPowerSeries;
6
7     FPS &operator+=(const FPS &r) {
8         if (r.size() > this->size()) this->resize(r.size());
9         for (int i = 0; i < (int)r.size(); i++)
10             (*this)[i] += r[i];
11         return *this;
12     }
13
14     FPS &operator+=(const mint &r) {
15         if (this->empty()) this->resize(1);
16         (*this)[0] += r;
17         return *this;
18     }
19
20     FPS &operator-=(const FPS &r) {
21         if (r.size() > this->size()) this->resize(r.size());
22         for (int i = 0; i < (int)r.size(); i++)
23             (*this)[i] -= r[i];
24         return *this;
25     }
26
27     FPS &operator-=(const mint &r) {
28         if (this->empty()) this->resize(1);
29         (*this)[0] -= r;
30     }
31 }

```

```

32 return *this;
33 }
34
35 FPS &operator*=(const mint &v) {
36     for (int k = 0; k < (int)this->size(); k++)
37         (*this)[k] *= v;
38     return *this;
39 }
40
41 FPS &operator/=(const FPS &r) {
42     if (this->size() < r.size()) {
43         this->clear();
44         return *this;
45     }
46     int n = this->size() - r.size() + 1;
47     if ((int)r.size() <= 64) {
48         FPS f(*this), g(r);
49         g.shrink();
50         mint coeff = g.back().inverse();
51         for (auto &x : g) x *= coeff;
52         int deg = (int)f.size() - (int)g.size() + 1;
53         int gs = g.size();
54         FPS quo(deg);
55         for (int i = deg - 1; i >= 0; i--) {
56             quo[i] = f[i + gs - 1];
57             for (int j = 0; j < gs; j++)
58                 f[i + j] -= quo[i] * g[j];
59         }
60         *this = quo * coeff;
61         this->resize(n, mint(0));
62         return *this;
63     }
64     return *this = ((*this).rev().pre(n) * r.rev().inv(n))
65                     .pre(n)
66                     .rev();
67 }
68
69 FPS &operator%=(const FPS &r) {
70     *this -= *this / r * r;
71     shrink();
72     return *this;
73 }
74
75 FPS operator+(const FPS &r) const {
76     return FPS(*this) += r;
77 }
78 FPS operator+(const mint &v) const {
79     return FPS(*this) += v;
80 }
81 FPS operator-(const FPS &r) const {
82     return FPS(*this) -= r;
83 }
84 FPS operator-(const mint &v) const {
85     return FPS(*this) -= v;
86 }
87 FPS operator*(const FPS &r) const {
88     return FPS(*this) *= r;
89 }
90 FPS operator*(const mint &v) const {
91     return FPS(*this) *= v;
92 }
93 FPS operator/(const FPS &r) const {
94     return FPS(*this) /= r;
95 }
96 FPS operator%(const FPS &r) const {
97     return FPS(*this) %= r;
98 }
99 FPS operator-() const {
100     FPS ret(this->size());
101     for (int i = 0; i < (int)this->size(); i++)
102         ret[i] = -(*this)[i];
103     return ret;
104 }
105
106 void shrink() {
107     while (this->size() && this->back() == mint(0))
108         this->pop_back();
109 }
110
111 FPS rev() const {
112     FPS ret(*this);
113     reverse(begin(ret), end(ret));
114     return ret;
115 }
116
117 FPS dot(FPS r) const {
118     FPS ret(min(this->size(), r.size()));
119     for (int i = 0; i < (int)ret.size(); i++)
120         ret[i] = (*this)[i] * r[i];
121     return ret;
122 }
123
124 FPS pre(int sz) const {
125     return FPS(begin(*this),

```

```
begin(*this) + min((int)this->size(), sz));
```

```

127 }
129 FPS operator>>(int sz) const {
130     if ((int)this->size() <= sz) return {};
131     FPS ret(*this);
132     ret.erase(ret.begin(), ret.begin() + sz);
133     return ret;
134 }
135 FPS operator<<(int sz) const {
136     FPS ret(*this);
137     ret.insert(ret.begin(), sz, mint(0));
138     return ret;
139 }
140
141 FPS diff() const {
142     const int n = (int)this->size();
143     FPS ret(max(0, n - 1));
144     mint one(1), coeff(1);
145     for (int i = 1; i < n; i++) {
146         ret[i - 1] = (*this)[i] * coeff;
147         coeff += one;
148     }
149     return ret;
150 }
151
152 FPS integral() const {
153     const int n = (int)this->size();
154     FPS ret(n + 1);
155     ret[0] = mint(0);
156     if (n > 0) ret[1] = mint(1);
157     auto mod = mint::get_mod();
158     for (int i = 2; i <= n; i++) {
159         ret[i] = (-ret[mod % i]) * (mod / i);
160     }
161     for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
162     return ret;
163 }
164
165 mint eval(mint x) const {
166     mint r = 0, w = 1;
167     for (auto &v : *this) r += w * v, w *= x;
168     return r;
169 }
170
171 FPS log(int deg = -1) const {
172     assert((*this)[0] == mint(1));
173     if (deg == -1) deg = (int)this->size();
174     return (this->diff() * this->inv(deg))
175         .pre(deg - 1)
176         .integral();
177 }
178
179 FPS pow(int64_t k, int deg = -1) const {
180     const int n = (int)this->size();
181     if (deg == -1) deg = n;
182     for (int i = 0; i < n; i++) {
183         if ((*this)[i] != mint(0)) {
184             if (i * k > deg) return FPS(deg, mint(0));
185             mint rev = mint(1) / (*this)[i];
186             FPS ret =
187                 (((*this * rev) >> i).log(deg) * k).exp(deg) *
188                 ((*this)[i].pow(k));
189             ret = (ret << (i * k)).pre(deg);
190             if ((int)ret.size() < deg) ret.resize(deg, mint(0));
191             return ret;
192         }
193     }
194     return FPS(deg, mint(0));
195 }
196
197 static void *ntt_ptr;
198 static void set_fft();
199 FPS &operator*=(const FPS &r);
200 void ntt();
201 void intt();
202 void ntt_doubling();
203 static int ntt_pr();
204 FPS inv(int deg = -1) const;
205 FPS exp(int deg = -1) const;
206 };
207 template <typename mint>
208 void *FormalPowerSeries<mint>::ntt_ptr = nullptr;

```

4.4. Theorems

4.4.1. Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

4.4.2. Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

4.4.3. Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

4.4.4. Erdős–Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

4.4.5. Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx \in X\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

5. Numeric

5.1. Barrett Reduction

```

1 using ull = unsigned long long;
2 using ul = __uint128_t;
3 // very fast calculation of a % m
4 struct reduction {
5     const ull m, d;
6     explicit reduction(ull m) : m(m), d(((ul)1 << 64) / m) {}
7     inline ull operator()(ull a) const {
8         ull q = (ull)((ul)d * a) >> 64;
9         return (a - q * m) >= m ? a - m : a;
10    }
11 };

```

5.2. Long Long Multiplication

```

1 using ull = unsigned long long;
2 using ll = long long;
3 using ld = long double;
4 // returns a * b % M where a, b < M < 2**63
5 ull mult(ull a, ull b, ull M) {
6     ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
7     return ret + M * (ret < 0) - M * (ret >= (ll)M);
8 }

```

5.3. Fast Fourier Transform

