Table of Contents

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

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

- Segment tree
 - <u>SparseSegmentTree.cpp</u>
 - <u>segmentTreeiterative.cpp</u>
 - segtreeBeat.cpp
 - <u>segtreeBeatforMod.cpp</u>
- Sparse table
 - <u>getIdxSparseTable.cpp</u>
 - <u>sparsesegmentTree.cpp</u>
 - <u>sparsetable.cpp</u>
- 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
 - <u>Hierholzer.cpp</u>
 - <u>SCCFloyed.cpp</u>
 - SCCkosaraju.cpp
 - SCCtarjan.cpp
 - artPoints.cpp
 - bellmanford.cpp
 - biConnected.cpp
 - <u>dijkstra.cpp</u>
 - <u>dijkstrasparsegraph.cpp</u>
 - <u>eulerciruiteundirectedgraph.cpp</u>
 - <u>eulerpathdirected.cpp</u>
 - <u>eulerpathundirected.cpp</u>
 - <u>floyd.cpp</u>
 - kthshortestpath.cpp
- Handbook material
- <u>Imgs</u>

- Masking
 - GeneratenextlexicographicalK-combination.cpp
 - Generatingallsubmaskingofkones.cpp
 - <u>genallpossiblesubmasking.cpp</u>
- Math
 - Counting
 - BurnsideLemma.cpp
 - FFT.cpp
 - <u>JosephusTheorem.cpp</u>
 - <u>MatrixExponential.cpp</u>
 - Mint.cpp
 - NTTprimemod.cpp
 - NTTrandommod.cpp
 - <u>checkpointinsidepolygon.cpp</u>
 - convolutionnttoptimized.cpp
 - <u>convolutionusingNTTkactl.cpp</u>
 - <u>derangement.cpp</u>
 - evaluatingpolynomialatnpoints.cpp
 - <u>lucas.cpp</u>
 - nCr.cpp
 - nCrDP.cpp
 - nCrO(r).cpp
 - nCrmod2.cpp
 - ncrO(n).cpp
 - nttoptimized.cpp
 - <u>simpsonintegration.cpp</u>
 - startbars.cpp
 - stirling.cpp
 - <u>sumxorrange.cpp</u>
 - Game Theory
 - Grundy 01.cpp
 - Grundy 02.cpp
 - Grundy String Game.cpp
 - <u>GrundyKnightsMoveInMatrix.cpp</u>
 - Matrix Nim.cpp
 - Misere Nim.cpp
 - Nim 1.cpp
 - Staircase Nim.cpp

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

- Notes
- Searching
 - countingsort.cpp
 - <u>doubleternarysearch.cpp</u>
 - 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
 - dividegridintokconnctedcompented01.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
 - <u>DoubleHashasint.cpp</u>
 - <u>customunorderdmap.cpp</u>
 - doublehashing.cpp

- hashinggrid.cpp
- k-hash.cpp
- stringhash.cpp
- <u>Lyndon</u>
 - <u>Duval.cpp</u>
 - mincyclicstring.cpp
- <u>suffixArray</u>
 - Kasai.cpp
 - RMQsuffixarray.cpp
 - buildsuffixArrayslowO(nlognlogn).cpp
 - <u>buildsuffixarrayveryfast.cpp</u>
 - problems
 - countpattern.cpp
 - <u>distinctsubstring.cpp</u>
 - kthdistinctsubstring.cpp
 - kthsmalleststring.cpp
 - noofdistinctstringoflengthx.cpp
 - noofsubstringatleastf.cpp
- trie
 - Binarytrie.cpp
 - trie.cpp
- Trees
 - CentroidDecomposition.cpp
 - HLD.cpp
 - <u>HLDedges.cpp</u>
 - <u>Treecenterusingbfs.cpp</u>
 - 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.
      * Can be applied on:
 3
       - monotonic slopes, monotonic queries
 4
       - monotonic slopes, random queries
 5
 6
 7
       Here's monotonic can be (ASCE, DESC)
      */
 8
     const || INF = LLONG_MAX;
 9
     const | NEG_INF = LLONG_MIN;
10
11
     struct mxCHT { // Convex Hull Trick for Maximum
12
       deque<long long> A; // Slopes
13
       deque<long long> B; // Intercepts
14
15
       // Modified 'bad' function with reversed inequality for Maximum CHT
16
       bool bad(int 11, int 12, int 13) {
17
18
         return (B[I3] - B[I1]) * (long double)(A[I1] - A[I2])
             >= (B[I2] - B[I1]) * (long double)(A[I1] - A[I3]); // Changed <= to >=
19
20
       }
21
22
       // Adds a line with slope `a` and intercept `b` when slopes are in ascending order
       void addASC(long long a, long long b) {
23
         A.push_back(a);
24
         B.push_back(b);
25
26
         // Remove the second last line if it's unnecessary
         while (A.size() >= 3 \&\& bad(A.size() - 3, A.size() - 2, A.size() - 1)) {
27
           A.erase(A.end() - 2);
28
           B.erase(B.end() - 2);
29
         }
30
       }
31
32
33
       // Adds a line with slope `a` and intercept `b` when slopes are in descending order
       void addDESC(long long a, long long b) {
34
         A.push_front(a);
35
         B.push_front(b);
36
         // Remove the second line if it's unnecessary
37
         while (A.size() >= 3 \&\& bad(0, 1, 2)) \{
38
           A.erase(A.begin() + 1);
39
40
           B.erase(B.begin() + 1);
41
         }
       }
42
```

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

CHT_minline.cpp

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

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

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

LineContainerMax_double.cpp

```
struct Line {
 1
       mutable double k, m, p;
 2
       bool operator<(const Line& o) const { return k < o.k; }
 3
      bool operator<(double x) const { return p < x; }
 4
 5
      };
 6
 7
      struct LineContainerMax : multiset<Line, less<>> {
       static constexpr double inf = numeric_limits<double>::infinity();
 8
       static constexpr double EPS = 1e-9;
 9
10
11
       double div(double a, double b) {
         return a / b;
12
       }
13
14
15
       bool isect(iterator x, iterator y) {
         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
18
         else x->p = div(y->m - x->m, x->k - y->k);
         return x->p >= y->p;
19
       }
20
21
22
       void add(double k, double m) {
         auto z = insert(\{k, m, O\}), 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) {
    assert(!empty());
    auto I = *lower_bound(x);
    return l.k * x + l.m;
}
```

LineContainerMin_double.cpp

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

34 };

LineContainer_minLine.cpp

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

Linecontainer_maxLine.cpp

```
1
      * Description: Container where you can add lines of the form kx+m, and guery
 2
      maximum values at points x.
      * Can be applied on:
 3
 4
        - monotonic slopes, monotonic queries
        - monotonic slopes, random queries
 5
        - random slopes, random queries
 6
      */
 7
 8
      struct Line {
 9
        mutable II k, m, p;
10
        bool operator<(const Line& o) const { return k < o.k; }
11
12
        bool operator<(II x) const { return p < x; }
      };
13
14
      struct LineContainerMax : multiset<Line, less<>> {
15
        // (for doubles, use inf = 1/.0, div(a,b) = a/b)
16
        static const II inf = LLONG_MAX;
17
        Il div(II a, II 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
26
        void add(II k, II m) {
27
          auto z = insert(\{k, m, O\}), y = z++, x = y;
          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
        Il query(Il x) {
33
          assert(!empty());
34
          auto I = *lower_bound(x);
35
          return l.k * x + l.m;
36
        }
37
38
      };
```

Divide and Conquer

DP_dnc.cpp

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

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

optimized_knapsack.cpp

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

Knuth's

DP_knuths_cpAlgo.cpp

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

Sum of subset

SOS_DP.cpp

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

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.
        For simplicity, we will consider all non-zero elements equal to 1.
 5
 6
        Time Complexity: O(n x m)
 7
 8
        Space Complexity: O(m)
      */
 9
      int zero_matrix(vector<vector<int>> a) {
10
        int n = a.size();
11
        int m = a[0].size();
12
13
        int ans = 0;
14
15
        vector<int> d(m, -1), d1(m), d2(m);
        stack<int> st;
16
        for (int i = 0; i < n; ++i) {
17
          for (int j = 0; j < m; ++j) {
18
            if(a[i][i] == 1)
19
              d[j] = i;
20
          }
21
22
23
          for (int j = 0; j < m; ++j) {
            while (!st.empty() && d[st.top()] <= d[j])
24
              st.pop();
25
26
            d1[j] = st.empty() ? -1 : st.top();
            st.push(j);
27
28
29
          while (!st.empty())
30
            st.pop();
31
          for (int j = m - 1; j \ge 0; --j) {
32
            while (!st.empty() && d[st.top()] <= d[j])
33
34
              st.pop();
            d2[j] = st.empty()? m: st.top();
35
            st.push(j);
36
37
          while (!st.empty())
38
39
            st.pop();
40
```

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

ans = max(ans, (i - d[j]) * (d2[j] - d1[j] - 1));

return ans;

}
```

Data Structure

2D Range Queries

2D_prefixsum.cpp

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

2d_sparsetable.cpp

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

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

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

3D_prefixsum.cpp

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

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

BIT2D.cpp

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

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

BIT2D_nlogn2.cpp

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

```
swap(a.first, a.second);
18
          sort(pts.begin(), pts.end());
19
20
          for (auto& a: pts) {
21
            swap(a.first, a.second);
            for (int on = std::upper_bound(ord.begin(), ord.end(), a.first) - ord.begin(); on <
22
      fw.size(); on += on \& -on)
              if (coord[on].empty() || coord[on].back() != a.second)
23
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
        T merge(Ta, Tb){}
30
          return a + b;
31
        }
32
        // point upd
33
34
        void upd(T x, T y, T v) {
          for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx < fw.size(); xx +=
35
      (xx - 3xx)
            for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) -
36
      coord[xx].begin(); yy < fw[xx].size(); yy += yy & -yy)
37
              fw[xx][yy] = merge(fw[xx][yy], v);
        }
38
39
40
        // point gry
        Tqry(Tx, Ty){
41
42
          T ans = 0;
          for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx > 0; xx -= xx & -
43
      xx)
            for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) -
44
      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(Tx1, Ty1, Tx2, Ty2) {
          return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) + qry(x1 - 1, y1 - 1);
51
        }
52
53
54
        // range upd
55
        void upd(T x1, T y1, T x2, T y2, T v) {
56
          upd(x1, y1, v);
```

segTree_2d.cpp

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

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

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

segment_sparse_table_2d.cpp

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

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

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

BIT

BIT.cpp

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

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

BIT_LAZY_MODIFICATION.cpp

```
1 /*
2
       BIT with LAZY MODIFICATION
       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
     // 1-base
6
7
     const int inf = 1e9 + 9;
     map<int,long long> tree; // LAZY MODIFICATION
8
     struct BIT {
9
       void add(int x, int val) {
10
         for(; x < \inf; x += (x \& -x)) tree[x] += val;
11
12
       }
       long long sum(int x){
13
         long long res = 0;
14
         for(;x > 0; x -= (x & -x)) res += tree[x];
15
16
         return res:
17
       long long sum(int I, int r){ return sum(r) - sum(I - 1); }
18
     };
19
```

BIT_Ranges.cpp

```
const int N = 2e5 + 9;
class BITrange {
  private:
  long long m[N], c[N];
  void add(int pos, long long mVal, long long cVal) {
  for (++pos; pos <= N; pos += (pos & (-pos))) {
    m[pos - 1] += mVal;
    c[pos - 1] += cVal;
}</pre>
```

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

BIT_with_coordinate_compression.cpp

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

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

Multiset_fenwick.cpp

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

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

mex_using_BIT.cpp

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

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

DSU

DSU.cpp

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

DynamicConnectivity_offline.cpp

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

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

RollbackUF.cpp

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

DSU on tree

sackDFS_keepBigChild.cpp

Serso Library

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

Serso Library

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

small_to_large_trick.cpp

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

```
mp[p][color[u]]++;
27
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);
          for(auto \&[c,frq]: mp[x]) mp[p][c] += frq;
31
        }
32
        ans[u] = (int) mp[p].size();
33
34
        return p;
      }
35
36
      int main(){
37
        ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
        #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
44
        for(int i = 0; i < n; ++i) cin >> color[i];
45
        for(int i = 0; i < n - 1; ++i){
          int u, v; cin >> u >> v;
46
47
          --U; --V;
          adj[u].emplace_back(v);
48
          adj[v].emplace_back(u);
49
        }
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

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

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

onlineRangeQueryD&C_benq.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long II;
#define rep(i, st, ed) for(int i = st; i < ed; i++)
#define f first</pre>
#define f first
```

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

online_static_rmq_dc.cpp

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

Kd-tree

kd-tree 3d.cpp

```
struct Point {
        long double x, y, z;
 2
        Point(long double x, long double y, long double z):x(x), y(y), z(z){}
 3
        bool operator<(const Point& p) const {</pre>
 4
          return x < p.x || (x == p.x && (y < p.y || (y == p.y && z < p.z)));
 5
        }
 6
      };
 7
 8
 9
      class KDTree {
      private:
10
        struct Node {
11
          Point point;
12
          Node* left;
13
14
          Node* right;
          Node(Point p) : point(p), left(nullptr), right(nullptr) {}
15
        };
16
        Node* root;
17
18
        Node* build(vector<Point>& points, int depth) {
19
          if (points.empty()) return nullptr;
20
          int axis = depth % 3;
21
          sort(points.begin(), points.end(), [axis](Point a, Point b) {
22
            if (axis == 0) return a.x < b.x;
23
            if (axis == 1) return a.y < b.y;
24
25
            return a.z < b.z;
          });
26
          int mid = points.size() / 2;
27
28
          Node* node = new Node(points[mid]);
          vector<Point> leftPoints(points.begin(), points.begin() + mid);
29
          vector<Point> rightPoints(points.begin() + mid + 1, points.end());
30
          node->left = build(leftPoints, depth + 1);
31
32
          node->right = build(rightPoints, depth + 1);
          return node;
33
        }
34
35
        long double dist(Point p1, Point p2) {
36
          long double res = (p1.x - p2.x) * (p1.x - p2.x);
37
          res += (p1.y - p2.y) * (p1.y - p2.y);
38
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;
         long double currentDist = dist(node->point, target);
45
         if (currentDist < bestDist && currentDist > 0) bestDist = currentDist; // Added
46
     currentDist > 0 check
47
         int axis = depth \% 3;
48
49
          Node* next = (axis == 0? target.x < node->point.x:
               (axis == 1? target.y < node->point.y : target.z < node->point.z)) ? node->left :
50
     node->right;
          Node* other = next == node->left? node->right: node->left;
51
52
          nearest(next, target, depth + 1, bestDist);
53
54
55
         long double axisDist = (axis == 0 ? abs(target.x - node->point.x):
                    (axis == 1? abs(target.y - node->point.y): abs(target.z - node->point.z)));
56
         if (axisDist < bestDist) {</pre>
57
           nearest(other, target, depth + 1, bestDist);
58
         }
59
       }
60
61
       void findMinDistanceHelper(Node* node, long double& minDist) {
62
         if (!node) return;
63
64
65
         // Find nearest point to current node's point
         long double currentMin = numeric_limits<long double>::max();
66
         nearest(root, node->point, 0, currentMin);
67
         minDist = min(minDist, currentMin);
68
69
70
         // Recurse on children
         findMinDistanceHelper(node->left, minDist);
71
         findMinDistanceHelper(node->right, minDist);
72
       }
73
74
     public:
75
       KDTree(vector<Point>& points) {
76
77
          root = build(points, 0);
       }
78
79
80
       long double nearest(Point target) {
         long double bestDist = numeric_limits<long double>::max();
81
```

Serso Library

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

kd-tree kactl.cpp

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

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

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

LCA

LCA.cpp

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

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

LCA_in_DSU.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long II;
#define rep(i, st, ed) for(int i = st; i < ed; i++)</pre>
```

