COMPETITIVE PROGRAMMING NOTEBOOK

GEMA ICMC

March 30, 2022

Contents

1 DP **4**
- 1.1 CHT . . . 4

2 Geometry **4**
- 2.1 HalfPlanesIntersection . . . 4
- 2.2 PointUtils . . . 6
- 2.3 Pick . . . 7
- 2.4 Point . . . 7
- 2.5 ConvexHull . . . 8

3 Strings **9**
- 3.1 AhoCorasick . . . 9
- 3.2 SuffixTree . . . 10
- 3.3 KMPAutomata . . . 12
- 3.4 SuffixArray . . . 12
- 3.5 SuffixAutomaton . . . 13
- 3.6 Hash . . . 14
- 3.7 KMP . . . 16
- 3.8 PalindromicTree . . . 16
- 3.9 Z . . . 17

4 DataStructures **18**
- 4.1 SegTreeLazy . . . 18
- 4.2 CartesianTree . . . 19
- 4.3 Paretto . . . 20
- 4.4 SparseTable . . . 20
- 4.5 TreapPersistent . . . 21
- 4.6 RandomizedHeap . . . 22
- 4.7 SegTree . . . 22
- 4.8 MergeSortTree . . . 24
- 4.9 Treap . . . 24
- 4.10 DSUPartial . . . 26
- 4.11 SegTreeBeats . . . 27
- 4.12 BIT2D . . . 29
- 4.13 DSURollback . . . 29
- 4.14 BIT . . . 30
- 4.15 SegTreeIteractive . . . 31
- 4.16 WaveletTree . . . 31
- 4.17 TwoPointers . . . 33
- 4.18 WaveletTreeToggle . . . 34
- 4.19 MinQueue . . . 36

5 Tree **37**
- 5.1 LCA . . . 37
- 5.2 Centroid . . . 37
- 5.3 Isomorphism . . . 39

6 Misc **40**
- 6.1 CoordinateCompressor . . . 40
- 6.2 OrderedSet . . . 41
- 6.3 Pragma . . . 41
- 6.4 Date . . . 41
- 6.5 BufferedWrite . . . 42

7 Graphs **43**
- 7.1 Kruskal . . . 43
- 7.2 EulerCycle . . . 44
- 7.3 DynamicConnectivity . . . 45
- 7.4 Kosaraju . . . 47
- 7.5 BlockCut . . . 48
- 7.6 DominatorTree . . . 49
- 7.7 TarjanSCC . . . 51

7.8 BellmanFord . . . 52
7.9 Hungarian . . . 52
7.10 EdmondsMDST . . . 53
7.11 MCMF . . . 54
7.12 TarjanBridges . . . 56
7.13 TwoSat . . . 57
7.14 Kuhn . . . 57
7.15 NegativeCycles . . . 58
7.16 DirectedMST . . . 59
7.17 Dinic . . . 60
7.18 CycleSimulation . . . 61
7.18.1 simple . . . 61
7.18.2 tortoise$_h$are . . . 62
7.18.3 full . . . 63

8 Solutions **64**
8.1 Geometry . . . 64
8.1.1 NearestTwoPoints . . . 64
8.2 Strings . . . 65
8.2.1 Hash . . . 65
8.3 Graph . . . 68
8.3.1 DagWidth . . . 68
8.4 SegmentTree . . . 69
8.4.1 sorting . . . 69
8.5 Greedy . . . 71
8.5.1 StableMarriage . . . 71
8.6 Math . . . 72
8.6.1 AndConvolution . . . 72
8.6.2 GrayCode . . . 73

9 Setup **73**
9.1 template . . . 73

10 Math **74**
10.1 GaussianElimination . . . 74
10.2 LinearRec . . . 76
10.3 EulerPhi . . . 77
10.4 FixedMatrix . . . 78
10.5 LinearSieve . . . 80
10.6 BerlekampMassey . . . 80
10.7 Montgomery64 . . . 81
10.8 Karatsuba . . . 82
10.9 Stirling . . . 84
10.10FWHT . . . 85
10.11NTT . . . 86
10.12MillerRabin . . . 88
10.13Sieve . . . 89
10.14PrimitiveRoot . . . 89
10.15FFT . . . 90
10.16Matrix . . . 91
10.17LagrangeInterpolation . . . 94
10.18PointsUnderLine . . . 95

1 DP

1.1 CHT

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

// Dynamic Hull. Usa s  inteiros, mas se der overflow, tem comentarios pra transformar pra double
namespace CHT {
  using ll = long long;
  const ll isQuery=-(1LL<<62);
  struct Line {
      ll m, b;int id;
      Line(ll m_, ll b_, int id_) : m(m_), b(b_), id(id_) {}
      mutable multiset<Line>::iterator it,e;
      const Line* succ() const {
          return next(it)==e ? 0 : &*next(it);
      }
      bool operator<(const Line& rhs) const {
          if (rhs.b!=isQuery) return m<rhs.m;
          const Line* s=succ();
          if (!s) return 0;
          ll x = rhs.m;
          return b - s->b < (s->m - m)*x;
          // se der overflow, substitua a linha de cima por
          // return b - s->b < double(s->m - m) * double(x);
      }
  };
  struct DynamicHull : public multiset<Line> {
      bool bad(iterator y) {
          auto z=next(y);
          if (y==begin()) {
              if (z==end()) return 0;
              return y->m == z->m && y->b <= z->b;
          }
          auto x=prev(y);
          if (z==end()) return y->m==x->m&&y->b<=x->b;
          return (x->b - y->b)*(z->m - y->m)>=(y->b - z->b)*(y->m - x->m);
          // se der overflow, substitua a linha de cima por
          // return double(x->b - y->b)*double(z->m - y->m) >= double(y->b - z->b)*double(y->m - x->
    m);
      }

      // O(logN), N = numero de linhas
      void insertLine(ll m, ll b, int id) {
          auto y = insert({m, b, id});
          y->it=y; y->e=end();
          if (bad(y)) {erase(y);return;}
          while (next(y)!=end() && bad(next(y))) erase(next(y));
          while (y!=begin() && bad(prev(y))) erase(prev(y));
      }

      // O(logN), N = numero de linhas
      pair<ll, int> getMax(ll x) {
          auto l=*lower_bound({x, isQuery, 0});
          return {l.m*x+l.b, l.id};
      }
  };
}
```

2 Geometry

2.1 HalfPlanesIntersection

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

using namespace std;

const double eps = 1e-9;
struct pt {
  double x, y;
  pt(){}
  pt(double _x, double _y): x(_x), y(_y){}

  pt operator-(pt q) const {
    return pt(x - q.x, y - q.y);
  }
  pt operator+(pt q) const {
```

```
    return pt(x + q.x, y + q.y);
  }
  double operator^(pt q) const {
    return x * q.y - q.x * y;
  }
};

struct line {
  double a, b, c;

  line(){}
  line(double _a, double _b, double _c): a(_a), b(_b), c(_c) {}
  line(pt p, pt q) {
    pt aux = q - p;
    a = -aux.y;
    b = aux.x;
    c = -a * p.x - b * p.y;
  }

  bool contains(pt p) {
    return a * p.x + b * p.y + c >= 0;
  }

  pt norm() {
    return pt(a, b);
  }

  pt any() {
    if(abs(b) < eps) return pt(-c/a, 0);
    return {0, -c/b};
  }

  static pt intersection(line l1, line l2, int & err) {
    const double eps = 1e-9;
    double a1, b1, c1;
    a1 = l1.a, b1 = l1.b, c1 = -l1.c;
    double a2, b2, c2;
    a2 = l2.a, b2 = l2.b, c2 = -l2.c;

    err= 0;

    double den = l1.norm()^l2.norm();

    if(abs(den) < eps) {
      if(abs(c1 - c2) < eps) err = 1; // s o a mesma reta
      err = 2; // nao h  intersec o
      return pt(0, 0);
    }

    double x = (c1 * b2 - c2 * b1)/den;
    double y = (a1 * c2 - a2 * c1)/den;
    return pt(x, y);
  }
};

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

/* Retorna se esse vector<line> vec, que contem halfplanes,   vazio ou n o */
bool intersection_not_empty(vector<line> vec) {
  // vec.emplace_back(1, 0, 1e9); // parede esquerda
  // vec.emplace_back(-1, 0, 1e9); // parede direita
  // vec.emplace_back(0, 1, 1e9); // parede inferior
  // vec.emplace_back(0, -1, 1e9); // parede superior

  pt cur = {0, 0}; // qualquer ponto dentro da interseccao dos semi planos
  shuffle(vec.begin(), vec.end(), rng);

  for(int i = 0; i < vec.size(); ++i) {
    line & l = vec[i];
    if(l.contains(cur)) continue;
    // acha o novo p, vai ser alguma interseccao de l com todos os outros ate agora

    pt hi, lo;
    int flaghi = 0, flaglo = 0;
    for(int j = 0; j < i; ++j) {
      int err;
      pt p = line::intersection(l, vec[j], err);
      if(err == 1) continue; // sao a mesma linha, nao acontece nada
      if(err == 2) {
```

```
        // significa que os semi planos paralelos s o contr rios
        if(!vec[j].contains(l.any()) and !l.contains(vec[j].any())) return false;
      } else {
        // dois casos
        if((l.norm()^vec[j].norm()) > 0) {
          // pra tras
          if(!flaghi or ((hi - p)^l.norm()) > 0) {
            hi = p;
            flaghi = 1;
          }
        } else {
          // pra frente
          if(!flaglo or ((p - lo)^l.norm()) > 0) {
            lo = p;
            flaglo = 1;
          }
        }
      }
    }

    if(flaghi + flaglo < 2) cur = flaghi? hi : flaglo? lo : l.any();
    else if(((hi - lo)^l.norm()) >= 0) cur = hi; // hi ou lo, tanto faz
    else return false; // intersecao vazia
  }

  return true;
}
```

2.2 PointUtils

```
// Uses struct Point declared in Point.cpp

/** Functions:
    - shoelace2 (area of polygon)
    - closest_pair_distance2 (minimum distance between 2 points)
    - segments_intersect (whether 2 segments intersect)
    - in_polygon (whether point is inside polygon)
    - is_clockwise (polyon is in clockwise or counter-clockwise order)
*/


/** returns 2 * area(polygon) */
template<typename T> T shoelace2(vector<Point<T>> const& p) {
  T ans = 0; int n = p.size();
  for (int i = 0, j = n-1; i < n; j = i, i++) ans += p[i].cross(p[j]);
  return abs(ans);
}

/** return the minimum distance**2 between 2 points in the plane */
template<typename T> T closest_pair_distance2(vector<Point<T>> p) {
  sort(p.begin(), p.end());

  vector<Point<T>> strip(p.size());
  auto find_closest = [&](auto&& find_closest, vector<Point<T>>& p, int l, int r)->T {
    if (r - l <= 0) return numeric_limits<T>::max();

    int m = (l + r) / 2;
    T distL = find_closest(find_closest, p, l, m);
    T distR = find_closest(find_closest, p, m + 1, r);
    T dist = min(distL, distR);

    int strip_index = 0;
    for (int i = l, j = m + 1; i <= m || j <= r; ) {
      if (j > r || (i <= m && p[i].y < p[j].y)) strip[strip_index++] = p[i++];
      else strip[strip_index++] = p[j++];
    }
    for (int i = l; i <= r; i++) p[i] = strip[i - l];

    strip_index = 0;
    for (int i = l; i <= r; i++) if ((p[i].x - p[m].x) * (p[i].x - p[m].x) < dist)
      strip[strip_index++] = p[i];

    for (int i = 0; i < strip_index; i++) {
      for (int j = i + 1; j < strip_index && (strip[j].y - strip[i].y) * (strip[j].y - strip[i].y) <
     dist; j++) {
        dist = min(dist, strip[i].dist2(strip[j]));
      }
    }

```

```
    return dist;
  };

  return find_closest(find_closest, p, 0, (int) p.size() - 1);
}

/** Checks whether 1-dimensional segments [a, b] and [c, d] intersect */
template<typename T> bool segments_intersect_1d(T a, T b, T c, T d) {
  if (a > b) swap(a, b);
  if (c > d) swap(c, d);
  return max(a, c) <= min(b, d);
}

/** Checks whether segments AB and CD intersect */
template <typename T> bool segments_intersect(Point<T> const& a, Point<T> const& b, Point<T> const&
    c, Point<T> const& d) {
  if (c.cross(a, d) == 0 && c.cross(b, d) == 0)
    return segments_intersect_1d(a.x, b.x, c.x, d.x) && segments_intersect_1d(a.y, b.y, c.y, d.y);
  return sign(a.cross(b, c)) != sign(a.cross(b, d)) &&
      sign(c.cross(d, a)) != sign(c.cross(d, b));
}

/** Check if point p is inside polygon. Return: 0: outside, 1: inside, 2: boundary.
 * BE CAREFUL: inf has to be greater than any other point, to make sure it isn't collinear with any
    edge */
template<typename T> int in_polygon(Point<T> p, vector<Point<T>> const& v) {
  int n = v.size(), count = 0;

  const T inf = 2e9;
  Point<T> p2{p.x + 1, inf};
  for (int i = 0, j = n-1; i < n; j = i, i++) {
    if (p.in_segment(v[i], v[j])) return 2;
    count += segments_intersect(p, p2, v[i], v[j]);
  }

  return count % 2;
}

/** Returns whether polygon points are in clockwise or counter-clockwise order */
template<typename T> bool is_clockwise(vector<Point<T>> const& v) {
  T area = 0; int n = v.size();
  for (int i = 0, j = n - 1; i < n; j = i, i++) area += v[i].cross(v[j]);
  return area >= 0;
}
```

2.3 Pick

```
/** Count integer (lattice) points inside and on boundary of polygon (can be concave)
 * BE CAREFUL: T must be INTEGER!!! */
template<typename T> pair<T, T> count_lattice_points(vector<Point<T>> const& v) {
  /** Pick's theorem: A = i + b/2 - 1 (area = inner_lattice + boundary_lattice/2 - 1) */
  pair<T, T> ans = {0, 0}; // <inside, boundary>
  T area2 = 0;
  int n = v.size();
  for (int i = 0, j = n-1; i < n; j = i, i++) {
    Point<T> dif = v[i] - v[j];
    dif.x = abs(dif.x), dif.y = abs(dif.y);
    ans.se += gcd(dif.x, dif.y);
    area2 += v[i].cross(v[j]);
  }
  area2 = abs(area2);
  ans.fi = (area2 - ans.se + 2) / 2;
  return ans;
}
```

2.4 Point

```
/** ==== GEOMETRY NOTEBOOK
    ================================================================================================ */
const double EPS = 1e-12;
template<typename T> inline char sign(T x) { return abs(x) < EPS ? 0 : x > 0 ? 1 : -1; }
template<typename T> struct Point {
  T x, y;

  Point() : x(), y() {}
  Point(T x, T y) : x(x), y(y) {}
  template<typename Tp> Point(Point<Tp> p) : x(p.x), y(p.y) {}

  // basic operators
```

```
  template<typename Tp> Point<T> operator+(Point<Tp> const& p) const { return Point<T>(x + p.x, y + p.y); }
  template<typename Tp> Point<T> operator-(Point<Tp> const& p) const { return Point<T>(x - p.x, y - p.y); }
  template<typename Tp> Point<T> operator*(Tp&& p) const { return Point<T>{x * p, y * p}; }
  template<typename Tp> Point<T> operator/(Tp&& p) const { return Point<T>{x / p, y / p}; }
  template<typename Tp> Point<T> operator%(Tp&& p) const { return Point<T>{x % p, y % p}; }

  template<typename Tp> Point<T>& operator+=(Point<Tp> const& p) { return (*this) = (*this) + p; }
  template<typename Tp> Point<T>& operator-=(Point<Tp> const& p) { return (*this) = (*this) - p; }
  template<typename Tp> Point<T>& operator*=(Tp&& p) { return (*this) = (*this) * p; }
  template<typename Tp> Point<T>& operator/=(Tp&& p) { return (*this) = (*this) / p; }
  template<typename Tp> Point<T>& operator%=(Tp&& p) { return (*this) = (*this) % p; }

  template<typename Tp> bool operator<(Point<Tp> const& p) const { return x == p.x ? y < p.y : x < p.x; }
  template<typename Tp> bool operator==(Point<Tp> const& p) const { return x == p.x && y == p.y; }

  inline T len2() const { return x * x + y * y; }
  inline double len() const { return hypot(x, y); }
  template<typename Tp> inline T dist2(Point<Tp> const& p) const { return (*this - p).len2();  }
  template<typename Tp> inline double dist(Point<Tp> const& p) const { return hypot(x - p.x, y - p.y); }

  template<typename Tp> inline T dot(Point<Tp> const& p) const { return x * p.x + y * p.y; }    // |u| * |v| * cos(alpha)
  template<typename Tp> inline T cross(Point<Tp> const& p) const { return x * p.y - y * p.x; }  // |u| * |v| * sin(alpha)
  template<typename Tp1, typename Tp2> inline T dot(Point<Tp1> const& a, Point<Tp2> const& b) const { return (a - *this).dot(b - *this); }
  template<typename Tp1, typename Tp2> inline T cross(Point<Tp1> const& a, Point<Tp2> const& b) const { return (a - *this).cross(b - *this); }

  /** Orientation of (*this) according to segment AB. 0: collinear, 1: right, -1: left */
  inline char orientation(Point const& a, Point const& b) const { return sign((*this - b).cross(b - a)); }

  /** Orthogonal projection of vector (*this) on vector u.
   * The point of projection of AB on AC will be at A + AC * AB.proj(AC). */
  inline double proj(Point const& u) const { return static_cast<double>(u.dot(*this)) / u.len2(); }

  /** Distance from (*this) to segment AB */
  inline double dist_to_segment(Point const& a, Point const& b) const {
    Point<T> p = *this;
    if (a.dist2(b) <= EPS) return p.dist(a);
    Point<double> ap = p - a, ab = b - a;
    // if projection doesnt lie on segment, the minimum distance will be to A or B
    double u = clamp(ap.proj(ab), 0., 1.);
    Point<double> c = ab * u + a;
    return p.dist(c);
  }

  /** Checks whether point (*this) is in segment AB */
  inline bool in_segment(Point const& a, Point const& b) const {
    Point<T> AB = b - a, AP = (*this) - a;
    return sign(AB.cross(AP)) == 0 && AB.dot(AP) >= 0 && AB.dot(AP) <= AB.len2();
  }
};

template<typename T> istream& operator>>(istream& in, Point<T>& p) { return in >> p.x >> p.y; }
template<typename T> ostream& operator<<(ostream& out, Point<T>& p) { return out << p.x << ' ' << p.y; }
using pt = Point<ll>;
/** ==== GEOMETRY NOTEBOOK
    ================================================================================================ */
```

2.5 ConvexHull

```
/** Convex hull excluding collinear points. O(n)
 * To include collinear points, change the <= operators of orientation to < */
template<typename T> vector<Point<T>> convex_hull(vector<Point<T>> v) {
  int n = v.size();
  if (n <= 2) return v;

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

  vector<Point<T>> ch(2 * n);
  int sz = 0;

```

```
  for (int i = 0; i < n; i++) {
    while(sz > 1 && v[i].orientation(ch[sz - 2], ch[sz - 1]) < 0) sz--;
    ch[sz++] = v[i];
  }
  for (int i = n-2, up_sz = sz; i >= 0; i--) {
    while(sz > up_sz && v[i].orientation(ch[sz - 2], ch[sz - 1]) < 0) sz--;
    ch[sz++] = v[i];
  }

  ch.resize(sz - 1);
  return ch;
}
```

3 Strings

3.1 AhoCorasick

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

using namespace std;

/*
  Complexidade: O(mk), one k    o tamanho do alfabeto e m eh o numero de strings. Mudar de map pra
    array caso k for muito grande

  Aplica  es , de acordo com o cp-algorithms:

  - Find all strings from a given set in a text (acha o numero de ocorrencias de cada string)
  - Finding the lexicographical smallest string of a given length that doesn't match any given
    strings
  - Finding the shortest string containing all given strings (solucao 2^n * n esperada)

  - Como AhoCorasick automata (por exemplo, dada uma sequencia de caracteres, diz se alguma string
    do set ja foi vista)
*/

struct AhoCorasick {

  struct node {
    map<char, int> ch;
    int par, suf, end;
  };

  vector<node> aho;

  int new_node() {
    aho.emplace_back();
    return (int) aho.size() - 1;
  }

  AhoCorasick() {
    aho.resize(1);
  }

  // Adiciona uma string no aho corasick
  void add(string & s) {
    int cur = 0;

    for(auto c : s) {
      if(aho[cur].ch.count(c) == 0) {
        int x = new_node();
        aho[cur].ch[c] = x;
        aho[x].par = cur;
      }
      cur = aho[cur].ch[c];
    }

    aho[cur].end++; // quantas strings esse no representa
  }

  // Calcula o suffix link dos nos
  void build() {
    queue<pair<char, int> > q;
    for(auto & [c, x] : aho[0].ch) {
      q.emplace(c, x);
    }

    while(q.size()) {
```

```
      char c; int cur;
      tie(c, cur) = q.front(); q.pop();

      int & j = aho[cur].suf;
      if(aho[cur].par) {
        j = aho[aho[cur].par].suf;
        while(j and aho[j].ch.count(c) == 0) j = aho[j].suf;
        if(aho[j].ch.count(c)) j = aho[j].ch[c];
      }

      // Se descomentado, guarda a quantidade de strings que s o sufixo da string representada por esse no
      // aho[cur].cnt = aho[cur].end + aho[aho[cur].suf].cnt;

      for(auto & [c, v] : aho[cur].ch) {
        q.emplace(c, v);
      }
    }
  }

  // Proximo estado do Aho Corasick automata. Retorna o no que representa a maior string sufixo da sequencia atual de caracteres
  int next_state(int st, char c) {
    while(st and aho[st].ch.count(c) == 0) st = aho[st].suf;
    if(aho[st].ch.count(c)) st = aho[st].ch[c];
    return st;
  }
};
```

3.2 SuffixTree

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

using namespace std;
typedef long long ll;

const int N = 200007;

struct no {
  int l, r, par, suf;
  string * s;
  map<char, int> f;
  int len() {return r - l + 1;}
  char operator[](int i) {return (*s)[l+i];}
};

no t[N + N];

int new_node(int l, int r, int par, string * str) {
  static int ptr = 0;
  ptr++;
  t[ptr].l = l;
  t[ptr].r = r;
  t[ptr].par = par;
  t[ptr].s = str;
  return ptr;
}


void print_node(int u) {
  printf("Vert = %d\t str = %s\t par = %d\n", u, (*t[u].s).substr(t[u].l, t[u].len()).c_str(), t[u].par);
}

void dfs(int u) {
  if(u) print_node(u);

  for(pair<char, int> g : t[u].f) {
    dfs(g.second);
  }
}

void build(string & s, int n) {
  int i, j, cn, cd, ns = 0;
  i = j = cn = cd = 0;

  t[0].r = -1;

  // invariante (cn, cd-1) representa a string S[i .. j-1]
```

```
48    for(j = 0; j < n; ++j) {
49      for(; i <= j; ++i) {
50        if((cd == t[cn].len() and t[cn].f.count(s[j])) or (cd != t[cn].len() and t[cn][cd] == s[j])) {
       // se eh o caso 1
51          // atualiza a invariante e vai pra proxima iteracao
52          if(cd == t[cn].len()) {
53            cn = t[cn].f[s[j]];
54            cd = 0;
55          }
56          cd++;
57          break;
58        } else if(cd == t[cn].len()) {
59          t[cn].f[s[j]] = new_node(j, n-1, cn, &s); // como esse novo noh eh uma folha, seu r eh n-1
      automaticamente
60          if(cn) { // voce nao esta na raiz
61            cn = t[cn].suf;
62            cd = t[cn].len();
63          }
64        } else if(cd < t[cn].len()) { // caso = 3
65          // Divide a aresta no meio, criando um novo noh interno e um novo noh folha
66          int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].par, &s);
67          t[t[mid].par].f[t[mid][0]] = mid;
68
69          t[mid].f[s[j]] = new_node(j, n-1, mid, &s);
70          t[mid].f[t[cn][cd]] = cn;
71          t[cn].l += cd;
72          t[cn].par = mid;
73          if(ns) t[ns].suf = mid;
74
75
76          // Parte 2
77          cn = t[mid].par;
78          int g; // g indica que o cn representa o cara s[i+1, g-1]. Quero atualizar cn ateh que g
      seja igual a j
79          if(cn) {
80            cn = t[cn].suf;
81            g = j - cd;
82          } else g = i+1;
83
84          while(g < j and g + t[t[cn].f[s[g]]].len() <= j) {
85            cn = t[cn].f[s[g]];
86            g += t[cn].len();
87          }
88          if(g == j) {
89            t[mid].suf = cn;
90            cd = t[cn].len();
91            ns = 0;
92          } else {
93            ns = mid;
94            cn = t[cn].f[s[g]];
95            cd = j - g;
96          }
97        }
98      }
99    }
100 }
101
102 bool find(string & s) {
103   int cn, cd;
104   cn = 0;
105   cd = -1;
106
107   for(int i = 0; i < s.size(); ++i) {
108     char c = s[i];
109     if(cd + 1 == t[cn].len()) {
110       if(t[cn].f.count(c)) {
111         cn = t[cn].f[c];
112         cd = 0;
113       } else return false;
114     } else {
115       if(c == t[cn][cd+1]) ++cd;
116       else return false;
117     }
118   }
119
120   return true;
121 }
122
123 int main() {
```

```
  string s;
  cin >> s;

  build(s, s.size());

  int q;
  cin >> q;

  dfs(0);

  for(int i = 0; i < q; ++i) {
    string str;
    cin >> str;

    if(find(str)) cout << "YES" << endl;
    else cout << "NO" << endl;
  }
}
```