```

1 template <typename T>
2 void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
3     vector<int> br(n);
4     for (int i = 1; i < n; i++) {
5         br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
6         if (br[i] > i) swap(a[i], a[br[i]]);
7     }
8     for (int len = 2; len <= n; len *= 2)
9         for (int i = 0; i < n; i += len)
10             for (int j = 0; j < len / 2; j++) {
11                 int pos = n / len * (inv ? len - j : j);
12                 T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
13                 a[i + j] = u + v, a[i + j + len / 2] = u - v;
14             }
15     if (T minv = T(1) / T(n); inv)
16         for (T &x : a) x *= minv;
17 }

```

Requires: Mod Struct

```

1
3 void ntt(vector<Mod> &a, bool inv, Mod primitive_root) {
    int n = a.size();
    Mod root = primitive_root ^ (MOD - 1) / n;
    vector<Mod> rt(n + 1, 1);
    for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
    fft_(n, a, rt, inv);
9 }
11 void fft(vector<complex<double>> &a, bool inv) {
    int n = a.size();
    vector<complex<double>> rt(n + 1);
    double arg = acos(-1) * 2 / n;
    for (int i = 0; i <= n; i++)
        rt[i] = {cos(arg * i), sin(arg * i)};
    fft_(n, a, rt, inv);
17 }

```

5.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```

1
3 void fwht(vector<Mod> &a, bool inv) {
    int n = a.size();
    for (int d = 1; d < n; d <= 1)
        for (int m = 0; m < n; m++)
            if (!(m & d)) {
                inv ? a[m] -= a[m | d] : a[m] += a[m | d]; // AND
                inv ? a[m | d] -= a[m] : a[m | d] += a[m]; // OR
                Mod x = a[m], y = a[m | d]; // XOR
                a[m] = x + y, a[m | d] = x - y; // XOR
            }
    if (Mod iv = Mod(1) / n; inv) // XOR
        for (Mod &i : a) i *= iv; // XOR
15 }

```

5.5. Subset Convolution

Requires: Mod Struct

```

1
3 #pragma GCC target("popcnt")
3 #include <immintrin.h>
5 void fwht(int n, vector<vector<Mod>> &a, bool inv) {
    for (int h = 0; h < n; h++)
        for (int i = 0; i < (1 << n); i++)
            if (!(i & (1 << h)))
                for (int k = 0; k <= n; k++)
                    inv ? a[i | (1 << h)][k] -= a[i][k]
                        : a[i | (1 << h)][k] += a[i][k];
11 }
13 // c[k] = sum(popcnt(i & j) == sz && i | j == k) a[i] * b[j]
13 vector<Mod> subset_convolution(int n, int sz,
    const vector<Mod> &a,
    const vector<Mod> &b) {
17 int len = n + sz + 1, N = 1 << n;
    vector<vector<Mod>> a(1 << n, vector<Mod>(len, 0)), b = a;
    for (int i = 0; i < N; i++)
        a[i][_mm_popcnt_u64(i)] = a[i],
        b[i][_mm_popcnt_u64(i)] = b[i];
    fwht(n, a, 0), fwht(n, b, 0);
    for (int i = 0; i < N; i++) {
        vector<Mod> tmp(len);
        for (int j = 0; j < len; j++)
            for (int k = 0; k <= j; k++)
                tmp[j] += a[i][k] * b[i][j - k];
        a[i] = tmp;
    }
    fwht(n, a, 1);
    vector<Mod> c(N);
    for (int i = 0; i < N; i++)
        c[i] = a[i][_mm_popcnt_u64(i) + sz];
    return c;
35 }

```

5.6. Linear Recurrences

5.6.1. Berlekamp-Massey Algorithm

```

1 template <typename T>
    vector<T> berlekamp_massey(const vector<T> &s) {
    int n = s.size(), l = 0, m = 1;
    vector<T> r(n), p(n);
    r[0] = p[0] = 1;
    T b = 1, d = 0;
    for (int i = 0; i < n; i++, m++, d = 0) {
        for (int j = 0; j <= l; j++) d += r[j] * s[i - j];
        if ((d != b) == 0) continue; // change if T is float
        auto t = r;
        for (int j = m; j < n; j++) r[j] -= d * p[j - m];
        if (l * 2 <= i) l = i + 1 - l, b *= d, m = 0, p = t;
    }
    return r.resize(l + 1), reverse(r.begin(), r.end()), r;
15 }

```

5.6.2. Linear Recurrence Calculation

```

1 template <typename T> struct lin_rec {
    using poly = vector<T>;
    poly mul(poly a, poly b, poly m) {
        int n = m.size();
        poly r(n);
        for (int i = n - 1; i >= 0; i--) {
            r.insert(r.begin(), 0), r.pop_back();
            T c = r[n - 1] + a[n - 1] * b[i];
            // c /= m[n - 1]; if m is not monic
            for (int j = 0; j < n; j++)
                r[j] += a[j] * b[i] - c * m[j];
        }
        return r;
    }
    poly pow(poly p, ll k, poly m) {
        poly r(m.size());
        r[0] = 1;
        for (; k >= 1; p = mul(p, p, m))
            if (k & 1) r = mul(r, p, m);
        return r;
    }
    T calc(poly t, poly r, ll k) {
        int n = r.size();
        poly p(n);
        p[1] = 1;
        poly q = pow(p, k, r);
        T ans = 0;
        for (int i = 0; i < n; i++) ans += t[i] * q[i];
        return ans;
    }
};
31

```

5.7. Matrices

5.7.1. Determinant

Requires: Mod Struct

```

1
3 Mod det(vector<vector<Mod>> a) {
    int n = a.size();
    Mod ans = 1;
    for (int i = 0; i < n; i++) {
        int b = i;
        for (int j = i + 1; j < n; j++)
            if (a[j][i] != 0) {
                b = j;
                break;
            }
        if (i != b) swap(a[i], a[b]), ans = -ans;
        ans *= a[i][i];
        if (ans == 0) return 0;
        for (int j = i + 1; j < n; j++) {
            Mod v = a[j][i] / a[i][i];
            if (v != 0)
                for (int k = i + 1; k < n; k++)
                    a[j][k] -= v * a[i][k];
        }
    }
    return ans;
23 }

```

```

1 double det(vector<vector<double>> a) {
    int n = a.size();
    double ans = 1;
    for (int i = 0; i < n; i++) {
        int b = i;
        for (int j = i + 1; j < n; j++)
            if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), ans = -ans;
        ans *= a[i][i];
        if (ans == 0) return 0;
        for (int j = i + 1; j < n; j++) {
            double v = a[j][i] / a[i][i];
            if (v != 0)
                for (int k = i + 1; k < n; k++)
                    a[j][k] -= v * a[i][k];
        }
    }
    return ans;
17 }
19 }

```

5.7.2. Inverse

```

1
3 // Returns rank.
// Result is stored in A unless singular (rank < n).
5 // For prime powers, repeatedly set
// A^{-1} = A^{-1} (2I - A*A^{-1}) (mod p^k)