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

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

lca_O(1).cpp

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

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

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

Minimum spanning tree

MST_Kruskal.cpp

```
1 struct DSU
2 {
```

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

MST_Prim's.cpp

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

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

Montomic stack and Queue

NxtorPrv_MinorMax.cpp

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

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

monotonicQueue.cpp

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

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

Offline Range Queries

Arithmetic_progression_prefix_sum.cpp

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

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

Ordered set

ordered_set.cpp

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

Serso Library

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

Persistent Segment tree

PersistentSegmentTree_counter.cpp

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

```
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

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

PresistentSegmentTree_serso.cpp

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

```
const int N = le6 + 9, BLOCK_SIZE = 460;
void add(int idx){

void remove(int idx){

void remove(int idx){
```

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

Segment tree

SparseSegmentTree.cpp

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

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

```
62
63
       public:
        SparseSegtree(int n, int q = 0): n(n) {
64
          if (q > 0) { tree.reserve(2 * q * __lg(n)); }
65
          tree.push_back(Node());
66
        }
67
68
        void range_set(int ql, int qr, int val) { range_set(0, 0, n - 1, ql, qr, val); }
69
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 = le9;
78
        SparseSegtree st(RANGE_SIZE + 1, query_num);
79
        int c = 0;
80
        for (int i = 0; i < query_num; i++) {</pre>
81
82
          int type, x, y;
          cin >> type >> x >> y;
83
          if (type == 1) {
84
            c = st.range_sum(x + c, y + c);
85
            cout << c << '\n';
86
          } else if (type == 2) {
87
            st.range_set(x + c, y + c, 1);
88
          }
89
        }
90
```

segmentTree_iterative.cpp

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

```
#define debug(x)
11
12
      #endif
13
      const int N = 2e5 + 9;
      int seg[4 * N];
14
      void update(int k, int x) {
15
        k += N;
16
        seg[k] = x; // update node with value
17
        k >>= 1;
18
        while (k > 0) {
19
          seg[k] = max(seg[2*k], seg[2*k+1]);
20
21
22
       }
23
      }
24
      int merge(int a, int b){
       // write code here
25
        return 0; // return value
26
27
      }
      int query(int a, int b) {
28
        a += N, b += N;
29
        int s = 0;
30
        while (a \le b) {
31
          if (a & 1) {
32
33
            s = merge(s, seg[a]);
34
            a++;
35
          }
          if (~b & 1) {
36
            s = merge(s, seg[b]);
37
38
            b--;
39
          }
          a >>= 1, b >>= 1;
40
41
        }
42
        return s;
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
51
52
```

segtreeBeat.cpp

```
#include <bits/stdc++.h>
      using namespace std;
 2
 3
      using II = long long;
 4
 5
      const int MAXN = 200001; // 1-based
 6
 7
      int N;
      II A[MAXN];
 8
 9
      struct Node {
10
        Il sum; // Sum tag
11
        Il max1; // Max value
12
13
        Il max2; // Second Max value
        Il maxc; // Max value count
14
        Il min1; // Min value
15
        Il min2; // Second Min value
16
        Il minc; // Min value count
17
        Il lazy; // Lazy tag
18
      } T[MAXN * 4];
19
20
      void merge(int t) {
21
        // sum
22
        T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;
23
24
        // max
25
        if(T[t << 1].max1 == T[t << 1 | 1].max1) {
26
          T[t].max1 = T[t << 1].max1;
27
28
          T[t].max2 = max(T[t << 1].max2, T[t << 1|1].max2);
          T[t].maxc = T[t << 1].maxc + T[t << 1| 1].maxc;
29
30
        } else {
          if (T[t << 1].max1 > T[t << 1|1].max1) {
31
            T[t].max1 = T[t << 1].max1;
32
            T[t].max2 = max(T[t << 1].max2, T[t << 1] 1].max1);
33
           T[t].maxc = T[t << 1].maxc;
34
35
         } else {
            T[t].max1 = T[t << 1 | 1].max1;
36
            T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
37
            T[t].maxc = T[t << 1 | 1].maxc;
38
         }
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);
          T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
46
        } else {
47
          if (T[t << 1].min1 < T[t << 1 | 1].min1) {
48
            T[t].min1 = T[t << 1].min1;
49
            T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
50
            T[t].minc = T[t << 1].minc;
51
52
          } else {
53
            T[t].min1 = T[t << 1 | 1].min1;
            T[t].min2 = min(T[t << 1].min1, T[t << 1] 1].min2);
54
            T[t].minc = T[t << 1 | 1].minc;
55
56
          }
        }
57
      }
58
59
60
      void push_add(int t, int tl, int tr, ll v) {
        if (v == 0) { return; }
61
        T[t].sum += (tr - tl + 1) * v;
62
63
        T[t].max1 += v;
        if (T[t].max2! = -IIINF) \{ T[t].max2 += v; \}
64
65
        T[t].min1 += v;
        if (T[t].min2 != IIINF) { T[t].min2 += v; }
66
        T[t].lazy += v;
67
      }
68
69
      // corresponds to a chmin update
70
      void push_max(int t, II v, bool I) {
71
        if (v \ge T[t].max]) \{ return; \}
72
        T[t].sum = T[t].max1 * T[t].maxc;
73
74
        T[t].max1 = v;
        T[t].sum += T[t].max1 * T[t].maxc;
75
76
        if (l) {
77
          T[t].min1 = T[t].max1;
78
        } else {
79
          if (v \leq T[t].min])
            T[t].min1 = v;
80
81
          } else if (v < T[t].min2) {</pre>
82
            T[t].min2 = v;
          }
83
        }
84
      }
85
```

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

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

```
172
          int tm = (tl + tr) >> 1;
173
174
          update\_chmax(l, r, v, t << 1, tl, tm);
          update\_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
175
176
          merge(t);
        }
 177
178
179
        Il query_sum(int I, int r, int t = 1, int tI = 0, int tr = N - 1) {
          if (r < tl | | tr < |) { return 0; }
180
 181
          if (I <= tl && tr <= r) { return T[t].sum; }</pre>
182
          pushdown(t, tl, tr);
183
184
          int tm = (tl + tr) >> 1;
185
          return query_sum(l, r, t << 1, tl, tm) +
              query_sum(1, r, t << 1 | 1, tm + 1, tr);
186
        }
187
188
189
        int main() {
          int Q:
190
 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) {
               int l, r;
199
200
               || x;
               cin >> l >> r >> x;
201
               update_chmin(l, r - 1, x);
202
             } else if (t == 1) {
203
204
               int l, r;
205
               || x;
               cin >> l >> r >> x;
206
207
               update_chmax(l, r - 1, x);
            } else if (t == 2) {
208
               int l, r;
209
210
               || x:
               cin >> l >> r >> x;
 211
               update_add(I, r - 1, x);
212
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
      int N, Q;
 6
 7
      long long tsum[MAXN * 4], tmax[MAXN * 4];
 8
 9
      void update_mod(int I, int r, long long v, int t = 1, int tI = 1, int tr = N) {
        if (r < tl || tr < l || tmax[t] < v) {
10
11
          return;
        } else if (tl == tr) {
12
          int val = tmax[t] % v;
13
          tsum[t] = tmax[t] = val;
14
          return;
15
        }
16
17
        int tm = (tl + tr) / 2;
18
        update_mod(l, r, v, t * 2, tl, tm);
        update_{mod}(I, r, v, t * 2 + 1, tm + 1, tr);
20
        tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
21
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(t| == tr) {
          tsum[t] = tmax[t] = v;
27
28
          return;
        }
29
30
        int tm = (tl + tr) / 2;
31
        if (i <= tm) {
32
33
          update_set(i, v, t * 2, tl, tm);
34
        } else {
          update_set(i, v, t * 2 + 1, tm + 1, tr);
35
36
        tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
37
        tmax[t] = max(tmax[t*2], tmax[t*2+1]);
38
```

Serso Library

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

Sparse table

getIdxSparseTable.cpp

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

sparse_segmentTree.cpp

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

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

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

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

sparse_table.cpp

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

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

Treap

Treap Builtin.cpp

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

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

Treap Implicit.cpp

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

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

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

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

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

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

Serso Library

```
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

```
/// 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
      struct node{
 6
 7
        int key, prior, size;
        struct node *I, *r;
        node() { }
        node(int v) {
10
          key = v;
11
          prior = rand();
12
13
          size = 1;
          I = r = NULL;
14
15
        node (int key, int prior) : key(key), prior(prior), I(NULL), r(NULL) { }
16
      };
17
18
      typedef node* pnode;
19
20
21
      struct Treap{
22
        pnode t;
23
        Treap(){}
24
        /// Returns size of treap
25
        int size(){
26
          return sz(t);
27
        }
28
29
        int sz(pnode t){
30
```

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

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

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

Xor Basis

combining_two_xor_basis.cpp

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

minOrMaxXorPathFrom1ToN.cpp

- 1 /*
- 2 Given an undirected connected graph with non-negative integer edge weights and node numbers from 1
- 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.
- 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 II;
      #define f first
10
      #define s second
11
      const int N = 1e5, LOG = 30;
12
      int Basis[N];
13
14
      void insert(int x){
        for(int i = LOG - 1; i \ge 0; --i){
15
          if(!(x >> i \& 1)) continue;
16
17
          if(Basis[i] == 0){
            Basis[i] = x;
18
19
            return;
          }
20
21
          x ^= Basis[i];
        }
22
23
24
      int min_xor(int x){
        for(int i = LOG - 1; i \ge 0; --i){
25
          if((x ^ Basis[i]) < x) x ^= Basis[i];
26
27
        }
28
        return x;
29
      }
30
31
      int a[N];
32
      bool vis[N];
      vector<pair<int,int>> adj[N];
33
34
      void dfs(int u , int p , int xor_sum){
35
        if(vis[u]){
          insert(xor_sum ^ a[u]);
36
          return;
37
38
        }
39
        vis[u] = 1;
        a[u] = xor_sum;
40
        for(auto &[v, w]: adj[u]) if(v != p){
41
          dfs(v , u , xor_sum ^ w);
42
        }
43
44
      }
      int main(){
45
46
        ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
        int n, m; cin >> n >> m;
47
        for(int i = 0; i < m; ++i){}
48
49
          int u , v , w; cin >> u >> v >> w;
50
          --u; --v;
```

```
adj[u].emplace_back(v, w);

adj[v].emplace_back(u, w);

dfs(0,-1,0);

cout << min_xor(a[n - 1]); // u can replace it with max xor
```

xor_basis_application.cpp

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

```
x ^= Basis[i];
35
        }
36
37
        return true;
38
      }
      int k_th(int k){
39
40
          Finding the k-th smallest xor_sum of all different subsequence xor_sum
41
        */
42
        int low = 1 << sz;
43
        int x = 0;
45
        for(int i = LOG - 1; i \ge 0; --i){
          if(!Basis[i]) continue;
46
          low /= 2;
47
          if( ((x >> i \& 1) \& \& (x >> i \& 1) \& \& (x >> i \& 1) \& \& (x >> i \& 1))
48
49
             x ^= Basis[i];
          }
50
          if(low < k) k = low;
52
        }
53
        return x;
54
      }
      int count(int x){
55
         if(!check(x)) return false;
56
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

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

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

Graph

01_BFS.cpp

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

BFS.cpp

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

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

Bridges.cpp

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

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

Hierholzer.cpp

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

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

```
// we've got the circuit, now print it in reverse
reverse(circuit.begin(), circuit.end());
for(auto &i: circuit) cout << i + 1 << ' ';</pre>
```

SCC_Floyed.cpp

```
#include <bits/stdc++.h>
 1
 2
     typedef long long II;
     #define s second
 3
 4
     #define f first
     #define rep(i, st, ed) for(int i = st; i < ed; i++)
 5
     const int N = 500;
 6
     using namespace std;
 7
 8
     void burn(){
 9
     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
     #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 reach[N][N], dag[N][N];
18
     vector<int> comp , sz;
     void SCC(int n){
19
      comp.clear(); comp.resize(n, -1);
20
      sz.clear();
21
22
      // Floyed
23
      rep(k, 0, n) rep(i, 0, n) rep(j, 0, n)
         reach[i][j] = (reach[i][k] && reach[k][j]); // Warshall Transitive closure
24
      int cnt = 0;
25
      // detect SCC
26
27
      rep(i, 0, n){
       if(comp[i] == -1){
28
29
        comp[i] = cnt++;
30
        rep(j, 0, n)
          if(reach[i][j] \&\& reach[j][i]) comp[j] = comp[i];
31
       }
32
      }
33
34
      sz.resize(cnt);
      rep(i, 0, n) sz[comp[i]]++;
35
      // Create Dag
36
```