3.3 KMPAutomata

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

using namespace std;


// what the string stores
template<typename T>
struct KMPAutomata {
  vector<int> phi;
  vector<unordered_map<T, int>> to;
  vector<T> str;

  KMPAutomata(){}
  KMPAutomata(const vector<T>& vec){
    build(vec);
  }

  KMPAutomata(const string& s) {
    build(vector<char>(s.begin(), s.end()));
  }

  void build(vector<T> s) {
    phi.resize(s.size());
    to.resize(s.size() + 1);
    str = s;

    phi[0] = 0;
    for (int i = 1; i < s.size(); i++) {
      int& j = phi[i];
      j = phi[i-1];
      while(j and s[j] != s[i]) j = phi[j-1];

      if (s[i] == s[j]) j++;
    }

  }

  // next state if I add character c
  int next(int j, T c) {

    if(to[j].count(c)) return to[j][c];
    int & memo = to[j][c];

    while(j and (j == str.size() or c != str[j])) j = phi[j - 1];

    if(c == str[j]) j++;

    return memo = j;
  }
};
```

3.4 SuffixArray

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

using namespace std;

namespace SuffixArray {
```

```
  // returns lcp array, with size a.size()-1, in which lcp[i] represents
  // the longest common prefix between suffix in position i and the one in i+1
  // Complexidade: O(n)
  vector<int> build_lcp(vector<int> const& a, string const& s) {
    int n = a.size();
    vector<int> pos(n), lcp(n-1);
    for(int i = 0; i < n; ++i) {
      pos[a[i]] = i;
    }

    int sz = 0;
    for(int i = 0; i < n; ++i) {
      if(pos[i] == n-1) continue;
      int j = a[pos[i] + 1];
      while(i + sz < n and j + sz < n and s[i + sz] == s[j + sz]) sz++;
      lcp[pos[i]] = sz;
      if(sz) sz--;
    }
    return lcp;
  }


  // Retorna um vetor com os indices de inicio de prefixo.
  // Complexidade: O(n logn )
  vector<int> build_sufix_array(string s) {
    s += '$';
    int n = s.size();
    vector<int> head(n), a(n), a1(n), c(n), c1(n);

    iota(a.begin(), a.end(), 0);
    sort(a.begin(), a.end(), [&](int i, int j) {
      return s[i] < s[j];
    });

    int cc = 0;
    for(int i = 0; i < n; ++i) {
      if(i == 0 or s[a[i]] != s[a[i-1]]) {
        c[a[i]] = cc;
        head[cc++] = i;
      } else c[a[i]] = c[a[i-1]];
    }

    int l = 1;
    while(l < n) {
      for(int i = 0; i < n; ++i) {
        int j = (a[i] - l + n)%n;
        a1[head[c[j]]++] = j;
      }
      cc = 0;

      head.assign(head.size(), 0);
      for(int i = 0; i < n; ++i) {
        if(i == 0 or c[a1[i]] != c[a1[i-1]] or c[(a1[i] + l)%n] != c[(a1[i-1] + l)%n]) {
          head[cc] = i;
          c1[a1[i]] = cc++;
        } else c1[a1[i]] = c1[a1[i-1]];
      }

      a = a1;
      c = c1;
      l <<= 1;
    }

    return a;
  }

};
```

3.5 SuffixAutomaton

```
/** Suffix automaton structure. O(N) to build */
struct SuffixAutomaton {
  struct State {
    int len, link;
    map<char, int> next;
  };

  vector<State> st;
```

```
  int last;

  SuffixAutomaton(string const& s) {
    int n = s.size();
    st.reserve(2 * n);

    st.push_back(State{0, -1});
    last = 0;

    for (int i = 0; i < n; i++) { // extend current char
      st.push_back(State{i + 1, 0});
      int r = (int) st.size() - 1;

      int p = last;
      while(p >= 0 && st[p].next.find(s[i]) == st[p].next.end()) {
        st[p].next[s[i]] = r;
        p = st[p].link;
      }

      if (p != -1) {
        int q = st[p].next[s[i]];
        if (st[p].len + 1 == st[q].len) {
          st[r].link = q;
        } else { // split and add q'
          st.push_back(State{st[p].len + 1, st[q].link, st[q].next});
          int qq = (int) st.size() - 1;

          st[q].link = st[r].link = qq;
          while(p >= 0 && st[p].next[s[i]] == q) {
            st[p].next[s[i]] = qq;
            p = st[p].link;
          }
        }
      }

      last = r;
    }
  }
};
```

3.6 Hash

```
// https://cses.fi/problemset/task/1110/

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

struct Hash {
  static constexpr int MOD[2] = {(int) 1e9+7, (int) 1e9+9};
  int val[2];

  Hash() { val[0] = val[1] = 0; }
  Hash(string const& s) { *this = calculateHash(s); }
  Hash(int x) { val[0] = x % MOD[0]; val[1] = x % MOD[1]; }
  Hash(int x, int y) { val[0] = x % MOD[0]; val[1] = y % MOD[1]; }

  inline static int add(int x, int y, int k) { x += y; if (x >= MOD[k]) x -= MOD[k]; return x; }
  inline static int sub(int x, int y, int k) { x -= y; if (x < 0) x += MOD[k]; return x; }
  inline static int mul(int x, int y, int k) { return 1ll * x * y % MOD[k]; }
  inline static int fpow(int x, int y, int k) {
    int r = 1;
    for (; y > 0; y /= 2, x = mul(x, x, k))
      if (y % 2 == 1) r = mul(r, x, k);
    return r;
  }
  inline static int divi(int x, int y, int k) { return mul(x, fpow(y, MOD[k] - 2, k), k); }
  inline static Hash pow(Hash x, int y) {
    Hash r = 1;
    for (; y >= 0; y /= 2, x *= x)
      if (y%2 == 1) r *= x;
    return r;
  }

  inline Hash operator+(Hash const& h) const { return Hash(add(val[0], h.val[0], 0), add(val[1], h.
    val[1], 1)); }
  inline Hash operator-(Hash const& h) const { return Hash(sub(val[0], h.val[0], 0), sub(val[1], h.
    val[1], 1)); }
  inline Hash operator*(Hash const& h) const { return Hash(mul(val[0], h.val[0], 0), mul(val[1], h.
    val[1], 1)); }
```

```
  inline Hash operator/(Hash const& h) const { return Hash(divi(val[0], h.val[0], 0), divi(val[1], h.val[1], 1)); }
  inline Hash& operator+=(Hash const& h) { return *this = *this + h; }
  inline Hash& operator-=(Hash const& h) { return *this = *this - h; }
  inline Hash& operator*=(Hash const& h) { return *this = *this * h; }
  inline Hash& operator/=(Hash const& h) { return *this = *this / h; }

  inline bool operator==(Hash const& h) const { return val[0] == h.val[0] && val[1] == h.val[1]; }
  inline bool operator!=(Hash const& h) const { return val[0] != h.val[0] || val[1] != h.val[1]; }

  inline static Hash calculateHash(string const& s, Hash const primes = Hash(31, 37)) {
    Hash cur = 0;
    Hash p = 1;
    for (char c : s) {
      cur += p * (c - 'a' + 1); // assuming that is a lowercase string
      p *= primes;
    }
    return cur;
  }
  inline static vector<Hash> calculateHashVector(string const& s, Hash const primes = Hash(31, 37)) {
    int n = s.size();
    Hash p = 1;
    vector<Hash> cur(n);
    for (int i = 0; i < n; i++) {
      if (i) cur[i] = cur[i-1];
      cur[i] += p * (s[i] - 'a' + 1);
      p *= primes;
    }
    return cur;
  }
  inline static vector<Hash> calculatePowerVector(Hash p, const int n) {
    vector<Hash> ans(n);
    ans[0] = 1;
    for (int i = 1; i < n; i++)
      ans[i] = ans[i-1] * p;
    return ans;
  }
  inline static Hash getRange(vector<Hash> const& v, vector<Hash> const& invPow, int l, int r) {
    return (v[r] - (l ? v[l-1] : 0)) * invPow[l];
  }
};

ostream& operator<<(ostream& out, Hash const& h) {
  return out << "[" << h.val[0] << "," << h.val[1] << "]";
}

#define hash UISHDUIAHSDU

/** NOTES:
 *    at the beginning, precalc:
 *    vector<Hash> hash = Hash::calculateHashVector(s, primes);
 *    vector<Hash> power = Hash::calculatePowerVector(primes, n);
 *    vector<Hash> invPow = Hash::calculatePowerVector(Hash(1)/primes, n);
 * */

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

inline void minimalRotation(string s) {
  int n = s.size();
  s += s;

  Hash primes(31, 29);
  vector<Hash> hash = Hash::calculateHashVector(s, primes);
  vector<Hash> power = Hash::calculatePowerVector(primes, n);
  vector<Hash> invPow = Hash::calculatePowerVector(Hash(1)/primes, n);

  // lexicographically compares two substrings: s[A..A+n] and s[B..B+n]
  auto cmp = [&](int const A, int const B) {
    if (s[A] != s[B]) return s[A] < s[B];
    Hash h1, h2;
    int lo = 2, hi = n, mi;

    while(lo < hi) {
      mi = (lo + hi) / 2;
      Hash h1 = Hash::getRange(hash, invPow, A, A + mi - 1);
      Hash h2 = Hash::getRange(hash, invPow, B, B + mi - 1);
      if (h1 != h2) hi = mi;
      else lo = mi + 1;
```

```
    }
    return s[A+hi-1] < s[B+hi-1];
  };

  vector<int> a(n);
  iota(a.begin(), a.end(), 0);
  nth_element(a.begin(), a.begin(), a.end(), cmp);
  cout << s.substr(a[0], n) << "\n";
}

int main() {
  ios::sync_with_stdio(false); cin.tie(NULL);
  string s; cin >> s;
  minimalRotation(s);
}
```

3.7 KMP

```
/** Generate prefix function of KMP. O(N) */
template<typename T> std::vector<int> prefixKMP(T const& v) {
  int n = v.size();
  std::vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    pi[i] = pi[i-1];
    while(pi[i] > 0 && v[pi[i]] != v[i])
      pi[i] = pi[pi[i]-1];
    if (v[i] == v[pi[i]]) pi[i]++;
  }
  return pi;
}
```

3.8 PalindromicTree

```
/*
  A palindromic tree stores all distinct palindromes that appear as substrings of a string S.
  There are at most |S| of them, believe me. Proof is by induction. If S = T + c, when adding
    character
  c, take the biggest palindrome created ending at c, let's call it P. If there would exist smaller
    palindromes
  created at this moment, they would have already appeared as a prefix of P. That means the only new
     palindrome
  is the biggest one.
  Tree has two roots: -1 and 0. All nodes in subtree of -1 have odd length, and all nodes in subtree
     of 0 have even
  length. An edge of this tree with character c (U -- c --> V) means that V = cUc. Edge (-1 -- c -->
     V) means that V = c.
  A suffix link U --> V means that V is the greatest suffix of U that is also a palindrome.
  Construction:

  Create roots 0 and -1 and suffix link of 0 goes to -1 and suffix link of -1 goes to itself.
  Invariant: keep "last" pointer as the pointer to the last node added. "last" starts at -1
  Add character c:
  Lets find greatest palindrome ending at c. To do that, let's find biggest suffix link (S) of last
    that has character c
  to it's left, and then do S -- c --> U if U is not already there.
  To find suffix link, find second largest suffix link of last that has c to it's left. If it doesn'
    t exist, suffix link is 0
  Properties:
  - Level of a node tells it's length
  - Suffix links of a node tells all nodes that end in a position
  - Suffix links forms a tree
  Applications:
  - Count number of distinct substrings that are palindromes
  - Count number of times a palindrome appears as substring
  - Find sum of lengths of all substring palindromes
  - Count triples i <= j < k such that Si,j is pal. and Sj+1,k it pal.
  - Find largest common palindrome substring between K strings.
  - Find kth smaller palindrome: precompute all palindromes and their frequencies and sort them
    using hash lcp
*/

#include <bits/stdc++.h>

using namespace std;

struct node {
  map<int, int> ch; // children
  int link, cnt; // how many times it appears as subtring
```

```
  int start; // first time this palindrome appears as subtring. If you want the last, just reverse the string
  int sz; // size of the palindrome
};

const int N = 100007;
node _tree[N + 2];
node * tree = _tree + 1;

void build(string & s, bool frequency_of_appearence = false) {
  tree[-1].link = -2;
  tree[-1].sz = -1;
  tree[0].link = -1;
  tree[0].sz = 0;

  int tt = 0;
  int last = -1;

  for(int i = 0; i < s.size(); ++i) {
    char c = s[i];
    // Add a new character
    while(i - tree[last].sz == 0 or s[i - tree[last].sz - 1] != c) last = tree[last].link;

    if(tree[last].ch.count(c) == 0) {
      ++tt;
      tree[tt].sz = tree[last].sz + 2;
      tree[tt].cnt = 0;
      tree[tt].start = i - tree[tt].sz + 1;

      tree[last].ch[c] = tt;

      // Finds suffix link
      int u = tree[last].link;
      while(u != -2 and s[i - tree[u].sz - 1] != c) u = tree[u].link;
      tree[tt].link = u == -2? 0 : tree[u].ch[c];
    }
    last = tree[last].ch[c];
    tree[last].cnt++;
    // cout << "Found palindrome " <<  last << " as " << s.substr(tree[last].start, tree[last].sz) << endl;
  }

  // If you want to calculate frequencies for each palindrome
  if(frequency_of_appearence) {
    for(int i = tt; i > 0; --i) {
      tree[tree[i].link].cnt += tree[i].cnt;
    }
  }
}

deque<char> d;
void dfs(int u) {
  cout << string(d.begin(), d.end()) << " appears " << tree[u].cnt << " times, and starts at " << tree[u].start << endl;
  for(auto & [c, v] : tree[u].ch) {
    d.push_front(c);
    if(u == -1) d.pop_front();
    d.push_back(c);
    dfs(v);
    if(u != -1) d.pop_front();
    d.pop_back();
  }
}

int main() {
  ios::sync_with_stdio(0); cin.tie(0);

  string s = "abacaba";
  build(s, true);

  dfs(-1);
  dfs(0);
}
```

3.9 Z

```
/** Generate Z function. O(N) */
template<typename T> std::vector<int> buildZ(T const& v) {
  int n = v.size();
```

```
  std::vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = std::min(r - i + 1, z[i - l]);
    while(i + z[i] < n && v[z[i]+i] == v[z[i]]) z[i]++;
    if (z[i] + i - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

4 DataStructures

4.1 SegTreeLazy

```
struct Seg {
  struct Node {
    ll v = 0; // initialize empty values; HERE
    ll lazy = 0;

    inline void apply(int i, int j, ll x) {
      lazy += x; // HERE
    }
  };

  int n;
  vector<Node> seg;
  Seg(int n) : n(n), seg(4 * n) {}

  Node merge(Node const& L, Node const& R) { // HERE (merge children)
    Node ret;
    ret.v = min(L.v, R.v);
    return ret;
  }

  void prop(int p, int i, int j) { // HERE (lazy propagation)
    if (seg[p].lazy) {
      seg[p].v += seg[p].lazy;
      if (i != j) {
        seg[p + p].lazy += seg[p].lazy;
        seg[p + p + 1].lazy += seg[p].lazy;
      }
      seg[p].lazy = 0;
    }
  }

  template<typename T>
  void build(int p, int i, int j, vector<T> const& v) {
    if (i == j) {
      seg[p].apply(i, j, v[i]);
      prop(p, i, j);
    } else {
      int m = (i + j) / 2;
      build(p + p, i, m, v);
      build(p + p + 1, m + 1, j, v);
      seg[p] = merge(seg[p + p], seg[p + p + 1]);
    }
  }

  template<typename T>
  inline void build(vector<T> const& v) {
    build(1, 0, n - 1, v);
  }

  Node query(int p, int i, int j, int l, int r) {
    prop(p, i, j);
    assert(i <= r && j >= l);
    if (i >= l && j <= r) return seg[p];
    int m = (i + j) / 2;
    if (m >= r) {
      prop(p + p + 1, m + 1, j);
      return query(p + p, i, m, l, r);
    } else if (m < l) {
      prop(p + p, i, m);
      return query(p + p + 1, m + 1, j, l, r);
    }
    Node L = query(p + p, i, m, l, r);
    Node R = query(p + p + 1, m + 1, j, l, r);
    return merge(L, R);
  }
```

```
  inline Node query(int l, int r) {
    return query(1, 0, n-1, l, r);
  }

  template<typename... T>
  void update(int p, int i, int j, int l, int r, T... x) {
    prop(p, i, j);
    if (i > r || j < l) return;
    if (i >= l && j <= r) {
      seg[p].apply(i, j, x...);
      prop(p, i, j);
      return;
    }
    int m = (i + j) / 2;
    update(p + p, i, m, l, r, x...);
    update(p + p + 1, m + 1, j, l, r, x...);
    seg[p] = merge(seg[p + p], seg[p + p + 1]);
  }

  template<typename... T>
  inline void update(int l, int r, T... x) {
    update(1, 0, n-1, l, r, x...);
  }
};
```

4.2 CartesianTree

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

using namespace std;

template<typename T>
struct Cartesian {
  vector<int> L, R;
  vector<T> vec;
  stack<int> st;

  // root in data structure is always 0. Actual root is R[0]
  Cartesian() {
    // sentinel
    st.emplace();
    L.eb(), R.eb();
    vec.eb();
  }

  // O(n) amortized after n invocations, but O(tree height) in worst case.
  void add(int x) {
    vec.eb(x);
    int u = vec.size()-1;
    L.eb(), R.eb();

    while(st.size() > 1 and vec[st.top()] > vec.back()) {
      st.pop();
    }

    int v = st.top();
    L.back() = R[v];
    R[v] = u;
    st.emplace(u);
  }

  // returns index of the root of the tree
  int root() const {
    return R[0];
  }

  // get value of node i
  T& operator[](int i) const {
    return vec[i];
  }

  int left(int i) const {
    return L[i];
  }

  int right(int i) const {
    return R[i];
  }
```

```
52
53    // in order traversal
54    void traverse(int u, ostream& os) const {
55      if(u == 0) return;
56      traverse(L[u], os);
57      traverse(R[u], os);
58      os << pre[vec[u]] << ' ';
59    }
60
61    // in order traversal
62    void traverse(ostream& os) const {
63      traverse(root(), os);
64    }
65
66    friend ostream& operator<<(ostream& os, const Cartesian& tree) {
67      tree.traverse(os);
68      return os;
69    }
70
71  };
```

4.3 Paretto

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  // a paretto is a structure that has the property that
5  // for every point X there is no other point Y such that
6  // X[0] < Y[0] and X[1] < Y[1]
7  template<typename K, typename V>
8  struct Paretto {
9      using ii = pair<K, V>;
10
11     set<ii> s;
12
13     // adds a point {x, y} to the set
14     void add_point(K const& x, V const& y) {
15         auto p = ii{x, y};
16         auto it = s.lower_bound(p);
17
18         if (it->second >= y) return;
19
20         auto p_it = s.insert(p).first;
21
22         while (p_it != s.begin() and prev(p_it)->second <= y) {
23             s.erase(prev(p_it));
24         }
25     }
26
27     // return the maximum y of a point A such that A[0] >= x
28     V get_best(K x) {
29         auto it = s.lower_bound(ii{x, numeric_limits<V>::min()});
30
31         if (it == s.end()) return numeric_limits<V>::min();
32         return it->second;
33     }
34 };
```

4.4 SparseTable

```
1  template<typename T>
2  struct SparseTable {
3    int n;
4    std::vector<std::vector<T>> mat;
5
6    // Function to apply range query
7    inline static T f(T x, T y) {
8      return min(x, y);
9    }
10
11   // initialize everything
12   SparseTable(std::vector<T> const& a) {
13     n = a.size();
14     const int max_log = 32 - __builtin_clz(n);
15     mat.resize(max_log);
16     mat[0] = a;
17     for (int j = 1; j < max_log; j++) {
18       mat[j].resize(n - (1 << j) + 1);
19       for (int i = 0; i <= n - (1 << j); i++) {
```

```
        mat[j][i] = f(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
      }
    }
  }

  // Query [from, to]. O(1)
  inline T query(int from, int to) const {
    assert(0 <= from && from <= to && to <= n - 1);
    int lg = 32 - __builtin_clz(to - from + 1) - 1;
    return f(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
  }
};
```

4.5 TreapPersistent

```

/*
  Implementa  o de Treap Persistente como array implicito, que    o caso mais comum
*/
namespace PersistentTreap {
  mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
  struct node {
    node * l, * r;
    int cnt, val; // valor que realmente eh guardado na treap

    node(int x = 0) {
      l = r = NULL;
      val = x;
    }
  };
  typedef node * treap;


  treap new_treap() {
    return NULL;
  }

  treap copy(treap t) {
    treap x = new node(t->c);
    memcpy(x, t, sizeof(node));
    return x;
  }

  void upd(treap t) {
    if(t) t->cnt = 1 + (t->l? t->l->cnt : 0) + (t->r? t->r->cnt : 0);
  }

  // splita a treap t colocando os primeiros j caras em 'l' e o resto em 'r'. t na verdade  n  o
    tocado, e logN novos  n  s  sao criados
  void split(treap t, treap & l, treap & r, int j, int cur = 0) {
    if(t == 0) return void(l = r = 0);

    int men = cur + (t->l? t->l->cnt : 0) + 1;

    if(men > j) {
      r = copy(t);
      split(t->l, l, r->l, j, cur);
      upd(r);
    } else {
      l = copy(t);
      split(t->r, l->r, r, j, men);
      upd(l);
    }
  }

  // t vira a concatenacao de 'l' e 'r'. 'l' e 'r' nao sao tocados
  void merge(treap & t, treap l, treap r) {
    if(!l or !r) t = (l? l : r);

    else if(rng()%(cn(l)+cn(r)) <= cn(l)) {
      t = copy(l);
      merge(t->r, t->r, r);
    } else {
      t = copy(r);
      merge(t->l, l, t->l);
    }

    upd(t);
  }
```

```
};
```

4.6 RandomizedHeap

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

mt19937 rd(10);
uniform_int_distribution<int> toss(0, 1);

// randomized heap https://cp-algorithms.com/data_structures/randomized_heap.html
// capable of O(log n) insertion, pop_min, melding (combining two heaps)
// and O(1) get_min
template<typename T>
struct RandomizedHeap {
    using ptr = unique_ptr<RandomizedHeap<T>>;

    T val;
    ptr l, r;

    RandomizedHeap() = default;
    RandomizedHeap(T x) : val(x), l(nullptr), r(nullptr) {};

    // combines two heaps a and b
    // consumes heap a and b, and procudes a new heap
    friend auto meld(ptr a, ptr b) -> ptr {
        if (!a) return b;
        if (!b) return a;

        // a is the root
        if (a->val > b->val) swap(a, b);

        // 50/50 chance to combine b with each child of a
        if (toss(rd)) a->l = meld(move(a->l), move(b));
        else a->r = meld(move(a->r), move(b));

        return a;
    }

    friend auto insert(ptr& a, T x) -> void {
        a = meld(move(a), make_unique<RandomizedHeap>(x));
    }

    friend auto pop(ptr& a) -> void {
        a = meld(move(a->l), move(a->r));
    }

    auto min() -> T {
        return val;
    }
};


int main() {
    RandomizedHeap<int>::ptr a, b;

    for (int i = 0; i < 10; i += 2) {
        insert(a, i);
        insert(b, i+1);
    }

    cout << a->min() << '\n';
    cout << b->min() << '\n';

    pop(a), pop(a);
    pop(b);

    auto c = meld(move(a), move(b));

    while (c) {
        cout << c->min() << ' ';
        pop(c);
    }
    cout << '\n';
}
```