```

```

7 // where A^{-1} starts as the inverse of A mod p,
  // and k is doubled in each step.
9
11 int matInv(vector<vector<double>> &A) {
12     int n = sz(A);
13     vi col(n);
14     vector<vector<double>> tmp(n, vector<double>(n));
15     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
16
17     rep(i, 0, n) {
18         int r = i, c = i;
19         rep(j, i, n) rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j, c = k;
20
21         if (fabs(A[r][c]) < 1e-12) return i;
22         A[i].swap(A[r]);
23         tmp[i].swap(tmp[r]);
24         rep(j, 0, n) swap(A[j][i], A[j][c]);
25         swap(tmp[j][i], tmp[j][c]);
26         swap(col[i], col[c]);
27         double v = A[i][i];
28         rep(j, i + 1, n) {
29             double f = A[j][i] / v;
30             A[j][i] = 0;
31             rep(k, i + 1, n) A[j][k] -= f * A[i][k];
32             rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
33         }
34         rep(j, i + 1, n) A[i][j] /= v;
35         rep(j, 0, n) tmp[i][j] /= v;
36         A[i][i] = 1;
37     }
38
39     for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
40         double v = A[j][i];
41         rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
42     }
43
44     rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
45     return n;
46 }
47
48 int matInv_mod(vector<vector<ll>> &A) {
49     int n = sz(A);
50     vi col(n);
51     vector<vector<ll>> tmp(n, vector<ll>(n));
52     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
53
54     rep(i, 0, n) {
55         int r = i, c = i;
56         rep(j, i, n) rep(k, i, n) if (A[j][k]) {
57             r = j;
58             c = k;
59             goto found;
60         }
61         return i;
62     found:
63     A[i].swap(A[r]);
64     tmp[i].swap(tmp[r]);
65     rep(j, 0, n) swap(A[j][i], A[j][c]);
66     swap(tmp[j][i], tmp[j][c]);
67     swap(col[i], col[c]);
68     ll v = modpow(A[i][i], mod - 2);
69     rep(j, i + 1, n) {
70         ll f = A[j][i] * v % mod;
71         A[j][i] = 0;
72         rep(k, i + 1, n) A[j][k] =
73             (A[j][k] - f * A[i][k]) % mod;
74         rep(k, 0, n) tmp[j][k] =
75             (tmp[j][k] - f * tmp[i][k]) % mod;
76     }
77     rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
78     rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
79     A[i][i] = 1;
80 }
81
82 for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
83     ll v = A[j][i];
84     rep(k, 0, n) tmp[j][k] =
85         (tmp[j][k] - v * tmp[i][k]) % mod;
86 }
87
88 rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
89     tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
90     return n;
91 }

```

5.7.3. Characteristic Polynomial

```

1 // calculate det(a - xI)
2
3 template <typename T>

```

```

vector<T> CharacteristicPolynomial(vector<vector<T>> a) {
    int N = a.size();
    for (int j = 0; j < N - 2; j++) {
        for (int i = j + 1; i < N; i++) {
            if (a[i][j] != 0) {
                swap(a[j + 1], a[i]);
                for (int k = 0; k < N; k++)
                    swap(a[k][j + 1], a[k][i]);
                break;
            }
        }
        if (a[j + 1][j] != 0) {
            T inv = T(1) / a[j + 1][j];
            for (int i = j + 2; i < N; i++) {
                if (a[i][j] == 0) continue;
                T coe = inv * a[i][j];
                for (int l = j; l < N; l++)
                    a[i][l] -= coe * a[j + 1][l];
                for (int k = 0; k < N; k++)
                    a[k][j + 1] += coe * a[k][i];
            }
        }
    }
    vector<vector<T>> p(N + 1);
    p[0] = {T(1)};
    for (int i = 1; i <= N; i++) {
        p[i].resize(i + 1);
        for (int j = 0; j < i; j++) {
            p[i][j + 1] -= p[i - 1][j];
            p[i][j] += p[i - 1][j] * a[i - 1][i - 1];
        }
        T x = 1;
        for (int m = 1; m < i; m++) {
            x *= -a[i - m][i - m - 1];
            T coe = x * a[i - m - 1][i - 1];
            for (int j = 0; j < i - m; j++)
                p[i][j] += coe * p[i - m - 1][j];
        }
    }
    return p[N];
}

```

5.7.4. Solve Linear Equation

```

1
2
3 typedef vector<double> vd;
4 const double eps = 1e-12;
5
6 // solves for x: A * x = b
7 int solveLinear(vector<vd> &A, vd &b, vd &x) {
8     int n = sz(A), m = sz(x), rank = 0, br, bc;
9     if (n) assert(sz(A[0]) == m);
10    vi col(m);
11    iota(all(col), 0);
12
13    rep(i, 0, n) {
14        double v, bv = 0;
15        rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
16            br = r, bc = c, bv = v;
17        if (bv <= eps) {
18            rep(j, i, n) if (fabs(b[j]) > eps) return -1;
19            break;
20        }
21        swap(A[i], A[br]);
22        swap(b[i], b[br]);
23        swap(col[i], col[bc]);
24        rep(j, 0, n) swap(A[j][i], A[j][bc]);
25        bv = 1 / A[i][i];
26        rep(j, i + 1, n) {
27            double fac = A[j][i] * bv;
28            b[j] -= fac * b[i];
29            rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
30        }
31        rank++;
32    }
33
34    x.assign(m, 0);
35    for (int i = rank; i--;) {
36        b[i] /= A[i][i];
37        x[col[i]] = b[i];
38        rep(j, 0, i) b[j] -= A[j][i] * b[i];
39    }
40    return rank; // (multiple solutions if rank < m)
41 }

```

5.8. Polynomial Interpolation

```

1

```

```

3 // returns a, such that a[0]x^0 + a[1]x^1 + a[2]x^2 + ...
// passes through the given points
5 typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
7   vd res(n), temp(n);
   rep(k, 0, n - 1) rep(i, k + 1, n) y[i] =
9   (y[i] - y[k]) / (x[i] - x[k]);
   double last = 0;
11  temp[0] = 1;
   rep(k, 0, n) rep(i, 0, n) {
13     res[i] += y[k] * temp[i];
     swap(last, temp[i]);
15     temp[i] -= last * x[k];
   }
17  return res;
}

```

5.9. Simplex Algorithm

```

1 // Two-phase simplex algorithm for solving linear programs
// of the form
3 //
//      maximize      c^T x
//      subject to    Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be
//             stored
13 //
// OUTPUT: value of the optimal solution (infinity if
//         unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b,
// and c as arguments. Then, call Solve(x).
21 typedef long double ld;
typedef vector<ld> vd;
typedef vector<vd> vvd;
typedef vector<int> vi;
25 const ld EPS = 1e-9;
27 struct LPSolver {
29   int m, n;
   vi B, N;
   vvd D;
31
33   LPSolver(const vvd &A, const vd &b, const vd &c)
       : m(b.size()), n(c.size()), N(n + 1), B(m),
         D(m + 2, vd(n + 2)) {
35     for (int i = 0; i < m; i++)
       for (int j = 0; j < n; j++) D[i][j] = A[i][j];
37     for (int i = 0; i < m; i++) {
       B[i] = n + i;
       D[i][n] = -1;
41     D[i][n + 1] = b[i];
     }
43     for (int j = 0; j < n; j++) {
       N[j] = j;
       D[m][j] = -c[j];
45     }
47     N[n] = -1;
     D[m + 1][n] = 1;
49   }
51   void Pivot(int r, int s) {
       double inv = 1.0 / D[r][s];
53     for (int i = 0; i < m + 2; i++)
       if (i != r)
55       for (int j = 0; j < n + 2; j++)
         if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
57     for (int j = 0; j < n + 2; j++)
       if (j != s) D[r][j] *= inv;
59     for (int i = 0; i < m + 2; i++)
       if (i != r) D[i][s] *= -inv;
61     D[r][s] = inv;
     swap(B[r], N[s]);
63   }
65   bool Simplex(int phase) {
       int x = phase == 1 ? m + 1 : m;
67     while (true) {
       int s = -1;
69     for (int j = 0; j <= n; j++) {
       if (phase == 2 && N[j] == -1) continue;
71     if (s == -1 || D[x][j] < D[x][s] ||
         D[x][j] == D[x][s] && N[j] < N[s])
73       s = j;
     }
75     if (D[x][s] > -EPS) return true;