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

SCC_kosaraju.cpp

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

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

SCC_tarjan.cpp

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

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

Serso Library

```
lowLink[i] = inStack[i] = deg[i] = 0;
60
61
       }
       for(int i = 0; i < n; i++)
62
        if(dfn[i] == -1) tarjan(i);
63
64
       genDag();
65
      }
66
      int main(){
67
       burn();
68
       int n, m; cin >> n >> m;
69
       adj.resize(n);
       for(int i = 0; i < m; i++){
70
71
        int u , v; cin >> u >> v;
        --U; --V;
72
        adj[u].emplace_back(v);
73
74
       }
75
       SCC(n);
```

artPoints.cpp

```
#include <bits/stdc++.h>
 1
 2
     typedef long long II;
     #define s second
 3
     #define f first
 4
 5
     #define rep(i, st, ed) for(int i = st; i < ed; i++)</pre>
 6
     using namespace std;
 7
     void burn(){
     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
 8
 9
     #ifndef ONLINE_JUDGE
       freopen("in.txt", "r", stdin);
10
       freopen("out.txt", "w", stdout);
11
       freopen("error.txt", "w", stderr);
12
      #endif
13
     }
14
     const int N = 1e5;
15
     //\\//\\//\\//\\//\\//
16
17
     vector<vector<int>> adj;
     int lowLink[N], dfn[N];
18
     set<int>artpoints;
19
20
     int ndfn;
     bool root = false;
21
     void tarjan(int u, int par){
22
      dfn[u] = lowLink[u] = ndfn++;
23
```

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

bellman_ford.cpp

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

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

biConnected.cpp

```
#include <bits/stdc++.h>
typedef long long II;
#define s second
#define f first
#define rep(i, st, ed) for(int i = st; i < ed; i++)</pre>
```

```
using namespace std;
 6
 7
     void burn(){
     ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
 8
     #ifndef ONLINE_JUDGE
 9
       freopen("in.txt", "r", stdin);
10
       freopen("out.txt", "w", stdout);
11
       freopen("error.txt", "w", stderr);
12
13
      #endif
     }
14
     const int N = 1e5;
15
     //\\//\\//\\//\\//\\//
16
     vector<vector<int>> adj;
17
     int lowLink[N], dfn[N];
18
     stack<pair<int,int>> comps;
19
20
     vector<vector<pair<int, int>>>bi_connected;
     int ndfn;
21
     pair<int,int> edge;
22
23
     bool root = false;
24
     void tarjan(int u, int par){
      dfn[u] = lowLink[u] = ndfn++;
25
      for(auto &v : adj[u]){
26
27
       if (v != par \&\& dfn[v] < dfn[u])
28
         comps.push(make_pair(u, v));
29
30
       if(dfn[v] == -1){
31
        tarjan(v, u);
        lowLink[u] = min(lowLink[u], lowLink[v]);
32
33
        if (lowLink[v] >= dfn[u]){
         bi_connected.emplace_back(vector<pair<int,int>>());
34
         dol
35
           edge = comps.top();
36
           comps.pop();
37
           bi_connected.back().emplace_back(edge);
38
39
         }while(edge.first != u || edge.second != v);
40
         reverse(bi_connected.back().begin(), bi_connected.back().end());
41
        }
42
43
       }else if(v != par){
        lowLink[u] = min(lowLink[u], dfn[v]);
44
       }
45
      }
46
     }
47
48
     int main(){
49
      //burn();
```

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

dijkstra.cpp

```
#include <bits/stdc++.h>
 2
      typedef long long II;
      #define s second
 3
      #define f first
 4
      using namespace std;
 5
      int main(){
 6
      /*
 7
        ----- Dijkstra Algrothim -----
        - use to Find Shortest path from Single Sourse to other vertices
 9
        - can used if weight of edge >= 0
10
        - Find answer in O(nlog(n))
11
12
       */
       int n, m; cin >> n >> m; // n: number of vertices, m: number of edges
13
       vector<pair<int,ll>> adj[n]; // {v , w}
14
       for(int i = 0 ; i < m ; i++){}
15
        int u, v; ll w; cin >> u >> v >> w;
16
        adj[u].emplace_back(v , w);
17
        adj[v].emplace_back(u, w);
18
19
20
       int Start; cin >> Start; // vertice that you start from
       Start--:
21
       priority_queue<pair<ll,int>> q; // {dis , u}
22
       vector<II> dis(n, le15);
23
24
       q.push({O , Start});
       dis[Start] = 0;
25
26
       while(q.size()){
```

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

dijkstra_sparse_graph.cpp

```
1
     /*
 2
      O(n^2 + m)
     */
 3
      const int INF = 100000000;
 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);
        p.assign(n, -1);
10
        vector<bool> u(n, false);
11
12
13
        d[s] = 0;
14
        for (int i = 0; i < n; i++) {
          int v = -1;
15
          for (int j = 0; j < n; j++) {
16
            if (!u[j] \&\& (v == -1 || d[j] < d[v]))
17
18
              v = j;
          }
19
20
21
          if(d[v] == INF)
22
            break;
23
          u[v] = true;
24
          for (auto edge : adj[v]) {
25
            int to = edge.first;
26
            int len = edge.second;
27
28
```

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

euler_ciruite_undirectedgraph.cpp

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

```
adj[v].emplace_back(u,i);
32
        }
33
34
        for(int i = 0; i < n; ++i)if ((int) adj[i].size() & 1){
          cout << "IMPOSSIBLE"; // Handling manual</pre>
35
36
          return 0;
        }
37
        // there could be more than one euler circuit if the graph aren't connected ans we will
38
      find one of them;
        stack<int> st;
39
        for(int i = 0; i < n; ++i) if(adj[i].size()){}
40
41
          st.push(i);
          break;
42
        }
43
        // if st.empty() --> no.edges = 0
44
        vector<int> path;
45
        while(!st.empty()){
46
          int v = st.top();
48
          int f=0;
49
          while(!adj[v].empty()) {
            auto [u,i] = adj[v].back();
50
            adj[v].pop_back();
51
            if (!vis[i]) {
52
53
              st.push(u);
              vis[i]=1;
55
              f=1;
              break;
56
            }
57
58
          }
          if (!f){
59
            path.emplace_back(v);
60
            st.pop();
61
62
          }
63
        }
        for (auto &i: path) cout << i + 1 << " ";
64
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
      #include <bits/stdc++.h>
 6
 7
      using namespace std;
 8
      typedef long long II;
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
 9
      #define f first
10
      #define s second
11
12
      const int N = 1e5 + 9; // no.of node
      vector<int> adj[N];
13
      vector<int> path; // euler path
14
15
      void dfs(int s){
        while((int) adj[s].size()){
16
          int u = adj[s].back();
17
          adj[s].pop_back();
18
          dfs(u);
19
        }
20
        path.emplace_back(s);
21
22
      }
23
      int main(){
24
        ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
        #ifndef ONLINE_JUDGE
25
        freopen("in.txt", "r", stdin);
26
        freopen("out.txt", "w", stdout);
27
        freopen("error.txt", "w", stderr);
28
29
        #endif
30
        int n,m; cin >> n >> m;
        int in[n] = {}, out[n] = {};
31
32
        for(int i = 0; i < m; ++i){}
33
          int x,y; cin >> x >> y;
34
          --x; --y;
          adj[x].emplace_back(y);
35
          in[y]++, out[x]++;
36
37
        }
        int a=0,b=0,c=0,s1=0,s2=0;
38
39
        for(int i = 0; i < n; ++i){
          if (in[i]==out[i]) c++;
40
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){
          cout << "IMPOSSIBLE";</pre>
45
          return 0;
46
        }
47
48
        if (!(c==n-2 \&\& a==1 \&\& b==1)){
```

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

euler_path_undirected.cpp

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

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

floyd.cpp

```
int n, m; cin >> n >> m;
vector<vector<ll>> adj(n, vector<ll>(n, OOLL));

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

for(int i = 0; i < m; ++i){
    int u, v; ll w; cin >> u >> v >> w;
    --u; --v;

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>
      using namespace std;
 6
     typedef long long II;
 7
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
 8
      #define f first
 9
      #define s second
10
      int main(){
11
        ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
12
        #ifndef ONLINE_JUDGE
13
        freopen("in.txt", "r", stdin);
14
        freopen("out.txt", "w", stdout);
15
        freopen("error.txt", "w", stderr);
16
        #endif
17
        int n, m, k; cin >> n >> m >> k;
18
19
        vector<pair<int,ll>> adj[n]; // \{v, w\}
        for(int i = 0; i < m; ++i){}
20
          int u, v; ll w; cin >> v >> w;
21
          --U; --V;
22
23
          adj[u].emplace_back(v , w);
24
        }
        priority_queue<pair<|l,int>> q; // {-w, \cup}
25
26
        vector<II> cnt(n), ans;
        int start = 0, end = n - 1;
27
        q.push({O , start});
28
        while(q.size() && cnt[end] < k){
29
30
          auto [d , u] = q.top(); q.pop();
          d *= -1;
31
          cnt[u]++;
32
          if(u == end) ans.emplace_back(d);
33
```

Handbook material

Imgs

Masking

Generate_next_lexicographical_K-combination.cpp

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

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   // proccess the current mask with k 1's
7  }
8   void GospersHack(int n,int k){
```

```
9  int sets=(1||<<k)-1;
10  int limit=(1||<<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

```
/*
Complexity O(3^n)
// m: mask, s: submask
for (int m=0; m<(1<<n); ++m)
for (int s=m; s; s=(s-1)&m)</pre>
```

Math

Counting

BurnsideLemma.cpp

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

- 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 modulo N)'th cell,
- which is in turn the same as its (1+2K modulo N)'th cell...until we get back to the 1st cell again when (P*K % N)=0.

15

- 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 modulo N) etc.

18

- 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 $sum(M^{cd}(K,N))/N$, 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 A0 = rotation by 0, A1 = rotation by 90...A3 = rotation by 270 degree
- 30 So we have cyclic group of 4 elements (possible rotations)

31

- Let's S = RWBR is a valid arrangement where R is on north, W on East, B on south and R on west
- So A1(S) = RRWB(rotation by 90 degree)

34

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

36

For AO we rotate S by O degree. So there are 3⁴ fixed points.

38

- For A1 we rotate S by 90 degree. If S and A1(S) will have to be same then,
- 40 north-east must have same color, east-south must have same color,
- same for south-west and west-north. Which means all side must have same color.
- 42 So there are 3 fixed points for A1.

43

- A3 is same as A1, because rotation by 270 degree does same as rotation by 90 degree.
- 45 So there are 3 fixed points for A3 too.

46

```
For A2 we rotate S by 180 degree. If S and A2(S) will have to be same then,
north-south must have same color and east-west must have same color.
So we have 3*3 fixed points for A2.

So there are total (3^4 + 3 + 3 + 3*3) = 96 fixed points.

And by Burnside's lemma, the answer to the original problem is 96/4 = 24.
```

FFT.cpp

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

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

```
78
        return res;
79
      }
     };
08
81
82
83
     //----//
84
     // Examples
85
     //----//
86
87
88
     int main() {
89
       int N;
       while (cin >> N) {
90
        vector<long long> a(N), b(N);
91
        for (int i = 0; i < N; ++i) cin >> a[i] >> b[i];
92
93
        auto res = FFT::mult(a, b);
94
        cout << 0 << endl;
        for (int i = 0; i < N*2-1; ++i) cout << res[i] << endl;
95
       }
96
```

JosephusTheorem.cpp

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

Matrix_Exponential.cpp

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

Mint.cpp

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

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

NTT_prime_mod.cpp

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

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

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

```
116
           return res;
         }
117
118
         long long modinv(long long a, int mod) {
119
           long long b = mod, u = 1, v = 0;
120
           while (b) {
121
             long long t = a / b;
122
             a = t * b, swap(a, b);
123
             u = t * v, swap(u, v);
124
125
           }
           υ %= mod;
126
           if (u < 0) u += mod;
127
128
           return u;
         }
129
130
131
         int calc_primitive_root(int mod) {
132
           if (mod == 2) return 1;
           if (mod == 167772161) return 3;
133
134
           if (mod == 469762049) return 3;
           if (mod == 754974721) return 11;
135
           if (mod == 998244353) return 3;
136
137
           int divs[20] = {};
           divs[0] = 2;
138
           int cnt = 1;
139
           long long x = (mod - 1) / 2;
140
           while (x \% 2 == 0) x /= 2;
141
           for (long long i = 3; i * i <= x; i += 2) {
142
             if (x \% i == 0) {
143
                divs[cnt++] = i;
144
                while (x \% i == 0) \times /= i;
145
             }
146
           }
147
148
           if (x > 1) divs[cnt++] = x;
149
           for (int g = 2;; g++) {
             bool ok = true;
150
             for (int i = 0; i < cnt; i++) {
151
                if(modpow(g, (mod - 1) / divs[i], mod) == 1) {
152
                  ok = false;
153
                  break;
154
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;
            while (size_a < N) size_a <<=1;
163
            while (size_b < M) size_b <<= 1;
164
            return max(size_a, size_b) << 1;</pre>
165
          }
166
167
          // number-theoretic transform
168
169
          template<class mint> void trans(vector<mint> &v, bool inv = false) {
            if (v.empty()) return;
170
            int N = (int)v.size();
171
172
            int MOD = v[0].get_mod();
            int PR = calc_primitive_root(MOD);
173
            static bool first = true;
174
175
            static vector<long long> vbw(30), vibw(30);
            if (first) {
176
              first = false:
177
              for (int k = 0; k < 30; ++k) {
178
                vbw[k] = modpow(PR, (MOD - 1) >> (k + 1), MOD);
179
                vibw[k] = modinv(vbw[k], MOD);
180
              }
 181
182
            }
            for (int i = 0, j = 1; j < N - 1; j++) {
183
              for (int k = N >> 1; k > (i ^= k); k >>= 1);
184
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];
              if (inv) bw = vibw[k];
189
190
              for (int i = 0; i < N; i += t) {
                mint w = 1;
191
                for (int j = 0; j < t/2; ++j) {
192
                  int j1 = i + j, j2 = i + j + t/2;
193
194
                  mint c1 = v[j1], c2 = v[j2] * w;
                  v[j1] = c1 + c2;
195
                  v[j2] = c1 - c2;
196
                  w *= bw:
197
                }
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
         // for garner
207
         static constexpr int MOD0 = 754974721;
208
209
         static constexpr int MOD1 = 167772161;
         static constexpr int MOD2 = 469762049;
210
         using mint0 = Fp<MOD0>;
211
         using mint1 = Fp<MOD1>;
212
213
         using mint2 = Fp<MOD2>;
214
         static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
215
         static const mint2 imod1 = 104391568; // modinv(MOD1, MOD2);
         static const mint2 imod01 = 187290749; // imod1 / MOD0;
216
217
218
         // small case (T = mint, long long)
         template<class T> vector<T> naive_mul(const vector<T> &A, const vector<T> &B) {
219
           if (A.empty() || B.empty()) return {};
220
221
           int N = (int)A.size(), M = (int)B.size();
           vector<T> res(N+M-1);
222
223
           for (int i = 0; i < N; ++i)
224
             for (int i = 0; i < M; ++i)
               res[i + j] += A[i] * B[j];
225
226
           return res:
         }
227
228
229
         // mul by convolution
         template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint>
230
       &B) {
231
           if (A.empty() || B.empty()) return {};
232
           int N = (int)A.size(), M = (int)B.size();
           if (min(N, M) < 30) return naive_mul(A, B);
233
           int MOD = A[0].get_mod();
234
235
           int size_fft = get_fft_size(N, M);
236
           if (MOD == 998244353) {
             vector<mint> a(size_fft), b(size_fft), c(size_fft);
237
238
             for (int i = 0; i < N; ++i) a[i] = A[i];
             for (int i = 0; i < M; ++i) b[i] = B[i];
239
             trans(a), trans(b);
240
             vector<mint> res(size_fft);
241
242
             for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];
```

