

Serso Library

Table of Contents

- [DP](#)
 - [Convex Hull](#)
 - [CHTmaxline.cpp](#)
 - [CHTminline.cpp](#)
 - [LineContainerMaxdouble.cpp](#)
 - [LineContainerMindouble.cpp](#)
 - [LineContainerminLine.cpp](#)
 - [LinecontainermaxLine.cpp](#)
 - [Divide and Conquer](#)
 - [DPdnc.cpp](#)
 - [Knapsack](#)
 - [knapsackwithbitset.cpp](#)
 - [optimizedknapsack.cpp](#)
 - [Knuth's](#)
 - [DPknuthscpAlgo.cpp](#)
 - [Sum of subset](#)
 - [SOSDP.cpp](#)
 - [Zero matrix](#)
 - [largestzerosubmatrix.cpp](#)
- [Data Structure](#)
 - [2D Range Queries](#)
 - [2Dprefixsum.cpp](#)
 - [2dsparsetable.cpp](#)
 - [3Dprefixsum.cpp](#)
 - [BIT2D.cpp](#)
 - [BIT2Dnlogn2.cpp](#)
 - [segTree2d.cpp](#)
 - [segmentsparsetable2d.cpp](#)
 - [BIT](#)
 - [BIT.cpp](#)
 - [BITLAZYMODIFICATION.cpp](#)
 - [BITRanges.cpp](#)

- [BITwithcoordinatecompression.cpp](#)
- [Multisetfenwick.cpp](#)
- [mexusingBIT.cpp](#)
- [DSU](#)
 - [DSU.cpp](#)
 - [DynamicConnectivityoffline.cpp](#)
 - [RollbackUF.cpp](#)
- [DSU on tree](#)
 - [sackDFSkeepBigChild.cpp](#)
 - [smalltolargetrick.cpp](#)
- [Divide and conquer](#)
 - [offlinestaticrmqdc.cpp](#)
 - [onlineRangeQueryD&Cbenq.cpp](#)
 - [onlinestaticrmqdc.cpp](#)
- [Kd-tree](#)
 - [kd-tree 3d.cpp](#)
 - [kd-tree kactl.cpp](#)
- [LCA](#)
 - [LCA.cpp](#)
 - [LCAinDSU.cpp](#)
 - [lcaO\(1\).cpp](#)
- [Minimum spanning tree](#)
 - [MSTKruskal.cpp](#)
 - [MSTPrim's.cpp](#)
- [Montomic stack and Queue](#)
 - [NxtorPrvMinorMax.cpp](#)
 - [monotonicQueue.cpp](#)
- [Offline Range Queries](#)
 - [Arithmeticprogressionprefixsum.cpp](#)
- [Ordered set](#)
 - [orderedset.cpp](#)
- [Persistent Segment tree](#)
 - [PersistentSegmentTreecounter.cpp](#)
 - [PersistentSegmentTreepointer.cpp](#)
 - [PresistentSegmentTreeserso.cpp](#)
- [SQRT and Mo's](#)
 - [MO-Algorithm.cpp](#)

- [Segment tree](#)
 - [SparseSegmentTree.cpp](#)
 - [segmentTreeiterative.cpp](#)
 - [segtreeBeat.cpp](#)
 - [segtreeBeatforMod.cpp](#)
- [Sparse table](#)
 - [getIdxSparseTable.cpp](#)
 - [sparsesegmentTree.cpp](#)
 - [sparsetable.cpp](#)
- [Treap](#)
 - [Treap Builtin.cpp](#)
 - [Treap Implicit.cpp](#)
 - [Treap.cpp](#)
- [Xor Basis](#)
 - [combiningtwoxorbasis.cpp](#)
 - [minOrMaxXorPathFrom1ToN.cpp](#)
 - [xorbasisapplication.cpp](#)
 - [xorbasislexicographicallylargest.cpp](#)
- [Graph](#)
 - [01BFS.cpp](#)
 - [BFS.cpp](#)
 - [Bridges.cpp](#)
 - [Hierholzer.cpp](#)
 - [SCCFloyed.cpp](#)
 - [SCCkosaraju.cpp](#)
 - [SCCTarjan.cpp](#)
 - [artPoints.cpp](#)
 - [bellmanford.cpp](#)
 - [biConnected.cpp](#)
 - [dijkstra.cpp](#)
 - [dijkstrasparsegraph.cpp](#)
 - [eulerciruiteundirectedgraph.cpp](#)
 - [eulerpathdirected.cpp](#)
 - [eulerpathundirected.cpp](#)
 - [floyd.cpp](#)
 - [kthshortestpath.cpp](#)
- [Handbook material](#)
- [Imgs](#)

- [Masking](#)
 - [GeneratenextlexicographicalK-combination.cpp](#)
 - [Generatingallsubmaskingofkones.cpp](#)
 - [genallpossiblesubmasking.cpp](#)
- [Math](#)
 - [Counting](#)
 - [BurnsideLemma.cpp](#)
 - [FFT.cpp](#)
 - [JosephusTheorem.cpp](#)
 - [MatrixExponential.cpp](#)
 - [Mint.cpp](#)
 - [NTTprimemod.cpp](#)
 - [NTTrandommod.cpp](#)
 - [checkpointinsidepolygon.cpp](#)
 - [convolutionnttoptimized.cpp](#)
 - [convolutionusingNTTkactl.cpp](#)
 - [derangement.cpp](#)
 - [evaluatingpolynomialatnpoints.cpp](#)
 - [lucas.cpp](#)
 - [nCr.cpp](#)
 - [nCrDP.cpp](#)
 - [nCrO\(r\).cpp](#)
 - [nCrmod2.cpp](#)
 - [ncrO\(n\).cpp](#)
 - [nttoptimized.cpp](#)
 - [simpsonintegration.cpp](#)
 - [startbars.cpp](#)
 - [stirling.cpp](#)
 - [sumxorrange.cpp](#)
 - [Game Theory](#)
 - [Grundy_01.cpp](#)
 - [Grundy_02.cpp](#)
 - [Grundy_String_Game.cpp](#)
 - [GrundyKnightsMoveInMatrix.cpp](#)
 - [Matrix Nim.cpp](#)
 - [Misere Nim.cpp](#)
 - [Nim_1.cpp](#)
 - [Staircase Nim.cpp](#)

- [General](#)
 - [int128.cpp](#)
 - [compdouble.cpp](#)
 - [convertdoubleint.cpp](#)
 - [floorceil.cpp](#)
 - [fraction.cpp](#)
 - [generatendigitsafterpoint.cpp](#)
 - [manualmultiply.cpp](#)
 - [random.cpp](#)
- [Geometry](#)
 - [3dGeometry.cpp](#)
 - [Distinctlinedetecting.cpp](#)
 - [Geometry.cpp](#)
 - [Point.cpp](#)
 - [Triangle.cpp](#)
 - [areaOfreactangles.cpp](#)
 - [circle.cpp](#)
 - [convexhull.cpp](#)
 - [dynamicconvexhull.cpp](#)
 - [getClosestPair.cpp](#)
 - [line.cpp](#)
 - [minimumenclosingcircle.cpp](#)
 - [polygon.cpp](#)
 - [sweepline.cpp](#)
- [Number theory](#)
 - [CRT-Offline.cpp](#)
 - [CRT-Online.cpp](#)
 - [CRT-OnlineNonCoPrimeModuli.cpp](#)
 - [Euler Phi.cpp](#)
 - [LinearDiophantineEquation.cpp](#)
 - [binarygcd.cpp](#)
 - [discretelog.cpp](#)
 - [extendedeuclidean.cpp](#)
 - [gcd.cpp](#)
 - [gcdnegativeinteger.cpp](#)
 - [mobius.cpp](#)
 - [modularMultiplicativeInverse.cpp](#)
 - [optimizedsieveLPF.cpp](#)

- [Notes](#)
- [Searching](#)
 - [countingsort.cpp](#)
 - [doubleternarysearch.cpp](#)
 - [mergesort.cpp](#)
 - [minNumberOfSwapsToSortTwoBinaryString.cpp](#)
 - [patientsort.cpp](#)
 - [radixsort.cpp](#)
- [Snippet](#)
 - [clock.cpp](#)
 - [fastCode.cpp](#)
 - [gccoptimize.cpp](#)
- [Standard Problems](#)
 - [Geometry](#)
 - [checkpointinconvex.cpp](#)
 - [countpointsincircle.cpp](#)
 - [pointin-shape.cpp](#)
 - [Implementation](#)
 - [coloringmatrixwithlargestarea.cpp](#)
 - [dividegridintokonnectedcompented01.cpp](#)
 - [NumberTheory](#)
 - [countnosubsequecwithlcm.cpp](#)
 - [countsubsequecegcdequal1.cpp](#)
 - [Tree](#)
 - [Noofpathsinsidespaths.cpp](#)
- [String](#)
 - [Aho corasick](#)
 - [AhoCorasick.cpp](#)
 - [Basic strings Algo](#)
 - [KMP.cpp](#)
 - [Zalgorithm.cpp](#)
 - [failerfun.cpp](#)
 - [menacher.cpp](#)
 - [Hashing](#)
 - [DoubleHashasint.cpp](#)
 - [customunorderdmap.cpp](#)
 - [doublehashing.cpp](#)

- [hashinggrid.cpp](#)
- [k-hash.cpp](#)
- [stringhash.cpp](#)
- [Lyndon](#)
 - [Duval.cpp](#)
 - [mincyclicstring.cpp](#)
- [suffixArray](#)
 - [Kasai.cpp](#)
 - [RMQsuffixarray.cpp](#)
 - [buildsuffixArraysloO\(nlognlogn\).cpp](#)
 - [buildsuffixarrayveryfast.cpp](#)
 - [problems](#)
 - [countpattern.cpp](#)
 - [distinctsubstring.cpp](#)
 - [kthdistinctsubstring.cpp](#)
 - [kthsmalleststring.cpp](#)
 - [noofdistinctstringoflengthx.cpp](#)
 - [noofsubstringatleastf.cpp](#)
- [trie](#)
 - [Binarytrie.cpp](#)
 - [trie.cpp](#)
- [Trees](#)
 - [CentroidDecomposition.cpp](#)
 - [HLD.cpp](#)
 - [HLDedges.cpp](#)
 - [Treecenterusingbfs.cpp](#)
 - [Treeflatten.cpp](#)
 - [bridgestree.cpp](#)
 - [centroid.cpp](#)
 - [isophrisim.cpp](#)
 - [treecenter.cpp](#)
 - [treediameterdfs.cpp](#)

DP

Convex Hull

CHT_maxline.cpp

```

1  /*
2  * Description: Container where you can add lines of the form  $kx+m$ , and query minimum
  values at points  $x$ .
3  * Can be applied on:
4  - monotonic slopes, monotonic queries
5  - monotonic slopes, random queries
6
7  Here's monotonic can be (ASCE , DESC)
8  */
9  const ll INF = LLONG_MAX;
10 const ll NEG_INF = LLONG_MIN;
11
12 struct mxCHT { // Convex Hull Trick for Maximum
13     deque<long long> A; // Slopes
14     deque<long long> B; // Intercepts
15
16     // Modified 'bad' function with reversed inequality for Maximum CHT
17     bool bad(int l1, int l2, int l3) {
18         return (B[l3] - B[l1]) * (long double)(A[l1] - A[l2])
19             >= (B[l2] - B[l1]) * (long double)(A[l1] - A[l3]); // Changed <= to >=
20     }
21
22     // Adds a line with slope `a` and intercept `b` when slopes are in ascending order
23     void addASC(long long a, long long b) {
24         A.push_back(a);
25         B.push_back(b);
26         // Remove the second last line if it's unnecessary
27         while (A.size() >= 3 && bad(A.size() - 3, A.size() - 2, A.size() - 1)) {
28             A.erase(A.end() - 2);
29             B.erase(B.end() - 2);
30         }
31     }
32
33     // Adds a line with slope `a` and intercept `b` when slopes are in descending order
34     void addDESC(long long a, long long b) {
35         A.push_front(a);
36         B.push_front(b);
37         // Remove the second line if it's unnecessary
38         while (A.size() >= 3 && bad(0, 1, 2)) {
39             A.erase(A.begin() + 1);
40             B.erase(B.begin() + 1);
41         }
42     }

```



```

43
44 // Evaluates the line at index `l` for a given `x`
45 long long f(int l, long long x) {
46     return A[l] * x + B[l];
47 }
48
49 // Queries the convex hull for maximum value at `x` when x-values are increasing
50 long long queryASC(long long x) {
51     while (A.size() >= 2 && f(0, x) < f(1, x)) { // Changed '>' to '<'
52         // Remove the front line if it's not optimal
53         A.pop_front();
54         B.pop_front();
55     }
56     if (A.empty())
57         return NEG_INF; // Changed from LLONG_MAX to LLONG_MIN
58     return f(0, x);
59 }
60
61 // Queries the convex hull for maximum value at `x` when x-values are decreasing
62 long long queryDESC(long long x) {
63     while (A.size() >= 2 && f(A.size() - 1, x) < f(A.size() - 2, x)) { // Changed '>' to '<'
64         // Remove the back line if it's not optimal
65         A.pop_back();
66         B.pop_back();
67     }
68     if (A.empty())
69         return NEG_INF; // Changed from LLONG_MAX to LLONG_MIN
70     return f(A.size() - 1, x);
71 }
72
73 // Optional: Query for random x-values using binary search
74 long long query(long long x) {
75     int lo = 0, hi = A.size() - 1;
76     long long res = NEG_INF; // Changed from LLONG_MAX to LLONG_MIN
77     while (lo <= hi) {
78         int mid = (lo + hi) / 2;
79         long long val = f(mid, x);
80         res = max(res, val); // Changed from min to max
81         if (mid + 1 < A.size() && f(mid + 1, x) > val) { // Changed '<=' to '>'
82             lo = mid + 1;
83         }
84         else if (mid - 1 >= 0 && f(mid - 1, x) > val) { // Changed '<=' to '>'
85             hi = mid - 1;
86         }

```

```

87     else {
88         break;
89     }
90 }
91 return res;
92 }

```

CHT_minline.cpp

```

1  /*
2   * Description: Container where you can add lines of the form  $kx+m$ , and query minimum
   values at points  $x$ .
3   * Can be applied on:
4   - monotonic slopes, monotonic queries
5   - monotonic slopes, random queries
6
7   Here's monotonic can be (ASCE , DESC)
8   */
9  struct mnCHT { // Convex Hull Trick for Minimum
10     deque<long long> A; // Slopes
11     deque<long long> B; // Intercepts
12
13
14     bool bad(int I1, int I2, int I3) {
15         return (B[I3] - B[I1]) * (long double) (A[I1] - A[I2])
16             <= (B[I2] - B[I1]) * (long double) (A[I1] - A[I3]);
17     }
18
19     // Adds a line with slope `a` and intercept `b` when slopes are in ascending order
20     void addASC(long long a, long long b) {
21         A.push_back(a);
22         B.push_back(b);
23         // Remove the second last line if it's unnecessary
24         while (A.size() >= 3 && bad(A.size() - 3, A.size() - 2, A.size() - 1)) {
25             A.erase(A.end() - 2);
26             B.erase(B.end() - 2);
27         }
28     }
29
30     // Adds a line with slope `a` and intercept `b` when slopes are in descending order
31     void addDESC(long long a, long long b) {
32         A.push_front(a);

```

```

33     B.push_front(b);
34     // Remove the second line if it's unnecessary
35     while (A.size() >= 3 && bad(0, 1, 2)) {
36         A.erase(A.begin() + 1);
37         B.erase(B.begin() + 1);
38     }
39 }
40
41 // Evaluates the line at index `l` for a given `x`
42 long long f(int l, long long x) {
43     return A[l] * x + B[l];
44 }
45
46 // Queries the convex hull for minimum value at `x` when x-values are increasing
47 long long queryASC(long long x) {
48     while (A.size() >= 2 && f(0, x) > f(1, x)) {
49         // Remove the front line if it's not optimal
50         A.pop_front();
51         B.pop_front();
52     }
53     if (A.empty())
54         return LLONG_MAX; // Return maximum value if no lines are left
55     return f(0, x);
56 }
57 // Queries the convex hull for minimum value at `x` when x-values are decreasing
58 long long queryDESC(long long x) {
59     while (A.size() >= 2 && f(A.size() - 1, x) > f(A.size() - 2, x)) {
60         // Remove the back line if it's not optimal
61         A.pop_back();
62         B.pop_back();
63     }
64     if (A.empty())
65         return LLONG_MAX; // Return maximum value if no lines are left
66     return f(A.size() - 1, x);
67 }
68
69 // Optional: Query for random x-values using binary search
70 long long query(long long x) {
71     int lo = 0, hi = A.size() - 1;
72     long long res = LLONG_MAX;
73     while (lo <= hi) {
74         int mid = (lo + hi) / 2;
75         long long val = f(mid, x);
76         res = min(res, val);

```

```

77     if (mid + 1 < A.size() && f(mid + 1, x) <= val) {
78         lo = mid + 1;
79     } else if (mid - 1 >= 0 && f(mid - 1, x) <= val) {
80         hi = mid - 1;
81     } else {
82         break;
83     }
84 }
85 return res;
86 }

```

LineContainerMax_double.cpp

```

1  struct Line {
2      mutable double k, m, p;
3      bool operator<(const Line& o) const { return k < o.k; }
4      bool operator<(double x) const { return p < x; }
5  };
6
7  struct LineContainerMax : multiset<Line, less<>> {
8      static constexpr double inf = numeric_limits<double>::infinity();
9      static constexpr double EPS = 1e-9;
10
11     double div(double a, double b) {
12         return a / b;
13     }
14
15     bool isect(iterator x, iterator y) {
16         if (y == end()) return x->p = inf, 0;
17         if (abs(x->k - y->k) < EPS) x->p = x->m > y->m ? inf : -inf;
18         else x->p = div(y->m - x->m, x->k - y->k);
19         return x->p >= y->p;
20     }
21
22     void add(double k, double m) {
23         auto z = insert({k, m, 0}), y = z++, x = y;
24         while (isect(y, z)) z = erase(z);
25         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
26         while ((y = x) != begin() && (--x)->p >= y->p)
27             isect(x, erase(y));
28     }
29

```

```

30 double query(double x) {
31     assert(!empty());
32     auto l = *lower_bound(x);
33     return l.k * x + l.m;
34 }
35 };

```

LineContainerMin_double.cpp

```

1  struct Line {
2      mutable double k, m, p;
3      bool operator<(const Line& o) const { return k > o.k; }
4      bool operator<(double x) const { return p < x; }
5  };
6
7  struct LineContainerMin : multiset<Line, less<>> {
8      static constexpr double inf = numeric_limits<double>::infinity();
9      static constexpr double EPS = 1e-9;
10     double div(double a, double b) {
11         return a / b;
12     }
13
14     bool isect(iterator x, iterator y) {
15         if (y == end()) return x->p = inf, 0;
16         if (abs(x->k - y->k) < EPS) x->p = x->m < y->m ? inf : -inf;
17         else x->p = div(y->m - x->m, x->k - y->k);
18         return x->p >= y->p;
19     }
20
21     void add(double k, double m) {
22         auto z = insert({k, m, 0}), y = z++, x = y;
23         while (isect(y, z)) z = erase(z);
24         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
25         while ((y = x) != begin() && (--x)->p >= y->p)
26             isect(x, erase(y));
27     }
28
29     double query(double x) {
30         assert(!empty());
31         auto l = *lower_bound(x);
32         return l.k * x + l.m;
33     }

```

34 };

LineContainer_minLine.cpp

```

1  /*
2  * Description: Container where you can add lines of the form  $kx+m$ , and query minimum
  values at points  $x$ .
3  * Can be applied on:
4  - monotonic slopes, monotonic queries
5  - monotonic slopes, random queries
6  - random slopes, random queries
7  */
8  struct Line {
9      mutable ll k, m, p;
10     bool operator<(const Line& o) const { return k > o.k; }
11     bool operator<(ll x) const { return p < x; }
12 };
13
14 struct LineContainerMin : multiset<Line, less<>> {
15     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
16     static const ll inf = LLONG_MAX;
17     ll div(ll a, ll b) { // floored division
18         return a / b - ((a ^ b) < 0 && a % b); }
19     bool isect(iterator x, iterator y) {
20         if (y == end()) return x->p = inf, 0;
21         if (x->k == y->k) x->p = x->m < y->m ? inf : -inf;
22         else x->p = div(y->m - x->m, x->k - y->k);
23         return x->p >= y->p;
24     }
25     void add(ll k, ll m) {
26         auto z = insert({k, m, 0}), y = z++, x = y;
27         while (isect(y, z)) z = erase(z);
28         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
29         while ((y = x) != begin() && (--x)->p >= y->p)
30             isect(x, erase(y));
31     }
32     ll query(ll x) {
33         assert(!empty());
34         auto l = *lower_bound(x);
35         return l.k * x + l.m;
36     }
37 };

```

Linecontainer_maxLine.cpp

```

1  /*
2  * Description: Container where you can add lines of the form  $kx+m$ , and query
    maximum values at points  $x$ .
3  * Can be applied on:
4  - monotonic slopes, monotonic queries
5  - monotonic slopes, random queries
6  - random slopes, random queries
7  */
8
9  struct Line {
10     mutable ll k, m, p;
11     bool operator<(const Line& o) const { return k < o.k; }
12     bool operator<(ll x) const { return p < x; }
13 };
14
15 struct LineContainerMax : multiset<Line, less<>> {
16     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
17     static const ll inf = LLONG_MAX;
18     ll div(ll a, ll b) { // floored division
19         return a / b - ((a ^ b) < 0 && a % b); }
20     bool isect(iterator x, iterator y) {
21         if (y == end()) return x->p = inf, 0;
22         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
23         else x->p = div(y->m - x->m, x->k - y->k);
24         return x->p >= y->p;
25     }
26     void add(ll k, ll m) {
27         auto z = insert({k, m, 0}), y = z++, x = y;
28         while (isect(y, z)) z = erase(z);
29         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
30         while ((y = x) != begin() && (--x)->p >= y->p)
31             isect(x, erase(y));
32     }
33     ll query(ll x) {
34         assert(!empty());
35         auto l = *lower_bound(x);
36         return l.k * x + l.m;
37     }
38 };

```

Divide and Conquer

DP_dnc.cpp

```

1  const int N = 5002;
2  int n,k;
3  ll dp[2][N] , p[N] , c[N][N];
4  int cost(int l , int r){ return; } // cost function
5  void dnc(int ind , int l , int r , int optl , int optr){
6      if(l > r) return;
7      pair<ll,int> bst{LLONG_MAX,-1};
8      int m = (l+r)/2;
9      for(int opt = optl; opt <= optr; ++opt){
10         /*
11             In case 0-base
12             bst = min(bst , { (opt ? dp[ind^1][opt-1] : 0) + c(opt,m) , opt });
13         */
14         bst = min(bst , {dp[ind^1][opt-1] + c(opt,m) , opt} );
15     }
16     dp[ind][m] = bst.f;
17     int opt = bst.s;
18     dnc(ind,l,m-1,optl,opt);
19     dnc(ind,m+1,r,opt,optr);
20 }
21 void run(){
22     for(int i = 1; i <= n; ++i) dp[0][i] = c[1][i];
23     for(int x = 1; x < k; ++x){
24         dnc(x&1,1,n,1,n);
25     }
26 }

```

Knapsack

knapsack_with_bitset.cpp

```

1  /*
2      optimized Knapsack problem work if
3      n <= 2e5
4      a1 + a2 + ... + an <= 1e6
5      Time complexity: O(n * sqrt(n))
6  */

```



```

7  template<int len = 1>
8  int solve(const vector<int> &v , int all){
9      if(v.empty()) return 0;
10     if(len < all)
11         return solve<min(2 * len , N)>(v, all);
12     map<int,int> frq;
13     int Max = 0;
14     for(auto &i : v) frq[i]++ , Max = max(Max , i);
15     // specific for a problem, think of something that can optimize the code.
16     vector<int> cur;
17     for(auto &[x , f] : frq){
18         int need = 1;
19         while(f){
20             need = min(need , f);
21             f -= need;
22             cur.emplace_back(need * x);
23             need *= 2;
24         }
25     }
26     bitset<len> dp;
27     dp[0] = 1;
28     for(auto &i : cur) dp |= (dp << i);
29     // specific for a problem.
30 }

```

optimized_knapsack.cpp

```

1  /*
2   Problem: For every possible sum , find the minimum number of elements that can form
   this sum.
3   Time complexity: O(n sqrt(n))
4   */
5   map<int,int> frq; // store the frq of every element in the array.
6   int dp[n + 1];
7   memset(dp , '?' , sizeof dp);
8   dp[0] = 0;
9   for(auto &[sz , f] : frq){
10       int cost = 1;
11       while(f){
12           cost = min(cost , f);
13           f -= cost;
14           for(int sum = n; sum >= cost * sz; sum--)

```

```

15     dp[sum] = min(dp[sum] , dp[sum - cost * sz] + cost);
16     cost *= 2;
17 }
18 }

```

Knuth's

DP_knuths_cpAlgo.cpp

```

1  int solve() {
2      int N;
3      ... // read N and input
4      int dp[N][N], opt[N][N];
5
6      auto C = [&](int i, int j) {
7          ... // Implement cost function C.
8      };
9
10     for (int i = 0; i < N; i++) {
11         opt[i][i] = i;
12         ... // Initialize dp[i][i] according to the problem
13     }
14
15     for (int i = N-2; i >= 0; i--) {
16         for (int j = i+1; j < N; j++) {
17             int mn = INT_MAX;
18             int cost = C(i, j);
19             for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j]); k++) {
20                 if (mn >= dp[i][k] + dp[k+1][j] + cost) {
21                     opt[i][j] = k;
22                     mn = dp[i][k] + dp[k+1][j] + cost;
23                 }
24             }
25             dp[i][j] = mn;
26         }
27     }
28
29     return dp[0][N-1];
30 }

```

Sum of subset

SOS_DP.cpp

```

1  const int N = 20;
2  int dp[1<<N];
3  int comb(int x , int y){
4      // depend on the problem
5      return x+y;
6  }
7  void mask_to_supermask(){
8      // Combing every msk to all its supermask
9      for(int x = 0; x < N; ++x){
10         for(int msk = 0; msk < (1<<N); msk++){
11             if(msk>>x&1) dp[msk] = comb(dp[msk] , dp[msk^(1<<x)]);
12         }
13     }
14 }
15 void mask_to_submask(){
16     //Combing every msk to all its submask
17     for(int x = 0; x < N; ++x){
18         for(int msk = (1<<N)-1; msk >= 0; msk--){
19             if(msk>>x&1) dp[msk^(1<<x)] = comb(dp[msk^(1<<x)] , dp[msk]);
20         }
21     }
22 }
23 void supermask_to_mask(){
24     // Combing every supermask to mask
25     for(int x = 0; x < N; ++x){
26         for(int msk = (1<<N) - 1; msk >= 0; --msk){
27             if(!(msk>>x&1)) dp[msk] = comb(dp[msk] , dp[msk|(1<<x)]);
28         }
29     }
30 }
31 void submask_to_mask(){
32     // combining every submask to mask
33     for(int x = 0; x < N; ++x){
34         for(int msk = 0; msk < N; ++msk){
35             if(!(msk>>x&1)) dp[msk | (1<<x)] = comb(dp[msk | (1<<x)] , dp[ms]);
36         }
37     }
38 }

```

Zero matrix

largest_zero_submatrix.cpp

```

1  /*
2   You are given a matrix with n rows and m columns.
3   Find the largest submatrix consisting of only zeros (a submatrix is a rectangular area
   of the matrix).
4   Elements of the matrix will be a[i][j], where i = 0...n - 1, j = 0... m - 1.
5   For simplicity, we will consider all non-zero elements equal to 1.
6
7   Time Complexity: O(n x m)
8   Space Complexity: O(m)
9  */
10 int zero_matrix(vector<vector<int>> a) {
11     int n = a.size();
12     int m = a[0].size();
13
14     int ans = 0;
15     vector<int> d(m, -1), d1(m), d2(m);
16     stack<int> st;
17     for (int i = 0; i < n; ++i) {
18         for (int j = 0; j < m; ++j) {
19             if (a[i][j] == 1)
20                 d[j] = i;
21         }
22
23         for (int j = 0; j < m; ++j) {
24             while (!st.empty() && d[st.top()] <= d[j])
25                 st.pop();
26             d1[j] = st.empty() ? -1 : st.top();
27             st.push(j);
28         }
29         while (!st.empty())
30             st.pop();
31
32         for (int j = m - 1; j >= 0; --j) {
33             while (!st.empty() && d[st.top()] <= d[j])
34                 st.pop();
35             d2[j] = st.empty() ? m : st.top();
36             st.push(j);
37         }
38         while (!st.empty())
39             st.pop();
40

```

```

41     for (int j = 0; j < m; ++j)
42         ans = max(ans, (i - d[j]) * (d2[j] - d1[j] - 1));
43     }
44     return ans;
45 }

```

Data Structure

2D Range Queries

2D_prefixsum.cpp

```

1  const int N = 500, M = 500;
2  int pref[N][M]; // 1-base
3  int n, m;
4  int qry(int from_x, int from_y, int to_x, int to_y){
5      return pref[to_x][to_y] - pref[from_x - 1][to_y] - pref[to_x][from_y - 1] + pref[from_x
6      - 1][from_y - 1];
7  }
8  void upd(int from_x, int from_y, int to_x, int to_y, int val){
9      pref[from_x][from_y] += val;
10     pref[to_x + 1][from_y] -= val;
11     pref[from_x][to_y + 1] -= val;
12     pref[to_x + 1][to_y + 1] += val;
13 }
14 void build(){
15     for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j){
16         pref[i][j] += pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1];
17     }
18 }
19 void init(){
20     for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j){
21         pref[i][j] = 0;
22     }
23 }

```

2d_sparsetable.cpp

```

1  /*
2      2D sparse table
3      - Memory allocation: O(nlog(n) x mlog(m))

```

```

4   - range [l..r] , r: inclusive
5   - Base: 0-index
6
7   Function description:
8       1. build(n , m , g): Build 2D sparse table of g[n][m]
9       2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
10      Time complexity: O(1)
11  */
12  #define vi vector<int>
13  #define vi2 vector<vi>
14  #define vi3 vector<vi2>
15  #define vi4 vector<vi3>
16  #define rep(i , st , ed) for(int i = st; i < ed; i++)
17  struct sparse_2d{
18      vi4 f;
19      void init(int n , int m , vi2 &g){
20          int lgN = 31 - __builtin_clz(n + 1);
21          int lgM = 31 - __builtin_clz(m + 1);
22          f = vi4(n , vi3(m , vi2(lgN , vi(lgM))));
23          rep(i , 0 , n) rep(j , 0 , m) f[i][j][0][0] = g[i][j];
24
25          rep(k1 , 0 , lgN) rep(k2 , 0 , lgM) if(k1 || k2){
26              rep(i , 0 , n - (1 << k1) + 1) rep(j , 0 , m - (1 << k2) + 1){
27                  if(k1 > 0) {
28                      f[i][j][k1][k2] = comb(f[i][j][k1 - 1][k2], f[i + (1 << (k1 - 1))][j][k1 - 1][k2]);
29                  }
30                  if(k2 > 0) {
31                      f[i][j][k1][k2] = comb(f[i][j][k1][k2 - 1], f[i][j + (1 << (k2 - 1))][k1][k2 - 1]);
32                  }
33                  if(k1 > 0 && k2 > 0) {
34                      f[i][j][k1][k2] = comb(comb(
35                          f[i][j][k1 - 1][k2 - 1],
36                          f[i + (1 << (k1 - 1))][j][k1 - 1][k2 - 1], comb(
37                          f[i][j + (1 << (k2 - 1))][k1 - 1][k2 - 1],
38                          f[i + (1 << (k1 - 1))][j + (1 << (k2 - 1))][k1 - 1][k2 - 1]
39                      )
40                  );
41              }
42          }
43      }
44  }
45  int comb(int a, int b){
46      return __gcd(a , b); // custom operator
47  }

```

```

48     int qry(int l, int d, int r, int u) {
49         int k1 = 31 - __builtin_clz(r - l + 1);
50         int k2 = 31 - __builtin_clz(u - d + 1);
51         return comb(comb(
52             f[l][d][k1][k2],
53             f[r - (1 << k1) + 1][d][k1][k2]),
54             comb(
55                 f[l][u - (1 << k2) + 1][k1][k2],
56                 f[r - (1 << k1) + 1][u - (1 << k2) + 1][k1][k2])
57     );
58 }

```

3D_prefixsum.cpp

```

1     const int N = 500, M = 500, K = 500;
2     int P[N][M][K]; // 1-base
3     int n, m, k;
4     int qry(int from_x, int from_y, int from_z, int to_x, int to_y, int to_z){
5         int result = P[to_x][to_y][to_z]
6             - P[from_x-1][to_y][to_z]
7             - P[to_x][from_y-1][to_z]
8             - P[to_x][to_y][from_z-1]
9             + P[from_x-1][from_y-1][to_z]
10            + P[from_x-1][to_y][from_z-1]
11            + P[to_x][from_y-1][from_z-1]
12            - P[from_x-1][from_y-1][from_z-1];
13    }
14    void upd(int from_x, int from_y, int from_z, int to_x, int to_y, int to_z, int val){
15        P[from_x][from_y][from_z] += val;
16        P[to_x + 1][from_y][from_z] -= val;
17        P[from_x][to_y + 1][from_z] -= val;
18        P[to_x][from_y][to_z + 1] -= val;
19        P[to_x + 1][to_y + 1][from_z] += val;
20        P[to_x + 1][from_y][to_z + 1] += val;
21        P[from_x][to_y + 1][to_z + 1] += val;
22        P[to_x + 1][to_y + 1][to_z + 1] -= val;
23    }
24    void build(vector<vector<vector<int>>> &A){
25        for (int x = 1; x <= N; ++x) {
26            for (int y = 1; y <= N; ++y) {
27                for (int z = 1; z <= N; ++z) {
28                    P[x][y][z] = A[x][y][z]

```

```

29         + P[x-1][y][z]
30         + P[x][y-1][z]
31         + P[x][y][z-1]
32         - P[x-1][y-1][z]
33         - P[x-1][y][z-1]
34         - P[x][y-1][z-1]
35         + P[x-1][y-1][z-1];
36     }
37 }
38 }
39 }
40 void init(){
41     for(int i = 1; i <= n; ++i) for(int j = 1; j <= m; ++j) for(int k = 1; k <= K; ++k){
42         P[i][j][k] = 0;
43     }
44 }

```

BIT2D.cpp

```

1  /*
2  -----BIT2D-----
3  - Memory allocation: O(n x m)
4  - range [l..r] , r: inclusive
5  - Base: 0-index
6
7  Function description:
8  1. init(n , m): initial a BIT with 2D grid[n][m]
9     Time complexity: O(n x 4)
10  2. upd(x , y , val): add to a cell(x ,y) value "val"
11     Time complexity: O( log(n) x log(m) ) , here log is small constanst
12  2. qry(lx , rx , ly , ry): find (sum,xor) of rectangle [lx...rx] x [ly...ry]
13     Time complexity: O(log(n) x log(m))
14
15  */
16  template<class T = int>
17  struct BIT2D {
18  private:
19      vector<vector<T>> tree;
20      int n , m;
21      T sum(int x, int y) {
22          T ret = 0;
23          for (int i = x + 1; i <= (i & (-i)); i += i & (-i)) {
24              for (int j = y + 1; j <= (j & (-j)); j += j & (-j)) {

```



```

25         ret += tree[i - 1][j - 1];
26     }
27 }
28 return ret;
29 }
30 public:
31 void init(int n, int m) {
32     this->n = n; this->m = m;
33     tree.assign(n, vector<T>(m, 0));
34 }
35 void upd(int x, int y, T val) {
36     for (int i = x + 1; i <= n; i += (i & (-i))) {
37         for (int j = y + 1; j <= m; j += (j & (-j))) {
38             tree[i - 1][j - 1] += val;
39         }
40     }
41 }
42 T qry(int sx, int sy, int ex, int ey) {
43     return sum(ex, ey) - sum(ex, sy - 1) - sum(sx - 1, ey) + sum(sx - 1, sy - 1);
44 }
45 T qry(int x, int y){ return sum(x, y, x, y); }
46 };

```

BIT2D_nlogn2.cpp

```

1 // O(N(logN)^2)
2 template<class T = int>
3 struct Bit2D {
4     vector<T> ord;
5     vector<vector<T>> fw, coord;
6
7     // pts needs all points that will be used in the upd
8     // if range upds remember to build with {x1, y1}, {x1, y2 + 1}, {x2 + 1, y1}, {x2 + 1, y2 + 1}
9     Bit2D(vector<pair<T, T>> pts) {
10         sort(pts.begin(), pts.end());
11         for (auto a : pts)
12             if (ord.empty() || a.first != ord.back())
13                 ord.push_back(a.first);
14         fw.resize(ord.size() + 1);
15         coord.resize(fw.size());
16
17         for (auto& a : pts)

```

```

18     swap(a.first, a.second);
19     sort(pts.begin(), pts.end());
20     for (auto& a : pts) {
21         swap(a.first, a.second);
22         for (int on = std::upper_bound(ord.begin(), ord.end(), a.first) - ord.begin(); on <
fw.size(); on += on & -on)
23             if (coord[on].empty() || coord[on].back() != a.second)
24                 coord[on].push_back(a.second);
25     }
26
27     for (int i = 0; i < fw.size(); i++)
28         fw[i].assign(coord[i].size() + 1, 0);
29 }
30 T merge(T a, T b){
31     return a + b;
32 }
33 // point upd
34 void upd(T x, T y, T v) {
35     for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx < fw.size(); xx +=
xx & -xx)
36         for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) -
coord[xx].begin(); yy < fw[xx].size(); yy += yy & -yy)
37             fw[xx][yy] = merge(fw[xx][yy], v);
38 }
39
40 // point qry
41 T qry(T x, T y) {
42     T ans = 0;
43     for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx > 0; xx -= xx & -
xx)
44         for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) -
coord[xx].begin(); yy > 0; yy -= yy & -yy)
45             ans = merge(ans, fw[xx][yy]);
46     return ans;
47 }
48
49 // range qry
50 T qry(T x1, T y1, T x2, T y2) {
51     return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) + qry(x1 - 1, y1 - 1);
52 }
53
54 // range upd
55 void upd(T x1, T y1, T x2, T y2, T v) {
56     upd(x1, y1, v);

```

```

57     upd(x1, y2 + 1, -v);
58     upd(x2 + 1, y1, -v);
59     upd(x2 + 1, y2 + 1, v);
60 }

```

segTree_2d.cpp

```

1  /*
2   2D segment Tree
3   - Memory allocation: O(4n x 4m)
4   - range [l..r] , r: inclusive
5   - Base: 0-index
6
7   Function description:
8   1. upd(r , c , val): update cell (r,c) with value val
9   Time complexity: O(log(n) x log(m))
10  2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
11  Time complexity: O(log(n) x log(m))
12  3. init(n , m): initial the segment tree
13  4. build(g): build the segment tree with grid[n][m]
14  Time complexity: O(n log(n) x m log(m))
15  */
16  struct Node{
17      int v;
18      Node(){ v = 0;}
19      Node(int x){ this->v = x; }
20      Node operator +(const Node &other) const{
21          Node res;
22          res.v = __gcd(v , other.v); // custom operator
23          return res;
24      }
25  };
26  struct segTree_2d{
27  private:
28      vector<vector<Node>> > t;
29      int n , m;
30      void build_y(int vx, int lx, vector<vector<int>> &g ,int rx, int vy, int ly, int ry) {
31          if (ly == ry) {
32              if (lx == rx){
33                  t[vx][vy] = Node(g[lx][ly]);
34              }
35              else{

```

```

36         t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
37     }
38 } else {
39     int my = (ly + ry) / 2;
40     build_y(vx, lx, g, rx, vy*2, ly, my);
41     build_y(vx, lx, g, rx, vy*2+1, my+1, ry);
42     t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
43 }
44 }
45 void build_x(vector<vector<int>> &g, int vx, int lx, int rx) {
46     if (lx != rx) {
47         int mx = (lx + rx) / 2;
48         build_x(g, vx*2, lx, mx);
49         build_x(g, vx*2+1, mx+1, rx);
50     }
51     build_y(vx, lx, g, rx, 1, 0, m-1);
52 }
53 Node qry_y(int vx, int vy, int tly, int try_, int ly, int ry) {
54     if (ly > ry)
55         return Node();
56     if (ly == tly && try_ == ry)
57         return Node(t[vx][vy]);
58     int tmy = (tly + try_) / 2;
59     return qry_y(vx, vy*2, tly, tmy, ly, min(ry, tmy)) +
60         qry_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
61 }
62
63 Node qry_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
64     if (lx > rx)
65         return Node();
66     if (lx == tlx && trx == rx)
67         return qry_y(vx, 1, 0, m-1, ly, ry);
68     int tmx = (tlx + trx) / 2;
69     return qry_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry) +
70         qry_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
71 }
72 void upd_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int y, int nval) {
73     if (ly == ry) {
74         if (lx == rx)
75             t[vx][vy] = nval;
76         else
77             t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
78     } else {
79         int my = (ly + ry) / 2;

```

```

80     if (y <= my)
81         upd_y(vx, lx, rx, vy*2, ly, my, x, y, nval);
82     else
83         upd_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, nval);
84     t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
85 }
86 }
87 void upd_x(int vx, int lx, int rx, int x, int y, int nval) {
88     if (lx != rx) {
89         int mx = (lx + rx) / 2;
90         if (x <= mx)
91             upd_x(vx*2, lx, mx, x, y, nval);
92         else
93             upd_x(vx*2+1, mx+1, rx, x, y, nval);
94     }
95     upd_y(vx, lx, rx, 1, 0, m-1, x, y, nval);
96 }
97 public:
98     void init(int n, int m){
99         this->n = n;
100        this->m = m;
101        int r = 1, c = 1;
102        while(r < n) r *= 2;
103        while(c < m) c *= 2;
104        t = vector<vector<Node>>(2 * r, vector<Node>(2 * c));
105    }
106    void build(vector<vector<int>> &g){
107        build_x(g, 1, 0, n - 1);
108    }
109    int qry(int x, int y, int xx, int yy){
110        return qry_x(1, 0, n - 1, x, xx, y, yy).v;
111    }
112    void upd(int r, int c, int val){
113        upd_x(1, 0, n - 1, r, c, val);
114    }

```

segment_sparse_table_2d.cpp

```

1  /*
2   2D segment sparse table
3   - Memory allocation: O(4n x mlog(m))
4   - range [l..r] , r: inclusive

```

```

5   - Base: 0-index
6
7   Function description:
8       1. build(n , m , n): build the segment tree with grid[n][m]
9           Time complexity: O(4n x mlog(m))
10      2. qry(lx , rx , ly , ry): find (min,max,sum,produc) of rectangle [lx...rx] x [ly...ry]
11           Time complexity: O(log(n))
12  */
13  template<class T = int>
14  struct sparse_2d {
15      int n, m, LOG , N;
16      T DEFAULT = 0;
17      vector<vector<vector<T>>> tree;
18
19      void build(int _n, int _m , vector<vector<T>> &a) {
20          n = _n;
21          m = _m;
22          LOG = 31 - __builtin_clz(m) + 1;
23
24          N = 1;
25          while(N < n) N *= 2; // N must be power of 2
26          tree = vector<vector<vector<T>>>(2 * N + 1, vector<vector<T>>(LOG, vector<T>(m
+ 1)));
27
28          for (int x = 0; x < n; x++)
29          {
30              for (int y = 0; y < m; y++)
31                  tree[N + x][0][y] = a[x][y];
32              for (int k = 1; k < LOG; k++)
33                  for (int y = 0; y + (1 << k) <= m; y++)
34                      tree[N + x][k][y] = comb(tree[N + x][k - 1][y], tree[N + x][k - 1][y + (1 << (k - 1))]);
35          }
36          for (int v = N - 1; v > 0; v--)
37              for (int k = 0; k < LOG; k++)
38                  for (int y = 0; y + (1 << k) <= m; y++)
39                      tree[v][k][y] = comb(tree[2 * v][k][y], tree[2 * v + 1][k][y]);
40          return;
41      }
42
43      T comb(T a, T b) {
44          return __gcd(a, b); // Change the custom operator
45      }
46
47      T qry(int v, int a, int b, int x1, int x2, int y1, int y2, int k) {

```

```

48     if (x1 <= a && b <= x2){
49         return comb(tree[v][k][y1], tree[v][k][y2]);
50     }
51     if (x1 >= b || a >= x2) return 0;
52     int mid = (a + b) / 2;
53     return comb(qry(2 * v, a, mid, x1, x2, y1, y2, k), qry(2 * v + 1, mid, b, x1, x2, y1, y2, k));
54 }
55
56 T qry(int lx, int ly, int rx, int ry) {
57     int k = 31 - __builtin_clz(ry - ly + 1);
58     return qry(1, 0, N, lx, rx + 1, ly, ry - (1 << k) + 1, k);
59 }

```

BIT

BIT.cpp

```

1  /*
2  -----BIT-----
3  Use: 1 - add value to element in the array
4       2 - sum of range in the array
5  Base: 0-index
6  Time complexity : add , sum >> O(log(n))
7
8  */
9  template<class T = int>
10 struct BIT{
11     vector<T> tree;
12     int n;
13     void init(int n){
14         this->n = n;
15         tree.assign(n , 0);
16     }
17     void add(int pos , T val){
18         for(pos++; pos <= n ; pos += (pos & (-pos)))
19             tree[pos - 1] += val;
20     }
21     T sum(int pos){
22         T ret = 0;
23         for(pos++; pos ; pos -= (pos & (-pos)))
24             ret += tree[pos - 1];
25         return ret;

```

```

26     }
27     T qry(int l, int r){ return sum(r) - sum(l - 1); }
28     T qry(int i){ return sum(i, i); }
29 };

```

BIT_LAZY_MODIFICATION.cpp

```

1  /*
2   BIT with LAZY MODIFICATION
3   use : So far we have presented BIT as a structure which is entirely allocated in memory
         during the initialization.
4   An advantage of this approach is that accessing tree[idx] requires a constant time.
         On the other hand, we might need to access only tree[idx] for a couple of different
         values of idx, e.g. log n different values, while we allocate much larger memory. This is
         especially aparent in the cases when we work with multidimensional BIT.
5  */
6  // 1-base
7  const int inf = 1e9 + 9;
8  map<int, long long> tree; // LAZY MODIFICATION
9  struct BIT {
10     void add(int x, int val) {
11         for(; x < inf; x += (x & -x)) tree[x] += val;
12     }
13     long long sum(int x){
14         long long res = 0;
15         for(; x > 0; x -= (x & -x)) res += tree[x];
16         return res;
17     }
18     long long sum(int l, int r){ return sum(r) - sum(l - 1); }
19 };

```

BIT_Ranges.cpp

```

1  const int N = 2e5 + 9;
2  class BITrange {
3  private:
4      long long m[N], c[N];
5      void add(int pos, long long mVal, long long cVal) {
6          for (++pos; pos <= N; pos += (pos & (-pos))) {
7              m[pos - 1] += mVal;
8              c[pos - 1] += cVal;

```



```

9      }
10     }
11     long long get(int pos) {
12         long long ret = 0;
13         int x = pos;
14         for (++pos; pos; pos -= (pos & (-pos))) {
15             ret += m[pos - 1] * x + c[pos - 1];
16         }
17         return ret;
18     }
19
20 public:
21     void init(int n) {
22         memset(m, 0, n * sizeof(m[0]));
23         memset(c, 0, n * sizeof(m[0]));
24     }
25     void addRange(int st, int en, long long val) {
26         if(st > en) return;
27         add(st, val, -val * (st - 1));
28         add(en + 1, -val, val * en);
29     }
30     void addIndex(int i, ll val){ addRange(i,i,val); }
31     ll qry(int l, int r){ return get(r) - get(l - 1); }
32     ll qry(int i){ return get(i) - get(i - 1); }
33 };