4.7 SegTree

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

```
using namespace std;

struct SegTree {
    struct Node {
        long long x;

        static auto join(Node const& lhs, Node const& rhs) -> Node {
            return {lhs.x + rhs.x};
        }

        void update(Node rhs) {
            x += rhs.x;
        }
    };

    int n;
    vector<Node> tree;

    SegTree(int n) :
        n(n), tree(n*4)
    {};

    SegTree(vector<Node> const& v) :
        n(v.size()), tree(n*4)
    {
        build(v, 0, 0, n-1);
    }

    auto build(vector<Node> const& v, int no, int l, int r) -> void {
        if (l == r) {
            tree[no] = v[l];
        }
        else {
            int m = (l+r)/2;
            build(v, no*2+1, l, m);
            build(v, no*2+2, m+1, r);

            tree[no] = Node::join(tree[no*2+1], tree[no*2+2]);
        }
    }

    auto get(int no, int l, int r, int a, int b) -> Node {
        if (a <= l and r <= b) {
            return tree[no];
        }

        int m = (l+r)/2;

        if (b <= m) return get(no*2+1, l, m, a, b);
        else if (a > m) return get(no*2+2, m+1, r, a, b);

        return Node::join(
            get(no*2+1, l, m, a, b),
            get(no*2+2, m+1, r, a, b)
        );
    }

    auto get(int a, int b) -> Node {
        return get(0, 0, n-1, a, b);
    }

    auto upd(int no, int l, int r, int p, Node const& val) -> void {
        if (l == r) {
            tree[no].update(val);
            return;
        }

        int m = (l+r)/2;

        if (p <= m) upd(no*2+1, l, m, p, val);
        else upd(no*2+2, m+1, r, p, val);

        tree[no] = Node::join(tree[no*2+1], tree[no*2+2]);
    }

    auto upd(int p, Node const& val) -> void {
        upd(0, 0, n-1, p, val);
    }
};
```

4.8 MergeSortTree

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

template<typename T>
struct MergeSortTree {
    int n;
    vector<vector<T>> tree;

    MergeSortTree(vector<T> const& v) :
        n(v.size()), tree(n * 4)
    {
        build(v, 0, 0, n-1);
    }

    auto build(vector<T> const& v, int no, int l, int r) -> void {
        if (l == r) {
            tree[no] = {v[l]};
        }
        else {
            int m = (l+r)/2;

            build(v, no*2+1, l, m);
            build(v, no*2+2, m+1, r);

            tree[no].resize(r-l+1);

            merge(
                tree[no*2+1].begin(), tree[no*2+1].end(),
                tree[no*2+2].begin(), tree[no*2+2].end(),
                tree[no].begin()
            );
        }
    }

    // returns number of elements in sorted vector v in [x, y)
    auto count_range(vector<T> const& v, int x, int y) -> int {
        return lower_bound(v.begin(), v.end(), y) - lower_bound(v.begin(), v.end(), x);
    }

    // return numbe
    auto count_query(int a, int b, int x, int y) -> int {
        return count_query(0, 0, n-1, a, b, x, y);
    }

    auto count_query(int no, int l, int r, int a, int b, int x, int y) -> int {
        if (a <= l and r <= b) {
            return count_range(tree[no], x, y);
        }
        else {
            int m = (l+r)/2;

            if (b <= m) {
                return count_query(no*2+1, l, m, a, b, x, y);
            }
            else if (a > m) {
                return count_query(no*2+2, m+1, r, a, b, x, y);
            }
            else {
                return count_query(no*2+1, l, m, a, b, x, y) +
                       count_query(no*2+2, m+1, r, a, b, x, y);
            }
        }
    }
};
```

4.9 Treap

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

mt19937 rd(10);
namespace Treap {

template<typename T>
struct Node {
    T val;
    int sz, h;
```

```
    Node *l, *r;
    bool flip = false;
    T sum = 0;

    Node(T const& v) : val(v), sz(1), h(rd()), l(nullptr), r(nullptr), sum(v) {};
};

using T = int;
using ptr = Node<T>*;

static void flush(ptr t) {
    if (!t) return;
    if (t->flip) {
        if (t->l) t->l->flip ^= true;
        if (t->r) t->r->flip ^= true;
        swap(t->l, t->r);
        t->flip = false;
    }
}

static void op(ptr t) {
    if (!t) return;

    t->sz = 1;
    t->sum = t->val;

    if (t->l) {
        t->sz += t->l->sz;
        t->sum += t->l->sum;
    }
    if (t->r) {
        t->sz += t->r->sz;
        t->sum += t->r->sum;
    }
}

static ptr merge(ptr l, ptr r) {
    if (!l) return r;
    if (!r) return l;
    flush(l), flush(r);

    if (l->h > r->h) {
        l->r = merge(l->r, r);
        op(l);
        return l;
    }
    else {
        r->l = merge(l, r->l);
        op(r);
        return r;
    }
}

static pair<ptr, ptr> split(ptr t, T const& v) {
    if (!t) return {};
    flush(t);

    ptr x;
    if (t->val < v) {
        tie(t->r, x) = split(t->r, v);
        op(t);
        return {t, x};
    }
    else {
        tie(x, t->l) = split(t->l, v);
        op(t);
        return {x, t};
    }
}

static pair<ptr, ptr> split_pos(ptr t, int pos) {
    if (!t) return {};
    flush(t);

    int p = (t->l ? t->l->sz : 0) + 1;

    ptr x;
    if (pos >= p) {
        tie(t->r, x) = split_pos(t->r, pos-p);
```

```
        op(t);
        return {t, x};
    }
    else {
        tie(x, t->l) = split_pos(t->l, pos);
        op(t);
        return {x, t};
    }
}

static void insert(ptr& root, T const& v) {
    auto [l, r] = split(root, v);
    l = merge(l, new Node(v));
    root = merge(l, r);
}

static void insert_pos(ptr& root, T const& v, int pos) {
    auto [l, r] = split_pos(root, pos);
    l = merge(l, new Node(v));
    root = merge(l, r);
}

static void erase(ptr& root, T const& v) {
    auto [l, aux] = split(root, v);
    ptr r;
    tie(aux, r) = split_pos(aux, 1);
    root = merge(l, r);
}

static void print(ptr root) {
    if (!root) return;
    flush(root);
    print(root->l);
    cout << root->val;
    print(root->r);
}

}; // end namespace Treap
```

4.10 DSUPartial

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

struct DSUPartial {
  int n, curTime;
  std::vector<int> par, sz, tim;

  // Union Find with union by size.
  // "Partially persistent": past states can be seen, but not modified
  // operations in O(log(n))
  DSUPartial(int n) : n(n) {
    par.resize(n + 1);
    tim.resize(n + 1);
    sz = std::vector<int>(n + 1, 1);
    iota(par.begin(), par.end(), 0);
    curTime = 0;
  }

  // find leader of current component on time t
  int find(int x, int t = INT_MAX) {
    return par[x] == x || tim[x] > t ? x : find(par[x], t);
  }

  // merge two components at time t
  // returns whether they don't already belong to the same
  inline bool merge(int u, int v, int t) {
    u = find(u, t), v = find(v, t);
    if (u == v) return false;

    if (sz[u] > sz[v]) std::swap(u, v);
    par[u] = v;
    tim[u] = t;
    sz[v] += sz[u];
    return true;
  }

  // merge two componnets
  // returns whether they don't already belong to the same
```

```
  inline bool merge(int u, int v) {
    if (merge(u, v, curTime + 1)) {
      ++curTime;
      return true;
    }
    return false;
  }

  // returns whether u and v are at the same component at time t
  inline bool same(int u, int v, int t) {
    return find(u, t) == find(v, t);
  }
};

// Pictionary. https://codeforces.com/gym/102078/problem/A
int main() {
  int n, m, q; scanf("%d%d%d", &n, &m, &q);

  DSUPartial uf(n);
  for (int k = m; k >= 1; k--) {
    for (int i = k + k; i <= n; i += k) {
      uf.merge(i, k, m - k + 1);
    }
  }

  while(q--) {
    int u, v; scanf("%d%d", &u, &v);
    int lo = 0, hi = m, mi;
    while(lo < hi) {
      mi = (lo + hi) / 2;
      if (uf.same(u, v, mi)) hi = mi;
      else lo = mi + 1;
    }
    printf("%d\n", hi);
  }
}
```

4.11 SegTreeBeats

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

/*
  Segment Tree Beats:

  Task 1:

  - For all l <= i <= r, change A_i to max(A_i, x)
  - Query for sum of [l, r]

  Task 2:

  - For all l <= i <= r, change A_i to max(A_i, x)
  - For all l <= i <= r, change A_i to min(A_i, x)
  - For all l <= i <= r, change A_i to A_i + x, x any integer
  - Query for sum of [l, r]

  Qualquer outro problema pode ser resolvido genericamente com o codigo, faltando so a analise da
    complexidade:

  void modify(int node, int l, int r, int ll, int rr) {
    if (break_condition()) return;
    if (tag_condition()) {
        puttag(node); return;
    }
    pushdown(node);
    int mid = (l + r) >> 1;
    modify(node * 2, l, mid, ll, rr);
    modify(node * 2 + 1, mid + 1, r, ll ,rr);
    update();
  }
*/

using namespace std;
typedef long long ll;

// Task 1
struct SegtreeBeats {
  static constexpr int inf = 0x3f3f3f3f;

```

```
#define mid ((l+r)>>1)
int n;
vector<ll> seg;
vector<int> mai, cnt_mai, seg_mai, lazy;

SegtreeBeats(int sz = 0): n(sz), seg(4*sz + 2), mai(4*sz + 2), cnt_mai(4*sz + 2), seg_mai(4 * sz + 2), lazy(4*sz + 2) {
  build(1, 0, sz-1);
}

SegtreeBeats(vector<int> & vec) {
  *this = SegtreeBeats(vec.size());

  for(int i = 0; i < n; ++i) {
    update(1, 0, n-1, i, i, vec[i]);
  }
}

void merge(int p, int l, int r) {
  if(mai[l] == mai[r]) {
    mai[p] = mai[l];
    cnt_mai[p] = cnt_mai[l] + cnt_mai[r];
    seg_mai[p] = max(seg_mai[l], seg_mai[r]);
  } else {
    if(mai[l] < mai[r]) swap(l, r);
    mai[p] = mai[l];
    cnt_mai[p] = cnt_mai[l];
    seg_mai[p] = max(seg_mai[l], mai[r]);
  }

  seg[p] = seg[l] + seg[r];
}

void build(int p, int l, int r) {
  if(l == r) {
    mai[p] = seg[p] = inf;
    seg_mai[p] = -inf; // menor valor que posso ter
    cnt_mai[p] = 1;
  } else {
    build(2 * p, l, mid);
    build(2 * p + 1, mid+1, r);
    merge(p, 2 * p, 2 * p + 1);
  }
  lazy[p] = inf;
}


void prop(int p, int l, int r) {
  if(lazy[p] < mai[p]) {
    seg[p] -= ll(mai[p] - lazy[p]) * cnt_mai[p];
    mai[p] = lazy[p];
  }

  if(l != r) {
    lazy[2 * p] = min(lazy[2 * p], lazy[p]);
    lazy[2 * p + 1] = min(lazy[2 * p + 1], lazy[p]);
  }

  lazy[p] = inf;
}

// para todo i em [a, b], atualiza seg_i = min(seg_i, x)
void update(int p, int l, int r, int a, int b, int x) {
  prop(p, l, r);
  if(r < a or l > b or mai[p] <= x) return;
  if(l >= a and r <= b and seg_mai[p] < x) {
    lazy[p] = min(lazy[p], x);
    prop(p, l, r);
  } else {
    update(2 * p, l, mid, a, b, x);
    update(2 * p + 1, mid+1, r, a, b, x);
    merge(p, 2 * p, 2 * p + 1);
  }
}

// retorna a soma de [a, b]
ll query(int p, int l, int r, int a, int b) {
  prop(p, l, r);
  if(r < a or l > b) return 0;
```

```
    if(l >= a and r <= b) return seg[p];
    return query(2 * p, l, mid, a, b) + query(2 * p + 1, mid+1, r, a, b);
  }

  #undef mid
};
```

4.12 BIT2D

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

template<typename T>
struct Bit2D {
    int n, m;
    vector<vector<T>> b;

    Bit2D(int n_, int m_) :
        n(n_+2), m(m_+2), b(n, vector<T>(m))
    {}

    // adds delta to position (x, y)
    auto add(int x, int y, T delta) -> void {
        x++, y++;
        for (int i = x; i < n; i += i&-i) {
            for (int j = y; j < n; j += j&-j) {
                b[i][j] += delta;
            }
        }
    }

    // returns sum of rectangle [0, x] x [0, y]
    auto get(int x, int y) -> T {
        x++, y++;
        T ans = 0;
        for (int i = x; i; i -= i&-i) {
            for (int j = y; j; j -= j&-j) {
                ans += b[i][j];
            }
        }
        return ans;
    }

    // returns sum of rectagle defined by [x1, x2] x [y1, y2]
    // generalizable to N-dimensional BIT using inclusion-exclusion principle
    auto get_rectangle(int x1, int x2, int y1, int y2) -> T {
        return get(x2, y2) - get(x1-1, y2) - get(x2, y1-1) + get(x1-1, y1-1);
    }
};
```

4.13 DSURollback

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

using namespace std;
typedef long long ll;
typedef pair<int, int> ii;

struct UnionFind {
  vector<int> un, sz;
  stack<ii> ops;
  stack<int> checkpoints;

  UnionFind(){}
  UnionFind(int n) {
    un.resize(n + 5);
    sz.assign(n + 5, 1);
    iota(un.begin(), un.end(), 0);
  }

  int find(int u) {
    return un[u] == u? u : find(un[u]);
  }

  void unite(int u, int v) {
    u = find(u); v = find(v);
    if(u == v) return;

    if(sz[v] > sz[u]) swap(u, v);
```

```
    ops.emplace(v, un[v]);
    un[v] = u;
    ops.emplace(u, sz[u]);
    sz[u] += sz[v];
  }

  void save() {
    checkpoints.emplace(ops.size());
  }

  void undo() {
    if(ops.size() == 0) return;
    assert(ops.size()%2 == 0);
    for(int i = 0; i < 2; i++) {
      auto [a, b] = ops.top(); ops.pop();
      if(i == 0) {
        sz[a] = b;
      } else {
        un[a] = b;
      }
    }
  }

  void rollback() {
    if(checkpoints.size() == 0) return;
    while(ops.size() > checkpoints.top()) {
      undo();
    }

    checkpoints.pop();
  }
};
```

4.14 BIT

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

// can be used with all operations that have an inverse
// can also be used in special situation if you dont need an inverse operation
// eg. the operation is max and the numbers added are increasing
template<typename T>
struct BIT {
// if you need it to cover a dynamic range such as from 0 to 1e9:
// change vector to map and choose an upper limit insted of b.size()s
    vector<T> b;

    BIT() = default;
    BIT(int n) : b(n+1) {};

// if you want to access position 0, add pos++ in the first line
    auto add(int pos, T x) -> void {
        for (int i = pos; i < (int)b.size(); i += i&-i) {
            b[i] += x;
        }
    }

// if you want to access position 0, add pos++ in the first line
    auto get(int pos) -> T {
        T r = 0;
        for (int i = pos; i > 0; i -= i&-i) {
            r += b[i];
        }
        return r;
    }

    auto get(int l, int r) -> T {
        return get(r) - get(l-1);
    }
};

// https://www.spoj.com/problems/INVCNT/
auto main() -> int {
    ios::sync_with_stdio(false);

    int t;
    cin >> t;

```

```
    while (t--) {
        int n;
        cin >> n;

        vector<int> v(n);
        for (auto& i : v) cin >> i;

        auto com = v;
        sort(com.begin(), com.end());

        for (auto& i : v) {
            i = lower_bound(com.begin(), com.end(), i) - com.begin() + 1;
        }

        BIT<int> bit(n);
        long long ans = 0;
        for (auto i : v) {
            ans += bit.get(i+1, n);
            bit.add(i, 1);
        }

        cout << ans << '\n';
    }
}
```

4.15 SegTreeIteractive

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

// Example for segtree of sum:
// int sum(int a, int b) { return a + b; }
// segtree<int, 0, sum> seg;
template<typename T, T zero, T (*op)(T, T) >
struct segtree {
  vector<T> seg;
  int n;
  segtree(int n_): n(n_) {
    seg.assign(n_ + n_ + 5, zero);
  }

  T query(int l, int r) {
    T ansl, ansr;
    ansl = ansr = zero;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if(l&1) ansl = op(ansl, seg[l++]);
      if(r&1) ansr = op(seg[--r], ansr);
    }
    return op(ansl, ansr);
  }

  void update(int p, T val) {
    for(seg[p += n] = val; p >>= 1;) {
      seg[p] = op(seg[2 * p], seg[2 * p + 1]);
    }
  }
};
```

4.16 WaveletTree

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

/** Wavelet Tree data structure. Input array is 0 based */
struct waveletTree {
  int lo, hi, mi; // minimum, maximum and (lo+hi)/2 element on array
  waveletTree *L, *R; // children
  vector<int> mp; // map how many elements went left

#define mapLeft(i) (mp[i]) /* original mapLeft */
#define mapRight(i) (i - mapLeft(i)) /* original mapRight */

  /** *beg points to the first element, *end points to after the last element (just like stl default
     functions).
   * Elements are in range [lo..hi], inclusive */
  waveletTree(int *beg, int *end, int lo, int hi) { // O(nlogA)
    L = R = NULL;
    this->lo = lo;
    this->hi = hi;
    this->mi = (lo + hi) / 2;
```

```
    mp.reserve(end - beg + 1);
    mp.push_back(0);
    for (auto it = beg; it != end; it++) {
      mp.push_back(mp.back() + ((*it) <= mi));
    }

    if (lo != hi) {
      auto pivot = stable_partition(beg, end, [&](int x) {
        return x <= mi;
      }); // split the vector
      L = new waveletTree(beg, pivot, lo, mi);
      R = new waveletTree(pivot, end, mi + 1, hi);
    }
  }

  /** K-th smallest element on range[l..r] */
  int kthSmallest(int k, int l, int r) {
    if (l > r) return -1; // out of bounds
    if (lo == hi) return lo; // leaf node

    int inLeft = mapLeft(r) - mapLeft(l-1);
    if (k <= inLeft) return L->kthSmallest(k, mapLeft(l-1)+1, mapLeft(r));
    else return R->kthSmallest(k-inLeft, mapRight(l-1)+1, mapRight(r));
  }

  /** Frequency of elements between [x..y] in array[l..r] */
  int rangeCount(int x, int y, int l, int r) {
    if (l > r || lo > y || hi < x) return 0; // out of bounds
    if (lo >= x && hi <= y) return r - l + 1; // total fit

    if (mi >= y) return L->rangeCount(x, y, mapLeft(l-1)+1, mapLeft(r));
    else if (mi <= x) return R->rangeCount(x, y, mapRight(l-1)+1, mapRight(r));
    else return L->rangeCount(x, mi, mapLeft(l-1)+1, mapLeft(r)) +
         R->rangeCount(mi+1, y, mapRight(l-1)+1, mapRight(r));
  }

  /** Swap elements a[i] and a[i+1] */
  void swapContiguous(int i) {
    if (lo == hi) return; // leaf node, no need to swap

    bool iLeft = mapLeft(i) == mapLeft(i-1) + 1; // if a[i] <= mi
    bool i1Left = (mapLeft(i+1)) == (mapLeft(i) + 1); // if a[i+1] <= mi

    if (iLeft && !i1Left) mapLeft(i)--;
    else if (!iLeft && i1Left) mapLeft(i)++;
    else if (iLeft && i1Left) L->swapContiguous(mapLeft(i));
    else R->swapContiguous(mapRight(i));
  }

  /** Push element k to end of array */
  void push_back(int k) {
    mp.push_back(mp.back() + (k <= mi));
    if (lo != hi) {
      if (k <= mi) L->push_back(k);
      else R->push_back(k);
    }
  }

  /** Pop element k from the end of array */
  void pop_back(int k) {
    mp.pop_back();
    if (lo != hi) {
      if (k <= mi) L->pop_back(k);
      else R->pop_back(k);
    }
  }

  ~waveletTree() {
    if (L) delete L;
    if (R) delete R;
  }

#undef mapLeft
#undef mapRight
};

const int MAXN = 1e5 + 5;
int a[MAXN];
```

```
99
100 // https://www.spoj.com/problems/ILKQUERY/
101 int main() {
102   int n, q; scanf("%d%d", &n, &q);
103
104   vector<int> all;
105   for (int i = 1; i <= n; i++) {
106     scanf("%d", a + i);
107     all.push_back(a[i]);
108   }
109   sort(all.begin(), all.end());
110   all.erase(unique(all.begin(), all.end()), all.end());
111   for (int i = 1; i <= n; i++)
112     a[i] = lower_bound(all.begin(), all.end(), a[i]) - all.begin();
113
114   int N = all.size();
115   vector<vector<int>> pos(N);
116   for (int i = 1; i <= n; i++) pos[a[i]].push_back(i - 1);
117
118   waveletTree T(a+1, a+n+1, 0, all.size());
119   while(q--) {
120     int k, i, l; scanf("%d%d%d", &k, &i, &l);
121     int ans = T.kthSmallest(k, 1, i + 1);
122     printf("%d\n", l <= (int) pos[ans].size() ? pos[ans][l - 1] : -1);
123   }
124
125   return 0;
126 }
```

4.17 TwoPointers

```
1 /*
2   General structure to solve two pointers problems. Inspired by Codeforces EDU lesson
3   https://codeforces.com/edu/course/2/lesson/9
4 */
5
6 struct twopointer {
7
8   struct state {
9
10    state() {} // init state for empty interval
11
12    void add(int x) {
13      // include code for adding x into current window
14    }
15
16    void rem(ll x) {
17      // include code for removing x from current window
18    }
19
20    // if the state is considered good
21    bool good() {
22      // include code to say if the state is good
23    }
24  };
25
26  vector<int> vec;
27
28  twopointer(vector<int> vec = {}) {
29    this->vec = vec;
30  }
31
32  // solve the two pointers problem where subinterval is good if interval is good (here, empty
      intervals should always be good)
33  // at each step, finds biggest interval for fixed r
34  int solve_sub() {
35    int l = 0;
36
37    state st;
38    for(int r = 0; r < vec.size(); ++r) {
39      st.add(vec[r]);
40
41      while(!st.good()) {
42        st.rem(vec[l++]);
43      }
44
45      // calculation step
46      // [l, r] contains the biggest good interval ending at r
47      ans += r - l + 1;
```