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

```
inv_[i] = -inv_[MOD_{i}] * (MOD_{i});
285
286
            finv_{i} = finv_{i} * inv_{i};
          }
287
        }
288
        constexpr T com(int n, int k) const noexcept {
289
          if (n < k || n < 0 || k < 0) return 0;
290
          return fact_[n] * finv_[k] * finv_[n-k];
291
        }
292
        constexpr T fact(int n) const noexcept {
293
          if (n < 0) return 0;
294
          return fact_[n];
295
        }
296
297
        constexpr T inv(int n) const noexcept {
          if (n < 0) return 0;
298
299
          return inv_[n];
300
        }
        constexpr T finv(int n) const noexcept {
301
          if (n < 0) return 0;
302
          return finv_[n];
303
        }
304
305
      };
306
307
308
       //----//
309
      // Examples
310
       //----//
311
312
      const int MOD = 998244353;
313
       using mint = Fp<MOD>;
314
315
      int main() {
316
        int N:
317
        cin >> N;
318
319
        map<int,long long> ma;
        for (int i = 0; i < N*2; ++i) {
320
          int h;
321
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;
         for (auto it: ma) {
328
           int n = it.second:
329
           vector<mint> pol(n/2+1, 1);
330
           for (int i = 0; i <= n/2; ++i) {
331
             pol[i] = bc.fact(n) * bc.finv(n - i*2) * bc.finv(i) / modpow(mint(2), i);
332
333
           }
           que.push({pol.size(), pol});
334
         }
335
         while (que.size() >= 2) {
336
337
           auto f = que.top().second; que.pop();
338
           auto g = que.top().second; que.pop();
           auto h = NTT::mul(f, g);
339
```

NTT_random_mod.cpp

```
#include <bits/stdc++.h>
 1
 2
      using namespace std;
 3
 4
     // modint
 5
      template<int MOD> struct Fp {
 6
        // inner value
 7
       long long val;
 8
 9
10
        // constructor
        constexpr Fp() : val(0) { }
11
        constexpr Fp(long long v) : val(v % MOD) {
12
          if (val < 0) val += MOD;
13
       }
14
15
        constexpr long long get() const { return val; }
        constexpr int get_mod() const { return MOD; }
16
17
        // arithmetic operators
18
        constexpr Fp operator + () const { return Fp(*this); }
19
        constexpr Fp operator - () const { return Fp(0) - Fp(*this); }
20
21
        constexpr Fp operator + (const Fp &r) const { return Fp(*this) += r; }
        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) {
25
26
          val += r.val;
27
          if (val >= MOD) val -= MOD;
          return *this;
28
        }
29
        constexpr Fp& operator -= (const Fp &r) {
30
31
          val = r.val;
          if (val < 0) val += MOD;
32
          return *this:
33
        }
34
        constexpr Fp& operator *= (const Fp &r) {
35
          val = val * r.val % MOD;
36
          return *this;
37
38
        }
        constexpr Fp& operator /= (const Fp &r) {
39
          long long a = r.val, b = MOD, u = 1, v = 0;
40
41
          while (b) {
            long long t = a / b;
42
            a = t * b, swap(a, b);
43
            u = t * v, swap(u, v);
44
45
          }
          val = val * u % MOD;
46
          if (val < 0) val += MOD;
47
          return *this:
48
        }
49
        constexpr Fp pow(long long n) const {
50
51
          Fp res(1), mul(*this);
          while (n > 0) {
52
            if (n \& 1) res *= mul;
53
            mul *= mul;
54
            n >>= 1;
55
56
          }
57
          return res;
        }
58
        constexpr Fp inv() const {
59
60
          Fp res(1), div(*this);
          return res / div;
61
        }
62
63
        // other operators
64
        constexpr bool operator == (const Fp &r) const {
65
          return this->val == r.val;
66
        }
67
```

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

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

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

```
197
                 w *= bw;
               }
198
             }
199
           }
200
           if (inv) {
201
             long long invN = modinv(N, MOD);
202
             for (int i = 0; i < N; ++i) v[i] = v[i] * invN;
203
           }
204
205
         }
206
         // for garner
207
208
         static constexpr int MOD0 = 754974721;
209
         static constexpr int MOD1 = 167772161;
         static constexpr int MOD2 = 469762049;
210
         using mint0 = Fp<MOD0>;
 211
212
         using mint1 = Fp<MOD1>;
         using mint2 = Fp<MOD2>;
213
         static const mint1 imod0 = 95869806; // modinv(MOD0, MOD1);
214
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 {};
           int N = (int)A.size(), M = (int)B.size();
221
222
           vector<T> res(N+M-1);
223
           for (int i = 0; i < N; ++i)
             for (int i = 0; i < M; ++i)
224
               res[i + j] += A[i] * B[j];
225
226
           return res;
227
         }
228
         // mul by convolution
229
230
         template<class mint> vector<mint> mul(const vector<mint> &A, const vector<mint>
       &B) {
231
           if (A.empty() || B.empty()) return {};
           int N = (int)A.size(), M = (int)B.size();
232
233
           if (min(N, M) < 30) return naive_mul(A, B);
234
           int MOD = A[0].get_mod();
           int size_fft = get_fft_size(N, M);
235
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];
             trans(a), trans(b);
240
241
              vector<mint> res(size_fft);
242
              for (int i = 0; i < size_fft; ++i) res[i] = a[i] * b[i];
             trans(res, true);
243
              res.resize(N + M - 1);
244
245
              return res;
           }
246
           vector<mint0> a0(size_fft, 0), b0(size_fft, 0), c0(size_fft, 0);
247
           vector<mint1> a1(size_fft, 0), b1(size_fft, 0), c1(size_fft, 0);
248
           vector<mint2> a2(size_fft, 0), b2(size_fft, 0), c2(size_fft, 0);
249
           for (int i = 0; i < N; ++i)
250
              aO[i] = A[i].val, a1[i] = A[i].val, a2[i] = A[i].val;
251
252
           for (int i = 0; i < M; ++i)
              bO[i] = B[i].val, b1[i] = B[i].val, b2[i] = B[i].val;
253
           trans(a0), trans(a1), trans(a2), trans(b0), trans(b1), trans(b2);
254
           for (int i = 0; i < size_fft; ++i) {
255
             cO[i] = aO[i] * bO[i];
256
257
             c1[i] = a1[i] * b1[i];
              c2[i] = a2[i] * b2[i];
258
259
           }
260
           trans(c0, true), trans(c1, true), trans(c2, true);
           mint mod0 = MOD0, mod01 = mod0 * MOD1;
261
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;
             int y2 = (imod01 * (c2[i] - y0) - imod1 * y1).val;
266
              res[i] = mod01 * y2 + mod0 * y1 + y0;
267
268
           }
269
           return res;
270
         }
271
       };
272
273
274
275
       // Examples
276
277
278
279
       void Yosupo_Convolution_mod_100000007() {
```

check_point_inside_polygon.cpp

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

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

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

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

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

```
200
         };
201
          vec2 p0 = *min_element(a.begin(), a.end(), [](vec2 a, vec2 b) {
202
            return make_pair(a.y, a.x) < make_pair(b.y, b.x);
203
204
         }):
         sort(a.begin(), a.end(), [&p0, &orientation](const vec2& a, const vec2& b) {
205
            int o = orientation(p0, a, b);
206
            if (o == 0)
207
208
              return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
209
            return o < 0;
210
 211
         }):
         if (include_collinear) {
 212
 213
            int i = (int)a.size()-1;
            while (i \ge 0 \&\& collinear(p0, a[i], a.back())) i--;
214
 215
            reverse(a.begin()+i+1, a.end());
         }
 216
 217
          vector<vec2> st;
 218
219
         for (int i = 0; i < (int)a.size(); i++) {
            while (st.size() > 1 \&\& !cw(st[st.size()-2], st.back(), a[i], include_collinear))
220
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);
          for(int i = 0; i < st.size(); i++){
227
            vec2 v0 = st[i];
228
            vec2 v1 = st[(i + 1) % st.size()];
229
230
            if(vO.x == v1.x && vO.y == v1.y) {
              continue:
231
            }
232
233
            ans.push_back(st[i]);
         }
234
235
236
          return ans;
237
       }
238
       //checks if the area of the triangle is the same as the three triangle areas formed by
239
        drawing lines from pt to the vertices.
240
        //i don't think triangle winding order matters
```

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

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

convolution_ntt_optimized.cpp

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

```
int fac[N], inv_fac[N];
 4
 5
      void pre(){
 6
        fac[0] = fac[1] = 1;
        inv_fac[0] = inv_fac[1] = 1;
 7
        for (int i = 2; i < N; ++i) {
 8
          inv_fac[i] = mod - 1|| * mod / i * inv_fac[mod % i] % mod;
 9
10
        }
        for (int i = 2; i < N; ++i) {
11
          fac[i] = 111 * i * fac[i - 1] % mod;
12
          inv_fac[i] = 1|| * inv_fac[i] * inv_fac[i - 1] % mod;
13
        }
14
      }
15
16
17
      Il fast_power(Il a, Il b){
        II res = 1;
18
        while (b){
19
20
          if (b & 1) res = res * a % mod;
          a = a * a % mod;
21
          b >>= 1;
22
        }
23
24
        return res;
25
      }
26
27
      Il mod_inv(Il a){
28
        return fast_power(a, mod - 2);
      }
29
30
      const int root = fast_power(3, 119);
31
      const int root_inv = mod_inv(root);
32
      const int root_pw = 23;
33
34
35
      int reverse(int num, int lg_n){
        int res = 0;
36
37
        for (int i = 0; i < lg_n; ++i) {
          if (num \& (1 << i)) res = 1 << (lg_n - 1 - i);
38
        }
39
40
        return res;
      }
41
42
43
      vector<vector<int>> roots, roots_inv;
      vector<int> rev;
44
45
46
      void pre(int n){
```

```
47
        int lg_n = _lg(n);
        rev.resize(n);
48
        roots.resize(lg_n);
49
        roots_inv.resize(lg_n);
50
        for (int i = 0; i < n; ++i) {
51
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) {
           wlen = 111 * wlen * wlen % mod;
57
           wlen_inv = 1|| * wlen_inv * wlen_inv % mod;
58
        }
59
60
        for (int I = Ig_n - 1; I >= 0; --I) {
61
62
           int len = 1 << (1 + 1);
63
           int w = 1, w_inv = 1;
           roots[l].resize(len / 2);
64
           roots_inv[l].resize(len / 2);
65
           for (int i = 0; i < len / 2; ++i) {
66
67
             roots[l][i] = w;
             roots_inv[l][i] = w_inv;
68
             w = 111 * w * wlen % mod;
69
70
             w_inv = 1|| * w_inv * wlen_inv % mod;
71
           wlen = 111 * wlen * wlen % mod;
72
           wlen_inv = 1|| * wlen_inv * wlen_inv % mod;
73
        }
74
      }
75
76
      void ntt(vector<int>& a, bool invert){
77
78
        int n = a.size();
        for (int i = 1; i < n; ++i) {
79
           if (i < rev[i]) swap(a[i], a[rev[i]]);</pre>
80
81
        }
        int mx = _lg(n);
82
        for (int I = 0; I < mx; ++I) {
83
           int len = 1 << (l + 1), shift = 1 << l;
84
85
           for (int i = 0; i < n; i += len) {
             for (int j = 0; j < len / 2; ++j) {
86
               int w = invert ? roots_inv[l][j] : roots[l][j];
87
               int u = a[i + j], v = 111 * a[i + j + shift] * w % mod;
88
               a[i + j] = (u + v) \% mod;
89
```

```
90
                a[i + j + shift] = (u - v + mod) \% mod;
             }
 91
           }
 92
         }
 93
 94
         if (invert){
 95
           int n_1 = mod_inv(n);
 96
           for(int &x: a) x = 111 * x * n_1 % mod;
 97
         }
 98
 99
       }
100
101
       vector<int> convolve(int n, int m, vector<int> a){
102
         int size = n * m + 1;
         int_n = 1;
103
104
         while (_n < size) _n <<= 1;
         a.resize(_n);
105
         pre(_n);
106
         ntt(a, false);
107
         for (int i = 0; i < n; ++i) a[i] = fast_power(a[i], n);
108
109
         ntt(a, true);
         a.resize(size);
110
```

convolution_using_NTT_kactl.cpp

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

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

derangement.cpp

```
#include <bits/stdc++.h>
typedef long long II;

#define s second
#define f first
const int N = le6;

using namespace std;

II D[N]
int main(){
```