```

BIT_with_coordinate_compression.cpp

```

1  template<class T = int>
2  struct BIT{
3      vector<T> tree , v;
4      int n;
5      void init(vector<T> vv){
6          this->v = vv;
7          sort(all(v));
8          v.erase(unique(v.begin() , v.end() ) , v.end());
9          this->n = v.size();
10         tree.assign(n , 0);
11     }
12     void add(T idx , T val){
13         int pos = lower_bound(v.begin(), v.end() , idx) - v.begin();
14         for(pos++; pos <= n ; pos += (pos & (-pos)))

```

```

15     tree[pos - 1] += val;
16 }
17 T sum(int pos){
18     pos = lower_bound(v.begin(), v.end(), pos) - v.begin();
19     T ret = 0;
20     for(pos++; pos ; pos -= (pos & (-pos)))
21         ret += tree[pos - 1];
22     return ret;
23 }
24 T qry(int l, int r){ return sum(r) - sum(l - 1); }
25 T get_idx(int i){ return qry(i, i); }
26 };

```

Multiset_fenwick.cpp

```

1  const int N = 1e5 + 9;
2  struct BIT{
3      vector<int> tree;
4      int n;
5      void init(int n){
6          this->n = n;
7          tree.assign(n, 0);
8      }
9      void add(int pos, int val){
10         for(pos++; pos <= n; pos += (pos & (-pos)))
11             tree[pos - 1] += val;
12     }
13     int sum(int pos){
14         int ret = 0;
15         for(pos++; pos ; pos -= (pos & (-pos)))
16             ret += tree[pos - 1];
17         return ret;
18     }
19     int sum(int l, int r){ return sum(r) - sum(l - 1); }
20     int getidx(int i){ return sum(i, i); }
21 };
22 struct MultiSet{
23     BIT B;
24     MultiSet(){ B.init(N); }
25     void insert(int val){ B.add(val, 1); }
26     void erase(int val){ B.add(val, -1); }
27     int count(int val){ return B.sum(val - 1, val); }

```

```

28     int size(){ return B.sum(N-1); }
29     int order_of_key(int key){
30         int l = 0 , r = N - 1, mid;
31         while(l < r){
32             mid = l + (r - l) / 2;
33             if(B.sum(mid) > key) r = mid;
34             else l = mid + 1;
35         }
36         return l;
37     }
38 }s;

```

mex_using_BIT.cpp

```

1 // It 's faster than set
2 struct BIT {
3     vector<int> tree;
4     int n;
5     void init(int n) {
6         this->n = 1 << ( __lg(n) + !(n & (n - 1)));
7         tree.assign(this->n + 1, 0);
8     }
9     void insert(int x, int v = 1) {
10         for (++x; x <= n; x += (x & (-x))) {
11             tree[x - 1] += v;
12         }
13     }
14     void erase(int x, int v = -1) {
15         for (++x; x <= n; x += (x & (-x))) {
16             tree[x - 1] += v;
17         }
18     }
19     int search(int v = 0) { // O(log(n))
20         int p = 0, idx = 0;
21         for (int sz = n >> 1; sz; sz >>= 1) {
22             if (tree[p + sz - 1] <= v) {
23                 p += sz;
24             }
25         }
26         return p;
27     }
28 }

```

```

29
30 // More function can be useful
31 struct Mex{
32     BIT missing;
33     vector<int> frq;
34     int n;
35     void init(int n){
36         missing.init(n + 2);
37         frq.resize(n + 2);
38         this->n = n + 2;
39     }
40     int get(int i){ return missing.search(); } // O(log(n))
41     void add(int v){ // O(log(n))
42         if(v > n) return;
43         if(++frq[v] == 1) missing.erase(v);
44     }
45     void rm(int v){ // O(log(n))
46         if(v > n) return;
47         if(--frq[v] == 0) missing.insert(v);
48     }
49 };

```

DSU

DSU.cpp

```

1  struct DSU{
2      vector<int> par , size;
3      DSU(int n){
4          par.resize(n); size.resize(n , 1);
5          for(int i = 0; i < n; i++) par[i] = i;
6      }
7      int get(int a){return par[a] = (par[a] == a) ? a : get(par[a]);}
8      void Union(int a , int b){
9          a = get(a); b = get(b);
10         if(a == b) return;
11         if(size[a] > size[b]) swap(a , b);
12         size[b] += size[a];
13         par[a] = b;
14     }
15     int same_Group(int a , int b){return get(a) == get(b);}
16 };

```

DynamicConnectivity_offline.cpp

```

1  int ans;
2
3  struct Query {
4      int u, v; // Vertices to connect
5      int l, r; // Time interval [l,r] when edge exists (in case large time use compression)
6  };
7
8  struct Elem {
9      int u; // Parent vertex
10     int v; // Child vertex being attached
11     int szU; // Original size of set u
12     int cnt; // Previous count of disjoint sets
13 };
14
15 struct DSURollback {
16     int cnt;
17     int currentTime = 0; // initialized
18     stack<Elem> st;
19     vector<int> sz, par;
20     vector<vector<pair<int, int>>> g;
21
22     DSURollback(int n) : cnt(n), currentTime(0) { // initialized in constructor
23         int seg = 1;
24         while (seg < n) seg *= 2;
25         g.resize(2 * seg + 5);
26         par.resize(n + 1);
27         sz.resize(n + 1, 1);
28         iota(par.begin(), par.end(), 0);
29     }
30
31     int findSet(int u) { return par[u] == u ? u : findSet(par[u]); }
32
33     void update(int u, int v) {
34         st.push({u, v, sz[u], cnt});
35         cnt--, par[v] = u;
36         sz[u] += sz[v];
37         ans = max(ans, currentTime); // using tracked time
38     }
39
40     void unionSet(int u, int v) {
41         u = findSet(u), v = findSet(v);

```

```

42     if (u != v) {
43         if (sz[u] < sz[v]) swap(u, v);
44         update(u, v);
45     }
46 }
47
48 void rollback(int x) {
49     while (st.size() > x) {
50         auto e = st.top(); st.pop();
51         cnt = e.cnt;
52         sz[e.u] = e.szU;
53         par[e.v] = e.v;
54     }
55 }
56
57 void traverse(int x, int lX, int rX, const int &l, const int &r, const int &u, const int &v) {
58     if (rX < l || lX > r) return;
59     if (lX >= l && rX <= r) { g[x].emplace_back(u, v); return; }
60     int m = (lX + rX) / 2;
61     traverse(x * 2, lX, m, l, r, u, v);
62     traverse(x * 2 + 1, m + 1, rX, l, r, u, v);
63 }
64
65 void solve(int x, int l, int r) {
66     int cur = st.size();
67     currentTime = l; // updating current time
68     for (auto &i : g[x]) unionSet(i.first, i.second);
69     if (l == r) { rollback(cur); return; }
70     int m = (l + r) / 2;
71     solve(x * 2, l, m);
72     solve(x * 2 + 1, m + 1, r);
73     rollback(cur);
74 }
75
76 void build(vector<Query> &queries) {
77     for (auto &q : queries) traverse(1, 0, cnt - 1, q.l, q.r, q.u, q.v);
78     solve(1, 0, cnt - 1);
79 }
80 };

```

RollbackUF.cpp

```

1  struct RollbackUF {
2      vector<int> e;          // Parent/size array (-ve size, +ve parent)
3      vector<pair<int,int>> st; // History for rollback (index, prev value)
4
5      RollbackUF(int n) : e(n, -1) {}
6
7      int size(int x) {      // Returns size of set containing x
8          return -e[find(x)];
9      }
10
11     int find(int x) {      // Gets root/representative of x's set
12         return e[x] < 0 ? x : find(e[x]);
13     }
14
15     int time() {          // Current operation count
16         return st.size();
17     }
18
19     void rollback(int t) { // Undo operations after time t
20         for(int i = time(); i > t; i--) {
21             e[st[i-1].first] = st[i-1].second;
22         }
23         st.resize(t);
24     }
25
26     bool join(int a, int b) { // Union sets of a and b, return true if joined
27         a = find(a);
28         b = find(b);
29         if(a == b) return false;
30
31         if(e[a] > e[b]) swap(a, b); // a will be root
32
33         st.push_back({a, e[a]}); // Save state for rollback
34         st.push_back({b, e[b]});
35         e[a] += e[b];           // Update size
36         e[b] = a;              // Set parent
37         return true;
38     }
39 };

```

DSU on tree

sackDFS_keepBigChild.cpp

```

1  /*
2  DSU on tree by keeping largest child
3  Let:
4      - st[u] dfs starting time of vertex u,
5      - ft[u] be it's finishing time and
6      - ver[time] is the vertex which it's starting time is equal to time.
7      - sz[u] is the size of subtree of node
8
9  Pros: You can keep segment tree for each node with using extra memory
10 */
11
12 const int N = 1e5; // TODO: change it to maximum possible N
13 int dfs_time = 0;
14 int st[N] , ft[N] , big[N] , ver[N] , sz[N];
15 vector<int> adj[N];
16 void preDFS(int u , int p){
17     st[u] = dfs_time++;
18     ver[ st[u] ] = u;
19     sz[u] = 1, big[u] = -1;
20     for(auto v : adj[u]) if(v != p){
21         preDFS(v,u);
22         sz[u] += sz[v];
23         if(big[u] == -1 || sz[v] > sz[ big[u] ]) big[u] = v;
24     }
25     ft[u] = dfs_time;
26 }
27 void sackDFS(int u, int p, bool keep){
28     int bigChild = big[u];
29     if(bigChild != -1) sackDFS(bigChild, u, 1); // bigChild marked as big and not cleared
30     // from cnt
31     for(auto v : adj[u]){
32         if(v != p && v != bigChild){
33             for(int p = st[v]; p < ft[v]; p++){
34                 // TODO: Add your information about ver[p]
35             }
36         }
37     }
38
39     // TODO: Add your information about u
40
41
42

```



```

43 // TODO: All information about the subtree of u is kept, and you can now query it.
44
45
46 if(keep == 0){
47     for(int p = st[u]; p < ft[u]; p++){
48         // TODO: Remove the added information about ver[p]
49
50     }
51 }
52
53 }
54 // Calling -> sackDFS(root, -1 , 0)

```

small_to_large_trick.cpp

```

1  /*
2   Classic problems: https://cses.fi/problemset/task/1139
3   You are given a rooted tree consisting of n nodes.
4   The nodes are numbered 1,2,...,n, and node 1 is the root. Each node has a color.
5   Your task is to determine for each node the number of distinct colors in the subtree of
   the node.
6   Time complexity:  $O(n \log n)$ 
7   Space complexity:  $O(n \log n)$ 
8  */
9  #include <bits/stdc++.h>
10 using namespace std;
11 typedef long long ll;
12 #define rep(i , st , ed) for(int i = st; i < ed; i++)
13 #define f first
14 #define s second
15 #define all(v) v.begin() , v.end()
16 #ifndef ONLINE_JUDGE
17 #define debug(x) cerr << #x << ": " << x << '\n';
18 #else
19 #define debug(x)
20 #endif
21 const int N = 2e5 + 9;
22 vector<int> adj[N];
23 map<int,int> mp[N];
24 int ans[N], color[N];
25 int dfs(int u, int par){
26     int p = u;

```

```

27     mp[p][color[u]]++;
28     for(auto &v : adj[u]) if(v != par){
29         int x = dfs(v , u);
30         if(mp[x].size() > mp[p].size()) swap(x,p);
31         for(auto &[c,frq]: mp[x]) mp[p][c] += frq;
32     }
33     ans[u] = (int) mp[p].size();
34     return p;
35 }
36 int main(){
37     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
38     #ifndef ONLINE_JUDGE
39     freopen("in.txt", "r", stdin);
40     freopen("out.txt", "w", stdout);
41     freopen("error.txt", "w", stderr);
42     #endif
43     int n; cin >> n;
44     for(int i = 0; i < n; ++i) cin >> color[i];
45     for(int i = 0; i < n - 1; ++i){
46         int u , v; cin >> u >> v;
47         --u; --v;
48         adj[u].emplace_back(v);
49         adj[v].emplace_back(u);
50     }
51     dfs(0,0);
52     for(int i = 0; i < n; ++i) cout << ans[i] << " ";
53 }

```

Divide and conquer

offline_static_rmq_dc.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define f first
5  #define s second
6  int main(){
7     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
8     int n , q; cin >> n >> q;
9     int a[n];
10    for(int i = 0; i < n; ++i) cin >> a[i];

```

```

11  int lt[n] , rg[n] , ans[q];
12  vector<array<int,3>> Queries;
13  for(int i = 0; i < q; ++i){
14      int l , r; cin >> l >> r;
15      --l; --r;
16      Queries.push_back({l , r , i});
17  }
18  auto comb = [&](int x , int y){ return min(x , y); };
19  function<void(int,int,vector<array<int,3>>)> solve = [&]( int l , int r ,
vector<array<int,3>> Queries){
20      if(Queries.empty()) return; // just optimization
21      if(l == r){
22          for(auto &[L , R , ind] : Queries) ans[ind] = a[l];
23          return;
24      }
25      int m = (l + r) / 2;
26      lt[m] = a[m]; rg[m + 1] = a[m + 1];
27      for(int i = m - 1; i >= l; --i) lt[i] = comb(a[i] , lt[i + 1]);
28      for(int i = m + 2; i <= r; ++i) rg[i] = comb(a[i] , rg[i - 1]);
29      vector<array<int,3>> tmp[2];
30
31      for(auto &[L , R , ind] : Queries){
32          if(L <= m && m < R){
33              ans[ind] = comb(lt[L] , rg[R]);
34          }else{
35              tmp[L > m].push_back({L , R , ind});
36          }
37      }
38      solve(l , m , tmp[0]); solve(m + 1 , r , tmp[1]);
39  };
40  solve(0 , n - 1 , Queries);
41  for(int i = 0; i < q; ++i){
42      cout << ans[i] << '\n';
43  }

```

onlineRangeQueryD&C_benq.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first

```

```

6  #define s second
7  #define all(v) v.begin() , v.end()
8  #ifndef ONLINE_JUDGE
9  #define debug(x) cerr << #x << ": " << x << '\n';
10 #else
11 #define debug(x)
12 #endif
13 template<class T, int SZ> struct RangeQuery {
14     int n;
15     T stor[SZ][32-__builtin_clz(SZ)], id = 1;
16     vector<T> a;
17     T comb (T a, T b) {
18         return mul(a,b); // associative operation
19     }
20     void fill(int l, int r, int ind) {
21         if (ind < 0) return;
22         int m = (l+r)/2;
23         T prod = id; for(int i = l; i < m; ++i) stor[i][ind] = prod = comb(a[i],prod);
24         prod = id; for(int i = m; i < r; ++i) stor[i][ind] = prod = comb(prod,a[i]);
25         fill(l,m,ind-1); fill(m,r,ind-1);
26     }
27     void init() {
28         n = 1; while ((1<<n) < int(a.size())) ++n;
29         a.resize(1 << n);
30         fill(0,(1<<n),n-1);
31     }
32     T query(int l, int r) {
33         if (l == r) return a[l];
34         int t = 31-__builtin_clz(r^l);
35         return comb(stor[l][t],stor[r][t]);
36     }
37 };
38
39 int main(){
40     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
41     #ifndef ONLINE_JUDGE
42     freopen("in.txt", "r", stdin);
43     freopen("out.txt", "w", stdout);
44     freopen("error.txt", "w", stderr);
45     #endif
46
47 }

```

online_static_rmq_dc.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int N = 2e5 + 9 , lg = 18;
5  int n , q , a[N];
6  int prec[lg][N] , msk[N];
7  int comb(int x , int y){ return min(x , y); }; // associative operator (eg. sum, prod, xor,or,
    and)
8  void dc(int l, int r, int lvl){
9      if(l == r) return;
10     int m = (l + r) / 2;
11     prec[lvl][m] = a[m];
12     prec[lvl][m + 1] = a[m + 1];
13     for(int i = m - 1; i >= l; --i) prec[lvl][i] = comb(a[i] , prec[lvl][i + 1]);
14     for(int i = m + 2; i <= r; ++i) prec[lvl][i] = comb(a[i] , prec[lvl][i - 1]);
15     for(int i = m + 1; i <= r; ++i) msk[i] |= (1 << lvl);
16     dc(l , m , lvl + 1); dc(m + 1 , r , lvl + 1);
17 };
18 int qry(int l , int r){
19     int k = __builtin_ctz(msk[l] ^ msk[r]);
20     return comb(prec[k][l] , prec[k][r]);
21 };
22 int main(){
23     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
24     #ifndef ONLINE_JUDGE
25     freopen("in.txt" , "r" , stdin);
26     freopen("out.txt" , "w" , stdout);
27     freopen("error.txt" , "w" , stderr);
28     #endif
29     cin >> n >> q;
30     for(int i = 0; i < n; ++i) cin >> a[i];
31     dc(0 , n , 0);
32     for(int i = 0; i < q; ++i){
33         int l , r; cin >> l >> r;
34         --l; --r;
35         if(l == r) cout << a[l] << '\n';
36         else cout << qry(l , r) << '\n';
37     }
38 }

```

Kd-tree

kd-tree 3d.cpp

```

1  struct Point {
2      long double x, y, z;
3      Point(long double x, long double y, long double z):x(x), y(y), z(z){}
4      bool operator<(const Point& p) const {
5          return x < p.x || (x == p.x && (y < p.y || (y == p.y && z < p.z)));
6      }
7  };
8
9  class KDTree {
10 private:
11     struct Node {
12         Point point;
13         Node* left;
14         Node* right;
15         Node(Point p) : point(p), left(nullptr), right(nullptr) {}
16     };
17     Node* root;
18
19     Node* build(vector<Point>& points, int depth) {
20         if (points.empty()) return nullptr;
21         int axis = depth % 3;
22         sort(points.begin(), points.end(), [axis](Point a, Point b) {
23             if (axis == 0) return a.x < b.x;
24             if (axis == 1) return a.y < b.y;
25             return a.z < b.z;
26         });
27         int mid = points.size() / 2;
28         Node* node = new Node(points[mid]);
29         vector<Point> leftPoints(points.begin(), points.begin() + mid);
30         vector<Point> rightPoints(points.begin() + mid + 1, points.end());
31         node->left = build(leftPoints, depth + 1);
32         node->right = build(rightPoints, depth + 1);
33         return node;
34     }
35
36     long double dist(Point p1, Point p2) {
37         long double res = (p1.x - p2.x) * (p1.x - p2.x);
38         res += (p1.y - p2.y) * (p1.y - p2.y);
39         res += (p1.z - p2.z) * (p1.z - p2.z);

```

```

40     return sqrtl(res);
41 }
42
43 void nearest(Node* node, Point target, int depth, long double& bestDist) {
44     if (!node) return;
45     long double currentDist = dist(node->point, target);
46     if (currentDist < bestDist && currentDist > 0) bestDist = currentDist; // Added
currentDist > 0 check
47
48     int axis = depth % 3;
49     Node* next = (axis == 0 ? target.x < node->point.x :
50         (axis == 1 ? target.y < node->point.y : target.z < node->point.z)) ? node->left :
node->right;
51     Node* other = next == node->left ? node->right : node->left;
52
53     nearest(next, target, depth + 1, bestDist);
54
55     long double axisDist = (axis == 0 ? abs(target.x - node->point.x) :
56         (axis == 1 ? abs(target.y - node->point.y) : abs(target.z - node->point.z)));
57     if (axisDist < bestDist) {
58         nearest(other, target, depth + 1, bestDist);
59     }
60 }
61
62 void findMinDistanceHelper(Node* node, long double& minDist) {
63     if (!node) return;
64
65     // Find nearest point to current node's point
66     long double currentMin = numeric_limits<long double>::max();
67     nearest(root, node->point, 0, currentMin);
68     minDist = min(minDist, currentMin);
69
70     // Recurse on children
71     findMinDistanceHelper(node->left, minDist);
72     findMinDistanceHelper(node->right, minDist);
73 }
74
75 public:
76     KDTree(vector<Point>& points) {
77         root = build(points, 0);
78     }
79
80     long double nearest(Point target) {
81         long double bestDist = numeric_limits<long double>::max();

```

```

82     nearest(root, target, 0, bestDist);
83     return bestDist;
84 }
85
86 long double findMinDistance() {
87     if (!root || (!root->left && !root->right)) return numeric_limits<long double>::max();
88     long double minDist = numeric_limits<long double>::max();
89     findMinDistanceHelper(root, minDist);
90     return minDist;
91 }

```

kd-tree kactl.cpp

```

1  /*
2  Quick Tips for KD-tree in Contest:
3  1. Randomize input points to avoid O(n) search on sorted data
4  2. If TLE on findMinDistance(), consider using simpler n^2 brute force
5  3. For nearest neighbor, if getting WA/TLE, check if input points are unique
6  4. Watch out for integer overflow in distance calculations
7  5. Consider using Manhattan distance if precision is an issue
8  */
9
10 template<class T>
11 struct Point {
12     typedef Point P;
13     T x, y;
14     explicit Point(T x=0, T y=0) : x(x), y(y) {}
15     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
16     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
17     P operator+(P p) const { return P(x+p.x, y+p.y); }
18     P operator-(P p) const { return P(x-p.x, y-p.y); }
19     T dist2() const { return x*x + y*y; } // Replace if u use manhattan distance {return abs(x)
+ abs(y); }
20     friend ostream& operator<<(ostream& os, P p) {
21         return os << "(" << p.x << "," << p.y << ")"; }
22 };
23
24 typedef long long ll;
25 typedef Point<ll> P;
26 const ll INF = numeric_limits<ll>::max();
27
28 bool on_x(const P& a, const P& b) { return a.x < b.x; }

```



```

29  bool on_y(const P& a, const P& b) { return a.y < b.y; }
30
31  struct Node {
32      P pt;
33      ll x0 = INF, x1 = -INF, y0 = INF, y1 = -INF;
34      Node *first = 0, *second = 0;
35
36      ll distance(const P& p) {
37          ll x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
38          ll y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
39          return (P(x,y) - p).dist2();
40      }
41
42      Node(vector<P>&& vp) : pt(vp[0]) {
43          for (P p : vp) {
44              x0 = min(x0, p.x); x1 = max(x1, p.x);
45              y0 = min(y0, p.y); y1 = max(y1, p.y);
46          }
47          if (vp.size() > 1) {
48              sort(vp.begin(), vp.end(), x1 - x0 >= y1 - y0 ? on_x : on_y);
49              int half = vp.size()/2;
50              first = new Node({vp.begin(), vp.begin() + half});
51              second = new Node({vp.begin() + half, vp.end()});
52          }
53      }
54  };
55
56  class KDTree {
57  private:
58      Node* root;
59
60      // O(log n) average, O(n) worst: Helper for nearest neighbor search
61      pair<ll, P> search(Node *node, const P& p) {
62          if (!node->first) {
63              return make_pair((p - node->pt).dist2(), node->pt);
64          }
65          Node *f = node->first, *s = node->second;
66          ll bfirst = f->distance(p), bsec = s->distance(p);
67          if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
68          auto best = search(f, p);
69          if (bsec < best.first)
70              best = min(best, search(s, p));
71          return best;
72      }

```

```

73
74 void findMinDistanceHelper(Node* node, long double& minDist) {
75     if (!node || (!node->first && !node->second)) return;
76
77     if (node->first && node->second) {
78         minDist = min(minDist, (long double)(node->first->pt - node->second->pt).dist2());
79     }
80
81     findMinDistanceHelper(node->first, minDist);
82     findMinDistanceHelper(node->second, minDist);
83 }
84
85 public:
86 KDTree(const vector<P>& vp) : root(new Node({vp.begin(), vp.end()})) {}
87
88 // O(log n) average: Finds nearest neighbor to query point
89 pair<ll, P> nearest(const P& p) {
90     return search(root, p);
91 }
92
93 long double findMinDistance() {
94     if (!root) return numeric_limits<long double>::max();
95     long double minDist = numeric_limits<long double>::max();
96     findMinDistanceHelper(root, minDist);
97     return sqrt(minDist);
98 }

```

LCA

LCA.cpp

```

1  const int N = 1e5 + 10, LOG = 20;
2  vector<int> adj[N];
3  struct lca{
4      int n;
5      vector<int> depth, parent[LOG];
6      void init(int n, int root = 0){
7          this->n = n;
8          depth.resize(n);
9          for(int i = 0; i < LOG; ++i) parent[i].resize(n);
10         dfs(root, root, 0);
11     }

```

```

12 void dfs(int u, int p, int d) {
13     depth[u] = d;
14     parent[0][u] = p;
15     for (int i = 1; i < LOG; ++i) {
16         parent[i][u] = parent[i - 1][parent[i - 1][u]];
17     }
18     for (int v : adj[u]) {
19         if (v == p) continue;
20         dfs(v, u, d + 1);
21     }
22 }
23 int kth_ancestor(int u, int k) {
24     for (int i = 0; i < LOG; ++i) {
25         if ((1 << i) & k) {
26             u = parent[i][u];
27         }
28     }
29     return u;
30 }
31 int LCA(int u, int v) {
32     if (depth[u] > depth[v]) swap(u, v);
33     int k = depth[v] - depth[u];
34     v = kth_ancestor(v, depth[v] - depth[u]);
35     if (u == v) return u;
36
37     for (int i = LOG - 1; ~i; --i) {
38         if (parent[i][u] != parent[i][v]) {
39             u = parent[i][u];
40             v = parent[i][v];
41         }
42     }
43     assert(parent[0][u] == parent[0][v]);
44     return parent[0][u];
45 }
46 };

```

LCA_in_DSU.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i, st, ed) for(int i = st; i < ed; i++)

```

```

5  #define f first
6  #define s second
7  const int LOG = 20;
8  struct DSU
9  {
10     vector<int> par , size , W[LOG] , up[LOG] , dep;
11     ll MST = 0;
12     int n;
13     DSU(int n){
14         this->n = n;
15         par.resize(n); size.resize(n , 1); dep.resize(n);
16         rep(i , 0 , n) par[i] = i;
17         rep(i , 0 , LOG) up[i].resize(n);
18     }
19     pair<int,int> get(int a){ // {par , depth}
20         if(par[a] == a) return {a , 0};
21         auto p = get(par[a]); p.s++;
22         return p;
23     } // no path compression
24     void Union(int a , int b , int w){
25         a = get(a).f; b = get(b).f;
26         if(a == b) return; // same component
27         if(size[a] > size[b]) swap(a , b);
28         size[b] += size[a];
29         par[a] = b;
30         MST += w;
31     }
32     void Build(){
33         rep(i , 0 , n){
34             up[0][i] = par[i];
35             dep[i] = get(i).s;
36         }
37         rep(x , 1 , LOG) rep(u , 0 , n){
38             up[x][u] = up[x - 1][ up[x - 1][u] ];
39         }
40     }
41     ll lca(int u , int v , int w){
42         int mx = 0;
43         if(dep[u] < dep[v]) swap(u , v);
44         int k = dep[u] - dep[v];
45         for(int i = LOG - 1; i >= 0; i--) if((k >> i) & 1){
46             mx = max(mx , W[i][u]);
47             u = up[i][u];
48         }

```

```

49     if(u == v) return ;
50     for(int i = LOG - 1; i >= 0; --i) if(up[i][u] != up[i][v]){
51         mx = max({mx , W[i][u] , W[i][v]});
52         u = up[i][u];
53         v = up[i][v];
54     }
55     assert(up[0][u] == up[0][v]);
56     return ;
57 }
58 };
59 struct edge{ int u , v , w; };
60 int main(){
61     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
62     #ifndef ONLINE_JUDGE
63     freopen("in.txt", "r", stdin);
64     freopen("out.txt", "w", stdout);
65     freopen("error.txt", "w", stderr);
66     #endif
67     int n , m; cin >> n >> m;
68     vector<edge> e(m); // {w , u , v}
69     rep(i , 0 , m){
70         cin >> e[i].u >> e[i].v >> e[i].w;
71         --e[i].u; --e[i].v;
72     }
73     int idx[m];
74     iota(idx , idx + m , 0);
75     sort(idx , idx + m , [&](int x , int y){
76         return e[x].w < e[y].w;
77     });
78     // Building MST
79     DSU d(n);
80     for(int i = 0; i < m; ++i){
81         int l = idx[i];
82         d.Union(e[l].u , e[l].v , e[l].w);
83     }
84     d.Build();
85

```

lca_O(1).cpp

```

1 // Problem Link: https://cses.fi/problemset/task/1688
2 #include <bits/stdc++.h>

```

```

3  using namespace std;
4  typedef long long ll;
5  #define rep(i , st , ed) for(int i = st; i < ed; i++)
6  #define f first
7  #define s second
8  #define all(v) v.begin() , v.end()
9  #ifndef ONLINE_JUDGE
10 #define debug(x) cerr << #x << ": " << x << '\n';
11 #else
12 #define debug(x)
13 #endif
14 const int N = 2e5 + 9;
15 vector<int> adj[N];
16 vector<int> euler_tour; // in time
17 int in[4 * N] , Timer;
18 void dfs(int u = 0 , int p = 0){
19     in[u] = Timer++;
20     euler_tour.emplace_back(u);
21     for(auto &v : adj[u]) if(v != p){
22         dfs(v , u);
23         euler_tour.emplace_back(u);
24         Timer++;
25     }
26 }
27 struct SparseTable {
28     vector<int> log;
29     vector<vector<pair<ll, int>>> spt;
30
31     void init(int n) {
32         log.assign(n + 1, 0);
33         for (int i = 2; i <= n; i++) {
34             log[i] = 1 + log[i / 2];
35         }
36         int k = log[n] + 1;
37         spt = vector<vector<pair<ll, int>>>(k, vector<pair<ll, int>>(n));
38         for (int i = 0; i < n; i++) {
39             spt[0][i] = { in[euler_tour[i]] , euler_tour[i] };
40         }
41         for (int j = 1; 1 <= j <= n; j++) {
42             for (int i = 0; i + (1 <= j) - 1 < n; i++) {
43                 spt[j][i] = merge(spt[j - 1][i], spt[j - 1][i + (1 <= (j - 1))]);
44             }
45         }
46     }

```

```

47     pair<ll, int> merge(pair<ll, int> &x, pair<ll, int> &y) {
48         if(x.f < y.f) return x;
49         return y;
50     }
51     pair<ll, int> query(int i, int j) {
52         int len = j - i + 1;
53         int k = log[len];
54         return merge(spt[k][i], spt[k][j - (1 << k) + 1]);
55     }
56     int lca(int i, int j){
57         if(in[i] > in[j]) swap(i, j);
58         return query(in[i], in[j]).s;
59     }
60 };
61 int main(){
62     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
63     #ifndef ONLINE_JUDGE
64     freopen("in.txt", "r", stdin);
65     freopen("out.txt", "w", stdout);
66     freopen("error.txt", "w", stderr);
67     #endif
68     int n, q; cin >> n >> q;
69     for(int i = 1; i < n; ++i){
70         int p; cin >> p;
71         --p;
72         adj[p].emplace_back(i);
73     }
74     dfs();
75     SparseTable spt;
76     spt.init(Timer);
77     while(q--){
78         int u, v; cin >> u >> v;
79         --u; --v;
80         cout << spt.lca(u, v) + 1 << '\n';
81     }

```

Minimum spanning tree

MST_Kruskal.cpp

```

1     struct DSU
2     {

```

```

3   vector<int> par , size;
4   DSU(int n){
5       par.resize(n); size.resize(n , 1);
6       rep(i , 0 , n) par[i] = i;
7   }
8   int get(int a){return par[a] = (par[a] == a) ? a : get(par[a]);}
9   void Union(int a, int b){
10      a = get(a); b = get(b);
11      if(a == b) return; // In the same Group
12      if(size[a] > size[b]) swap(a , b);
13      size[b] += size[a];
14      par[a] = b;
15  }
16  };
17  void sol(){
18      int n , m; cin >> n >> m;
19      vector<array<int,3>> e(m); // {w , u , v}
20      rep(i , 0 , m) cin >> e[i][1] >> e[i][2] >> e[i][0];
21      sort(e.begin() , e.end());
22      ll ans = 0;
23      DSU d(n);
24      for(auto &[w , u , v] : e){
25          --u; --v;
26          if(d.get(u) == d.get(v)) continue;
27          d.Union(u , v);
28          ans += w;
29      }
30      cout << ans;
31  }

```

MST_Prim's.cpp

```

1   int n , m; cin >> n >> m;
2   vector<pair<int,int>> adj[n]; // {v , w}
3   for(int i = 0; i < m; ++i){
4       int u , v , w; cin >> u >> v >> w;
5       --u; --v;
6       adj[u].emplace_back(v , w);
7       adj[v].emplace_back(u , w);
8   }
9   priority_queue<pair<int,int>> q; // {-w , u}
10  vector<int> vis(n);

```



```

11  q.push({0 , 0}); // start from any node
12  ll cost = 0;
13  while(q.size()){
14      auto [d , u] = q.top(); q.pop();
15      if(vis[u]) continue;
16      d *= -1; vis[u] = 1;
17      cost += d;
18      for(auto &[v , w] : adj[u]) if(vis[v] == 0){
19          q.push({-w , v});
20      }
21  }
22  cout << cost;

```

Montomic stack and Queue

NxtorPrv_MinorMax.cpp

```

1  vector<int> getNextMin(vector<int> &arr) {
2      stack<int> st;
3      vector<int> res(arr.size(), arr.size());
4      for (int i = 0; i < arr.size(); i++) {
5          while (!st.empty() && arr[st.top()] > arr[i]) {
6              res[st.top()] = i;
7              st.pop();
8          }
9          st.push(i);
10     }
11     return res;
12 }
13
14 vector<int> getPrevMin(vector<int> &arr) {
15     stack<int> st;
16     vector<int> res(arr.size(), -1);
17     for (int i = arr.size() - 1; i >= 0; i--) {
18         while (!st.empty() && arr[st.top()] >= arr[i]) {
19             res[st.top()] = i;
20             st.pop();
21         }
22         st.push(i);
23     }
24     return res;
25 }

```

```

26
27 vector<int> getNextMax(vector<int> &arr) {
28     stack<int> st;
29     vector<int> res(arr.size(), arr.size());
30     for (int i = 0; i < arr.size(); i++) {
31         while (!st.empty() && arr[st.top()] < arr[i]) {
32             res[st.top()] = i;
33             st.pop();
34         }
35         st.push(i);
36     }
37     return res;
38 }
39
40 vector<int> getPrevMax(vector<int> &arr) {
41     stack<int> st;
42     vector<int> res(arr.size(), -1);
43     for (int i = arr.size() - 1; i >= 0; i--) {
44         while (!st.empty() && arr[st.top()] <= arr[i]) {
45             res[st.top()] = i;
46             st.pop();
47         }
48         st.push(i);
49     }
50     return res;
51 }

```

monotonicQueue.cpp

```

1  /**
2   * Monotonic queue to keep track of the minimum and the maximum
3   * elements so far in the queue in amortized time of O(1).
4   */
5  template<class T>
6  class monotonic_queue {
7      queue<T> qu;
8      deque<T> mx, mn;
9  public:
10     void push(T v) {
11         qu.push(v);
12         while (mx.size() && mx.back() < v) mx.pop_back();
13         mx.push_back(v);

```

```

14     while (mn.size() && mn.back() > v) mn.pop_back();
15     mn.push_back(v);
16 }
17 void pop() {
18     if (mx.front() == qu.front()) mx.pop_front();
19     if (mn.front() == qu.front()) mn.pop_front();
20     qu.pop();
21 }
22 T front() const {
23     return qu.front();
24 }
25 T max() const {
26     return mx.front();
27 }
28 T min() const {
29     return mn.front();
30 }
31 size_t size() const {
32     return qu.size();
33 }
34 };

```

Offline Range Queries

Arithmetic_progression_prefix_sum.cpp

```

1  const int N = 1e6 + 5;
2  ll d[N], P[N];
3  void inc(int l, int r, int k, int b){
4      // add for each x : [l, r], v_x += k * (x - l) + b
5      if(l > r) return;
6      d[l + 1] += k;
7      d[r + 1] -= k;
8
9      P[l] += b;
10     P[r + 1] -= b + 1LL * k * (r - l);
11 }
12 void build(){
13     for(int i = 1; i < N; ++i){
14         d[i] += d[i - 1];
15         P[i] += P[i - 1] + d[i];
16     }

```

```

17 }
18
19 \\ -----
20 const int N = 1e6 + 5;
21 ll d[N] , P[N];
22 void inc(int l , int r , int k , int b){
23     // add for each x : [l , r] , v_x += k * (x - l + 1) + b
24     if(l > r) return;
25     d[l] += k;
26     d[r + 1] -= k;
27
28     P[l] += b;
29     P[r + 1] -= b + 1LL * k * (r - l + 1);
30 }
31 void build(){
32     for(int i = 1; i < N; ++i){
33         d[i] += d[i - 1];
34         P[i] += P[i - 1] + d[i];
35     }
36 }

```

Ordered set

ordered_set.cpp

```

1 // only in less_equal lower_bound work such as upper_bound;
2 // by *find_by_order given index --> val ;
3 // by order_of_key given value --> index ;
4 // erase and insert in log(n) ;
5 // used norm policy data struc when no duplicate the same element;
6
7 #include <ext/pb_ds/assoc_container.hpp> // Common file
8 #include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
9 using namespace __gnu_pbds;
10 template<class T> using ordered_set = tree<T, null_type, less_equal<T> , rb_tree_tag ,
    tree_order_statistics_node_update> ;
11 struct ordered__set{
12
13     ordered_set< ll > se ;
14     void erase( ll val ){
15         if( se.size() == 0 || *se.find_by_order( se.size() - 1 ) < val || *se.lower_bound( val - 1 ) !=
            val ) return ;

```

```

16     se.erase( se.lower_bound( --val ) );
17 }
18 int lower_bound( ll val ){ // log --> return index ;
19     if( se.size() == 0 || *se.find_by_order( se.size() - 1 ) < val ) return -1;
20     return se.order_of_key( *se.lower_bound( --val ) );
21 }
22 int upper_bound( ll val ){ return lower_bound( val + 1ll ); }
23 void insert( ll val ){ se.insert( val ); }
24 ll operator[](int idx) { return *se.find_by_order( idx ); }
25 int size() { return se.size(); }
26 void clr( ) { se.clear(); }
27 };

```

Persistent Segment tree

PersistentSegmentTree_counter.cpp

```

1  const int N = 1e6+9;
2  int n , m , id;
3  struct Node{
4      int l , r , s;
5  }tree[20*N];
6  int newLeaf(int v){
7      tree[id] = Node{-1,-1,v};
8      return id++;
9  }
10 int newPar(int l , int r){
11     tree[id] = Node{l,r,0};
12     tree[id].s += tree[l].s;
13     tree[id].s += tree[r].s;
14     return id++;
15 }
16 int build(int l = 0 , int r = m){
17     if(r - l == 1) return newLeaf(0);
18     int mid = (l+r)/2;
19     return newPar( build(l,mid) , build(mid,r) );
20 }
21 int copy(int i , int v , int x , int l = 0 , int r = m){
22     if(r - l == 1) return newLeaf(tree[x].s + v);
23     int mid = (l+r)/2;
24     if(i < mid) return newPar( copy(i,v , tree[x].l , l , mid) , tree[x].r );
25     return newPar( tree[x].l , copy(i,v , tree[x].r , mid , r) );

```

```

26 }
27 int qry(int lx , int rx , int prv , int cur , int l = 0 , int r = m){
28     if(l >= lx && r <= rx) return tree[cur].s - tree[prv].s;
29     if(r <= lx || l >= rx) return 0;
30     int mid = (l+r)/2;
31     return qry(lx,rx, tree[prv].l , tree[cur].l ,l,mid) + qry(lx,rx, tree[prv].r ,
    tree[cur].r,mid,r);
32 }

```

PersistentSegmentTree_pointer.cpp

```

1  struct Node{
2      Node* l , *r;
3      int s;
4      Node(int s): s(s) , l(NULL) , r(NULL){};
5      Node(Node *l , Node *r): l(l) , r(r){
6          this->s = 0;
7          if(l != NULL) this->s += l->s;
8          if(r != NULL) this->s += r->s;
9      }
10 };
11 Node* newLeaf(int v){ return new Node{v}; }
12 Node* newPar(Node *l , Node *r){ return new Node(l,r); }
13 Node* build(int l = 0 , int r = m){
14     if(r - l == 1) return newLeaf(0);
15     int mid = (l+r)/2;
16     return newPar( build(l,mid) , build(mid,r) );
17 }
18 Node* copy(int i , int v , Node* x , int l = 0 , int r = m){
19     if(r-l== 1) return newLeaf(x->s + v);
20     int mid = (r+l) / 2;
21     if(i < mid) return newPar( copy(i,v, x->l , l , mid) , x->r );
22     return newPar( x->l , copy(i,v , x->r , mid , r) );
23 }
24 int qry(int lx , int rx , Node* prv , Node* cur , int l = 0 , int r = m){
25     if(l >= lx && r <= rx) return cur->s - prv->s;
26     if(r <= lx || l >= rx) return 0;
27     int mid = (l+r)/2;
28     return qry(lx,rx,prv->l , cur->l,l,mid) + qry(lx,rx,prv->r , cur->r,mid,r);
29 }

```

PresistentSegmentTree_serso.cpp

```

1  const int N = 1e6+9;
2  int n , m , id;
3  struct Node{
4      int l , r , s;
5  }tree[20*N];
6  int getL(int x){ return ~x ? tree[x].l : -1; }
7  int getR(int x){ return ~x ? tree[x].r : -1; }
8  int getS(int x){ return ~x ? tree[x].s : 0; }
9
10 int newLeaf(int v){
11     tree[id] = Node{-1,-1,v};
12     return id++;
13 }
14 int newPar(int l , int r){
15     tree[id] = Node{l,r, getS(l) + getS(r)};
16     return id++;
17 }
18 int copy(int i , int v , int x , int l = 0 , int r = m){
19     if(r - l == 1) return newLeaf( getS(x) + v);
20     int mid = (l+r)/2;
21     if(i < mid) return newPar( copy(i,v , getL(x) , l , mid) , getR(x));
22     return newPar( getL(x) , copy(i,v , getR(x) , mid , r) );
23 }
24 int qry(int lx , int rx , int prv , int cur , int l = 0 , int r = m){
25     if(l >= lx && r <= rx) return getS(cur) - getS(prv);
26     if(r <= lx || l >= rx) return 0;
27     int mid = (l+r)/2;
28     return qry(lx,rx, getL(prv) , getL(cur) ,l,mid) + qry(lx,rx, getR(prv) , getR(cur) , mid , r);
29 }

```

SQRT and Mo's

MO-Algorithm.cpp

```

1  const int N = 1e6 + 9, BLOCK_SIZE = 460;
2  void add(int idx){
3
4  }
5  void remove(int idx){
6

```

```

7   }
8   struct Query{
9       int l , r , id;
10      bool operator <(const Query &other) const{
11          int n1 = l / BLOCK_SIZE , n2 = other.l / BLOCK_SIZE;
12          if(n1 != n2) return n1 < n2;
13          return n1 % 2 ? r > other.r : r < other.r;
14      }
15  };
16  void Mo(vector<Query> &query){
17      sort(all(query));
18      int mo_l = 0 , mo_r = -1;
19      for(auto &q : query){
20          while(mo_l < q.l) remove(mo_l++);
21          while(mo_l > q.l) add(--mo_l);
22          while(mo_r > q.r) remove(mo_r--);
23          while(mo_r < q.r) add(++mo_r);
24          // calculate answer of Query
25      }
26  }

```

Segment tree

SparseSegmentTree.cpp

```

1   #include <bits/stdc++.h>
2   using namespace std;
3
4   class SparseSegtree {
5   private:
6       struct Node {
7           int freq = 0;
8           int lazy = 0;
9           int left = -1;
10          int right = -1;
11      };
12      vector<Node> tree;
13      const int n;
14      int timer = 0;
15
16      int comb(int a, int b) { return a + b; }
17

```



```

18 void apply(int cur, int len, int val) {
19     if (val == 1) {
20         tree[cur].lazy = val;
21         tree[cur].freq = len * val;
22     }
23 }
24
25 void push_down(int cur, int l, int r) {
26     if (tree[cur].left == -1) {
27         tree[cur].left = ++timer;
28         tree.push_back(Node());
29     }
30     if (tree[cur].right == -1) {
31         tree[cur].right = ++timer;
32         tree.push_back(Node());
33     }
34     int m = (l + r) / 2;
35     apply(tree[cur].left, m - l + 1, tree[cur].lazy);
36     apply(tree[cur].right, r - m, tree[cur].lazy);
37     tree[cur].lazy = 0;
38 }
39
40 void range_set(int cur, int l, int r, int ql, int qr, int val) {
41     if (qr < l || ql > r) { return; }
42     if (ql <= l && r <= qr) {
43         apply(cur, r - l + 1, val);
44     } else {
45         push_down(cur, l, r);
46         int m = (l + r) / 2;
47         range_set(tree[cur].left, l, m, ql, qr, val);
48         range_set(tree[cur].right, m + 1, r, ql, qr, val);
49         tree[cur].freq =
50             comb(tree[tree[cur].left].freq, tree[tree[cur].right].freq);
51     }
52 }
53
54 int range_sum(int cur, int l, int r, int ql, int qr) {
55     if (qr < l || ql > r) { return 0; }
56     if (ql <= l && r <= qr) { return tree[cur].freq; }
57     push_down(cur, l, r);
58     int m = (l + r) / 2;
59     return comb(range_sum(tree[cur].left, l, m, ql, qr),
60               range_sum(tree[cur].right, m + 1, r, ql, qr));
61 }

```

```

62
63 public:
64 SparseSegtree(int n, int q = 0) : n(n) {
65     if (q > 0) { tree.reserve(2 * q * __lg(n)); }
66     tree.push_back(Node());
67 }
68
69 void range_set(int ql, int qr, int val) { range_set(0, 0, n - 1, ql, qr, val); }
70
71 int range_sum(int ql, int qr) { return range_sum(0, 0, n - 1, ql, qr); }
72 };
73
74 int main() {
75     int query_num;
76     cin >> query_num;
77     const int RANGE_SIZE = 1e9;
78     SparseSegtree st(RANGE_SIZE + 1, query_num);
79
80     int c = 0;
81     for (int i = 0; i < query_num; i++) {
82         int type, x, y;
83         cin >> type >> x >> y;
84         if (type == 1) {
85             c = st.range_sum(x + c, y + c);
86             cout << c << '\n';
87         } else if (type == 2) {
88             st.range_set(x + c, y + c, 1);
89         }
90     }

```

segmentTree_iterative.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i, st, ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  #define all(v) v.begin(), v.end()
8  #ifndef ONLINE_JUDGE
9  #define debug(x) cerr << #x << ": " << x << '\n';
10 #else

```

```

11  #define debug(x)
12  #endif
13  const int N = 2e5 + 9;
14  int seg[4 * N];
15  void update(int k, int x) {
16      k += N;
17      seg[k] = x; // update node with value
18      k >>= 1;
19      while (k > 0) {
20          seg[k] = max(seg[2*k], seg[2*k+1]);
21          k >>= 1;
22      }
23  }
24  int merge(int a, int b){
25      // write code here
26      return 0; // return value
27  }
28  int query(int a, int b) {
29      a += N, b += N;
30      int s = 0;
31      while (a <= b) {
32          if (a & 1) {
33              s = merge(s, seg[a]);
34              a++;
35          }
36          if (~b & 1) {
37              s = merge(s, seg[b]);
38              b--;
39          }
40          a >>= 1, b >>= 1;
41      }
42      return s;
43  }
44  int main(){
45      ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
46      #ifndef ONLINE_JUDGE
47          freopen("in.txt", "r", stdin);
48          freopen("out.txt", "w", stdout);
49          freopen("error.txt", "w", stderr);
50      #endif
51
52  }

```

segtreeBeat.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int MAXN = 200001; // 1-based
6
7  int N;
8  ll A[MAXN];
9
10 struct Node {
11     ll sum; // Sum tag
12     ll max1; // Max value
13     ll max2; // Second Max value
14     ll maxc; // Max value count
15     ll min1; // Min value
16     ll min2; // Second Min value
17     ll minc; // Min value count
18     ll lazy; // Lazy tag
19 } T[MAXN * 4];
20
21 void merge(int t) {
22     // sum
23     T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;
24
25     // max
26     if (T[t << 1].max1 == T[t << 1 | 1].max1) {
27         T[t].max1 = T[t << 1].max1;
28         T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max2);
29         T[t].maxc = T[t << 1].maxc + T[t << 1 | 1].maxc;
30     } else {
31         if (T[t << 1].max1 > T[t << 1 | 1].max1) {
32             T[t].max1 = T[t << 1].max1;
33             T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max1);
34             T[t].maxc = T[t << 1].maxc;
35         } else {
36             T[t].max1 = T[t << 1 | 1].max1;
37             T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
38             T[t].maxc = T[t << 1 | 1].maxc;
39         }
40     }
41