```
    }
    return ans;
  }

  // solve the two pointers problem where super interval is good if interval is good (here, empty intervals should always be BAD!, otherwise, by definition, all intervals should be good)
  // at each step, finds smallest interval for fixed r
  int solve_sup(ll k) {
    int l = 0;

    ll ans = 0;

    state st;
    for(int r = 0; r < vec.size(); ++r) {
      st.add(vec[r]);

      while(st.good()) {
        st.rem(vec[l++]);
      }

      // do calculation step

      // here, [l, r] is the first not good interval. [l-1, r] should have the smallest good interval
      // assume that if l == 0, there was no good interval yet
      ans += l;
    }
    return ans;
  }
};
```

4.18 WaveletTreeToggle

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

/** Wavelet Tree data structure. Input array is 0 based */
struct waveletTree {
  int lo, hi, mi; // minimum, maximum and (lo+hi)/2 element on array
  waveletTree *L, *R; // children

  vector<int> mp; // map how many elements went left
  struct Bit { // count how many indices are active
    vector<int> bt;
    Bit() {}

    inline void init(int n) {
      bt.assign(n, 0);
      for (int i = 1; i < n; i++) update(i, 1);
    }

    inline void update(int i, int x) {
      for (; i < (int) bt.size(); i += i&-i)
        bt[i] += x;
    }

    inline int query(int i) {
      int r = 0;
      for (; i > 0; i -= i&-i)
        r += bt[i];
      return r;
    }
    inline int query(int i, int j) {
      return query(j) - query(i - 1);
    }
  } bt;

#define mapLeft(i) (mp[i]) /* original mapLeft */
#define mapRight(i) (i - mapLeft(i)) /* original mapRight */

  /** *beg points to the first element, *end points to after the last element (just like stl default functions).
   * Elements are in range [lo..hi], inclusive */
  waveletTree(int *beg, int *end, int lo, int hi) { // O(nlogA)
    L = R = NULL;
    this->lo = lo;
    this->hi = hi;
    this->mi = (lo + hi) / 2;

```

```
    bt.init(end - beg + 1);

    mp.reserve(end - beg + 1);
    mp.push_back(0);
    for (auto it = beg; it != end; it++)
      mp.push_back(mp.back() + ((*it) <= mi));

    if (lo != hi) {
      auto pivot = stable_partition(beg, end, [&](int x) {
        return x <= mi;
      }); // split the vector
      L = new waveletTree(beg, pivot, lo, mi);
      R = new waveletTree(pivot, end, mi + 1, hi);
    }
  }

  /** K-th smallest element on range[l..r] */
  int kthSmallest(int k, int l, int r) {
    if (l > r) return -1; // out of bounds
    if (lo == hi) return lo; // leaf node

    int inLeft = mapLeft(r) - mapLeft(l-1);
    if (k <= inLeft) return L->kthSmallest(k, mapLeft(l-1)+1, mapLeft(r));
    else return R->kthSmallest(k-inLeft, mapRight(l-1)+1, mapRight(r));
  }

  /** Frequency of elements between [x..y] in array[l..r] */
  int rangeCount(int x, int y, int l, int r) {
    if (l > r || lo > y || hi < x) return 0; // out of bounds
    if (lo >= x && hi <= y) return bt.query(l, r);
    if (mi >= y) return L->rangeCount(x, y, mapLeft(l-1)+1, mapLeft(r));
    else if (mi <= x) return R->rangeCount(x, y, mapRight(l-1)+1, mapRight(r));
    else return L->rangeCount(x, mi, mapLeft(l-1)+1, mapLeft(r)) +
          R->rangeCount(mi+1, y, mapRight(l-1)+1, mapRight(r));
  }

  /** Toggle i-th active state (switch ON/OFF) */
  void toggle(int i) {
    if (bt.query(i, i)) bt.update(i, -1);
    else bt.update(i, 1);
    if (lo != hi) {
      if (mapLeft(i) == mapLeft(i-1) + 1) L->toggle(mapLeft(i));
      else R->toggle(mapRight(i));
    }
  }

  ~waveletTree() {
    if (L) delete L;
    if (R) delete R;
  }

#undef mapLeft
#undef mapRight
};

const int MAXN = 1e5 + 5;
int a[MAXN];

// https://www.spoj.com/problems/ILKQUERY2/
int main() {
  int n, q; scanf("%d%d", &n, &q);

  vector<int> all;
  for (int i = 1; i <= n; i++) {
    scanf("%d", a + i);
    all.push_back(a[i]);
  }
  sort(all.begin(), all.end());
  all.erase(unique(all.begin(), all.end()), all.end());
  for (int i = 1; i <= n; i++)
    a[i] = lower_bound(all.begin(), all.end(), a[i]) - all.begin();

  waveletTree T(a+1, a+n+1, 0, all.size());
  while(q--) {
    int op; scanf("%d", &op);
    if (op == 0) {
      int i, l, k; scanf("%d%d%d", &i, &l, &k);
      if (binary_search(all.begin(), all.end(), k)) {
        k = lower_bound(all.begin(), all.end(), k) - all.begin();
```

```
125         printf("%d\n", T.rangeCount(k, k, i + 1, l + 1));
126       }
127       else puts("0");
128     } else {
129       int i; scanf("%d", &i);
130       T.toggle(i + 1);
131     }
132   }
133
134   return 0;
135 }
```

4.19 MinQueue

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

template<typename T, typename Cmp = less<T>>
struct MinQueue {
    constexpr static auto cmp = Cmp{};

    deque<pair<T, int>> q;
    int l = 0, r = 0;

    auto push(T const& val) -> void {
        while (!q.empty() and !cmp(q.back().first,val)) {
            q.pop_back();
        }
        q.emplace_back(val, r++);
    }

    auto pop() -> void {
        if (!q.empty() and q.front().second == l) {
            q.pop_front();
        }
        l++;
    }

    auto get() -> T {
        return q.front().first;
    }

    auto empty() -> bool {
        return q.empty();
    }

    auto size() -> int {
        return r-l;
    }
};

// ------------------- Another implementation, using two stacks (bigger code, but more powerful)
    ---------------------- //

template<class T, T F(T, T) = std::min<T>>
struct minstack {
  stack<pair<T, T> > s;

  void push(T x) {
    if (!s.size()) s.push({x, x});
    else s.push({x, F(s.top().second, x)});
  }
  T top() { return s.top().first; }
  T pop() {
    T ans = s.top().first;
    s.pop();
    return ans;
  }
  int size() { return s.size(); }
  T min() { return s.top().second; }
};

template<class T, T F(T, T) = std::min<T>>
struct minqueue {
  minstack<T, F> s1, s2;

  void push(T x) { s1.push(x); }
  void move() {
    if (s2.size()) return;
```

```
    while (s1.size()) {
      T x = s1.pop();
      s2.push(x);
    }
  }
  T front() { return move(), s2.top(); }
  T pop() { return move(), s2.pop(); }
  int size() { return s1.size()+s2.size(); }
  T min() {
    if (!s1.size()) return s2.min();
    else if (!s2.size()) return s1.min();
    return F(s1.min(), s2.min());
  }
};
```

5 Tree

5.1 LCA

```
/** Tree struct with LCA (binary lifting) implemented. EDGE struct can be changed, according to the
    problem. */
const int LOG = 20;
struct Tree {
  struct Edge { int to; }; // you can also add some weight here, etc
  int n;
  vector<int> level;
  vector<vector<int>> par;
  vector<vector<Edge>> edges;
  Tree(int n) : n(n), edges(n) {}

  // add edge from->to. 0-based please.
  void add_edge(int from, int to) {
    edges[from].push_back(Edge{to});
    edges[to].push_back(Edge{from});
  }

  // Initialize in O(n logn)
  inline void init_lca(int root) {
    par.resize(n, vector<int>(LOG));
    level.resize(n);

    auto dfs_lca = [&](auto&& dfs_lca, int u, int p)->void {
      par[u][0] = p;
      for (Edge const& ed : edges[u]) if (ed.to != p) {
        level[ed.to] = level[u] + 1;
        dfs_lca(dfs_lca, ed.to, u);
      }
    };

    dfs_lca(dfs_lca, root, root);
    for (int b = 1; b < LOG; b++) {
      for (int i = 0; i < n; i++) {
        par[i][b] = par[par[i][b-1]][b-1];
      }
    }
  }

  // Finds LCA in O(logn)
  int lca(int u, int v) {
    if (level[u] < level[v]) swap(u, v);
    for (int b = LOG-1; b >= 0; b--) if (level[u] - (1 << b) >= level[v])
      u = par[u][b];
    if (u == v) return u;
    for (int b = LOG-1; b >= 0; b--) if (par[u][b] != par[v][b]) {
      u = par[u][b];
      v = par[v][b];
    }
    return par[u][0];
  }
};
```

5.2 Centroid

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

int const maxn = 101010;

```

```cpp
vector<int> g[maxn];
int cvis[maxn];

namespace Centroid {
    int sz[maxn];

    int dfs_sz(int x, int p = -1) {
        sz[x] = 1;
        for (int u : g[x]) {
            if (u == p or cvis[u]) continue;
            sz[x] += dfs_sz(u, x);
        }
        return sz[x];
    }

    int dfs_cent(int tot, int x, int p = -1) {
        for (int u : g[x]) {
            if (u == p or cvis[u]) continue;
            if (sz[u] > tot/2) return dfs_cent(tot, u, x);
        }
        return x;
    }

    int centroid(int x) {
        return dfs_cent(dfs_sz(x), x);
    }

} // end namespace Centroid

// example problem
// find number of simple paths in tree with lenght in range [l, r]

struct BIT {
    int n;
    vector<int> b;

    BIT(int n_) : n(n_+1), b(n) {};

    int get(int p) {
        int r = 0;
        for (int i = p+1; i > 0; i -= i&-i) {
            r += b[i];
        }
        return r;
    }

    int get(int a, int b) {
        return get(b) - get(a-1);
    }

    void add(int p, int v) {
        for (int i = p+1; i < n; i += i&-i) {
            b[i] += v;
        }
    }
};

BIT bit(maxn);

void dfs_add(int x, int p = -1, int d = 1, int to_add = 1) {
    bit.add(d, to_add);
    for (int u : g[x]) {
        if (u == p or cvis[u]) continue;
        dfs_add(u, x, d+1, to_add);
    }
}

long long dfs_get(int l, int r, int x, int p = -1, int d = 1) {
    if (d > r) return 0;
    long long ans = bit.get(max(0, l - d),  r - d);
    for (int u : g[x]) {
        if (u == p or cvis[u]) continue;
        ans += dfs_get(l, r, u, x, d + 1);
    }
    return ans;
}

long long solve(int l, int r, int x) {
    x = Centroid::centroid(x);
```

```
    cvis[x] = 1;

    long long ans = 0;

    bit.add(0, 1);
    for (int u : g[x]) {
        if (cvis[u]) continue;
        ans += dfs_get(l, r, u);
        dfs_add(u);
    }
    dfs_add(x, -1, 0, -1);

    for (int u : g[x]) {
        if (cvis[u]) continue;

        ans += solve(l, r, u);
    }

    return ans;
}

int main () {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    int n, l, r;
    cin >> n >> l >> r;

    for (int i = 0; i < n-1; i++) {
        int a, b;
        cin >> a >> b;

        g[a].push_back(b);
        g[b].push_back(a);
    }

    cout << solve(l, r, 1) << "\n";;
}
```

5.3 Isomorphism

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

#define fi first
#define se second

using namespace std;

const int N = 1e5 + 7;

vector<int> e[N]; // tree

namespace TreeIsomorphism {
  int lev[N], pai[N];

  map<vector<int>, int> mp;
  int cur;

  void dfs(int u, int p = -1, int l = 0) {
    pai[u] = p;
    lev[u] = l;
    for(int v : e[u]) if(v != p) dfs(v, u, l + 1);
  }

  // find centers of tree rooted in r
  ii find_centers(int n) {
    int rr;
    for(int k = 0; k < 2; ++k) {
      int r = k? rr : 1;

      dfs(r);

      rr = 1;
      for(int i = 1; i <= n; ++i)
        if(lev[i] > lev[rr]) rr = i;

      if(!k) r = rr;
    }

    vector<int> aux;
    for(; rr != -1; rr = pai[rr]) aux.pb(rr);
```

```
    int sz = aux.size();
    if(sz%2) return ii(aux[sz/2], 0);
    return ii(aux[sz/2], aux[sz/2 - 1]);
  }

  int dfs_hash(int u, int p = -1) {
    vector<ll> vec;

    for(auto v : e[u]) {
      if(v != p) {
        int h = dfs_hash(v);
        vec.eb(h);
      }
    }

    sort(vec.begin(), vec.end());
    if(mp[vec] == 0) mp[vec] = ++cur;
    return mp[vec];
  }

  int rooted_unlabeled_tree_hash(int n, int r) {
    return dfs_hash(r);
  }

  ii unrooted_unlabeled_tree_hash(int n) {
    ii centers = find_centers(n, 1);

    int r1 = rooted_unlabeled_tree_hash(n, centers.fi);
    int r2 = rooted_unlabeled_tree_hash(n, centers.se);

    return {r1, r2};
  }
}

int main() {
  ios::sync_with_stdio(0); cin.tie(0);
}
```

6 Misc

6.1 CoordinateCompressor

```
/*
  Easy-to-use util DS for coordinate compression of any tyoe
  Just initialize it with a vector, or call 'add()', and you can access
  compressed values with [] operator.
*/

template<typename T>
struct CoordCompressor {
  vector<T> vec;

  CoordCompressor(): sorted(false) {}
  CoordCompressor(const vector<T>& v): sorted(false) {
    add(v);
  }
  template <typename ...Ts>
  void add(const Ts&... args) {
    if(sorted_) throw runtime_error("Can't add elements after accessing them");
    (vec.push_back(args), ...);
  }
  void add(const vector<T>& v) {
    if(sorted_) throw runtime_error("Can't add elements after accessing them");
    for(auto x : v)
      vec.push_back(x);
  }

  // Returns order of x in O(logN)
  int operator[](const T& x) {
    if(!sorted_) {
      sorted_ = true;
      sort(vec.begin(), vec.end());
    }
    return lower_bound(vec.begin(), vec.end(), x) - vec.begin();
  }

private:
```

```
  bool sorted_;
};

// usage:
// CoordCompressor<ll> comp;
// comp.add({10, 1344, 56});
// comp.add(123, 42);
// int idx = comp[123] // returns 3
```

6.2 OrderedSet

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename tA, typename tB=null_type> using ord_set = tree<tA, tB, std::less<tA>,
    rb_tree_tag, tree_order_statistics_node_update>;
/** Instructions:
 * If you want to use it like a set: ord_set<int> st;
 * If you want to use it like a map: ord_set<int, int> mp;
 * ====== Functions ======
 * insert, erase, lower_bound, upper_bound, find, just like normal set
 * s.order_of_key(k); -----> qtt of strictly smaller than k, O(logN)
 * s.find_by_order(k); ----> iterator to k-th element counting from 0, in O(logN)
*/
```

6.3 Pragma

```
// https://codeforces.com/blog/entry/96344

#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

/*
  - O3: Auto-vectorize the code if the mentioned architectures allow it. This can make your code
    much faster by using SIMD
    (single instruction, multiple data) which kinda parallelizes your code on an instruction level.
    More info below.
    Function inlining     inlines functions aggressively if possible (and no, marking functions as
    inline doesn't inline functions,
    nor does it give hints to the compiler)
    Unrolls loops more aggressively than O2 (this might lead to instruction cache misses if
    generated code size is too large)

  - unroll-loops: Enables aggressive loop unrolling, which reduces the number of branches and
    optimizes parallel computation,
    but might increase code size too much and lead to instruction cache misses.

  - avx2: Instruction set that provide 8, 16 and 32 byte vector instructions
  (i.e., you can do some kinds of operations on pairs of 8 aligned integers at the same time).

  - lzcont: constant time __builtin_clz

  - popcnt: constant time __builtin_popcount

  - bmi, bmi2: These are bit manipulation instruction sets.
  They provide even more bitwise operations like ctz, blsi, and pdep
*/
```

6.4 Date

```
struct _Date {
  vector<int> mth, mth_sum;
  _Date() {
    mth = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    mth_sum = mth;
    partial_sum(mth.begin(), mth.end(), mth_sum.begin());
  }
};

struct Date: _Date {
  int d, m, y;

  int mdays() { return mth[m] + (m == 2) * leap(); }
  int ydays() { return 365 + leap(); }

  int msum()  { return mth_sum[m-1] + (m > 2) * leap(); }
  int ysum()  { return 365*(y-1) + (y-1)/4 - (y-1)/100 + (y-1)/400; }

  int count() { return (d-1) + msum() + ysum(); }
```

```
20
21 public:
22   Date() : d(1), m(1), y(1) {}
23   Date(int d, int m, int y) : d(d), m(m), y(y) {}
24   Date(int days) : d(1), m(1), y(1) { advance(days); }
25
26   // se o ano eh bissexto
27   bool leap() { return (y%4 == 0 and y%100) or (y%400 == 0); }
28
29   int weekDay() {
30     int x = y - (m<3);
31     return (x + x/4 - x/100 + x/400 + mth_sum[m-1] + d + 6)%7;
32   }
33
34   void advance(int days) {
35     days += count();
36     d = m = 1, y = 1 + days/366;
37     days -= count();
38     while(days >= ydays()) days -= ydays(), y++;
39     while(days >= mdays()) days -= mdays(), m++;
40     d += days;
41   }
42 };
```

6.5 BufferedWrite

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct WriteCache {
5   static constexpr int buf_size = 1<<12;
6   static constexpr int size_mask = buf_size - 1;
7
8   int last = 0;
9   char buf[buf_size];
10
11  void flush() {
12    fwrite(buf, 1, last, stdout);
13    last = 0;
14  }
15
16    template<typename T, enable_if_t<is_same_v<T, char>, bool> = true>
17  void print(T c) {
18    // DBG((int)c);
19    buf[last++] = c;
20    if (last == buf_size) flush();
21  }
22
23  template<typename T, enable_if_t<is_convertible_v<T, string_view>, bool> = true>
24  void print(T const& s_raw) {
25    auto s = string_view(s_raw);
26    int len = s.size();
27
28    if (last + len >= buf_size) {
29      int old_last = last;
30      int to_end = buf_size - last;
31      copy(s.begin(), s.begin() + to_end, buf + last);
32      last = buf_size;
33      flush();
34      int i = to_end;
35      for (; i + buf_size < len; i += buf_size) {
36        copy(s.begin() + i, s.begin() + i + buf_size, buf);
37        last = buf_size;
38        flush();
39      }
40      copy(s.begin() + i, s.end(), buf);
41      last = (old_last + len) & size_mask;
42    }
43    else {
44      copy(s.begin(), s.end(), buf + last);
45      last += len;
46    }
47  }
48
49  template<typename T, enable_if_t<is_integral_v<T> && !is_same_v<T, char>, bool> = true>
50  void print(T x) {
51    static char int_buf[50];
52    auto* beg = end(int_buf);
53
```

```
    bool neg = x < 0;
    if (neg) x = -x;

    do {
      --beg;
      *beg = x%10 + '0';
      x /= 10;
    } while (x);

    if (neg) {
      --beg;
      *beg = '-';
    }
    auto sv = string_view(beg, end(int_buf) - beg);
    print(sv);
  }

  template<typename T, typename... Args>
  void operator()(T const& a, Args const& ...args) {
    print<T>(a);
    if constexpr (sizeof...(args) >= 1) {
      (*this)(args...);
    }
  }

  ~WriteCache() {
    if (last) flush();
  }
} print;


// usage
// print("asdiuahsi ", 3, ' ', __int128(5), " ", string("aa"), '\n');
```

7 Graphs

7.1 Kruskal

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

namespace Kruskal {
  struct UF {
    vector<int> par;
    UF(int n) {
      par.resize(n);
      iota(par.begin(), par.end(), 0);
    }
    int find(int x) {
      return par[x] == x ? x : par[x] = find(par[x]);
    }
    void merge(int u, int v) {
      u = find(u), v = find(v);
      par[u] = v;
    }
  };

  struct Edge {
    int u, v;
    long long w;
    Edge() {}
    Edge(int u, int v, long long w) : u(u), v(v), w(w) {}
    bool operator<(Edge const& e) const { return w < e.w; }
  };

  vector<Edge> kruskal(int n, vector<Edge>& edg) {
    sort(edg.begin(), edg.end());
    UF uf(n + 1);
    vector<Edge> ans;
    for (Edge const& e : edg) {
      if (uf.find(e.u) != uf.find(e.v)) {
        uf.merge(e.u, e.v);
        ans.push_back(e);
      }
    }
    return ans;
  }
}
```

```
41
42 int main() {
43   using namespace Kruskal;
44 }
```

7.2 EulerCycle

```
1  /*
2    Checa se existe e encontra um caminho euleriano num grafo dado.
3    Works for euler cycle and euler path. If you want euler cycle, change line 36
4  */
5
6  #include <bits/stdc++.h>
7  #define pb push_back
8
9  using namespace std;
10
11 const int N = 100007;
12
13 int deg[N][2], un[N], sz[N];
14 stack<int> edges[N];
15
16 int find(int u) {
17   return un[u] == u? u : un[u] = find(un[u]);
18 }
19
20 void unite(int u, int v) {
21   u = find(u); v = find(v);
22   if(u == v) return;
23   if(sz[u] < sz[v]) swap(u, v);
24   sz[u] += sz[v];
25   un[v] = u;
26 }
27
28 // Retorna 0 se o grafo estiver invalido e o inicio do caminho se estiver correto
29 int check(int n) {
30   int cnt = 0, u;
31   for(int i = 1; i <= n; ++i) {
32     if(abs(deg[i][0] - deg[i][1]) > 1) return 0;
33     cnt += deg[i][0] - deg[i][1] != 0;
34     if(deg[i][1] - deg[i][0] == 1) u = i;
35   }
36   if(cnt == 2 and sz[find(1)] == n) return u; // if euler cycle, change to 'cnt == 0'
37   else return 0;
38 }
39
40 // DFS for euler tour
41 vector<int> path;
42 void dfs(int u) {
43   int v;
44   while(edges[u].size()) {
45     v = edges[u].top();
46     edges[u].pop();
47     dfs(v);
48   }
49   path.pb(u);
50 }
51
52 int main() {
53   int n, m, u, v;
54   cin >> n >> m;
55
56   for(int i = 1; i <= n; ++i) {
57     un[i] = i;
58     sz[i] = 1;
59   }
60
61   for(int i = 0; i < m; ++i) {
62     cin >> u >> v;
63     edges[u].push(v);
64     deg[u][1]++;
65     deg[v][0]++;
66     unite(u, v);
67   }
68
69   u = check(n);
70   if(u == 0) cout << "NO" << endl;
71   else {
72     cout << "YES" << endl;
```

```
    dfs(u);
    reverse(path.begin(), path.end());
    for(int i = 0; i < path.size(); ++i) {
      cout << path[i] << " ";
    }
    cout << endl;
  }
}
```

7.3 DynamicConnectivity

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

using ii = pair<int, int>;

struct Dsu {
    int n;
    vector<int> pai, w;
    vector<ii> rb;

    Dsu(int n_) : n(n_), pai(n+1), w(n+1, 1) {
        iota(pai.begin(), pai.end(), 0);
    }

    int find(int x) {
        if (pai[x] == x) return x;
        return find(pai[x]);
    }

    void join(int a, int b) {
        a = find(a), b = find(b);

        if (a != b) {
            if (w[a] < w[b]) swap(a, b);
            rb.emplace_back(b, pai[b]);
            rb.emplace_back(a, w[a]);

            pai[b] = a;
            w[a] += w[b];
            n--;
        }
        else {
            rb.emplace_back(0, 0);
        }
    }

    void rollback() {
        if (rb.back() == ii{0, 0}) {
            rb.pop_back();
            return;
        }

        auto f = [this] (auto& v) {
            auto [pos, val] = this->rb.back();
            this->rb.pop_back();
            v[pos] = val;
        };

        f(w);
        f(pai);
        n++;
    }
};

struct DynConnectivity {
    int n, k;
    vector<vector<ii>> tree;
    Dsu dsu;

    DynConnectivity(int n_, int k_) : n(n_), k(k_), tree((k+1)*4), dsu(n) {};

    void add(int no, int l, int r, int a, int b, ii const& u) {
        if (a <= l and r <= b) {
            tree[no].push_back(u);
            return;
        }
        if (l == r) return;

```

```
        int m = (l+r)/2;

        if (a <= m) add(no*2, l, m, a, b, u);
        if (b > m) add(no*2+1, m+1, r, a, b, u);
    }

    // marks the existence of the edge u between the times a and b
    void add(int a, int b, ii const& u) {
        add(1, 0, k, a, b, u);
    }

    void solve(vector<int>& res, int no, int l, int r) {
        for (auto& [a, b] : tree[no]) {
            dsu.join(a, b);
        }

        if (l == r) {
            res[l] = dsu.n;
        }
        else {
            int m = (l+r)/2;
            solve(res, no*2, l, m);
            solve(res, no*2+1, m+1, r);
        }

        for (int i = 0; i < (int)tree[no].size(); i++) dsu.rollback();
    }

    vector<int> solve() {
        vector<int> v(k+1);
        solve(v, 1, 0, k);
        return v;
    }
};