Serso Library

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

evaluating_polynomial_at_n_points.cpp

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

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

lucas.cpp

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

Serso Library

nCr.cpp

```
1  || bpow(|| n, || x) { return !x ? 1 : bpow(n * n % mod, x >> 1) * (x & 1 ? n : 1) % mod; }
2  || linv(|| b) { return bpow(b, mod - 2); }
3  || 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  || C(|| n, || k) { return fact[n] * factinv[n - k] % mod * factinv[k] % mod; }</pre>
```

nCr_DP.cpp

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

nCr_O(r).cpp

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

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

nCr_mod2.cpp

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

ncr_O(n).cpp

```
const int N = 5e6 + 100;
const int mod = 1e9 + 7;

ll fact[N];

ll inv[N]; //mod inverse for i

ll invfact[N]; //mod inverse for i!

void init() {
  fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;

for (long long i = 2; i < N; i++) {</pre>
```

```
fact[i] = (fact[i-1] * i) % mod;
          inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
10
11
          invfact[i] = (inv[i] * invfact[i - 1]) % mod;
       }
12
      }
13
14
      Il 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]) %
17
            mod:
18
19
```

ntt_optimized.cpp

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

```
29
      void pre(int n){
30
        int \lg_n = _{\lg(n)};
31
32
        rev.resize(n);
        roots.resize(lg_n);
33
        roots_inv.resize(lg_n);
34
        for (int i = 0; i < n; ++i) {
35
           rev[i] = reverse(i, lg_n);
36
        }
37
38
39
        int wlen = root, wlen_inv = root_inv;
        for (int i = 0; i < root_pw - lg_n; ++i) {
40
           wlen = 111 * wlen * wlen % mod;
41
          wlen_inv = 1|| * wlen_inv * wlen_inv % mod;
42
        }
43
44
        for (int I = Ig_n - 1; I >= 0; --I) {
45
46
           int len = 1 << (l + 1);
           int w = 1, w_inv = 1;
47
           roots[l].resize(len / 2);
48
           roots_inv[l].resize(len / 2);
49
50
           for (int i = 0; i < len / 2; ++i) {
             roots[l][i] = w;
51
             roots_inv[l][i] = w_inv;
52
53
             w = 111 * w * wlen % mod;
             w_inv = 1|| * w_inv * wlen_inv % mod;
54
          }
55
56
          wlen = 111 * wlen * wlen % mod;
           wlen_inv = 1|| * wlen_inv * wlen_inv % mod;
57
        }
58
59
      }
60
61
      void ntt(vector<int>& a, bool invert){
        int n = a.size();
62
63
        for (int i = 1; i < n; ++i) {
           if (i < rev[i]) swap(a[i], a[rev[i]]);</pre>
64
        }
65
        int mx = __lg(n);
66
        for (int I = 0; I < mx; ++I) {
67
68
           int len = 1 << (l + 1), shift = 1 << l;
           for (int i = 0; i < n; i += len) {
69
             for (int i = 0; i < len / 2; ++i) {
70
71
               int w = invert ? roots_inv[l][j] : roots[l][j];
72
               int u = a[i + j], v = 111 * a[i + j + shift] * w % mod;
```

```
a[i + j] = (u + v) \% mod;
73
74
               a[i + j + shift] = (u - v + mod) \% mod;
            }
75
          }
76
        }
77
78
79
        if (invert){
          int n_1 = mod_inv(n);
80
          for(int &x: a) x = 111 * x * n_1 % mod;
81
        }
82
      }
83
84
      vector<int> convolve(int n, int m, vector<int> a){
85
        int size = n * m + 1;
86
        int _n = 1;
87
        while (_n < size) _n <<= 1;
88
89
        a.resize(_n);
        pre(_n);
90
        ntt(a, false);
91
        for (int i = 0; i < n; ++i) a[i] = fast_power(a[i], n);
92
93
        ntt(a, true);
        a.resize(size);
94
95
        return a;
```

simpson_integration.cpp

```
const int N = 1000 * 1000; // number of steps (already multiplied by 2)
2
3
     double simpson_integration(double a, double b){
        double h = (b - a) / N;
4
5
        double s = f(a) + f(b); // a = x_0 and b = x_2
        for (int i = 1; i \le N - 1; ++i) { // Refer to final Simpson's formula
6
          double x = a + h * i;
7
          s += f(x) * ((i \& 1) ? 4 : 2);
8
9
        }
        s *= h / 3;
10
11
        return s;
12
```

startbars.cpp

```
const int mod = le9 + 7;
      const int N = 5e6 + 9;
 2
 3
      II fac[N]. inv[N]:
      void preprocess() {
 4
        for (int i = 0; i < N; i++) {
 5
           if (i < 2) {
 6
 7
             fac[i] = inv[i] = 1;
          } else {
 8
             fac[i] = 111 * i * fac[i - 1] % mod;
 9
             inv[i] = mod - 1|| * mod / i * inv[mod % i] % mod;
10
          }
11
        }
12
        for (int i = 2; i < N; i++) {
14
           inv[i] = 111 * inv[i] * inv[i - 1] % mod;
        }
15
      }
16
      || ncr(|| n, || r) {
17
        if(r > n || n < 0 || r < 0) return 0; // manual handling
18
        return 1|| * fac[n] * (1|| * inv[r] * inv[n - r] % mod) % mod;
19
20
21
      Il starsBars(Il stars, Il boxes) {
        if (boxes == 0) return stars == 0;
22
        return ncr(stars - 1, boxes - 1);
23
24
```

stirling.cpp

```
1
2
       #Stirling numbe
       1. S(r, n), represents the number of ways that we can arrange r objects around
3
     indistinguishable circles of length n,
       and every circle n must have at least one object around it.
4
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)
       Time Complexity: O(r * n)
8
9
       Auxiliary Space : O(r * n)
10
     int stirling_number(int n,int k){
11
       if(k==0)return n==0;
12
       if(n==0)return 0;
13
```

```
return stirling_number(n-1,k-1)+(n-1)* stirling_number(n-1,k);

return stirling_number(n-1,k-1)+(n-1)* stirling_number(n-1,k);
```

sum_xor_range.cpp

Game Theory

Grundy 01.cpp

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

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

Grundy 02.cpp

```
    /// Include My Code Template
    #include <bits/stdc++.h>
    using namespace std;
    /**
    Initially there are n piles.
    A pile is formed by some stones.
```

```
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.
        If a player cannot do so, he/she loses the game.
11
      **/
12
13
14
      #define Size 100005
15
      int N;
16
17
      int A[1005];
18
      int DP[10005];
19
      int call(int cur){
20
        if(cur%2 == 0) return cur/2;
21
22
        while(cur%2 != 0) cur/=2;
23
        return cur/2;
24
      }
25
26
      int main(){
27
        int nCase;
        sf("%d",&nCase);
28
        mems(DP,-1);
29
30
        for(int cs = 1;cs \le nCase;cs + +){
          sf("%d",&N);
31
32
          int res = 0;
          for(int i = 0;i < N;i++){
33
            sf("%d",&A[i]);
34
35
            res = res ^ call(A[i]);
36
          if(res == 0) pf("Case %d: Bob\n",cs);
37
          else pf("Case %d: Alice\n",cs);
38
39
        }
40
        return 0;
41
```

Grundy String Game.cpp

```
/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;
/**
```

```
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
      #define Size 100005
12
13
14
      int N;
15
      int DP[255];
16
      vector<int> res;
      string s;
17
18
      int call(int cur){
19
        if(cur <= 0) return DP[cur] = 0;</pre>
20
21
        if(DP[cur] != -1) return DP[cur];
        vector<int> grundy;
22
23
        for(int p = 0; p < = cur/2; p++){
24
          int d1 = p - 2, d2 = cur - p - 2 - 1;
25
          grundy.push_back(call(d1) ^ call(d2));
26
        }
27
28
        make_unique(grundy);
29
        if(grundy.size() == 0 || grundy[0] != 0) return DP[cur] = 0;
30
        int f = 0;
31
        for(int i = 1; i < grundy.size(); i++){
32
33
          if(grundy[i] == f) continue;
          f++:
34
          if(grundy[i] != f) return DP[cur] = f;
35
        }
36
        return DP[cur] = grundy[grundy.size()-1] + 1;
37
38
      }
39
40
      bool isPossible(int pos){
41
        string ss = s;
42
        if(ss[pos] == 'X') return false;
43
        ss[pos] = 'X';
        for(int i = 1; i < N-1; i++){}
44
45
          if(ss[i] == 'X'){
            if(ss[i-1] == 'X' && ss[i+1] == 'X') return true;
46
          }
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] == '.'){
             if(ss[i-1] == 'X' \&\& ss[i+1] == 'X') return false;
53
          }
54
55
        }
56
        int i = 0, last X = -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);
            lastX = i;
62
63
            j++;
64
             break;
65
          }
66
          i++;
67
        }
        while(i<N){
68
          if(ss[i] == 'X'){
69
            int cnt = i - last X - 2 - 2 - 1;
70
71
            xorr = xorr ^ call(cnt);
             lastX = i;
72
          }
73
74
          j++;
        }
75
76
        int cnt = i - lastX - 2 - 1;
        xorr = xorr ^ call(cnt);
77
        if(xorr == 0) return true;
78
        return false;
79
80
      }
81
      int main(){
82
        fast_cin;
83
84
        mems(DP,-1);
85
        int nCase;
        cin >> nCase;
86
        for(int cs = 1;cs \le nCase;cs + +){
87
88
          cin >> s;
          N = s.length();
89
90
          res.clear();
          for(int i = 0;i < N;i++){
91
92
             if(isPossible(i) == true){
```

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

Grundy_Knights_Move_In_Matrix.cpp

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

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

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

```
4
     /**
 5
        Each cell of matrix is a pile.
 6
        In each move a player can move stone to down or right cell.
        Who gives last move win.
 8
 9
10
11
      #define INF 99999999
12
13
14
      int main() {
15
        int nCase,r,c,a;
16
        sf("%d", &nCase);
17
        for (int cs = 1; cs \leq nCase; cs++) {
18
          scanf("%d %d", &r, &c);
19
          int xsum = 0;
20
21
          for(int i = 1;i<=r;i++){
            for(int j = 1; j <= c; j++){
22
              sf("%d",&a);
23
24
              if((r+c)\%2!=(i+j)\%2) xsum^= a;
            }
25
          }
26
          if(xsum != 0) printf("Case %d: win\n", cs);
27
28
          else printf("Case %d: lose\n", cs);
29
        }
        return 0;
30
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
     #include <bits/stdc++.h>
6
7
     using namespace std;
8
     typedef long long II;
     #define f first
9
     #define s second
10
```

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

Nim 1.cpp

```
/// Include My Code Template
 2
     #include <bits/stdc++.h>
     using namespace std;
 3
 4
     /**
 5
       Given N piles a player can remove any number of stone from a pile to it's left.
 6
      **/
 7
 8
     #define INF 99999999
 9
10
11
     int main() {
        int nCase,n,a,b;
12
        sf("%d", &nCase);
13
        for (int cs = 1; cs \leq nCase; cs++) {
14
15
          scanf("%d", &n);
         int xsum = 0;
16
          for(int i = 0; i < n; i++) {
17
            scanf("%d", &a);
18
            xsum ^= (a);
19
         }
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

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

Serso Library

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

General

_int128.cpp

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

comp_double.cpp

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

```
// Double compare
if (fabs(a - b) < EPS)
return 0;
return (a > b ? 1 : -1);
}
```

convert_double_int.cpp

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

floor_ceil.cpp

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

fraction.cpp

```
long long gcd (long long a, long long b) {
while (b) {
```

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

generate_n_digits_after_point.cpp

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

10

manual_multiply.cpp

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

random.cpp

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

Geometry

3dGeometry.cpp

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

31

Distinct_line_detecting.cpp

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

Geometry.cpp

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

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

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

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

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

```
191
       int circleCircleIntersection(const Point<> &c1, const double &r1, const Point<> &c2,
       const double &r2, Point<> &res1, Point<> &res2) {
           if (c] == c2 && fabs(r1 - r2) < EPS) {
192
193
               res1 = res2 = c1;
               return fabs(r1) < EPS?1: INT32_MAX;
194
           }
195
196
           double len = length(vec(c1,c2));
           if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
197
               Point <> d, c;
198
               double r:
199
               if(r1 > r2)
200
201
                   d = vec(c1,c2), c = c1, r = r1;
202
               else
                   d = vec(c2,c1), c = c2, r = r2;
203
               res1 = res2 = normalize(d) * r + c;
204
               return 1; // intersect in one point
205
206
           }
           if (len > r1 + r2 || len < fabs(r1 - r2))
207
               return 0; // intersect on two points
208
209
           double a = cosRule(r2, r1, len);
           Point<> clc2 = normalize(vec(cl,c2)) * rl;
210
 211
           resl = polar(clc2, a) + cl;
           res2 = polar(clc2, -a) + cl;
212
           return 2; // intersect in one point
213
214
215
       // Circle From 3 Points
216
       bool circle3(const Point<> &p1, const Point<> &p2, const Point<> &p3,
           Point<>& cen. double& r) {
217
           Point <> m1 = (p1 + p2) / 2;
218
           Point <> m2 = (p2 + p3) / 2;
219
220
           Point<> perp1 = vec(p1, p2);
221
           perpl = perpl.perp();
           Point<> perp2 = vec(p2, p3);
222
           perp2 = perp2.perp();
223
           bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
224
225
           r = length(vec(cen,p1));
           return res;
226
       }
227
       double lengthSqr(const Point<> a){ return a.x * a.x + a.y * a.y; }
228
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));
```

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

Point.cpp

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

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

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

Triangle.cpp

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

```
inline Id cosRule(Id a, Id b, Id c) {
    Id denom = 2.0L * b * c;
    if (fabs(denom) < le-9) return 0.0L;
    return acos(min(max((b * b + c * c - a * a) / denom, -1.0L), 1.0L));
}
int main() { return 0; }</pre>
```

areaOfreactangles.cpp

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

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

circle.cpp

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

```
}
44
45
                       // Circle from Three Points: returns true if successful
                       bool circle3(const Point&p1, const Point&p2, const Point&p3, Point&cen, Id&r){
47
                               Id a = cross(p1, p2, p3);
48
                               if(abs(a) < EPS) return false; // Collinear points</pre>
49
                               cen = ((norm(p2) - norm(p1)) * Point(0,1) - (norm(p3) - norm(p1)) * Point(1,0)) / (2*a) + (norm(p3) - norm(p3) - norm(p3)) / (2*a) + (norm(p3) - norm(p3)) / (2*a) + (
50
                       pl;
                               r = abs(cen - p1);
 51
 52
                               return true;
                       }
 53
54
                       // Point Position Relative to Circle: -1 inside, 0 outside, 1 on boundary
 55
                       int circlePoint(const Point& cen, Id r, const Point& p){
56
                               Id dist2 = norm(p - cen), r2 = r*r;
 57
                               if(abs(dist2 - r2) < EPS) return 1;</pre>
58
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