```

```

77     int r = -1;
     for (int i = 0; i < m; i++) {
79       if (D[i][s] < EPS) continue;
       if (r == -1 ||
81         D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
         (D[i][n + 1] / D[i][s]) ==
83         (D[r][n + 1] / D[r][s]) &&
         B[i] < B[r])
85         r = i;
     }
87     if (r == -1) return false;
     Pivot(r, s);
89   }
91   ld Solve(vd &x) {
       int r = 0;
93     for (int i = 1; i < m; i++)
       if (D[i][n + 1] < D[r][n + 1]) r = i;
95     if (D[r][n + 1] < -EPS) {
       Pivot(r, n);
97     if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
       return numeric_limits<ld>::infinity();
99     for (int i = 0; i < m; i++)
       if (B[i] == -1) {
101       int s = -1;
       for (int j = 0; j <= n; j++)
103       if (s == -1 || D[i][j] < D[i][s] ||
         D[i][j] == D[i][s] && N[j] < N[s])
105       s = j;
       Pivot(i, s);
107     }
     }
109     if (!Simplex(2)) return numeric_limits<ld>::infinity();
     x = vd(n);
111     for (int i = 0; i < m; i++)
       if (B[i] < n) x[B[i]] = D[i][n + 1];
113     return D[m][n + 1];
   }
115 };
117 int main() {
119   const int m = 4;
   const int n = 3;
121   ld _A[m][n] = {
     {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
123   ld _b[m] = {10, -4, 5, -5};
   ld _c[n] = {1, -1, 0};
125
   vvd A(m);
127   vd b(_b, _b + m);
   vd c(_c, _c + n);
129   for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);
131
   LPSolver solver(A, b, c);
   vd x;
133   ld value = solver.Solve(x);
135
   cerr << "VALUE: " << value << endl; // VALUE: 1.29032
   cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
137   for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
   cerr << endl;
139   return 0;
}

```

6. Geometry

6.1. Point

```

1 template <typename T> struct P {
   T x, y;
3   P(T x = 0, T y = 0) : x(x), y(y) {}
   bool operator<(const P &p) const {
5     return tie(x, y) < tie(p.x, p.y);
   }
7   bool operator==(const P &p) const {
     return tie(x, y) == tie(p.x, p.y);
9   }
   P operator-() const { return {-x, -y}; }
11   P operator+(P p) const { return {x + p.x, y + p.y}; }
   P operator-(P p) const { return {x - p.x, y - p.y}; }
13   P operator*(T d) const { return {x * d, y * d}; }
   P operator/(T d) const { return {x / d, y / d}; }
15   T dist2() const { return x * x + y * y; }
   double len() const { return sqrt(dist2()); }
17   P unit() const { return *this / len(); }
   friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
19   friend T cross(P a, P b) { return a.x * b.y - a.y * b.x; }
   friend T cross(P a, P b, P o) {
21     return cross(a - o, b - o);
   }
23 };

```

```
using pt = P<ll>;
```

6.1.1. Quarternion

```
1 constexpr double PI = 3.141592653589793;
2 constexpr double EPS = 1e-7;
3 struct Q {
4     using T = double;
5     T x, y, z, r;
6     Q(T r = 0) : x(0), y(0), z(0), r(r) {}
7     Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
8     friend bool operator==(const Q &a, const Q &b) {
9         return (a - b).abs2() <= EPS;
10    }
11    friend bool operator!=(const Q &a, const Q &b) {
12        return !(a == b);
13    }
14    Q operator-() { return Q(-x, -y, -z, -r); }
15    Q operator+(const Q &b) const {
16        return Q(x + b.x, y + b.y, z + b.z, r + b.r);
17    }
18    Q operator-(const Q &b) const {
19        return Q(x - b.x, y - b.y, z - b.z, r - b.r);
20    }
21    Q operator*(const T &t) const {
22        return Q(x * t, y * t, z * t, r * t);
23    }
24    Q operator*(const Q &b) const {
25        return Q(r * b.x + x * b.r + y * b.z - z * b.y,
26                r * b.y - x * b.z + y * b.r + z * b.x,
27                r * b.z + x * b.y - y * b.x + z * b.r,
28                r * b.r - x * b.x - y * b.y - z * b.z);
29    }
30    Q operator/(const Q &b) const { return *this * b.inv(); }
31    T abs2() const { return r * r + x * x + y * y + z * z; }
32    T len() const { return sqrt(abs2()); }
33    Q conj() const { return Q(-x, -y, -z, r); }
34    Q unit() const { return *this * (1.0 / len()); }
35    Q inv() const { return conj() * (1.0 / abs2()); }
36    friend T dot(Q a, Q b) {
37        return a.x * b.x + a.y * b.y + a.z * b.z;
38    }
39    friend Q cross(Q a, Q b) {
40        return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
41                a.x * b.y - a.y * b.x);
42    }
43    friend Q rotation_around(Q axis, T angle) {
44        return axis.unit() * sin(angle / 2) + cos(angle / 2);
45    }
46    Q rotated_around(Q axis, T angle) {
47        Q u = rotation_around(axis, angle);
48        return u * *this / u;
49    }
50    friend Q rotation_between(Q a, Q b) {
51        a = a.unit(), b = b.unit();
52        if (a == -b) {
53            // degenerate case
54            Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
55                : cross(a, Q(0, 1, 0));
56            return rotation_around(ortho, PI);
57        }
58        return (a * (a + b)).conj();
59    }
60 };
```

6.1.2. Spherical Coordinates

```
1 struct car_p {
2     double x, y, z;
3 };
4 struct sph_p {
5     double r, theta, phi;
6 };
7
8 sph_p conv(car_p p) {
9     double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
10    double theta = asin(p.y / r);
11    double phi = atan2(p.y, p.x);
12    return {r, theta, phi};
13 }
14 car_p conv(sph_p p) {
15     double x = p.r * cos(p.theta) * sin(p.phi);
16     double y = p.r * cos(p.theta) * cos(p.phi);
17     double z = p.r * sin(p.theta);
18     return {x, y, z};
19 }
```

6.2. Segments

```
1 // for non-collinear ABCD, if segments AB and CD intersect
2 bool intersects(pt a, pt b, pt c, pt d) {
3     if (cross(b, c, a) * cross(b, d, a) > 0) return false;
4     if (cross(d, a, c) * cross(d, b, c) > 0) return false;
```

```
5     return true;
6 }
7 // the intersection point of lines AB and CD
8 pt intersect(pt a, pt b, pt c, pt d) {
9     auto x = cross(b, c, a), y = cross(b, d, a);
10    if (x == y) {
11        // if(abs(x, y) < 1e-8) {
12        // is parallel
13    } else {
14        return d * (x / (x - y)) - c * (y / (x - y));
15    }
16 }
```