// example solution
// solution for the problem - Dynamic Connectivity
// https://cses.fi/problemset/task/2133/
// given a graph with n vertex and m edges originally and k toggle edge
// operations, find the number of components before the first and
// after each operation

int main() {
  ios_base::sync_with_stdio(false), cin.tie(nullptr);

    int n, m, k;
    cin >> n >> m >> k;

    DynConnectivity dyn(n, k);

    set<ii> g;
    map<ii, int> gt;

    while (m--) {
        int x, y;
        cin >> x >> y;

        if (x > y) swap(x, y);

        gt[{x, y}] = 0;
    }

    // for every edge finds the times of start and end of the edge
    for (int i = 1; i <= k; i++) {
        int op, x, y;
        cin >> op >> x >> y;

        if (x > y) swap(x, y);

        if (op == 1) {
            gt[{x, y}] = i;
        }
        else {
            auto it = gt.find({x, y});
            int ini = it->second;
            gt.erase(it);

            dyn.add(ini, i-1, {x, y});
        }
```

```
    }

    // the edges that exist afer everything is done
    for (auto [p, key] : gt) {
        dyn.add(key, k, p);
    }

    auto res = dyn.solve();

    for (int i = 0; i <= k; i++) {
        cout << res[i] << " \n"[i==k];
    }
}
```

7.4 Kosaraju

```
#include "bits/stdc++.h"

#define pb push_back

using namespace std;

namespace Kosaraju {
  #define ind(x) (x-1)

  vector<vector<int>> g, rev;
  vector<int> col, topo;

  void dfs_topo(int u) {
    col[ind(u)] = 1;
    for(int v : g[ind(u)]) {
      if(!col[ind(v)]) dfs_topo(v);
    }
    topo.pb(u);
  }

  void dfs_scc(int u, int c) {
    col[ind(u)] = c;
    for(int v : rev[ind(u)]) {
      if(!col[ind(v)]) dfs_scc(v, c);
    }
  }

  // receives a graph, where g_[u-1] contains nodes adjacent to node u
  // and returns the colors (ids of SCCs of each node of the input graph)
  // Also, optionally returns the edges of the new graph.
  // WARNING: Output graph may have MULTIPLE EDGES!!!!
  vector<int> solve(const vector<vector<int>> & g_, vector<vector<int>> * out = nullptr) {
    g = g_;
    int n = g.size();

    rev.assign(n, vector<int>());
    col.assign(n, 0);

    // calc reverse edges
    for(int i = 1; i <= n; ++i) {
      for(int v : g[ind(i)]) {
        rev[ind(v)].pb(i);
      }
    }

    for(int i = 1; i <= n; ++i) {
      if(!col[ind(i)]) dfs_topo(i);
    }
    fill(col.begin(), col.end(), 0);

    assert(topo.size() == n);
    reverse(topo.begin(), topo.end());
    int c = 0;
    for(int u : topo) {
      if(!col[ind(u)]) dfs_scc(u, ++c);
    }

    if(out) {
      vector<vector<int>> ret(c);
      for(int i = 1; i <= n; ++i) {
        for(int v : g[ind(i)]) {

          if(col[ind(i)] != col[ind(v)]) {
```

```
            // cout << col[ind(i)] << endl;
            // cout << col[ind(v)] << endl;
            ret[ind(col[ind(i)])].pb(col[ind(v)]);
          }
        }
      }
      *out = ret;
    }

    return col;
  }
};
```

7.5 BlockCut

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

using ii = pair<int, int>;
using Gr = vector<vector<int>>;

/*
    Calculates cut edges (bridges), cut nodes (articulation points),
    bicconnected components and builds block-cut tree, all in O(n + m)

    Block cut tree properties:
    => all cut nodes will go to internal nodes; equivalently:
        - cut node cant be leaf, all leaves are blocks
        - cut node has degree at least 2
    => internal block nodes can have 0 nodes
    => degree of cut node in BCT is less or equal to degree of correspondent node in original graph
*/

struct TarjanMisc {
    vector<int> num, low;
    int at = 1;

    // you can also store edge ids
    // important if you have parallel edges
    vector<ii> edges;
    vector<int> art; // art[u] = 1 if u is articulation point
    stack<int> stk; // aux stack for biconnected components calculation
    vector<vector<int>> comps; // the biconnected components

    void dfs(Gr const& g, int u, int p = -1) {
        num[u] = low[u] = at++;
        int nchild = 0;
        stk.push(u);

        for (int v : g[u]) {
            // if graph has parallel edged you should assing an id to every edge
            // and in the next line check if the edge id is the same as the parent edge
            if (v == p) continue;
            if (num[v] != 0) {
                low[u] = min(low[u], num[v]);
                continue;
            }
            dfs(g, v, u);
            low[u] = min(low[u], low[v]);
            nchild++;

            if (low[v] > num[u]) {
                edges.emplace_back(u, v);
            }
            if (low[v] >= num[u]) {
                if(p != -1) {
                    art[u] = 1;
                }

                comps.push_back({u});
                while (comps.back().back() != v)
                    comps.back().push_back(stk.top()), stk.pop();
            }
        }

        if (p == -1 and nchild > 1) art[u] = 1;
    }

    // 0 indexed please
```

```
    TarjanMisc(Gr const& g) : num(g.size()), low(g.size()), art(g.size()), stk(), comps() {
        int n = g.size();
        for (int i = 0; i < n; i++) {
            if (num[i] == 0) {
                dfs(g, i);
            }
        }
    }

    vector<int> get_articulations() {
        int n = num.size();
        vector<int> ans;
        for (int i = 0; i < n; i++) {
            if (art[i]) {
                ans.push_back(i);
            }
        }
        return ans;
    }

    vector<ii> get_bridges() {
        return edges;
    }

    vector<vector<int>> get_biconnected_components() {
        return comps;
    }

    // to[u] will store to which new node u will go in the tree.
    // In case u is articulation point, it will go to its dedicated node
    Gr get_block_cut_tree(vector<int> * to = nullptr) {
        Gr tree;
        int n = num.size();
        vector<int> id(n);

        int cur = 0;
        auto new_node = [&]() {
            tree.emplace_back();
            return cur++;
        };

        for (int u = 0; u < n; ++u)
            if (art[u]) id[u] = new_node();

        for (auto &comp : comps) {
            int node = new_node();
            for (int u : comp)
                if (!art[u]) id[u] = node;
                else {
                    tree[node].push_back(id[u]);
                    tree[id[u]].push_back(node);
                }
        }

        if(to) {
            (*to) = id;
        }

        return tree;
    }
};
```

7.6 DominatorTree

```
/*
 *  Calculates dominator tree in O(M * log(N))
 *  A dominator of a node u is a node v such that all paths from source to u goes trough v. In other words,
 *  if you take v off the graph, s and u become disconnected.
 *  A immeadiate dominator idom(u) := the dominator of u closest to u. In the s-DFS tree, it would be the lowest dominator of u
 *  The dominator tree of the graph is the tree were s is the root and edges are idom(u) -> u
 *
 *  This can also be calculated in O(N * M) more easily with N DFSs, where in i-th DFS you remove node i from the graph
 *
 *  In the end, parent of u in the dominator tree will be pred(u) = inv_tim[idom[tim[u]]], unless u is the source, then it will be itself.
 *  If u is not reachable by the source, pred(u) will be 0
```

```
 *
 *  BE CAREFUL if in your problems there can be nodes not reachable by the source
 *  BE CAREFUL that label of nodes are changed to the visit time of nodes for every array except "e"
     and "rev"
 */

#include <bits/stdc++.h>

#define pb push_back
#define fi first
#define se second

using namespace std;
typedef pair<int, int> ii;

namespace DominatorTree {
  const int N = 100007;

  vector<int> e[N], rev[N]; // graph, reverse graph
  vector<int> sd[N]; // semidominated, sd[u] = {v | sdom[v] = u}

  int tim[N], inv_tim[N], par[N]; // visit time, inverse function of visit time, parent in the DFS
    tree
  int sdom[N], idom[N]; // semidominator, immediate dominator
  int un[N], path[N]; // DSU stuff for path compression min query

  // Finds the guy that has least sdom in the ancestors of u, and uses path compression to optimize
    it
  ii query(int u) {
    if(u == un[u]) return ii(u, u);
    int p;
    tie(p, un[u]) = query(un[u]);
    if(sdom[p] < sdom[path[u]]) path[u] = p;
    return ii(path[u], un[u]);
  }

  int tt;
  void dfs(int u) {
    tim[u] = ++tt;
    inv_tim[tt] = u;
    for(int v : e[u]) {
      if(!tim[v]) {
        dfs(v);
        par[tim[v]] = tim[u];
      }
    }
  }
  void build() {
    for(int u = 1; u <= tt; ++u) sdom[u] = idom[u] = un[u] = path[u] = u;
    for(int u = tt; u >= 1; --u) {
      for(int v : rev[inv_tim[u]]) {
        v = tim[v];
        if(v == 0) continue;
        if(v < u) sdom[u] = min(sdom[u], sdom[v]);
        else sdom[u] = min(sdom[u], sdom[query(v).fi]);
      }
      sd[sdom[u]].pb(u);

      for(int v : sd[u]) {
        int best = query(v).fi;
        if(sdom[best] >= u) idom[v] = u;
        else idom[v] = best;
      }

      for(int v : e[inv_tim[u]]) {
        v = tim[v];
        if(v == 0) continue;
        if(par[v] == u) un[v] = u; // if u->v is tree edge, add it
      }
    }
    for(int u = 1; u <= tt; ++u)
      if(idom[u] != sdom[u]) idom[u] = idom[idom[u]];
  }

};

int main() {
  using namespace DominatorTree;

```

```
    // Reads n = number of vertices, m = number of edges and s = source vertice.
    int n, m, s; cin >> n >> m >> s;

    // Read the graph
    for(int i = 0; i < m; ++i) {
        int u, v; cin >> u >> v;
        e[u].pb(v);
        rev[v].pb(u); // Need to add reversed graph
    }

    dfs(s);
    build();

    for(int i = 1; i <= n; ++i) {
        cout << inv_tim[idom[tim[i]]] << ' ';
    }
    cout << endl;
}
```

7.7 TarjanSCC

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

struct TarjanSCC {
    using Gr = vector<vector<int>>;

    stack<int> st;
    vector<int> num, low, vis;
    vector<int> sccs;
    int idx = 1, n_sccs = 0;

    // 0 indexed please
    TarjanSCC(Gr const& g) : num(g.size()), low(g.size()), vis(g.size()), sccs(g.size()) {
        int n = g.size();
        for (int i = 0; i < n; i++) {
            if (num[i] == 0) dfs(g, i);
        }
    }

    void dfs(Gr const& g, int u) {
        num[u] = low[u] = idx++;
        st.push(u);
        vis[u] = 1;

        for (int v : g[u]) {
            if (num[v] == 0) {
                dfs(g, v);
                low[u] = min(low[u], low[v]);
            }
            else if (vis[v]) {
                low[u] = min(low[u], num[v]);
            }
        }

        if (low[u] == num[u]) {
            while (true) {
                int x = st.top();
                st.pop();
                vis[x] = 0;
                sccs[x] = n_sccs;
                if (x == u) break;
            }
            n_sccs++;
        }
    }

    // returns pair< vector v, where: v[x] = index of x's scc,
    //               int n, where : n = number of sccs >
    // the sccs come in reverse topological order
    // scc[0] < scc[1] < scc[2] < ...
    // scc[x] represent the set of vertex with scc index x
    pair<vector<int>, int> get_sccs() {
        return {sccs, n_sccs};
    }

    // returns the same as above + compressed graph of the components
    tuple<Gr, vector<int>, int> compressed_g(Gr const& gr) {
        auto gc = vector<vector<int>>(n_sccs);
```

```
        int n = gr.size();
        for (int u = 0; u < n; u++) {
            for (int v : gr[u]) {
                if (sccs[u] != sccs[v]) {
                    gc[sccs[u]].push_back(sccs[v]);
                }
            }
        }
        for (auto& l : gc) {
            sort(l.begin(), l.end());
            l.erase(unique(l.begin(), l.end()), l.end());
        }

        return {gc, sccs, n_sccs};
    }
};
```

7.8 BellmanFord

```
const ll INF = 1e18;
struct Edge { int u, v; ll w; };

// Bellman Ford algorithm, and check for negative edges. O(N * M).
bool bellman_ford(vector<Edge> const& edges, vector<ll>& d, int n) {
  d.assign(n, INF);
  d[0] = 0;

  for (int it = 0; it < n; it++) {
    for (Edge const& ed : edges) {
      int u = ed.u, v = ed.v; ll w = ed.w;
      if (d[u] < INF && d[v] > d[u] + w) {
        d[v] = max(-INF, d[u] + w);
      }
    }
  }

  // If it's possible to relax some edge, there is a negative cycle;
  // you can ignore the following code if it isn't necessary to deal with negative edges

  // negative[i] is true if there is a negative cycle in path 0..i
  vector<bool> negative(n);

  for (int it = 0; it < n; it++) {
    for (Edge const& ed : edges) {
      int u = ed.u, v = ed.v; ll w = ed.w;
      if (d[u] == INF) continue;

      if (d[v] > d[u] + w) {
        d[v] = d[u] + w;
        negative[v] = true;
      }

      if (negative[u]) negative[v] = true;
    }
  }

  // do something with negative edges here
  return negative[n - 1]; // in this case I return if there is a negative cycle from 0 to n - 1
}
```

7.9 Hungarian

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

// finds an answer for the assignment problem using
// the hungarian algorithm in O(n^2*m) [O(n^3)]
// everything is 1 indexed
// ans[0] is the cost of the minimum assignment
struct Hungarian {
    constexpr static int inf = numeric_limits<int>::max();

    Hungarian(vector<vector<int>> const& a) :
            n((int)a.size() - 1), m((int)a[0].size() - 1),
            u(n+1), v(m+1), p(m+1), way(m+1)
    {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
```

```
            vector<int> minv (m+1, inf);
            vector<char> used (m+1, false);

            do {
                used[j0] = true;
                int i0 = p[j0],  delta = inf,  j1;
                for (int j = 1; j <= m; j++) {
                    if (!used[j]) {
                        int cur = a[i0][j] - u[i0] - v[j];
                        if (cur < minv[j])
                            minv[j] = cur,  way[j] = j0;
                        if (minv[j] < delta)
                            delta = minv[j],  j1 = j;
                    }
                }
                for (int j = 0; j <= m; j++) {
                    if (used[j])
                        u[p[j]] += delta,  v[j] -= delta;
                    else
                        minv[j] -= delta;
                }
                j0 = j1;
            } while (p[j0] != 0);

            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
    }

    vector<int> get_ans() {
        vector<int> ans (n+1);
        for (int j=1; j <= m; j++) {
            ans[p[j]] = j;
        }
        ans[0] = -v[0];
        return ans;
    }

    int n, m;
    vector<int> u, v, p, way;
};
```

7.10 EdmondsMDST

```
/**
 * Minimum Directed Spanning Tree (Edmonds Algorithm). O(E logV)
 * Original Source: https://github.com/kth-competitive-programming/kactl/blob/master/content/graph/
    DirectedMST.h */
namespace MDST {
  struct RollbackUF {
    vector<int> e; vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return st.size(); }
    void rollback(int t) {
      for (int i = time(); i --> t;)
        e[st[i].first] = st[i].second;
      st.resize(t);
    }
    bool join(int a, int b) {
      a = find(a), b = find(b);
      if (a == b) return false;
      if (e[a] > e[b]) swap(a, b);
      st.push_back({a, e[a]});
      st.push_back({b, e[b]});
      e[a] += e[b]; e[b] = a;
      return true;
    }
  };
  struct Edge {
    int a, b; ll w;
    Edge() {}
    Edge(int a, int b, ll w) : a(a), b(b), w(w) {}

  };
```

```
  struct Node { /// lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
      key.w += delta;
      if (l) l->delta += delta;
      if (r) r->delta += delta;
      delta = 0;
    }
    Edge top() { prop(); return key; }
  };
  Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
  }
  void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

  pair<ll, vector<int>> dmst(int n, int r, vector<Edge> const& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, Edge{-1,-1, 0}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    for (int s = 0; s < n; s++) {
      int u = s, qi = 0, w;
      while (seen[u] < 0) {
        if (!heap[u]) return {-1,{}};
        Edge e = heap[u]->top();
        heap[u]->delta -= e.w, pop(heap[u]);
        Q[qi] = e, path[qi++] = u, seen[u] = s;
        res += e.w, u = uf.find(e.a);
        if (seen[u] == s) { /// found cycle, contract
          Node* cyc = 0;
          int end = qi, time = uf.time();
          do cyc = merge(cyc, heap[w = path[--qi]]);
          while (uf.join(u, w));
          u = uf.find(u), heap[u] = cyc, seen[u] = -1;
          cycs.push_front({u, time, {&Q[qi], &Q[end]}});
        }
      }
      for (int i = 0; i < qi; i++) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u,t,comp] : cycs) { // restore sol (optional)
      uf.rollback(t);
      Edge inEdge = in[u];
      for (auto& e : comp) in[uf.find(e.b)] = e;
      in[uf.find(inEdge.b)] = inEdge;
    }
    for (int i = 0; i < n; i++) par[i] = in[i].a;
    return {res, par};
  }
}
```

7.11 MCMF

```
/** Minimum Cost Maximum Flow using Johnson's Algo (potential function). O(V^2 * logV + V * E) */

template<typename T>
struct MCMF {
  T flow_inf = numeric_limits<T>::max() / 2;
  using pti = pair<T, int>;
  using pii = pair<int, int>;

  struct Edge {
    int to;
    T cost, cap, flow;
    int rid;

    Edge() : to(), cost(), cap(), flow(), rid() {}
    Edge(int to, T cost, T cap, T flow, int rid) :
        to(to), cost(cost), cap(cap), flow(flow), rid(rid) {}
```

```
17  };
18
19  vector<vector<Edge>> edges;
20  vector<T> h, dist;
21  vector<pii> par;
22
23  int n;
24  MCMF(int n) : n(n) {
25    edges.resize(n+1);
26    h.resize(n+1);
27    dist.resize(n+1);
28    par.resize(n+1);
29  }
30
31  void add_edge(int u, int v, T cost, T cap) {
32    edges[u].emplace_back(v, cost, cap, 0, (int) edges[v].size());
33    edges[v].emplace_back(u, -cost,  0, 0, (int) edges[u].size() - 1);
34  }
35
36  void bellman_ford(int s) {
37    fill(h.begin(), h.end(), flow_inf);
38    h[s] = 0;
39
40    for (int i = 0; i < n - 1; i++) {
41      for (int at = 0; at < n; at++) {
42        for (int j = 0; j < (int) edges[at].size(); j++) {
43          if (!edges[at][j].cap)  continue;
44          int next = edges[at][j].to;
45          T w = edges[at][j].cost;
46          h[next] = min(h[next], h[at] + w);
47        }
48      }
49    }
50  }
51
52  bool dijkstra(int s, int t) {
53    fill(dist.begin(), dist.end(), flow_inf);
54    fill(par.begin(), par.end(), mk(-1, -1));
55    priority_queue<pti, vector<pti>, greater<pti> > pq;
56
57    dist[s] = 0;
58    pq.emplace(0, s);
59    bool ret = false;
60
61    while (!pq.empty()) {
62      int at = pq.top().se;
63      T d = pq.top().fi;
64      pq.pop();
65
66      if (at == t) ret = true;
67      if (d != dist[at]) continue;
68
69      for (int i = 0; i < (int) edges[at].size(); i++) {
70        Edge edg = edges[at][i];
71        int next = edg.to;
72        if (edg.cap - edg.flow <= 0) continue;
73        T w = dist[at] + edg.cost + h[at] - h[next];
74        if (dist[next] > w) {
75          dist[next] = w;
76          par[next] = {at, i};
77          pq.emplace(dist[next], next);
78        }
79      }
80    }
81
82    for (int i = 0; i <= n; i++)
83      if (h[i] < flow_inf and dist[i] < flow_inf)
84        h[i] += dist[i];
85
86    return ret;
87  }
88
89  pair<T, T> flow(int s, int t) {
90    T cost = 0, flow = 0;
91    bellman_ford(s); // remover essa linha se o grafo nao tiver pesos negativos
92
93    while (dijkstra(s, t)) {
94      T f = flow_inf;
95      for (int at = t; at != s; at = par[at].fi) {
```

```
        Edge edg = edges[par[at].fi][par[at].se];
        f = min(f, edg.cap - edg.flow);
      }
      flow += f;
      for (int at = t; at != s; at = par[at].fi) {
        Edge &edg = edges[par[at].fi][par[at].se];
        edg.flow += f;
        edges[edg.to][edg.rid].flow -= f;
        cost += edg.cost * f;
      }
    }

    return mk(cost, flow);
  }
};
```

7.12 TarjanBridges

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

using ii = pair<int, int>;
using Gr = vector<vector<int>>;

struct TarjanBridges {
    vector<int> num, low;
    int at = 1;

    // you can also store edge ids
    // important if you have parallel edges
    vector<ii> edges;
    vector<int> art;

    void dfs(Gr const& g, int u, int p = -1) {
        num[u] = low[u] = at++;
        int nchild = 0;

        for (int v : g[u]) {
            // if graph has parallel edged you should assing an id to every edge
            // and in the next line check if the edge id is the same as the parent edge
            if (v == p) continue;
            if (num[v] != 0) {
                low[u] = min(low[u], num[v]);
                continue;
            }
            dfs(g, v, u);
            low[u] = min(low[u], low[v]);
            nchild++;

            if (low[v] > num[u]) {
                edges.emplace_back(u, v);
            }
            if (p != -1 and low[v] >= num[u]) {
                art[u] = 1;
            }
        }

        if (p == -1 and nchild > 1) art[u] = 1;
    }

    // 0 indexed please
    TarjanBridges(Gr const& g) : num(g.size()), low(g.size()), art(g.size()) {
        int n = g.size();
        for (int i = 0; i < n; i++) {
            if (num[i] == 0) {
                dfs(g, i);
            }
        }
    }

    vector<int> get_articulations() {
        int n = num.size();
        vector<int> ans;
        for (int i = 0; i < n; i++) {
            if (art[i]) {
                ans.push_back(i);
            }
        }
        return ans;
```

```
    }

    vector<ii> get_bridges() {
        return edges;
    }
};
```

7.13 TwoSat

```
struct TwoSat {
  int n; // amount of VERTICES (2 * variables)

  vector<vector<int>> edges, reved;

  // N is the amount of VERTICES!!! n = 2 * VARIABLES!!!
  TwoSat(int n) : n(n), edges(n), reved(n) {}

  // i is variable, return vertex id
  inline static int pos(int i) { return i + i; }
  inline static int neg(int i) { return i + i + 1; }
  inline static int getNot(int i) { return i ^ 1; }

  inline void addEdge(int u, int v) { // 0-based pls!
    edges[u].push_back(v);
    edges[getNot(v)].push_back(getNot(u));
    reved[v].push_back(u);
    reved[getNot(u)].push_back(getNot(v));
  }

  void dfsOrder(int u, vector<int>& vis, vector<int>& order) {
    vis[u] = 1;
    for (int v : edges[u]) if (!vis[v])
        dfsOrder(v, vis, order);
    order.pb(u);
  }

  void dfsSCC(int u, int c, vector<int>& comp) {
    comp[u] = c;
    for (int v : reved[u]) if (!comp[v])
        dfsSCC(v, c, comp);
  }

  /** verify satisfiability */
  inline bool isSat(vector<int>& states) {
    states.assign(n / 2, -1);
    vector<int> color(n);
    vector<int> order;
    for (int i = 0; i < n; i++) if (!color[i])
        dfsOrder(i, color, order);
    reverse(order.begin(), order.end());
    fill(color.begin(), color.end(), 0);
    for (int i = 0, c = 0; i < n; i++) if (!color[order[i]])
        dfsSCC(order[i], ++c, color);
    for (int i = 0; i < n; i += 2) {
      if (color[i] == color[i+1])
        return false;
    }
    for (int i = n - 1; i >= 0; i--) {
      if (states[order[i] / 2] == -1)
        states[order[i] / 2] = !(order[i]&1);
    }
    return true;
  }
};
```

7.14 Kuhn

```
/*
  Vertices sao one based.
  n = quantos vertices da primeira particao
  m = quantos vertices da segunda particao