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

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

dynamic_convex_hull.cpp

```
typedef long double ld;
struct Line {
    Il m, b;
    mutable function<const Line *()> succ;

bool operator<(const Line &other) const {
    return m < other.m;
}</pre>
```

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

getClosestPair.cpp

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

line.cpp

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

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

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

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

minimum_enclosing_circle.cpp

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

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

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

polygon.cpp

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

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

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

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

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    }</pre>
```

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

Number theory

CRT-Offline.cpp

```
    /// Chinese Reminder Theorem
    /// Returns the smallest number x such that,
    /// x % num[i] = rem[i] for each i
    /// Numbers in num[] are pairwise co prime
```

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

CRT-Online.cpp

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

CRT-OnlineNonCoPrimeModuli.cpp

```
// #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
       1. Call clear()
 5
        2. Add equations using addEquation() method
 6
        3. Call solve() to get \{x, N\} pair, where x is the unique solution modulo N.
 7
 8
        Assumptions: LCM of all mods will fit into long long.
     */
 9
10
11
      class ChineseRemainderTheorem {
        typedef long long vlong;
12
        typedef pair<vlong,vlong> pll;
13
14
        /** CRT Equations stored as pairs of vector. See addEgation()*/
15
        vector<pll> equations;
16
17
     public:
18
        void clear() {
19
          equations.clear();
20
       }
21
22
23
        /** Add equation of the form x = r \pmod{m}^*
        void addEquation( vlong r, vlong m ) {
24
25
          equations.push_back({r, m});
26
       }
        pll solve() {
27
28
          if (equations.size() == 0) return {-1,-1}; /// No equations to solve
29
          vlong a1 = equations[0].first;
30
          vlong m1 = equations[0].second;
31
          al \%= ml;
32
          /** Initially x = a_0 (mod m_0)*/
33
34
          /** Merge the solution with remaining equations */
35
          for (int i = 1; i < equations.size(); i++) {
36
37
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;
38
39
40
            vlong g = \underline{gcd}(m1, m2);
            if (a1 % g!= a2 % g) return \{-1,-1\}; /// Conflict in equations
41
42
            /** Merge the two equations*/
43
44
            vlong p, q;
```

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

Euler Phi.cpp

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

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

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

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

binary_gcd.cpp

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

discrete_log.cpp

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

```
if (b & 1) {
 5
            res = (res * 111 * a) % m;
 6
          }
          a = (a * 111 * a) % m;
 7
          b >>= 1;
 8
        }
 9
10
        return res;
11
      }
12
      int solve(int a, int b, int m) {
13
        a \% = m, b \% = m;
14
        int n = sqrt(m) + 1;
15
        map<int, int> vals;
16
        for (int p = 1; p \le n; ++p)
17
          vals[powmod(a, p * n, m)] = p;
18
        for (int q = 0; q \le n; ++q) {
19
          int cur = (powmod(a, q, m) * 111 * b) % m;
20
21
          if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
22
23
            return ans;
          }
24
25
        }
26
        return -1;
27
```

extended_euclidean.cpp

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

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

mobius.cpp

```
mobius Function (m)
2
3
      if i has a squared factor: m(i) = 0
      else m(i) = (-1)^r, r : number of distinct prime the i has
4
      */
5
      const int N = 1e5;
6
7
      int mobius[N], sieve[N];
8
      void gen_mobius(){
      for(int i = 1; i < N; i++) \{mobius[i] = sieve[i] = 1;\}
9
       sieve[1] = 0;
10
11
      for(long long i = 2; i < N; i++){
        if(sieve[i]){
12
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
        Modular multiplicative inverse when M and A are coprime or gcd(A, M) = 1:
 3
 4
        Time Complexity: O(log M)
 5
        Auxiliary Space: O(1)
 6
      */
 7
 8
      \| gcd(\| a, \| b, \| \& x, \| \& y) \|
        x = 1, y = 0;
 9
        ||x| = 0, y| = 1, a| = a, b| = b;
10
        while (b1) {
11
12
          int q = a1 / b1;
          tie(x, x1) = make_tuple(x1, x - q * x1);
13
14
          tie(y, y1) = make_tuple(y1, y - q * y1);
          tie(a1, b1) = make_tuple(b1, a1 - q * b1);
        }
16
17
        return al;
18
      \| \operatorname{inv}(\| A, \| M) \|
19
        // modular inverse of A mod M
20
        \|x, y\|
21
22
        IIg = gcd(A,M,x,y);
        if(g != 1){ // Inverse doesn't exist
23
          exit(3);
24
        }
25
        II res = (x \% M + M) \% M;
26
27
        return res;
      }
28
29
30
31
      ## Fermat's little theorem
32
       Modular multiplicative inverse when M is prime:
33
       Time Complexity: O(log M)
34
       Auxiliary Space: O(1)
35
```

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

optimized_sieve_LPF.cpp

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

Notes

Searching

counting_sort.cpp

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

double_ternary_search.cpp

```
long double I = 0, r = le9, mid;
int cnt = 400;
while(cnt--){
double g = I + (r - I) / 3,
    h = r - (r - I) / 3;
if(f(g) < f(h)) r = h; // get minumum value
else I = g;
}</pre>
```

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

minNumberOfSwapsToSortTwoBinaryString.cpp

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

patient_sort.cpp

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

radix_sort.cpp

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

Snippet

clock.cpp

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

fastCode.cpp

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

gcc_optimize.cpp

```
// reduce the runnig time
pragma GCC optimize("Ofast")

#pragma GCC target("avx2,bmi,bmi2,popcnt,lzcnt")

#pragma GCC optimize("Ofast,unroll-loops")
```

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

Standard Problems

Geometry

check_point_in_convex.cpp

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

```
26
      bool collinear(Point a, Point b, Point c) { return orientation(a, b, c) == 0; }
27
28
29
      void convex_hull(vector<Point>& a, bool include_collinear = false) {
        Point pO = *min_element(a.begin(), a.end(), [](Point a, Point b) {
30
          return make_pair(a.y, a.x) < make_pair(b.y, b.x);
31
        });
32
        sort(a.begin(), a.end(), [&p0](const Point& a, const Point& b) {
33
          int o = orientation(p0, a, b);
34
          if (o == 0)
35
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
36
              (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
37
          return o < 0;
38
        });
39
        if (include_collinear) {
40
          int i = (int)a.size()-1;
41
          while (i \ge 0 \&\& collinear(p0, a[i], a.back())) i--;
42
          reverse(a.begin()+i+1, a.end());
43
        }
44
45
        vector<Point> st;
46
47
        for (int i = 0; i < (int)a.size(); i++) {
          while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
48
            st.pop_back();
          st.push_back(a[i]);
50
        }
51
52
53
        a = st;
54
      double cross(Point a, Point b){ return a.x*b.y - b.x*a.y; }
55
      int cross(Point a, Point b, Point c){
56
        Point d = vec(b,a), e = vec(c,a);
57
58
        return cross(d,e);
59
      }
      double dot(Point a, Point b){
60
        return a.x*b.x + a.y*b.y;
61
62
      }
      bool pointOnLine(Point a, Point b, Point p){
63
        return fabs(cross(vec(a,b),vec(a,p))) < eps;</pre>
64
65
      }
66
      bool pointOnRay(Point a, Point b, Point p){
        return dot(vec(a,p),vec(a,b)) > -eps;
67
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
       vector < Point > v(N);
 73
       void prepare(int n) {
 74
 75
         int pos = 0;
 76
         for (int i = 0; i < n; i++) {
           if (make_pair(v[i].x, v[i].y) < make_pair(v[pos].x, v[pos].y))</pre>
 77
             pos = i;
 78
 79
         rotate(v.begin(), v.begin() + pos, v.end());
 80
 81
       void print(Point p){
 82
         cerr << p.x << " " << p.y << "\n";
 83
       }
 84
 85
       bool fun(Point p,int n)
 86
         int idx = 1;
 87
         if(pointOnsegment(v[0],v[n-1],p)) return 1;
 88
         if(cross(v[0],v[n-1],p) < 0) return 0;
 89
         if(cross(v[0],v[1],p) > 0) return 0;
 90
         int l = 1, r = n - 2;
 91
         int ans = -1;
 92
         while(r >= 1)
 93
         {
 94
           int mid = l + (r-l)/2;
 95
           if(cross(v[0],v[mid],p) \le 0) | = mid+1, ans = mid;
 96
 97
           else r=mid-1;
         }
 98
         if(ans == -1) return 0;
 99
         return (cross(v[ans],v[(ans+1)%n],p) <= 0);
100
101
       }
       double area(Point a, Point b, Point c)
102
103
         return 0.5*abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
104
105
       void solve()
106
107
108
         int n,m; cin >> n >> m;
         vector<Point> vans(n);
109
         for(int i=0; i < n; ++i)
110
         {
 111
           cin >> vans[i].x >> vans[i].y;
112
```

```
113
         }
         for(int i=0; i < m; ++i) cin >> v[i].x >> v[i].y;
114
         convex_hull(v);
115
         prepare(v.size());
116
117
         m = v.size();
         int ans = 0;
118
         for(int i=0; i < n; ++i) ans += fun(vans[i],m);
119
         cout << ans << "\n";
120
       }
121
       int32_t main()
122
123
124
         int t=1; //cin >> t;
         while(t--)
125
         {
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>
     using namespace std;
 5
     typedef long long II;
 6
 7
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
      #define f first
 8
 9
      #define s second
      int EPS = 10000, Precision = 4;
10
77
      long long read(){
        // 1.2345 --> 12345
12
        string s;
13
        cin >> s;
14
        int x=s.find('.');
15
        if(x==-1) return stoll(s+"0000");
16
        string one=s.substr(0,x);
17
        string two=s.substr(x+1);
18
19
        while(two.size()< Precision) two+="0";</pre>
        return stoll(one+two);
20
21
      Il ceil(II a, II b){
22
        if(b < 0) a *= -1, b *= -1;
23
```

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

point_in-shape.cpp

Serso Library

```
1
 2
        Problem Link: https://codeforces.com/gym/104447/problem/H
 3
        You are given a polygon of n, vertices and q queries.
 4
        Each query consists of a point (x,y)
 5
        and you have to check if it is inside (including the borders) or outside the polygon.
 6
 7
        The points of the polygon are given in clockwise order with the property that either
 8
      xi=xi-1
        or yi=yi-1
 9
        (but not both), indicating that the edges are parallel to either the x- or y-axis, Also no
10
      two edges intersect
        (endpoints are not considered into the intersections).
11
12
        The first line of the input contains an integer n 4 \le n \le 1e5), representing the number of
13
      vertices in the polygon.
14
        The next n lines each contain two integers xi and yi (0≤xi,yi≤106), representing the x-
15
      and y-coordinates, respectively, of the i
        -th vertex of the polygon in clockwise order.
16
        The next q lines each contain two integers x and y (0 \le x,y \le 1e6), representing the x-
17
      and y-coordinates,
        respectively, of a point to be checked whether it is inside or outside the polygon.
18
      */
19
      #include<iostream>
20
21
      #include <bits/stdc++.h>
22
      #define II long long
23
      #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
24
25
      using namespace std;
      const int N = 1e6 + 5, mod = 1e9 + 7, M = 17, inf = 2e9, sq = 632;
26
      vector<vector<int> > x[N], y[N], queries[N];
27
28
29
      bool inside(vector<vector<int> > &v, int val) {
        int low = 0, high = v.size() - 1;
30
        while (low <= high) {
31
          int mid = low + high >> 1;
32
          if(v[mid][1] < val){
33
            low = mid + 1;
34
          else if (v[mid][0] > val) {
35
36
            high = mid - 1;
          } else {
37
            return true;
38
          }
39
```

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

```
83
           }
         }
84
         for (int i = 0; i < N; i++) {
85
           sort(x[i].begin(), x[i].end());
86
           sort(y[i].begin(), y[i].end());
 87
         }
88
         int q;
89
         cin >> q;
90
         vector<int> ans(q, 0);
 91
         for (int i = 0; i < q; i++) {
92
93
           int a, b;
           cin >> a >> b;
94
            b++:
95
           if (inside(x[a], b) || inside(y[b], a)) {
96
97
              ans[i] = true;
           } else {
98
99
              queries[a].push_back({b, i});
           }
100
101
         }
         for (int i = 0; i < N; i++) {
102
103
           for (auto j: x[i]) {
104
              add_range(j[0] + 1, j[1] - 1);
              freq[j[2]][j[0]]++;
105
              freq[j[2]][j[1]]++;
106
107
           for (auto j: queries[i]) {
108
109
              int cnt = query(j[0]);
              int cntUP = freq[1][j[0]];
110
              int cntDown = freq[0][j[0]];
111
              cnt += (max(cntDown, cntUP) - min(cntDown, cntUP)) / 2;
112
              ans[j[1]] = cnt % 2;
113
           }
114
115
         }
         for (auto i: ans) {
116
           cout << (i ? "YES\n" : "NO\n");
117
         }
118
119
       }
120
121
122
       int main() {
         10
123
         int t = 1;
124
125
        // cin >> t;
```

Implementation

coloring_matrix_with_largest_area.cpp

```
1
 2
          Problem Link:- https://codeforces.com/gym/104874/problem/K
        Given a matrix n x m, each cell (i, j) can be
        1. g(i,j) = '.' -> empty cell
 4
        2. g(i, j) = A' king, "There is exactly one letter 'A'"
 5
        3. g(i, j) = child represented as charachter between 'z' -> 'z', it's proved that letters
      are distinct.
 7
        Output the same matrix, replacing each character '.' with the lowercase letter,
        corresponding to the owner of the containing this cell
        "where e area of the king that belongs to his favorite child is as large as possible."
        Test Cases:
10
        Input:
11
          68
12
          ....X.
13
14
         .F.....
          ...A....
15
16
          ....P..
17
          ..L....
18
        Output:
19
          xxxxxxXx
20
          fFaaaaaa
21
          ffaAaaaa
22
23
          ffaaaaaa
24
          IIIIPpp
25
          IILIIppp
26
      #include <bits/stdc++.h>
27
      using namespace std;
28
29
      typedef long long II;
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
30
      #define f first
31
32
      #define s second
     const int N = 1001;
33
```