6.3. Convex Hull

```
1 // returns a convex hull in counterclockwise order
2 // for a non-strict one, change cross >= to >
3 vector<pt> convex_hull(vector<pt> p) {
4     sort(ALL(p));
5     if (p[0] == p.back()) return {p[0]};
6     int n = p.size(), t = 0;
7     vector<pt> h(n + 1);
8     for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9         for (pt i : p) {
10             while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
11                 t--;
12             h[t++] = i;
13         }
14     return h.resize(t), h;
15 }
```

6.3.1. 3D Hull

```
1
2
3 typedef Point3D<double> P3;
4
5 struct PR {
6     void ins(int x) { (a == -1 ? a : b) = x; }
7     void rem(int x) { (a == x ? a : b) = -1; }
8     int cnt() { return (a != -1) + (b != -1); }
9     int a, b;
10 };
11
12 struct F {
13     P3 q;
14     int a, b, c;
15 };
16
17 vector<F> hull3d(const vector<P3> &A) {
18     assert(sz(A) >= 4);
19     vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
20     #define E(x, y) E[f.x][f.y]
21     vector<F> FS;
22     auto mf = [&](int i, int j, int k, int l) {
23         P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
24         if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
25         F f{q, i, j, k};
26         E(a, b).ins(k);
27         E(a, c).ins(j);
28         E(b, c).ins(i);
29         FS.push_back(f);
30     };
31     rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
32     mf(i, j, k, 6 - i - j - k);
33
34     rep(i, 4, sz(A)) {
35         rep(j, 0, sz(FS)) {
36             F f = FS[j];
37             if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
38                 E(a, b).rem(f.c);
39                 E(a, c).rem(f.b);
40                 E(b, c).rem(f.a);
41                 swap(FS[j - 1], FS.back());
42                 FS.pop_back();
43             }
44         }
45         int nw = sz(FS);
46         rep(j, 0, nw) {
47             F f = FS[j];
48             #define C(a, b, c)
49             if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
50             C(a, b, c);
51             C(a, c, b);
52             C(b, c, a);
53         }
54     }
55     for (F &it : FS)
56         if ((A[it.b] - A[it.a])
57             .cross(A[it.c] - A[it.a])
58             .dot(it.q) <= 0)
59             swap(it.c, it.b);
60     return FS;
61 };
```

6.4. Angular Sort

```
1 auto angle_cmp = [](const pt &a, const pt &b) {
2     auto btm = [](const pt &a) {
3         return a.y < 0 || (a.y == 0 && a.x < 0);
4     };
5     return make_tuple(btm(a), a.y * b.x, abs2(a)) <
6         make_tuple(btm(b), a.x * b.y, abs2(b));
7 };
8 void angular_sort(vector<pt> &p) {
9     sort(p.begin(), p.end(), angle_cmp);
10 }
```

6.5. Convex Polygon Minkowski Sum

```
1 // O(n) convex polygon minkowski sum
2 // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
4     auto diff = [](vector<pt> &c) {
5         auto rcmp = [](pt a, pt b) {
6             return pt{a.y, a.x} < pt{b.y, b.x};
7         };
8         rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9         c.push_back(c[0]);
10        vector<pt> ret;
11        for (int i = 1; i < c.size(); i++)
12            ret.push_back(c[i] - c[i - 1]);
13        return ret;
14    };
15    auto dp = diff(p), dq = diff(q);
16    pt cur = p[0] + q[0];
17    vector<pt> d(dp.size() + dq.size(), ret = {cur});
18    // include angle_cmp from angular-sort.cpp
19    merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
20    // optional: make ret strictly convex (UB if degenerate)
21    int now = 0;
22    for (int i = 1; i < d.size(); i++) {
23        if (cross(d[i], d[now]) == 0) d[now] = d[i];
24        else d[++now] = d[i];
25    }
26    d.resize(now + 1);
27    // end optional part
28    for (pt v : d) ret.push_back(cur = cur + v);
29    return ret.pop_back(), ret;
30 }
```

6.6. Point In Polygon

```
1 bool on_segment(pt a, pt b, pt p) {
2     return cross(a, b, p) == 0 && dot((p - a), (p - b)) <= 0;
3 }
4 // p can be any polygon, but this is O(n)
5 bool inside(const vector<pt> &p, pt a) {
6     int cnt = 0, n = p.size();
7     for (int i = 0; i < n; i++) {
8         pt l = p[i], r = p[(i + 1) % n];
9         // change to return 0; for strict version
10        if (on_segment(l, r, a)) return 1;
11        cnt ^= ((a.y < l.y) - (a.y < r.y)) * cross(l, r, a) > 0;
12    }
13    return cnt;
14 }
```

6.6.1. Convex Version

```
1 // no preprocessing version
2 // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
4 bool is_inside(const vector<pt> &c, pt p) {
5     int n = c.size(), l = 1, r = n - 1;
6     if (cross(c[0], c[1], p) < 0) return false;
7     if (cross(c[n - 1], c[0], p) < 0) return false;
8     while (l < r - 1) {
9         int m = (l + r) / 2;
10        T a = cross(c[0], c[m], p);
11        if (a > 0) l = m;
12        else if (a < 0) r = m;
13        else return dot(c[0] - p, c[m] - p) <= 0;
14    }
15    if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
16    else return cross(c[l], c[r], p) >= 0;
17 }
18 // with preprocessing version
19 vector<pt> vecs;
20 pt center;
21 // p must be a strict convex hull, counterclockwise
22 // BEWARE OF OVERFLOWS!!
23 void preprocess(vector<pt> p) {
24     for (auto &v : p) v = v * 3;
25     center = p[0] + p[1] + p[2];
26     center.x /= 3, center.y /= 3;
27     for (auto &v : p) v = v - center;
28 }
```

```
29 vecs = (angular_sort(p), p);
30 }
31 bool intersect_strict(pt a, pt b, pt c, pt d) {
32     if (cross(b, c, a) * cross(b, d, a) > 0) return false;
33     if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
34     return true;
35 }
36 // if point is inside or on border
37 bool query(pt p) {
38     p = p * 3 - center;
39     auto pr = upper_bound(ALL(vecs), p, angle_cmp);
40     if (pr == vecs.end()) pr = vecs.begin();
41     auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
42     return !intersect_strict({0, 0}, p, pl, *pr);
43 }
```

6.7. Closest Pair

```
1 vector<pll> p; // sort by x first!
2 bool cmpy(const pll &a, const pll &b) const {
3     return a.y < b.y;
4 }
5 ll sq(ll x) { return x * x; }
6 // returns (minimum dist)^2 in [l, r]
7 ll solve(int l, int r) {
8     if (r - l <= 1) return 1e18;
9     int m = (l + r) / 2;
10    ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
11    auto pb = p.begin();
12    inplace_merge(pb + l, pb + m, pb + r, cmpy);
13    vector<pll> s;
14    for (int i = l; i < r; i++)
15        if (sq(p[i].x - mid) < d) s.push_back(p[i]);
16    for (int i = 0; i < s.size(); i++)
17        for (int j = i + 1;
18             j < s.size() && sq(s[j].y - s[i].y) < d; j++)
19            d = min(d, dis(s[i], s[j]));
20    return d;
21 }
```