  A contagem da primeira particao e da segunda sao independentes

  Qual vertice u conectou?
    resposta: match[u] - n
  Qual vertice v conectou?
    resposta: match[n + v]
*/
```

```cpp
struct Kuhn {
  int n, m;
  vector<int> vis, match;
  vector<vector<int>> e;

  Kuhn(){}
  Kuhn(int n, int m) {
    this->n = n;
    this->m = m;
    e.resize(n + 5);
    match.assign(n + m + 5, 0);
    vis.resize(n + m + 5);
  }

  void add_edge(int u, int v) {
    e[u].pb(n + v);
  }

  int dfs(int u) {
    if(vis[u]) return 0;
    vis[u] = 1;

    for(int v : e[u]) {
      if(match[v] == 0 or (match[v] and dfs(match[v]))) {
        match[u] = v;
        match[v] = u;
        return 1;
      }
    }
    return 0;
  }
  int solve() {
    int flag, tot = 0;

    do {
      flag = 0;
      fill(vis.begin(), vis.end(), 0);
      for(int u = 1; u <= n; ++u) {
        if(match[u] == 0 and dfs(u)) tot += (flag = 1);
      }

    } while(flag);
    return tot;
  }
};
```

7.15 NegativeCycles

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

using ll = long long;
using ii = pair<int, int>;
using Gr = vector<vector<ii>>;

// returns empty if there is no negative cycle in graph g
// else returns any negative cycle in g
// complexity O(V * E), but actually much faster in almost all cases
// if graph has no negative cycles expected time O(E)
optional<vector<int>> negative_cycle(Gr& g) {
    int n = g.size();
    g.emplace_back();
    for (int i = 0; i < n; i++) {
        g.back().emplace_back(i, 0);
    }
    n++;

    int src = n-1;

    vector<char> in_queue(n);
    vector<int> len(n), pre(n);
    vector<ll> d(n, numeric_limits<ll>::max() / 2);
    deque<int> q;

    q.push_back(src);
    in_queue[src] = 1;
    d[src] = 0;

    auto recover = [&] (int u) {
```

```
        vector<char> in_stack(n);
        stack<int> s;

        while (!in_stack[u]) {
            s.push(u);
            in_stack[u] = 1;
            u = pre[u];
        }

        vector<int> cyc = {u};
        while (s.top() != u) {
            cyc.push_back(s.top());
            s.pop();
        }

        return cyc;
    };

    while (!q.empty()) {
        int u = q.front();
        q.pop_front();

        in_queue[u] = 0;

        for (auto [v, w] : g[u]) {
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                pre[v] = u;
                len[v] = len[u] + 1;

                // cycle detected
                if (len[v] == n) {
                    return recover(v);
                }

                if (!in_queue[v]) {
                    if (!q.empty() and d[q.front()] >= d[v])
                        q.push_front(v);
                    else
                        q.push_back(v);

                    in_queue[v] = 1;
                }
            }
        }
    }

    return {};
}
```

7.16 DirectedMST

```
/** Source: https://github.com/kth-competitive-programming/kactl/blob/master/content/graph/
    DirectedMST.h */

/** Finds a minimum spanning tree/arborescence rooted at some node of a directed graph */

struct RollbackUF { // RollBack union find, O(logn)
  vector<int> e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool merge(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};

```

```cpp
struct Edge { int a, b; ll w; };

struct Node { /// lazy skew heap node
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vector<int>> dmst(int n, int r, vector<Edge> const& g) { // O(E logV)
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vector<int> seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, Edge{-1,-1, 0}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  for (int s = 0; s < n; s++) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) { /// found cycle, contract
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.merge(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    for (int i = 0; i < qi; i++) in[uf.find(Q[i].b)] = Q[i];
  }

  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  for (int i = 0; i < n; i++) par[i] = in[i].a;
  return {res, par};
}
```

7.17 Dinic

```cpp
struct Dinic {
  const long long flow_inf = 1e18;

  struct FlowEdge {
    int from, to;
    long long cap, flow = 0;
    FlowEdge(int from, int to, long long cap) : from(from), to(to), cap(cap) {}
  };

  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  vector<int> level, ptr;

  Dinic(int n) : adj(n), level(n), ptr(n) {}

```

```
  inline void add_edge(int from, int to, long long cap) {
    adj[from].push_back(edges.size());
    edges.emplace_back(from, to, cap);
    adj[to].push_back(edges.size());
    edges.emplace_back(to, from, 0);
  }

  inline bool bfs(int src, int snk) {
    fill(level.begin(), level.end(), -1);
    level[src] = 0;

    vector<int> q = {src};
    for (int i = 0; i < (int) q.size(); i++) {
      int u = q[i];
      for (int id : adj[u]) {
        if (edges[id].cap - edges[id].flow > 0 && level[edges[id].to] == -1) {
          level[edges[id].to] = level[u] + 1;
          q.push_back(edges[id].to);
        }
      }
    }

    return level[snk] != -1;
  }

  long long dfs(int u, int snk, long long pushed) {
    if (u == snk || pushed == 0) return pushed;

    for (int& cid = ptr[u]; cid < (int) adj[u].size(); cid++) {
      int id = adj[u][cid];
      int v = edges[id].to;

      if (level[u] + 1 != level[v] || edges[id].cap - edges[id].flow < 1) continue;
      long long tr = dfs(v, snk, min(pushed, edges[id].cap - edges[id].flow));
      if (tr == 0) continue;

      edges[id].flow += tr;
      edges[id^1].flow -= tr;
      return tr;
    }

    return 0;
  }

  long long flow(int src, int snk) {
    long long f = 0;
    while (bfs(src, snk)) {
      fill(ptr.begin(), ptr.end(), 0);
      while (long long pushed = dfs(src, snk, flow_inf)) f += pushed;
    }
    return f;
  }
};
```

7.18 CycleSimulation

7.18.1 simple

```
/*
  Resolve o problema: dado um grafo funcional valorado, com O(N) estados e um tempo T, qual estado
    se termina ap s uma soma de pesos igual T for percorrida
  Geralmente nesses problemas, o T significa a dura  o da simula  o , e os pesos das arestas
    representam o tempo da transi  o de um estado para outro.
  Vers o simples: os pesos das arestas s o 1.
*/

#include <bits/stdc++.h>

using namespace std;

struct State {

  State next() {

  }

  int hash() {

  }
```

```
};

struct Simulator {

  map<int, int> vis;

  State Simulate(int t, State cur) {
    int period = 0;

    while(t > 0) {
      if(vis.count(cur.hash())) {
        period -= vis[cur.hash()];
        break;
      }

      vis[cur.hash()] = period;
      cur = cur.next();
      period++;
      t--;
    }

    if(t) t %= period;
    while(t--) {
      cur = cur.next();
    }

    return cur;
  }
};
```

7.18.2 tortoise$_h$are

```
/*
  Resolve o problema: dado um grafo funcional valorado, com O(N) estados e um tempo T, qual estado se termina ap s uma soma de pesos igual T for percorrida
  Geralmente nesses problemas, o T significa a dura  o da simula  o, e os pesos das arestas representam o tempo da transi  o de um estado para outro.
  Vers o fera que inclui encontrar o periodo do ciclo com o algoritmo da lebre e tartaruga, removendo um log do map de visitado

  // Acho que com pesos 0 funciona
*/

#include <bits/stdc++.h>

using namespace std;

struct State {

  // representacao unica do estado
  int hash() {

  }

  // vai pro proximo estado, e retorna o tempo atravessado
  int next() {

  }

  // Faz o passo final (quando sobre um tantinho de tempo mas ainda nao da pra ir no proximo estado)
  State finish(int tim) {

  }
};

struct Simulator {

  map<int, int> vis;

  // Retorna periodo do ciclo. Possivelmente muda o estado e decrementa o tempo. Retorna 0 se o ciclo nao for encontrado antes do tempo esgotar
  int find_period(State & cur, int & t) {
    vis.clear();

    int period = 0;
    while(t > 0) {
      if(vis.count(cur.hash()) ) {
        period -= vis[cur.hash()];
        break;
      }
```

```
45
46        vis[cur.hash()] = period;
47        State aux = cur;
48        int tim = cur.next();
49
50        if(t - tim < 0) {
51          cur = aux;
52          return 0;
53        }
54        period += tim;
55        t -= tim;
56      }
57      return period;
58    }
59
60    // retorna periodo do ciclo
61    int find_period_tortoise_hare(State & cur, int & t) {
62      State tor = cur, hare = cur;
63
64      // Acha o ponto de inicio do ciclo
65      do {
66        tor.next();
67        hare.next(); hare.next();
68      } while(tor.hash() != hare.hash());
69
70      // Calcula o periodo do ciclo
71      int per = 0;
72      do {
73        per += tor.next();
74      } while(tor.hash() != hare.hash());
75
76      // Faz a simulacao ate o inicio do ciclo
77      do {
78        State aux = cur;
79        int tim = cur.next();
80        if(t - tim < 0) {
81          cur = aux;
82          return 0;
83        }
84        t -= tim;
85      } while(cur.hash() != tor.hash());
86
87      return per;
88    }
89
90    // Roda a simulacao
91    State Simulate(int t, State cur, bool use_tortoise_hare = false) {
92      int period = use_tortoise_hare? find_period_tortoise_hare(cur, t) : find_period(cur, t);
93
94      if(period) t %= period;
95
96      while(t > 0) {
97        State aux = cur;
98        int tim = cur.next();
99        if(t - tim < 0) {
100         cur = aux;
101         break;
102       }
103       t -= tim;
104     }
105
106     cur = cur.finish(t);
107
108     return cur;
109   }
110 };
```

7.18.3 full

```
1  /*
2    Resolve o problema: dado um grafo funcional valorado, com O(N) estados e um tempo T, qual estado
       se termina ap s uma soma de pesos igual T for percorrida
3    Geralmente nesses problemas, o T significa a dura  o da simula  o , e os pesos das arestas
       representam o tempo da transi  o de um estado para outro.
4
5    // Acho que com pesos 0 funciona
6  */
7
8  #include <bits/stdc++.h>
9
```

```
using namespace std;

struct State {

  // representacao unica do estado
  int hash() {

  }

  // vai pro proximo estado, e retorna o tempo atravessado
  int next() {

  }

  // Faz o passo final (quando sobre um tantinho de tempo mas ainda nao da pra ir no proximo estado)
  State finish(int tim) {

  }
};

struct Simulator {

  map<int, int> vis;

  // recebe o tempo total e o estado inicial
  State Simulate(int t, State cur) {
    int period = 0;

    while(t > 0) {

      if(vis.count(cur.hash()) ) {
        period -= vis[cur.hash()];
        break;
      }

      vis[cur.hash()] = period;
      State aux = cur;
      int tim = cur.next();

      if(t - tim < 0) {
        period = 0;
        cur = aux;
        break;
      }
      period += tim;
      t -= tim;
    }

    if(period) t %= period;

    while(t > 0) {
      State aux = cur;
      int tim = cur.next();
      if(t - tim < 0) {
        cur = aux;
        break;
      }
      t -= tim;
    }

    cur = cur.finish(t);

    return cur;
  }
};
```

8 Solutions

8.1 Geometry

8.1.1 NearestTwoPoints

```
/** return the minimum distance**2 between 2 points in the plane. O(N logn) */
template<typename T> T closest_pair_distance2(vector<Point<T>> p) {
  sort(p.begin(), p.end());

  vector<Point<T>> strip(p.size());
  auto find_closest = [&](auto&& find_closest, vector<Point<T>>& p, int l, int r)->T {
```

```
    if (r - l <= 0) return numeric_limits<T>::max();

    int m = (l + r) / 2;
    T distL = find_closest(find_closest, p, l, m);
    T distR = find_closest(find_closest, p, m + 1, r);
    T dist = min(distL, distR);

    int strip_index = 0;
    for (int i = l, j = m + 1; i <= m || j <= r; ) {
      if (j > r || (i <= m && p[i].y < p[j].y)) strip[strip_index++] = p[i++];
      else strip[strip_index++] = p[j++];
    }
    for (int i = l; i <= r; i++) p[i] = strip[i - l];

    strip_index = 0;
    for (int i = l; i <= r; i++) if ((p[i].x - p[m].x) * (p[i].x - p[m].x) < dist)
      strip[strip_index++] = p[i];

    for (int i = 0; i < strip_index; i++) {
      for (int j = i + 1; j < strip_index && (strip[j].y - strip[i].y) * (strip[j].y - strip[i].y) <
     dist; j++) {
        dist = min(dist, strip[i].dist2(strip[j]));
      }
    }

    return dist;
  };

  return find_closest(find_closest, p, 0, (int) p.size() - 1);
}
```

8.2 Strings

8.2.1 Hash

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

#define hash UISHDUIAHSDU
struct Hash {
  static constexpr int MOD[2] = {(int) 1e9+7, (int) 1e9+9};
  int val[2];

  Hash() { val[0] = val[1] = 0; }
  Hash(string const& s) { *this = calculateHash(s); }
  Hash(int x) { val[0] = x % MOD[0]; val[1] = x % MOD[1]; }
  Hash(int x, int y) { val[0] = x % MOD[0]; val[1] = y % MOD[1]; }

  static int add(int x, int y, int k) { x += y; if (x >= MOD[k]) x -= MOD[k]; return x; }
  static int sub(int x, int y, int k) { x -= y; if (x < 0) x += MOD[k]; return x; }
  static int mul(int x, int y, int k) { return 1ll * x * y % MOD[k]; }
  static int fpow(int x, int y, int k) {
    int r = 1;
    for (; y > 0; y /= 2, x = mul(x, x, k))
      if (y % 2 == 1) r = mul(r, x, k);
    return r;
  }
  static int divi(int x, int y, int k) { return mul(x, fpow(y, MOD[k] - 2, k), k); }
  static Hash pow(Hash x, int y) {
    Hash r = 1;
    for (; y >= 0; y /= 2, x *= x)
      if (y%2 == 1) r *= x;
    return r;
  }

  Hash operator+(Hash const& h) const { return Hash(add(val[0], h.val[0], 0), add(val[1], h.val[1],
    1)); }
  Hash operator-(Hash const& h) const { return Hash(sub(val[0], h.val[0], 0), sub(val[1], h.val[1],
    1)); }
  Hash operator*(Hash const& h) const { return Hash(mul(val[0], h.val[0], 0), mul(val[1], h.val[1],
    1)); }
  Hash operator/(Hash const& h) const { return Hash(divi(val[0], h.val[0], 0), divi(val[1], h.val
    [1], 1)); }
  Hash& operator+=(Hash const& h) { return *this = *this + h; }
  Hash& operator-=(Hash const& h) { return *this = *this - h; }
  Hash& operator*=(Hash const& h) { return *this = *this * h; }
  Hash& operator/=(Hash const& h) { return *this = *this / h; }

  bool operator==(Hash const& h) const { return val[0] == h.val[0] && val[1] == h.val[1]; }
```

```
  bool operator!=(Hash const& h) const { return val[0] != h.val[0] || val[1] != h.val[1]; }

  static Hash calculateHash(string const& s, Hash const primes = Hash(31, 37)) {
    Hash cur = 0;
    Hash p = 1;
    for (char c : s) {
      cur += p * (c - 'a' + 1); // assuming that is a lowercase string
      p *= primes;
    }
    return cur;
  }
  static vector<Hash> calculateHashVector(string const& s, Hash const primes = Hash(31, 37)) {
    int n = s.size();
    Hash p = 1;
    vector<Hash> cur(n);
    for (int i = 0; i < n; i++) {
      if (i) cur[i] = cur[i-1];
      cur[i] += p * (s[i] - 'a' + 1);
      p *= primes;
    }
    return cur;
  }
  static vector<Hash> calculatePowerVector(Hash p, const int n) {
    vector<Hash> ans(n);
    ans[0] = 1;
    for (int i = 1; i < n; i++)
      ans[i] = ans[i-1] * p;
    return ans;
  }
};

ostream& operator<<(ostream& out, Hash const& h) {
  return out << "[" << h.val[0] << "," << h.val[1] << "]";
}

vector<int> divisors(string const& s) { // return vector of indices i s.t. s[0..i] divides s
  int n = s.size();

  Hash primes(31, 37);

  vector<Hash> curPrime = Hash::calculatePowerVector(primes, n + 1);
  Hash total = s;

  vector<int> ans;
  for (int sz = 1; sz <= n; sz++) if (n % sz == 0) {
      Hash curHash = s.substr(0, sz);
      Hash finalHash = 0;
      for (int i = 0; i < n; i += sz) finalHash += curHash * curPrime[i];
      if (finalHash == total) ans.push_back(sz - 1);
    }

  return ans;
}
int commonDivisors(string const& a, string const& b) {
  auto v1 = divisors(a);
  auto v2 = divisors(b);
  int ans = 0;
  for (int i = 0; i < (int) min(a.size(), b.size()); i++) {
    if (a[i] != b[i]) break;
    ans += binary_search(v1.begin(), v1.end(), i) && binary_search(v2.begin(), v2.end(), i);
  }
  return ans;
}

// https://codeforces.com/contest/182/problem/D
int main() {
  ios::sync_with_stdio(false); cin.tie(NULL);
  string a, b; cin >> a >> b;
  cout << commonDivisors(a, b) << "\n";
}
```

```
// https://cses.fi/problemset/task/1110/

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

#define hash UISHDUIAHSDU
```

```cpp
struct Hash {
  static constexpr int MOD[2] = {(int) 1e9+7, (int) 1e9+9};
  int val[2];

  Hash() { val[0] = val[1] = 0; }
  Hash(string const& s) { *this = calculateHash(s); }
  Hash(int x) { val[0] = x % MOD[0]; val[1] = x % MOD[1]; }
  Hash(int x, int y) { val[0] = x % MOD[0]; val[1] = y % MOD[1]; }

  static int add(int x, int y, int k) { x += y; if (x >= MOD[k]) x -= MOD[k]; return x; }
  static int sub(int x, int y, int k) { x -= y; if (x < 0) x += MOD[k]; return x; }
  static int mul(int x, int y, int k) { return 1ll * x * y % MOD[k]; }
  static int fpow(int x, int y, int k) {
    int r = 1;
    for (; y > 0; y /= 2, x = mul(x, x, k))
      if (y % 2 == 1) r = mul(r, x, k);
    return r;
  }
  static int divi(int x, int y, int k) { return mul(x, fpow(y, MOD[k] - 2, k), k); }
  static Hash pow(Hash x, int y) {
    Hash r = 1;
    for (; y >= 0; y /= 2, x *= x)
      if (y%2 == 1) r *= x;
    return r;
  }

  Hash operator+(Hash const& h) const { return Hash(add(val[0], h.val[0], 0), add(val[1], h.val[1], 1)); }
  Hash operator-(Hash const& h) const { return Hash(sub(val[0], h.val[0], 0), sub(val[1], h.val[1], 1)); }
  Hash operator*(Hash const& h) const { return Hash(mul(val[0], h.val[0], 0), mul(val[1], h.val[1], 1)); }
  Hash operator/(Hash const& h) const { return Hash(divi(val[0], h.val[0], 0), divi(val[1], h.val[1], 1)); }
  Hash& operator+=(Hash const& h) { return *this = *this + h; }
  Hash& operator-=(Hash const& h) { return *this = *this - h; }
  Hash& operator*=(Hash const& h) { return *this = *this * h; }
  Hash& operator/=(Hash const& h) { return *this = *this / h; }

  bool operator==(Hash const& h) const { return val[0] == h.val[0] && val[1] == h.val[1]; }
  bool operator!=(Hash const& h) const { return val[0] != h.val[0] || val[1] != h.val[1]; }

  static Hash calculateHash(string const& s, Hash const primes = Hash(31, 37)) {
    Hash cur = 0;
    Hash p = 1;
    for (char c : s) {
      cur += p * (c - 'a' + 1); // assuming that is a lowercase string
      p *= primes;
    }
    return cur;
  }
  static vector<Hash> calculateHashVector(string const& s, Hash const primes = Hash(31, 37)) {
    int n = s.size();
    Hash p = 1;
    vector<Hash> cur(n);
    for (int i = 0; i < n; i++) {
      if (i) cur[i] = cur[i-1];
      cur[i] += p * (s[i] - 'a' + 1);
      p *= primes;
    }
    return cur;
  }
  static vector<Hash> calculatePowerVector(Hash p, const int n) {
    vector<Hash> ans(n);
    ans[0] = 1;
    for (int i = 1; i < n; i++)
      ans[i] = ans[i-1] * p;
    return ans;
  }
};

ostream& operator<<(ostream& out, Hash const& h) {
  return out << "[" << h.val[0] << "," << h.val[1] << "]";
}

inline void minimalRotation(string s) {
  int n = s.size();
  s += s;

```

```
  Hash primes(31, 29);
  vector<Hash> hash = Hash::calculateHashVector(s, primes);
  vector<Hash> power = Hash::calculatePowerVector(primes, n);

  // lexicographically compares two substrings: s[A..A+n] and s[B..B+n]
  auto cmp = [&](int const A, int const B) {
    if (s[A] != s[B]) return s[A] < s[B];
    Hash h1, h2;
    int lo = 2, hi = n, mi;

    while(lo < hi) {
      mi = (lo + hi) / 2;
      h1 = hash[A + mi - 1] - (A ? hash[A - 1] : 0);
      h2 = hash[B + mi - 1] - (B ? hash[B - 1] : 0);
      if (A < B) h1 *= power[B - A];
      else h2 *= power[A - B];

      if (h1 != h2) hi = mi;
      else lo = mi + 1;
    }
    return s[A+hi-1] < s[B+hi-1];
  };

  vector<int> a(n);
  iota(a.begin(), a.end(), 0);
  nth_element(a.begin(), a.begin(), a.end(), cmp);
  cout << s.substr(a[0], n) << "\n";
}

int main() {
  ios::sync_with_stdio(false); cin.tie(NULL);
  string s; cin >> s;
  minimalRotation(s);
}
```

8.3 Graph

8.3.1 DagWidth

```
/* https://maps20.kattis.com/problems/thewrathofkahn */
/*
 * Uses Dilworth's theorem (https://www.google.com.br/amp/s/www.geeksforgeeks.org/dilworths-theorem/amp/), that
 * says that the largest anti-chain of a partially ordered set is equal to the minimum number of chains that covers the set
 *
 * Translating to graph world, chain is a path, partially ordered set is a DAG and the largest anti-chain is the width of the DAG
 * We only have to calculate minimum number of paths to cover a DAG, which is a classic problem and can be calculated with matching
 *
 * This problem is equivalent to calculating the minimum number of *disjoint* paths to cover the transitive closure of the DAG,
 * which is a DAG where the existence of edge u->v and existence of edge v->w implies existence of edge u->w. Basically, there is
 * an edge u->v in the transitive closure of the DAG if there is a path from u to v in the original DAG.
 *
 * To calculate this with matching, we can see the graph as if initially there are n paths of length 1 (each node alone). A matching
 * u -> v means, then, that we are merging the path that ends in u with the path that starts with v. So to calculate this we
 * duplicate the graph into a bipartite graph, and add edge u from the left side to v of the right side if u->v is present in the DAG.
 * Since everytime we add an edge we subtract the number of paths by one, because we want the maximum matching, we will achieve the minimum
 * number of paths.
 *
 * Another way to see this is as if each edge of the matching will become an edge of the path. But wait, each node can match up with two nodes, not one.
 * That's why we duplicate the nodes, into one (let's call it A) that only receives incoming edges, and another (let's call it B) which only has
 * outgoing edges coming out of it, and there is an implicit edge from A to B to construct the paths. That way, each node can only match with one other node,
 * and matching algorithm can work correctly.
```

```
 */
#include "Kuhn.cpp"


namespace DagWidth {

  const int N = 1e5 + 7;
  vector<int> e[N];
  bool vis[N];

  void dfs(int u) {
    vis[u] = true;
    for(int v : e[u]) {
      if(!vis[v]) dfs(v);
    }
  }
  int solve(int n) {
    // build closure graph