```

```

42 // min
43 if (T[t << 1].min1 == T[t << 1 | 1].min1) {
44     T[t].min1 = T[t << 1].min1;
45     T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min2);
46     T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
47 } else {
48     if (T[t << 1].min1 < T[t << 1 | 1].min1) {
49         T[t].min1 = T[t << 1].min1;
50         T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
51         T[t].minc = T[t << 1].minc;
52     } else {
53         T[t].min1 = T[t << 1 | 1].min1;
54         T[t].min2 = min(T[t << 1].min1, T[t << 1 | 1].min2);
55         T[t].minc = T[t << 1 | 1].minc;
56     }
57 }
58 }
59
60 void push_add(int t, int tl, int tr, ll v) {
61     if (v == 0) { return; }
62     T[t].sum += (tr - tl + 1) * v;
63     T[t].max1 += v;
64     if (T[t].max2 != -llINF) { T[t].max2 += v; }
65     T[t].min1 += v;
66     if (T[t].min2 != llINF) { T[t].min2 += v; }
67     T[t].lazy += v;
68 }
69
70 // corresponds to a chmin update
71 void push_max(int t, ll v, bool l) {
72     if (v >= T[t].max1) { return; }
73     T[t].sum -= T[t].max1 * T[t].maxc;
74     T[t].max1 = v;
75     T[t].sum += T[t].max1 * T[t].maxc;
76     if (l) {
77         T[t].min1 = T[t].max1;
78     } else {
79         if (v <= T[t].min1) {
80             T[t].min1 = v;
81         } else if (v < T[t].min2) {
82             T[t].min2 = v;
83         }
84     }
85 }

```

```

86
87 // corresponds to a chmax update
88 void push_min(int t, ll v, bool l) {
89     if (v <= T[t].min1) { return; }
90     T[t].sum -= T[t].min1 * T[t].minc;
91     T[t].min1 = v;
92     T[t].sum += T[t].min1 * T[t].minc;
93     if (l) {
94         T[t].max1 = T[t].min1;
95     } else {
96         if (v >= T[t].max1) {
97             T[t].max1 = v;
98         } else if (v > T[t].max2) {
99             T[t].max2 = v;
100         }
101     }
102 }
103
104 void pushdown(int t, int tl, int tr) {
105     if (tl == tr) return;
106     // sum
107     int tm = (tl + tr) >> 1;
108     push_add(t << 1, tl, tm, T[t].lazy);
109     push_add(t << 1 | 1, tm + 1, tr, T[t].lazy);
110     T[t].lazy = 0;
111
112     // max
113     push_max(t << 1, T[t].max1, tl == tm);
114     push_max(t << 1 | 1, T[t].max1, tm + 1 == tr);
115
116     // min
117     push_min(t << 1, T[t].min1, tl == tm);
118     push_min(t << 1 | 1, T[t].min1, tm + 1 == tr);
119 }
120
121 void build(int t = 1, int tl = 0, int tr = N - 1) {
122     T[t].lazy = 0;
123     if (tl == tr) {
124         T[t].sum = T[t].max1 = T[t].min1 = A[tl];
125         T[t].maxc = T[t].minc = 1;
126         T[t].max2 = -llINF;
127         T[t].min2 = llINF;
128         return;

```

```

129     }
130
131     int tm = (tl + tr) >> 1;
132     build(t << 1, tl, tm);
133     build(t << 1 | 1, tm + 1, tr);
134     merge(t);
135 }
136
137 void update_add(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
138     if (r < tl || tr < l) { return; }
139     if (l <= tl && tr <= r) {
140         push_add(t, tl, tr, v);
141         return;
142     }
143     pushdown(t, tl, tr);
144
145     int tm = (tl + tr) >> 1;
146     update_add(l, r, v, t << 1, tl, tm);
147     update_add(l, r, v, t << 1 | 1, tm + 1, tr);
148     merge(t);
149 }
150
151 void update_chmin(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
152     if (r < tl || tr < l || v >= T[t].max1) { return; }
153     if (l <= tl && tr <= r && v > T[t].max2) {
154         push_max(t, v, tl == tr);
155         return;
156     }
157     pushdown(t, tl, tr);
158
159     int tm = (tl + tr) >> 1;
160     update_chmin(l, r, v, t << 1, tl, tm);
161     update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
162     merge(t);
163 }
164
165 void update_chmax(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
166     if (r < tl || tr < l || v <= T[t].min1) { return; }
167     if (l <= tl && tr <= r && v < T[t].min2) {
168         push_min(t, v, tl == tr);
169         return;
170     }
171     pushdown(t, tl, tr);

```

```

172
173     int tm = (tl + tr) >> 1;
174     update_chmax(l, r, v, t << 1, tl, tm);
175     update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
176     merge(t);
177 }
178
179 ll query_sum(int l, int r, int t = 1, int tl = 0, int tr = N - 1) {
180     if (r < tl || tr < l) { return 0; }
181     if (l <= tl && tr <= r) { return T[t].sum; }
182     pushdown(t, tl, tr);
183
184     int tm = (tl + tr) >> 1;
185     return query_sum(l, r, t << 1, tl, tm) +
186            query_sum(l, r, t << 1 | 1, tm + 1, tr);
187 }
188
189 int main() {
190     int Q;
191
192     cin >> N >> Q;
193     for (int i = 0; i < N; i++) { cin >> A[i]; }
194     build();
195     for (int q = 0; q < Q; q++) {
196         int t;
197         cin >> t;
198         if (t == 0) {
199             int l, r;
200             ll x;
201             cin >> l >> r >> x;
202             update_chmin(l, r - 1, x);
203         } else if (t == 1) {
204             int l, r;
205             ll x;
206             cin >> l >> r >> x;
207             update_chmax(l, r - 1, x);
208         } else if (t == 2) {
209             int l, r;
210             ll x;
211             cin >> l >> r >> x;
212             update_add(l, r - 1, x);
213         } else if (t == 3) {
214             int l, r;

```


segtreeBeat_forMod.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 100001;
5
6  int N, Q;
7  long long tsum[MAXN * 4], tmax[MAXN * 4];
8
9  void update_mod(int l, int r, long long v, int t = 1, int tl = 1, int tr = N) {
10     if (r < tl || tr < l || tmax[t] < v) {
11         return;
12     } else if (tl == tr) {
13         int val = tmax[t] % v;
14         tsum[t] = tmax[t] = val;
15         return;
16     }
17
18     int tm = (tl + tr) / 2;
19     update_mod(l, r, v, t * 2, tl, tm);
20     update_mod(l, r, v, t * 2 + 1, tm + 1, tr);
21     tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
22     tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);
23 }
24
25 void update_set(int i, long long v, int t = 1, int tl = 1, int tr = N) {
26     if (tl == tr) {
27         tsum[t] = tmax[t] = v;
28         return;
29     }
30
31     int tm = (tl + tr) / 2;
32     if (i <= tm) {
33         update_set(i, v, t * 2, tl, tm);
34     } else {
35         update_set(i, v, t * 2 + 1, tm + 1, tr);
36     }
37     tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
38     tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);

```

```

39 }
40
41 long long query(int l, int r, int t = 1, int tl = 1, int tr = N) {
42     if (r < tl || tr < l) {
43         return 0;
44     } else if (l <= tl && tr <= r) {
45         return tsum[t];
46     }
47
48     int tm = (tl + tr) / 2;
49     return query(l, r, t * 2, tl, tm) + query(l, r, t * 2 + 1, tm + 1, tr);
50 }
51
52 int main() {
53     cin >> N >> Q;
54     for (int i = 1; i <= N; i++) {
55         long long a;
56         cin >> a;
57         update_set(i, a);
58     }
59     for (int q = 0; q < Q; q++) {
60         int t;
61         cin >> t;
62         if (t == 1) {
63             int l, r;
64             cin >> l >> r;
65             cout << query(l, r) << '\n';
66         } else if (t == 2) {
67             int l, r;
68             long long x;
69             cin >> l >> r >> x;
70             update_mod(l, r, x);
71         } else if (t == 3) {
72             int i;
73             long long x;
74             cin >> i >> x;
75             update_set(i, x);
76         }
77     }
78 }

```

Sparse table

getIdxSparseTable.cpp

```

1  struct SparseTable {
2      vector<ll> A;
3      vector<int> log;
4      vector<vector<pair<ll, int>>> spt;
5
6      void init(vector<ll> &a) {
7          int n = a.size();
8          A = a;
9          log.assign(n + 1, 0);
10         for (int i = 2; i <= n; i++) {
11             log[i] = 1 + log[i / 2];
12         }
13         int k = log[n] + 1;
14         spt = vector<vector<pair<ll, int>>>(k, vector<pair<ll, int>>(n));
15         for (int i = 0; i < n; i++) {
16             spt[0][i] = { A[i], i };
17         }
18         for (int j = 1; 1 << j <= n; j++) {
19             for (int i = 0; i + (1 << j) - 1 < n; i++) {
20                 spt[j][i] = merge(spt[j - 1][i], spt[j - 1][i + (1 << (j - 1))]);
21             }
22         }
23     }
24
25     pair<ll, int> merge(pair<ll, int> &x, pair<ll, int> &y) {
26         // choose x or y
27     }
28
29     pair<ll, int> query(int i, int j) {
30         int len = j - i + 1;
31         int k = log[len];
32         return merge(spt[k][i], spt[k][j - (1 << k) + 1]);
33     }
34 };

```

sparse_segmentTree.cpp

```

1  #include <bits/stdc++.h>
2  #pragma GCC optimize("O3")
3  #define FOR(i, x, y) for (int i = x; i < y; i++)

```

```

4  #define MOD 1000000007
5  typedef long long ll;
6  using namespace std;
7
8  struct Node {
9      int sum, lazy, tl, tr, l, r;
10     Node() : sum(0), lazy(0), l(-1), r(-1) {}
11 };
12
13 const int MAXN = 123456;
14 Node segtree[64 * MAXN];
15 int cnt = 2;
16
17 void push_lazy(int node) {
18     if (segtree[node].lazy) {
19         segtree[node].sum = segtree[node].tr - segtree[node].tl + 1;
20         int mid = (segtree[node].tl + segtree[node].tr) / 2;
21         if (segtree[node].l == -1) {
22             segtree[node].l = cnt++;
23             segtree[segtree[node].l].tl = segtree[node].tl;
24             segtree[segtree[node].l].tr = mid;
25         }
26         if (segtree[node].r == -1) {
27             segtree[node].r = cnt++;
28             segtree[segtree[node].r].tl = mid + 1;
29             segtree[segtree[node].r].tr = segtree[node].tr;
30         }
31         segtree[segtree[node].l].lazy = segtree[segtree[node].r].lazy = 1;
32         segtree[node].lazy = 0;
33     }
34 }
35
36 void update(int node, int l, int r) {
37     push_lazy(node);
38     if (l == segtree[node].tl && r == segtree[node].tr) {
39         segtree[node].lazy = 1;
40         push_lazy(node);
41     } else {
42         int mid = (segtree[node].tl + segtree[node].tr) / 2;
43         if (segtree[node].l == -1) {
44             segtree[node].l = cnt++;
45             segtree[segtree[node].l].tl = segtree[node].tl;
46             segtree[segtree[node].l].tr = mid;
47         }

```

```

48     if (segtree[node].r == -1) {
49         segtree[node].r = cnt++;
50         segtree[segtree[node].r].tl = mid + 1;
51         segtree[segtree[node].r].tr = segtree[node].tr;
52     }
53
54     if (l > mid) update(segtree[node].r, l, r);
55     else if (r <= mid) update(segtree[node].l, l, r);
56     else {
57         update(segtree[node].l, l, mid);
58         update(segtree[node].r, mid + 1, r);
59     }
60
61     push_lazy(segtree[node].l);
62     push_lazy(segtree[node].r);
63     segtree[node].sum =
64         segtree[segtree[node].l].sum + segtree[segtree[node].r].sum;
65 }
66 }
67
68 int query(int node, int l, int r) {
69     push_lazy(node);
70     if (l == segtree[node].tl && r == segtree[node].tr)
71         return segtree[node].sum;
72     else {
73         int mid = (segtree[node].tl + segtree[node].tr) / 2;
74         if (segtree[node].l == -1) {
75             segtree[node].l = cnt++;
76             segtree[segtree[node].l].tl = segtree[node].tl;
77             segtree[segtree[node].l].tr = mid;
78         }
79         if (segtree[node].r == -1) {
80             segtree[node].r = cnt++;
81             segtree[segtree[node].r].tl = mid + 1;
82             segtree[segtree[node].r].tr = segtree[node].tr;
83         }
84
85         if (l > mid) return query(segtree[node].r, l, r);
86         else if (r <= mid) return query(segtree[node].l, l, r);
87         else
88             return query(segtree[node].l, l, mid) +
89                 query(segtree[node].r, mid + 1, r);
90     }

```

```

91     }
92
93     int main() {
94         iosstream::sync_with_stdio(false);
95         cin.tie(0);
96         int m;
97         cin >> m;
98
99         segtree[1].sum = 0;
100        segtree[1].lazy = 0;
101        segtree[1].tl = 1;
102        segtree[1].tr = 1e9;
103
104        int c = 0;
105        FOR(_, 0, m) {
106            int d, x, y;
107            cin >> d >> x >> y;
108            if (d == 1) {
109                c = query(1, x + c, y + c);
110                cout << c << '\n';
111            } else update(1, x + c, y + c);
112        }

```

sparse_table.cpp

```

1  template<class T>
2  struct Sparetable{
3      vector<vector<T>>> v;
4      int n , LOG;
5      void init(vector<T> &a){
6          this->n = (int) a.size();
7          this->LOG = 0;
8          int size = 1;
9          while(size <= n) size *= 2 , LOG++;
10         v.assign(n , vector<T>(LOG));
11         for (int i = 0; i < n; i++)v[i][0] = a[i];
12         for (int j = 1; (1 << j) <= n; j++){
13             for (int i = 0; (i + (1 << j) - 1) < n; i++){
14                 v[i][j] = merge(v[i][j - 1] , v[i + (1 << (j - 1))][j - 1]);
15             }
16         }

```

```

17     }
18     T merge(T a, T b){
19         return min(a, b); // change the operation
20     }
21     T qry(int l, int r){
22         int len = r - l + 1;
23         int j = 31 - __builtin_clz(len);
24         T res = merge(v[l][j], v[r - (1 << j) + 1][j]);
25         return res; // determine what you want to return
26     }
27 };

```

Treap

Treap Builtin.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /// Importing policy_based_data_structure:
5  #include <ext/pb_ds/assoc_container.hpp>
6  #include <ext/pb_ds/tree_policy.hpp>
7  using namespace __gnu_pbds;
8  /// Importing ends here.
9
10 struct vals{ /// struct for declaring struct type pb_ds:
11     int num;
12     int typ;
13     vals(int a, int b){
14         num = a;
15         typ = b;
16     }
17     bool operator <(const vals& other) const {
18         return num > other.num;
19     }
20 };
21
22 /// Supports all the operations of a set including two additional features:
23 /// 1. find_by_order(k) # Returns an iterator pointing to the k-th smallest element (zero
    based).
24 /// 2. order_of_key(k) # Returns the number of elements strictly smaller than k.
25

```

```

26  int main(){
27      /// typedef original name to pb_ds for simplicity:
28      typedef tree <int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> pb_ds;
29      /// Ordered Set
30      pb_ds treap;
31
32      treap.insert(2); /// Insert an element
33      treap.insert(3);
34
35      cout << treap.order_of_key(5) << endl; /// Returns number of elements smaller than k
36
37      pb_ds::iterator it = treap.find_by_order(0); /// Returns an iterator pointing to the k-th
smallest element
38      cout << *it << endl; /// Print the element
39
40      /// Iterate though the elements (similar to a set)
41      for(pb_ds::iterator it = treap.begin(); it != treap.end(); it++){
42          cout << *it << endl;
43      }
44
45      treap.erase(2); /// Erase an element
46      treap.clear(); /// Clear the treap
47
48      /// Struct type pb_ds. Ordering depends on the operator overloading inside the
struct.
49      typedef tree <vals, null_type, less<vals>, rb_tree_tag,
tree_order_statistics_node_update> pb_ds_st;
50
51      /// Ordered Multiset. Notice the less_equal<int> parameter.
52      typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_multiset;
53
54      ordered_multiset tmset;
55      tmset.insert(2);
56      tmset.insert(2);
57      /// Erasing is very tricky in Ordered Multiset
58      tmset.erase(tmset.find_by_order(tmset.order_of_key(2))); /// Erase 2 (Just one of the
2's get erased)
59
60      /// Pair <int,int> type Ordered Set
61      typedef tree<pair<int, int>, null_type, less_equal<pair<int, int>>, rb_tree_tag,
tree_order_statistics_node_update> ordered_pair;
62

```


Treap Implicit.cpp

```

1  /// Implicit Treap Implementation
2  /// Can handle most of the operations we do in segment tree
3  /// Ex: Range update, range query
4  /// Additionally also handles insert or erase at any position, reverse a range
5  /// Call clear() to clear the treap, then use each function carefully following the
   comments
6  /// If there is propagation, uncomment propagate() inside the functions
7  /// Note 1: All the operations are zero based
8  /// Note 2: Remember to propagate if you try to access the treap nodes outside split
   and merge
9
10 struct node{
11     int size, prior;
12     int sum, prop, key, mnn;
13     bool rev;
14     struct node *l, *r;
15     node() { }
16     node(int v) {
17         key = v;
18         prior = rand();
19         size = 1;
20         l = r = NULL;
21         sum = prop = 0;
22         mnn = inf;
23     }
24     node(int key, int prior) : key(key), prior(prior), l(NULL), r(NULL) { }
25 };
26
27 typedef node* pnode;
28
29 struct Treap{
30     pnode t;
31     Treap(){}
32
33     /// Returns size of treap
34     int size(){
35         return sz(t);
36     }
37

```

```

38  int sz(pnode t){
39      return t ? t->size:0;
40  }
41
42  int sum(pnode t){
43      return t ? t->sum:0;
44  }
45
46  int mnn(pnode t){
47      return t ? t->mnn:inf;
48  }
49
50  void upd_node(pnode t){
51      if(t){
52          t->size = sz(t->l) + 1 + sz(t->r);
53          //t->sum = sum(t->l) + t->key + sum(t->r); /// If you need sum
54          t->mnn = min(t->key, min(mnn(t->l), mnn(t->r)));
55      }
56  }
57
58  /// Works like segment tree propagation
59  void propagate(pnode t){
60      if(!t) return;
61      if(t->prop>0){ /// Propagate range addition
62          if(t->l){
63              t->l->key += t->prop;
64              t->l->sum += sz(t->l)*t->prop;
65              t->l->prop += t->prop;
66              t->l->mnn += t->prop;
67          }
68          if(t->r){
69              t->r->key += t->prop;
70              t->r->sum += sz(t->r)*t->prop;
71              t->r->prop += t->prop;
72              t->r->mnn += t->prop;
73          }
74          t->prop = 0;
75      }
76      if(t->rev){ /// Propagate range reverse
77          swap(t->l, t->r);
78          if(t->l) t->l->rev ^= true;
79          if(t->r) t->r->rev ^= true;
80          t->rev = false;
81      }

```

```

82     }
83
84     /// Split t into l and r such that all elements in l is < key and
85     /// all elements in r is >= than key
86     void split(pnode t, pnode &l, pnode &r, int key, int add = 0){
87         if(!t){
88             l = r = NULL;
89             return;
90         }
91         propagate(t);
92         int cur_key = add + sz(t->l);
93         if(cur_key < key)
94             split(t->r, t->r, r, key, add + 1 + sz(t->l)), l = t;
95         else
96             split(t->l, l, t->l, key, add), r = t;
97         upd_node(t);
98     }
99
100    /// Merge l and r into t, where all elements in l
101    /// is less than all elements in r
102    void merge(pnode &t, pnode l, pnode r){
103        propagate(l);
104        propagate(r);
105        if(!l || !r) t = l ? l : r;
106        else if(l->prior > r->prior) merge(l->r, l->r, r), t = l;
107        else merge(r->l, l, r->l), t = r;
108        upd_node(t);
109    }
110
111    void insert(pnode &t, int pos, pnode it){
112        pnode l, r, tmp;
113        split(t, l, r, pos);
114        merge(tmp, l, it);
115        merge(t, tmp, r);
116        upd_node(t);
117    }
118
119    void insertEnd(pnode &t, pnode it){
120        pnode l, r, tmp;
121        merge(t, t, it);
122        upd_node(t);
123    }
124

```

```

125 void erase(pnode &t, int key){
126     pnode t1, t2, nt1, nt2;
127     split(t, t1, t2, key+1);
128     split(t1, nt1, nt2, key);
129     merge(t, nt1, t2);
130     upd_node(t);
131     free(nt2);
132 }
133
134 int get(pnode &t, int key, int add = 0){
135     if(!t) return 0;
136     propagate(t);
137     int cur_key = add + sz(t->l);
138     if(cur_key == key){
139         return t->key;
140     }else{
141         if(cur_key < key) return get(t->r, key, add + 1 + sz(t->l));
142         else return get(t->l, key, add);
143     }
144     upd_node(t);
145 }
146
147 void print(pnode t){
148     if(!t) return;
149     propagate(t);
150     print(t->l);
151     cerr << t->key << " ";
152     print(t->r);
153 }
154
155 void nullify(pnode t){
156     if(t == NULL) return;
157     nullify(t->l); nullify(t->r);
158     delete t;
159     t->l = NULL; t->r = NULL; t = NULL;
160     free(t);
161 }
162
163 /// Insert val at position p in the treap
164 void insert(int p, int val){
165     pnode it = new node(val);
166     insert(t, p, it);
167 }

```

```

168
169     /// Insert val at the end of the treap
170     void insertEnd(int val){
171         pnode it = new node(val);
172         insertEnd(t, it);
173     }
174
175     /// Erase the element at p from the treap
176     void erase(int p){
177         erase(t, p);
178     }
179
180     /// Returns the value at position p
181     int get(int p){
182         return get(t, p);
183     }
184
185     /// Print all the elements in treap in sorted order
186     void print(){
187         cerr<<"\nPRINT TREAP: ";
188         print(t);
189         cerr<<"\n";
190     }
191
192     /// Clear the treap
193     void clear(){
194         nullify(t);
195         t = NULL;
196     }
197
198     /// Get the minimum in range u to v
199     int getRangeMin(int u, int v){
200         pnode tv, tvn, tu, tuv;
201         split(t, tv, tvn, v+1);
202         split(tv, tu, tuv, u);
203
204         int res = min(tuv->key, min(mnn(tuv->l), mnn(tuv->r)));
205         merge(tv, tu, tuv);
206         merge(t, tv, tvn);
207         return res;
208     }
209
210     /// Get the sum of range u to v

```

```

211 int getRangeSum(int u, int v){
212     pnode tv, tvn, tu, tuv;
213     split(t, tv, tvn, v+1);
214     split(tv, tu, tuv, u);
215
216     int res = tuv->sum;
217     merge(tv, tu, tuv);
218     merge(t, tv, tvn);
219     return res;
220 }
221
222 /// Rotate(right) the range from u to v k times
223 void updateRangeRotate(int u, int v, int k){
224     pnode tv, tvn, tu, tuv;
225     split(t, tv, tvn, v+1);
226     split(tv, tu, tuv, u);
227
228     int len = v - u + 1;
229     k %= len;
230
231     pnode tuv1, tuv2;
232     split(tuv, tuv1, tuv2, len-k);
233
234     merge(tuv, tuv2, tuv1);
235     merge(tv, tu, tuv);
236     merge(t, tv, tvn);
237 }
238
239 /// Reverse the range from u to v
240 void updateRangeReverse(int u, int v){
241     pnode tv, tvn, tu, tuv;
242     split(t, tv, tvn, v+1);
243     split(tv, tu, tuv, u);
244
245     tuv->rev ^= true;
246
247     merge(tv, tu, tuv);
248     merge(t, tv, tvn);
249 }
250
251 /// Add val to each node in range u to v
252 void updateRangeAdd(int u, int v, int val){

```

```

253     pnode tv, tvn;
254     split(t, tv, tvn, v+1);
255
256     pnode tu, tuv;
257     split(tv, tu, tuv, u);
258
259     tuv->key += val;
260     tuv->sum += sz(tuv)*val;

```

Treap.cpp

```

1  /// Treap which support multiple entry, works like a multiset
2  /// If you want to use like a set then just erase the element before insert
3  /// All the functions are similar to the built in treap
4  /// Every function works in log(N) except unite()
5
6  struct node{
7      int key, prior, size;
8      struct node *l, *r;
9      node() { }
10     node(int v) {
11         key = v;
12         prior = rand();
13         size = 1;
14         l = r = NULL;
15     }
16     node (int key, int prior) : key(key), prior(prior), l(NULL), r(NULL) { }
17 };
18
19 typedef node* pnode;
20
21 struct Treap{
22     pnode t;
23     Treap(){}
24
25     /// Returns size of treap
26     int size(){
27         return sz(t);
28     }
29
30     int sz(pnode t){

```

```

31     return t ? t->size:0;
32 }
33
34 void upd_sz(pnode t){
35     if(t) t->size = sz(t->l) + 1 + sz(t->r);
36 }
37
38 /// Split t into l and r such that all elements in l
39 /// is less than key and all elements in r is greater than key
40 void split(pnode t, pnode &l, pnode &r, int key){
41     if(!t) l = r = NULL;
42     else if(t->key < key) split(t->r, t->r, r, key), l = t;
43     else split(t->l, l, t->l, key), r = t;
44     upd_sz(t);
45 }
46
47 /// Merge l and r into t, where all elements in l
48 /// is less than all elements in r
49 void merge(pnode &t, pnode l, pnode r){
50     if(!l || !r) t = l ? l : r;
51     else if(l->prior > r->prior) merge(l->r, l->r, r), t = l;
52     else merge(r->l, l, r->l), t = r;
53     upd_sz(t);
54 }
55
56 /// Unite two different treap l and r into a new treap
57 /// Complexity O(N)
58 pnode unite (pnode l, pnode r) {
59     if (!l || !r) return l ? l : r;
60     if (l->prior < r->prior) swap (l, r);
61     pnode lt, rt;
62     split (r, lt, rt, l->key);
63     l->l = unite (l->l, lt);
64     l->r = unite (l->r, rt);
65     return l;
66 }
67
68 void insert(pnode &t, pnode it){
69     if(!t) t = it;
70     else if(it->prior > t->prior) split(t, it->l, it->r, it->key), t = it;
71     else insert(t->key < it->key ? t->r:t->l, it);
72     upd_sz(t);
73 }
74

```



```

75 void erase(pnode &t, int key){
76     if(!t) return;
77     else if(t->key == key){
78         pnode temp = t; merge(t, t->l, t->r); free(temp);
79     }else{
80         erase(t->key < key ? t->r:t->l,key);
81     }
82     upd_sz(t);
83 }
84
85 void init(pnode &t, int c){
86     t->prior = rand(); t->size = 1; t->l = t->r = NULL;
87     t->key = c;
88 }
89
90 void print(pnode t){
91     if(!t) return;
92     print(t->l);
93     cerr << t->key << " " << endl;
94     print(t->r);
95 }
96
97 int getKth(pnode temp, int par, int k){
98     if(temp == NULL) return 0;
99     int currSize = par + sz(temp->l) + 1;
100     if(currSize == k) return temp->key;
101
102     else if(currSize <= k) return getKth(temp->r, currSize, k);
103     else return getKth(temp->l, par, k);
104 }
105
106 int orderOf(pnode temp, int k){
107     int x = 0;
108     if(temp == NULL) return 0;
109     if(temp->key < k) return sz(temp->l) + 1 + orderOf(temp->r, k);
110     else return orderOf(temp->l, k);
111 }
112
113 void nullify(pnode t){
114     if(t == NULL) return;
115     nullify(t->l); nullify(t->r);
116     delete t;
117     t->l = NULL; t->r = NULL; t = NULL;

```

```

118     free(t);
119 }
120
121 /// Insert k in the treap
122 void insert(int k){
123     pnode it = new node(k);
124     insert(t, it);
125 }
126
127 /// Erase k for the treap
128 void erase(int k){
129     erase(t, k);
130 }
131
132 /// Returns the k'th smallest element in treap(0 based)
133 int find_by_order(int k){
134     return getKth(t, 0, k+1);
135 }
136
137 /// Returns number of elements less than k
138 int order_of_key(int k){
139     return orderOf(t, k);
140 }
141
142 /// Print all the elements in treap in sorted order
143 void print(){
144     cerr<<"\nPRINT TREAP: ";
145     print(t);
146     cerr<<"\n\n";
147 }
148
149 /// Clear the treap
150 void clear(){
151     nullify(t);
152     t = NULL;
153 }

```

Xor Basis

combining_two_xor_basis.cpp

```

1  struct Basis{
2      int basis[LOG];
3      Basis(){ memset(basis ,0, sizeof basis); }
4      void insert(int x){
5          for(int i = LOG - 1; i >= 0; --i){
6              if(!(x >> i & 1)) continue;
7              if(basis[i] == 0){
8                  basis[i] = x;
9                  return;
10             }
11             x ^= basis[i];
12         }
13     }
14     void insert(Basis &other){
15         for(int i = LOG - 1; i >= 0; --i) if(other.basis[i]){
16             insert(other.basis[i]);
17         }
18     }
19     int max_xor(){
20         int x = 0;
21         for(int i = LOG - 1; i >= 0; --i){
22             if((x >> i & 1) || basis[i] == 0) continue;
23             x ^= basis[i];
24         }
25         return x;
26     }
27     void reset(){ memset(basis ,0, sizeof basis); }
28 }

```

minOrMaxXorPathFrom1ToN.cpp

```

1  /*
2   Given an undirected connected graph with non-negative integer edge weights and
   node numbers from 1
3   to N, find a path from node 1 to node N such that the XOR of the weights of the edges
   along the path is maximized.
4   A path can pass through certain nodes or edges repeatedly. This means you are allowed
   to revisit the same node more than once, and if an edge is traversed multiple times, its
   weight must be included in the XOR each time it is used.
5   The following M lines each contain three integers u, v, and w — representing an edge
   between nodes u and v with a weight w.
6   */

```

```

7  #include <bits/stdc++.h>
8  using namespace std;
9  typedef long long ll;
10 #define f first
11 #define s second
12 const int N = 1e5 , LOG = 30;
13 int Basis[N];
14 void insert(int x){
15     for(int i = LOG - 1; i >= 0; --i){
16         if(!(x >> i & 1)) continue;
17         if(Basis[i] == 0){
18             Basis[i] = x;
19             return;
20         }
21         x ^= Basis[i];
22     }
23 }
24 int min_xor(int x){
25     for(int i = LOG - 1; i >= 0; --i){
26         if( (x ^ Basis[i]) < x) x ^= Basis[i];
27     }
28     return x;
29 }
30
31 int a[N];
32 bool vis[N];
33 vector<pair<int,int>> adj[N];
34 void dfs(int u , int p , int xor_sum){
35     if(vis[u]){
36         insert(xor_sum ^ a[u]);
37         return;
38     }
39     vis[u] = 1;
40     a[u] = xor_sum;
41     for(auto &[v , w] : adj[u]) if(v != p){
42         dfs(v , u , xor_sum ^ w);
43     }
44 }
45 int main(){
46     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
47     int n , m; cin >> n >> m;
48     for(int i = 0; i < m; ++i){
49         int u , v , w; cin >> u >> v >> w;
50         --u; --v;

```

```

51     adj[u].emplace_back(v , w);
52     adj[v].emplace_back(u , w);
53 }
54 dfs(0,-1,0);
55 cout << min_xor(a[n - 1]); // u can replace it with max xor

```

xor_basis_application.cpp

```

1  const int LOG = 30; // log(max(a))
2  int Basis[LOG] , sz;
3  void insert(int x){
4      /*
5       Insert vector into basis
6       */
7      for(int i = LOG - 1; i >= 0; --i){
8          if(!(x >> i & 1)) continue;
9          if(Basis[i] == 0){
10             Basis[i] = x;
11             ++sz;
12             return;
13         }
14         x ^= Basis[i];
15     }
16 }
17 int max_xor(){
18     /*
19     Return the maximum xor_sum over all different subsequence
20     */
21     int x = 0;
22     for(int i = LOG - 1; i >= 0; --i){
23         if(x >> i & 1) continue;
24         x ^= Basis[i];
25     }
26     return x;
27 }
28 bool check(int x){
29     /*
30     Check if there is a subsequence that xor_sum = x
31     */
32     for(int i = 0; i < LOG; ++i){
33         if(!((x >> i) & 1)) continue;
34         if(Basis[i] == 0) return false;

```

```

35     x ^= Basis[i];
36 }
37 return true;
38 }
39 int k_th(int k){
40     /*
41      Finding the k-th smallest xor_sum of all different subsequence xor_sum
42      */
43     int low = 1 << sz;
44     int x = 0;
45     for(int i = LOG - 1; i >= 0; --i){
46         if(!Basis[i]) continue;
47         low /= 2;
48         if( (!(x >> i & 1) && low < k) || ((x >> i & 1) && low >= k) ){
49             x ^= Basis[i];
50         }
51         if(low < k) k -= low;
52     }
53     return x;
54 }
55 int count(int x){
56     if(!check(x)) return false;
57     return pow(2 , n - sz); // where n is the total size of array and sz is the size of basis
58 }

```

xor_basis_lexicographically_largest.cpp

```

1  const int LOG = 30;
2  struct Basis{
3      int basis[LOG];
4      int lt[LOG];
5      Basis(){
6          memset(basis,0,sizeof basis);
7          memset(lt,-1,sizeof lt);
8      }
9      void insert(int x , int ind){
10         for(int i = LOG - 1; i >= 0; --i){
11             if(!(x >> i & 1)) continue;
12             if(lt[i] == -1){
13                 lt[i] = ind;
14                 basis[i] = x;
15                 return;

```

```

16     }
17     if(lt[i] < ind){
18         swap(lt[i] , ind);
19         swap(basis[i] , x);
20     }
21     x ^= basis[i];
22 }
23 }
24 int max_xor(int ind){
25     int x = 0;
26     for(int i = LOG - 1; i >= 0; --i){
27         if((x >> i & 1) || (lt[i] < ind) ) continue;
28         x ^= basis[i];
29     }
30     return x;
31 }
32 void reset(){ memset(lt , -1, sizeof lt); memset(basis , 0 , sizeof basis); }
33 }

```

Graph

01_BFS.cpp

```

1  /*
2      ##### O-1 BFS #####
3      Optimized algothim from dikjstra can used when weight is (0 , 1)
4      Time complexity: O(n + m)
5  */
6  // O-1 BFS
7  deque<int> q; //{x , y}
8  vector<int>> dis(n , INT32_MAX);
9  q.push_front();
10 dis[] = 0;
11 while(!q.empty()){
12     // if new weight incearse by 1 >> push_back in deque
13     // if new weight still the same >> push_front in deque
14 }

```

BFS.cpp

```

1  #include <bits/stdc++.h>

```

```

2  typedef long long ll;
3  #define s second
4  #define f first
5  using namespace std;
6  int main(){
7      /*
8          ----- BFS Algorithim -----
9          - use to Find Shortest path from Single Source to other vertices
10         - can used if weight of edge == 1
11         - Find answer in O(n + m)
12      */
13     int n , m; cin >> n >> m; // n : number of vertices , m : number of edges
14     vector<int> adj[n];
15     for(int i = 0 ; i < m ; i++){
16         int u , v; cin >> u >> v;
17         adj[u].emplace_back(v);
18         adj[v].emplace_back(u);
19     }
20     int Start; cin >> Start; // vertice that you start from
21     Start--;
22     queue<int> q;
23     vector<int> dis(n , -1);
24     vector<int> par(n , -1);
25     dis[Start] = 0; q.push(Start);
26     while(q.size()){
27         int u = q.front(); q.pop();
28         for(auto &v : adj[u]){
29             if(!~dis[v]){
30                 dis[v] = dis[u] + 1;
31                 q.push(v);
32                 par[v] = u;
33             }
34         }
35     }
36     // Find the path
37     vector<int> path;
38     function<void(int)> gen = [&](int i){
39         path.emplace_back(i);
40         if(~par[i]) gen(par[i]);
41     };
42     reverse(path.begin() , path.end());
43 }

```


Bridges.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  const int N = 1e5 + 9;
8  vector<vector<pair<int,int>>> adj;
9  vector<vector<int>> BridgeTree;
10 vector<int> lowLink , dfn , isBridge , comp;
11 int ndfn , comp_num , total;
12 void tarjan(int u , int par){
13     dfn[u] = lowLink[u] = ndfn++;
14     for(auto &[v , id] : adj[u]){
15         if(dfn[v] == -1){
16             tarjan(v , u);
17             lowLink[u] = min(lowLink[u] , lowLink[v]);
18             if(lowLink[v] == dfn[v]){
19                 isBridge[id] = true;
20                 total++;
21             }
22         }else if(v != par){
23             lowLink[u] = min(lowLink[u] , dfn[v]);
24         }
25     }
26 }
27 void Find_component(int u , int par){
28     comp[u] = comp_num;
29     for(auto &[v , id] : adj[u]) if(comp[v] == -1 && isBridge[id] == 0)
30         Find_component(v , u);
31 }
32 pair<int , int> diameter(int u , int par = -1)
33 {
34     int diam = 0;
35     int mxHeights[3] = {-1, -1, -1}; // keep 2 highest trees
36     for(auto &v : BridgeTree[u]) if(v != par)
37     {
38         auto p = diameter(v , u);
39         diam = max(diam , p.f);
40         mxHeights[0] = p.s+1;
41         sort(mxHeights , mxHeights+3);

```

```

42     }
43     for(int i = 0; i < 3; i++) if(mxHeights[i] == -1)
44         mxHeights[i] = 0;
45     diam = max(diam, mxHeights[1] + mxHeights[2]);
46     return {diam, mxHeights[2]};
47 }
48 int main(){
49     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
50     #ifndef ONLINE_JUDGE
51         freopen("in.txt", "r", stdin);
52         freopen("out.txt", "w", stdout);
53         freopen("error.txt", "w", stderr);
54     #endif
55     int q; cin >> q;
56     while(q--){
57         int n, m; cin >> n >> m;
58         // reset
59         dfn.assign(n, -1);
60         comp.assign(n, -1);
61         lowLink.assign(n, 0);
62         isBridge.assign(m, 0);
63         ndfn = comp_num = total = 0;
64         adj.assign(n, vector<pair<int,int>>());
65
66         for(int i = 0; i < m; i++){
67             int u, v; cin >> u >> v;
68             --u; --v;
69             adj[u].emplace_back(v, i);
70             adj[v].emplace_back(u, i);
71         }
72         // Finding Bridges using Tarjan algo.
73         tarjan(0, 0);
74         // dfs to group all the maximal components together, so that we can shrink it to one
node
75         for(int i = 0; i < n; i++) if(comp[i] == -1){
76             Find_component(i, i);
77             comp_num++;
78         }
79         // shrinking all the maximal components to one node
80         BridgeTree.assign(comp_num, vector<int>());
81         for(int u = 0; u < n; u++) for(auto &[v, id] : adj[u]) if(isBridge[id]){
82             BridgeTree[comp[u]].emplace_back(comp[v]);
83         }
84         // Finding the diameter of the Bridgestree

```

```

85     int d = diameter(O , O).f;
86 }

```

Hierholzer.cpp

```

1  /*
2   # Hierholzer's Algorithm for directed graph
3
4   Euler circuit is a path that traverses every edge of a graph, and the path ends on the
   starting vertex
5   Problem: Given a directed Eulerian graph, print an Euler circuit
6
7   restrictions:
8   A directed graph has an eulerian cycle if following conditions are true
9       1. All vertices with nonzero degree belong to a single strongly connected component.
10      2. In degree is equal to the out degree for every vertex.
11
12   Idea:
13   Choose any starting vertex v, and follow a trail of edges from that vertex until
   returning to v. It is not possible to get stuck at any vertex other than v,
14   because indegree and outdegree of every vertex must be same, when the trail enters
   another vertex w there must be an unused edge leaving w.
15   The tour formed in this way is a closed tour, but may not cover all the vertices and
   edges of the initial graph.
16   As long as there exists a vertex u that belongs to the current tour, but that has
   adjacent edges not part of the tour,
17   start another trail from u, following unused edges until returning to u, and join the tour
   formed in this way to the previous tour.
18
19
20   Time complexity : O(V+E)
21   Space complexity : O(V+E)
22 */
23
24
25 // Don't forget to check if the graph has an euler circuite or not.
26 #include <bits/stdc++.h>
27 using namespace std;
28 typedef long long ll;
29 #define rep(i , st , ed) for(int i = st; i < ed; i++)
30 #define f first
31 #define s second

```

```

32 #define all(v) v.begin() , v.end()
33 #ifndef ONLINE_JUDGE
34 #define debug(x) cerr << #x << ": " << x << '\n';
35 #else
36 #define debug(x)
37 #endif
38 const int N = 1e5 + 9;
39 vector<int> adj[N];
40 int main(){
41     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
42     #ifndef ONLINE_JUDGE
43     freopen("in.txt", "r", stdin);
44     freopen("out.txt", "w", stdout);
45     freopen("error.txt", "w", stderr);
46     #endif
47     int n , m; cin >> n >> m;
48     for(int i = 0; i < m; ++i){
49         int u , v; cin >> u >> v;
50         --u; --v;
51         adj[u].emplace_back(v);
52     }
53     vector<int> edge_count(n);
54     for (int i = 0; i < n; i++) edge_count[i] = adj[i].size();
55     stack<int> cur_path; // Maintain a stack to keep vertices
56     vector<int> circuit; // vector to store final circuit
57     cur_path.push(0); // start from any vertex
58     int cur_v = 0; // curent vertex
59     while (!cur_path.empty()){
60         if (edge_count[cur_v]){ // If there's remaining edge
61             cur_path.push(cur_v); // Push the vertex
62             int next_v = adj[cur_v].back(); // Find the next vertex using an edge
63             edge_count[cur_v]--; // and remove that edge
64             adj[cur_v].pop_back();
65             cur_v = next_v; // Move to next vertex
66         }
67
68         else{ // back-track to find remaining circuit
69             circuit.push_back(cur_v);
70             // Back-tracking
71             cur_v = cur_path.top();
72             cur_path.pop();
73         }
74     }
75

```

```

1 #include <bits/stdc++.h>
2 typedef long long ll;
3 #define s second
4 #define f first
5 #define rep(i , st , ed) for(int i = st ; i < ed ; i++)
6 const int N = 500;
7 using namespace std;
8 void burn(){
9 ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
10 #ifndef ONLINE_JUDGE
11     freopen("in.txt" , "r" , stdin);
12     freopen("out.txt" , "w" , stdout);
13     freopen("error.txt" , "w" , stderr);
14 #endif
15 }
16 //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
17 int reach[N][N] , dag[N][N];
18 vector<int> comp , sz;
19 void SCC(int n){
20     comp.clear(); comp.resize(n , -1);
21     sz.clear();
22     // Floyd
23     rep(k , 0 , n) rep(i , 0 , n) rep(j , 0 , n)
24         reach[i][j] |= (reach[i][k] && reach[k][j]); // Warshall Transitive closure
25     int cnt = 0;
26     // detect SCC
27     rep(i , 0 , n){
28         if(comp[i] == -1){
29             comp[i] = cnt++;
30             rep(j , 0 , n)
31                 if(reach[i][j] && reach[j][i]) comp[j] = comp[i];
32         }
33     }
34     sz.resize(cnt);
35     rep(i , 0 , n) sz[comp[i]]++;
36     // Create Dag

```

```

37     rep(i , 0 , n) rep(j , 0 , n)
38         if(reach[i][j]) dag[comp[i]][comp[j]] = 1;
39     }
40     int main(){
41         burn();
42         int n , m; cin >> n >> m;
43         for(int i = 0 ; i < m ; i++){
44             int u , v; cin >> u >> v;
45             --u; --v;
46             reach[u][v] = 1;
47         }
48         SCC(n);
49     }

```

SCC_kosaraju.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  #define all(v) v.begin() , v.end()
8  const int N = 2e5 + 9;
9  vector<int> adj[N], adj_rev[N];
10 bool used[N];
11 vector<int> order, component;
12
13 void dfs1(int v) {
14     used[v] = true;
15     for (auto u : adj[v])
16         if (!used[u])
17             dfs1(u);
18     order.push_back(v);
19 }
20 void dfs2(int v) {
21     used[v] = true;
22     component.push_back(v);
23     for (auto u : adj_rev[v])
24         if (!used[u]) dfs2(u);
25 }
26

```

```

27  int main() {
28      #ifndef ONLINE_JUDGE
29      freopen("in.txt", "r", stdin);
30      freopen("out.txt", "w", stdout);
31      freopen("error.txt", "w", stderr);
32      #endif
33      int n,m; cin >> n >> m;
34      for (int i = 0; i < m; ++i) {
35          int a, b; cin >> a >> b;
36          --a; --b;
37          adj[a].push_back(b);
38          adj_rev[b].push_back(a);
39      }
40      for(int i = 0; i < n; ++i) used[i] = false;
41      for (int i = 0; i < n; i++) if (!used[i]){
42          dfs1(i);
43      }
44      for(int i = 0; i < n; ++i) used[i] = false;
45      reverse(order.begin(), order.end());
46      for (auto v : order)if (!used[v]) {
47          dfs2 (v);
48          component.clear();
49      }
50  }

```

SCC_tarjan.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  #define rep(i , st , ed) for(int i = st ; i < ed ; i++)
6  using namespace std;
7  void burn(){
8  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
9  #ifndef ONLINE_JUDGE
10     freopen("in.txt", "r", stdin);
11     freopen("out.txt", "w", stdout);
12     freopen("error.txt", "w", stderr);
13 #endif
14 }
15 const int N = 1e5;

```

```

16  //\\//\\//\\//\\//\\//\\//\\//\\//\\//
17  vector<vector<int>> adj , dag , comps;
18  int comp[N] , inStack[N] , lowLink[N] , dfn[N] , deg[N];
19  stack<int> st;
20  int ndfn;
21  void tarjan(int u){
22      dfn[u] = lowLink[u] = ndfn++;
23      inStack[u] = true;
24      st.push(u);
25      for(auto &v : adj[u]){
26          if(dfn[v] == -1){
27              tarjan(v);
28              lowLink[u] = min(lowLink[u] , lowLink[v]);
29          }else if(inStack[v]){
30              lowLink[u] = min(lowLink[u] , dfn[v]);
31          }
32      }
33      if(dfn[u] == lowLink[u]){
34          // head of component
35          int x = -1;
36          comps.emplace_back(vector<int>());
37          while(x != u){
38              x = st.top(); st.pop(); inStack[x] = 0;
39              comps.back().emplace_back(x);
40              comp[x] = comps.size() - 1;
41          }
42      }
43  }
44  void genDag(){
45      dag.resize(comps.size());
46      for(int u = 0 ; u < adj.size() ; u++){
47          for(auto &v : adj[u]){
48              if(comp[u] != comp[v]){
49                  dag[comp[u]].emplace_back(comp[v]);
50                  deg[comp[v]]++;
51              }
52          }
53      }
54  }
55  void SCC(int n){
56      ndfn = 0;
57      comps.clear();
58      rep(i , 0 , n){
59          dfn[i] = -1;

```


[illegible]