```
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 \ge 0 \&\& x < n; }
      bool validY(int y) { return y \ge 0 \& y < m; }
38
      void solveRange(int x1, int y1, int x2, int y2, vector<vector<char>> &g) {
39
         if (!validX(x1) || !validX(x2) || !validY(y1) || !validY(y2))
40
41
           return;
         for (int i = x1; i \le x2; i++) {
42
           int last = y1 - 1;
43
44
           char lastC = ' ';
           for (int j = y1; j \le y2; j++) {
45
             if (g[i][j] != '.') {
46
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;
               lastC = cur;
52
             }
53
           }
54
55
           if (lastC != ' ') {
             for (int j = y2; j > last; j--) {
56
57
               g[i][j] = lastC;
             }
58
           } else {
59
             if (i - 1 \ge x) \& g[i - 1][y] != '.') {
60
61
               for (int j = y1; j \le y2; j++) {
62
                  g[i][j] = tolower(g[i - 1][j]);
               }
63
64
             }
           }
65
66
         }
         for (int i = x2; i >= x1; i--) {
67
           if(g[i][y1] == '.'){
68
             assert(i!= x2);
69
70
             for (int j = y1; j \le 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 \le x2; i++) {
```

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

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

divide_grid_into_k_connctedcompented_01.cpp

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

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

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

Number_Theory

count_no_subsequec_with_lcm.cpp

```
1
      Problem Link: https://atcoder.jp/contests/abc349/tasks/abc349_f
2
      Problem description:
3
       You are given a sequence of positive integers
4
       A=(A1, A2,..., An) of length N and a positive integer M. Find the number, modulo
5
     998244353, of non-empty and not necessarily contiguous subsequences of
       A such that the least common multiple (LCM) of the elements in the subsequence is
6
7
       M. Two subsequences are distinguished if they are taken from different positions in the
     sequence, even if they coincide as sequences. Also, the LCM of a sequence with a single
     element is that element itself.
8
9
       Constraints
       1<= N <= 2e5
10
       1 <= M <= 1e16
11
       1 <= Ai <= 1e16
12
13
14
      My sol:
        Time complexity: O((2^x)^3) --> can be decreased to O(3^x) using submasking
15
     method
```

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

Serso Library

```
60
        }
61
        if(M == 1){cout << (bpw(2, one) - 1 + mod) \% mod; return 0;}
        vector<ll>dp(1 << m);
62
        dp[0] = bpw(2, cnt[0]);
63
        for(int i = 1; i < (1 << m); ++i){
64
65
          IIc = bpw(2, cnt[i]) - 1;
          // Previous mask
66
          for(int j = (1 << m) - 1; j >= 0; --j){
67
            dp[i | j] += dp[j] * c;
68
69
            dp[i | j] %= mod;
          }
70
71
        }
        cout << dp[(1 << m) - 1];
72
```

count_subsequece_gcd_equal_1.cpp

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

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

Tree

No_of_paths_insides_paths.cpp

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

Serso Library

```
- Given n towns connected by n-1 roads forming a tree
 5
      - m trade agreements between towns
      - Each agreement affects all towns on shortest path between si and ti
 6
 7
      - Towns u,v can trade if shortest path between them covered by an agreement
      - Must count valid trading pairs where u<v
 8
 9
10
      Input:
11
      - n, m (2 \le n, m \le 105)
      - n-1 lines: xi,yi road connections
12
13
      - m lines: si,ti trade agreement paths
14
      Output:
15
      - Number of valid trading pairs
16
17
      #include <bits/stdc++.h>
18
19
      using namespace std;
20
      typedef long long II;
      #define f first
21
22
      #define s second
23
      const int N = le5 + 9; // TODO: change it to maximum possible N
24
      const int LOG = 17;
      int dfs_time = 0, cur_pos = 0;
25
      int st[N], ft[N], big[N], ver[N], sz[N];
26
27
      int head[N], par[N], dep[N], up[N][LOG];
      int n, m, bst[N];
28
29
      Il ans;
30
      vector<int> adj[N] , paths[N];
31
32
      // Lazy segment tree
33
      struct node{
34
       int mn, cnt;
        node(int x){mn = x; cnt = 1;}
35
       node(){};
36
        node operator + (const node other) const{
37
38
          node res;
          res.mn = min(mn, other.mn);
39
          if(mn < other.mn) res.cnt = cnt;</pre>
40
          else if(other.mn < mn) res.cnt = other.cnt;</pre>
41
42
          else res.cnt = cnt + other.cnt:
43
          return res;
       }
44
45
     } tree[4*N];
46
      int lazy[4*N];
47
```

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

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

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

```
176
         }
177
178
       }
179
       // Compute LCA
180
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
         for(int i = LOG - 1; i >= 0; --i) if(up[x][i]!=up[y][i]){
187
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){
         for(auto &v : adj[u]) if(v != p){
197
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);
         freopen("out.txt", "w", stdout);
206
         freopen("error.txt", "w", stderr);
207
         #endif
208
         cin >> n >> m;
209
         for(int i = 1; i < n; ++i){
210
211
           int u,v; cin >> u >> v;
           --U; --V;
212
213
           adj[u].emplace_back(v);
           adj[v].emplace_back(u);
214
         }
215
216
217
         preDFS(0,0);
```

```
decomposition(0,0);
218
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]);
           bst[y] = min(bst[y], dep[lca]);
229
           if(lca == x) continue;
230
231
           paths[x].emplace_back(y);
```

String

Aho corasick

AhoCorasick.cpp

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

Serso Library

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

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

Basic strings Algo

KMP.cpp

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

```
// 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();
        vector<int> z(n);
 6
        int left = 0, right = 0;
 7
 8
        for(int i = 1; i < n; i++) {
 9
          if(i <= right) {</pre>
10
             z[i] = min(right - i + 1, z[i - left]);
11
12
          while(i + z[i] < n && s[z[i]] == s[z[i] + i]) {
             z[i]++;
13
          }
14
          if(i + z[i] - 1 > right) {
15
16
             left = i;
17
             right = i + z[i] - 1;
          }
18
19
        }
20
        return z;
21
```

failer_fun.cpp

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

menacher.cpp

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

```
3
        string t;
        for(auto &val : s) {
 4
 5
          † += '#';
          t += val;
 6
 7
        }
        † += '#';
 8
        return t;
 9
10
      }
11
      // Main Manacher's algorithm implementation
12
13
      vector<int> build(string &s) {
        string t = transform(s);
14
        int n = t.size();
15
        vector<int>p(n, O);
16
        int l = 0, r = 0;
17
18
        for(int i = 0; i < n; i++) {
19
20
          p[i] = (i < r) ? min(r - i, p[l + r - i]) : 1;
          while(i - p[i] >= 0 && i + p[i] < n && t[i + p[i]] == t[i - p[i]]) {
21
            p[i]++;
22
          }
23
24
          if(i + p[i] > r) {
25
            r = i + p[i];
            I = i - p[i];
26
27
          }
        }
28
29
        return p;
30
      }
31
      // Check if substring [I,r] is palindrome using p array
32
      bool IsPalindrome(int I, int r, vector<int>& p) {
33
34
        int center = (l + r) / 2;
35
        bool odd = (1 % 2 == r % 2);
        int newCenter = 2 * center + !odd + 1;
36
        return (r - l + 1) \le (p[newCenter] - 1);
37
      }
38
39
40
      // Returns {start_index, end_index} of longest palindrome substring using p array
      pair<int,int> LongestPalindromeSubStr(vector<int>& p) {
41
42
        int maxLength = 0;
       int start = 0, end = 0;
43
        int n = p.size() / 2; // Original string length
44
45
46
        for(int i = 0; i < n; i++) {
```

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

Hashing

Double_Hash_as_int.cpp

```
const int N = 2e5 + 9;
      const int mod[] = {(int)le9 + 7, 998244353};
 2
      int o; // Which mod wher I current use.
 3
      mt19937 random_seed(time(0));
 4
      int rnd(int I, int r){
 5
        uniform_int_distribution<int> dist(l, r);
 6
        return dist(random_seed);
 7
 8
      }
      \parallel bpw(\parallel a, \parallel b) \parallel
 9
        a \% = mod[o];
10
        II res = 1;
11
        while(b){
12
          if(b \% 2) res = (res * a) \% mod[o];
13
          a = (a * a) % mod[o];
14
          b >>= 1;
15
16
        }
17
        return res;
```

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

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

custom_unorderd_map.cpp

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

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

double_hashing.cpp

```
#include <bits/stdc++.h>
 1
 2
      using namespace std;
      typedef long long II;
 3
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
 4
      #define f first
 5
      #define s second
 6
      const int N = 1e6 + 9;
 7
      const int mod[] = {(int)le9 + 7, 998244353};
 8
 9
      II p[2], pw[2][N];
      mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
10
      int gen(int I, int r) { return rng() \% (r - I + 1) + I; }
11
      II get(II x, II y) \{ return (x + 1) * 1e9 + y; \} // return Double hashing = get(hash[0], hash[1]) \}
12
      void init(){
13
        for(int o = 0; o < 2; o++){
14
          p[o] = gen(31, 39); // Generate Base randomly
15
16
          pw[o][0] = 1;
          for(int i = 1; i < N; ++i)
17
            pw[o][i] = pw[o][i - 1] * p[o] % mod[o];
18
19
        }
20
      }
      vector<II> gen_prefix(string &s){
21
22
        int n = (int) s.size();
23
        vector<II> ans(n);
        Il pre[2][n];
24
        pre[0][0] = pre[1][0] = (s[0] - 'a' + 1);
25
        for(int i = 1; i < n; ++i) {
26
27
          for(int o = 0; o < 2; o++){
```

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

hashing_grid.cpp

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

k-hash.cpp

```
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++) {
         ans[k] = 0;
5
         pw[k] = 1;
6
       }
7
       for (auto i:s) {
8
         for (int k = 0; k < K; k++) {
9
           ans[k] += 111 * pw[k] * (i - '0') % mods[k];
10
11
           ans[k] \%= mods[k];
           pw[k] = (III) pw[k] * b % mods[k];
12
         }
13
14
       }
15
       return ans;
     }
16
     // probability of collision = 1/10^{9 * 6} = 1/10^{54}
17
```

string_hash.cpp

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

```
19
      struct stringHashing {
20
21
        vector<II> preHash1, preHash2, sufHash1, sufHash2;
22
        void init(string &s) {
23
          preHash1 = hash(s, preHash1, pow1, mod1);
24
          preHash2 = hash(s, preHash2, pow2, mod2);
25
26
          sufHash1 = hash(s, sufHash1, pow1, mod1, true);
          sufHash2 = hash(s, sufHash2, pow2, mod2, true);
27
        }
28
29
        vector<II> hash(string &s, vector<II> &h, vector<II> &p, int m, bool rev = false) {
30
          int n = s.size();
31
          h.resize(n);
32
          int st = 0, en = n, delta = 1;
33
          if (rev)
34
            st = n - 1, en = -1, delta = -1;
35
36
          int i = 0;
          while (st != en) {
37
            h[st] = (s[st] - 'a' + 1) * p[i] % m;
38
            if (i!= 0) {
39
              h[st] += h[st - delta];
40
              if(h[st] >= m)
41
                h[st] = m;
42
43
            }
44
            st += delta;
            j++;
45
46
          }
47
          return h;
        }
48
49
        Il query(int I, int r) {
50
51
          IIh1 = preHash1[r], h2 = preHash2[r];
          h1 = (I = 0 ? 0 : preHash1[I - 1]);
52
          h2 = (I = 0?0: preHash2[I - 1]);
53
          if (h1 < 0)h1 += mod1;
54
          if (h2 < 0)h2 += mod2;
55
56
          h1 = h1 * pow1[N - I] % mod1;
          h2 = h2 * pow2[N - I] % mod2;
57
          return h1 * h2;
58
        }
59
60
61
        Il revQuery(int I, int r) {
62
          int n = sufHash1.size();
```

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

Lyndon

Duval.cpp

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

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

min_cyclic_string.cpp

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

Serso Library

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

suffixArray

Kasai.cpp

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

```
23 return lcp;
24 }
```

RMQ_suffixarray.cpp

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

```
if(pos1 \ge n) pos1 = n;
38
39
             if(pos2 \ge n) pos2 = n;
40
41
             pair < int, int > cur = {c[p[i]], c[pos1]};
             pair < int, int > prev = {c[p[i-1]], c[pos2]};
42
             if(cur != prev) ++classes;
43
             cn[p[i]] = classes - 1;
44
          }
45
          c.swap(cn);
46
          ans.emplace_back(c);
47
48
        }
        p.erase(p.begin());
49
       return ans;
50
51
      }
52
      // Get LCP of suffixes starting at i,j
53
      // Time: O(logn)
54
55
      int LCP(int i, int j, vector<vector<int>> &c) {
       int ans = 0, n = c[0].size(), lg_n = c.size();
56
        for(int k = lg_n - 1; k \ge 0; k--) {
57
          if(c[k][i\% n] == c[k][j\% n]) {
58
59
            ans += 1 << k;
            i += 1 << k;
60
            j += 1 << k;
61
          }
62
63
        }
64
       return ans;
65
      }
66
      // Compare same length suffixes starting at i,j
67
      // Returns: -1 if i<j, 0 if equal, 1 if i>j
68
      // Time: O(1)
69
70
      int compare(int i, int j, int l, vector<vector<int>> &c) {
       int n = c[0].size();
71
       int k = 31 - \underline{\text{builtin\_clz}(l)};
72
        pair<int,int> a = {c[k][i], c[k][(i+l-(1<<k))%n]};
73
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
      // Compare substrings s[l1..r1] vs s[l2..r2]
78
      // Returns: -1 if s1<s2, 0 if equal, 1 if s1>s2
79
80
      // Time: O(logn)
      int compare(int | 1, int | 1, int | 2, int | 2, string &s, vector<vector<int>> &c) {
81
```

```
int lcp = LCP(l1, l2, c);
int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1;
lcp = min({lcp, len1, len2});
if(lcp == len1 && lcp == len2) return 0;
if(lcp == len1) return -1;
if(lcp == len2) return 1;
return s[l1 + lcp] < s[l2 + lcp] ? -1:1;</pre>
```

build_suffixArray_slow_O(nlognlogn).cpp

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

build_suffixarray_veryfast.cpp