6.8. Minimum Enclosing Circle

```
1
2
3 typedef Point<double> P;
4 double ccRadius(const P &A, const P &B, const P &C) {
5     return (B - A).dist() * (C - B).dist() * (A - C).dist() /
6         abs((B - A).cross(C - A)) / 2;
7 }
8 P ccCenter(const P &A, const P &B, const P &C) {
9     P b = C - A, c = B - A;
10    return A + (b * c.dist2() - c * b.dist2()).perp() /
11        b.cross(c) / 2;
12 }
13 pair<P, double> mec(vector<P> ps) {
14     shuffle(all(ps), mt19937(time(0)));
15     P o = ps[0];
16     double r = 0, EPS = 1 + 1e-8;
17     rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
18         o = ps[i], r = 0;
19         rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
20             o = (ps[i] + ps[j]) / 2;
21             r = (o - ps[i]).dist();
22             rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
23                 o = ccCenter(ps[i], ps[j], ps[k]);
24                 r = (o - ps[i]).dist();
25             }
26         }
27     }
28     return {o, r};
29 }
```

6.9. Delaunay Triangulation

```
1
2
3 typedef Point<ll> P;
4 typedef struct Quad *Q;
5 typedef __int128_t lll; // (can be ll if coords are < 2e4)
6 P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
7
8 struct Quad {
9     bool mark;
10    Q o, rot;
11    P p;
12    P F() { return r()->p; }
13    Q r() { return rot->rot; }
14    Q prev() { return rot->o->rot; }
15    Q next() { return r()->prev(); }
16 };
17
18 bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
19     lll p2 = p.dist2(), A = a.dist2() - p2,
```

```

    B = b.dist2() - p2, C = c.dist2() - p2;
21 return p.cross(a, b) * C + p.cross(b, c) * A +
    p.cross(c, a) * B >
23 0;
}
25 Q makeEdge(P orig, P dest) {
    Q q[] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
27         new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
    rep(i, 0, 4) q[i]-->o = q[-i & 3],
29         q[i]-->rot = q[(i + 1) & 3];
    return *q;
31 }
void splice(Q a, Q b) {
33     swap(a->o->rot->o, b->o->rot->o);
    swap(a->o, b->o);
35 }
Q connect(Q a, Q b) {
37     Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
39     splice(q->r(), b);
    return q;
41 }

pair<Q, Q> rec(const vector<P> &s) {
43     if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]),
45         b = makeEdge(s[1], s.back());
        if (sz(s) == 2) return {a, a->r()};
47         splice(a->r(), b);
        auto side = s[0].cross(s[1], s[2]);
49         Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
51     }

#define H(e) e->F(), e->p
55 #define valid(e) (e->F().cross(H(base)) > 0)
    Q A, B, ra, rb;
    int half = sz(s) / 2;
57     tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)});
59     while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
    Q base = connect(B->r(), A);
63     if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base;
65

#define DEL(e, init, dir)
    Q e = init->dir;
    if (valid(e))
69         while (circ(e->dir->F(), H(base), e->F())) {
            Q t = e->dir;
            splice(e, e->prev());
71             splice(e->r(), e->r()->prev());
            e = t;
73         }
    for (;;) {
75         DEL(LC, base->r(), o);
        DEL(RC, base, prev());
77         if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
79             base = connect(RC, base->r());
        else base = connect(base->r(), LC->r());
81     }
    return {ra, rb};
83 }

// returns [A_0, B_0, C_0, A_1, B_1, ...]
87 // where A_i, B_i, C_i are counter-clockwise triangles
vector<P> triangulate(vector<P> pts) {
89     sort(all(pts));
    assert(unique(all(pts)) == pts.end());
91     if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
93     vector<Q> q = {e};
    int qi = 0;
95     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD
97     {
        Q c = e;
        do {
99             c->mark = 1;
            pts.push_back(c->p);
            q.push_back(c->r());
            c = c->next();
            } while (c != e);
101         }
    ADD;
    pts.clear();
107     while (qi < sz(q))
        if (!(e = q[qi++]>mark) ADD;
109     return pts;
111 }

```

6.9.1. Slower Version

```

1
3 template <class P, class F>
void delaunay(vector<P> &ps, F trifun) {
5     if (sz(ps) == 3) {
        int d = (ps[0].cross(ps[1], ps[2]) < 0);
7         trifun(0, 1 + d, 2 - d);
    }
9     vector<P3> p3;
    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
11     if (sz(p3) > 3)
        for (auto t : hull3d(p3))
13         if ((p3[t.b] - p3[t.a])
            .cross(p3[t.c] - p3[t.a])
            .dot(P3(0, 0, 1)) < 0)
15             trifun(t.a, t.c, t.b);
17 }

```

6.10. Half Plane Intersection

```

1 struct Line {
    Point P;
3     Vector v;
    bool operator<(const Line &b) const {
5         return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
    }
7 };
bool OnLeft(const Line &L, const Point &p) {
9     return Cross(L.v, p - L.P) > 0;
}
11 Point GetIntersection(Line a, Line b) {
    Vector u = a.P - b.P;
13     Double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
15 }
int HalfplaneIntersection(Line *L, int n, Point *poly) {
17     sort(L, L + n);

    int first, last;
    Point *p = new Point[n];
21     Line *q = new Line[n];
    q[first = last = 0] = L[0];
23     for (int i = 1; i < n; i++) {
        while (first < last && !OnLeft(L[i], p[last - 1]))
25         last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
27         q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29             last--;
            if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31         }
        if (first < last)
33             p[last - 1] = GetIntersection(q[last - 1], q[last]);
    }
35     while (first < last && !OnLeft(q[first], p[last - 1]))
        last--;
37     if (last - first <= 1) return 0;
    p[last] = GetIntersection(q[last], q[first]);
39

    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
41     return m;
43 }

```

7. Strings

7.1. Knuth-Morris-Pratt Algorithm

```

1
3 vector<int> pi(const string &s) {
    vector<int> p(s.size());
5     for (int i = 1; i < s.size(); i++) {
        int g = p[i - 1];
7         while (g && s[i] != s[g]) g = p[g - 1];
        p[i] = g + (s[i] == s[g]);
9     }
    return p;
11 }
vector<int> match(const string &s, const string &pat) {
13     vector<int> p = pi(pat + '\0' + s), res;
    for (int i = p.size() - s.size(); i < p.size(); i++)
15         if (p[i] == pat.size())
            res.push_back(i - 2 * pat.size());
17     return res;
}