```

24  for(auto &v : adj[u]){
25      if(dfn[v] == -1){
26          tarjan(v, u);
27          lowLink[u] = min(lowLink[u] , lowLink[v]);
28          if (lowLink[v] >= dfn[u]){
29              if (dfn[u] == 0 && root == false)
30                  root = true;
31              else artpoints.emplace(u);
32          }
33      }else if(v != par){
34          lowLink[u] = min(lowLink[u] , dfn[v]);
35      }
36  }
37 }
38 int main(){
39     burn();
40     int n , m; cin >> n >> m;
41     for (int i = 0; i < n; ++i)
42         dfn[i] = -1;
43     adj.resize(n);
44     for(int i = 0 ; i < m ; i++){
45         int u , v; cin >> u >> v;
46         --u; --v;
47         adj[u].emplace_back(v);
48         adj[v].emplace_back(u);
49     }
50     tarjan(0, -1);
51     // ALL articulation points are stored in **artspoints** set
52 }

```

bellman_ford.cpp

```

1  struct Edge {
2      int a, b, cost;
3  };
4
5  int n, m;
6  vector<Edge> edges;
7  const int INF = 1000000000;
8
9  void solve()
10 {

```

```

11  vector<int> d(n);
12  vector<int> p(n, -1);
13  int x;
14  for (int i = 0; i < n; ++i) {
15      x = -1;
16      for (Edge e : edges) {
17          if (d[e.a] + e.cost < d[e.b]) {
18              d[e.b] = d[e.a] + e.cost;
19              p[e.b] = e.a;
20              x = e.b;
21          }
22      }
23  }
24
25  if (x == -1) {
26      cout << "No negative cycle found.";
27  } else {
28      for (int i = 0; i < n; ++i)
29          x = p[x];
30
31      vector<int> cycle;
32      for (int v = x;; v = p[v]) {
33          cycle.push_back(v);
34          if (v == x && cycle.size() > 1)
35              break;
36      }
37      reverse(cycle.begin(), cycle.end());
38
39      cout << "Negative cycle: ";
40      for (int v : cycle)
41          cout << v << ' ';
42      cout << endl;
43  }
44  }

```

biConnected.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  #define rep(i, st, ed) for(int i = st; i < ed; i++)

```



```

50  int n , m; cin >> n >> m;
51  for (int i = 0; i < n; ++i)
52      dfn[i] = -1;
53  adj.resize(n);
54  for(int i = 0 ; i < m ; i++){
55      int u , v; cin >> u >> v;
56      --u; --v;
57      adj[u].emplace_back(v);
58      adj[v].emplace_back(u);
59  }
60  tarjan(0, -1);
61  // bi_connected vector stores all the edges in each biconnected component
62  // bi_connected.size() is the number of biconnected componenets

```

dijkstra.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  using namespace std;
6  int main(){
7      /*
8          ----- Dijkstra Algrothim -----
9          - use to Find Shortest path from Single Sourse to other vertices
10         - can used if weight of edge >= 0
11         - Find answer in O(nlog(n))
12         */
13     int n , m; cin >> n >> m; // n : number of vertices , m : number of edges
14     vector<pair<int,ll>> adj[n]; // {v , w}
15     for(int i = 0 ; i < m ; i++){
16         int u , v; ll w; cin >> u >> v >> w;
17         adj[u].emplace_back(v , w);
18         adj[v].emplace_back(u , w);
19     }
20     int Start; cin >> Start; // vertice that you start from
21     Start--;
22     priority_queue<pair<ll,int>> q; // {dis , u}
23     vector<ll> dis(n , 1e15);
24     q.push({0 , Start});
25     dis[Start] = 0;
26     while(q.size()){

```

```

27     auto [d, u] = q.top().s; q.pop();
28     if(-d != dis[u]) continue;
29     vis[u] = true;
30     for(auto &[v, w] : adj[u]){
31         if(dis[v] > dis[u] + w){ // relaxing
32             dis[v] = dis[u] + w;
33             q.push({-dis[v], v});
34         }
35     }
36 }
37 }

```

dijkstra_sparse_graph.cpp

```

1  /*
2  O(n^2 + m)
3  */
4  const int INF = 1000000000;
5  vector<vector<pair<int, int>>> adj;
6
7  void dijkstra(int s, vector<int> & d, vector<int> & p) {
8      int n = adj.size();
9      d.assign(n, INF);
10     p.assign(n, -1);
11     vector<bool> u(n, false);
12
13     d[s] = 0;
14     for (int i = 0; i < n; i++) {
15         int v = -1;
16         for (int j = 0; j < n; j++) {
17             if (!u[j] && (v == -1 || d[j] < d[v]))
18                 v = j;
19         }
20
21         if (d[v] == INF)
22             break;
23
24         u[v] = true;
25         for (auto edge : adj[v]) {
26             int to = edge.first;
27             int len = edge.second;
28

```

```

29     if (d[v] + len < d[to]) {
30         d[to] = d[v] + len;
31         p[to] = v;
32     }
33 }
34 }
35 }

```

euler_ciruite_undirectedgraph.cpp

```

1  /*
2   Eulerian Circuit is an Eulerian Path that starts and ends on the same vertex.
3
4   Restrictions:
5   An undirected graph has Eulerian cycle if following two conditions are true.
6       1. All vertices with non-zero degree are connected. We don't care about vertices
        with zero degree
7       because they don't belong to Eulerian Cycle or Path (we only consider all edges).
8       2. All vertices have even degree.
9
10 */
11 #include <bits/stdc++.h>
12 using namespace std;
13 typedef long long ll;
14 #define rep(i , st , ed) for(int i = st; i < ed; i++)
15 #define f first
16 #define s second
17 const int N = 1e5 + 9 , M = 2e5 + 9;
18 vector<pair<int,int>> adj[N];
19 int vis[M];
20 int main(){
21     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
22     #ifndef ONLINE_JUDGE
23     freopen("in.txt", "r", stdin);
24     freopen("out.txt", "w", stdout);
25     freopen("error.txt", "w", stderr);
26     #endif
27     int n, m; cin >> n >> m;
28     for(int i = 0; i < m; ++i){
29         int u,v; cin >> u >> v;
30         --u; --v;
31         adj[u].emplace_back(v,i);

```

```

32     adj[v].emplace_back(u,i);
33 }
34 for(int i = 0; i < n; ++i) if ((int) adj[i].size() & 1){
35     cout << "IMPOSSIBLE"; // Handling manual
36     return 0;
37 }
38 // there could be more than one euler circuit if the graph aren't connected ans we will
find one of them;
39 stack<int> st;
40 for(int i = 0; i < n; ++i) if(adj[i].size()){
41     st.push(i);
42     break;
43 }
44 // if st.empty() --> no.edges = 0
45 vector<int> path;
46 while(!st.empty()){
47     int v = st.top();
48     int f=0;
49     while(!adj[v].empty()) {
50         auto [u,i] = adj[v].back();
51         adj[v].pop_back();
52         if (!vis[i]) {
53             st.push(u);
54             vis[i]=1;
55             f=1;
56             break;
57         }
58     }
59     if (!f){
60         path.emplace_back(v);
61         st.pop();
62     }
63 }
64 for (auto &i: path) cout << i + 1 << " ";
65 }

```

euler_path_directed.cpp

```

1  /*
2   Finding euler path in directed graph
3   Time complexity: O(N + M)
4   Space complexity: O(N + M)

```



```

5  */
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef long long ll;
9  #define rep(i , st , ed) for(int i = st; i < ed; i++)
10 #define f first
11 #define s second
12 const int N = 1e5 + 9; // no.of node
13 vector<int> adj[N];
14 vector<int> path; // euler path
15 void dfs(int s){
16     while((int) adj[s].size()){
17         int u = adj[s].back();
18         adj[s].pop_back();
19         dfs(u);
20     }
21     path.emplace_back(s);
22 }
23 int main(){
24     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
25     #ifndef ONLINE_JUDGE
26     freopen("in.txt", "r", stdin);
27     freopen("out.txt", "w", stdout);
28     freopen("error.txt", "w", stderr);
29     #endif
30     int n,m; cin >> n >> m;
31     int in[n] = {}, out[n] = {};
32     for(int i = 0; i < m; ++i){
33         int x,y; cin >> x >> y;
34         --x; --y;
35         adj[x].emplace_back(y);
36         in[y]++, out[x]++;
37     }
38     int a=0,b=0,c=0,s1=0,s2=0;
39     for(int i = 0; i < n; ++i){
40         if (in[i]==out[i]) c++;
41         if (in[i]-out[i]==1){ b++; s2=i; }
42         if (in[i]-out[i]==-1){ a++; s1=i; }
43     }
44     if (s1 != 0 || s2 != n - 1){
45         cout << "IMPOSSIBLE";
46         return 0;
47     }
48     if (!(c==n-2 && a==1 && b == 1)){

```

```

49     cout << "IMPOSSIBLE";
50     return 0;
51 }
52 dfs(0);
53 if (path.size() != m + 1){
54     cout << "IMPOSSIBLE";
55     return 0;
56 }
57 reverse(path.begin(), path.end());
58 for (auto &i: path) cout << i + 1 << " ";
--

```

euler_path_undirected.cpp

```

1  /*
2   Finding Euler path in undirected graph
3   Time complexity: O(M)
4   Space complexity: O(M + N)
5
6   Standard problem: https://cses.fi/problemset/task/1691
7  */
8  #include <bits/stdc++.h>
9  using namespace std;
10 typedef long long ll;
11 #define rep(i, st, ed) for(int i = st; i < ed; i++)
12 #define f first
13 #define s second
14 const int N = 1e5 + 9, M = 2e5 + 9;
15 vector<pair<int,int>> adj[N];
16 int vis[M];
17 int main(){
18     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
19     #ifndef ONLINE_JUDGE
20     freopen("in.txt", "r", stdin);
21     freopen("out.txt", "w", stdout);
22     freopen("error.txt", "w", stderr);
23     #endif
24     int n, m; cin >> n >> m;
25     for(int i = 0; i < m; ++i){
26         int u,v; cin >> u >> v;
27         --u; --v;
28         adj[u].emplace_back(v,i);
29         adj[v].emplace_back(u,i);

```

```

30     }
31     for(int i = 0; i < n; ++i) if ((int) adj[i].size() & 1){
32         cout << "IMPOSSIBLE"; // Handling manual
33         return 0;
34     }
35     stack<int> st;
36     st.push(0);
37     vector<int> path;
38     while(!st.empty()){
39         int v = st.top();
40         int f=0;
41         while(!adj[v].empty()) {
42             auto [u,i] = adj[v].back();
43             adj[v].pop_back();
44             if (!vis[i]) {
45                 st.push(u);
46                 vis[i]=1;
47                 f=1;
48                 break;
49             }
50         }
51         if (!f){
52             path.emplace_back(v);
53             st.pop();
54         }
55     }
56     if ((int) path.size() != m + 1){
57         cout << "IMPOSSIBLE"; // Handling manual
58         return 0;
59     }
60     for (auto &i: path) cout << i + 1 << " ";
61 }

```

floyd.cpp

```

1  int n , m; cin >> n >> m;
2  vector<vector<ll>> adj(n , vector<ll>(n , OOLL));
3  for(int i = 0; i < n; ++i) adj[i][i] = 0;
4  for(int i = 0; i < m; ++i){
5      int u , v; ll w; cin >> u >> v >> w;
6      --u; --v;
7      adj[u][v] = min(adj[u][v] , w);

```

```

8  adj[v][u] = min(adj[v][u] , w);
9  }
10 // Floyd
11 rep(k , 0 , n) rep(i , 0 , n) rep(j , 0 , n){
12  adj[i][j] = min(adj[i][j] , adj[i][k] + adj[k][j]);
13  }

```

kth_shortest_path.cpp

```

1  /*
2   Finding the First k's shortest path in
3   O(m * k) such that m : no. of edges
4   */
5  #include <bits/stdc++.h>
6  using namespace std;
7  typedef long long ll;
8  #define rep(i , st , ed) for(int i = st; i < ed; i++)
9  #define f first
10 #define s second
11 int main(){
12  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
13  #ifndef ONLINE_JUDGE
14  freopen("in.txt", "r", stdin);
15  freopen("out.txt", "w", stdout);
16  freopen("error.txt", "w", stderr);
17  #endif
18  int n , m , k; cin >> n >> m >> k;
19  vector<pair<int,ll>> adj[n]; // {v , w}
20  for(int i = 0; i < m; ++i){
21      int u , v; ll w; cin >> u >> v >> w;
22      --u; --v;
23      adj[u].emplace_back(v , w);
24  }
25  priority_queue<pair<ll,int>> q; // {-w , u}
26  vector<ll> cnt(n) , ans;
27  int start = 0 , end = n - 1;
28  q.push({0 , start});
29  while(q.size() && cnt[end] < k){
30      auto [d , u] = q.top(); q.pop();
31      d *= -1;
32      cnt[u]++;
33      if(u == end) ans.emplace_back(d);

```

```

34     if(cnt[u] <= k){
35         for(auto &[v, w] : adj[u]) if(cnt[v] < k)
36             q.push({-(w + d), v});
37     }
38 }
39 sort(ans.begin(), ans.end());
40 for(auto &w : ans) cout << w << " ";
41 }

```

Handbook material

Imgs

Masking

Generate_next_lexicographical_K-combination.cpp

```

1  bool next_combination(vector<int>& a, int n) {
2      int k = (int)a.size();
3      for (int i = k - 1; i >= 0; i--) {
4          if (a[i] < n - k + i + 1) {
5              a[i]++;
6              for (int j = i + 1; j < k; j++)
7                  a[j] = a[j - 1] + 1;
8              return true;
9          }
10     }
11     return false;
12 }

```

Generating_all_submasking_of_k_ones.cpp

```

1  /*
2      Gosper's Hack (Bankers sequence)
3      Time Complexity: nCk
4  */
5  void f(int mask){
6      // process the current mask with k 1's
7  }
8  void GopersHack(int n,int k){

```

```

9     int sets=(1ll<<k)-1;
10    int limit=(1ll<<n);
11    while(sets<limit){
12        f(sets);
13        int c= sets & -sets;
14        int r= sets + c;
15        sets=(( (r^sets) >>2 ) /c ) | r;
16    }
17 }

```

gen_all_possible_submasking.cpp

```

1  /*
2     ----> Generating All possible submasks <-----
3     - Iterating through all masks with their submasks. Complexity  $O(3^n)$ 
4  */
5  // m : mask , s : submask
6  for (int m=0; m<(1<<n); ++m)
7      for (int s=m; s; s=(s-1)&m)

```

Math

Counting

BurnsideLemma.cpp

```

1  /// Burnside Lemma Notes:
2
3  /**
4   Problem 1: Consider a circular stripe of N cells and we are given M colors.
5   In how many ways we can color the stripe. 2 ways are same if we can make one from
6   other using rotation.
7
8   Here, X is a set of all colored stripes (it has  $M^N$  elements),
9   G is the group of its rotations (it has N elements: rotation by 0 cells, by 1 cell...by (N-1)
10  cells),
11  An orbit is exactly the set of all stripes that can be obtained from each other using
12  rotations,
13  So the number of orbits will be the number of distinct stripes up to a rotation.
14
15

```

12 Now let's apply the lemma, and find the number of stripes that are fixed by the rotation by K cells.

13 If a stripe becomes itself after rotating by K cells, then its 1st cell must have the same color as its $(1+K \text{ modulo } N)$ 'th cell,

14 which is in turn the same as its $(1+2K \text{ modulo } N)$ 'th cell...until we get back to the 1st cell again when $(P \cdot K \% N) = 0$.

15

16 This will happen when $P = N/\gcd(K, N)$, and thus we get $N/\gcd(K, N)$ cells that must all be of the same color.

17 Analogously, the same amount of cells must be of the same color starting with cell 2, $(2+K \text{ modulo } N)$ etc.

18

19 Thus, all cells are separated into $\gcd(K, N)$ groups, with each group being of one color, and that yields us $M^{\gcd(K, N)}$ choices.

20 And by Burnside's lemma, the answer to the original problem is $\sum(M^{\gcd(K, N)})/N$, for K from 0 to $N-1$

21 `**/`

22

23 `/**`

24 Problem 2: You have 4 red, 4 white, and 4 blue identical dinner plates.

25 In how many different ways can you set a square table with one plate on each side?

26 2 ways are same if we can make one from other using rotation.

27

28 We have four possible rotations (clockwise) 0, 90, 180 and 270 degrees.

29 Let's A_0 = rotation by 0, A_1 = rotation by 90... A_3 = rotation by 270 degree

30 So we have cyclic group of 4 elements (possible rotations)

31

32 Let's $S = RWBR$ is a valid arrangement where R is on north, W on East, B on south and R on west

33 So $A_1(S) = RRWB$ (rotation by 90 degree)

34

35 Now using Burnside lemma let's find how many arrangements are fixed under various rotations.

36

37 For A_0 we rotate S by 0 degree. So there are 3^4 fixed points.

38

39 For A_1 we rotate S by 90 degree. If S and $A_1(S)$ will have to be same then,

40 north-east must have same color, east-south must have same color,

41 same for south-west and west-north. Which means all side must have same color.

42 So there are 3 fixed points for A_1 .

43

44 A_3 is same as A_1 , because rotation by 270 degree does same as rotation by 90 degree.

45 So there are 3 fixed points for A_3 too.

46

47 For A2 we rotate S by 180 degree. If S and A2(S) will have to be same then,
 48 north-south must have same color and east-west must have same color.
 49 So we have 3*3 fixed points for A2.
 50
 51 So there are total $(3^4 + 3 + 3 + 3*3) = 96$ fixed points.
 52 And by Burnside's lemma, the answer to the original problem is $96/4 = 24$.

FFT.cpp

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  using namespace std;
5
6
7  // FFT
8  namespace FFT {
9      using DD = double;
10     const DD PI = acos(-1);
11
12     struct Comp {
13         DD real, imag;
14         Comp(DD real = 0, DD imag = 0) : real(real), imag(imag) {}
15         friend inline ostream& operator << (ostream &s, const Comp &c) {
16             return s << '<' << c.real << ',' << c.imag << '>';
17         }
18         inline Comp operator + (const Comp &c) {
19             return {real + c.real, imag + c.imag};
20         }
21         inline Comp operator - (const Comp &c) {
22             return {real - c.real, imag - c.imag};
23         }
24         inline Comp operator * (const Comp &c) {
25             return {real * c.real - imag * c.imag,
26                     real * c.imag + imag * c.real};
27         }
28         inline Comp operator * (DD a) {
29             return {real * a, imag * a};
30         }
31         inline Comp operator / (DD a) {
32             return {real / a, imag / a};
33         }

```



```

34     };
35
36     // FFT
37     void trans(vector<Comp> &v, bool inv = false) {
38         int n = (int)v.size();
39         for (int i = 0, j = 1; j < n-1; j++) {
40             for (int k = n>>1; k > (i ^ k); k >>= 1);
41             if (i > j) swap(v[i], v[j]);
42         }
43         for (int t = 2; t <= n; t <= 1) {
44             DD ang = acos(-1.0) * 2 / t;
45             if (inv) ang = -ang;
46             for (int i = 0; i < n; i += t) {
47                 for (int j = 0; j < t/2; ++j) {
48                     Comp w = {cos(ang * j), sin(ang * j)};
49                     int j1 = i + j, j2 = i + j + t/2;
50                     Comp c1 = v[j1], c2 = v[j2] * w;
51                     v[j1] = c1 + c2;
52                     v[j2] = c1 - c2;
53                 }
54             }
55         }
56         if (inv) for (int i = 0; i < n; ++i) v[i] = v[i]/n;
57     }
58
59     // A * B
60     vector<long long> mult(const vector<long long> &A,
61                           const vector<long long> &B) {
62         int size_a = 1; while (size_a < A.size()) size_a <= 1;
63         int size_b = 1; while (size_b < B.size()) size_b <= 1;
64         int size_fft = max(size_a, size_b) <= 1;
65
66         vector<Comp> cA(size_fft), cB(size_fft), cC(size_fft);
67         for (int i = 0; i < A.size(); ++i) cA[i] = {(DD)A[i], 0};
68         for (int i = 0; i < B.size(); ++i) cB[i] = {(DD)B[i], 0};
69
70         trans(cA); trans(cB);
71         for (int i = 0; i < size_fft; ++i) cC[i] = cA[i] * cB[i];
72         trans(cC, true);
73
74         vector<long long> res((int)A.size() + (int)B.size() - 1);
75         for (int i = 0; i < res.size(); ++i) {
76             res[i] = (long long)(cC[i].real + 0.5);
77         }

```

```

78     return res;
79 }
80 };
81
82
83
84 //-----//
85 // Examples
86 //-----//
87
88 int main() {
89     int N;
90     while (cin >> N) {
91         vector<long long> a(N), b(N);
92         for (int i = 0; i < N; ++i) cin >> a[i] >> b[i];
93         auto res = FFT::mult(a, b);
94         cout << 0 << endl;
95         for (int i = 0; i < N*2-1; ++i) cout << res[i] << endl;
96     }

```

JosephusTheorem.cpp

```

1  /// Given a group of n men arranged in a circle under the edict that every k'th man
2  /// will be executed going around the circle until only one remains.
3  /// Find out who will be the final survivor.
4
5  int josephus(int n, int k){
6      if (n == 1) return 1;
7      return (josephus(n - 1, k) + k-1) % n + 1;
8  }
9
10 /// log base solution when k is 2
11 int josephus(int n){
12     int p = 1;
13     while (p <= n) p *= 2;
14     return (2*n) - p + 1;
15 }

```

Matrix_Exponential.cpp

```

1  struct Mat {
2      ll mat[3][3];
3      ll row, col;
4
5      Mat(ll _r, ll _c) : row(_r), col(_c) {
6          memset(mat, 0, sizeof mat);
7
8      }
9
10     Mat operator *(const Mat& b) const {
11         Mat Product(row, b.col);
12         for(int i = 0; i < row; ++i) {
13             for(int k = 0; k < col; ++k) {
14                 if(mat[i][k] != 0) {
15                     for(int j = 0; j < b.col; ++j) {
16                         Product.mat[i][j] += mat[i][k] * b.mat[k][j] % mod;
17                     }
18                 }
19
20             }
21             for(int j = 0; j < b.col; ++j) {
22                 Product.mat[i][j] %= mod;
23             }
24         }
25         return Product;
26     }
27 };
28
29 Mat power(Mat a, ll b) {
30     Mat res(a.row, a.col);
31     for(int i = 0; i < res.row; ++i) res.mat[i][i] = 1;
32     while(b > 0) {
33         if(b & 1) {
34             res = res * a;
35         }
36         a = a * a;
37         b >>= 1;
38     }
39     return res;
40 }

```

Mint.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define rrep(i,a,b) for(int i=a;i>=b;i--)
6
7  #define f first
8  #define s second
9  template<int MOD> struct ModInt {
10     static const int Mod = MOD; unsigned x; ModInt() : x(0) { }
11     ModInt(signed sig) { x = sig < 0 ? sig % MOD + MOD : sig % MOD; }
12     ModInt(signed long long sig) { x = sig < 0 ? sig % MOD + MOD : sig % MOD; }
13     int get() const { return (int)x; }
14     ModInt &operator+=(ModInt that) { if ((x += that.x) >= MOD) x -= MOD; return *this; }
15     ModInt &operator-=(ModInt that) { if ((x += MOD - that.x) >= MOD) x -= MOD; return *this; }
16     ModInt &operator*=(ModInt that) { x = (unsigned long long)x * that.x % MOD; return *this; }
17     ModInt &operator/=(ModInt that) { return *this *= that.inverse(); }
18     ModInt operator+(ModInt that) const { return ModInt(*this) += that; }
19     ModInt operator-(ModInt that) const { return ModInt(*this) -= that; }
20     ModInt operator*(ModInt that) const { return ModInt(*this) *= that; }
21     ModInt operator/(ModInt that) const { return ModInt(*this) /= that; }
22     ModInt inverse() const { long long a = x, b = MOD, u = 1, v = 0;
23         while (b) { long long t = a / b; a -= t * b; std::swap(a, b); u -= t * v; std::swap(u, v); }
24         return ModInt(u); }
25     bool operator==(ModInt that) const { return x == that.x; }
26     bool operator!=(ModInt that) const { return x != that.x; }
27     ModInt operator-() const { ModInt t; t.x = x == 0 ? 0 : Mod - x; return t; }
28 };
29 template<int MOD> ostream& operator<<(ostream& st, const ModInt<MOD> a) { st << a.get(); return st; };
30 template<int MOD> ModInt<MOD> operator^(ModInt<MOD> a, unsigned long long k) {
31     ModInt<MOD> r = 1; while (k) { if (k & 1) r *= a; a *= a; k >>= 1; } return r; }
32 template<typename T, int FAC_MAX> struct Comb { vector<T> fac, ifac;
33     Comb(){fac.resize(FAC_MAX,1);ifac.resize(FAC_MAX,1);rep(i,1,FAC_MAX)fac[i]=fac[i-1]*i;
34     ifac[FAC_MAX-1]=T(1)/fac[FAC_MAX-1];rrep(i,FAC_MAX-2,1)ifac[i]=ifac[i+1]*T(i+1);}
35     T aPb(int a, int b) { if (b < 0 || a < b) return T(0); return fac[a] * ifac[a - b]; }
36     T aCb(int a, int b) { if (b < 0 || a < b) return T(0); return fac[a] * ifac[a - b] * ifac[b]; }
37     T nHk(int n, int k) { if (n == 0 && k == 0) return T(1); if (n <= 0 || k < 0) return 0;
38     return aCb(n + k - 1, k); } // nHk = (n+k-1)Ck : n is separator
39     T pairCombination(int n) {if(n%2==1)return T(0);return fac[n]*ifac[n/2]/(T(2)^(n/2));}
40     // combination of paris for n

```

```

41 };
42 typedef ModInt<998244353> mint;
43 int main(){
44     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
45     #ifndef ONLINE_JUDGE
46     freopen("in.txt", "r", stdin);
47     freopen("out.txt", "w", stdout);
48     freopen("error.txt", "w", stderr);
49     #endif
50     const int N = 1e5;
51     Comb<mint , N> com;

```

NTT_prime_mod.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4
5  // modint
6  template<int MOD> struct Fp {
7      // inner value
8      long long val;
9
10     // constructor
11     constexpr Fp() : val(0) {}
12     constexpr Fp(long long v) : val(v % MOD) {
13         if (val < 0) val += MOD;
14     }
15     constexpr long long get() const { return val; }
16     constexpr int get_mod() const { return MOD; }
17
18     // arithmetic operators
19     constexpr Fp operator + () const { return Fp(*this); }
20     constexpr Fp operator - () const { return Fp(0) - Fp(*this); }
21     constexpr Fp operator + (const Fp &r) const { return Fp(*this) += r; }
22     constexpr Fp operator - (const Fp &r) const { return Fp(*this) -= r; }
23     constexpr Fp operator * (const Fp &r) const { return Fp(*this) *= r; }
24     constexpr Fp operator / (const Fp &r) const { return Fp(*this) /= r; }
25     constexpr Fp& operator += (const Fp &r) {
26         val += r.val;
27         if (val >= MOD) val -= MOD;
28         return *this;

```

```

29     }
30     constexpr Fp& operator -= (const Fp &r) {
31         val -= r.val;
32         if (val < 0) val += MOD;
33         return *this;
34     }
35     constexpr Fp& operator *= (const Fp &r) {
36         val = val * r.val % MOD;
37         return *this;
38     }
39     constexpr Fp& operator /= (const Fp &r) {
40         long long a = r.val, b = MOD, u = 1, v = 0;
41         while (b) {
42             long long t = a / b;
43             a -= t * b, swap(a, b);
44             u -= t * v, swap(u, v);
45         }
46         val = val * u % MOD;
47         if (val < 0) val += MOD;
48         return *this;
49     }
50     constexpr Fp pow(long long n) const {
51         Fp res(1), mul(*this);
52         while (n > 0) {
53             if (n & 1) res *= mul;
54             mul *= mul;
55             n >>= 1;
56         }
57         return res;
58     }
59     constexpr Fp inv() const {
60         Fp res(1), div(*this);
61         return res / div;
62     }
63
64     // other operators
65     constexpr bool operator == (const Fp &r) const {
66         return this->val == r.val;
67     }
68     constexpr bool operator != (const Fp &r) const {
69         return this->val != r.val;
70     }
71     constexpr Fp& operator ++ () {
72         ++val;

```

```

73     if (val >= MOD) val -= MOD;
74     return *this;
75 }
76 constexpr Fp& operator -- () {
77     if (val == 0) val += MOD;
78     --val;
79     return *this;
80 }
81 constexpr Fp operator ++ (int) const {
82     Fp res = *this;
83     ++*this;
84     return res;
85 }
86 constexpr Fp operator -- (int) const {
87     Fp res = *this;
88     --*this;
89     return res;
90 }
91 friend constexpr istream& operator >> (istream &is, Fp<MOD> &x) {
92     is >> x.val;
93     x.val %= MOD;
94     if (x.val < 0) x.val += MOD;
95     return is;
96 }
97 friend constexpr ostream& operator << (ostream &os, const Fp<MOD> &x) {
98     return os << x.val;
99 }
100 friend constexpr Fp<MOD> pow(const Fp<MOD> &r, long long n) {
101     return r.pow(n);
102 }
103 friend constexpr Fp<MOD> inv(const Fp<MOD> &r) {
104     return r.inv();
105 }
106 };
107
108 namespace NTT {
109     long long modpow(long long a, long long n, int mod) {
110         long long res = 1;
111         while (n > 0) {
112             if (n & 1) res = res * a % mod;
113             a = a * a % mod;
114             n >>= 1;
115         }

```

```

116     return res;
117 }
118
119 long long modinv(long long a, int mod) {
120     long long b = mod, u = 1, v = 0;
121     while (b) {
122         long long t = a / b;
123         a -= t * b, swap(a, b);
124         u -= t * v, swap(u, v);
125     }
126     u %= mod;
127     if (u < 0) u += mod;
128     return u;
129 }
130
131 int calc_primitive_root(int mod) {
132     if (mod == 2) return 1;
133     if (mod == 167772161) return 3;
134     if (mod == 469762049) return 3;
135     if (mod == 754974721) return 11;
136     if (mod == 998244353) return 3;
137     int divs[20] = {};
138     divs[0] = 2;
139     int cnt = 1;
140     long long x = (mod - 1) / 2;
141     while (x % 2 == 0) x /= 2;
142     for (long long i = 3; i * i <= x; i += 2) {
143         if (x % i == 0) {
144             divs[cnt++] = i;
145             while (x % i == 0) x /= i;
146         }
147     }
148     if (x > 1) divs[cnt++] = x;
149     for (int g = 2;; g++) {
150         bool ok = true;
151         for (int i = 0; i < cnt; i++) {
152             if (modpow(g, (mod - 1) / divs[i], mod) == 1) {
153                 ok = false;
154                 break;
155             }
156         }
157         if (ok) return g;
158     }

```



```

159     }
160
161     int get_fft_size(int N, int M) {
162         int size_a = 1, size_b = 1;
163         while (size_a < N) size_a <<= 1;
164         while (size_b < M) size_b <<= 1;
165         return max(size_a, size_b) << 1;
166     }
167
168     // number-theoretic transform
169     template<class mint> void trans(vector<mint> &v, bool inv = false) {
170         if (v.empty()) return;
171         int N = (int)v.size();
172         int MOD = v[0].get_mod();
173         int PR = calc_primitive_root(MOD);
174         static bool first = true;
175         static vector<long long> vbw(30), vibw(30);
176         if (first) {
177             first = false;
178             for (int k = 0; k < 30; ++k) {
179                 vbw[k] = modpow(PR, (MOD - 1) >> (k + 1), MOD);
180                 vibw[k] = modinv(vbw[k], MOD);
181             }
182         }
183         for (int i = 0, j = 1; j < N - 1; j++) {
184             for (int k = N >> 1; k > (i ^ k); k >>= 1);
185             if (i > j) swap(v[i], v[j]);
186         }
187         for (int k = 0, t = 2; t <= N; ++k, t <<= 1) {
188             long long bw = vbw[k];
189             if (inv) bw = vibw[k];
190             for (int i = 0; i < N; i += t) {
191                 mint w = 1;
192                 for (int j = 0; j < t/2; ++j) {
193                     int j1 = i + j, j2 = i + j + t/2;
194                     mint c1 = v[j1], c2 = v[j2] * w;
195                     v[j1] = c1 + c2;
196                     v[j2] = c1 - c2;
197                     w *= bw;
198                 }
199             }
200         }
201         if (inv) {

```

```

202     long long invN = modinv(N, MOD);
203     for (int i = 0; i < N; ++i) v[i] = v[i] * invN;
204 }
205 }
206
207 // for garner
208 static constexpr int MOD0 = 754974721;
209 static constexpr int MOD1 = 167772161;
210 static constexpr int MOD2 = 469762049;
211 using mint0 = Fp<MOD0>;
212 using mint1 = Fp<MOD1>;
213 using mint2 = Fp<MOD2>;
214 static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
215 static const mint2 imod1 = 104391568; // modinv(MOD1, MOD2);
216 static const mint2 imod01 = 187290749; // imod1 / MOD0;
217
218 // small case (T = mint, long long)
219 template<class T> vector<T> naive_mul(const vector<T> &A, const vector<T> &B) {
220     if (A.empty() || B.empty()) return {};
221     int N = (int)A.size(), M = (int)B.size();
222     vector<T> res(N + M - 1);
223     for (int i = 0; i < N; ++i)
224         for (int j = 0; j < M; ++j)
225             res[i + j] += A[i] * B[j];
226     return res;
227 }
228
229 // mul by convolution
230 template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint>
&B) {
231     if (A.empty() || B.empty()) return {};
232     int N = (int)A.size(), M = (int)B.size();
233     if (min(N, M) < 30) return naive_mul(A, B);
234     int MOD = A[0].get_mod();
235     int size_fft = get_fft_size(N, M);
236     if (MOD == 998244353) {
237         vector<mint> a(size_fft), b(size_fft), c(size_fft);
238         for (int i = 0; i < N; ++i) a[i] = A[i];
239         for (int i = 0; i < M; ++i) b[i] = B[i];
240         trans(a), trans(b);
241         vector<mint> res(size_fft);
242         for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];

```

```

243     trans(res, true);
244     res.resize(N + M - 1);
245     return res;
246 }
247 vector<mint0> a0(size_fft, 0), b0(size_fft, 0), c0(size_fft, 0);
248 vector<mint1> a1(size_fft, 0), b1(size_fft, 0), c1(size_fft, 0);
249 vector<mint2> a2(size_fft, 0), b2(size_fft, 0), c2(size_fft, 0);
250 for (int i = 0; i < N; ++i)
251     a0[i] = A[i].val, a1[i] = A[i].val, a2[i] = A[i].val;
252 for (int i = 0; i < M; ++i)
253     b0[i] = B[i].val, b1[i] = B[i].val, b2[i] = B[i].val;
254 trans(a0), trans(a1), trans(a2), trans(b0), trans(b1), trans(b2);
255 for (int i = 0; i < size_fft; ++i) {
256     c0[i] = a0[i] * b0[i];
257     c1[i] = a1[i] * b1[i];
258     c2[i] = a2[i] * b2[i];
259 }
260 trans(c0, true), trans(c1, true), trans(c2, true);
261 mint mod0 = MOD0, mod01 = mod0 * MOD1;
262 vector<mint> res(N + M - 1);
263 for (int i = 0; i < N + M - 1; ++i) {
264     int y0 = c0[i].val;
265     int y1 = (imod0 * (c1[i] - y0)).val;
266     int y2 = (imod01 * (c2[i] - y0) - imod1 * y1).val;
267     res[i] = mod01 * y2 + mod0 * y1 + y0;
268 }
269 return res;
270 }
271 };
272
273 // Binomial coefficient
274 template<class T> struct BiCoef {
275     vector<T> fact_, inv_, finv_;
276     constexpr BiCoef() {}
277     constexpr BiCoef(int n) noexcept : fact_(n, 1), inv_(n, 1), finv_(n, 1) {
278         init(n);
279     }
280     constexpr void init(int n) noexcept {
281         fact_.assign(n, 1), inv_.assign(n, 1), finv_.assign(n, 1);
282         int MOD = fact_[0].getmod();
283         for (int i = 2; i < n; i++) {
284             fact_[i] = fact_[i-1] * i;

```

```

285     inv_[i] = -inv_[MOD%i] * (MOD/i);
286     finv_[i] = finv_[i-1] * inv_[i];
287 }
288 }
289 constexpr T com(int n, int k) const noexcept {
290     if (n < k || n < 0 || k < 0) return 0;
291     return fact_[n] * finv_[k] * finv_[n-k];
292 }
293 constexpr T fact(int n) const noexcept {
294     if (n < 0) return 0;
295     return fact_[n];
296 }
297 constexpr T inv(int n) const noexcept {
298     if (n < 0) return 0;
299     return inv_[n];
300 }
301 constexpr T finv(int n) const noexcept {
302     if (n < 0) return 0;
303     return finv_[n];
304 }
305 };
306
307
308
309 //-----//
310 // Examples
311 //-----//
312
313 const int MOD = 998244353;
314 using mint = Fp<MOD>;
315
316 int main() {
317     int N;
318     cin >> N;
319     map<int, long long> ma;
320     for (int i = 0; i < N*2; ++i) {
321         int h;
322         cin >> h;
323         ma[h]++;
324     }
325     BiCoef<mint> bc(N*2+1);
326

```

```

327     priority_queue<pair<int,vector<mint>>, vector<pair<int,vector<mint>>>,
greater<pair<int,vector<mint>>>> que;
328     for (auto it : ma) {
329         int n = it.second;
330         vector<mint> pol(n/2+1, 1);
331         for (int i = 0; i <= n/2; ++i) {
332             pol[i] = bc.fact(n) * bc.finv(n - i*2) * bc.finv(i) / modpow(mint(2), i);
333         }
334         que.push({pol.size(), pol});
335     }
336     while (que.size() >= 2) {
337         auto f = que.top().second; que.pop();
338         auto g = que.top().second; que.pop();
339         auto h = NTT::mul(f, g);

```

NTT_random_mod.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4
5  // modint
6  template<int MOD> struct Fp {
7      // inner value
8      long long val;
9
10     // constructor
11     constexpr Fp() : val(0) {}
12     constexpr Fp(long long v) : val(v % MOD) {
13         if (val < 0) val += MOD;
14     }
15     constexpr long long get() const { return val; }
16     constexpr int get_mod() const { return MOD; }
17
18     // arithmetic operators
19     constexpr Fp operator + () const { return Fp(*this); }
20     constexpr Fp operator - () const { return Fp(0) - Fp(*this); }
21     constexpr Fp operator + (const Fp &r) const { return Fp(*this) += r; }
22     constexpr Fp operator - (const Fp &r) const { return Fp(*this) -= r; }
23     constexpr Fp operator * (const Fp &r) const { return Fp(*this) *= r; }

```

```

24  constexpr Fp operator / (const Fp &r) const { return Fp(*this) /= r; }
25  constexpr Fp& operator += (const Fp &r) {
26      val += r.val;
27      if (val >= MOD) val -= MOD;
28      return *this;
29  }
30  constexpr Fp& operator -= (const Fp &r) {
31      val -= r.val;
32      if (val < 0) val += MOD;
33      return *this;
34  }
35  constexpr Fp& operator *= (const Fp &r) {
36      val = val * r.val % MOD;
37      return *this;
38  }
39  constexpr Fp& operator /= (const Fp &r) {
40      long long a = r.val, b = MOD, u = 1, v = 0;
41      while (b) {
42          long long t = a / b;
43          a -= t * b, swap(a, b);
44          u -= t * v, swap(u, v);
45      }
46      val = val * u % MOD;
47      if (val < 0) val += MOD;
48      return *this;
49  }
50  constexpr Fp pow(long long n) const {
51      Fp res(1), mul(*this);
52      while (n > 0) {
53          if (n & 1) res *= mul;
54          mul *= mul;
55          n >>= 1;
56      }
57      return res;
58  }
59  constexpr Fp inv() const {
60      Fp res(1), div(*this);
61      return res / div;
62  }
63
64  // other operators
65  constexpr bool operator == (const Fp &r) const {
66      return this->val == r.val;
67  }

```

```

68     constexpr bool operator != (const Fp &r) const {
69         return this->val != r.val;
70     }
71     constexpr Fp& operator ++ () {
72         ++val;
73         if (val >= MOD) val -= MOD;
74         return *this;
75     }
76     constexpr Fp& operator -- () {
77         if (val == 0) val += MOD;
78         --val;
79         return *this;
80     }
81     constexpr Fp operator ++ (int) const {
82         Fp res = *this;
83         ++*this;
84         return res;
85     }
86     constexpr Fp operator -- (int) const {
87         Fp res = *this;
88         --*this;
89         return res;
90     }
91     friend constexpr istream& operator >> (istream &is, Fp<MOD> &x) {
92         is >> x.val;
93         x.val %= MOD;
94         if (x.val < 0) x.val += MOD;
95         return is;
96     }
97     friend constexpr ostream& operator << (ostream &os, const Fp<MOD> &x) {
98         return os << x.val;
99     }
100    friend constexpr Fp<MOD> pow(const Fp<MOD> &r, long long n) {
101        return r.pow(n);
102    }
103    friend constexpr Fp<MOD> inv(const Fp<MOD> &r) {
104        return r.inv();
105    }
106 };
107
108 namespace NTT {
109     long long modpow(long long a, long long n, int mod) {
110         long long res = 1;

```

```

111     while (n > 0) {
112         if (n & 1) res = res * a % mod;
113         a = a * a % mod;
114         n >>= 1;
115     }
116     return res;
117 }
118
119 long long modinv(long long a, int mod) {
120     long long b = mod, u = 1, v = 0;
121     while (b) {
122         long long t = a / b;
123         a -= t * b, swap(a, b);
124         u -= t * v, swap(u, v);
125     }
126     u %= mod;
127     if (u < 0) u += mod;
128     return u;
129 }
130
131 int calc_primitive_root(int mod) {
132     if (mod == 2) return 1;
133     if (mod == 167772161) return 3;
134     if (mod == 469762049) return 3;
135     if (mod == 754974721) return 11;
136     if (mod == 998244353) return 3;
137     int divs[20] = {};
138     divs[0] = 2;
139     int cnt = 1;
140     long long x = (mod - 1) / 2;
141     while (x % 2 == 0) x /= 2;
142     for (long long i = 3; i * i <= x; i += 2) {
143         if (x % i == 0) {
144             divs[cnt++] = i;
145             while (x % i == 0) x /= i;
146         }
147     }
148     if (x > 1) divs[cnt++] = x;
149     for (int g = 2;; g++) {
150         bool ok = true;
151         for (int i = 0; i < cnt; i++) {
152             if (modpow(g, (mod - 1) / divs[i], mod) == 1) {
153                 ok = false;

```



```

154         break;
155     }
156 }
157 if(ok) return g;
158 }
159 }
160
161 int get_fft_size(int N, int M) {
162     int size_a = 1, size_b = 1;
163     while (size_a < N) size_a <<= 1;
164     while (size_b < M) size_b <<= 1;
165     return max(size_a, size_b) << 1;
166 }
167
168 // number-theoretic transform
169 template<class mint> void trans(vector<mint> &v, bool inv = false) {
170     if (v.empty()) return;
171     int N = (int)v.size();
172     int MOD = v[0].get_mod();
173     int PR = calc_primitive_root(MOD);
174     static bool first = true;
175     static vector<long long> vbw(30), vibw(30);
176     if (first) {
177         first = false;
178         for (int k = 0; k < 30; ++k) {
179             vbw[k] = modpow(PR, (MOD - 1) >> (k + 1), MOD);
180             vibw[k] = modinv(vbw[k], MOD);
181         }
182     }
183     for (int i = 0, j = 1; j < N - 1; j++) {
184         for (int k = N >> 1; k > (i ^ k); k >>= 1);
185         if (i > j) swap(v[i], v[j]);
186     }
187     for (int k = 0, t = 2; t <= N; ++k, t <<= 1) {
188         long long bw = vbw[k];
189         if (inv) bw = vibw[k];
190         for (int i = 0; i < N; i += t) {
191             mint w = 1;
192             for (int j = 0; j < t/2; ++j) {
193                 int j1 = i + j, j2 = i + j + t/2;
194                 mint c1 = v[j1], c2 = v[j2] * w;
195                 v[j1] = c1 + c2;
196                 v[j2] = c1 - c2;

```

```

197         w *= bw;
198     }
199 }
200 }
201 if (inv) {
202     long long invN = modinv(N, MOD);
203     for (int i = 0; i < N; ++i) v[i] = v[i] * invN;
204 }
205 }
206
207 // for Garner
208 static constexpr int MOD0 = 754974721;
209 static constexpr int MOD1 = 167772161;
210 static constexpr int MOD2 = 469762049;
211 using mint0 = Fp<MOD0>;
212 using mint1 = Fp<MOD1>;
213 using mint2 = Fp<MOD2>;
214 static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
215 static const mint2 imod1 = 104391568; // modinv(MOD1, MOD2);
216 static const mint2 imod01 = 187290749; // imod1 / MOD0;
217
218 // small case (T = mint, long long)
219 template<class T> vector<T> naive_mul(const vector<T> &A, const vector<T> &B) {
220     if (A.empty() || B.empty()) return {};
221     int N = (int)A.size(), M = (int)B.size();
222     vector<T> res(N + M - 1);
223     for (int i = 0; i < N; ++i)
224         for (int j = 0; j < M; ++j)
225             res[i + j] += A[i] * B[j];
226     return res;
227 }
228
229 // mul by convolution
230 template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint>
&B) {
231     if (A.empty() || B.empty()) return {};
232     int N = (int)A.size(), M = (int)B.size();
233     if (min(N, M) < 30) return naive_mul(A, B);
234     int MOD = A[0].get_mod();
235     int size_fft = get_fft_size(N, M);
236     if (MOD == 998244353) {
237         vector<mint> a(size_fft), b(size_fft), c(size_fft);

```

```

238     for (int i = 0; i < N; ++i) a[i] = A[i];
239     for (int i = 0; i < M; ++i) b[i] = B[i];
240     trans(a), trans(b);
241     vector<mint> res(size_fft);
242     for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];
243     trans(res, true);
244     res.resize(N + M - 1);
245     return res;
246 }
247 vector<mint0> a0(size_fft, 0), b0(size_fft, 0), c0(size_fft, 0);
248 vector<mint1> a1(size_fft, 0), b1(size_fft, 0), c1(size_fft, 0);
249 vector<mint2> a2(size_fft, 0), b2(size_fft, 0), c2(size_fft, 0);
250 for (int i = 0; i < N; ++i)
251     a0[i] = A[i].val, a1[i] = A[i].val, a2[i] = A[i].val;
252 for (int i = 0; i < M; ++i)
253     b0[i] = B[i].val, b1[i] = B[i].val, b2[i] = B[i].val;
254 trans(a0), trans(a1), trans(a2), trans(b0), trans(b1), trans(b2);
255 for (int i = 0; i < size_fft; ++i) {
256     c0[i] = a0[i] * b0[i];
257     c1[i] = a1[i] * b1[i];
258     c2[i] = a2[i] * b2[i];
259 }
260 trans(c0, true), trans(c1, true), trans(c2, true);
261 mint mod0 = MOD0, mod01 = mod0 * MOD1;
262 vector<mint> res(N + M - 1);
263 for (int i = 0; i < N + M - 1; ++i) {
264     int y0 = c0[i].val;
265     int y1 = (imod0 * (c1[i] - y0)).val;
266     int y2 = (imod01 * (c2[i] - y0) - imod1 * y1).val;
267     res[i] = mod01 * y2 + mod0 * y1 + y0;
268 }
269 return res;
270 }
271 };
272
273
274
275 //-----//
276 // Examples
277 //-----//
278
279 void Yosupo_Convolution_mod_1000000007() {

```