```
vector<int> buildSuffixArray(string s) {
    int n = s.size(); s += "$";
    vector<int> p(n+1), c(n+1), c1(n+1), cnt(max(256, n+1)), p1(n+1);
    for(int i = 0; i <= n; i++) p[i] = i, c[i] = s[i];
}</pre>
```

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

problems

count_pattern.cpp

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

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

distinct_substring.cpp

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

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

kth_distinct_substring.cpp

```
1
     Problem link: https://cses.fi/problemset/task/2108/
 2
      You are given a string of length n.
     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);
        string s; cin >> s;
 8
        int n = s.size();
 9
        auto p = buildSuffixArray(s);
10
        auto lcp = Kasai(s,p);
11
        II k; cin >> k;
12
13
        if(k <= n - p[0]){ cout << s.substr(p[0], k); return 0; }
        k = n - p[0];
14
        for(int i = 0; i < lcp.size(); ++i){
15
16
          int len = n - p[i+1] - lcp[i];
          if(k <= len){ cout << s.substr(p[i+1],k+lcp[i]); return 0; }</pre>
17
          k -= len;
18
        }
19
      }
20
```

kth_smallest_string.cpp

```
1 /*
    Problem link: https://cses.fi/problemset/task/2109/
2
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
    string s; cin >> s;
6
      int n = s.size();
7
      auto p = buildSuffixArray(s);
8
```

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

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.
```

```
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;
      int n = s.size();
 8
      auto p = buildSuffixArray(s);
 9
      auto lcp = Kasai(s,p);
10
11
12
      Il cnt[n+2]{};
13
14
      auto upd = [\&](int I, int r, int v){
15
      if(l > r)return;
        cnt[l] += v; cnt[r+1] -= v;
16
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];
      for(int i = 1; i \le n; ++i) cnt[i] += n - i + 1;
22
23
      for(int i = 1; i <= n; ++i) cout << cnt[i] << ' ';</pre>
```

no_of_substring_at_least_f.cpp

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

```
19
           // Optimize first phase with counting sort
20
21
           vector<int> cnt(ALPHA);
22
           for(auto ch : s) cnt[ch]++;
           for(int i = 1; i < ALPHA; i++) cnt[i] += cnt[i-1];</pre>
23
           for(int i = n-1; i \ge 0; i--) p[--cnt[s[i]]] = i;
24
25
26
           c[p[0]] = 0;
           int classes = 1;
27
           for(int i = 1; i < n; i++) {
28
29
             if(s[p[i]] != s[p[i-1]]) classes++;
             c[p[i]] = classes - 1;
30
          }
31
        }
32
33
34
        vector<int> pn(n), cn(n);
        for(int h = 0; (1 << h) < h; ++h) {
35
36
           int len = 1 << h;
          for(int i = 0; i < n; i++) {
37
             pn[i] = p[i] - len;
38
             if(pn[i] < 0) pn[i] += n;
39
           }
40
41
           // Optimize counting sort for each phase
42
43
           vector<int> cnt(n);
           for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
44
           for(int i = 1; i < n; i++) cnt[i] += cnt[i-1];
45
           for(int i = n-1; i \ge 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
46
47
           cn[p[0]] = 0;
48
           int classes = 1;
49
           for(int i = 1; i < n; i++) {
50
51
             pair < int, int > cur = {c[p[i]], c[(p[i] + len) % n]};
             pair < int, int > prev = {c[p[i-1]], c[(p[i-1] + len) % n]};
52
53
             if(cur != prev) ++classes;
             cn[p[i]] = classes - 1;
54
          }
55
           c.swap(cn);
56
        }
57
        p.erase(p.begin());
58
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);
         for(int i = 0; i < n; i++) rank[p[i]] = i;
65
66
         for(int i = 0, k = 0; i < n; i++) {
 67
           if(rank[i] == n-1) {
68
69
             k = 0;
             continue;
70
 71
           }
 72
           int j = p[rank[i] + 1];
           while(i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
 73
74
           lcp[rank[i]] = k;
           if(k) k--;
 75
         }
 76
         return lcp;
 77
       }
 78
 79
80
       // Optimized stack-based nearest smaller element
       pair<vector<int>, vector<int>> getMinBounds(const vector<int>& arr) {
 81
         int n = arr.size();
 82
         vector<int> left(n), right(n);
 83
         vector<int> st;
84
         st.reserve(n):
 85
86
         // Get left bounds
 87
         for(int i = 0; i < n; i++) {
88
           while(!st.empty() && arr[st.back()] > arr[i]) st.pop_back();
89
90
           left[i] = st.empty() ? -1: st.back();
           st.push_back(i);
 91
         }
92
93
         // Clear stack for right bounds
94
         st.clear();
95
96
 97
         // Get right bounds
         for(int i = n-1; i \ge 0; i--) {
98
           while(!st.empty() && arr[st.back()] >= arr[i]) st.pop_back();
99
           right[i] = st.empty()?n:st.back();
100
           st.push_back(i);
101
         }
102
103
         return {left, right};
104
       }
105
```

```
106
       void solve() {
107
          string s;
108
          cin >> s;
109
110
          int n = s.size();
 111
          auto p = buildSuffixArray(s);
112
          auto lcp = buildLCP(s, p);
113
          auto [L, R] = getMinBounds(Icp);
114
115
          // Pre-allocate vectors to avoid resizing
116
 117
          vector<ll> frq(n+9);
          vector<vector<int>> group(n+1);
118
          for(int i = 0; i < n+1; i++) group[i].reserve(n/2);
119
120
          // Group LCP values
121
          for(int i = 0; i < (int)|cp.size(); ++i) {
122
            group[lcp[i]].push_back(i);
123
124
         }
125
          // Calculate frequencies
126
127
          for(int len = n; len > 0; --len) {
            for(size_t j = 0; j < group[len].size();) {
128
              int x = group[len][j];
129
              int I = L[x], r = R[x];
130
              int f = r - l;
131
              int minh = len;
132
133
              if(l \ge 0) minh = min(minh, len - lcp[l]);
134
135
              if(r < (int)lcp.size()) minh = min(minh, len - lcp[r]);
136
              frq[f] += 1LL * f * minh;
137
138
139
              // Skip processed indices
              while(j < group[len].size() && group[len][j] <= r) ++j;</pre>
140
           }
141
          }
142
143
          // Calculate cumulative frequencies
144
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;
         cin >> q;
150
151
         while(q--) {
           int x;
152
153
           cin >> x;
           cout << (x > s.size() ? 0 : frq[x]) << '\n';
154
         }
155
       }
156
157
       int main() {
158
         ios::sync_with_stdio(false);
159
         cin.tie(nullptr);
160
161
162
         int tc;
         cin >> tc;
163
         while(tc--) solve();
164
```

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

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

trie.cpp

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

Trees

CentroidDecomposition.cpp

```
const int N = le5 + 9;
vector<int> adj[N];
struct CentroidDecomposition{
  vector<int> removal , sz;
  CentroidDecomposition(int n){
  removal.assign(n , 0);
  sz.assign(n , 0);
  build(0, -1);
```

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

HLD.cpp

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

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

Serso Library

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

HLD_edges.cpp

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

```
if(~heavy[u])
29
          decomposition(heavy[u], h);
30
31
        for(auto \&[v, w]: adj[u]) if(v != par[u] \&\& v != heavy[u])
          decomposition(v, v);
32
      }
33
34
35
      void buildWeightArray() {
36
        values.resize(n);
        for(int u = 0; u < n; u++) {
37
          for(auto [v, w] : adj[u]) {
38
39
            if(dep[u] < dep[v]) {
              values[pos[v]] = w;
40
            }
41
          }
42
43
        }
44
      }
45
46
      int queryPath(int u, int v) {
        int ans = -1e9;
47
        for(; head[v]!= head[v]; v = par[head[v]]) {
48
          if(dep[head[v]]) > dep[head[v]]) swap(u,v);
49
          // process range [pos[head[v]], pos[v]]
50
        }
51
        if(dep[u] > dep[v]) swap(u,v);
52
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]
        return 0; // replace with actual query
62
      }
63
64
65
      int main() {
        ios::sync_with_stdio(0); cin.tie(0);
66
        int m, q; cin >> n >> m >> q;
67
68
        for(int i = 0; i < m; i++) {
69
70
          int u, v, w; cin >> u >> v >> w;
          --u; --v;
71
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:
          1- Starting BFS from any node.
 4
          2- Find the farthest node (Start) from it.
 5
          3- Starting BFS from (Start).
 6
 7
          5- Find the farthest node (End) from (Start)
          6- The path from (Start) and (End) is one possible diameter for the tree.
 8
      */
 9
      int n; cin >> n;
10
        for(int i = 0; i < n - 1; ++i){
11
          int u , v; cin >> u >> v;
12
          --U; --V;
13
          adj[u].emplace_back(v);
14
          adj[v].emplace_back(u);
        }
16
        auto bfs = [\&](int st){}
17
          vector<int> dis(n, -1);
18
          queue<int> q;
19
          q.push(st); dis[st] = 0;
20
           while(q.size()){
21
              int u = q.front(); q.pop();
22
             for(auto &v : adj[u]) if(dis[v] == -1){
23
                dis[v] = dis[u] + 1; pre[v] = u;
24
                q.push(v);
25
             }
26
27
           }
            return max_element(dis.begin(), dis.end()) - dis.begin();
28
29
        };
        int Start = bfs(0) , End = bfs(Start);
30
        vector<int> v;
31
        for(int i = End;; i = pre[i]){}
32
33
          v.emplace_back(i);
```

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

Tree_flatten.cpp

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

bridgestree.cpp

```
#include <bits/stdc++.h>
 2
      using namespace std;
 3
      typedef long long II;
      #define rep(i, st, ed) for(int i = st; i < ed; i++)
 4
 5
      #define f first
      #define s second
      const int N = 3e5 + 9:
 7
      vector<pair<int,int>> adj[N] , edges;
 8
      vector<int> BridgeTree[N];
 9
10
      int lowLink[N], dfn[N], comp[N], ndfn, comp_num;
      bool isBridge[N] , vis[N];
11
      void tarjan(int u , int par){
12
       dfn[u] = lowLink[u] = ndfn++;
13
14
       for(auto &[v , id] : adj[u]){
        if(dfn[v] == -1){
15
        tarjan(v, u);
         lowLink[u] = min(lowLink[u], lowLink[v]);
17
          if(lowLink[v] == dfn[v]){
18
            int uu = u, vv = v;
19
20
            if(uu > vv) swap(uu , vv);
21
            isBridge[id] = true;
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
       vis[u] = true;
29
        comp[u] = comp_num;
30
       for(auto &[v, id]: adj[u]) if(vis[v] == 0 && isBridge[id] == 0)
31
          Find_component(v, u);
32
33
34
     pair<int, int> diameter(int u, int par = -1)
35
        int diam = 0;
36
37
        int mxHeights[3] = {-1, -1, -1}; // keep 2 highest trees
        for(auto &v : BridgeTree[u]) if(v != par)
38
       {
39
40
          auto p = diameter(v, u);
          diam = max(diam, p.f);
41
          mxHeights[0] = p.s+1;
42
          sort(mxHeights, mxHeights+3);
43
44
       }
        for(int i = 0; i < 3; i++)if(mxHeights[i] == -1)
45
          mxHeights[i] = 0;
46
        diam = max(diam, mxHeights[1] + mxHeights[2]);
47
        return {diam, mxHeights[2]};
48
     }
49
     int main(){
50
        ios::sync_with_stdio(0); cin.tie(NULL); cout.tie(0);
51
        #ifndef ONLINE_JUDGE
52
        freopen("in.txt", "r", stdin);
53
        freopen("out.txt", "w", stdout);
54
55
        freopen("error.txt", "w", stderr);
        #endif
56
        int n, m; cin >> n >> m;
57
        for(int i = 0; i < m; i++){
58
59
          int u , v; cin >> u >> v;
          --u; --v;
60
          adj[u].emplace_back(v, i);
61
62
          adj[v].emplace_back(u , i);
          edges.emplace_back(u, v);
63
       }
64
65
        // Finding Bridges using Tarjan algo.
        for(int i = 0; i < n; i++){ dfn[i] = -1; lowLink[i] = 0; }
66
```

```
ndfn = 0;
67
68
       tarjan(0, 0);
       // dfs to group all the maximal components together, so that we can shrink it to one
69
     node
       for(int i = 0; i < n; i++) if(vis[i] == 0){
70
71
         Find_component(i, i);
72
         comp_num++;
73
       }
74
       // shrinking all the maximal components to one node
       for(int i = 0; i < m; i++) {
75
         if(isBridge[i]) {
76
77
           BridgeTree[comp[edges[i].f]].emplace_back(comp[edges[i].s]);
           BridgeTree[comp[edges[i].s]].emplace_back(comp[edges[i].f]);
78
         }
79
       }
08
       // Finding the diameter of the Bridgestree
81
82
       int d = diameter(0, 0).f;
       cout << d;
83
```

centroid.cpp

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

```
21 in main
22 dfs(0,-1)
23 */
```

isophrisim.cpp

```
#include <bits/stdc++.h>
     typedef long long II;
 2
 3
     #define s second
     #define f first
 4
     #define rep(i, st, ed) for(int i = st; i < ed; i++)
 5
     using namespace std;
 6
 7
     const int N = 4000;
     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
14
      #endif
15
     }
     16
17
     int n;
     vector<int> adj[N];
18
     string get_subcan(int u , vector<vector<string>> &subcan){
      sort(subcan[u].begin() , subcan[u].end());
20
      string ans = "(";
21
22
      for(auto &s : subcan[u]) ans += s;
      ans += ")";
23
24
      return ans;
25
     void tree_isophrisim(){
26
      vector<int> deg(n);
27
      queue<int> leaf;
28
29
      vector<vector<string>> subcan(n);
      rep(i, 0, n){
30
       deg[i] = adj[i].size();
31
       if(deg[i] <= 1) leaf.push(i);</pre>
32
33
34
      int rem = n;
      while(rem > 2){}
35
36
       int sz = leaf.size():
```

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

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

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

tree_diameter_dfs.cpp

```
pair<int, int> diameter(int u, int par = -1)

int diam = 0;
int mxHeights[3] = {-1, -1, -1}; // keep 2 highest trees
```

Serso Library

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