```

7.2. Aho-Corasick Automaton

```

1 struct Aho_Corasick {
2     static const int maxc = 26, maxn = 4e5;
3     struct NODES {
4         int Next[maxc], fail, ans;
5     };
6     NODES T[maxn];
7     int top, qtop, q[maxn];
8     int get_node(const int &fail) {
9         fill_n(T[top].Next, maxc, 0);
10        T[top].fail = fail;
11        T[top].ans = 0;
12        return top++;
13    }
14    int insert(const string &s) {
15        int ptr = 1;
16        for (char c : s) { // change char id
17            c -= 'a';
18            if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
19            ptr = T[ptr].Next[c];
20        }
21        return ptr;
22    } // return ans_last_place
23    void build_fail(int ptr) {
24        int tmp;
25        for (int i = 0; i < maxc; i++)
26            if (T[ptr].Next[i]) {
27                tmp = T[ptr].fail;
28                while (tmp != 1 && !T[tmp].Next[i])
29                    tmp = T[tmp].fail;
30                if (T[tmp].Next[i] != T[ptr].Next[i])
31                    if (T[tmp].Next[i]) tmp = T[tmp].Next[i];
32                T[T[ptr].Next[i]].fail = tmp;
33                q[qtop++] = T[ptr].Next[i];
34            }
35    }
36    void AC_auto(const string &s) {
37        int ptr = 1;
38        for (char c : s) {
39            while (ptr != 1 && !T[ptr].Next[c]) ptr = T[ptr].fail;
40            if (T[ptr].Next[c]) {
41                ptr = T[ptr].Next[c];
42                T[ptr].ans++;
43            }
44        }
45    }
46    void Solve(string &s) {
47        for (char &c : s) // change char id
48            c -= 'a';
49        for (int i = 0; i < qtop; i++) build_fail(q[i]);
50        AC_auto(s);
51        for (int i = qtop - 1; i > -1; i--)
52            T[T[q[i]].fail].ans += T[q[i]].ans;
53    }
54    void reset() {
55        qtop = top = q[0] = 1;
56        get_node(1);
57    }
58    } AC;
59    // usage example
60    string s, S;
61    int n, t, ans_place[50000];
62    int main() {
63        Tie cin >> t;
64        while (t--) {
65            AC.reset();
66            cin >> S >> n;
67            for (int i = 0; i < n; i++) {
68                cin >> s;
69                ans_place[i] = AC.insert(s);
70            }
71            AC.Solve(S);
72            for (int i = 0; i < n; i++)
73                cout << AC.T[ans_place[i]].ans << '\n';
74        }
75    }

```

7.3. Suffix Array

```

1
2
3 // sa[i]: starting index of suffix at rank i
4 // 0-indexed, sa[0] = n (empty string)
5 // lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
6 struct SuffixArray {
7     vector<int> sa, lcp;
8     SuffixArray(string &s,
9         int lim = 256) { // or basic_string<int>
10        int n = sz(s) + 1, k = 0, a, b;
11        vector<int> x(all(s) + 1), y(n), ws(max(n, lim)),
12            rank(n);
13        sa = lcp = y, iota(all(sa), 0);

```

```

15     for (int j = 0, p = 0; p < n;
16         j = max(1, j * 2), lim = p) {
17         p = j, iota(all(y), n - j);
18         for (int i = 0; i < n; i++)
19             if (sa[i] >= j) y[p++] = sa[i] - j;
20         fill(all(ws), 0);
21         for (int i = 0; i < n; i++) ws[x[i]]++;
22         for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
23         for (int i = n; i--;) sa[-ws[x[y[i]]]] = y[i];
24         swap(x, y), p = 1, x[sa[0]] = 0;
25         for (int i = 1; i < n; i++)
26             a = sa[i - 1], b = sa[i],
27
28             x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
29                 ? p - 1 : p++;
30     }
31     for (int i = 1; i < n; i++) rank[sa[i]] = i;
32     for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
33         for (k && k--, j = sa[rank[i] - 1];
34             s[i + k] == s[j + k]; k++);
35 }
36 };

```

7.4. Suffix Tree

```

1 struct SAM {
2     static const int maxc = 26; // char range
3     static const int maxn = 10010; // string len
4     struct Node {
5         Node *green, *edge[maxc];
6         int max_len, in, times;
7     } *root, *last, reg[maxn * 2];
8     int top;
9     Node *get_node(int _max) {
10        Node *re = &reg[top++];
11        re->in = 0, re->times = 1;
12        re->max_len = _max, re->green = 0;
13        for (int i = 0; i < maxc; i++) re->edge[i] = 0;
14        return re;
15    }
16    void insert(const char c) { // c in range [0, maxc)
17        Node *p = last;
18        last = get_node(p->max_len + 1);
19        while (p && !p->edge[c])
20            p->edge[c] = last, p = p->green;
21        if (!p) last->green = root;
22        else {
23            Node *pot_green = p->edge[c];
24            if ((pot_green->max_len) == (p->max_len + 1))
25                last->green = pot_green;
26            else {
27                Node *wish = get_node(p->max_len + 1);
28                wish->times = 0;
29                while (p && p->edge[c] == pot_green)
30                    p->edge[c] = wish, p = p->green;
31                for (int i = 0; i < maxc; i++)
32                    wish->edge[i] = pot_green->edge[i];
33                wish->green = pot_green->green;
34                pot_green->green = wish;
35                last->green = wish;
36            }
37        }
38    }
39    Node *q[maxn * 2];
40    int ql, qr;
41    void get_times(Node *p) {
42        ql = 0, qr = -1, reg[0].in = 1;
43        for (int i = 1; i < top; i++) reg[i].green->in++;
44        for (int i = 0; i < top; i++)
45            if (!reg[i].in) q[++qr] = &reg[i];
46        while (ql <= qr) {
47            q[ql]->green->times += q[ql]->times;
48            if (!(--q[ql]->green->in)) q[++qr] = q[ql]->green;
49            ql++;
50        }
51    }
52    void build(const string &s) {
53        top = 0;
54        root = last = get_node(0);
55        for (char c : s) insert(c - 'a'); // change char id
56        get_times(root);
57    }
58    // call build before solve
59    int solve(const string &s) {
60        Node *p = root;
61        for (char c : s)
62            if (!(p = p->edge[c - 'a'])) // change char id
63                return 0;
64        return p->times;
65    }
66 };

```

7.5. Cocke-Younger-Kasami Algorithm

```

1
3 struct rule {
4     // s -> xy
5     // if y == -1, then s -> x (unit rule)
6     int s, x, y, cost;
7 };
8 int state;
9 // state (id) for each letter (variable)
10 // lowercase letters are terminal symbols
11 map<char, int> rules;
12 vector<rule> cnf;
13 void init() {
14     state = 0;
15     rules.clear();
16     cnf.clear();
17 }
18 // convert a cfg rule to cnf (but with unit rules) and add
19 // it
20 void add_to_cnf(char s, const string &p, int cost) {
21     if (!rules.count(s)) rules[s] = state++;
22     for (char c : p)
23         if (!rules.count(c)) rules[c] = state++;
24     if (p.size() == 1) {
25         cnf.push_back({rules[s], rules[p[0]], -1, cost});
26     } else {
27         // length >= 3 -> split
28         int left = rules[s];
29         int sz = p.size();
30         for (int i = 0; i < sz - 2; i++) {
31             cnf.push_back({left, rules[p[i]], state, 0});
32             left = state++;
33         }
34         cnf.push_back(
35             {left, rules[p[sz - 2]], rules[p[sz - 1]], cost});
36     }
37 }
38 constexpr int MAXN = 55;
39 vector<long long> dp[MAXN][MAXN];
40 // unit rules with negative costs can cause negative cycles
41 vector<bool> neg_INF[MAXN][MAXN];
42
43 void relax(int l, int r, rule c, long long cost,
44           bool neg_c = 0) {
45     if (!neg_INF[l][r][c.s] &&
46         (neg_INF[l][r][c.x] || cost < dp[l][r][c.s])) {
47         if (neg_c || neg_INF[l][r][c.x]) {
48             dp[l][r][c.s] = 0;
49             neg_INF[l][r][c.s] = true;
50         } else {
51             dp[l][r][c.s] = cost;
52         }
53     }
54 }
55
56 void bellman(int l, int r, int n) {
57     for (int k = 1; k <= state; k++)
58         for (rule c : cnf)
59             if (c.y == -1)
60                 relax(l, r, c, dp[l][r][c.x] + c.cost, k == n);
61 }
62
63 void cyk(const string &s) {
64     vector<int> tok;
65     for (char c : s) tok.push_back(rules[c]);
66     for (int i = 0; i < tok.size(); i++) {
67         for (int j = 0; j < tok.size(); j++) {
68             dp[i][j] = vector<long long>(state + 1, INT_MAX);
69             neg_INF[i][j] = vector<bool>(state + 1, false);
70         }
71         dp[i][i][tok[i]] = 0;
72         bellman(i, i, tok.size());
73     }
74     for (int r = 1; r < tok.size(); r++) {
75         for (int l = r - 1; l >= 0; l--) {
76             for (int k = l; k < r; k++)
77                 for (rule c : cnf)
78                     if (c.y != -1)
79                         relax(l, r, c,
80                             dp[l][k][c.x] + dp[k + 1][r][c.y] +
81                             c.cost);
82             bellman(l, r, tok.size());
83         }
84     }
85 }
86
87 // usage example
88 int main() {
89     init();
90     add_to_cnf('S', "aSc", 1);
91     add_to_cnf('S', "BBB", 1);
92     add_to_cnf('S', "SB", 1);
93     add_to_cnf('B', "b", 1);