```

280     const int MOD = 1000000007;
281     using mint = Fp<MOD>;
282
283     int N, M;
284     cin >> N >> M;
285     vector<mint> a(N), b(M);
286     for (int i = 0; i < N; ++i) cin >> a[i];
287     for (int i = 0; i < M; ++i) cin >> b[i];

```

check_point_inside_polygon.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  typedef __int128 lll;
4  typedef long double ld;
5  typedef __float128 lld;
6  using namespace std;
7
8  ld pi = acos(-1);
9  ld epsilon = 1e-9;
10
11 struct vec2 {
12     ld x, y;
13     vec2() {this->x = 0; this->y = 0;}
14     vec2(ld x, ld y) {this->x = x; this->y = y;}
15 };
16
17 vec2 add(vec2 a, vec2 b){
18     vec2 ret;
19     ret.x = a.x + b.x;
20     ret.y = a.y + b.y;
21     return ret;
22 }
23
24 vec2 sub(vec2 a, vec2 b) {
25     vec2 ret;
26     ret.x = a.x - b.x;
27     ret.y = a.y - b.y;
28     return ret;
29 }
30

```

```
31  ld cross(vec2 a, vec2 b) {
32      return a.x * b.y - a.y * b.x;
33  }
34
35  ld dot(vec2 a, vec2 b) {
36      return a.x * b.x + a.y * b.y;
37  }
38
39  ld length(vec2 a) {
40      return sqrt(a.x * a.x + a.y * a.y);
41  }
42
43  ld lerp(ld t0, ld t1, ld x0, ld x1, ld t) {
44      ld slope = (x1 - x0) / (t1 - t0);
45      return x0 + slope * (t - t0);
46  }
47
48  vec2 mul(vec2 a, ld s) {
49      a.x *= s;
50      a.y *= s;
51      return a;
52  }
53
54  vec2 normalize(vec2 a){
55      ld len = length(a);
56      vec2 ret;
57      ret.x = a.x / len;
58      ret.y = a.y / len;
59      return ret;
60  }
61
62  //angle from the +x axis in range (-pi, pi)
63  ld polar_angle(vec2 a) {
64      return atan2(a.y, a.x);
65  }
66
67  //project a onto b
68  vec2 project(vec2 a, vec2 b) {
69      b = normalize(b);
70      ld proj_mag = dot(a, b);
71      return mul(b, proj_mag);
72  }
73
74  vec2 rotateCCW(vec2 a, ld theta) {
```

```

75     vec2 ret(0, 0);
76     ret.x = a.x * cos(theta) - a.y * sin(theta);
77     ret.y = a.x * sin(theta) + a.y * cos(theta);
78     return ret;
79 }
80
81 //returns the coefficients s and t, where p1 + v1 * s = p2 + v2 * t
82 vector<ld> lineLineIntersect(vec2 p1, vec2 v1, vec2 p2, vec2 v2) {
83     if(cross(v1, v2) == 0){
84         return {};
85     }
86     ld s = cross(sub(p2, p1), v2) / cross(v1, v2);
87     ld t = cross(sub(p1, p2), v1) / cross(v2, v1);
88     return {s, t};
89 }
90
91 ld tri_area(vec2 t1, vec2 t2, vec2 t3) {
92     vec2 v1 = sub(t1, t2);
93     vec2 v2 = sub(t2, t3);
94     return abs(cross(v1, v2) / 2.0);
95 }
96
97 //returns the distance along the ray from ray_a to the nearest point on the circle.
98 ld rayCircleIntersect(vec2 ray_a, vec2 ray_b, vec2 center, ld radius) {
99     vec2 ray_dir = normalize(sub(ray_b, ray_a));
100    vec2 to_center = sub(center, ray_a);
101    vec2 center_proj = add(ray_a, mul(ray_dir, dot(ray_dir, to_center)));
102    ld center_proj_len = length(sub(center, center_proj));
103    //radius^2 = center_proj_len^2 + int_depth^2
104    //int_depth = sqrt(radius^2 - center_proj_len^2)
105    ld int_depth = sqrt(radius * radius - center_proj_len * center_proj_len);
106    return dot(ray_dir, to_center) - int_depth;
107 }
108
109 //sector area of circle
110 ld sector_area(ld theta, ld radius) {
111     return radius * radius * pi * ((theta) / (2.0 * pi));
112 }
113
114 ld chord_area(ld theta, ld radius) {
115     ld sector = sector_area(theta, radius);
116     ld tri_area = radius * radius * cos(theta) * sin(theta);
117     return sector - tri_area;

```

```

118 }
119
120 //dist = distance from center
121 ld chord_area_dist(ld dist, ld radius) {
122     ld theta = acos(dist / radius);
123     return chord_area(theta, radius);
124 }
125
126 //length of chord
127 ld chord_area_length(ld length, ld radius) {
128     ld theta = asin((length / 2.0) / radius);
129     return chord_area(theta, radius);
130 }
131
132 //given a point inside and outside a circle, find the point along the line that intersects
the circle.
133 vec2 find_circle_intersect(vec2 in, vec2 out, vec2 c_center, ld c_radius) {
134     //just binary search :D
135     //i think we can reduce this to some sort of quadratic.
136     ld low = 0;
137     ld high = 1;
138     ld len = length(sub(in, out));
139     vec2 norm = normalize(sub(out, in));
140     while(abs(high - low) > epsilon) {
141         ld mid = (high + low) / 2.0;
142         vec2 mid_pt = add(in, mul(norm, len * mid));
143         ld mid_dist = length(sub(mid_pt, c_center));
144         if(mid_dist < c_radius) {
145             low = mid;
146         }
147         else {
148             high = mid;
149         }
150     }
151     return add(in, mul(norm, len * low));
152 }
153
154 //returns the area of the polygon.
155 //winding direction doesn't matter
156 //polygon can be self intersecting i think...
157 ld polygon_area(vector<vec2>& poly) {
158     ld area = 0;
159     for(int i = 0; i < poly.size(); i++){

```

```

160     vec2 v0 = poly[i];
161     vec2 v1 = poly[(i + 1) % poly.size()];
162     area += cross(v0, v1);
163 }
164 return abs(area / 2.0);
165 }
166
167 //assuming that the density of the polygon is uniform, the centroid is the center of
    mass.
168 //winding direction matters...
169 vec2 polygon_centroid(vector<vec2>& poly) {
170     vec2 c = vec2();
171     for(int i = 0; i < poly.size(); i++){
172         vec2 v0 = poly[i];
173         vec2 v1 = poly[(i + 1) % poly.size()];
174         ld p = cross(v0, v1);
175         c.x += (v0.x + v1.x) * p;
176         c.y += (v0.y + v1.y) * p;
177     }
178     ld area = polygon_area(poly);
179     c.x /= (6.0 * area);
180     c.y /= (6.0 * area);
181     return c;
182 }
183
184 //i believe this gives in CCW order, have to verify though.
185 vector<vec2> convex_hull(vector<vec2> a, bool include_collinear = false) {
186     function<int(vec2, vec2, vec2)> orientation = [](vec2 a, vec2 b, vec2 c) -> int {
187         ld v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
188         if (v < 0) return -1; // clockwise
189         if (v > 0) return +1; // counter-clockwise
190         return 0;
191     };
192
193     function<bool(vec2, vec2, vec2)> collinear = [&orientation](vec2 a, vec2 b, vec2 c) ->
    bool {
194         return orientation(a, b, c) == 0;
195     };
196
197     function<bool(vec2, vec2, vec2, bool)> cw = [&orientation](vec2 a, vec2 b, vec2 c,
    bool include_collinear) -> bool {
198         int o = orientation(a, b, c);
199         return o < 0 || (include_collinear && o == 0);

```



```

200     };
201
202     vec2 p0 = *min_element(a.begin(), a.end(), [](vec2 a, vec2 b) {
203         return make_pair(a.y, a.x) < make_pair(b.y, b.x);
204     });
205     sort(a.begin(), a.end(), [&p0, &orientation](const vec2& a, const vec2& b) {
206         int o = orientation(p0, a, b);
207         if (o == 0)
208             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
209                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
210         return o < 0;
211     });
212     if (include_collinear) {
213         int i = (int)a.size()-1;
214         while (i >= 0 && collinear(p0, a[i], a.back())) i--;
215         reverse(a.begin()+i+1, a.end());
216     }
217
218     vector<vec2> st;
219     for (int i = 0; i < (int)a.size(); i++) {
220         while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
221             st.pop_back();
222         st.push_back(a[i]);
223     }
224
225     //make sure there are no duplicate vertices
226     vector<vec2> ans(0);
227     for(int i = 0; i < st.size(); i++){
228         vec2 v0 = st[i];
229         vec2 v1 = st[(i + 1) % st.size()];
230         if(v0.x == v1.x && v0.y == v1.y) {
231             continue;
232         }
233         ans.push_back(st[i]);
234     }
235
236     return ans;
237 }
238
239 //checks if the area of the triangle is the same as the three triangle areas formed by
drawing lines from pt to the vertices.
240 //i don't think triangle winding order matters

```

```

241 bool point_inside_triangle(vec2 pt, vec2 t0, vec2 t1, vec2 t2) {
242     ld a1 = abs(cross(sub(t1, t0), sub(t2, t0)));
243     ld a2 = abs(cross(sub(t0, pt), sub(t1, pt))) + abs(cross(sub(t1, pt), sub(t2, pt))) +
abs(cross(sub(t2, pt), sub(t0, pt)));
244     return abs(a1 - a2) < epsilon;
245 }
246
247 //runs in O(n * log(n)) time.
248 //has to do O(n * log(n)) preprocessing, but after preprocessing can answer queries
online in O(log(n))
249 vector<bool> points_inside_convex_hull(vector<vec2>& pts, vector<vec2>& hull) {
250     vector<bool> ans(pts.size(), false);
251
252     //edge case
253     if(hull.size() <= 2){
254         return ans;
255     }
256
257     //find point of hull that has minimum x coordinate
258     //if multiple elements have same x, then minimum y.
259     int pivot_ind = 0;
260     for(int i = 1; i < hull.size(); i++){
261         if(hull[i].x < hull[pivot_ind].x || (hull[i].x == hull[pivot_ind].x && hull[i].y <
hull[pivot_ind].y)) {
262             pivot_ind = i;
263         }
264     }
265
266     //sort all the remaining elements according to polar angle to the pivot
267     vector<vec2> h_pts(0);
268     vec2 pivot = hull[pivot_ind];
269     for(int i = 0; i < hull.size(); i++){
270         if(i != pivot_ind) {
271             h_pts.push_back(hull[i]);
272         }
273     }
274     sort(h_pts.begin(), h_pts.end(), [&pivot](vec2& a, vec2& b) -> bool {
275         return polar_angle(sub(a, pivot)) < polar_angle(sub(b, pivot));
276     });
277
278     //for each point we want to check, compute it's polar angle, then binary search for
the sector that should contain it

```

```

279     for(int i = 0; i < pts.size(); i++){
280         vec2 pt = pts[i];
281         ld pt_ang = polar_angle(sub(pt, pivot));
282         int low = 0;
283         int high = h_pts.size() - 2;
284         int tri_ind = low;
285         while(low <= high) {
286             int mid = low + (high - low) / 2;
287             if(polar_angle(sub(h_pts[mid], pivot)) <= pt_ang) {
288                 tri_ind = max(tri_ind, mid);
289                 low = mid + 1;
290             }
291             else {
292                 high = mid - 1;
293             }
294         }
295         ans[i] = point_inside_triangle(pt, pivot, h_pts[tri_ind], h_pts[tri_ind + 1]);
296     }
297
298     return ans;
299 }
300
301 signed main() {
302     ios_base::sync_with_stdio(false);
303     cin.tie(NULL);
304
305     int n, m;
306     cin >> n >> m;
307     vector<vec2> topping(n), crust(m);
308     for(int i = 0; i < n; i++){
309         cin >> topping[i].x >> topping[i].y;
310     }
311     for(int i = 0; i < m; i++){
312         cin >> crust[i].x >> crust[i].y;
313     }
314     crust = convex_hull(crust);

```

convolution_ntt_optimized.cpp

```

1  const int mod = (119 << 23) + 1;
2  const int N = 1e6 + 1;

```

```

3  int fac[N], inv_fac[N];
4
5  void pre(){
6      fac[0] = fac[1] = 1;
7      inv_fac[0] = inv_fac[1] = 1;
8      for (int i = 2; i < N; ++i) {
9          inv_fac[i] = mod - 1ll * mod / i * inv_fac[mod % i] % mod;
10     }
11     for (int i = 2; i < N; ++i) {
12         fac[i] = 1ll * i * fac[i - 1] % mod;
13         inv_fac[i] = 1ll * inv_fac[i] * inv_fac[i - 1] % mod;
14     }
15 }
16
17 ll fast_power(ll a, ll b){
18     ll res = 1;
19     while (b){
20         if (b & 1) res = res * a % mod;
21         a = a * a % mod;
22         b >>= 1;
23     }
24     return res;
25 }
26
27 ll mod_inv(ll a){
28     return fast_power(a, mod - 2);
29 }
30
31 const int root = fast_power(3, 119);
32 const int root_inv = mod_inv(root);
33 const int root_pw = 23;
34
35 int reverse(int num, int lg_n){
36     int res = 0;
37     for (int i = 0; i < lg_n; ++i) {
38         if (num & (1 << i)) res |= 1 << (lg_n - 1 - i);
39     }
40     return res;
41 }
42
43 vector<vector<int>> roots, roots_inv;
44 vector<int> rev;
45
46 void pre(int n){

```

```

47     int lg_n = __lg(n);
48     rev.resize(n);
49     roots.resize(lg_n);
50     roots_inv.resize(lg_n);
51     for (int i = 0; i < n; ++i) {
52         rev[i] = reverse(i, lg_n);
53     }
54
55     int wlen = root, wlen_inv = root_inv;
56     for (int i = 0; i < root_pw - lg_n; ++i) {
57         wlen = 1ll * wlen * wlen % mod;
58         wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
59     }
60
61     for (int l = lg_n - 1; l >= 0; --l) {
62         int len = 1 << (l + 1);
63         int w = 1, w_inv = 1;
64         roots[l].resize(len / 2);
65         roots_inv[l].resize(len / 2);
66         for (int i = 0; i < len / 2; ++i) {
67             roots[l][i] = w;
68             roots_inv[l][i] = w_inv;
69             w = 1ll * w * wlen % mod;
70             w_inv = 1ll * w_inv * wlen_inv % mod;
71         }
72         wlen = 1ll * wlen * wlen % mod;
73         wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
74     }
75 }
76
77 void ntt(vector<int>& a, bool invert){
78     int n = a.size();
79     for (int i = 1; i < n; ++i) {
80         if (i < rev[i]) swap(a[i], a[rev[i]]);
81     }
82     int mx = __lg(n);
83     for (int l = 0; l < mx; ++l) {
84         int len = 1 << (l + 1), shift = 1 << l;
85         for (int i = 0; i < n; i += len) {
86             for (int j = 0; j < len / 2; ++j) {
87                 int w = invert ? roots_inv[l][j] : roots[l][j];
88                 int u = a[i + j], v = 1ll * a[i + j + shift] * w % mod;
89                 a[i + j] = (u + v) % mod;

```

```

90         a[i + j + shift] = (u - v + mod) % mod;
91     }
92 }
93 }
94
95 if (invert){
96     int n_1 = mod_inv(n);
97     for(int &x: a) x = 1ll * x * n_1 % mod;
98 }
99 }
100
101 vector<int> convolve(int n, int m, vector<int> a){
102     int size = n * m + 1;
103     int _n = 1;
104     while (_n < size) _n <= 1;
105     a.resize(_n);
106     pre(_n);
107     ntt(a, false);
108     for (int i = 0; i < _n; ++i) a[i] = fast_power(a[i], n);
109     ntt(a, true);
110     a.resize(size);

```

convolution_using NTT_kactl.cpp

```

1  /*
2   Convolution two polynomial in  $O(n \log n)$  instead of  $O(n^2)$ 
3   */
4
5  #define vi vector<int>
6  #define rep(x,l,r) for(int x = l; x < r; ++x)
7  #define sz(x) (size(x))
8  const ll mod = (119 << 23) + 1, root = 62; // = 998244353
9  // For  $p < 2^{30}$  there is also e.g.  $5 << 25$ ,  $7 << 26$ ,  $479 << 21$ 
10 // and  $483 << 21$  (same root). The last two are  $> 10^9$ .
11 ll modpow(ll b, ll e) {
12     ll ans = 1;
13     for (; e; b = b * b % mod, e /= 2)
14         if (e & 1) ans = ans * b % mod;
15     return ans;
16 }
17 typedef vector<ll> vl;

```

```

18 void ntt(vl &a) {
19     int n = sz(a), L = 31 - __builtin_clz(n);
20     static vl rt(2, 1);
21     for (static int k = 2, s = 2; k < n; k *= 2, s++) {
22         rt.resize(n);
23         ll z[] = {1, modpow(root, mod >> s)};
24         rep(i, k, 2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
25     }
26     vi rev(n);
27     rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
28     rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
29     for (int k = 1; k < n; k *= 2)
30         for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
31             ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
32             a[i + j + k] = ai - z + (z > ai ? mod : 0);
33             ai += (ai + z >= mod ? z - mod : z);
34         }
35     }
36     vl conv(const vl &a, const vl &b) {
37         if (a.empty() || b.empty()) return {};
38         int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
39             n = 1 << B;
40         int inv = modpow(n, mod - 2);
41         vl L(a), R(b), out(n);
42         L.resize(n), R.resize(n);
43         ntt(L), ntt(R);
44         rep(i, 0, n)
45             out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
46         ntt(out);
47         return {out.begin(), out.begin() + s};
48     }

```

derangement.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  const int N = 1e6;
6  using namespace std;
7  ll D[N]
8  int main(){

```

```

9  /*
10  ----- Derangements-----
11  derangement is a permutation of n elements with no fixed point
12  there is no i such that p_i == i
13  recursive Formula:
14  1.
15   D(n)=n*D(n-1)+(-1)^n
16  2.
17   D(n)=(n-1)(D(n-1)+D(n-2))
18  Find more information : https://brilliant.org/wiki/derangements/
19  */
20  D[2] = 1; // Base-case
21  for(int i = 3; i < N; i++){
22      D[i] = i * D[i - 1] + pow(-1, i);
23  }
24  }

```

evaluating_polynomial_at_n_points.cpp

```

1  //evalating a polynomial at a number of points under a mod, the length of the
   polynomial is a power of 2
2  #include <bits/stdc++.h>
3  typedef long long ll;
4  using namespace std;
5
6  int n;
7  ll m, q, a[1 << 20], q_pow[1 << 20];
8
9  vector<ll> dft(int k = 1, int idx = 0) {
10     if (k == n) return {a[idx]};
11     else {
12         vector<ll> even = dft(k * 2, idx);
13         vector<ll> odd = dft(k * 2, idx | k);
14
15         int mid = n / k / 2;
16         vector<ll> ans;
17         for (int i = 0; i < 2 * mid; i++)
18             ans.push_back((even[i % mid] + q_pow[k * i] * odd[i % mid] % m) %
19                             m);
20         return ans;
21     }
22 }

```



```

23
24 int main() {
25     cin.tie(0)->sync_with_stdio(0);
26     cin >> n >> m >> q;
27     for (int i = 0; i < n; i++) cin >> a[i];
28
29     q_pow[0] = 1;
30     for (int i = 1; i < n; i++) q_pow[i] = q * q_pow[i - 1] % m;
31
32     vector<ll> ans = dff();
33     ll tot = 0;
34     for (ll i : ans) tot = (tot + i) % m;
35     cout << tot << '\n';
36     for (int i = 1; i < n; i++) cout << ans[i] << ' ';
37     cout << ans[0];
38     return 0;
39 }

```

lucas.cpp

```

1  const int N = 2e6 + 100;
2  const int mod = 1e6 + 3;
3  ll fact[N];
4  ll inv[N]; //mod inverse for i
5  ll invfact[N]; //mod inverse for i!
6  void init() {
7      fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
8      for (long long i = 2; i < N; i++) {
9          fact[i] = (fact[i - 1] * i) % mod;
10         inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
11         invfact[i] = (inv[i] * invfact[i - 1]) % mod;
12     }
13 }
14 ll nCr(int n, int r) {
15     if (r > n || n < 0 || r < 0) return 0; // manual handling
16     return (((fact[n] * invfact[r]) % mod) * invfact[n - r]) % mod;
17 }
18 int lucas(int n, int r) {
19     if (r == 0) return 1;
20     int res = 1;
21     while (r) {
22         res = 1LL * res * nCr(n % mod, r % mod) % mod;

```

```

23     n /= mod; r /= mod;
24 }
25 return res;
26 }

```

nCr.cpp

```

1  ll bpow(ll n, ll x) { return !x ? 1 : bpow(n * n % mod, x >> 1) * (x & 1 ? n : 1) % mod; }
2  ll inv(ll b) { return bpow(b, mod - 2); }
3  ll fact[N], factinv[N];
4  void init(){for (int i = 0; i < N; ++i) fact[i] = i ? fact[i - 1] * i % mod : 1, factinv[i] =
    inv(fact[i]);}
5  ll C(ll n, ll k) { return fact[n] * factinv[n - k] % mod * factinv[k] % mod; }

```

nCr_DP.cpp

```

1  const int N = 61;
2  ll nCr[N][N];
3  void gen(){
4      nCr[0][0] = 1;
5      for (int i = 0; i < N; ++i) {
6          nCr[i][0] = nCr[i][i] = 1;
7          for (int j = 1; j < i; ++j) {
8              nCr[i][j] = (nCr[i-1][j] + nCr[i-1][j-1]);
9          }
10     }
11 }

```

nCr_O(r).cpp

```

1  const int N = 5000;
2  const int mod = 1e9 + 7;
3  ll fact[N];
4  ll inv[N]; //mod inverse for i
5  ll invfact[N]; //mod inverse for i!
6  void init() {
7      fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
8      for (long long i = 2; i < N; i++) {
9          fact[i] = (fact[i - 1] * i) % mod;

```

```

10     inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
11     invfact[i] = (inv[i] * invfact[i - 1]) % mod;
12 }
13 }
14
15 ll nCr(int n, int r) {
16     if (r > n) {
17         return 0;
18     }
19     if (r < 0) return 1;
20     ll res = 1;
21     for (int i = 0; i < r; ++i) {
22         res = res * (n - i) % mod;
23     }
24     res = res * invfact[r] % mod;
25     return res;
26 }

```

nCr_mod2.cpp

```

1  int C(int n, int k) {
2      while (n > 0 || k > 0) {
3          if ((k & 1) > (n & 1)) {
4              return 0;
5          }
6          n >>= 1;
7          k >>= 1;
8      }
9      return 1;
10 }

```

ncr_O(n).cpp

```

1  const int N = 5e6 + 100;
2  const int mod = 1e9 + 7;
3  ll fact[N];
4  ll inv[N]; //mod inverse for i
5  ll invfact[N]; //mod inverse for i!
6  void init() {
7      fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
8      for (long long i = 2; i < N; i++) {

```

```

9      fact[i] = (fact[i - 1] * i) % mod;
10     inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
11     invfact[i] = (inv[i] * invfact[i - 1]) % mod;
12 }
13 }
14
15 ll nCr(int n, int r) {
16     if(r > n || n < 0 || r < 0) return 0; // manual handling
17     return (((fact[n] * invfact[r]) % mod) * invfact[n - r]) %
18         mod;
19 }

```

ntt_optimized.cpp

```

1  ll fast_power(ll a, ll b){
2      ll res = 1;
3      while (b){
4          if (b & 1) res = res * a % mod;
5          a = a * a % mod;
6          b >>= 1;
7      }
8      return res;
9  }
10
11 ll mod_inv(ll a){
12     return fast_power(a, mod - 2);
13 }
14
15 const int root = fast_power(3, 119);
16 const int root_inv = mod_inv(root);
17 const int root_pw = 23;
18
19 int reverse(int num, int lg_n){
20     int res = 0;
21     for (int i = 0; i < lg_n; ++i) {
22         if (num & (1 << i)) res |= 1 << (lg_n - 1 - i);
23     }
24     return res;
25 }
26
27 vector<vector<int>> roots, roots_inv;
28 vector<int> rev;

```

```

29
30 void pre(int n){
31     int lg_n = __lg(n);
32     rev.resize(n);
33     roots.resize(lg_n);
34     roots_inv.resize(lg_n);
35     for (int i = 0; i < n; ++i) {
36         rev[i] = reverse(i, lg_n);
37     }
38
39     int wlen = root, wlen_inv = root_inv;
40     for (int i = 0; i < root_pw - lg_n; ++i) {
41         wlen = 1ll * wlen * wlen % mod;
42         wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
43     }
44
45     for (int l = lg_n - 1; l >= 0; --l) {
46         int len = 1 << (l + 1);
47         int w = 1, w_inv = 1;
48         roots[l].resize(len / 2);
49         roots_inv[l].resize(len / 2);
50         for (int i = 0; i < len / 2; ++i) {
51             roots[l][i] = w;
52             roots_inv[l][i] = w_inv;
53             w = 1ll * w * wlen % mod;
54             w_inv = 1ll * w_inv * wlen_inv % mod;
55         }
56         wlen = 1ll * wlen * wlen % mod;
57         wlen_inv = 1ll * wlen_inv * wlen_inv % mod;
58     }
59 }
60
61 void ntT(vector<int>& a, bool invert){
62     int n = a.size();
63     for (int i = 1; i < n; ++i) {
64         if (i < rev[i]) swap(a[i], a[rev[i]]);
65     }
66     int mx = __lg(n);
67     for (int l = 0; l < mx; ++l) {
68         int len = 1 << (l + 1), shift = 1 << l;
69         for (int i = 0; i < n; i += len) {
70             for (int j = 0; j < len / 2; ++j) {
71                 int w = invert ? roots_inv[l][j] : roots[l][j];
72                 int u = a[i + j], v = 1ll * a[i + j + shift] * w % mod;

```

```

73     a[i + j] = (u + v) % mod;
74     a[i + j + shift] = (u - v + mod) % mod;
75 }
76 }
77 }
78
79 if (invert){
80     int n_1 = mod_inv(n);
81     for(int &x: a) x = 1ll * x * n_1 % mod;
82 }
83 }
84
85 vector<int> convolve(int n, int m, vector<int> a){
86     int size = n * m + 1;
87     int _n = 1;
88     while (_n < size) _n <= 1;
89     a.resize(_n);
90     pre(_n);
91     ntt(a, false);
92     for (int i = 0; i < _n; ++i) a[i] = fast_power(a[i], n);
93     ntt(a, true);
94     a.resize(size);
95     return a;

```

simpson_integration.cpp

```

1  const int N = 1000 * 1000; // number of steps (already multiplied by 2)
2
3  double simpson_integration(double a, double b){
4      double h = (b - a) / N;
5      double s = f(a) + f(b); // a = x_0 and b = x_2n
6      for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's formula
7          double x = a + h * i;
8          s += f(x) * ((i & 1) ? 4 : 2);
9      }
10     s *= h / 3;
11     return s;
12 }

```

startbars.cpp

```

1  const int mod = 1e9 + 7;
2  const int N = 5e6 + 9;
3  ll fac[N], inv[N];
4  void preprocess() {
5      for (int i = 0; i < N; i++) {
6          if (i < 2) {
7              fac[i] = inv[i] = 1;
8          } else {
9              fac[i] = 1ll * i * fac[i - 1] % mod;
10             inv[i] = mod - 1ll * mod / i * inv[mod % i] % mod;
11         }
12     }
13     for (int i = 2; i < N; i++) {
14         inv[i] = 1ll * inv[i] * inv[i - 1] % mod;
15     }
16 }
17 ll ncr(ll n, ll r) {
18     if (r > n || n < 0 || r < 0) return 0; // manual handling
19     return 1ll * fac[n] * (1ll * inv[r] * inv[n - r] % mod) % mod;
20 }
21 ll starsBars(ll stars, ll boxes) {
22     if (boxes == 0) return stars == 0;
23     return ncr(stars - 1, boxes - 1);
24 }

```

stirling.cpp

```

1  /*
2      #Stirling numbe
3      1. S(r, n), represents the number of ways that we can arrange r objects around
        indistinguishable circles of length n,
4      and every circle n must have at least one object around it.
5      2. S(n,k) as the different ways to cut n different elements into k undifferentiated non-
        empty subsets. For example, S(5,3) denotes to:25
6
7      S[i][j] = S[i-1][j-1] + s[i-1][j] * (i-1)
8      Time Complexity : O(r * n)
9      Auxiliary Space : O(r * n)
10 */
11 int stirling_number(int n,int k){
12     if(k==0)return n==0;
13     if(n==0)return 0;

```

```

14     return stirling_number(n-1,k-1)+(n-1)* stirling_number(n-1,k);
15
16 }

```

sum_xor_range.cpp

```

1  ll range_xor(ll l, ll r) {
2      ll res = 0;
3      if (l&1) res ^= l++;
4      if (!(r&1)) res ^= r--;
5      if ((r-l+1)/2&1) res ^= 1;
6      return res;
7  }

```

Game Theory

Grundy 01.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**
6   Initially there are n piles.
7   A pile is formed by some cells.
8   Alice starts the game and they alternate turns.
9   In each tern a player can pick any pile and divide it into two unequal piles.
10  If a player cannot do so, he/she loses the game.
11  **/
12
13  #define Size 100005
14
15  int N;
16  int A[105];
17  int DP[10005];
18
19  int call(int cur){
20      if(cur <= 2) return 0;
21      if(DP[cur] != -1) return DP[cur];
22      vector<int> grundy;
23      for(int d1 = 1;d1<=cur/2;d1++){

```



```

24     int d2 = cur-d1;
25     if(d1 != d2){
26         grundy.pb(call(d1) ^ call(d2));
27     }
28 }
29 make_unique(grundy);
30 if(grundy[0] != 0) return DP[cur] = 0;
31 int f = 0;
32 for(int i = 1; i < grundy.size(); i++){
33     if(grundy[i] == f) continue;
34     f++;
35     if(grundy[i] != f) return DP[cur] = f;
36 }
37 return DP[cur] = grundy[grundy.size()-1] + 1;
38 }
39
40 int main(){
41     int nCase;
42     sf("%d", &nCase);
43     mems(DP, -1);
44     for(int cs = 1; cs <= nCase; cs++){
45         sf("%d", &N);
46         int res = 0;
47         for(int i = 0; i < N; i++){
48             sf("%d", &A[i]);
49             res = res ^ call(A[i]);
50         }
51         if(res == 0) pf("Case %d: Bob\n", cs);
52         else pf("Case %d: Alice\n", cs);
53     }
54     return 0;
55 }

```

Grundy 02.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**
6   Initially there are n piles.
7   A pile is formed by some stones.

```

```

8   Alice starts the game and they alternate turns.
9   In each tern a player can pick any pile and remove some stones.
10  At least 1 and at most half of stones on that pile.
11  If a player cannot do so, he/she loses the game.
12  **/
13
14  #define Size 100005
15
16  int N;
17  int A[1005];
18  int DP[10005];
19
20  int call(int cur){
21      if(cur%2 == 0) return cur/2;
22      while(cur%2 != 0) cur/=2;
23      return cur/2;
24  }
25
26  int main(){
27      int nCase;
28      sf("%d",&nCase);
29      mems(DP,-1);
30      for(int cs = 1;cs<=nCase;cs++){
31          sf("%d",&N);
32          int res = 0;
33          for(int i = 0;i<N;i++){
34              sf("%d",&A[i]);
35              res = res ^ call(A[i]);
36          }
37          if(res == 0) pf("Case %d: Bob\n",cs);
38          else pf("Case %d: Alice\n",cs);
39      }
40      return 0;
41  }

```

Grundy String Game.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**

```

```

6    Given a string of dot and X.
7    XXX is winning position.
8    A player can place an X at any dot position.
9    **/
10
11
12    #define Size 100005
13
14    int N;
15    int DP[255];
16    vector<int> res;
17    string s;
18
19    int call(int cur){
20        if(cur <= 0) return DP[cur] = 0;
21        if(DP[cur] != -1) return DP[cur];
22        vector<int> grundy;
23
24        for(int p = 0;p<=cur/2;p++){
25            int d1 = p - 2,d2 = cur - p - 2 - 1;
26            grundy.push_back(call(d1) ^ call(d2));
27        }
28
29        make_unique(grundy);
30        if(grundy.size() == 0 || grundy[0] != 0) return DP[cur] = 0;
31        int f = 0;
32        for(int i = 1;i<grundy.size();i++){
33            if(grundy[i] == f) continue;
34            f++;
35            if(grundy[i] != f) return DP[cur] = f;
36        }
37        return DP[cur] = grundy[grundy.size()-1] + 1;
38    }
39
40    bool isPossible(int pos){
41        string ss = s;
42        if(ss[pos] == 'X') return false;
43        ss[pos] = 'X';
44        for(int i = 1;i<N-1;i++){
45            if(ss[i] == 'X'){
46                if(ss[i-1] == 'X' && ss[i+1] == 'X') return true;
47            }
48        }
49        for(int i = 1;i<N-1;i++){

```

```

50     if(ss[i] == 'X'){
51         if(ss[i-1] == 'X' || ss[i+1] == 'X') return false;
52     }else if(ss[i] == '.'){
53         if(ss[i-1] == 'X' && ss[i+1] == 'X') return false;
54     }
55 }
56 int i = 0, lastX = -1;
57 int xorr = 0;
58 while(i < N){
59     if(ss[i] == 'X'){
60         int cnt = i - lastX - 2 - 1;
61         xorr = xorr ^ call(cnt);
62         lastX = i;
63         i++;
64         break;
65     }
66     i++;
67 }
68 while(i < N){
69     if(ss[i] == 'X'){
70         int cnt = i - lastX - 2 - 2 - 1;
71         xorr = xorr ^ call(cnt);
72         lastX = i;
73     }
74     i++;
75 }
76 int cnt = i - lastX - 2 - 1;
77 xorr = xorr ^ call(cnt);
78 if(xorr == 0) return true;
79 return false;
80 }
81
82 int main(){
83     fast_cin;
84     mems(DP, -1);
85     int nCase;
86     cin >> nCase;
87     for(int cs = 1; cs <= nCase; cs++){
88         cin >> s;
89         N = s.length();
90         res.clear();
91         for(int i = 0; i < N; i++){
92             if(isPossible(i) == true){

```

```

93         res.pb(i+1);
94     }
95 }
96 pf("Case %d:",cs);
97 int Sz = res.size();
98 for(int i = 0;i<Sz;i++){
99     pf(" %d",res[i]);
100 }
101 if(Sz == 0) pf(" 0\n");
102 else pf("\n");
103 }

```

Grundy_Knights_Move_In_Matrix.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**
6   Given a matrix and some knights with their positions.
7   Some possible moves are also given.
8   A player can move any knight from it's cell using any given possible move.
9   Last player who gives move win.
10 **/
11
12
13 #define Size 100005
14
15 int N;
16 int DP[1055][1055];
17 int dx[] = {1, -1, -1, -2, -3, -2};
18 int dy[] = {-2, -3, -2, -1, -1, 1};
19
20 bool isValid(int R,int C){
21     if(R<0 || C<0 || R>=1055 || C>=1055) return false;
22     return true;
23 }
24
25 int call(int R,int C){
26     //pf("Cur R: %d , C: %d\n",R,C);
27     if(R == 0 && C == 0) return 0;

```

```

28     if(DP[R][C] != -1) return DP[R][C];
29     vector<int> grundy;
30     for(int i = 0; i < 6; i++) {
31         int nR = R + dx[i];
32         int nC = C + dy[i];
33         if(isValid(nR,nC) == false) continue;
34         grundy.pb(call(nR,nC));
35     }
36     make_unique(grundy);
37     if(grundy.size() == 0) return DP[R][C] = 0;
38     if(grundy[0] != 0) return DP[R][C] = 0;
39     int f = 0;
40     for(int i = 1; i < grundy.size(); i++){
41         if(grundy[i] == f) continue;
42         f++;
43         if(grundy[i] != f) return DP[R][C] = f;
44     }
45     return DP[R][C] = grundy[grundy.size()-1] + 1;
46 }
47
48 int main(){
49     int nCase,x,y;
50     sf("%d",&nCase);
51     mems(DP,-1);
52     for(int cs = 1; cs <= nCase; cs++){
53         sf("%d",&N);
54         int res = 0;
55         for(int i = 0; i < N; i++){
56             sf("%d %d",&x,&y);
57             res = res ^ call(x,y);
58         }
59         if(res == 0) pf("Case %d: Bob\n",cs);
60         else pf("Case %d: Alice\n",cs);
61     }
62     return 0;
63 }

```

Matrix Nim.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;

```

```

4
5  /**
6   Each cell of matrix is a pile.
7   In each move a player can move stone to down or right cell.
8   Who gives last move win.
9  */
10
11
12  #define INF 999999999
13
14
15  int main() {
16      int nCase,r,c,a;
17      sf("%d", &nCase);
18      for (int cs = 1; cs <= nCase; cs++) {
19          scanf("%d %d", &r, &c);
20          int xsum = 0;
21          for(int i = 1;i<=r;i++){
22              for(int j = 1;j<=c;j++){
23                  sf("%d",&a);
24                  if((r+c)%2 != (i+j)%2) xsum ^= a;
25              }
26          }
27          if(xsum != 0) printf("Case %d: win\n", cs);
28          else printf("Case %d: lose\n", cs);
29      }
30      return 0;
31  }

```

Misere Nim.cpp

```

1  /**
2   In misere nim who gives last move win.
3   When all the pile have 1 stone then xor sum doesn't work (where you need to handle as
   a special case)
4  */
5
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef long long ll;
9  #define f first
10 #define s second

```

```

11  int main(){
12      ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
13
14      int tc; cin >> tc;
15      while(tc--){
16          int n; cin >> n;
17          int xor_sum = 0 , ones = 0;
18          for(int i = 0; i < n; ++i){
19              int x; cin >> x;
20              ones += (x==1);
21              xor_sum ^= x;
22          }
23          if(ones == n) cout << (n&1 ? "Second\n" : "First\n");
24          else cout << (xor_sum ? "First\n" : "Second\n");
25      }
26  }

```

Nim 1.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**
6   Given N piles a player can remove any number of stone from a pile to it's left.
7   **/
8
9  #define INF 999999999
10
11  int main() {
12      int nCase,n,a,b;
13      sf("%d", &nCase);
14      for (int cs = 1; cs <= nCase; cs++) {
15          scanf("%d", &n);
16          int xsum = 0;
17          for(int i = 0; i < n; i++) {
18              scanf("%d", &a);
19              xsum ^= (a);
20          }
21          if(xsum != 0) printf("Case %d: Alice\n", cs);
22          else printf("Case %d: Bob\n", cs);
23      }

```



```

24     return 0;
25 }

```

Staircase Nim.cpp

```

1  /// Include My Code Template
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  /**
6   Given a tree and coins in each vertex.
7   A player can move any no of coins to it's parent vertex.
8   Last player move wins.
9   In this problem for query a parent can be changed in the tree.
10 **/
11
12
13 #define Size 100005
14
15 int n, cnt, q, uu, vv;
16 int st[100005], en[100005], color[100005], parent[100005];
17 int whiteXor[100005], blackXor[100005], c[100005];
18 vector<int> Graph[100005];
19
20 void dfs(int pos, int par, int colr) {
21     cnt++;
22     st[pos] = cnt;
23     color[pos] = colr;
24     parent[pos] = par;
25
26     if(colr == 0) whiteXor[pos] = c[pos];
27     else blackXor[pos] = c[pos];
28
29     for (int i=0; i<Graph[pos].size(); i++) {
30         int j = Graph[pos][i];
31         if (j == par) continue;
32         dfs(j, pos, colr^1);
33         blackXor[pos] ^= blackXor[j];
34         whiteXor[pos] ^= whiteXor[j];
35     }
36     en[pos] = cnt;
37     return;

```

```

38 }
39
40 int main () {
41     scanf("%d", &n);
42     for (int i=1; i<=n; i++) {
43         scanf("%d", &c[i]);
44     }
45     for (int i=2; i<=n; i++) {
46         scanf("%d %d", &uu, &vv);
47         Graph[uu].pb(vv);
48         Graph[vv].pb(uu);
49     }
50     cnt = 0;
51     dfs(1,1,1);
52     scanf("%d", &q);
53     while (q--) {
54         scanf("%d %d", &uu, &vv);
55         if ( st[uu] <= st[vv] && en[uu] >= en[vv] ) {
56             printf("INVALID\n");
57         } else {
58             int cww = whiteXor[1], cbb = blackXor[1];
59             if (color[parent[uu]] == color[vv]) {
60                 if (cww > 0) {
61                     printf("YES\n");
62                 } else {
63                     printf("NO\n");
64                 }
65             } else {
66                 cww ^= whiteXor[uu];
67                 cww ^= blackXor[uu];
68                 if (cww > 0) {
69                     printf("YES\n");
70                 } else {
71                     printf("NO\n");
72                 }
73             }
74         }
75     }
76     return 0;
77 }

```

General

__int128.cpp

```

1  // Define int128 types and I/O operators
2  typedef __int128 int128;
3  typedef unsigned __int128 uint128;
4
5  // Helper functions for int128 I/O
6  ostream& operator << (ostream &os, int128 num) {
7      string str;
8      if(num == 0) return os << "0";
9      bool neg = false;
10     if(num < 0) {
11         neg = true;
12         num = -num;
13     }
14     while(num) {
15         str.push_back('0' + num % 10);
16         num /= 10;
17     }
18     if(neg) str.push_back('-');
19     reverse(str.begin(), str.end());
20     return os << str;
21 }
22
23 istream& operator >> (istream &is, int128 &num) {
24     string str;
25     is >> str;
26     num = 0;
27     bool neg = false;
28     for(char c : str) {
29         if(c == '-') neg = true;
30         else num = num * 10 + (c - '0');
31     }
32     if(neg) num = -num;
33     return is;
34 }

```

comp_double.cpp

```

1  #define ld long double
2  const ld EPS = 1e-6;
3  int dcmp(const ld &a, const ld &b) {

```

```

4    // Double compare
5    if (fabs(a - b) < EPS)
6        return 0;
7
8    return (a > b ? 1 : -1);
9 }

```

convert_double_int.cpp

```

1    int EPS = 10000 , Precision = 4;
2    long long read(){
3        // 1.2345 --> 12345
4        string s; cin >> s;
5        int x=s.find('.');
6        if(x==-1) return stoll(s+"0000");
7        string one=s.substr(0,x);
8        string two=s.substr(x+1);
9        while(two.size()< Precision) two+="0";
10       return stoll(one+two);
11   }

```

floor_ceil.cpp

```

1    ll ceil(ll a, ll b){
2        if(b < 0) a *= -1, b *= -1;
3        if(a < 0) return a / b;
4        else return (a + b - 1) / b;
5    }
6
7    ll floor(ll a, ll b){
8        if(b < 0) a *= -1, b *= -1;
9        if(a > 0) return a / b;
10       else return (a - b + 1) / b;
11   }

```

fraction.cpp

```

1    long long gcd(long long a, long long b) {
2        while (b) {

```

```

3     a %= b;
4     swap(a, b);
5 }
6     return a;
7 }
8 struct frac{
9     // fraction n / d
10    long long n ,d;
11    // constructor
12    frac(){n = 0; d = 1;}
13    frac(ll n , ll d): n(n) , d(d){simplify();};
14    bool operator < (const frac &other) const{
15        return n * other.d < d * other.n;
16    }
17    frac operator + (const frac &f) const{
18        frac ans {n * f.d + f.n * d , d * f.d};
19        ans.simplify();
20        return ans;
21    }
22    frac operator - (const frac &f) const{
23        frac ans {n * f.d - f.n * d , d * f.d};
24        ans.simplify();
25        return ans;
26    }
27    void Abs(){n = abs(n); d = abs(d);}
28    void simplify(){
29        long long g = gcd(abs(n) , abs(d));
30        n /= g; d /= g;
31    }
32 };

```

generate_n_digits_after_point.cpp

```

1  /*
2   ===== having a / b , generate n digits after point.=====
3   e.g : 1 / 3 = 0.333333333333333333333333333333
4   */
5   ll a , b; cin >> a >> b;
6   vector<int> v(n);
7   for(int i = 0; i < n; i++){
8       v[i] = a / b;
9       a = (a%b) * 10;

```

manual_multiply.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  #define rep(i, st, ed) for(int i = st; i < ed; i++)
6  using namespace std;
7  void burn(){
8  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
9  #ifndef ONLINE_JUDGE
10     freopen("in.txt", "r", stdin);
11     freopen("out.txt", "w", stdout);
12     freopen("error.txt", "w", stderr);
13 #endif
14 }
15 ///////////////////////////////////////////////////////////////////
16
17 int main(){
18     burn();
19     string s1, s2; cin>> s1 >> s2;
20     reverse(s1.begin(), s1.end());
21     reverse(s2.begin(), s2.end());
22     int MXN = 2 * max(s1.size(), s2.size()) + 9;
23     int a[MXN];
24     memset(a, 0, sizeof(a));
25     for (int i=0; i<s1.length(); i++) {
26         for (int j=0; j<s2.length(); j++) {
27             a[i + j] += (s1[i] - '0') * (s2[j] - '0');
28         }
29     }
30     for (int i=0; i<MXN - 1; i++) {
31         a[i + 1] += a[i] / 10;
32         a[i] %= 10;
33     }
34     int i = MXN - 1;
35     while (i > 0 && a[i] == 0) i--;
36     for (; i>=0; i--) cout<<a[i];
37     cout<<endl;
38 }

```

random.cpp

```

1  mt19937 random_seed(time(0));
2  long long rnd(long long l, long long r){
3      uniform_int_distribution<long long> dist(l, r);
4      return dist(random_seed);
5  }

```

Geometry

3dGeometry.cpp

```

1  struct Point {
2      double x, y, z;
3  };
4  long double eps = 1.2e-6;
5  Point cross(Point a, Point b, Point c) {
6      Point result;
7      result.x = (b.y - a.y) * (c.z - a.z) - (b.z - a.z) * (c.y - a.y);
8      result.y = (b.z - a.z) * (c.x - a.x) - (b.x - a.x) * (c.z - a.z);
9      result.z = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
10     return result;
11 }
12
13 double dot(Point a, Point b) {
14     return a.x * b.x + a.y * b.y + a.z * b.z;
15 }
16
17 double TetrahedronVolume(Point a, Point b, Point c, Point d) {
18     Point crossProduct = cross(a, b, c);
19     Point vectorAD = {d.x - a.x, d.y - a.y, d.z - a.z};
20     return dot(crossProduct, vectorAD) / 6.0;
21 }
22
23 bool intersect(Point a, Point b, Point c, Point p, Point q) {
24     if (TetrahedronVolume(p, a, b, c) * TetrahedronVolume(q, a, b, c) < 0 &&
25         TetrahedronVolume(p, q, a, b) * TetrahedronVolume(p, q, b, c) > 0 &&
26         TetrahedronVolume(p, q, b, c) * TetrahedronVolume(p, q, c, a) > 0)
27     {
28         return true;
29     }
30     return false;

```

31 }

Distinct_line_detecting.cpp

```

1  //ay + bx = c
2  void fix(int &a, int &b, int &c){
3      int g = __gcd(__gcd(a,b),c);
4      a/=g,b/=g,c/=g;
5      if(a < 0 || (a == 0 && b < 0))
6      {
7          a*=-1, b*=-1, c*=-1;
8      }
9  }
10 array<int,3> Line(int dx, int dy, Point P)
11 {
12     int a = dx;
13     int b = -dy;
14     int c = P.y() * a + P.x() * b;
15     fix(a,b,c);
16     return {a,b,c};
17 }
```

Geometry.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  #define all(v) v.begin() , v.end()
8  #ifndef ONLINE_JUDGE
9  #define debug(x) cerr << #x << ": " << x << '\n';
10 #else
11 #define debug(x)
12 #endif
13 #define ld long double
14 const ld EPS = 1e-6;
15 #define vec(a,b) ((b) - (a))
16 template<typename T = double> struct Point {
17     typedef Point P;
```