    Kuhn kuhn(n, n);
    for(int i = 1; i <= n; ++i) {
      for(int j = 1; j <= n; ++j) vis[j] = false;

      dfs(i);

      for(int j = 1; j <= n; ++j) {
        if(i == j) continue;
        if(vis[j]) kuhn.add_edge(i, j);
      }
    }

    return kuhn.solve();
  }
};
```

8.4 SegmentTree

8.4.1 sorting

```
// Calcula o n mero de compara  es de um quick sort
// https://www.codechef.com/problems/SORTING
/*
  Supondo que n o  h  randomiza  o do quicksort, e que um indice fixo (nesse problema, o indice
    do meio), vai ser usado como pivot para fazer as compara  es. Repare que como n o
    randomizado, um numero quadratico de compara  es pode ser feito
  Observa  es:
    - Na recurs o do quicksort sempre dividimos o alfabeto atual entre o conjunto menor que o pivot
     e o conjunto maior.
    - Para uma chamada atual da recurs o, o n mero de compara  es sempre  o tamanho do vetor
    - N o importa qual  o indice (do array original) escolhido, e sim como o alfabeto
    particionado

  A ideia  manter o range [L, R] atual do alfabeto (assim como na wavelet)
  Com isso, pra somar o tamanho desse vetor ,  s  saber quantos caras existem com esse range do
    alfabeto. Pra achar o k- simo, da pra fazer com wavelet
  (onde queremos saber o n/2- simo de um array onde os valores do array original s o os indices e
    o indice do array original sao os valores). Aqui eu fiz com seg persistente

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

#define eb emplace_back
#define mid ((l+r)>>1)

using namespace std;
typedef long long ll;
typedef pair<int, int> ii;

struct node {
  int val;
  node * l, * r;

  node() {
    val = 0;
    l = r = 0;
  }
};
```

```

node * update(node * cur, int l, int r, int j, int x) {
  if(r < j or l > j) return cur;
  node * u = cur? new node(*cur) : new node();
  if(l == r) {
    u->val = x;
  } else {
    u->l = update(u->l, l, mid, j, x);
    u->r = update(u->r, mid+1, r, j, x);
    u->val = (u->l? u->l->val : 0) + (u->r? u->r->val : 0);
  }
  return u;
}

inline int getl(node * u) {
  if(u == NULL or u->l == NULL) return 0;
  return u->l->val;
}
inline int getr(node * u) {
  if(u == NULL or u->r == NULL) return 0;
  return u->r->val;
}

int kth(node * u, node * v, int l, int r, int k) {

  if(l == r) return l;

  // quantos tem na esquerda
  int left = getl(v) - getl(u);
  if(left >= k) return kth(u?u->l:0, v?v->l:0, l, mid, k);
  else return kth(u?u->r:0, v?v->r:0, mid+1, r, k-left);
}


const int N = 5e5 + 7;
node * ver[N];
int label[N];
int n;
// lo = indice do menor valor no vetor ordenado
// hi = indice do maior valor no vetor ordenado
ll go(int lo, int hi) {
  if(lo >= hi) return 0;
  int tot = (ver[hi]? ver[hi]->val:0) - (ver[lo-1]?ver[lo-1]->val : 0);
  int middle = kth(ver[lo-1], ver[hi], 1, n, (tot+1)/2); // meio do indice
  middle = label[middle];
  return go(lo, middle-1) + go(middle+1, hi) + tot;
}

int main() {
   cin >> n;

  vector<ii> vec;
  for(int i = 1; i <= n; ++i) {
    int x;
    cin >> x;
    vec.eb(x, i);
  }

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

  ver[0] = NULL;

  for(int i = 1; i <= n; ++i) {
    int x, j; tie(x, j) = vec[i-1];
    ver[i] = update(ver[i-1], 1, n, j, 1);
    label[j] = i;
  }


  cout << go(1, n) << endl;




}
```

8.5 Greedy

8.5.1 StableMarriage

```cpp
// solution for https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3616

#include <bits/stdc++.h>

using namespace std;

/*
  O casamento estavel tradicional    entre N homens e N mulheres.
  Aqui a gente assume que esses numeros podem diferir, o que implica que alguem vai ficar sem par
  Aqui nos temos N homens e M mulheres (N proposers e M proposees). O resultado vai ser otimo pros proposers
  homens (proposers) vao de 1 a N
  mulheres (proposees) vao de 1 a M
*/
struct StableMarriage {
  int n, m;
  vector<vector<int> > prior; // lista de preferencias dos homens
  vector<unordered_map<int, int> > rank; // rank[i][x] = indice do homem x (do primeiro conjunto), na lista de preferencias da mulher i (do segundo conjunto)

  // n proposers, m proposees
  StableMarriage(int n, int m) {
    this->n = n, this->m = m;
    prior.resize(n + 1);
    rank.resize(m + 1);
  }

  void setPreferencesMan(int x, vector<int> vec) {
    prior[x] = vec;
  }

  void setPreferencesWoman(int y, vector<int> vec) {
    int cur = 0;
    for(int x : vec) {
      rank[y][x] = cur++;
    }
  }

  // retorna ans (1-based!!!), onde ans[x] = mulher com quem x casou, ou 0 se ele ficou encalhado
  // voce deve ignorar ans[0]
  vector<int> solve() {
    vector<int> ans(n + 1, 0), cur(n + 1, 0), match(m + 1, 0);
    queue<int> q;

    for(int i = 1; i <= n; ++i) {
      q.push(i); // fila de rejeitados
    }

    while(q.size()) {
      int x = q.front(); q.pop();

      for(int & i = cur[x]; i < prior[x].size() and !ans[x]; i++) {
        int y = prior[x][i];
        if(!match[y] or rank[y][match[y]] > rank[y][x]) { // se a mulher nao ta noivada, ou se ela prefere eu
          q.push(match[y]); // coloca o noivo dela novamente na fila de rejeitados
          ans[match[y]] = 0;

          // faz um novo noivado
          match[y] = x;
          ans[x] = y;
        }
      }
    }

    return ans;
  }

};

void _solve() {
  int n; cin >> n;

```

```
  StableMarriage stab(n, n);
  for(int i = 1; i <= n; ++i) {
    vector<int> vec(n);
    for(int j = 0; j < n; ++j) {
      cin >> vec[j];
    }
    stab.setPreferencesMan(i, vec);
  }

  for(int i = 1; i <= n; ++i) {
    vector<int> vec(n);
    for(int j = 0; j < n; ++j) {
      cin >> vec[j];
    }
    stab.setPreferencesWoman(i, vec);
  }

  vector<int> ans = stab.solve();

  for(int i = 1; i < ans.size(); ++i) {
    cout << ans[i] << '\n';
  }
}

int main() {
  int t; cin >> t;
  while(t--) {
    _solve();
    if(t) cout << "\n";
  }
}
```

8.6 Math

8.6.1 AndConvolution

```
// https://csacademy.com/contest/archive/task/and-closure/statement/

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

#define pb push_back
#define eb emplace_back
#define mk make_pair
#define fi first
#define se second
#define mset(a, b) memset(a, b, sizeof(a))
#define DBG(x) cout << "[" << #x << "]: " << x << endl
using ll = long long;
using pii = pair<int, int>;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
#ifdef _WIN32
#define getchar_unlocked _getchar_nolock
#endif

template<class Ty> Ty randint(Ty a, Ty b) { return uniform_int_distribution<Ty>(a, b)(rng); }
template<class num> inline void rd(num& x) {
  char c, neg = 0; while(isspace(c = getchar_unlocked()));
  if(!isdigit(c)) neg = (c == '-'), x = 0;
  else x = c - '0';
  while(isdigit(c = getchar_unlocked())) x = (x << 3) + (x << 1) + c - '0';
  x = neg ? -x : x; }
template <class Ty, class... Args> inline void rd(Ty& x, Args&... args) { rd(x); rd(args...); }

const int MAXN = 1 << 20, INF = 0x3f3f3f3f;
const int MOD = 1e9 + 7;

void fwht(vector<ll>& v, bool invert) {
  int n = v.size();
  for (int mlen = 1; mlen + mlen <= n; mlen <<= 1) {
    for (int i = 0; i < n; i += mlen + mlen) {
      for (int j = 0; j < mlen; j++) {
        ll a = v[i + j], b = v[i + j + mlen]; // solving (a + bx) for x = {1, -1}. if `or
    convolution`, use roots x=1, x=0
        v[i + j] = b;
        v[i + j + mlen] = a + b;
        if (invert) v[i + j] = b - a, v[i + j + mlen] = a;
```

```
41
42          if (v[i+j] < 0) v[i+j] += MOD;
43          if (v[i+j+mlen] >= MOD) v[i+j+mlen] -= MOD;
44        }
45      }
46    }
47  }
48
49  inline void mul(vector<ll>& f1, vector<ll> const& f2) {
50    for (int i = 0; i < MAXN; i++) f1[i] = f1[i] * f2[i] % MOD;
51  }
52
53  inline vector<ll> getFreq(vector<ll> const& a) {
54    vector<ll> f(MAXN);
55    for (ll x : a) f[x]++;
56    for (ll& x : f) x %= MOD;
57    return f;
58  }
59
60  inline int solve(vector<ll>& v, int y) {
61    v = getFreq(v);
62    fwht(v, false);
63    vector<ll> r = v;
64    for (y--; y > 0; y /= 2, mul(v, v)) if (y&1) mul(r, v);
65    fwht(r, true);
66    int ans = 0;
67    for (int i = 0; i < MAXN; i++) ans += r[i] != 0;
68    return ans;
69  }
70
71  int main() {
72    int n; rd(n);
73    vector<ll> a(n+1);
74    for (int i = 0; i < n; i++) rd(a[i]);
75    printf("%d\n", solve(a, n));
76  }
```

8.6.2 GrayCode

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /**
6   * Fun  es para a sequ ncia de strings bin rias 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,
       1100, ...
7   * A sequencia    uma travessia por todos as strings bin rias de certo tamanho, tal que a
       dist ncia de Hamming
8   * (n mero de bits diferentes) de cada string adjacente    1.
9   * Exemplo, [gray(0) = 000, ..., gray(7) = 100]    a sequencia de gray de strings bin rias de
       tamanho 3
10  * Essa sequencia tamb m    um ciclo hamiltoniano de custo m nimo.
11  * Algo parecido tamb m pode ser encontrado para permuta  es que diferem por um swap de numeros
       adjacentes
12  */
13 namespace GrayCode {
14
15   int gray(int k) {
16     return k^(k >> 1);
17   }
18
19   int order(int n) {
20     int ans = 0;
21     for(; n; n >>= 1)
22       ans ^= n;
23     return ans;
24   }
25 };
```

9 Setup

9.1 template

```
1  #include "bits/stdc++.h"
```

```
#define pb push_back
#define fi first
#define se second
#define eb emplace_back

using namespace std;
typedef long long ll;
typedef pair<int, int> ii;

int main() {
  ios::sync_with_stdio(0); cin.tie(0);
}
```

10 Math

10.1 GaussianElimination

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

using namespace std;

namespace GaussianElimination {
  const double EPS = 1e-9;
  const int INF = 2; // it doesn't actually have to be infinity or a big number

  // Retorna o numero de solu es do sistema (0, 1 ou INF). No caso de 1, ans cont m a solu  o
  // Complexidade: O(min(n, m) * m) - n    o numero de colunas e m    o n mero de linhas
  int gauss(vector<vector<double>> a, vector<double> & ans) {
    int n = a.size();
    int m = a[0].size() - 1;

    vector<int> where(m, -1);
    for(int col = 0, row = 0; col < m and row < n; ++col) {
      int sel = row;
      for(int i = row; i < n; ++i)
        if(abs(a[i][col]) > abs(a[sel][col]))
          sel = i;

      if(abs(a[sel][col]) < EPS)
        continue;

      for(int i = col; i <= m; ++i)
        swap(a[sel][i], a[row][i]);

      where[col] = row;

      for(int i = 0; i < n; ++i)
        if(i != row) {
          double c = a[i][col] / a[row][col];
          for(int j = col; j <= m; ++j)
            a[i][j] -= a[row][j] * c;
        }
      ++row;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; ++i)
      if (where[i] != -1)
        ans[i] = a[where[i]][m] / a[where[i]][i];
    for(int i = 0; i < n; ++i) {
      double sum = 0;
      for(int j = 0; j < m; ++j)
        sum += ans[j] * a[i][j];
      if(abs(sum - a[i][m]) > EPS)
        return 0;
    }

    for (int i = 0; i < m; ++i)
      if (where[i] == -1)
        return INF;
    return 1;
  }

  int fast_pow(int a, int b, int k) {
    int ans = 1;
```

```
59      while(b) {
60        if(b&1) ans = 1LL * ans * a % k;
61        b >>= 1;
62        a = 1LL * a * a % k;
63      }
64      return ans;
65    }
66
67    int inverse(int x, int k) {
68      return fast_pow(x, k - 2, k);
69    }
70
71    int gauss(vector<vector<int>> a, vector<int> & ans, int k) {
72      int n = a.size();
73      int m = a[0].size() - 1;
74
75      vector<int> where(m, -1);
76      for(int col = 0, row = 0; col < m and row < n; ++col) {
77        int sel = row;
78        for(int i = row; i < n; ++i)
79          if(abs(a[i][col]) > abs(a[sel][col]))
80            sel = i;
81
82        if(!abs(a[sel][col]))
83          continue;
84
85        for(int i = col; i <= m; ++i)
86          swap(a[sel][i], a[row][i]);
87
88        where[col] = row;
89
90        for(int i = 0; i < n; ++i)
91          if(i != row) {
92            int c = a[i][col] * inverse(a[row][col], k) % k;
93            for(int j = col; j <= m; ++j) {
94              a[i][j] -= a[row][j] * c % k;
95              if(a[i][j] < 0) a[i][j] += k;
96            }
97
98          }
99        ++row;
100     }
101
102     ans.assign(m, 0);
103     for (int i = 0; i < m; ++i)
104       if (where[i] != -1)
105         ans[i] = a[where[i]][m] * inverse(a[where[i]][i], k) % k;
106     for(int i = 0; i < n; ++i) {
107       int sum = 0;
108       for(int j = 0; j < m; ++j) {
109         sum += ans[j] * a[i][j] % k;
110         sum %= k;
111       }
112       if(sum % k != a[i][m] % k)
113         return 0;
114     }
115
116     int tot = 1;
117     for (int i = 0; i < m; ++i)
118       if (where[i] == -1)
119         tot++;
120     return 1;
121   }
122
123   // Retorna o numero de solu es do sistema (0, 1 ou INF). No caso de 1, ans cont m a solu  o
124   // Complexidade: O(n * m) - n    o numero de colunas e m    o n mero de linhas
125   template<int N>
126   int gauss (vector<bitset<N>> a, int n, int m, bitset<N> & ans) {
127     vector<int> where (m, -1);
128     for(int col = 0, row = 0; col < m and row < n; ++col) {
129       for(int i = row; i < n; ++i)
130         if(a[i][col]) {
131           swap(a[i], a[row]);
132           break;
133         }
134       if(!a[row][col])
135         continue;
136       where[col] = row;
137
```

```
      for(int i = 0; i < n; ++i)
        if(i != row && a[i][col])
          a[i] ^= a[row];
      ++row;
    }

    // The rest of implementation is the same as above
    ans = 0;
    for(int i = 0; i < m; ++i)
      if(where[i] != -1)
        ans[i] = a[where[i]][m];

    for(int i = 0; i < n; ++i) {
      int sum = 0;
      for(int j = 0; j < m; ++j)
        sum ^= ans[j] * a[i][j];
      if(sum^a[i][m])
        return 0;
    }

    for (int i = 0; i < m; ++i)
      if (where[i] == -1)
        return INF;
    return 1;
  }
};
```

10.2 LinearRec

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

// evaluates linear reccurences in O(n^2*log(k)) time
struct LinRec {
    using ll = long long;

    constexpr static int mod = 1e9 + 7;

public:
    // reccurence f[i] defined by vectors c and s
    // f[i+1] = c[0]*f[i] + c[1]*f[i-1] + ... + c[n-1]*f[i-n+1]
    // f[0] = s[0], f[1] = s[1], ..., f[n-1] = s[n-1]
    LinRec(vector<int> const& c_, vector<int> const& s_) : n(c_.size()), c(c_), s(s_) {
        cache[0] = {0, 1};

        for (int i = 1; i < (int)cache.size(); i++) {
            cache[i] = mul(cache[i-1], cache[i-1]);
        }
    }

    // evaluates G(x ^ k mod char_poly(f[x])) = G(x ^ k) = f[k]
    // where G(f[x]) = sum(c[i]*s[i]) from 0 to n
    // O(n^2 * log(k)) time
    ll kth(ll k) {
        if (k < n) return s[k];

        // x ^ k mod char_poly(f[x])
        auto x = fexp(k);

        ll r = 0;
        // G(x)
        for (int i = 0; i < n; i++) {
            r += (ll) x[i] * s[i] % mod;
        }

        return r % mod;
    }

private:
    int n;
    vector<int> c, s;
    array<vector<int>, 64> cache;

    // computes a[x] * b[x] mod characteristic_poly(s)
    // optimized naive (O(n^2)) implementation
    vector<int> mul(vector<int> const& a, vector<int> const& b) {
        // guarantees an >= bn
        if (a.size() < b.size()) return mul(b, a);
```

```
50
51          int const an = a.size(), bn = b.size();
52          vector<int> res(an + bn - 1);
53
54          // multiplies a and b, computing per exponent
55          for (int i = 0; i < an + bn - 1; i++) {
56              int x = min(i, an - 1);
57              int y = i - x;
58
59              ll acc = 0;
60
61              if (i < bn) {
62                  for (; x >= 0; x--, y++) {
63                      acc += (ll) a[x] * b[y] % mod;
64                  }
65              }
66              else {
67                  for (; y < bn; x--, y++) {
68                      acc += (ll) a[x] * b[y] % mod;
69                  }
70              }
71
72              res[i] = acc % mod;
73          }
74
75
76          // reduces mod char_poly(s)
77          for (int i = an+bn-2; i >= n; i--) {
78              for (int j = n-1; j >= 0; j--) {
79                  res[i-j-1] += res[i] * c[j];
80              }
81          }
82
83          res.resize(min(an + bn - 1, n));
84          return res;
85      }
86
87      vector<int> fexp(ll k) {
88          vector<int> ans = {1};
89          for (int i = 0; k > 0; k >>= 1, i++) {
90              if (k&1) ans = mul(ans, cache[i]);
91          }
92          return ans;
93      }
94  };
95
96  int main() {
97
98      LinRec a({1, 3}, {1, 2});
99
100     int x;
101     cin >> x;
102
103     cout << a.kth(x) << "\n";
104 }
```

10.3 EulerPhi

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5
6  /*
7   * Phi(x) = qtd de 1 <= y < x tal que gcd(x, y) = 1
8   * a^Phi(N) = 1 (mod N) para qualquer a coprimo com N.
9   *    uma fun  o multiplicativa
10  * A soma de phi(d) para todo d que divide N    N
11  */
12
13 namespace EulerPhi {
14   const int N = 1000007;
15   int phi[N];
16
17   // Builds every phi[x] for small x (x <= 10^6)
18   void build() {
19     for(int i = 1; i < N; ++i) {
20       phi[i] = i;
```

```
    }
    for(int i = 1; i < N; ++i) {
      for(int j = i + i; j < N; j += i) {
        phi[j] -= phi[i];
      }
    }
  }

  map<int, int> factor(int n) {
    map<int, int> ans;
    for(int i = 2; i * i <= n; ++i) {
      while(n%i == 0) {
        ans[i]++;
        n /= i;
      }
    }
    if(n > 1) ans[n]++;
    return ans;
  }

  // finds Phi in sqrt(n)
  int Phi(int n) {
    auto fact = factor(n);
    int ans = 1;

    for(auto [a, b] : fact) {
      ans *= fast_pow(a, b, n+1) - fast_pow(a, b-1, n+1);
    }
    return ans;
  }
}
```

10.4 FixedMatrix

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

using namespace std;

using ll = long long;

template<typename T, int n, int m, int mod = -1>
struct Matrix {
  static constexpr bool has_mod = (mod != -1);

  array<T, n*m> mat = {};

  // matrix indexing is done via operator()
  // ie for a matrix m, m(1, 0), accesses the first element of the second row
  constexpr inline T const& operator()(int i, int j) const {
    return mat[i*m + j];
  }

  constexpr inline T& operator()(int i, int j) {
    return mat[i*m + j];
  }

  // static constructors
  static Matrix from_vecs(vector<vector<T>> const& v) {
    Matrix r;
        for (int i = 0; i < n; i++) {
            copy(v[i].begin(), v[i].end(), r.mat.begin() + i*m);
        }
    return r;
    }

  constexpr static Matrix identity() {
    Matrix r{};
    for(int i = 0; i < n; ++i) r(i, i) = 1;
    return r;
  }

  constexpr static Matrix zeros() {
    return Matrix{};
  }

  // operators
  constexpr Matrix& operator+=(Matrix const& rhs) {
    for(int i = 0; i < n; ++i) {
```

```
45        for(int j = 0; j < m; ++j) {
46          (*this)(i, j) += rhs(i, j);
47          if constexpr (has_mod) {
48            if ((*this)(i, j) >= mod) (*this)(i, j) -= mod;
49          }
50        }
51      }
52      return *this;
53    }
54
55    constexpr Matrix operator+(Matrix const& rhs) const {
56      auto ans = *this;
57      return ans += rhs;
58    }
59
60    constexpr Matrix& operator-=(Matrix const& rhs) {
61      for(int i = 0; i < n; i++) {
62        for(int j = 0; j < m; j++) {
63          (*this)(i, j) -= rhs(i, j);
64          if constexpr (has_mod) {
65            if ((*this)(i, j) < 0) (*this)(i, j) += mod;
66          }
67        }
68      }
69      return *this;
70    }
71
72    constexpr Matrix operator-(Matrix const& rhs) const {
73      auto ans = *this;
74      return ans -= rhs;
75    }
76
77    template<int p>
78    constexpr Matrix<T, n, p, mod> operator*(Matrix<T, m, p, mod> const& rhs) const {
79      Matrix<T, n, p, mod> res{};
80      for (int i = 0; i < n; i++) {
81        // for large matrices swap the next two lines for better cache locality
82        for (int j = 0; j < p; j++) {
83          for (int k = 0; k < m; k++) {
84            if constexpr (has_mod) res(i, j) = ((ll)res(i, j) + (ll)(*this)(i, k) * rhs(k, j)) % mod;
85            else res(i, j) += (*this)(i, k) * rhs(k, j);
86          }
87        }
88      }
89      return res;
90    }
91
92    constexpr Matrix<T, n, n, mod> operator^(ll k) const {
93      // garantees matrix is square
94      static_assert(n == m, "n != m");
95
96      auto a = *this;
97      auto ans = Matrix<T, n, n, mod>::identity();
98      while(k) {
99        if(k&1) ans = ans * a;
100       a = a * a;
101       k >>= 1;
102     }
103     return ans;
104   }
105
106   constexpr Matrix& operator*=(T x) {
107     for (auto& i : mat) {
108       if constexpr (has_mod) i = 1ll * i * x % mod;
109       else i *= x;
110     }
111     return *this;
112   }
113
114   constexpr Matrix operator*(T x) const {
115     auto ans = *this;
116     return ans *= x;
117   }
118
119   constexpr bool operator==(Matrix<T, n, m, mod> const& rhs) const {
120     return mat == rhs.mat;
121   }
122
123   constexpr Matrix<T, m, n, mod> transposed() const {
```

```
    auto ans = Matrix<T, m, n, mod>{};
    for(int i = 0; i < n; ++i) {
      for(int j = 0; j < m; ++j) {
        ans(j, i) = (*this)(i, j);
      }
    }
    return ans;
  }

  friend ostream& operator<<(ostream& os, const Matrix& mat) {
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; ++j) {
        os << mat(i, j) << ' ';
      }
      os << '\n';
    }
    return os;
  }
};
```

10.5 LinearSieve

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

vector<int> linear_sieve(int n) {
    vector<int> low(n+1);
    vector<int> pr;

    for (int i = 2; i <= n; i++) {
        if (low[i] == 0) {
            low[i] = i;
            pr.push_back(i);
        }

        for (int j : pr) {
            if (j > low[i] or i*j > n) break;
            low[i * j] = j;
        }
    }

    return pr;
}
```

10.6 BerlekampMassey

```
/*
  Acha a menor recorrencia linear para a sequ ncia dada como input.
  Por exemplo, para a sequ ncia
  0 1 1 2 3 5 8 13
  A recorrencia encontrada ser  f(x) = 1 * f(x-1) + 1 * f(x-2)