```

```

93     cyk("abbbbc");
94     // dp[0][s.size() - 1][rules[start]] = min cost to
95     // generate s
96     cout << dp[0][5][rules['S']] << '\n'; // 7
97     cyk("acbc");
98     cout << dp[0][3][rules['S']] << '\n'; // INT_MAX
99     add_to_cnf('S', "S", -1);
100    cyk("abbbbc");
101    cout << neg_INF[0][5][rules['S']] << '\n'; // 1
102 }

```

7.6. Z Value

```

1 int z[n];
2 void zval(string s) {
3     // z[i] => longest common prefix of s and s[i:], i > 0
4     int n = s.size();
5     z[0] = 0;
6     for (int b = 0, i = 1; i < n; i++) {
7         if (z[b] + b <= i) z[i] = 0;
8         else z[i] = min(z[i - b], z[b] + b - i);
9         while (s[i + z[i]] == s[z[i]]) z[i]++;
10        if (i + z[i] > b + z[b]) b = i;
11    }
12 }

```

7.7. Manacher's Algorithm

```

1 int z[n];
2 void manacher(string s) {
3     // z[i] => longest odd palindrome centered at i is
4     // s[i - z[i] ... i + z[i]]
5     // to get all palindromes (including even length),
6     // insert a '#' between each s[i] and s[i + 1]
7     int n = s.size();
8     z[0] = 0;
9     for (int b = 0, i = 1; i < n; i++) {
10        if (z[b] + b >= i)
11            z[i] = min(z[2 * b - i], b + z[b] - i);
12        else z[i] = 0;
13        while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
14            s[i + z[i] + 1] == s[i - z[i] - 1])
15            z[i]++;
16        if (z[i] + i > z[b] + b) b = i;
17    }
18 }

```

7.8. Minimum Rotation

```

1 int min_rotation(string s) {
2     int a = 0, n = s.size();
3     s += s;
4     for (int b = 0; b < n; b++) {
5         for (int k = 0; k < n; k++) {
6             if (a + k == b || s[a + k] < s[b + k]) {
7                 b += max(0, k - 1);
8                 break;
9             }
10            if (s[a + k] > s[b + k]) {
11                a = b;
12                break;
13            }
14        }
15    }
16    return a;
17 }

```

7.9. Palindromic Tree

```

1
2
3 struct palindromic_tree {
4     struct node {
5         int next[26], fail, len;
6         int cnt,
7         num; // cnt: appear times, num: number of pal. suf.
8         node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
9             for (int i = 0; i < 26; ++i) next[i] = 0;
10        }
11    };
12    vector<node> St;
13    vector<char> s;
14    int last, n;
15    palindromic_tree() : St(2), last(1), n(0) {
16        St[0].fail = 1, St[1].len = -1, s.pb(-1);
17    }
18    inline void clear() {
19        St.clear(), s.clear(), last = 1, n = 0;
20        St.pb(0), St.pb(-1);
21        St[0].fail = 1, s.pb(-1);
22    }
23    inline int get_fail(int x) {
24        while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;

```

```

25     return x;
26 }
27 inline void add(int c) {
28     s.push_back(c -= 'a'), ++n;
29     int cur = get_fail(last);
30     if (!St[cur].next[c]) {
31         int now = SZ(St);
32         St.pb(St[cur].len + 2);
33         St[now].fail = St[get_fail(St[cur].fail)].next[c];
34         St[cur].next[c] = now;
35         St[now].num = St[St[now].fail].num + 1;
36     }
37     last = St[cur].next[c], ++St[last].cnt;
38 }
39 inline void count() { // counting cnt
40     auto i = St.rbegin();
41     for (; i != St.rend(); ++i) {
42         St[i->fail].cnt += i->cnt;
43     }
44 }
45 inline int size() { // The number of diff. pal.
46     return SZ(St) - 2;
47 }
};

```

8. Debug List

- 1 - Pre-submit:
 - Did you make a typo when copying a template?
 - 3 - Test more cases if unsure.
 - Write a naive solution and check small cases.
 - 5 - Submit the correct file.
- 7 - General Debugging:
 - Read the whole problem again.
 - 9 - Have a teammate read the problem.
 - Have a teammate read your code.
 - 11 - Explain you solution to them (or a rubber duck).
 - Print the code and its output / debug output.
 - 13 - Go to the toilet.
- 15 - Wrong Answer:
 - Any possible overflows?
 - 17 - > `__int128` ?
 - Try `-ftrapv` or `#pragma GCC optimize("trapv")`
 - 19 - Floating point errors?
 - > `long double` ?
 - 21 - turn off math optimizations
 - check for `'=='`, `'>='`, `acos(1.000000001)`, etc.
 - 23 - Did you forget to sort or unique?
 - Generate large and worst "corner" cases.
 - 25 - Check your `'m' / 'n'`, `'i' / 'j'` and `'x' / 'y'`.
 - Are everything initialized or reset properly?
 - 27 - Are you sure about the STL thing you are using?
 - Read cppreference (should be available).
 - 29 - Print everything and run it on pen and paper.
- 31 - Time Limit Exceeded:
 - Calculate your time complexity again.
 - 33 - Does the program actually end?
 - Check for `while(q.size())` etc.
 - 35 - Test the largest cases locally.
 - Did you do unnecessary stuff?
 - 37 - e.g. pass vectors by value
 - e.g. `memset` for every test case
 - 39 - Is your constant factor reasonable?
- 41 - Runtime Error:
 - Check memory usage.
 - 43 - Forget to clear or destroy stuff?
 - > `vector::shrink_to_fit()`
 - 45 - Stack overflow?
 - Bad pointer / array access?
 - 47 - Try `-fsanitize=address`
 - Division by zero? NaN's?