```

18     T x, y;
19     explicit Point(T x=0, T y=0) : x(x), y(y) {}
20     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
21     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
22     P operator+(P p) const { return P(x+p.x, y+p.y); }
23     P operator-(P p) const { return P(x-p.x, y-p.y); }
24     P operator*(T d) const { return P(x*d, y*d); }
25     P operator/(T d) const { return P(x/d, y/d); }
26     T dot(P p) const { return x*p.x + y*p.y; }
27     T cross(P p) const { return x*p.y - y*p.x; }
28     T cross(P a, P b) const { return (a-*this).cross(b-*this); }
29     T dist2() const { return x*x + y*y; }
30     double dist() const { return sqrt((double)dist2()); }
31     // angle to x-axis in interval [-pi, pi]
32     double angle() const { return atan2(y, x); }
33     P unit() const { return *this/dist(); } // makes dist()==1
34     P perp() const { return P(-y, x); } // rotates +90 degrees
35     P normal() const { return perp().unit(); }
36     // returns point rotated 'a' radians ccw around the origin
37     P rotate(double a) const {
38         return P(x*cos(a)-y*sin(a), x*sin(a)+y*cos(a)); }
39     friend ostream& operator<<(ostream& os, P p) {
40         return os << "(" << p.x << ", " << p.y << ")";
41     }
42     friend istream &operator>>(istream &os, P &p) {
43         return os >> p.x >> p.y;
44     }
45
46     // Project point onto line through a and b (assuming a != b).
47     P projectOnLine(const P &a, const P &b) const {
48         P ab = a.getVector(b);
49         P ac = a.getVector(*this);
50         return a + ab * ac.dot(ab) / a.dist2(b);
51     }
52
53     // Project point c onto line segment through a and b (assuming a != b).
54     P projectOnSegment(const P &a, const P &b) const {
55         P &c = *this;
56         P ab = a.getVector(b);
57         P ac = a.getVector(c);
58
59         long double r = dot(ac, ab), d = a.dist2(b);
60         if (r < 0) return a;
61         if (r > d) return b;

```

```

62
63     return a + ab * r / d;
64 }
65
66 P reflectAroundLine(const P &a, const P &b) const {
67     return projectOnLine(a, b) * 2 - (*this);
68 }
69
70 };
71 int dcmp(const Id &a, const Id &b) {
72     // Double compare
73     if (fabs(a - b) < EPS)
74         return 0;
75
76     return (a > b ? 1 : -1);
77 }
78 // length of vector
79 double length(const Point<> a){
80     return sqrt(a.x * a.x + a.y * a.y);
81 }
82 // cross product
83 double cross(const Point<> &a, const Point<> &b){
84     return a.x * b.y - b.x * a.y;
85 }
86 // Check if there is intersect two lines or not and return the intersection point
87 bool intersect(const Point<> &a, const Point<> &b,
88     const Point<> &p, const Point<> &q, Point<> &ret) {
89     //handle degenerate cases (2 parallel lines, 2 identical lines, line is 1 point)
90     double d1 = cross(p - a, b - a);
91     double d2 = cross(q - a, b - a);
92     ret = (q * d1 - p * d2) / (d1 - d2);
93     if (fabs(d1 - d2) > EPS) return 1;
94     return 0;
95 }
96
97 // dot product
98 double dot(const Point<> a, const Point<> b){ return a.x * b.x + a.y * b.y; }
99
100 // Point On Line
101 bool pointOnLine(const Point<> &a, const Point<> &b, const Point<> &p) {
102     // determine the point "p" is in the line or not
103     return fabs(cross(vec(a,b),vec(a,p))) < EPS;
104 }

```

```

105 // Is Point On Ray
106 bool pointOnRay(const Point<>& a, const Point<>& b, const Point<>& p) {
107     //IMP NOTE: a,b,p must be collinear
108     return dot(vec(a,p), vec(a,b)) > -EPS;
109 }
110 // Point On Segment
111 bool pointOnSegment(const Point<>& a, const Point<>& b, const Point<>& p) {
112     //el satr da momken y3mel precision error
113     if(!pointOnLine(a,b,p)) return 0;
114     return pointOnRay(a, b, p) && pointOnRay(b, a, p);
115 }
116 //Point Line Dist
117 double pointLineDist(const Point<>& a, const Point<>& b, const Point<>& p) {
118     // shortest distance between line and point
119     return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
120 }
121 // Point Segment Dist
122 double pointSegmentDist(const Point<> &a, const Point<> &b,const Point<> &p){
123     // shortest distance between segment and point
124     if (dot(vec(a,b),vec(a,p)) < EPS)
125         return length(vec(a,p));
126     if (dot(vec(b,a),vec(b,p)) < EPS)
127         return length(vec(b,p));
128     return pointLineDist(a, b, p);
129 }
130 // Count the number of Lattice Point in segment
131 int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
132     return abs(__gcd(x1 - x2, y1 - y2)) + 1;
133 }
134
135 template<class P> bool onSegment(P s, P e, P p) {
136     // check if point (p) on line (s , e)
137     return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
138 }
139
140 template<class P> vector<P> segInter(P a, P b, P c, P d) {
141     // The intersection between two segment, return the index section point
142     auto oa = c.cross(d, a), ob = c.cross(d, b),
143         oc = a.cross(b, c), od = a.cross(b, d);
144     // Checks if intersection is single non-endpoint point.
145     if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
146         return {(a * ob - b * oa) / (ob - oa)};
147     set<P> s;

```

```

148     if (onSegment(c, d, a)) s.insert(a);
149     if (onSegment(c, d, b)) s.insert(b);
150     if (onSegment(a, b, c)) s.insert(c);
151     if (onSegment(a, b, d)) s.insert(d);
152     return {s.begin(), s.end()};
153 }
154 vector<vector<Point<int>>> createLine(int x, int y, int d){
155     //
156     vector<vector<Point<int>>> ret;
157     ret.push_back({Point<int>(x+d, y), Point<int>(x, y+d)});
158     ret.push_back({Point<int>(x-d, y), Point<int>(x, y+d)});
159     ret.push_back({Point<int>(x-d, y), Point<int>(x, y-d)});
160     ret.push_back({Point<int>(x+d, y), Point<int>(x, y-d)});
161     return ret;
162 }
163
164 template<class P>
165 vector<P> circleLine(P c, double r, P a, P b) {
166     // the intersection of line and circle
167     P ab = b - a, p = a + ab * (c - a).dot(ab) / ab.dist2();
168     double s = a.cross(b, c), h2 = r * r - s * s / ab.dist2();
169     if (h2 < 0) return {};
170     if (h2 == 0) return {p};
171     P h = ab.unit() * sqrt(h2);
172     return {p - h, p + h};
173 }
174 // Cosine Rule
175 //get angle opposite to side a
176 double cosRule(double a, double b, double c) {
177     // Handle denom = 0
178     double res = (b * b + c * c - a * a) / (2 * b * c);
179     if (res > 1)
180         res = 1;
181     if (res < -1)
182         res = -1;
183     return acos(res);
184 }
185
186 Point<> normalize(const Point<> p){ return ((p) / length(p)); }
187 Point<> polar(const Point<> &r, double t){
188     return Point{r.x * cos(t) - r.y * sin(t), r.x * sin(t) + r.y * cos(t)};
189 }
190 // Circle Circle Intersection

```

```

191 int circleCircleIntersection(const Point<> &c1, const double &r1, const Point<> &c2,
    const double &r2, Point<> &res1, Point<> &res2) {
192     if (c1 == c2 && fabs(r1 - r2) < EPS) {
193         res1 = res2 = c1;
194         return fabs(r1) < EPS ? 1 : INT32_MAX;
195     }
196     double len = length(vec(c1,c2));
197     if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
198         Point<> d, c;
199         double r;
200         if (r1 > r2)
201             d = vec(c1,c2), c = c1, r = r1;
202         else
203             d = vec(c2,c1), c = c2, r = r2;
204         res1 = res2 = normalize(d) * r + c;
205         return 1; // intersect in one point
206     }
207     if (len > r1 + r2 || len < fabs(r1 - r2))
208         return 0; // intersect on two points
209     double a = cosRule(r2, r1, len);
210     Point<> c1c2 = normalize(vec(c1,c2)) * r1;
211     res1 = polar(c1c2, a) + c1;
212     res2 = polar(c1c2, -a) + c1;
213     return 2; // intersect in one point
214 }
215 // Circle From 3 Points
216 bool circle3(const Point<> &p1, const Point<> &p2, const Point<> &p3,
217     Point<> &cen, double &r) {
218     Point<> m1 = (p1 + p2) / 2;
219     Point<> m2 = (p2 + p3) / 2;
220     Point<> perp1 = vec(p1, p2);
221     perp1 = perp1.perp();
222     Point<> perp2 = vec(p2, p3);
223     perp2 = perp2.perp();
224     bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
225     r = length(vec(cen,p1));
226     return res;
227 }
228 double lengthSqr(const Point<> a){ return a.x * a.x + a.y * a.y; }
229 // check Point according to circle (in boundary, inside , outside)
230 int circlePoint(const Point<> &cen, const double &r, const Point<> &p) {
231     double lensqr = lengthSqr(vec(cen,p));

```

```

232     if (fabs(lensqr - r * r) < EPS)
233         return 1; // In the Boundary
234     if (lensqr < r * r)
235         return -1; // In the circle
236     return 0; // Out the circle
237 }
238 // Maximum triangle inside a circle
239 double maxAreaTriangleInsideCircle(double a){
240     return 3 * sqrt((double)3) * a * a / 4;
241 }
242 // find the Slope
243 pair<int,int> slope(pair<int,int> u, pair<int,int> v)
244 {
245     int dy = v.s-u.s;
246     int dx = v.f-u.f;
247     if(dx == 0) return {0,0};
248     if(dy == 0) return{0,1};
249     int sgn = (dy < 0) ^ (dx < 0);
250     if(sgn) sgn = -1; else sgn = 1;
251     return {sgn*abs(dx)/(abs(__gcd(dy,dx))),abs(dy)/(abs(__gcd(dy,dx)))};
252 }
253
254 int main(){
255     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);

```

Point.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ld = long double;
4  using ll = long long;
5  using pii = pair<int,int>;
6  using Point = complex<ld>;
7  const ld EPS = 1e-9;
8  #define X real()
9  #define Y imag()
10
11 // Core geometry operations
12 ld cross(const Point& a, const Point& b) { return imag(conj(a) * b); }
13 ld dot(const Point& a, const Point& b) { return real(conj(a) * b); }
14 ld dist2(const Point& a, const Point& b) { return norm(a - b); }

```

```

15  ld dist(const Point& a, const Point& b) { return abs(a - b); }
16  ld angle(const Point& a, const Point& b) { return atan2(cross(a,b), dot(a,b)); }
17
18  // Orientation check
19  int ccw(const Point& a, const Point& b, const Point& c) {
20      ld cr = cross(b-a, c-a);
21      if(cr > EPS) return +1; // CCW
22      if(cr < -EPS) return -1; // CW
23      if(dot(b-a, c-a) < 0) return +2; // c--a--b
24      if(norm(b-a) < norm(c-a)) return -2; // a--b--c on line
25      return 0; // Colinear and overlapping
26  }
27
28  // Integer slope
29  pii slope(const pii& a, const pii& b) {
30      int dy = b.second - a.second, dx = b.first - a.first;
31      if(dx == 0) return {0,0}; // Vertical
32      if(dy == 0) return {0,1}; // Horizontal
33      int g = __gcd(abs(dy), abs(dx));
34      dy /= g; dx /= g;
35      if(dx < 0) dy = -dy, dx = -dx;
36      return {dy,dx};
37  }
38
39  // Point operations
40  Point rotate(const Point& p, const Point& c, ld theta) {
41      return (p-c) * polar((ld)1.0, theta) + c;
42  }
43
44  Point proj(const Point& p, const Point& a, const Point& b) {
45      Point ab = b-a;
46      return a + ab * dot(p-a, ab) / norm(ab);
47  }
48
49  Point refl(const Point& p, const Point& a, const Point& b) {
50      Point prj = proj(p,a,b);
51      return prj * (ld)2.0 - p;
52  }
53
54  // Input/Output
55  istream& operator>>(istream& is, Point& p) {
56      ld x,y; is >> x >> y; p = Point(x,y);
57      return is;
58  }

```

```

59
60 ostream& operator<<(ostream& os, const Point& p) {
61     return os << p.X << ' ' << p.Y;
62 }
63
64 void solve() {
65     // Your solution here
66     Point a, b;
67     cin >> a >> b;
68     cout << dist(a,b) << '\n';
69 }
70
71 int main() {
72     ios::sync_with_stdio(0);
73     cin.tie(0);
74
75     int t = 1;
76     // cin >> t; // Uncomment for multiple test cases
77     while(t--) solve();
78
79     return 0;

```

Triangle.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long double ld;
5  typedef complex<ld> Point;
6
7  inline ld cross(const Point& a, const Point& b) { return a.real() * b.imag() - a.imag() *
    b.real(); }
8  inline ld lengthSqr(const Point& p) { return norm(p); }
9  inline ld triangleAreaBH(ld base, ld height) { return base * height / 2.0L; }
10 inline ld triangleArea2SidesAngle(ld a, ld b, ld angle) { return fabs(a * b * sin(angle)) /
    2.0L; }
11 inline ld triangleArea2AnglesSide(ld ang1, ld ang2, ld side) {
12     return fabs(side * side * sin(ang1) * sin(ang2) / (2.0L * sin(ang1 + ang2)));
13 }
14 inline ld triangleArea3Sides(ld a, ld b, ld c) {
15     ld s = (a + b + c) / 2.0L;
16     return sqrt(s * (s - a) * (s - b) * (s - c));

```



```

17 }
18 inline ld cosRule(ld a, ld b, ld c) {
19     ld denom = 2.0L * b * c;
20     if (fabs(denom) < 1e-9) return 0.0L;
21     return acos(min(max((b * b + c * c - a * a) / denom, -1.0L), 1.0L));
22 }
23
24 int main() { return 0; }

```

areaOfrectangles.cpp

```

1  /*
2   Given n rectange calculate the area of total rectange (take in care that rectange can
   intersect)
3
4   */
5   #include <bits/stdc++.h>
6
7   using namespace std;
8   typedef long long ll;
9   typedef array<int,4> Operation;
10  const int maxN = 1e5;
11  const int SZ = 9e6;
12
13  int N, lo[SZ], hi[SZ];
14  ll area, delta[SZ], score[SZ];
15  Operation op[2*maxN];
16
17  int len(int i){
18      return hi[i]-lo[i]+1;
19  }
20
21  void pull(int i){
22      if(lo[i] == hi[i]) score[i] = (delta[i] > 0 ? 1 : 0);
23      else score[i] = (delta[i] > 0 ? len(i) : score[2*i] + score[2*i+1]);
24  }
25
26  void build(int i, int l, int r){
27      lo[i] = l; hi[i] = r;
28      if(l == r) return;
29      int m = l+(r-l)/2;
30      build(2*i, l, m);
31      build(2*i+1, m+1, r);

```

```

32 }
33
34 void increment(int i, int l, int r, ll val){
35     if(l > hi[i] || r < lo[i]) return;
36     if(l <= lo[i] && hi[i] <= r){
37         delta[i] += val;
38         pull(i);
39         return;
40     }
41     increment(2*i, l, r, val);
42     increment(2*i+1, l, r, val);
43     pull(i);
44 }
45
46 ll query(){
47     return score[1];
48 }
49
50 int main(){
51     scanf("%d", &N);
52     for(int i = 0, a, b, c, d; i < N; i++){
53         scanf("%d %d %d %d", &a, &b, &c, &d);
54         op[2*i] = {1, b, a+1, c};
55         op[2*i+1] = {-1, d, a+1, c};
56     }
57     sort(op, op+2*N, [](Operation A, Operation B){
58         return (A[1] == B[1] ? A[0] < B[0] : A[1] < B[1]);
59     });
60
61     build(1, -1e6-5, 1e6+5);
62     int lst = -1e6;
63     for(int i = 0; i < 2*N; i++){
64         int t = op[i][0], y = op[i][1], x1 = op[i][2], x2 = op[i][3];
65         area += (y-lst) * query();
66         increment(1, x1, x2, t);
67         lst = y;
68     }
69
70     printf("%lld\n", area);
71 }

```

circle.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long double ld;
4  typedef complex<ld> Point;
5  const ld EPS = 1e-9;
6
7  // Cross product of OA and OB vectors
8  ld cross(const Point& O, const Point& A, const Point& B) { return imag((A - O) * conj(B -
  O)); }
9
10 // Dot product of vectors A and B
11 ld dot(const Point& A, const Point& B) { return real(A * conj(B)); }
12
13 // Circle-Line Intersection: returns intersection points
14 vector<Point> circleLine(const Point& c, ld r, const Point& a, const Point& b){
15     Point ab = b - a; ld ab2 = norm(ab);
16     ld t = dot(ab, c - a) / ab2;
17     Point p = a + ab * t; ld s2 = r*r - norm(c - p);
18     if(s2 < -EPS) return {};
19     if(abs(s2) < EPS) return {p};
20     ld s = sqrt(s2); Point h = ab / abs(ab) * s;
21     return {p - h, p + h};
22 }
23
24 // Cosine Rule: angle opposite to side a
25 ld cosRule(ld a, ld b, ld c){
26     ld res = (b*b + c*c - a*a)/(2*b*c);
27     return acos(max(-1.0L, min(1.0L, res)));
28 }
29
30 // Circle-Circle Intersection: returns number of intersections and sets res1, res2
31 int circleCircle(const Point& c1, ld r1, const Point& c2, ld r2, Point& res1, Point& res2){
32     Point d = c2 - c1; ld dist = abs(d);
33     if(dist < EPS && abs(r1 - r2) < EPS) return INT32_MAX; // Infinite intersections
34     if(dist > r1 + r2 + EPS || dist < abs(r1 - r2) - EPS) return 0; // No intersection
35     ld a = (r1*r1 - r2*r2 + dist*dist)/(2*dist);
36     ld h2 = r1*r1 - a*a;
37     if(h2 < -EPS) return 0;
38     ld h = h2 < EPS ? 0 : sqrt(h2);
39     Point p = c1 + d * (a / dist);
40     if(h == 0){ res1 = p; return 1; }
41     Point offset = d * (h / dist) * Point(0,1);
42     res1 = p + offset; res2 = p - offset;
43     return 2;

```

```

44 }
45
46 // Circle from Three Points: returns true if successful
47 bool circle3(const Point& p1, const Point& p2, const Point& p3, Point& cen, Id& r){
48     Id a = cross(p1, p2, p3);
49     if(abs(a) < EPS) return false; // Collinear points
50     cen = ((norm(p2) - norm(p1)) * Point(0,1) - (norm(p3) - norm(p1)) * Point(1,0)) / (2*a) +
    p1;
51     r = abs(cen - p1);
52     return true;
53 }
54
55 // Point Position Relative to Circle: -1 inside, 0 outside, 1 on boundary
56 int circlePoint(const Point& cen, Id r, const Point& p){
57     Id dist2 = norm(p - cen), r2 = r*r;
58     if(abs(dist2 - r2) < EPS) return 1;
59     return (dist2 < r2) ? -1 : 0;
60 }
61
62 int main(){
63     ios::sync_with_stdio(false);
64     cin.tie(0);
65

```

convex_hull.cpp

```

1  #define double long double
2  struct Point {
3      double x, y;
4  };
5
6  int orientation(Point a, Point b, Point c) {
7      double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
8      if (v < 0) return -1; // clockwise
9      if (v > 0) return +1; // counter-clockwise
10     return 0;
11 }
12
13 bool cw(Point a, Point b, Point c, bool include_collinear) {
14     int o = orientation(a, b, c);
15     return o < 0 || (include_collinear && o == 0);
16 }

```

```

17  bool collinear(Point a, Point b, Point c) { return orientation(a, b, c) == 0; }
18
19  void convex_hull(vector<Point>& a, bool include_collinear = false) {
20      Point p0 = *min_element(a.begin(), a.end(), [](Point a, Point b) {
21          return make_pair(a.y, a.x) < make_pair(b.y, b.x);
22      });
23      sort(a.begin(), a.end(), [&p0](const Point& a, const Point& b) {
24          int o = orientation(p0, a, b);
25          if (o == 0)
26              return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
27                  < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
28          return o < 0;
29      });
30      if (include_collinear) {
31          int i = (int)a.size()-1;
32          while (i >= 0 && collinear(p0, a[i], a.back())) i--;
33          reverse(a.begin()+i+1, a.end());
34      }
35
36      vector<Point> st;
37      for (int i = 0; i < (int)a.size(); i++) {
38          while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
39              st.pop_back();
40          st.push_back(a[i]);
41      }
42
43      a = st;
44  }
45  double area(Point a, Point b, Point c)
46  {
47      return 0.5*abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
48  }

```

dynamic_convex_hull.cpp

```

1  typedef long double ld;
2  struct Line {
3      ll m, b;
4      mutable function<const Line *> succ;
5
6      bool operator<(const Line &other) const {
7          return m < other.m;

```

```

8     }
9
10    bool operator<(const ll &x) const {
11        const Line *s = succ();
12        if (!s)
13            return 0;
14        return b - s->b < (s->m - m) * x;
15    }
16 };
17 // will maintain upper hull for maximum
18 struct HullDynamic : public multiset<Line, less<>> {
19     bool bad(iterator y) {
20         auto z = next(y);
21         if (y == begin()) {
22             if (z == end())
23                 return 0;
24             return y->m == z->m && y->b <= z->b;
25         }
26         auto x = prev(y);
27         if (z == end())
28             return y->m == x->m && y->b <= x->b;
29         return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m - x->m);
30     }
31
32     void insert_line(ll m, ll b) { // log(n)
33         m *= -1;
34         b *= -1;
35         auto y = insert({m, b});
36         y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
37         if (bad(y)) {
38             erase(y);
39             return;
40         }
41         while (next(y) != end() && bad(next(y)))
42             erase(next(y));
43         while (y != begin() && bad(prev(y)))
44             erase(prev(y));
45     }
46
47     ll query(ll x) { // log(n)
48         if(size() == 0) return 5e18; // degerated case
49         auto l = *lower_bound(x);
50         return -(l.m * x + l.b);
51     }

```

getClosestPair.cpp

```

1  #define type double
2  #define MapIterator map<type, multiset<type> >::iterator
3  #define SetIterator multiset<type>::iterator
4
5  const int SIZE = 10000; //Maximum number of points
6  type x[SIZE], y[SIZE]; //Coordinates of points
7  int N; //Number of points
8  double INF = INT_MAX;
9  double getClosestPair() {
10     map<type, multiset<type> > points;
11     for (int i = 0; i < N; i++)
12         points[x[i]].insert(y[i]);
13     double d = INF;
14     for (MapIterator xitr1 = points.begin(); xitr1 != points.end(); xitr1++){
15         for (SetIterator yitr1 = (*xitr1).second.begin(); yitr1!= (*xitr1).second.end(); yitr1++) {
16             type x1 = (*xitr1).first, y1 = *yitr1;
17             MapIterator xitr3 = points.upper_bound(x1 + d);
18             for (MapIterator xitr2 = xitr1; xitr2 != xitr3; xitr2++)
19             {
20                 type x2 = (*xitr2).first;
21                 SetIterator yitr2 = (*xitr2).second.lower_bound(y1 - d);
22                 SetIterator yitr3 = (*xitr2).second.upper_bound(y1 + d);
23                 for (SetIterator yitr4 = yitr2; yitr4 != yitr3; yitr4++) {
24                     if (xitr1 == xitr2 && yitr1 == yitr4)
25                         continue; //same point     type y2 = *yitr4;
26                     d = min(d, hypot(x1 - x2, y1 - y2));
27                 }
28             }
29         }
30     }
31     return d;
32 }

```

line.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;

```

```

3  typedef long double ld;
4  typedef complex<ld> Point;
5  const ld EPS = 1e-9;
6  #define X real()
7  #define Y imag()
8
9  // Geometry Helpers
10 ld cross(const Point& a, const Point& b) { return imag(conj(a) * b); }
11 ld dot(const Point& a, const Point& b) { return real(conj(a) * b); }
12 ld length_sq(const Point& a) { return norm(a); }
13 ld length_(const Point& a) { return abs(a); }
14
15 // Point on Line
16 bool pointOnLine(const Point& a, const Point& b, const Point& p) {
17     return abs(cross(b - a, p - a)) < EPS;
18 }
19
20 // Point on Ray
21 bool pointOnRay(const Point& a, const Point& b, const Point& p) {
22     return dot(p - a, b - a) > -EPS;
23 }
24
25 // Point on Segment
26 bool pointOnSegment(const Point& a, const Point& b, const Point& p) {
27     return pointOnLine(a, b, p) && pointOnRay(a, b, p) && pointOnRay(b, a, p);
28 }
29
30 // Distance from Point to Line
31 ld pointLineDist(const Point& a, const Point& b, const Point& p) {
32     return abs(cross(b - a, p - a)) / length_(b - a);
33 }
34
35 // Distance from Point to Segment
36 ld pointSegmentDist(const Point& a, const Point& b, const Point& p){
37     if(dot(b - a, p - a) < EPS) return length_(p - a);
38     if(dot(a - b, p - b) < EPS) return length_(p - b);
39     return pointLineDist(a, b, p);
40 }
41
42 // Line Intersection
43 bool intersectLines(const Point& a, const Point& b, const Point& p, const Point& q,
44     Point& r) {
45     ld d1 = cross(p - a, b - a);
46     ld d2 = cross(q - a, b - a);

```



```

46     if(abs(d1 - d2) < EPS) return false;
47     r = (q * d1 - p * d2) / (d1 - d2);
48     return true;
49 }
50 vector<Point> segInter(const Point& a, const Point& b, const Point& c, const Point& d) {
51     Point r;
52     if(intersectLines(a, b, c, d, r)) {
53         // Check if r is on both segments
54         if(pointOnSegment(a, b, r) && pointOnSegment(c, d, r))
55             return {r};
56     }
57     auto on = [&](const Point& s, const Point& e, const Point& p) -> bool {
58         return pointOnSegment(s, e, p);
59     };
60     vector<Point> res;
61     if(on(a, b, c)) res.emplace_back(c);
62     if(on(a, b, d)) res.emplace_back(d);
63     if(on(c, d, a)) res.emplace_back(a);
64     if(on(c, d, b)) res.emplace_back(b);
65     sort(res.begin(), res.end(), [&](const Point& x, const Point& y) -> bool {
66         return real(x) < real(y) || (abs(real(x) - real(y)) < EPS && imag(x) < imag(y));
67     });
68     res.erase(unique(res.begin(), res.end(), [&](const Point& x, const Point& y) -> bool {
69         return abs(x - y) < EPS;
70     }), res.end());
71     return res;
72 }
73
74 // Count Lattice Points on Segment
75 int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
76     return abs(__gcd(x1 - x2, y1 - y2)) + 1;
77 }
78
79 // Create Lines (e.g., for cross shapes)
80 vector<pair<Point, Point>> createLines(int x, int y, int d){
81     return {
82         {Point(x + d, y), Point(x, y + d)},
83         {Point(x - d, y), Point(x, y + d)},
84         {Point(x - d, y), Point(x, y - d)},
85         {Point(x + d, y), Point(x, y - d)}
86     };
87 }
88 istream& operator>>(istream& is, Point& p) {

```

```

89     ld x,y; is >> x >> y; p = Point(x,y);
90     return is;
91 }
92
93 ostream& operator<<(ostream& os, const Point& p) {
94     return os << p.X << ' ' << p.Y;
95 }
96 // Example Usage
97 int main(){
98     ios::sync_with_stdio(false);
99     cin.tie(0);
100
101     int n = 4;
102     vector<Point> v(n);
103     for(auto &i : v) cin >> i;
104     Point q;
105     intersectLines(v[0], v[1], v[2], v[3], q);

```

minimum_enclosing_circle.cpp

```

1  const double EPS = 1e-9;
2
3  #define EQ(a, b) (fabs((a) - (b)) <= EPS)
4  #define LE(a, b) ((a) <= (b) + EPS)
5
6  typedef std::pair<double, double> point;
7  #define x first
8  #define y second
9
10 double sqnorm(const point &a) { return a.x*a.x + a.y*a.y; }
11 double norm(const point &a) { return sqrt(sqnorm(a)); }
12
13 struct circle {
14     double h, k, r;
15
16     circle() : h(0), k(0), r(0) {}
17     circle(double h, double k, double r) : h(h), k(k), r(fabs(r)) {}
18
19     // Circle with the line segment ab as a diameter.
20     circle(const point &a, const point &b) {
21         h = (a.x + b.x)/2.0;

```

```

22     k = (a.y + b.y)/2.0;
23     r = norm(point(a.x - h, a.y - k));
24 }
25
26 // Circumcircle of three points.
27 circle(const point &a, const point &b, const point &c) {
28     double an = sqnorm(point(b.x - c.x, b.y - c.y));
29     double bn = sqnorm(point(a.x - c.x, a.y - c.y));
30     double cn = sqnorm(point(a.x - b.x, a.y - b.y));
31     double wa = an*(bn + cn - an);
32     double wb = bn*(an + cn - bn);
33     double wc = cn*(an + bn - cn);
34     double w = wa + wb + wc;
35     if (EQ(w, 0)) {
36         throw std::runtime_error("No circumcircle from collinear points.");
37     }
38     h = (wa*a.x + wb*b.x + wc*c.x)/w;
39     k = (wa*a.y + wb*b.y + wc*c.y)/w;
40     r = norm(point(a.x - h, a.y - k));
41 }
42
43 bool contains(const point &p) const {
44     return LE(sqnorm(point(p.x - h, p.y - k)), r*r);
45 }
46 };
47
48 template<class It>
49 circle minimum_enclosing_circle(It lo, It hi) {
50     if (lo == hi) {
51         return circle(0, 0, 0);
52     }
53     if (lo + 1 == hi) {
54         return circle(lo->x, lo->y, 0);
55     }
56     std::random_shuffle(lo, hi);
57     circle res(*lo, *(lo + 1));
58     for (It i = lo + 2; i != hi; ++i) {
59         if (res.contains(*i)) {
60             continue;
61         }
62         res = circle(*lo, *i);
63         for (It j = lo + 1; j != i; ++j) {
64             if (res.contains(*j)) {
65                 continue;

```

```

66     }
67     res = circle(*i, *j);
68     for (It k = lo; k != j; ++k) {
69         if (!res.contains(*k)) {
70             res = circle(*i, *j, *k);
71         }
72     }
73 }
74 }
75 return res;

```

polygon.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long double ld;
4  typedef complex<ld> Point;
5  #define X real()
6  #define Y imag()
7  const ld EPS = 1e-9;
8
9  // Cross product
10 ld cross(const Point& a, const Point& b) {
11     return imag(conj(a) * b);
12 }
13 ld cross(const Point& O, const Point& A, const Point& B) {
14     return imag((A - O) * conj(B - O));
15 }
16
17 // Polygon area (absolute)
18 //cross function is the needed function
19 ld polygonArea(const vector<Point>& p) {
20     ld res = 0;
21     for(int i = 0; i < p.size(); i++) res += cross(p[i], p[(i+1)%p.size()]);
22     return abs(res) / 2.0;
23 }
24
25 // Polygon centroid
26 //cross function is needed here
27 Point polygonCentroid(const vector<Point>& p) {
28     Point c(0,0); ld A = 0;
29     for(int i = 0; i < p.size(); i++) {

```

```

30     int j = (i+1) % p.size();
31     ld cross_prod = cross(p[i], p[j]);
32     c += (p[i] + p[j]) * cross_prod;
33     A += cross_prod;
34 }
35 return c / (3.0 * A);
36 }
37
38 // Line intersection, returns true if intersecting
39 //cross function is needed here
40 bool intersect(const Point& a, const Point& b, const Point& p, const Point& q, Point& r) {
41     ld d1 = cross(p - a, b - a);
42     ld d2 = cross(q - a, b - a);
43     if(abs(d1 - d2) < EPS) return false;
44     r = (q * d1 - p * d2) / (d1 - d2);
45     return true;
46 }
47
48 // Sutherland–Hodgman polygon clipping
49 //cross function is needed here as well as the intersect function
50 void sortCounterClockwise(vector<Point>& p) {
51     Point c = polygonCentroid(p);
52     sort(p.begin(), p.end(), [&](const Point& a, const Point& b) -> bool {
53         ld angle_a = arg(a - c);
54         ld angle_b = arg(b - c);
55         return angle_a < angle_b;
56     });
57 }
58 void polygonCut(const vector<Point>& subject, const Point& a, const Point& b,
vector<Point>& res) {
59     res.clear();
60     for(int i = 0; i < subject.size(); i++) {
61         int j = (i+1) % subject.size();
62         ld cross1 = cross(b - a, subject[i] - a);
63         ld cross2 = cross(b - a, subject[j] - a);
64         bool in1 = cross1 > EPS, in2 = cross2 > EPS;
65         if(in1) res.push_back(subject[i]);
66         if(in1 != in2) { Point r; if(intersect(a, b, subject[i], subject[j], r)) res.push_back(r); }
67     }
68 }
69 //for identifying the number of lattice points in a polygon
70 int picksTheorem(int a, int b) {
71     // a: area of polygon, b: no. lattice Points in the boundaries
72     return a - b / 2 + 1;

```

```

73 }
74 int picksTheorem(vector<Point>& p, bool b = 0) {
75     // Point sorted in counter clock-wise;
76     Id area = 0;
77     int bound = 0;
78     int sz = (int) p.size();
79     for(int i = 0; i < sz; i++) {
80         int j = (i + 1) % sz;
81         area += cross(p[i], p[j]);
82         Point v = p[j] - p[i];
83         bound += abs(__gcd((int)round(real(v)), (int)round(imag(v))));
84     }
85     area /= 2;
86     area = fabs(area);
87     return round(area - bound / 2 + 1) + b * bound;
88 }
89
90 // Convex polygon intersection
91 //polygoncut(cross, intersect) function is needed
92 void convexIntersect(const vector<Point>& p, const vector<Point>& q, vector<Point>&
res) {
93     res = q;
94     for(int i = 0; i < p.size(); i++) {
95         int j = (i+1) % p.size();
96         vector<Point> temp;
97         polygonCut(res, p[i], p[j], temp);
98         res = temp; if(res.empty()) break;
99     }
100 }
101 // Cross product of vectors OA and OB (returns z-component)
102
103
104 vector<Point> convex_hull(vector<Point> pts, bool include_collinear = false) {
105     int n = pts.size(), k = 0;
106     if(n <= 1) return pts;
107     sort(pts.begin(), pts.end(), [&](const Point& a, const Point& b) -> bool {
108         return (real(a) < real(b)) || (abs(real(a) - real(b)) < EPS && imag(a) < imag(b));
109     });
110
111     vector<Point> hull(2 * n);
112     for(int i = 0; i < n; ++i) {
113         while(k >= 2 && (include_collinear ? cross(hull[k-2], hull[k-1], pts[i]) < 0
114             : cross(hull[k-2], hull[k-1], pts[i]) <= 0))

```

```

115     k--;
116     hull[k++] = pts[i];
117 }
118 for(int i = n-2, t = k+1; i >= 0; --i){
119     while(k >= t && (include_collinear ? cross(hull[k-2], hull[k-1], pts[i]) < 0
120         : cross(hull[k-2], hull[k-1], pts[i]) <= 0))
121         k--;
122     hull[k++] = pts[i];
123 }
124 hull.resize(k-1);
125 return hull;
126 }
127 istream& operator>>(istream& is, Point& p) {
128     ld x,y; is >> x >> y; p = Point(x,y);
129     return is;
130 }
131
132 ostream& operator<<(ostream& os, const Point& p) {
133     return os << p.X << ' ' << p.Y;
134 }
135 // Example usage
136 int main(){
137     ios::sync_with_stdio(false);
138     cin.tie(0);
139
140     int n; cin >> n;
141     vector<Point> poly1(n); for(auto &p : poly1) cin >> p;
142     sortCounterClockwise(poly1);
143     poly1 = convex_hull(poly1,1);
144     cout << poly1.size() << '\n';

```

sweep_line.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int bit[2000005];
5
6  void update(int i, int x) {
7      for (; i < 2000005; i += i & (-i)) bit[i] += x;
8  }

```

```

9  int query(int i) {
10     int sum = 0;
11     for (; i > 0; i -= i & (-i)) sum += bit[i];
12     return sum;
13 }
14
15 int n;
16 vector<array<int, 4>> v;
17
18 int main() {
19     cin.tie(0)->sync_with_stdio(0);
20
21     cin >> n;
22     for (int i = 0, x1, y1, x2, y2; i < n; ++i) {
23         cin >> x1 >> y1 >> x2 >> y2;
24         if (y1 == y2) v.push_back({y1, 2, x1, x2});
25         else {
26             v.push_back({y1, 1, x1, 1});
27             v.push_back({y2, 3, x1, 1});
28         }
29     }
30     sort(begin(v), end(v));
31
32     long long ans = 0;
33     for (auto x : v) {
34         x[2] += 1000001, x[3] += 1000001;
35         if (x[1] == 1) update(x[2], 1);
36         else if (x[1] == 2) ans += query(x[3]) - query(x[2] - 1);
37         else update(x[2], -1);
38     }
39     cout << ans << '\n';
40 }

```

Number theory

CRT-Offline.cpp

```

1  /// Chinese Remainder Theorem
2  /// Returns the smallest number x such that,
3  /// x % num[i] = rem[i] for each i
4  /// Numbers in num[] are pairwise co prime
5

```



```

6  LL num[Size];
7  LL rem[Size];
8
9  LL CRT(int N){ /// N is size of num/rem
10     LL prod = 1;
11     for (int i = 0; i < N; i++){
12         prod *= num[i];
13     }
14     LL result = 0;
15     for (int i = 0; i < N; i++){
16         LL pp = prod / num[i];
17         result += rem[i] * modInv(pp, num[i]) * pp;
18     }
19     return (result % prod);
20 }

```

CRT-Online.cpp

```

1  /// A is the list of reminders
2  /// M is the list of mod values
3  /// Doesn't work when the mod values aren't pairwise co-prime
4  vector<LL> A, M;
5
6  LL CRT(vector<LL> &A, vector<LL> &M){
7      myint a1 = A[0], m1 = M[0];
8      FOR(i, 1, SZ(A)-1){
9          LL a2 = A[i], m2 = M[i];
10         LL p, q;
11         ext_gcd(m1, m2, &p, &q);
12         LL mod = m1*m2;
13         LL x = (((a1*m2)%mod)*q)%mod + (((a2*m1)%mod)*p)%mod)%mod;
14         a1 = x;
15         m1 = mod;
16         if(a1 < 0) a1 += mod;
17     }
18     if(a1 < 0) a1 += m1;
19     return a1;
20 }

```

CRT-OnlineNonCoPrimeModuli.cpp

```

1 // #define __int128 LL /// Change here if __int128 is not supported
2
3 /**
4  A CRT solver which works even when moduli are not pairwise coprime
5  1. Call clear()
6  2. Add equations using addEquation() method
7  3. Call solve() to get {x, N} pair, where x is the unique solution modulo N.
8  Assumptions: LCM of all mods will fit into long long.
9  */
10
11 class ChineseRemainderTheorem {
12     typedef long long vlong;
13     typedef pair<vlong,vlong> pll;
14
15     /** CRT Equations stored as pairs of vector. See addEquation()*/
16     vector<pll> equations;
17
18 public:
19     void clear() {
20         equations.clear();
21     }
22
23     /** Add equation of the form  $x = r \pmod{m}$ */
24     void addEquation( vlong r, vlong m ) {
25         equations.push_back({r, m});
26     }
27     pll solve() {
28         if (equations.size() == 0) return {-1,-1}; /// No equations to solve
29
30         vlong a1 = equations[0].first;
31         vlong m1 = equations[0].second;
32         a1 %= m1;
33         /** Initially  $x = a_0 \pmod{m_0}$ */
34
35         /** Merge the solution with remaining equations */
36         for ( int i = 1; i < equations.size(); i++ ) {
37             vlong a2 = equations[i].first;
38             vlong m2 = equations[i].second;
39
40             vlong g = __gcd(m1, m2);
41             if ( a1 % g != a2 % g ) return {-1,-1}; /// Conflict in equations
42
43             /** Merge the two equations*/
44             vlong p, q;

```

```

45     ext_gcd(m1/g, m2/g, &p, &q);
46
47     vlong mod = m1 / g * m2;
48     vlong x = ( (__int128)a1 * (m2/g) % mod *q % mod + (__int128)a2 * (m1/g) % mod * p
% mod ) % mod;
49
50     /** Merged equation*/
51     a1 = x;
52     if ( a1 < 0 ) a1 += mod;
53     m1 = mod;
54 }
55 return {a1, m1};
56 }

```

Euler Phi.cpp

```

1  #define Max 1000000
2  int phi[Max];
3
4  void euler_phi(){
5      phi[1] = 1;
6      for(int i = 2; i < Max; i++){
7          if(!phi[i]){
8              phi[i] = i-1;
9              for(int j = (i<2); j < Max; j+=i){
10                 if(!phi[j]){
11                     phi[j] = j;
12                 }
13                 phi[j] = phi[j]/i*(i-1);
14             }
15         }
16     }
17 }

```

Linear_Diophantine_Equation.cpp

```

1  /*
2  ## Linear Diophantine Equation
3  A Linear Diophantine Equation (in two variables) is an equation of the general form:
4  a.x + b.y = c

```

```

5   where a,b are given intergs, and x,y are unknown integers.
6
7   In this code, we consider several classical problems on these equations:
8
9   1. finding one solution
10  2. finding all solutions
11  3. finding the number of solutions and the solutions themselves in a given interval
12  4.finding a solution with minimum value of  $x + y$ 
13  */
14
15  // ## Finding one solution
16  int gcd(int a, int b, int& x, int& y) {
17      if (b == 0) {
18          x = 1;
19          y = 0;
20          return a;
21      }
22      int x1, y1;
23      int d = gcd(b, a % b, x1, y1);
24      x = y1;
25      y = x1 - y1 * (a / b);
26      return d;
27  }
28
29  bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
30      g = gcd(abs(a), abs(b), x0, y0);
31      if (c % g) {
32          return false;
33      }
34
35      x0 *= c / g;
36      y0 *= c / g;
37      if (a < 0) x0 = -x0;
38      if (b < 0) y0 = -y0;
39      return true;
40  }
41
42  // ## Find number of solution when x: [minx, maxx] , y: [miny,maxy]
43  void shift_solution(int & x, int & y, int a, int b, int cnt) {
44      x += cnt * b;
45      y -= cnt * a;
46  }
47
48  int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {

```

```

49     int x, y, g;
50     if (!find_any_solution(a, b, c, x, y, g))
51         return 0;
52     a /= g;
53     b /= g;
54
55     int sign_a = a > 0 ? +1 : -1;
56     int sign_b = b > 0 ? +1 : -1;
57
58     shift_solution(x, y, a, b, (minx - x) / b);
59     if (x < minx)
60         shift_solution(x, y, a, b, sign_b);
61     if (x > maxx)
62         return 0;
63     int lx1 = x;
64
65     shift_solution(x, y, a, b, (maxx - x) / b);
66     if (x > maxx)
67         shift_solution(x, y, a, b, -sign_b);
68     int rx1 = x;
69
70     shift_solution(x, y, a, b, -(miny - y) / a);
71     if (y < miny)
72         shift_solution(x, y, a, b, -sign_a);
73     if (y > maxy)
74         return 0;
75     int lx2 = x;
76
77     shift_solution(x, y, a, b, -(maxy - y) / a);
78     if (y > maxy)
79         shift_solution(x, y, a, b, sign_a);
80     int rx2 = x;
81
82     if (lx2 > rx2)
83         swap(lx2, rx2);
84     int lx = max(lx1, lx2);
85     int rx = min(rx1, rx2);
86
87     if (lx > rx)
88         return 0;
89     return (rx - lx) / abs(b) + 1;
90 }
91 /*

```

```

92  ## finding the number of solutions and the solutions themselves in a given interval
93  Once we have lx and rx.
94  Just need to iterate through
95   $x = lx + k * (b / g)$  for all  $k \geq 0$  until  $x = rx$ 
96  and find the corresponding y values using the equation  $a.x + b.y = c$ 
97  */
98
99  /*
100  ## Find the solution with minimum value of  $x + y$ 
101  Here x and y also need to be given some restriction, otherwise, the answer may
    become negative infinity.
102  1. Find any solution (x , y) for the equations.
103
104  minimum value =  $x + y + k * (b - a) / g$ 
105
106  if( $a < b$ ) select smallest possible value of k
107  if( $a > b$ ) select the largest possible value of k
108  if( $a == b$ ) all solution will have the same sum  $x + y$ .

```

binary_gcd.cpp

```

1  int gcd(int a, int b) {
2      if (!a || !b)
3          return a | b;
4      unsigned shift = __builtin_ctz(a | b);
5      a >>= __builtin_ctz(a);
6      do {
7          b >>= __builtin_ctz(b);
8          if (a > b)
9              swap(a, b);
10         b -= a;
11     } while (b);
12     return a << shift;
13 }

```

discrete_log.cpp

```

1  int powmod(int a, int b, int m) {
2      int res = 1;
3      while (b > 0) {

```

```

4     if (b & 1) {
5         res = (res * 1ll * a) % m;
6     }
7     a = (a * 1ll * a) % m;
8     b >>= 1;
9 }
10 return res;
11 }
12
13 int solve(int a, int b, int m) {
14     a %= m, b %= m;
15     int n = sqrt(m) + 1;
16     map<int, int> vals;
17     for (int p = 1; p <= n; ++p)
18         vals[powmod(a, p * n, m)] = p;
19     for (int q = 0; q <= n; ++q) {
20         int cur = (powmod(a, q, m) * 1ll * b) % m;
21         if (vals.count(cur)) {
22             int ans = vals[cur] * n - q;
23             return ans;
24         }
25     }
26     return -1;
27 }

```

extended_euclidean.cpp

```

1 // a * x + b * y = gcd(a, b)
2 int gcd(int a, int b, int& x, int& y) {
3     if (b == 0) {
4         x = 1;
5         y = 0;
6         return a;
7     }
8     int x1, y1;
9     int d = gcd(b, a % b, x1, y1);
10    x = y1;
11    y = x1 - y1 * (a / b);
12    return d;
13 }

```

gcd.cpp

```

1  int gcd(int a, int b) {
2      while (b) {
3          a %= b;
4          swap(a, b);
5      }
6      return a;
7  }

```

gcd_negative_integer.cpp

```

1  int gcd(int a, int b, int& x, int& y) {
2      x = 1, y = 0;
3      int x1 = 0, y1 = 1, a1 = a, b1 = b;
4      while (b1) {
5          int q = a1 / b1;
6          tie(x, x1) = make_tuple(x1, x - q * x1);
7          tie(y, y1) = make_tuple(y1, y - q * y1);
8          tie(a1, b1) = make_tuple(b1, a1 - q * b1);
9      }
10     return a1;
11 }

```

mobius.cpp

```

1  /*
2   mobius Function (m)
3   if i has a squared factor : m(i) = 0
4   else m(i) = (-1)^r , r : number of distinct prime the i has
5  */
6  const int N = 1e5;
7  int mobius[N], sieve[N];
8  void gen_mobius(){
9      for(int i = 1; i < N; i++) {mobius[i] = sieve[i] = 1;}
10     sieve[1] = 0;
11     for(long long i = 2; i < N; i++){
12         if(sieve[i]){
13             for(long long j = i; j < N; j += i){
14                 sieve[j] = 0;

```



```

15     mobius[j] = (j % (i * i) == 0) ? 0 : -mobius[j];
16 }
17 }
18 }
19 }

```

modularMultiplicativeInverse.cpp

```

1  /*
2  ## Extended Euclidean algorithms
3  Modular multiplicative inverse when M and A are coprime or gcd(A, M) = 1:
4
5  Time Complexity: O(log M)
6  Auxiliary Space: O(1)
7  */
8  ll gcd(ll a, ll b, ll& x, ll& y) {
9      x = 1, y = 0;
10     ll x1 = 0, y1 = 1, a1 = a, b1 = b;
11     while (b1) {
12         int q = a1 / b1;
13         tie(x, x1) = make_tuple(x1, x - q * x1);
14         tie(y, y1) = make_tuple(y1, y - q * y1);
15         tie(a1, b1) = make_tuple(b1, a1 - q * b1);
16     }
17     return a1;
18 }
19 ll inv(ll A, ll M){
20     // modular inverse of A mod M
21     ll x, y;
22     ll g = gcd(A, M, x, y);
23     if(g != 1){ // Inverse doesn't exist
24         exit(3);
25     }
26     ll res = (x % M + M) % M;
27     return res;
28 }
29
30 //-----
31 /*
32 ## Fermat's little theorem
33 Modular multiplicative inverse when M is prime:
34 Time Complexity: O(log M)
35 Auxiliary Space: O(1)

```

```

36  */
37  ll bpw(ll a , ll b, ll mod){
38      ll res = 1;
39      a %= mod; // avoid overflow from a * a
40      while(b){
41          if(b % 2) res = (res * a) % mod;
42          b /= 2;
43          a = (a * a) % mod;
44      }
45  }
46  ll inv(ll N, ll M){
47      return bpw(N , M - 2);
48  }

```

optimized_sieve_LPF.cpp

```

1  vector<int> prime, lpf;
2  void Sieve(int n) {
3      prime.clear();
4      lpf.assign(n + 1, 0);
5
6      lpf[1] = 1;
7      for (int i = 2; i <= n; i++) {
8          if (lpf[i] == 0) {
9              lpf[i] = i;
10             prime.push_back(i);
11         }
12         for (int j = 0; j < (int) prime.size() && i * prime[j] <= n; j++) {
13             lpf[i * prime[j]] = prime[j];
14             if (prime[j] == lpf[i]) break;
15         }
16     }
17 }

```

Notes

Searching

counting_sort.cpp

```

1  /*
2      --Counting Sorting--
3      use:- It's an algothim to sort the array using bookkeeping array aka(frequency array)
4      Time Complexity:- O(max(a_1 , a_2 ,... , n));
5  */
6  #include <bits/stdc++.h>
7  using namespace std;
8  typedef long long ll;
9  #define rep(i , st , ed) for(int i = st; i < ed; i++)
10 #define f first
11 #define s second
12 const int N = 1e5 + 9;
13 int fre[N];
14 int main(){
15     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
16     int n; cin >> n;
17     int a[n] , b[n];
18     for(int i = 0; i < n; ++i) cin >> a[i] , fre[a[i]]++;
19     // counting-sorting
20     int idx = 0;
21     for(int i = 0; i < N; ++i) for(int j = 0; j < fre[i]; j++)
22         b[idx++] = i;
23     // b >> contains array a after sorting
24 }

```

double_ternary_search.cpp

```

1  long double l = 0 , r = 1e9 , mid;
2  int cnt = 400;
3  while(cnt--){
4      double g = l + (r - l) / 3,
5          h = r - (r - l) / 3;
6      if(f(g) < f(h)) r = h; // get minumum value
7      else l = g;
8  }

```

merge_sort.cpp

```

1  /*
2      --Merge sort--
3      Use: sorting the array

```

```

4   Time Complexity: O(n log n)
5   Space Complexity: O(n)
6   */
7   #include <bits/stdc++.h>
8   using namespace std;
9   typedef long long ll;
10  #define rep(i , st , ed) for(int i = st; i < ed; i++)
11  #define f first
12  #define s second
13  const int N = 1e5 + 1;
14  int a[N] , v1[N] , v2[N];
15  void mergeSort(int l , int r){
16      int mid = (l + r) / 2;
17      int n = mid - l , m = r - mid;
18      for(int i = l; i < mid; ++i) v1[i - l] = a[i];
19      for(int i = mid; i < r; ++i) v2[i - mid] = a[i];
20      int i = l , idx1 = 0 , idx2 = 0;
21      while(idx1 < n && idx2 < m){
22          if(v1[idx1] <= v2[idx2]) a[i++] = v1[idx1++];
23          else a[i++] = v2[idx2++];
24      }
25      while(idx1 < n) a[i++] = v1[idx1++];
26      while(idx2 < m) a[i++] = v2[idx2++];
27  }
28  void merge(int l , int r){
29      if(r - l == 1) return; // interval of length 1
30      int mid = (l + r) / 2; // [l , mid[ , [mid , r[
31      merge(l , mid);
32      merge(mid , r);
33      mergeSort(l , r);
34  }
35  int main(){
36      ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
37      #ifndef ONLINE_JUDGE
38      freopen("in.txt" , "r" , stdin);
39      freopen("out.txt" , "w" , stdout);
40      freopen("error.txt" , "w" , stderr);
41      #endif
42      int n; cin >> n;
43      for(int i = 0; i < n; ++i) cin >> a[i];
44      merge(0 , n);
45      for(int i = 0; i < n; ++i) cout << a[i] << " "; // array after sorted
46  }

```

minNumberOfSwapsToSortTwoBinaryString.cpp

```

1  #include <ext/pb_ds/assoc_container.hpp> // Common file
2  #include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update
3  using namespace __gnu_pbds;
4  template<class T> using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
   tree_order_statistics_node_update>;
5  int solve(string &s1, string &s2){
6      if(s1.size() != s2.size() || count(s1.begin(), s1.end(), '1') != count(s2.begin(), s2.end(), '1')){
7          return INT32_MAX; // Two strings can't be equal
8      }
9      ordered_set<int> pos[2];
10     for(int i = 0; i < s1.size(); ++i){
11         pos[s1[i] == '1'].insert(i);
12     }
13     ll ans = 0;
14     for(auto &ch : s2){
15         int f = (ch == '1');
16         assert(pos[f].size());
17         int i = *pos[f].find_by_order(0);
18         int ope = pos[f ^ 1].order_of_key(i);
19         ans += ope;
20         pos[f].erase(pos[f].lower_bound(i - 1));
21     }
22     return ans;
23 }

```

patient_sort.cpp

```

1  void LIS(vector<int> &v, vector<int> &ans){
2      vector<int> lis(v.size());
3      int cnt = 0;
4      for (int i = 0; i < v.size(); ++i) {
5          int pos = lower_bound(lis.begin(), lis.begin() + cnt, v[i]) - lis.begin();
6          lis[pos] = v[i];
7          if(pos == cnt) cnt++;
8          ans[i] = pos + 1;
9      }
10 }

```

radix_sort.cpp

```

1  vector<int> radix_sort(vector<int> v){
2      int n = v.size();
3      const int MAX = 16;
4      ll p10 = 1;
5      for(int i = 0; i < MAX; ++i){
6          vector<int> f(10) , tmp(n);
7          for(auto &x : v) f[x / p10 % 10]++;
8          for(int i = 1; i < 10; ++i) f[i] += f[i - 1];
9          for(int i = n - 1; i >= 0; --i){
10             tmp[ --f[v[i] / p10 % 10] ] = v[i];
11         }
12
13         swap(v , tmp);
14         p10 *= 10;
15
16     }
17     return v;
18 }
```

Snippet

clock.cpp

```

1  // Give the running time of code in ms.
2  ll get_time(){ return 1000 * clock() / CLOCKS_PER_SEC; }
```

fastCode.cpp

```

1  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
```

gcc_optimize.cpp

```

1  // reduce the running time
2  #pragma GCC optimize("Ofast")
3  #pragma GCC target("avx2,bmi,bmi2,popcnt,lzcnt")
4  #pragma GCC optimize("Ofast,unroll-loops")
```

```

5  #pragma GCC target("avx2")
6  // runtime errors with overflow
7  #pragma GCC optimize("trapv")
8
9
10
11 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
12 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
13 #pragma GCC target("avx,avx2,fma")
14 #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,fma")

```

Standard Problems

Geometry

check_point_in_convex.cpp

```

1  /*
2   Standard Problem: https://codeforces.com/gym/104968/problem/H
3   Given a convex polygon
4   Given a n point check if this point inside a polygon or not in O(log n)
5   by pre-processing O(n)
6  */
7  #include <bits/stdc++.h>
8  using namespace std;
9  #define double long double
10 #define int long long
11 #define vec(a,b) {b.x-a.x,b.y-a.y}
12 long long const N = 1e5;
13 struct Point { int x, y; };
14 double const eps = 1e-7;
15 int orientation(Point a, Point b, Point c) {
16     double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
17     if (v < 0) return -1; // clockwise
18     if (v > 0) return +1; // counter-clockwise
19     return 0;
20 }
21
22
23 bool cw(Point a, Point b, Point c, bool include_collinear) {
24     int o = orientation(a, b, c);
25     return o < 0 || (include_collinear && o == 0);

```

```

26 }
27 bool collinear(Point a, Point b, Point c) { return orientation(a, b, c) == 0; }
28
29 void convex_hull(vector<Point>& a, bool include_collinear = false) {
30     Point p0 = *min_element(a.begin(), a.end(), [](Point a, Point b) {
31         return make_pair(a.y, a.x) < make_pair(b.y, b.x);
32     });
33     sort(a.begin(), a.end(), [&p0](const Point& a, const Point& b) {
34         int o = orientation(p0, a, b);
35         if (o == 0)
36             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
37                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
38         return o < 0;
39     });
40     if (include_collinear) {
41         int i = (int)a.size()-1;
42         while (i >= 0 && collinear(p0, a[i], a.back())) i--;
43         reverse(a.begin()+i+1, a.end());
44     }
45
46     vector<Point> st;
47     for (int i = 0; i < (int)a.size(); i++) {
48         while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
49             st.pop_back();
50         st.push_back(a[i]);
51     }
52
53     a = st;
54 }
55 double cross(Point a, Point b){ return a.x*b.y - b.x*a.y; }
56 int cross(Point a, Point b, Point c){
57     Point d = vec(b,a), e = vec(c,a);
58     return cross(d,e);
59 }
60 double dot(Point a, Point b){
61     return a.x*b.x + a.y*b.y;
62 }
63 bool pointOnLine(Point a, Point b, Point p){
64     return fabs(cross(vec(a,b),vec(a,p))) < eps;
65 }
66 bool pointOnRay(Point a, Point b, Point p){
67     return dot(vec(a,p),vec(a,b)) > -eps;
68 }
69 bool pointOnsegment(Point a, Point b, Point p){

```



```

70     if(!pointOnLine(a,b,p)) return 0;
71     return pointOnRay(a,b,p) && pointOnRay(b,a,p);
72 }
73 vector<Point> v(N);
74 void prepare(int n) {
75     int pos = 0;
76     for (int i = 0; i < n; i++) {
77         if (make_pair(v[i].x, v[i].y) < make_pair(v[pos].x, v[pos].y))
78             pos = i;
79     }
80     rotate(v.begin(), v.begin() + pos, v.end());
81 }
82 void print(Point p){
83     cerr << p.x << " " << p.y << "\n";
84 }
85 bool fun(Point p,int n)
86 {
87     int idx = 1;
88     if(pointOnsegment(v[0],v[n-1],p)) return 1;
89     if(cross(v[0],v[n-1],p) < 0) return 0;
90     if(cross(v[0],v[1],p) > 0) return 0;
91     int l = 1, r = n - 2;
92     int ans = -1;
93     while(r >= l)
94     {
95         int mid = l + (r-l)/2;
96         if(cross(v[0],v[mid],p) <= 0) l = mid+1, ans = mid;
97         else r=mid-1;
98     }
99     if(ans == -1) return 0;
100     return (cross(v[ans],v[(ans+1)%n],p) <= 0);
101 }
102 double area(Point a, Point b, Point c)
103 {
104     return 0.5*abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
105 }
106 void solve()
107 {
108     int n,m; cin >> n >> m;
109     vector<Point> vans(n);
110     for(int i=0; i < n; ++i)
111     {
112         cin >> vans[i].x >> vans[i].y;

```

```

113     }
114     for(int i=0; i < m; ++i) cin >> v[i].x >> v[i].y;
115     convex_hull(v);
116     prepare(v.size());
117     m = v.size();
118     int ans = 0;
119     for(int i=0; i < n; ++i) ans += fun(vans[i],m);
120     cout << ans << "\n";
121 }
122 int32_t main()
123 {
124     int t=1; //cin >> t;
125     while(t--)
126     {
127         solve();

```

count_points_in_circle.cpp

```

1  /*
2  Problem link: https://atcoder.jp/contests/abc191/tasks/abc191\_d
3  */
4  #include <bits/stdc++.h>
5  using namespace std;
6  typedef long long ll;
7  #define rep(i , st , ed) for(int i = st; i < ed; i++)
8  #define f first
9  #define s second
10 int EPS = 10000 , Precision = 4;
11 long long read(){
12     // 1.2345 --> 12345
13     string s;
14     cin >> s;
15     int x=s.find('.');
16     if(x==-1) return stoll(s+"0000");
17     string one=s.substr(0,x);
18     string two=s.substr(x+1);
19     while(two.size()< Precision) two+="0";
20     return stoll(one+two);
21 }
22 ll ceil(ll a, ll b){
23     if(b < 0) a *= -1, b *= -1;

```

```

24     if(a < 0) return a / b;
25     else return (a + b - 1) / b;
26 }
27
28 ll floor(ll a, ll b){
29     if(b < 0) a *= -1, b *= -1;
30     if(a > 0) return a / b;
31     else return (a - b + 1) / b;
32 }
33 ll BS(ll x, ll z){
34     // find maximum y : y ^ 2 <= z ^ 2 - x ^ 2
35     ll l = 0, r = z, mid;
36     while(l < r){
37         mid = l + (r - l + 1) / 2;
38         if(mid * mid <= z * z - x * x) l = mid;
39         else r = mid - 1;
40     }
41     return l;
42 }
43 int main(){
44     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
45     #ifndef ONLINE_JUDGE
46     freopen("in.txt", "r", stdin);
47     freopen("out.txt", "w", stdout);
48     freopen("error.txt", "w", stderr);
49     #endif
50     ll x = read(), y = read(), r = read();
51     ll l_x = floor(x - r, EPS) * EPS;
52     ll r_x = ceil(x + r, EPS) * EPS;
53     ll cnt = 0;
54     for(ll i = l_x; i <= r_x; i += EPS){
55         if(abs(i - x) > r) continue;
56         ll len = BS(abs(x - i), r);
57         ll l_y = ceil(y - len, EPS);
58         ll r_y = floor(y + len, EPS);
59         if(r_y >= l_y) cnt += r_y - l_y + 1;
60     }
61     cout << cnt;
62 }

```

point_in-shape.cpp

```

1  /*
2   Problem Link: https://codeforces.com/gym/104447/problem/H
3   You are given a polygon of n, vertices and q queries.
4
5   Each query consists of a point (x,y)
6   and you have to check if it is inside (including the borders) or outside the polygon.
7
8   The points of the polygon are given in clockwise order with the property that either
9    $x_i = x_{i-1}$ 
10  or  $y_i = y_{i-1}$ 
11  (but not both), indicating that the edges are parallel to either the x- or y-axis, Also no
12  two edges intersect
13  (endpoints are not considered into the intersections).
14
15  The first line of the input contains an integer n ( $4 \leq n \leq 1e5$ ), representing the number of
16  vertices in the polygon.
17
18  The next n lines each contain two integers  $x_i$  and  $y_i$  ( $0 \leq x_i, y_i \leq 10^6$ ), representing the x-
19  and y-coordinates, respectively, of the i
20  -th vertex of the polygon in clockwise order.
21
22  The next q lines each contain two integers x and y ( $0 \leq x, y \leq 1e6$ ), representing the x-
23  and y-coordinates,
24  respectively, of a point to be checked whether it is inside or outside the polygon.
25  */
26
27 #include<iostream>
28 #include <bits/stdc++.h>
29
30 #define ll long long
31 #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
32 using namespace std;
33 const int N = 1e6 + 5, mod = 1e9 + 7, M = 17, inf = 2e9, sq = 632;
34 vector<vector<int>> > x[N], y[N], queries[N];
35
36 bool inside(vector<vector<int>> > &v, int val) {
37     int low = 0, high = v.size() - 1;
38     while (low <= high) {
39         int mid = low + high >> 1;
40         if (v[mid][1] < val) {
41             low = mid + 1;
42         } else if (v[mid][0] > val) {
43             high = mid - 1;
44         } else {
45             return true;
46         }
47     }
48 }

```

```

40     }
41     return false;
42 }
43
44 int bit[N];
45
46 void add(int idx, int val) {
47     for (; idx < N; idx += idx & -idx) {
48         bit[idx] += val;
49     }
50 }
51
52 void add_range(int l, int r) {
53     if (l > r) return;
54     add(l, 1);
55     add(r + 1, -1);
56 }
57
58 int query(int idx) {
59     int ans = 0;
60     for (; idx; idx -= idx & -idx) {
61         ans += bit[idx];
62     }
63     return ans;
64 }
65
66 map<int, int> freq[2];
67
68 void doWork() {
69     int n;
70     cin >> n;
71     vector<pair<int, int> > v(n);
72     for (int i = 0; i < n; i++) {
73         cin >> v[i].first >> v[i].second;
74         v[i].second++;
75     }
76     for (int i = 0; i < n; i++) {
77         if (v[i].first == v[(i + 1) % n].first) {
78             int y1 = v[i].second, y2 = v[(i + 1) % n].second;
79             x[v[i].first].push_back({min(y1, y2), max(y1, y2), y1 < y2});
80         } else {
81             int x1 = v[i].first, x2 = v[(i + 1) % n].first;
82             y[v[i].second].push_back({min(x1, x2), max(x1, x2)});

```

```

83     }
84 }
85 for (int i = 0; i < N; i++) {
86     sort(x[i].begin(), x[i].end());
87     sort(y[i].begin(), y[i].end());
88 }
89 int q;
90 cin >> q;
91 vector<int> ans(q, 0);
92 for (int i = 0; i < q; i++) {
93     int a, b;
94     cin >> a >> b;
95     b++;
96     if (inside(x[a], b) || inside(y[b], a)) {
97         ans[i] = true;
98     } else {
99         queries[a].push_back({b, i});
100     }
101 }
102 for (int i = 0; i < N; i++) {
103     for (auto j: x[i]) {
104         add_range(j[0] + 1, j[1] - 1);
105         freq[j[2]][j[0]]++;
106         freq[j[2]][j[1]]++;
107     }
108     for (auto j: queries[i]) {
109         int cnt = query(j[0]);
110         int cntUP = freq[1][j[0]];
111         int cntDown = freq[0][j[0]];
112         cnt += (max(cntDown, cntUP) - min(cntDown, cntUP)) / 2;
113         ans[j[1]] = cnt % 2;
114     }
115 }
116 for (auto i: ans) {
117     cout << (i ? "YES\n" : "NO\n");
118 }
119 }
120
121
122 int main() {
123     IO
124     int t = 1;
125     // cin >> t;

```

```

126     for (int i = 1; i <= t; i++) {
127         // cout << "Case #" << i << ": ";

```

Implementation

coloring_matrix_with_largest_area.cpp

```

1  /*
2      Problem Link:- https://codeforces.com/gym/104874/problem/K
3      Given a matrix n x m, each cell (i, j) can be
4      1. g(i, j) = '.' -> empty cell
5      2. g(i, j) = 'A' king , "There is exactly one letter 'A'"
6      3. g(i, j) = child represented as character between 'z' -> 'z' , it's proved that letters
       are distinct.
7      Output the same matrix, replacing each character '.' with the lowercase letter,
8      corresponding to the owner of the containing this cell
9      "where the area of the king that belongs to his favorite child is as large as possible."
10     Test Cases:
11     Input:
12         6 8
13         .....X.
14         .F.....
15         ...A....
16         .....
17         .....P..
18         ..L.....
19     Output:
20         xxxxxxXx
21         fFaaaaaa
22         ffaAaaaa
23         ffaaaaaa
24         llllPpp
25         llLlppp
26  */
27  #include <bits/stdc++.h>
28  using namespace std;
29  typedef long long ll;
30  #define rep(i, st, ed) for(int i = st; i < ed; i++)
31  #define f first
32  #define s second
33  const int N = 1001;

```

```

34 int n , m , curl , curJ;
35 pair<int, int> L, R; // Diamtions of Maxiumum area
36 bool islower(char ch){ return ch >= 'a' && ch <= 'z'; }
37 bool validX(int x){ return x >= 0 && x < n; }
38 bool validY(int y){ return y >= 0 && y < m; }
39 void solveRange(int x1, int y1, int x2, int y2, vector<vector<char>> &g) {
40     if (!validX(x1) || !validX(x2) || !validY(y1) || !validY(y2))
41         return;
42     for (int i = x1; i <= x2; i++) {
43         int last = y1 - 1;
44         char lastC = '.';
45         for (int j = y1; j <= y2; j++) {
46             if (g[i][j] != '.') {
47                 char cur = g[i][j] - 'A' + 'a';
48                 for (int temp = j - 1; temp > last; temp--) {
49                     g[i][temp] = cur;
50                 }
51                 last = j;
52                 lastC = cur;
53             }
54         }
55         if (lastC != '.') {
56             for (int j = y2; j > last; j--) {
57                 g[i][j] = lastC;
58             }
59         } else {
60             if (i - 1 >= x1 && g[i - 1][y1] != '.') {
61                 for (int j = y1; j <= y2; j++) {
62                     g[i][j] = tolower(g[i - 1][j]);
63                 }
64             }
65         }
66     }
67     for (int i = x2; i >= x1; i--) {
68         if (g[i][y1] == '.') {
69             assert(i != x2);
70             for (int j = y1; j <= y2; j++) {
71                 g[i][j] = tolower(g[i + 1][j]);
72             }
73         }
74     }
75 }
76 void paint(int x1, int y1, int x2, int y2, char c , vector<vector<char>> &g) {
77     for (int i = x1; i <= x2; i++) {

```



```

78     for (int j = y1; j <= y2; j++) {
79         g[i][j] = c;
80     }
81 }
82 }
83 void zero_matrix(vector<vector<char>> a) {
84     n = a.size();
85     m = a[0].size();
86     vector<int> d(m, -1), d1(m), d2(m);
87     stack<int> st;
88     int mx = 0;
89     for (int i = 0; i < n; i++) {
90         for (int j = 0; j < m; j++) {
91             if (a[i][j] != '.') {
92                 d[j] = i;
93             }
94         }
95         for (int j = 0; j < m; j++) {
96             while (!st.empty() && d[st.top()] <= d[j]) {
97                 st.pop();
98             }
99             d1[j] = st.empty() ? -1 : st.top();
100            st.push(j);
101        }
102        while (!st.empty()) {
103            st.pop();
104        }
105        for (int j = m - 1; j >= 0; j--) {
106            while (!st.empty() && d[st.top()] <= d[j]) {
107                st.pop();
108            }
109            d2[j] = st.empty() ? m : st.top();
110            st.push(j);
111        }
112        while (!st.empty()) {
113            st.pop();
114        }
115        for (int j = 0; j < m; j++) {
116            pair<int, int> tempL = {d[j] + 1, d1[j] + 1};
117            pair<int, int> tempR = {i, d2[j] - 1};
118            int area = (tempR.first - tempL.first + 1) * (tempR.second - tempL.second + 1);
119            if (curl >= tempL.first && curl <= tempR.first && curJ >= tempL.second && curJ <=
tempR.second &&

```

```

120         area >= mx) {
121             L = tempL;
122             R = tempR;
123             mx = area;
124         }
125     }
126 }
127 }
128
129 int main(){
130     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
131     #ifndef ONLINE_JUDGE
132     freopen("in.txt", "r", stdin);
133     freopen("out.txt", "w", stdout);
134     freopen("error.txt", "w", stderr);
135     #endif
136     cin >> n >> m;
137     vector<vector<char>> g(n, vector<char>(m));
138     int st_x, st_y;
139     rep(i, 0, n) rep(j, 0, m){
140         cin >> g[i][j];
141         if(g[i][j] == 'A') curl = i, curJ = j, g[i][j] = '.';
142     }
143
144     g[curl][curJ] = '.';
145     zero_matrix(g);
146     paint(L.first, L.second, R.first, R.second, char('a'), g);
147     g[curl][curJ] = char('A');
148
149
150     int x1 = L.first, y1 = L.second, x2 = R.first, y2 = R.second;
151     solveRange(0, 0, x1 - 1, m - 1, g);
152     solveRange(x1, 0, x2, y1 - 1, g);
153     solveRange(x2 + 1, 0, n - 1, m - 1, g);
154     solveRange(x1, y2 + 1, x2, m - 1, g);
155
156     rep(i, 0, n){
157         rep(j, 0, m){
158             cout << g[i][j];
159         }
160         cout << '\n';

```

divide_grid_into_k_connctedcompented_01.cpp

```

1  /*
2  Problem Statement:
3  Given a rectangular grid of size  $n \times m$ , create a map of '0' and '1' terrain types where:
4  1. Only two terrain types are allowed ('0' and '1')
5  2. There must be exactly k connected components
6     - Connected components are adjacent cells of same value
7     - Cells are adjacent if they share a side
8
9  Input: Three integers n, m, k where:
10 -  $1 \leq n \leq 1000$  (rows)
11 -  $1 \leq m \leq 1000$  (columns)
12 -  $1 \leq k \leq n \times m$  (desired components)
13
14 Output:
15 - "YES" + grid solution if possible
16 - "NO" if impossible
17 */
18 void solve() {
19     int n, m, k;
20     cin >> n >> m >> k;
21     int a[n][m];
22
23     // Case 1: One-dimensional grid (single row or column)
24     if (n == 1 || m == 1) {
25         cout << "YES\n";
26         int num = 0;
27         for (int i = 0; i < n; i++) {
28             for (int j = 0; j < m; j++) {
29                 cout << num;
30                 if (k > 1) {
31                     num = 1 - num;
32                     k--;
33                 }
34             }
35             cout << '\n';
36         }
37         return;
38     }
39
40     // Case 2: Impossible case - when components would equal  $n*m-1$ 
41     if (k == n * m - 1) {

```

```

42     cout << "NO\n";
43     return;
44 }
45
46 // Case 3: All other cases
47 cout << "YES\n";
48 int num = 0;
49 int flag = 0;
50
51 // Adjust k if (k+1) is divisible by m
52 if ((k + 1) % m == 0) {
53     k--;
54     flag = 1;
55 }
56
57 // Fill the grid with alternating values until k components are created
58 int i, j;
59 for (i = 0; i < n; i++) {
60     for (j = 0; j < m; j++) {
61         a[i][j] = num;
62         num = 1 - num;
63         k--;
64         if (k <= 0) break;
65     }
66     if (k <= 0) break;
67     if (m % 2 == 0) num = 1 - num;
68 }
69
70 // Fill remaining cells in the current row
71 if (i == 0) {
72     for (int x = j + 1; x < m; x++) {
73         a[i][x] = 1 - num;
74     }
75 } else if (j + 1 < m) {
76     a[i][j + 1] = num;
77 }
78
79 // Fill remaining rows by copying values from above
80 for (int x = 0; x < m; x++) {
81     int st;
82     if (x <= j + 1 || i == 0) {
83         st = i;
84     } else {
85         st = i - 1;

```

```

86     }
87     for (int y = st + 1; y < n; y++) {
88         a[y][x] = a[st][x];
89     }
90 }
91
92 // Handle special case when (k+1) was divisible by m
93 if (flag) {
94     a[n - 1][m - 1] = 1 - a[n - 1][m - 2];
95 }
96
97 // Print the final grid
98 for (int i = 0; i < n; i++) {
99     for (int j = 0; j < m; j++) {
100         cout << a[i][j];
101     }
102     cout << '\n';
103 }

```

Number_Theory

count_no_subsequec_with_lcm.cpp

```

1  /*
2   Problem Link: https://atcoder.jp/contests/abc349/tasks/abc349\_f
3   Problem description:
4   You are given a sequence of positive integers
5   A=(A1, A2,... , An) of length N and a positive integer M. Find the number, modulo
6   998244353, of non-empty and not necessarily contiguous subsequences of
7   A such that the least common multiple (LCM) of the elements in the subsequence is
8   M. Two subsequences are distinguished if they are taken from different positions in the
9   sequence, even if they coincide as sequences. Also, the LCM of a sequence with a single
10  element is that element itself.
11
12  Constraints
13  1 ≤ N ≤ 2e5
14  1 ≤ M ≤ 1e16
15  1 ≤ Ai ≤ 1e16
16
17  My sol:
18  Time complexity:  $O(2^x \cdot 3)$  --> can be decreased to  $O(3^x)$  using submasking
19  method

```

```

16     such that x: no. of primes in M
17 */
18 #include <bits/stdc++.h>
19 using namespace std;
20 typedef long long ll;
21 #define rep(i , st , ed) for(int i = st; i < ed; i++)
22 #define f first
23 #define s second
24 const int mod = 998244353;
25 ll bpw(ll a , ll b){
26     ll res = 1;
27     while(b){
28         if(b % 2) res = res * a % mod;
29         b /= 2;
30         a = a * a % mod;
31     }
32     return res;
33 }
34 int main(){
35     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
36     #ifndef ONLINE_JUDGE
37     freopen("in.txt", "r", stdin);
38     freopen("out.txt", "w", stdout);
39     freopen("error.txt", "w", stderr);
40     #endif
41     ll n , m; cin >> n >> m;
42     vector<ll> dis;
43     ll M = m;
44     for(ll i = 2; i * i <= m; ++i) if(m % i == 0){
45         ll d = 1;
46         while(m % i == 0){ m /= i; d *= i; }
47         dis.emplace_back(d);
48     }
49     if(m > 1) dis.emplace_back(m);
50     m = (int) dis.size();
51     int cnt[1 << m]{} , one = 0;
52     for(int i = 0; i < n; ++i){
53         ll x; cin >> x;
54         if(__gcd(x , M) != x) continue;
55         ll mask = 0;
56         for(int j = 0; j < m; ++j) if(x % dis[j] == 0)
57             mask |= (1 << j);
58         one += (x == 1);
59         cnt[mask]++;

```

```

1 // problem link: https://codeforces.com/contest/803/problem/F
2 #include <bits/stdc++.h>
3 typedef long long ll;
4 #define s second
5 #define f first
6 #define add(a , b) a = (a + b + mod) % mod
7 #define rep(i , st , ed) for(int i = st ; i < ed ; i++)
8 using namespace std;
9 void burn(){
10 ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
11 #ifndef ONLINE_JUDGE
12     freopen("in.txt" , "r" , stdin);
13     freopen("out.txt" , "w" , stdout);
14     freopen("error.txt" , "w" , stderr);
15 #endif
16 }
17 //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
18 const int N = 100009 , mod = 1e9 + 7;
19 ll mobius[N] , sieve[N] , g[N] , fre[N];
20 void gen_mobius(){
21     for(int i = 1; i < N; i++) {mobius[i] = sieve[i] = 1;}
22     sieve[1] = 0;
23     for(long long i = 2; i < N; i++){
24         if(sieve[i]){
25             for(long long j = i; j < N; j += i){
26                 sieve[j] = 0;

```

```

27     mobius[j] = (j % (i * i) == 0) ? 0 : -mobius[j];
28 }
29 }
30 }
31 }
32 ll bin_exp(ll a , ll b){
33     ll ret = 1;
34     while(b){
35         if(b % 2) ret = (ret * a) % mod;
36         a = (a * a) % mod;
37         b /= 2;
38     }
39     return ret;
40 }
41 int main(){
42     burn();
43     ll n; cin >> n;
44     rep(i , 0 , n){
45         int x; cin >> x;
46         fre[x]++;
47     }
48     gen_mobius();
49     for(int i = 2; i < N; i++){
50         for(int j = i; j < N; j += i){
51             g[i] += fre[j];
52         }
53     }
54     ll ans = (bin_exp(2 , n) - 1 + mod) % mod;
55     // count number of subsequence the gcd > 1
56     for(int i = 2; i < N; i++){
57         add(ans, mobius[i] * ((bin_exp(2 , g[i]) - 1 + mod) % mod) % mod);
58     }
59     cout << ans;
60 }

```

Tree

No_of_paths_insideways_paths.cpp

```

1  /*
2  Problem link: https://codeforces.com/group/Rilx5irOux/contest/564406/problem/H
3  Problem Statement:

```



```

4   - Given n towns connected by n-1 roads forming a tree
5   - m trade agreements between towns
6   - Each agreement affects all towns on shortest path between si and ti
7   - Towns u,v can trade if shortest path between them covered by an agreement
8   - Must count valid trading pairs where u<v
9
10  Input:
11  - n, m (2≤n,m≤105)
12  - n-1 lines: xi,yi road connections
13  - m lines: si,ti trade agreement paths
14
15  Output:
16  - Number of valid trading pairs
17  */
18  #include <bits/stdc++.h>
19  using namespace std;
20  typedef long long ll;
21  #define f first
22  #define s second
23  const int N = 1e5 + 9; // TODO: change it to maximum possible N
24  const int LOG = 17;
25  int dfs_time = 0, cur_pos = 0;
26  int st[N], ft[N], big[N], ver[N], sz[N];
27  int head[N], par[N], dep[N], up[N][LOG];
28  int n, m, bst[N];
29  ll ans;
30  vector<int> adj[N], paths[N];
31
32  // Lazy segment tree
33  struct node{
34      int mn, cnt;
35      node(int x){ mn = x; cnt = 1;}
36      node(){};
37      node operator + (const node other) const{
38          node res;
39          res.mn = min(mn, other.mn);
40          if(mn < other.mn) res.cnt = cnt;
41          else if(other.mn < mn) res.cnt = other.cnt;
42          else res.cnt = cnt + other.cnt;
43          return res;
44      }
45
46  } tree[4*N];
47  int lazy[4*N];

```

```

48 void push_down(int x , int par){
49     lazy[x] += lazy[par];
50     tree[x].mn += lazy[par];
51 }
52 void propogate(int x , int l , int r){
53     if(r - l == 1) return;
54     push_down(2*x+1,x);
55     push_down(2*x+2,x);
56     lazy[x] = 0;
57 }
58 void build(int x = 0 , int l = 0 , int r = n){
59     if(r - l == 1){ tree[x] = node(0); return; }
60     int mid = (l+r)/2;
61     build(2*x+1,l,mid);
62     build(2*x+2,mid,r);
63     tree[x] = tree[2*x+1] + tree[2*x+2];
64 }
65 void upd(int lx , int rx , int v, int x = 0 , int l = 0 , int r = n){
66     propogate(x,l,r);
67     if(l >= lx && r <= rx){
68         tree[x].mn += v;
69         lazy[x] = v;
70         return;
71     }
72     if(r <= lx || l >= rx) return;
73     int mid = (l+r)/2;
74     upd(lx,rx,v,2*x+1,l,mid);
75     upd(lx,rx,v,2*x+2,mid,r);
76     tree[x] = tree[2*x+1] + tree[2*x+2];
77 }
78 node qry(int lx , int rx , int x = 0 , int l = 0 , int r = n){
79     propogate(x,l,r);
80     if(l >= lx && r <= rx) return tree[x];
81     if(r <= lx || l >= rx) return node(2e9);
82     int mid = (l+r)/2;
83     return qry(lx,rx,2*x+1,l,mid) + qry(lx,rx,2*x+2,mid,r);
84 }
85 int countZero(int l , int r){
86     if(l > r) return 0;
87     auto ans = qry(l,r+1);
88     if(ans.mn) return 0;
89     // cerr << "\n" << "Min: " << ans.mn << " cnt: " << ans.cnt << " sz: " << r - l + 1 << '\n';
90     return ans.cnt;

```

```

91  }
92
93  // HLD
94  void preDFS(int u = 0, int p = 0){
95      sz[u] = 1, big[u] = -1;
96
97      // Build LCA
98      dep[u] = dep[p] + 1;
99      up[u][0] = p;
100     par[u] = p;
101     for(int x = 1; x < LOG; ++x){
102         up[u][x] = up[ up[u][x-1] ][x-1];
103     }
104
105     for(auto v : adj[u]) if(v != p){
106         preDFS(v,u);
107         sz[u] += sz[v];
108         if(big[u] == -1 || sz[v] > sz[ big[u] ]) big[u] = v;
109     }
110 }
111
112 void decomposition(int u = 0, int h = 0){
113     head[u] = h;
114     st[u] = dfs_time++;
115     // cerr << u + 1 << ' ';
116     ver[ st[u] ] = u;
117
118     if(~big[u])
119         decomposition(big[u], h);
120     for(auto &v : adj[u]) if(v != par[u] && v != big[u])
121         decomposition(v,v);
122     ft[u] = dfs_time;
123 }
124 void upd_path(int u , int v , int val){
125     int ans = 0;
126     for(; head[u] != head[v]; v = par[head[v]]){
127         if(dep[head[u]] > dep[head[v]]) swap(u,v);
128         // process interval: [ st[head[v]] , st[v] ]
129         upd(st[head[v]] , st[v] + 1, val);
130
131     }
132     if(dep[u] > dep[v]) swap(u,v);
133     // process interval: [ st[u] , st[v] ]

```

```

134     upd(st[u] , st[v] + 1, val);
135 }
136
137 // DSU on tree
138 void sackDFS(int u, int p, bool keep){
139     int bigChild = big[u];
140
141     // run a dfs on small childs
142     for(auto v : adj[u]){
143         if(v != p && v != bigChild) sackDFS(v, u, 0);
144     }
145
146     if(bigChild != -1) sackDFS(bigChild, u, 1); // bigChild marked as big and not cleared
    from cnt
147     for(auto v : adj[u]){
148         if(v != p && v != bigChild){
149             for(int p = st[v]; p < ft[v]; p++){
150                 // Add your information about ver[p]
151                 int x = ver[p];
152                 // cerr << "+" << x + 1 << '\n';
153                 for(auto &y : paths[x]) upd_path(x,y,+1);
154             }
155
156         }
157
158     }
159     // Add your information about u
160     // cerr << "+" << u + 1 << '\n';
161     for(auto &y : paths[u]) upd_path(u,y,+1);
162
163     // All information about the subtree of u is kept, and you can now query it.
164     // cerr << "u: " << u + 1 << " , ans: " << (n - ft[u]) - countZero(ft[u] , n-1) << '\n';
165     // cerr << "After: " << n - ft[u] << ' ' << " zero: " << countZero(ft[u] , n - 1) << '\n';
166     ans += (n - ft[u]) - countZero(ft[u] , n-1);
167     ans += dep[u] - bst[u];
168
169     if(keep == 0){
170         for(int p = st[u]; p < ft[u]; p++){
171             // Remove the added information about ver[p]
172             int x = ver[p];
173             // cerr << "-" << x+1 << '\n';
174             for(auto &y : paths[x]) upd_path(x,y,-1);
175         }

```

```

176     }
177
178 }
179
180 // Compute LCA
181 int getLCA(int x , int y){
182     if(dep[x] < dep[y]) swap(x,y);
183     int k = dep[x] - dep[y];
184     for(int i = 0; i < LOG; ++i) if((k>>i) & 1) x = up[x][i];
185     if(x == y) return x;
186
187     for(int i = LOG - 1; i >= 0; --i) if(up[x][i] != up[y][i]){
188         x = up[x][i];
189         y = up[y][i];
190     }
191
192     assert(up[x][0] == up[y][0]);
193     return up[x][0];
194 }
195
196 void _dfs(int u , int p){
197     for(auto &v : adj[u]) if(v != p){
198         _dfs(v,u);
199         bst[u] = min(bst[u] , bst[v]);
200     }
201 }
202 int main(){
203     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
204     #ifndef ONLINE_JUDGE
205     freopen("in.txt", "r", stdin);
206     freopen("out.txt", "w", stdout);
207     freopen("error.txt", "w", stderr);
208     #endif
209     cin >> n >> m;
210     for(int i = 1; i < n; ++i){
211         int u,v; cin >> u >> v;
212         --u; --v;
213         adj[u].emplace_back(v);
214         adj[v].emplace_back(u);
215     }
216
217     preDFS(0,0);

```

```

218     decomposition(0,0);
219     build();
220
221     for(int i = 0; i < n; ++i) bst[i] = dep[i];
222
223     for(int i = 0; i < m; ++i){
224         int x,y; cin >> x >> y;
225         --x; --y;
226         if(st[x] > st[y]) swap(x,y);
227         int lca = getLCA(x,y);
228         bst[x] = min(bst[x] , dep[lca]);
229         bst[y] = min(bst[y] , dep[lca]);
230         if(lca == x) continue;
231         paths[x].emplace_back(y);

```

String

Aho corasick

AhoCorasick.cpp

```

1  struct AhoCorasick {
2      // Modify these values based on input alphabet
3      // alpha: size of alphabet (26 for lowercase letters)
4      // first: first character of alphabet ('a' for lowercase letters)
5      enum { alpha = 26, first = 'a' };
6
7      struct Node {
8          int nxt[alpha];    // Next state transition for each character
9          int sufink;        // Suffix link points to longest proper suffix
10         int start = -1;    // Start index of pattern in original array
11         int end = -1;      // Index in backup of longest matched suffix pattern
12         int nmatches = 0;  // Count of matched strings ending at this node
13
14         Node(int v) {
15             memset(nxt, v, sizeof nxt);
16         }
17     };
18
19     vector<Node> v;        // Stores all nodes of the trie

```

```

20     vector<int> backup;    // Stores pattern indices with longest matching suffixes
21                             // Returns -1 if no match exists
22                             // Note: All patterns must be distinct when using backup
23
24     // Inserts a pattern into the automaton
25     // Time: O(|s|) where |s| is pattern length
26     void insert(string &s, int id) {
27         assert(s.size()); // Empty patterns not allowed
28         int node = 0;
29         for(auto &c : s) {
30             int &m = v[node].nxt[c - first];
31             if(m == -1) {
32                 node = m = v.size();
33                 v.emplace_back(-1);
34             }
35             else node = m;
36         }
37         if(v[node].end == -1) v[node].start = id;
38         backup.emplace_back(v[node].end);
39         v[node].end = id;
40         v[node].nmatches++;
41     }
42
43     // Builds Aho-Corasick automaton from patterns
44     // Time: O(26N) where N = sum of all pattern lengths
45     // - Creates suffix links
46     // - Allows duplicate patterns
47     // - For large alphabets, split symbols into chunks with sentinel bits
48     AhoCorasick(vector<string> &pat): v(1, -1) {
49         for(int i = 0; i < pat.size(); ++i)
50             insert(pat[i], i);
51
52         v[0].sufink = v.size(); // Dummy node as suffix link of root
53         v.emplace_back(0);
54
55         queue<int> q;
56         q.push(0);           // BFS from root to build suffix links
57
58         while(q.size()) {
59             int node = q.front();
60             q.pop();
61             int prv = v[node].sufink;
62
63             for(int i = 0; i < alpha; ++i) {

```

```

64         int &x = v[node].nxt[i], y = v[prv].nxt[i];
65         if(x == -1) x = y;
66         else {
67             v[x].suflink = y;
68             (v[x].end == -1 ? v[x].end : backup[v[x].start]) = v[y].end;
69             v[x].nmatches += v[y].nmatches;
70             q.push(x);
71         }
72     }
73 }
74 }
75
76 // Returns index of longest word ending at each position, or -1 if none
77 // Time: O(|word|) where |word| is text length
78 vector<int> find(string &word) {
79     int node = 0;
80     vector<int> res;
81     for(auto &c : word) {
82         node = v[node].nxt[c - first];
83         res.push_back(v[node].end);
84     }
85     return res;
86 }
87
88 // Finds all patterns starting at each position (shortest first)
89 // Time: O(NM) where N = text length, M = number of matches
90 // Can find up to N√N matches if no duplicate patterns
91 vector<vector<int>> findAll(vector<string> &pat, string word) {
92     vector<int> r = find(word);
93     vector<vector<int>> res(word.size());
94     for(int i = 0; i < word.size(); ++i) {
95         int ind = r[i];
96         while(ind != -1) {
97             res[i - pat[ind].size() + 1].push_back(ind);
98             ind = backup[ind];
99         }
100     }
101     return res;
102 }

```

Basic strings Algo

KMP.cpp

```

1  void KMP(string str, string pat)
2  {
3      int n = str.length();
4      int m = pat.length();
5      vector<int> longestPrefix = fail_fun(pat);
6
7      for(int i = 0, k = 0; i < n; i++) {
8          // as long as we can't add one more character in k, get best next prefix
9          while (k > 0 && pat[k] != str[i])
10             k = longestPrefix[k - 1];
11
12         // if we match character in the pattern, move in pattern
13         if (pat[k] == str[i])
14             k++;
15
16         // if we matched, print it and let's find one more matching
17         if (k == m) {
18             cout<<i - m + 1<<"\n";
19             k = longestPrefix[k - 1]; // fail to next best suffix
20         }
21     }
22 }
23
24 vector<int> fail_fun(string s) {
25     int n = (int)s.length();
26     vector<int> pi(n);
27     for (int i = 1; i < n; i++) {
28         int j = pi[i-1];
29         while (j > 0 && s[i] != s[j])
30             j = pi[j-1];
31         if (s[i] == s[j])
32             j++;
33         pi[i] = j;
34     }
35     return pi;
36 }

```

Z_algorithm.cpp

```

1  // Z-Function: Returns array z where z[i] is length of longest common prefix

```

```

2 // between s[0..n-1] and s[i..n-1]
3 // Time: O(n), Memory: O(n)
4 vector<int> zFunction(string s) {
5     int n = s.size();
6     vector<int> z(n);
7     int left = 0, right = 0;
8     for(int i = 1; i < n; i++) {
9         if(i <= right) {
10             z[i] = min(right - i + 1, z[i - left]);
11         }
12         while(i + z[i] < n && s[z[i]] == s[z[i] + i]) {
13             z[i]++;
14         }
15         if(i + z[i] - 1 > right) {
16             left = i;
17             right = i + z[i] - 1;
18         }
19     }
20     return z;
21 }

```

failer_fun.cpp

```

1 vector<int> fail_fun(string s) {
2     int n = (int)s.length();
3     vector<int> pi(n);
4     for (int i = 1; i < n; i++) {
5         int j = pi[i-1];
6         while (j > 0 && s[i] != s[j])
7             j = pi[j-1];
8         if (s[i] == s[j])
9             j++;
10        pi[i] = j;
11    }
12    return pi;
13 }

```

menacher.cpp

```

1 // Helper function to transform string for Manacher's algorithm
2 string transform(string &s) {

```

```

3     string t;
4     for(auto &val : s) {
5         t += '#';
6         t += val;
7     }
8     t += '#';
9     return t;
10 }

11
12 // Main Manacher's algorithm implementation
13 vector<int> build(string &s) {
14     string t = transform(s);
15     int n = t.size();
16     vector<int> p(n, 0);
17     int l = 0, r = 0;
18
19     for(int i = 0; i < n; i++) {
20         p[i] = (i < r) ? min(r - i, p[l + r - i]) : 1;
21         while(i - p[i] >= 0 && i + p[i] < n && t[i + p[i]] == t[i - p[i]]) {
22             p[i]++;
23         }
24         if(i + p[i] > r) {
25             r = i + p[i];
26             l = i - p[i];
27         }
28     }
29     return p;
30 }
31
32 // Check if substring [l,r] is palindrome using p array
33 bool isPalindrome(int l, int r, vector<int>& p) {
34     int center = (l + r) / 2;
35     bool odd = (l % 2 == r % 2);
36     int newCenter = 2 * center + !odd + 1;
37     return (r - l + 1) <= (p[newCenter] - 1);
38 }
39
40 // Returns {start_index, end_index} of longest palindrome substring using p array
41 pair<int,int> LongestPalindromeSubStr(vector<int>& p) {
42     int maxLength = 0;
43     int start = 0, end = 0;
44     int n = p.size() / 2; // Original string length
45
46     for(int i = 0; i < n; i++) {

```

```

47     // Even length palindromes
48     int newCenter = 2 * i + 2;
49     int len = p[newCenter] - 1;
50     if(len > maxLength) {
51         maxLength = len;
52         start = i - len/2 + 1;
53         end = i + len/2;
54     }
55
56     // Odd length palindromes
57     newCenter = 2 * i + 1;
58     len = p[newCenter] - 1;
59     if(len > maxLength) {
60         maxLength = len;
61         start = i - len/2;
62         end = i + len/2;
63     }
64 }
65
66 return {start, end};

```

Hashing

Double_Hash_as_int.cpp

```

1  const int N = 2e5 + 9;
2  const int mod[] = { (int)1e9 + 7, 998244353 };
3  int o; // Which mod wher I current use.
4  mt19937 random_seed(time(0));
5  int rnd(int l, int r){
6      uniform_int_distribution<int> dist(l, r);
7      return dist(random_seed);
8  }
9  ll bpw(ll a, ll b){
10     a %= mod[o];
11     ll res = 1;
12     while(b){
13         if(b % 2) res = (res * a) % mod[o];
14         a = (a * a) % mod[o];
15         b >>= 1;
16     }
17     return res;

```

```

18 }
19 struct Mint{
20     ll x;
21     Mint(ll x = 0){ this->x = x % mod[o]; }
22     Mint operator +(const Mint &other) const{ return (x + other.x) % mod[o]; }
23     Mint operator -(const Mint &other) const{ return (x - other.x + mod[o]) % mod[o]; }
24     Mint operator *(const Mint &other) const{ return (x * other.x) % mod[o]; }
25     Mint operator /(const Mint &other) const{ return (x * bpw(other.x , mod[o] - 2)) %
mod[o]; }
26     bool operator ==(const Mint &other) const{ return x == other.x; }
27     bool operator !=(const Mint &other) const{ return x != other.x; }
28 };
29
30 Mint p[2] , pw[2][N];
31 void init(){
32     for(o = 0; o < 2; o++){
33         p[o] = Mint( rnd(31 , 39) );
34         pw[o][0] = Mint(1);
35         for(int i = 1; i < N; ++i){
36             pw[o][i] = pw[o][i - 1] * p[o];
37         }
38     }
39 }
40 struct Hash{
41     Mint pref[2] , suff[2];
42     int len;
43     Hash(char ch = '?'){
44         if(ch == '?'){
45             pref[0] = pref[1] = suff[0] = suff[1] = Mint(0);
46             len = 0;
47             return;
48         }
49         len = 1;
50         for(o = 0; o < 2; ++o){
51             pref[o] = suff[o] = Mint(ch - 'a' + 1);
52         }
53     }
54     Hash operator +(const Hash &other) const{
55         Hash res;
56         res.len = len + other.len;
57         for(o = 0; o < 2; ++o){
58             res.pref[o] = pref[o] + other.pref[o] * pw[o][len];
59             res.suff[o] = other.suff[o] + suff[o] * pw[o][other.len];
60         }

```

```

61     return res;
62 }
63 void rev(){
64     for(o = 0; o < 2; ++o) swap(pref[o] , suff[o]);
65 }
66 bool is_palindrome() const{
67     for(o = 0; o < 2; ++o){
68         if(pref[o] != suff[o]) return false;
69     }
70     return true;
71 }
72 };
73 Hash excludePrefix(Hash s1 , Hash s2){
74     Hash res;
75     res.len = s1.len - s2.len;
76     for(o = 0; o < 2; ++o){
77         res.pref[o] = s1.pref[o] - s2.pref[o];
78         res.pref[o] = res.pref[o] / pw[o][s2.len];
79
80         res.suff[o] = s1.suff[o] - s2.suff[o] * pw[o][s1.len - s2.len];
81     }
82     return res;
83 }
84 Hash excludeSuffix(Hash s1 , Hash s2){
85     Hash res;
86     res.len = s1.len - s2.len;
87     for(o = 0; o < 2; ++o){
88         res.pref[o] = s1.pref[o] - s2.pref[o] * pw[o][s1.len - s2.len];
89
90         res.suff[o] = s1.suff[o] - s2.suff[o];
91         res.suff[o] = res.suff[o] / pw[o][s2.len];
92     }
93     return res;

```

custom_unorderd_map.cpp

```

1  struct custom_hash {
2      static uint64_t splitmix64(uint64_t x) {
3          // http://xorshift.di.unimi.it/splitmix64.c
4          x += 0x9e3779b97f4a7c15;
5          x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
6          x = (x ^ (x >> 27)) * 0x94d049bb133111eb;

```

```

7     return x ^ (x >> 31);
8 }
9
10 size_t operator()(uint64_t x) const {
11     static const uint64_t FIXED_RANDOM =
12     chrono::steady_clock::now().time_since_epoch().count();
13     return splitmix64(x + FIXED_RANDOM);
14 }
15
16 // to initialize
17 unordered_map<long long, int, custom_hash> mp;

```

double_hashing.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  const int N = 1e6 + 9;
8  const int mod[] = { (int)1e9 + 7, 998244353 };
9  ll p[2] , pw[2][N];
10 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
11 int gen(int l, int r) { return rng() % (r - l + 1) + l; }
12 ll get(ll x, ll y) { return (x + 1) * 1e9 + y; } // return Double hashing = get(hash[0] , hash[1])
13 void init(){
14     for(int o = 0; o < 2; o++){
15         p[o] = gen(31 , 39); // Generate Base randomly
16         pw[o][0] = 1;
17         for(int i = 1; i < N; ++i)
18             pw[o][i] = pw[o][i - 1] * p[o] % mod[o];
19     }
20 }
21 vector<ll> gen_prefix(string &s){
22     int n = (int) s.size();
23     vector<ll> ans(n);
24     ll pre[2][n];
25     pre[0][0] = pre[1][0] = (s[0] - 'a' + 1);
26     for(int i = 1; i < n; ++i) {
27         for(int o = 0; o < 2; o++){

```

```

28     pre[o][i] = (pre[o][i - 1] + (s[i] - 'a' + 1) * pw[o][i]) % mod[o];
29 }
30 }
31 for(int i = 0; i < n; ++i)
32     ans[i] = get(pre[i][0], pre[i][1]);
33 return ans;
34 }
35 int main(){
36     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
37     #ifndef ONLINE_JUDGE
38     freopen("in.txt", "r", stdin);
39     freopen("out.txt", "w", stdout);
40     freopen("error.txt", "w", stderr);
41     #endif
42     init();
43 }

```

hashing_grid.cpp

```

1  const ll MOD1 = (1LL<<61) - 1, MOD2 = (1LL<<49) - 1, BASE1 = 999983, BASE2 = 99991;
2  ll hash_matrix[2010][2010];
3  int64_t MUL(uint64_t a, uint64_t b, uint64_t HashMod) {
4      uint64_t l1 = (uint32_t) a, h1 = a >> 32, l2 = (uint32_t) b, h2 = b >> 32;
5      uint64_t l = l1 * l2, m = l1 * h2 + l2 * h1, h = h1 * h2;
6      uint64_t ret = (l & HashMod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
7      ret = (ret & HashMod) + (ret >> 61);
8      ret = (ret & HashMod) + (ret >> 61);
9      return (int64_t) ret - 1;
10 }
11 ll compute(int lx, int rx, int ly, int ry, vector<vector<char>> &pattern) {
12     for (int i = lx; i <= rx; i++) {
13         ll rowHash = 0;
14         for (int j = ly; j <= ry; j++) rowHash = (MUL(rowHash, BASE1, MOD1) + (pattern[i][j] -
15             'a' + 1)) % MOD1;
16         hash_matrix[i][0] = rowHash;
17     }
18     ll finaleHash = 0;
19     for (int j = lx; j <= rx; j++) {
20         finaleHash = (MUL(finaleHash, BASE2, MOD2) + hash_matrix[j][0]) % MOD2;
21     }
22     return finaleHash;
23 }

```


k-hash.cpp

```

1  const int mods[] = {1000000007, 1000000009, 1000000021, 1000000033,
    1000000087, 1000000093, 1000000097, 1000000103, 1000000123, 1000000181},
    b = 10, K = 6;
2  array<int, K> gethash(string &s) {
3      array<int, K> ans, pw;
4      for (int k = 0; k < K; k++) {
5          ans[k] = 0;
6          pw[k] = 1;
7      }
8      for (auto i : s) {
9          for (int k = 0; k < K; k++) {
10             ans[k] += 1ll * pw[k] * (i - '0') % mods[k];
11             ans[k] %= mods[k];
12             pw[k] = (1ll) pw[k] * b % mods[k];
13         }
14     }
15     return ans;
16 }
17 // probability of collision = 1 / 10^{9 * 6} = 1 / 10^{54}

```

string_hash.cpp

```

1  mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
2
3  ll rand(ll l, ll r) {
4      return uniform_int_distribution<ll>(l, r)(rng);
5  }
6
7  const int mod1 = 1e9 + 7, mod2 = 1e9 + 9;
8  const int p1 = rand(1e7, 1e8), p2 = rand(1e7, 1e8);
9  const int N = 1e6 + 1;
10 vector<ll> pow1(N + 1), pow2(N + 1);
11
12 void init() {
13     pow1[0] = pow2[0] = 1;
14     for (int i = 1; i <= N; i++) {
15         pow1[i] = p1 * pow1[i - 1] % mod1;
16         pow2[i] = p2 * pow2[i - 1] % mod2;
17     }
18 }

```

```

19
20 struct stringHashing {
21     vector<ll> preHash1, preHash2, sufHash1, sufHash2;
22
23     void init(string &s) {
24         preHash1 = hash(s, preHash1, pow1, mod1);
25         preHash2 = hash(s, preHash2, pow2, mod2);
26         sufHash1 = hash(s, sufHash1, pow1, mod1, true);
27         sufHash2 = hash(s, sufHash2, pow2, mod2, true);
28     }
29
30     vector<ll> hash(string &s, vector<ll> &h, vector<ll> &p, int m, bool rev = false) {
31         int n = s.size();
32         h.resize(n);
33         int st = 0, en = n, delta = 1;
34         if (rev)
35             st = n - 1, en = -1, delta = -1;
36         int i = 0;
37         while (st != en) {
38             h[st] = (s[st] - 'a' + 1) * p[i] % m;
39             if (i != 0) {
40                 h[st] += h[st - delta];
41                 if (h[st] >= m)
42                     h[st] -= m;
43             }
44             st += delta;
45             i++;
46         }
47         return h;
48     }
49
50     ll query(int l, int r) {
51         ll h1 = preHash1[r], h2 = preHash2[r];
52         h1 -= (l == 0 ? 0 : preHash1[l - 1]);
53         h2 -= (l == 0 ? 0 : preHash2[l - 1]);
54         if (h1 < 0) h1 += mod1;
55         if (h2 < 0) h2 += mod2;
56         h1 = h1 * pow1[N - l] % mod1;
57         h2 = h2 * pow2[N - l] % mod2;
58         return h1 * h2;
59     }
60
61     ll revQuery(int l, int r) {
62         int n = sufHash1.size();

```

```

63     ll h1 = sufHash1[l], h2 = sufHash2[l];
64     h1 -= (r == (n - 1) ? 0 : sufHash1[r + 1]);
65     h2 -= (r == (n - 1) ? 0 : sufHash2[r + 1]);
66     if (h1 < 0) h1 += mod1;
67     if (h2 < 0) h2 += mod2;
68     h1 = h1 * pow1[N - n + r + 1] % mod1;
69     h2 = h2 * pow2[N - n + r + 1] % mod2;
70     return h1 * h2;
71 }

```

Lyndon

Duval.cpp

```

1  // Duval Algorithm: Lyndon Factorization
2  // Returns lexicographically decreasing Lyndon words
3  // Time: O(n), Space: O(n)
4  vector<string> duval(string const& s) {
5      int n = s.size();
6      int i = 0;
7      vector<string> fact;
8
9      while(i < n) {
10         int j = i + 1, k = i;
11         while(j < n && s[k] <= s[j]) {
12             if(s[k] < s[j]) k = i;
13             else k++;
14             j++;
15         }
16         while(i <= k) {
17             fact.push_back(s.substr(i, j - k));
18             i += j - k;
19         }
20     }
21     return fact;
22 }
23
24 // For index only version (more efficient):
25 vector<int> duvalldx(string const& s) {
26     int n = s.size();
27     int i = 0;
28     vector<int> starts; // Starting positions

```

```

29
30 while(i < n) {
31     starts.push_back(i); // Add start of current factor
32     int j = i + 1, k = i;
33     while(j < n && s[k] <= s[j]) {
34         if(s[k] < s[j]) k = i;
35         else k++;
36         j++;
37     }
38     while(i <= k) i += j - k;
39 }
40 return starts;
41 }
42
43 // Usage example:
44 /*
45 string s = "abccab";
46 auto factors = duval(s);
47 // factors = ["abc", "ab"]
48
49 auto idx = duvalidx(s);
50 // idx = [0, 3] meaning factors start at positions 0 and 3
51 */

```

min_cyclic_string.cpp

```

1 // Minimum cyclic shift of string using Duval algorithm
2 // Returns number of positions to shift right to get lexicographically minimum string
3 // Example: "cba" -> 2 (shift right 2 to get "abc")
4 // Time: O(n), Space: O(1)
5 int min_cyclic_shift(string s) {
6     s += s;
7     int n = s.size();
8     int i = 0, shift = 0;
9
10    while(i < n/2) {
11        shift = i;
12        int j = i + 1, k = i;
13        while(j < n && s[k] <= s[j]) {
14            if(s[k] < s[j]) k = i;
15            else k++;
16            j++;
17        }

```

```

18     while(i <= k) i += j - k;
19 }
20
21     return shift;
22 }
23
24 // Usage:
25 /*
26 string s = "cba"
27 int shift = min_cyclic_shift(s); // returns 2
28 rotate(s.begin(), s.begin() + shift, s.end()); // gets "abc"
29
30 s = "aaaa"
31 shift = min_cyclic_shift(s); // returns 0 (already minimum)
32 */

```

suffixArray

Kasai.cpp

```

1  /*
2   Build the LCP array using kasai algorithm in O(n) time
3   s: string , p: suffix array
4
5   lcp[i]: longest common prefix of suffix[i] , suffix[i + 1]
6   */
7   vector<int> Kasai(string const& s, vector<int> const& p) {
8       int n = s.size();
9       vector<int> rank(n, 0);
10      for(int i = 0; i < n; i++) rank[p[i]] = i;
11      int k = 0;
12      vector<int> lcp(n-1, 0);
13      for (int i = 0; i < n; i++) {
14          if(rank[i] == n - 1) {
15              k = 0;
16              continue;
17          }
18          int j = p[rank[i] + 1];
19          while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
20          lcp[rank[i]] = k;
21          if(k) k--;
22      }

```

```

23     return lcp;
24 }

```

RMQ_suffixarray.cpp

```

1  // Builds RMQ table for suffix array comparisons
2  // Returns equivalence classes for each power of 2
3  // Time: O(nlogn), Memory: O(nlogn)
4  vector<vector<int>> buildRMQSuffixArray(string s) {
5      const int ALPHA = 256;
6      s += '$';
7      int n = s.size();
8      vector<int> p(n), c(n), cnt(max(n, ALPHA), 0);
9      vector<vector<int>> ans;
10     for(int i = 0; i < n; i++) cnt[s[i]]++;
11     for(int i = 1; i < ALPHA; i++) cnt[i] += cnt[i-1];
12     for(int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
13
14     int classes = 1;
15     for(int i = 1; i < n; i++) {
16         if(s[p[i]] != s[p[i-1]]) classes++;
17         c[p[i]] = classes - 1;
18     }
19     ans.emplace_back(c);
20     vector<int> pn(n), cn(n);
21     for(int h = 0; (1 << h) < n; ++h) {
22         for(int i = 0; i < n; i++) {
23             pn[i] = p[i] - (1 << h);
24             if(pn[i] < 0) pn[i] += n;
25         }
26         fill(cnt.begin(), cnt.begin() + classes, 0);
27
28         for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
29         for(int i = 1; i < classes; i++) cnt[i] += cnt[i-1];
30         for(int i = n-1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
31
32         cn[p[0]] = 0;
33         classes = 1;
34         for(int i = 1; i < n; i++) {
35             int pos1 = p[i] + (1 << h);
36             int pos2 = p[i-1] + (1 << h);
37             // Replace modulo with comparison and subtraction

```

```

38     if(pos1 >= n) pos1 -= n;
39     if(pos2 >= n) pos2 -= n;
40
41     pair<int,int> cur = {c[p[i]], c[pos1]};
42     pair<int,int> prev = {c[p[i-1]], c[pos2]};
43     if(cur != prev) ++classes;
44     cn[p[i]] = classes - 1;
45 }
46 c.swap(cn);
47 ans.emplace_back(c);
48 }
49 p.erase(p.begin());
50 return ans;
51 }
52
53 // Get LCP of suffixes starting at i,j
54 // Time: O(logn)
55 int LCP(int i, int j, vector<vector<int>> &c) {
56     int ans = 0, n = c[0].size(), lg_n = c.size();
57     for(int k = lg_n - 1; k >= 0; k--) {
58         if(c[k][i % n] == c[k][j % n]) {
59             ans += 1 << k;
60             i += 1 << k;
61             j += 1 << k;
62         }
63     }
64     return ans;
65 }
66
67 // Compare same length suffixes starting at i,j
68 // Returns: -1 if i<j, 0 if equal, 1 if i>j
69 // Time: O(1)
70 int compare(int i, int j, int l, vector<vector<int>> &c) {
71     int n = c[0].size();
72     int k = 31 - __builtin_clz(l);
73     pair<int,int> a = {c[k][i], c[k][(i+l-(1<<k))%n]};
74     pair<int,int> b = {c[k][j], c[k][(j+l-(1<<k))%n]};
75     return a == b ? 0 : a < b ? -1 : 1;
76 }
77
78 // Compare substrings s[l1..r1] vs s[l2..r2]
79 // Returns: -1 if s1<s2, 0 if equal, 1 if s1>s2
80 // Time: O(logn)
81 int compare(int l1, int r1, int l2, int r2, string &s, vector<vector<int>> &c) {

```

```

82     int lcp = LCP(l1, l2, c);
83     int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1;
84     lcp = min({lcp, len1, len2});
85     if(lcp == len1 && lcp == len2) return 0;
86     if(lcp == len1) return -1;
87     if(lcp == len2) return 1;
88     return s[l1 + lcp] < s[l2 + lcp] ? -1 : 1;

```

build_suffixArray_slow_O(nlognlogn).cpp

```

1  vector<int> buildSuffixArray(string const & s){
2      int n = s.length();
3      vector<int> sa(n + 1), rank(n + 1);
4      for(int i = 0; i <= n; ++i) {
5          sa[i] = i;
6          rank[i] = i < n ? s[i] : -1;
7      }
8      auto rankf = [&](int i) { return i <= n ? rank[i] : -1; };
9      vector<int> nxt(n + 1);
10     for (int k = 1; k <= n; k <= 1) {
11         auto cmp = [&](int i, int j) { return make_pair(rank[i], rankf(i + k)) <
make_pair(rank[j], rankf(j + k)); };
12         sort(sa.begin(), sa.end(), cmp);
13         nxt[sa[0]] = 0;
14         for(int i = 1; i <= n; ++i){
15             nxt[sa[i]] = nxt[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
16         }
17         rank.swap(nxt);
18     }
19     sa.erase(sa.begin());
20     return sa;
21 }

```

build_suffixarray_veryfast.cpp

```

1  vector<int> buildSuffixArray(string s) {
2      int n = s.size(); s += "$";
3      vector<int> p(n+1), c(n+1), cl(n+1), cnt(max(256, n+1)), pl(n+1);
4      for(int i = 0; i <= n; i++) p[i] = i, c[i] = s[i];
5

```



```

6   for(int k = 0; (1 << k) <= n; k++) {
7       int len = 1 << k;
8
9       fill(cnt.begin(), cnt.end(), 0);
10      for(int i = 0; i <= n; i++) cnt[c[min(n, p[i] + len)]]++;
11      for(int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i-1];
12      for(int i = n; i >= 0; i--) p1[--cnt[c[min(n, p[i] + len)]]] = p[i];
13
14      fill(cnt.begin(), cnt.end(), 0);
15      for(int i = 0; i <= n; i++) cnt[c[p1[i]]]++;
16      for(int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i-1];
17      for(int i = n; i >= 0; i--) p[--cnt[c[p1[i]]]] = p1[i];
18
19      c1[p[0]] = 0;
20      for(int i = 1; i <= n; i++)
21          c1[p[i]] = c1[p[i-1]] + (c[p[i]] != c[p[i-1]] || c[min(n, p[i]+len) != c[min(n, p[i-1]+len)]]);
22      c.swap(c1);
23      if(c[p[n]] == n) break;
24  }
25
26  vector<int> res;
27  for(int i = 1; i <= n; i++) res.push_back(p[i]);
28  return res;
29  }

```

problems

count_pattern.cpp

```

1   // Problem link: https://cses.fi/problemset/task/2103/
2   pair<int,int> findPattern(const string &text, const string& pat, vector<int> &sa) {
3       /*
4           Count the no. of occurence of pattern "pat" into "text" into O(2log(n))
5           sa: Suffix order
6       */
7       int n = sa.size();
8       auto compare = [&](int pos) {
9           return text.compare(sa[pos], pat.length(), pat);
10      };
11
12      // Find leftmost occurrence
13      int left = 0, right = n-1;

```

```

14     while(left < right) {
15         int mid = left + (right - left) / 2;
16         if(compare(mid) >= 0) right = mid;
17         else left = mid + 1;
18     }
19     if(compare(left) != 0) return {-1, -1};
20     int start = left;
21
22     // Find rightmost occurrence
23     right = n-1;
24     while(left < right) {
25         int mid = left + (right - left + 1) / 2;
26         if(compare(mid) <= 0) left = mid;
27         else right = mid - 1;
28     }
29
30     return {start, left};
31 }
32
33 \\-----
34 auto find_string = [&](const string &pat)->pair<int,int> {
35     int st = 0, ed = n;
36     auto cmp = [&](int a, int b) {
37         if (a == -1)
38             return pat[i] < text[b + i];
39         return text[a + i] < pat[i];
40     };
41     for (int i = 0; i < pat.size() && st < ed; i++) {
42         st = lower_bound(p.begin() + st, p.begin() + ed, -1, cmp)
43             - p.begin();
44         ed = upper_bound(p.begin() + st, p.begin() + ed, -1, cmp)
45             - p.begin();
46     }
47     if(st >= ed) return {-1,-1};
48     return {st, ed - 1};
49 };

```

distinct_substring.cpp

```

1     int main(){
2         ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
3         string s; cin >> s;
4         int n = s.size();

```

```

5     auto p = buildSuffixArray(s);
6     auto lcp = Kasai(s, p);
7     ll distinct = 1LL*n*(n+1)/2 - accumulate(lcp.begin(),lcp.end(), 0LL);
8     cout << distinct << '\n';
9 }

```

kth_distinct_substring.cpp

```

1  /*
2  Problem link: https://cses.fi/problemset/task/2108/
3  You are given a string of length n.
4  If all of its distinct substrings are ordered lexicographically, what is the kth smallest of
   them?
5  */
6  int main(){
7      ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
8      string s; cin >> s;
9      int n = s.size();
10     auto p = buildSuffixArray(s);
11     auto lcp = Kasai(s,p);
12     ll k; cin >> k;
13     if(k <= n - p[0]){ cout << s.substr(p[0], k); return 0; }
14     k -= n - p[0];
15     for(int i = 0; i < lcp.size(); ++i){
16         int len = n - p[i+1] - lcp[i];
17         if(k <= len){ cout << s.substr(p[i+1], k+lcp[i]); return 0; }
18         k -= len;
19     }
20 }

```

kth_smallest_string.cpp

```

1  /*
2  Problem link: https://cses.fi/problemset/task/2109/
3  You are given a string of length n.
4  If all of its substrings (not necessarily distinct) are ordered lexicographically, what is the
   kth smallest of them?
5  */
6  string s; cin >> s;
7      int n = s.size();
8      auto p = buildSuffixArray(s);

```

```

9    auto lcp = Kasai(s,p);
10
11    auto k_th = [&](ll k)->pair<int,int>{ // {index , length}
12        if(k <= n - p[0]) return {0, k};
13        k -= n - p[0];
14        for(int i = 0; i < lcp.size(); ++i){
15            int len = n - p[i+1] - lcp[i];
16            if(k <= len) return {i+1, k + lcp[i]};
17            k -= len;
18        }
19        assert(false);
20    };
21
22    ll k; cin >> k;
23    ll lo = 1, hi = 1LL * n*(n+1) / 2 - accumulate(lcp.begin(), lcp.end(), 0LL), md;
24
25    while(lo < hi){
26        md = lo + (hi - lo) / 2;
27
28        auto [i, len] = k_th(md);
29
30        ll sum = len;
31        for(int j = 0; j < i; ++j) sum += n - p[j];
32        int x = len;
33        for(int j = i+1; j < n; ++j){
34            x = min(x, lcp[j-1]);
35            sum += x;
36        }
37
38        if(sum >= k) hi = md;
39        else lo = md + 1;
40
41    }
42
43    auto [i,len] = k_th(lo);
44    cout << s.substr(p[i], len);

```

no_of_distinct_string_of_length_x.cpp

```

1    /*
2    Problem link: https://cses.fi/problemset/task/2110/
3    You are given a string of length n.

```

```

4   For every integer between 1, n you need to print the number of distinct substrings of
    that length.
5
6   */
7   string s; cin >> s;
8   int n = s.size();
9   auto p = buildSuffixArray(s);
10  auto lcp = Kasai(s,p);
11
12  ll cnt[n+2]{};
13
14  auto upd = [&](int l , int r, int v){
15      if(l > r) return;
16      cnt[l] += v; cnt[r+1] -= v;
17  };
18
19  for(int i = 0; i < lcp.size(); ++i) upd(1,lcp[i] , -1);
20
21  for(int i = 1; i <= n; ++i) cnt[i] += cnt[i-1];
22  for(int i = 1; i <= n; ++i) cnt[i] += n - i + 1;
23  for(int i = 1; i <= n; ++i) cout << cnt[i] << ' ';

```

no_of_substring_at_least_f.cpp

```

1   /*
2   Given a string S and several frequencies Fi.
3   For each Fi output the number of substrings of S (the characters of substring should be
    contiguous)
4   that occur at least Fi times in S. Note, that we consider two substrings distinct
5   if they have distinct length, or they have distinct starting indices.
6   problem link: https://www.codechef.com/problems/ANUSAR
7
8
9   */
10  #include <bits/stdc++.h>
11  using namespace std;
12  using ll = long long;
13
14  vector<int> buildSuffixArray(string s) {
15      const int ALPHA = 256;
16      s += '$';
17      int n = s.size();
18      vector<int> p(n), c(n);

```

```

19 {
20     // Optimize first phase with counting sort
21     vector<int> cnt(ALPHA);
22     for(auto ch : s) cnt[ch]++;
23     for(int i = 1; i < ALPHA; i++) cnt[i] += cnt[i-1];
24     for(int i = n-1; i >= 0; i--) p[--cnt[s[i]]] = i;
25
26     c[p[0]] = 0;
27     int classes = 1;
28     for(int i = 1; i < n; i++) {
29         if(s[p[i]] != s[p[i-1]]) classes++;
30         c[p[i]] = classes - 1;
31     }
32 }
33
34 vector<int> pn(n), cn(n);
35 for(int h = 0; (1 << h) < n; ++h) {
36     int len = 1 << h;
37     for(int i = 0; i < n; i++) {
38         pn[i] = p[i] - len;
39         if(pn[i] < 0) pn[i] += n;
40     }
41
42     // Optimize counting sort for each phase
43     vector<int> cnt(n);
44     for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
45     for(int i = 1; i < n; i++) cnt[i] += cnt[i-1];
46     for(int i = n-1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
47
48     cn[p[0]] = 0;
49     int classes = 1;
50     for(int i = 1; i < n; i++) {
51         pair<int,int> cur = {c[p[i]], c[(p[i] + len) % n]};
52         pair<int,int> prev = {c[p[i-1]], c[(p[i-1] + len) % n]};
53         if(cur != prev) ++classes;
54         cn[p[i]] = classes - 1;
55     }
56     c.swap(cn);
57 }
58 p.erase(p.begin());
59 return p;
60 }
61
62 vector<int> buildLCP(const string& s, const vector<int>& p) {

```

```

63     int n = s.size();
64     vector<int> rank(n), lcp(n-1);
65     for(int i = 0; i < n; i++) rank[p[i]] = i;
66
67     for(int i = 0, k = 0; i < n; i++) {
68         if(rank[i] == n-1) {
69             k = 0;
70             continue;
71         }
72         int j = p[rank[i] + 1];
73         while(i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
74         lcp[rank[i]] = k;
75         if(k) k--;
76     }
77     return lcp;
78 }
79
80 // Optimized stack-based nearest smaller element
81 pair<vector<int>, vector<int>> getMinBounds(const vector<int>& arr) {
82     int n = arr.size();
83     vector<int> left(n), right(n);
84     vector<int> st;
85     st.reserve(n);
86
87     // Get left bounds
88     for(int i = 0; i < n; i++) {
89         while(!st.empty() && arr[st.back()] > arr[i]) st.pop_back();
90         left[i] = st.empty() ? -1 : st.back();
91         st.push_back(i);
92     }
93
94     // Clear stack for right bounds
95     st.clear();
96
97     // Get right bounds
98     for(int i = n-1; i >= 0; i--) {
99         while(!st.empty() && arr[st.back()] >= arr[i]) st.pop_back();
100        right[i] = st.empty() ? n : st.back();
101        st.push_back(i);
102    }
103
104    return {left, right};
105 }

```

```

106
107 void solve() {
108     string s;
109     cin >> s;
110     int n = s.size();
111
112     auto p = buildSuffixArray(s);
113     auto lcp = buildLCP(s, p);
114     auto [L, R] = getMinBounds(lcp);
115
116     // Pre-allocate vectors to avoid resizing
117     vector<ll> frq(n+9);
118     vector<vector<int>> group(n+1);
119     for(int i = 0; i < n+1; i++) group[i].reserve(n/2);
120
121     // Group LCP values
122     for(int i = 0; i < (int)lcp.size(); ++i) {
123         group[lcp[i]].push_back(i);
124     }
125
126     // Calculate frequencies
127     for(int len = n; len > 0; --len) {
128         for(size_t j = 0; j < group[len].size(); j) {
129             int x = group[len][j];
130             int l = L[x], r = R[x];
131             int f = r - l;
132             int minh = len;
133
134             if(l >= 0) minh = min(minh, len - lcp[l]);
135             if(r < (int)lcp.size()) minh = min(minh, len - lcp[r]);
136
137             frq[f] += 1LL * f * minh;
138
139             // Skip processed indices
140             while(j < group[len].size() && group[len][j] <= r) ++j;
141         }
142     }
143
144     // Calculate cumulative frequencies
145     for(int i = n-1; i > 1; --i) frq[i] += frq[i+1];
146     frq[1] = 1LL * n * (n + 1) / 2;
147
148     // Process queries

```



```

149     int q;
150     cin >> q;
151     while(q--) {
152         int x;
153         cin >> x;
154         cout << (x > s.size() ? 0 : frq[x]) << '\n';
155     }
156 }
157
158 int main() {
159     ios::sync_with_stdio(false);
160     cin.tie(nullptr);
161
162     int tc;
163     cin >> tc;
164     while(tc--) solve();

```

trie

Binary_trie.cpp

```

1  const int N = 2e5 * 30 + 9;
2  int nxt[N][2], isEnd[N], cntNode, frq[N];
3  void add(int x){
4      int node = 0;
5      for(int i = 30; i >= 0; --i){
6          int cur = (x >> i) & 1;
7          if(nxt[node][cur] == 0)
8              nxt[node][cur] = ++cntNode;
9          node = nxt[node][cur];
10         frq[node]++;
11     }
12     isEnd[node] = true;
13 }
14 void erase(int x){
15     int node = 0;
16     for(int i = 30; i >= 0; --i){
17         int cur = (x >> i) & 1;
18         if(nxt[node][cur] == 0) return; // this number "x" doesn't exist, (handle it manual)
19         node = nxt[node][cur];
20         frq[node]--;

```

```

21     }
22     if(frq[node] == 0) isEnd[node] = false;
23 }

```

trie.cpp

```

1  const int N = 1e6 + 9;
2  int nxt[N][27], cntNode, isEnd[N], frq[N], n;
3  int get(char ch){ return ch - 'a'; }
4  void add(string &s){
5      int node = 0;
6      for(auto &ch : s){
7          if(nxt[node][get(ch)] == 0)
8              nxt[node][get(ch)] = ++cntNode;
9          node = nxt[node][get(ch)];
10         frq[node]++;
11     }
12     isEnd[node] = true;
13 }
14 void remove(string &s){
15     int node = 0;
16     for(auto &ch : s){
17         if(nxt[node][get(ch)] == 0) return; // this word "s" doesn't exist, (handle it manual)
18         node = nxt[node][get(ch)];
19         frq[node]--;
20     }
21     if(frq[node] == 0) isEnd[node] = false;
22 }

```

Trees

CentroidDecomposition.cpp

```

1  const int N = 1e5 + 9;
2  vector<int> adj[N];
3  struct CentroidDecomposition{
4      vector<int> removal, sz;
5      CentroidDecomposition(int n){
6          removal.assign(n, 0);
7          sz.assign(n, 0);
8          build(0, -1);

```

```

9     }
10    void build(int u , int p){
11        int n = dfs(u , p);
12        int centriod = getCentriod(u , p , n);
13        removal[centriod] = true;
14        // depend on the problem
15
16        for(auto &v : adj[centriod]) if(!removal[v])
17            build(v , centriod);
18    }
19    int dfs(int u , int p){
20        sz[u] = 1;
21        for(auto &v : adj[u]) if(v != p && !removal[v])
22            sz[u] += dfs(v , u);
23        return sz[u];
24    }
25    int getCentriod(int u , int p , int n){
26        for(auto &v : adj[u]) if(v != p && !removal[v]){
27            if(sz[v] * 2 > n)
28                return getCentriod(v , u , n);
29        }
30        return u;
31    }
32
33 };

```

HLD.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  #define all(v) v.begin() , v.end()
8  #ifndef ONLINE_JUDGE
9  #define debug(x) cerr << #x << ": " << x << '\n';
10 #else
11 #define debug(x)
12 #endif
13 const int N = 2e5 + 9;
14 vector<int> adj[N];

```

```

15  int heavy[N], head[N], par[N], pos[N], dep[N], cur_pos;
16  int dfs(int u = 0){
17      heavy[u] = -1; // node is leaf
18      int size = 1, max_size = 0;
19      for(auto &v : adj[u]) if(v != par[u]){
20          par[v] = u;
21          dep[v] = dep[u] + 1;
22          int cur = dfs(v);
23          if(cur > max_size){
24              heavy[u] = v;
25              max_size = cur;
26          }
27          size += cur;
28      }
29      return size;
30  }
31  void decomposition(int u = 0, int h = 0){
32      head[u] = h;
33      pos[u] = cur_pos++;
34      if(~heavy[u])
35          decomposition(heavy[u], h);
36      for(auto &v : adj[u]) if(v != par[u] && v != heavy[u])
37          decomposition(v, v);
38  }
39  int path(int u, int v){
40      int ans = 0;
41      for(; head[u] != head[v]; v = par[head[v]]){
42          if(dep[head[u]] > dep[head[v]]) swap(u, v);
43          // process interval: [ pos[head[v]], pos[v] ]
44
45      }
46      if(dep[u] > dep[v]) swap(u, v);
47      // process interval: [ pos[u], pos[v] ]
48
49      return ans;
50  }
51
52
53  int main(){
54      ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
55      #ifndef ONLINE_JUDGE
56          freopen("in.txt", "r", stdin);
57          freopen("out.txt", "w", stdout);
58          freopen("error.txt", "w", stderr);

```

```

59     #endif
60     int n,q; cin >> n >> q;
61     vector<int> a(n);
62     for(int i = 0; i < n; ++i) cin >> a[i];
63     for(int i = 0; i < n - 1; ++i){
64         int u , v; cin >> u >> v;
65         --u; --v;
66         adj[u].emplace_back(v);
67         adj[v].emplace_back(u);
68     }
69     dfs(); decomposition();

```

HLD_edges.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 2e5 + 9;
4
5  vector<pair<int,int>> adj[N];
6  int heavy[N], head[N], par[N], pos[N], dep[N], sz[N], cur_pos, n;
7  vector<int> values;
8
9  int dfs(int u = 0) {
10     heavy[u] = -1;
11     sz[u] = 1;
12     int max_size = 0;
13     for(auto &[v, w] : adj[u]) if(v != par[u]) {
14         par[v] = u;
15         dep[v] = dep[u] + 1;
16         int cur = dfs(v);
17         if(cur > max_size) {
18             heavy[u] = v;
19             max_size = cur;
20         }
21         sz[u] += cur;
22     }
23     return sz[u];
24 }
25
26 void decomposition(int u = 0, int h = 0) {
27     head[u] = h;
28     pos[u] = cur_pos++;

```

```

29     if(~heavy[u])
30         decomposition(heavy[u], h);
31     for(auto &[v, w] : adj[u]) if(v != par[u] && v != heavy[u])
32         decomposition(v, v);
33 }
34
35 void buildWeightArray() {
36     values.resize(n);
37     for(int u = 0; u < n; u++) {
38         for(auto [v, w] : adj[u]) {
39             if(dep[u] < dep[v]) {
40                 values[pos[v]] = w;
41             }
42         }
43     }
44 }
45
46 int queryPath(int u, int v) {
47     int ans = -1e9;
48     for(; head[u] != head[v]; v = par[head[v]]) {
49         if(dep[head[u]] > dep[head[v]]) swap(u,v);
50         // process range [pos[head[v]], pos[v]]
51     }
52     if(dep[u] > dep[v]) swap(u,v);
53     if(u != v) {
54         // process range [pos[u] + 1, pos[v]]
55     }
56     return ans;
57 }
58
59 // Range to query: [pos[u] + 1, pos[u] + sz[u] - 1]
60 int querySubtree(int u) {
61     // process range [pos[u] + 1, pos[u] + sz[u] - 1]
62     return 0; // replace with actual query
63 }
64
65 int main() {
66     ios::sync_with_stdio(0); cin.tie(0);
67     int m, q; cin >> n >> m >> q;
68
69     for(int i = 0; i < m; i++) {
70         int u, v, w; cin >> u >> v >> w;
71         --u; --v;
72         adj[u].push_back({v, w});

```

```

73     adj[v].push_back({u, w});
74 }
75
76 dfs();
77 decomposition();
78 buildWeightArray();

```

Tree_center_using_bfs.cpp

```

1  /*
2   Finding Tree Center and Tree diameter in Time O(n).
3   Algo :
4   1- Starting BFS from any node.
5   2- Find the farthest node (Start) from it.
6   3- Starting BFS from (Start).
7   5- Find the farthest node (End) from (Start)
8   6- The path from (Start) and (End) is one possible diameter for the tree.
9  */
10 int n; cin >> n;
11 for(int i = 0; i < n - 1; ++i){
12     int u , v; cin >> u >> v;
13     --u; --v;
14     adj[u].emplace_back(v);
15     adj[v].emplace_back(u);
16 }
17 auto bfs = [&](int st){
18     vector<int> dis(n , -1);
19     queue<int> q;
20     q.push(st); dis[st] = 0;
21     while(q.size()){
22         int u = q.front(); q.pop();
23         for(auto &v : adj[u]) if(dis[v] == -1){
24             dis[v] = dis[u] + 1; pre[v] = u;
25             q.push(v);
26         }
27     }
28     return max_element(dis.begin() , dis.end()) - dis.begin();
29 };
30 int Start = bfs(0) , End = bfs(Start);
31 vector<int> v;
32 for(int i = End;; i = pre[i]){
33     v.emplace_back(i);

```

```

34     if(i == Start) break;
35 }
36 int center = v[v.size() / 2];

```

Tree_flatten.cpp

```

1  const int N = 1e5 + 9;
2  int l[N] , r[N] , timer;
3  vector<int> adj[N];
4  void dfs(int u , int par){
5      l[u] = ++timer;
6      for(auto &v : adj[u]) if(v != par)
7          dfs(v , u);
8      r[u] = timer;
9      // For each subTree u : [ l[u] , r[u] ]
10 }

```

bridgestree.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  #define rep(i , st , ed) for(int i = st; i < ed; i++)
5  #define f first
6  #define s second
7  const int N = 3e5 + 9;
8  vector<pair<int,int>> adj[N] , edges;
9  vector<int> BridgeTree[N];
10 int lowLink[N] , dfn[N] , comp[N] , ndfn , comp_num;
11 bool isBridge[N] , vis[N];
12 void tarjan(int u , int par){
13     dfn[u] = lowLink[u] = ndfn++;
14     for(auto &[v , id] : adj[u]){
15         if(dfn[v] == -1){
16             tarjan(v , u);
17             lowLink[u] = min(lowLink[u] , lowLink[v]);
18             if(lowLink[v] == dfn[v]){
19                 int uu = u , vv = v;
20                 if(uu > vv) swap(uu , vv);
21                 isBridge[id] = true;
22             }

```



```

23     }else if(v != par){
24         lowLink[u] = min(lowLink[u] , dfn[v]);
25     }
26 }
27 }
28 void Find_component(int u , int par){
29     vis[u] = true;
30     comp[u] = comp_num;
31     for(auto &[v , id] : adj[u]) if(vis[v] == 0 && isBridge[id] == 0)
32         Find_component(v , u);
33 }
34 pair<int, int> diameter(int u, int par = -1)
35 {
36     int diam = 0;
37     int mxHeights[3] = {-1, -1, -1}; // keep 2 highest trees
38     for(auto &v : BridgeTree[u]) if(v != par)
39     {
40         auto p = diameter(v , u);
41         diam = max(diam, p.f);
42         mxHeights[0] = p.s+1;
43         sort(mxHeights, mxHeights+3);
44     }
45     for(int i = 0; i < 3; i++)if(mxHeights[i] == -1)
46         mxHeights[i] = 0;
47     diam = max(diam, mxHeights[1] + mxHeights[2]);
48     return {diam, mxHeights[2]};
49 }
50 int main(){
51     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
52     #ifndef ONLINE_JUDGE
53     freopen("in.txt", "r", stdin);
54     freopen("out.txt", "w", stdout);
55     freopen("error.txt", "w", stderr);
56     #endif
57     int n, m; cin >> n >> m;
58     for(int i = 0; i < m; i++){
59         int u , v; cin >> u >> v;
60         --u; --v;
61         adj[u].emplace_back(v, i);
62         adj[v].emplace_back(u , i);
63         edges.emplace_back(u , v);
64     }
65     // Finding Bridges using Tarjan algo.
66     for(int i = 0; i < n; i++){ dfn[i] = -1; lowLink[i] = 0; }

```

```

67     ndfn = 0;
68     tarjan(0, 0);
69     // dfs to group all the maximal components together, so that we can shrink it to one
    node
70     for(int i = 0; i < n; i++) if(vis[i] == 0){
71         Find_component(i, i);
72         comp_num++;
73     }
74     // shrinking all the maximal components to one node
75     for(int i = 0; i < m; i++) {
76         if(isBridge[i]) {
77             BridgeTree[comp[edges[i].f]].emplace_back(comp[edges[i].s]);
78             BridgeTree[comp[edges[i].s]].emplace_back(comp[edges[i].f]);
79         }
80     }
81     // Finding the diameter of the Bridgestree
82     int d = diameter(0, 0).f;
83     cout << d;

```

centroid.cpp

```

1  /*
2   Simply Centroid is a node if we delete it. It makes some subtrees where every subtree
   size must be less than sz/2 { sz is the size of the current tree T.}
3  */
4  vector<int> adj[N] , centriod, sz(N);
5  int n;
6  void dfs(int u , int par){
7      sz[u] = 1;
8      bool is_Centriod = true;
9      for(auto &v : adj[u]){
10         if(v != par){
11             dfs(v , u);
12             sz[u] += sz[v];
13             if(sz[v] * 2 > n) is_Centriod = false;
14         }
15     }
16     // check above tree
17     if((n - sz[u]) * 2 > n) is_Centriod = false;
18     if(is_Centriod) centriod.emplace_back(u);
19 }
20 /*

```

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  #define rep(i , st , ed) for(int i = st ; i < ed ; i++)
6  using namespace std;
7  const int N = 4000;
8  void burn(){
9  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
10 #ifndef ONLINE_JUDGE
11     freopen("in.txt" , "r" , stdin);
12     freopen("out.txt" , "w" , stdout);
13     freopen("error.txt" , "w" , stderr);
14 #endif
15 }
16 ///////////////////////////////////////////////////////////////////
17 int n;
18 vector<int> adj[N];
19 string get_subcan(int u , vector<vector<string>> &subcan){
20     sort(subcan[u].begin() , subcan[u].end());
21     string ans = "(";
22     for(auto &s : subcan[u]) ans += s;
23     ans += ")";
24     return ans;
25 }
26 void tree_isophrisim(){
27     vector<int> deg(n);
28     queue<int> leaf;
29     vector<vector<string>> subcan(n);
30     rep(i , 0 , n){
31         deg[i] = adj[i].size();
32         if(deg[i] <= 1) leaf.push(i);
33     }
34     int rem = n;
35     while(rem > 2){
36         int sz = leaf.size();

```

```

37     while(sz--){
38         rem--;
39         int u = leaf.front(); leaf.pop();
40         string temp = get_subcan(u , subcan);
41         for(auto &v : adj[u]){
42             subcan[v].emplace_back(temp);
43             deg[v]--;
44             if(deg[v] == 1) leaf.push(v);
45         }
46     }
47 }
48 vector<string> ans; // contain all possible canonical
49 int c1 = leaf.front(); leaf.pop();
50 if(leaf.empty()){
51     // tree has one center
52     ans.emplace_back(get_subcan(c1 , subcan));
53 }else{
54     // tree has 2 center
55     int c2 = leaf.front();
56     string temp1 = get_subcan(c1 , subcan),
57         temp2 = get_subcan(c2 , subcan);
58     subcan[c1].push_back(temp2);
59     subcan[c2].push_back(temp1);
60     ans.emplace_back(get_subcan(c1 , subcan));
61     ans.emplace_back(get_subcan(c2 , subcan));
62 }
63 }
64 int main(){
65     burn();
66 }

```

tree_center.cpp

```

1  #include <bits/stdc++.h>
2  typedef long long ll;
3  #define s second
4  #define f first
5  #define rep(i , st , ed) for(int i = st ; i < ed ; i++)
6  using namespace std;
7  const int N = 4000;
8  void burn(){
9  ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);

```

```

10  #ifndef ONLINE_JUDGE
11      freopen("in.txt", "r", stdin);
12      freopen("out.txt", "w", stdout);
13      freopen("error.txt", "w", stderr);
14  #endif
15  }
16  ///////////////////////////////////////////////////////////////////
17  int n;
18  vector<int> adj[N];
19  void tree_center(){
20      vector<int> deg(n);
21      queue<int> leaf;
22      rep(i, 0, n){
23          deg[i] = adj[i].size();
24          if(deg[i] <= 1) leaf.push(i);
25      }
26      int rem = n;
27      while(rem > 2){
28          int sz = leaf.size();
29          while(sz--){
30              rem--;
31              int u = leaf.front(); leaf.pop();
32              for(auto &v : adj[u]){
33                  deg[v]--;
34                  if(deg[v] == 1) leaf.push(v);
35              }
36          }
37      }
38      int c1 = leaf.front(); leaf.pop();
39      int c2 = (leaf.size()) ? leaf.front() : -1;
40  }
41  int main(){
42      burn();
43  }

```

tree_diameter_dfs.cpp

```

1  pair<int, int> diameter(int u, int par = -1)
2  {
3      int diam = 0;
4      int mxHeights[3] = {-1, -1, -1}; // keep 2 highest trees
5

```

```
6
7  for(auto &v : adj[u]) if(v != par)
8  {
9      pair<int, int> p = diameter(v, u);
10     diam = max(diam, p.first);
11
12     // Keep only the 2 maximum children
13     mxHeights[0] = p.second+1;
14     sort(mxHeights, mxHeights+3);
15 }
16
17 for(int i = 0; i < 3; ++i) if(mxHeights[i] == -1)
18     mxHeights[i] = 0;
19
20 diam = max(diam, mxHeights[1] + mxHeights[2]);
21
22 return make_pair(diam, mxHeights[2]);
23 }
```