  Como aprendido no blog https://codeforces.com/blog/entry/61306
*/

#include <bits/stdc++.h>

#define pb push_back

using namespace std;
typedef long long ll;

namespace BerlekampMassey {
  const int M = 1e9 + 7;

  ll fast_pow(ll a, ll b) {
    ll ans = 1;

    while(b) {
      if(b&1) ans = ans * a % M;
      b >>= 1;
      a = a * a % M;
    }
    return ans;
  }

```

```
  inline ll inv(ll x) {
    return fast_pow(x, M - 2);
  }

  /*
    'vec' eh o array com os primeiros vec.size() numeros da recorrencia
    'cur' eh o array de coeficientes multiplicados,
    e deve ser interpretado como vec[i] = cur[0] * vec[i-1] + cur[1] * vec[i-2] + ...
  */
  vector<ll> solve(vector<ll> vec) {
    const int n = vec.size();

    vector<vector<ll> > S(n, vector<ll>()); // S[i] = recorrencia calculada para o prefixo i, que avalia em 0 para todos os indices de 0 ate i e avalia em 1 para o i+1
    vector<ll> cur = {0}; // recorrencia atual

    S[0] = {1};

    for(int i = 1; i < n; ++i) {
      ll res = 0;
      assert((int) cur.size() <= i);
      for(int j = 0; j < cur.size(); ++j) {
        res = (res + cur[j] * vec[i - 1 - j] % M) % M;
      }

      if(vec[i] == res) continue; // deu certo

      ll v = (res - vec[i] + M) % M; // v tal que vec[i] + v = res

      // Calcula S[i]
      S[i].pb(M-1 * inv(v) % M);
      for(ll x : cur) S[i].pb(x * inv(v) % M);

      // recalcula cur
      int k = 0;
      for(int j = 0; j < i; ++j) {
        if(S[j].size() == 0) continue;
        if(S[j].size() + i - 1 - j <= S[k].size() + i - 1 - k) k = j;
      }

      vector<ll> aux(max(int(S[k].size()) + i - 1 - k, (int) cur.size()));
      for(int j = 0; j < aux.size(); ++j) {
        ll x = j < (i - 1 - k)? 0 : S[k][j - (i - 1 - k)];
        aux[j] = ((vec[i] - res + M) * x % M) + (j < cur.size()? cur[j] : 0);
        aux[j] %= M;
      }

      cur = aux;
    }

    return cur;
  }
};
```

10.7 Montgomery64

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

using u128 = __uint128_t;
using i128 = __int128_t;
using u64 = uint64_t;
using i64 = int64_t;

namespace {
    inline u64 hi(u128 x) {
        return u64(x >> 64);
    }

    inline u64 lo(u128 x) {
        return u64(x);
    }
}

struct Montgomery64 {
    Montgomery64(u64 n) : mod(n), inv(1), r2(-n % n) {
        for (int i = 0; i < 6; i++) {
            inv *= 2 - n * inv;
```

```
        }

        one = r2;

        r2 = u128(r2) * r2 % n;
    }

    u64 mod, inv, r2, one;

    u64 init(u64 x) {
        return mult(x, r2);
    }

    u64 reduce(u128 x) {
        u64 q = lo(x) * inv;
        i64 a = hi(x) - hi(u128(q) * mod);
        if (a < 0)
            a += mod;
        return a;
    }

    u64 mult(u64 a, u64 b) {
        return reduce(u128(a) * b);
    }
};

// creation + 1 mult has higher time as 1 int128 mulmod
// becomes faster afer around 4 multiplications + inits
// up to 6 times faster when doing lots of multiplications
// modulo de same number
// ------------------------------------------------------------
//                    benchmark results
// ------------------------------------------------------------
// int64modmul/1           19.0 ns         19.0 ns     36185356
// int64modmul/2           53.6 ns         53.6 ns     12638112
// int64modmul/4            117 ns          117 ns      6188974
// int64modmul/8            231 ns          231 ns      3026745
// int64modmul/16           465 ns          465 ns      1499313
// int64modmul/32           954 ns          954 ns       718907
// int64modmul/64          1900 ns         1899 ns       369682
// int64modmul/128         3782 ns         3780 ns       183062
// int64modmul/256         7663 ns         7659 ns        90720
// int64modmul/512        15193 ns        15185 ns        45400
// int64modmul/1024       30072 ns        30051 ns        23488
// int64modmul/2048       59431 ns        59398 ns        10000
// int64modmul/4096      112081 ns       112020 ns         7679
// ------------------------------------------------------------
// int64mont/1             56.8 ns         56.7 ns     12108520
// int64mont/2             58.7 ns         58.6 ns     11591214
// int64mont/4             63.9 ns         63.8 ns     10452542
// int64mont/8             80.0 ns         80.0 ns      8625561
// int64mont/16             116 ns          116 ns      5997047
// int64mont/32             188 ns          188 ns      3744182
// int64mont/64             334 ns          334 ns      2051937
// int64mont/128            620 ns          619 ns      1113791
// int64mont/256           1200 ns         1199 ns       587902
// int64mont/512           2321 ns         2320 ns       298828
// int64mont/1024          4631 ns         4628 ns       152314
// int64mont/2048          9186 ns         9182 ns        75944
// int64mont/4096         18280 ns        18272 ns        38717
```

10.8 Karatsuba

```
namespace Karatsuba {
  /** Karatsuba method. O(n ^ (log2_3)) ~ O(n^1.58) */
  template<typename T>
  inline void karatsuba(vector<T> const& a, vector<T> const& b, vector<T>& ans, int n) {
    if (n <= 64) {
      for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
          ans[i + j] += a[i] * b[j];
      return;
    }

    int mid = n / 2;
    vector<T> _a(mid), _b(mid);
    vector<T> E(n);

```

```
    karatsuba(a, b, ans, mid);
    for (int i = 0; i < mid; i++) {
      _a[i] = a[i + mid];
      _b[i] = b[i + mid];
    }
    karatsuba(_a, _b, E, mid);

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

    for (int i = 0; i < mid; i++) {
      _a[i] = a[i] + a[i + mid];
      _b[i] = b[i] + b[i + mid];
    }
    fill(E.begin(), E.end(), 0);
    karatsuba(_a, _b, E, mid);

    for (int i = 0; i < mid; i++) {
      const T tmp = ans[i + mid];
      ans[i + mid] += E[i] - ans[i] - ans[i + n];
      ans[i + mid + mid] += E[i + mid] - tmp - ans[i + n + mid];
    }
  }

  /** Multiplies two polynomials a * b = ans. O(n ^ (log2_3)) ~ O(n^1.58) */
  template<typename T>
  inline void mult(vector<T> const& a, vector<T> const& b, vector<T>& ans) {
    int n = 1;
    while(n < (int) max(a.size(), b.size())) n *= 2;
    vector<T> _a(a.begin(), a.end());
    vector<T> _b(b.begin(), b.end());
    _a.resize(n);
    _b.resize(n);
    ans.assign(n + n, 0);
    karatsuba(_a, _b, ans, n);
  }

  /** Multiplies two BigIntegers using base 10^p.
  *  T must be int128 for p=9. I guess that long long works for p < 9. */
  template<typename T, int p> inline void mult_bigint(string s1, string s2, string& ans) {
    T base = 1;
    for (int i = 0; i < p; i++) base *= 10;

    reverse(s1.begin(), s1.end());
    reverse(s2.begin(), s2.end());

    vector<T> a, b, c;
    for (int i = 0; i < (int) s1.size(); i += p) {
      T cur = 0;
      for (int j = i + p - 1; j >= i; j--)
        cur = cur * 10 + (j < (int) s1.size() ? s1[j] - '0' : 0);
      a.push_back(cur);
    }
    for (int i = 0; i < (int) s2.size(); i += p) {
      T cur = 0;
      for (int j = i + p - 1; j >= i; j--)
        cur = cur * 10 + (j < (int) s2.size() ? s2[j] - '0' : 0);
      b.push_back(cur);
    }

    mult(a, b, c);

    c.push_back(0);

    T carry = 0;
    for (int i = 0; i < (int) c.size(); i++) {
      carry += c[i];
      c[i] = carry % base;
      carry /= base;
    }
    while(!c.empty() && c.back() == 0) c.pop_back();
    reverse(c.begin(), c.end());

    ans = "";
    for (int i = 0; i < (int) c.size(); i++) {
      string cur;
      while(c[i] > 0) cur += (c[i] % 10) + '0', c[i] /= 10;
      reverse(cur.begin(), cur.end());

      if (i != 0) cur = string(p - (int) cur.size(), '0') + cur;
```

```
      ans += cur;
    }
    if (ans.empty()) ans = "0";
  }
}
```

10.9 Stirling

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

using namespace std;
using ll = long long;



/*
 * Stirling Numbers do primeiro e segundo tipo
 * 1o: F(n+1, k) = F(n, k-1) + n * F(n, k)
 * 2o: F(n+1, k) = F(n, k-1) + k * F(n, k)
 *
 */

namespace Stirling {
  const int N = 5007;
  int stirling[N][N];

  /** Usa a recorrencia F(n+1, k) = F(n, k-1) + ? * F(n, k), onde ? = N se kind = 1, e ? = K se kind = 2 */
  void build(int kind, int mod = 1e9 + 7) {
    stirling[0][0] = 1;
    for(int n = 1; n < N; n++) {
      for(int k = 1; k < N; k++) {
        const int x = kind == 1? n-1 : k;
        stirling[n][k] = (stirling[n-1][k-1] + 1LL * x * stirling[n-1][k] % mod) % mod;
      }
    }
  }

  int fast_pow(int a, int b, int mod) {
    int ans = 1;
    while(b) {
      if(b&1) ans = 1LL * ans * a % mod;
      b >>= 1;
      a = 1LL * a * a % mod;
    }
    return ans;
  }

  /**
   * Retorna um vector com S1(n, 1), S1(n, 2), ..., S1(n, n)
   * Faz isso calculando x * (x + 1) * (x + 2) * ... * (x + n - 1)
   * S1(n, k) = coeficiente de x^k
   * Nao esquecer de configurar corretamente as constantes do NTT, dependendo do seu MOD
   * Complexidade: O(n log(n))
   * Precisa incluir codigo de NTT
   */
  vector<int> stirling1(int n) {

    int nn = 1;
    while(nn < n) nn <<= 1;

    vector<vector<int> > pol(nn);

    // inicializa (x + i)
    for(int i = 0; i < n; ++i) {
      pol[i] = {i, 1};
    }

    // preenche o resto com (1)
    for(int i = n; i < nn; ++i) {
      pol[i] = {1};
    }

    // multiplica todos os s[i] com NTT
    for(int j = nn; j > 1; j >>= 1){
      int hn = j >> 1;
      for(int i = 0; i < hn; ++i) {
        pol[i] = NTT::multiply(pol[i], pol[i + hn]);
```

```
      }
    }

    return pol[0];
  }

  /** Calcula S2(n, k) em O(k log(n)), usando f rmula de convolu  o */
  int stirling2(int n, int k, int mod) {

    // calcula fatorial inverso
    vector<ll> inv_fat(k+1);
    inv_fat[0] = 1;
    for(int i = 1; i <= k; ++i) inv_fat[i] = inv_fat[i-1] * i % mod;
    inv_fat[k] = fast_pow(inv_fat[k], mod - 2, mod);
    for(int i = k-1; i >= 0; --i) inv_fat[i] = inv_fat[i+1] * (i+1) % mod;

    int ans = 0;
    for(int i = 0; i <= k; ++i) {
      int a = i%2? mod - inv_fat[i] : inv_fat[i];
      int b = inv_fat[k - i] * fast_pow(k - i, n, mod) % mod;
      ans = (ans + 1LL * a * b % mod) % mod;
    }

    return ans;
  }

  /**
   * Calcula S2(n, k) pra todo k de 0 ate n, usando convolucao com NTT
   * Nao esquecer de configurar corretamente as constantes do NTT, dependendo do seu MOD
   * Precisa incluir codigo de NTT
   * Complexidade: O(n log(n))
   */
  vector<int> stirling2(int n, int mod) {

    // calcula fatorial inverso
    vector<ll> inv_fat(n+1);
    inv_fat[0] = 1;
    for(int i = 1; i <= n; ++i) inv_fat[i] = inv_fat[i-1] * i % mod;
    inv_fat[n] = fast_pow(inv_fat[n], mod - 2, mod);
    for(int i = n-1; i >= 0; --i) inv_fat[i] = inv_fat[i+1] * (i+1) % mod;

    vector<int> a(n+1), b(n+1);
    for(int i = 0; i <= n; ++i) {
      a[i] = i%2? mod - inv_fat[i] : inv_fat[i];
      b[i] = inv_fat[i] * fast_pow(i, n, mod) % mod;
    }

    return NTT::multiply(a, b);
  }

  /** Paridade de S2(n, k), em O(1) */
  int second_kind_parity(int n, int k) {
    return ( (n-k) & ((k-1)/2) ) == 0;
  }
};
```

10.10 FWHT

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

using ll = long long;

namespace FWHT {
  /**
   * Fast Walsh-Hamadard Transform. N must be a power of 2. O(n logn)
   * (ax^i) (*) (bx^j) = ab * x^{i (op) j}, where op can be:
   *     Xor: u = a + b, v = a - b
   *      Or: u = a,     v = a + b
   *     And: u = b,     v = a + b
   * Used for xor-convolution or something like that.
   */
  void fwht(vector<ll>& v, bool invert) {
    int n = v.size();
    for (int mlen = 1; mlen + mlen <= n; mlen <<= 1) {
      for (int i = 0; i < n; i += mlen + mlen) {
        for (int j = 0; j < mlen; j++) {
          ll a = v[i + j], b = v[i + j + mlen]; // solving (a + bx) for x = {1, -1}.
```

```
          v[i + j] = a + b; // when x = 1, a+bx = a+b
          v[i + j + mlen] = a - b; // when x = -1, a+bx = a-b
          if (invert) v[i + j] >>= 1, v[i + j + mlen] >>= 1; // if u=a+b, v=a-b => a=(u+v)/2, b=(u-v)/2
        }
      }
    }
  }

  /** performs xor convolution and returns vector of frequency for each number.
   *                                n = 2^k, where a_i, b_i <= 2^k for each i.
   * basically is: for x in a: for y in b: ans[x^y]++; but in O(n logn)
   */
  vector<ll> xor_convolution(vector<ll> const& a, vector<ll> const& b, int n) {
    vector<ll> f1(n), f2(n);
    for (ll x : a) f1[x]++;
    for (ll x : b) f2[x]++;
    fwht(f1, false);
    fwht(f2, false);
    for (int i = 0; i < n; i++) f1[i] *= f2[i];
    fwht(f1, true);
    return f1;
  }
}

int main() {
  int n; scanf("%d", &n);
  vector<ll> a(n), b(n);
  for (int i = 0; i < n; i++) scanf("%lld", &a[i]);
  for (int i = 0; i < n; i++) scanf("%lld", &b[i]);
  FWHT::xor_convolution(a, b, 1 << 20);
}
```

10.11 NTT

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

using ll = long long;

/**
 * NTT eh o mesmo que FFT, so que com um *n-th root* diferente
 * MOD tem que ser um primo do tipo p = c * 2^k + 1
 * Com isso, a n-th root existe pra n = 2^k.
 * Essa n-th root    g^c, onde g    um *primitive root* de 2^k
 */

namespace NTT {
  const int mod =   998244353;// 998244353  7340033
  const int root =  363395222;// 15311432 5
  const int root_1 =  704923114;// 469870224  4404020
  const int root_pw = 1 << 19;//   1 << 23; 1 << 20;
  //          ~5*10^5    ~8*10^6  ~10^6

  int fast_pow(int a, int b, int m) {
    int ans = 1;
    while(b) {
      if(b&1) ans = 1LL * ans * a % m;
      a = 1LL * a * a % m;
      b >>= 1;
    }
    return ans;
  }
  int inverse(int x, int m) { return fast_pow(x, m - 2, m); }

  /* <============================== INICIO ======================================> */
  /** S   quando as constantes acima n o forem suficientes */

  map<int, int> factor(int n) {
    map<int, int> ans;
    for(int i = 2; i * i <= n; ++i) {
      while(n%i == 0) {
        ans[i]++;
        n /= i;
      }
    }
    if(n > 1) ans[n]++;
    return ans;
```

```
}

int Phi(int n) {
  auto fact = factor(n);
  int ans = 1;

  for(auto [a, b] : fact) {
    ans *= fast_pow(a, b, n + 1) - fast_pow(a, b-1, n + 1);
  }
  return ans;
}

int prim_root(int n) {
    int phi = Phi(n);
    auto fact = factor(phi);

  for (int res=2; res<=n; ++res) {
    if(__gcd(res, n) > 1) continue;
    bool ok = true;
    for (auto [a, b] : fact) {
      ok &= fast_pow(res, phi / a, n) != 1;
      if(!ok) break;
    }
    if (ok) return res;
  }
    return -1;
}

// Generates NTT constants for any Mod and prints it on the screen
void generate(int Mod) {

  int n = 1;
  int aux = Mod-1;
  while((aux&1) == 0) aux >>= 1, n <<= 1;
  int c = aux;

  // g = primitive root de Mod
  int g = prim_root(Mod);
  if(g == -1) {
    printf("No constants could be found, cant find primitive root of %d\n", n);
    return;
  }
  int root = fast_pow(g, c, Mod);
  int root_1 = inverse(root, Mod);
  printf("mod = %d, root = %d, root_1 = %d root_pw = %d\n", Mod, root, root_1, n);
  assert(fast_pow(root, n, Mod) == 1);
  // g^c
}
/* ================================>  FIM  <====================================== */


inline void ntt(vector<int>& a, bool invert) {
  int n = a.size();

  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(a[i], a[j]);
  }

  for (int len = 2; len <= n; len <<= 1) {
    int wlen = invert ? root_1 : root;

    for (int i = len; i < root_pw; i <<= 1)
      wlen = 1LL * wlen * wlen % mod;

    for (int i = 0; i < n; i += len) {
      int w = 1;
      for (int j = 0; j < len / 2; j++) {
        int u = a[i+j], v = 1LL * a[i+j+len/2] * w % mod;
        a[i+j] = u + v < mod ? u + v : u + v - mod;
        a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
        w = 1LL * w * wlen % mod;
      }
    }
```

```
    }


    if (invert) {
      int n_1 = inverse(n, mod);
      for (int & x : a)
        x = 1LL * x * n_1 % mod;
    }
  }

  vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<int> ca(a.begin(), a.end()), cb(b.begin(), b.end());
    int n = 1;
    while(n < (int) max(a.size(), b.size())) n <<= 1;
    n <<= 1;

    ca.resize(n); cb.resize(n);

    ntt(ca, false); ntt(cb, false);

    for (int i = 0; i < n; i++) ca[i] = 1LL * ca[i] * cb[i] % mod;
    ntt(ca, true);

    return ca;
  }
};
```

10.12 MillerRabin

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

namespace MillerRabin {
  using ll = long long;

  ll modmul(ll a, ll b, ll M) {
    #ifdef __SIZEOF_INT128__
      return (__int128) a * (__int128) b % M;
    #endif

    ll ans = 0;
    while(b) {
      if(b&1) {
        ans += a;
        if(ans >= M) ans -= M;
      }
      b >>= 1;
      a += a;
      if(a >= M) a -= M;
    }
    return ans;
  }

  ll fast_pow(ll a, ll b, ll M) {
    ll ans = 1;
    while(b) {
      if(b&1) ans = modmul(ans, a, M);
      b >>= 1;
      a = modmul(a, a, M);
    }
    return ans;
  }

  bool isProbablePrime(ll n) {
    if(n <= 1) return false;
    ll d = n-1;
    int k = 0;
    while(!(d&1)) {
      k++;
      d >>= 1;
    }

    auto miller = [&](ll x) {
      ll cur = fast_pow(x, d, n);
      if(cur == 1 or cur == n-1) return false;
      for(int tmp = k; --tmp > 0;) {
        cur = modmul(cur, cur, n);
        if(cur == n-1) return false;
```

```
      }
      return true;
    };
    // first 12 prime numbers (deterministic for n <= 2^64)
    for(ll x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
      if(x == n) return true;
      if(miller(x)) return false;
    }
    return true;
  }
};
```

10.13 Sieve

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

vector<int> sieve(int n) {
    vector<char> s(n/2 + 1);
    for (int i = 1; i <= (sqrt(n) + 1) / 2; i++) {
        if (s[i] == 0) {
            for (int j = 2*(i*i + i); j <= (n-1)/2; j += 2*i + 1) {
                s[j] = 1;
            }
        }
    }

    vector<int> primes;
    if (n >= 2) primes.push_back(2);

    for (int i = 1; i <= (n-1) / 2; i ++) {
        if (s[i] == 0) {
            primes.push_back(2*i+1);
        }
    }

    return primes;
}
```

10.14 PrimitiveRoot

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

using ll = long long;

/**
 * A raiz primitiva de N    um inteiro g do grupo multiplicativo de N tal que phi(N)    o primeiro k
     tal que g^k = 1 (mod N)
 * Em outras palavras, g cobre todos os outros elementos do grupo multiplicativo de N.
 * A raiz primitiva de N existe se e somente se alguma das restri  es s o satisfeitas:
 * - N = 1, 2 ou 4
 * - N = p^k, onde p > 2    um primo
 * - N = 2 * p^k, onde p > 2    um primo
 * Se alguma raiz primitiva existir, o n mero de raizes primitivas de N    phi(phi(N)).
 * A complexidade desse algoritmo pode ser at  Nlog(N), mas a complexidade esperada    O(log(N)^6).
 */

namespace PrimitiveRoot {
  int fast_pow(int a, int b, int mod) {
    int ans = 1;
    while(b) {
      if(b&1) ans = 1LL * ans * a % mod;
      a = 1LL * a * a % mod;
      b >>= 1;
    }
    return ans;
  }

  map<int, int> factor(int n) {
    map<int, int> ans;
    for(int i = 2; i * i <= n; ++i) {
      while(n%i == 0) {
        ans[i]++;
        n /= i;
      }
```

```
    }
    if(n > 1) ans[n]++;
    return ans;
  }

  int Phi(int n) {
    auto fact = factor(n);
    int ans = 1;

    for(auto [a, b] : fact) {
      ans *= fast_pow(a, b, n+1) - fast_pow(a, b-1, n+1);
    }
    return ans;
  }

  bool exists(int n) {
    auto fact = factor(n);
    return  n == 1 or n == 2 or n == 4 or
        (fact.size() == 1 and fact.begin()->first > 2) or
        (fact.size() == 2 and fact[2] == 1);
  }

  int find(int n) {
    if(n == 1) return 1;
    if(!exists(n)) return -1;

      int phi = Phi(n);
      auto fact = factor(phi);

    for (int res=2; res<=n; ++res) {
      if(__gcd(res, n) > 1) continue; // nao esta no grupo multiplicativo
      bool ok = true;
      for (auto [a, b] : fact) {
        ok &= fast_pow(res, phi / a, n) != 1;
        if(!ok) break;
      }
      if (ok) return res;
    }
      return -1;
  }

};
```

10.15 FFT

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

using ll = long long;

namespace FFT {
  const double PI = acos(-1);

  struct base { // real + imag * i
    double real, imag;
    base() : real(0), imag(0) {}
    base(double x) : real(x), imag(0) {}
    base(double real, double imag) : real(real), imag(imag) {}
    base operator*(base const& b) { return base(real * b.real - imag * b.imag, real * b.imag + imag
    * b.real); }
    base operator+(base const& b) { return base(real + b.real, imag + b.imag); }
    base operator-(base const& b) { return base(real - b.real, imag - b.imag); }
    base operator/(double const b) { return base(real / b, imag); }
    base& operator*=(base const& b) { return *this = *this * b; }
    base& operator+=(base const& b) { return (real += b.real, imag += b.imag), *this; }
    base& operator-=(base const& b) { return (real -= b.real, imag -= b.imag), *this; }
    base& operator/=(double const b) { return (real /= b), *this; }
  }; // if you don't want to code this struct, simply use `using base=complex<double>;`

  inline void fft(vector<base>& a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
      int bit = n >> 1;
      for (; j >= bit; bit >>= 1) j -= bit;
      j += bit;
      if (i < j) swap (a[i], a[j]);
    }

```

```
    for (int len = 2; len <= n; len <<= 1) {
      double ang = 2 * PI/len * (invert ? -1 : 1);
      base wlen(cos(ang), sin(ang)), w, u, v;
      for (int i = 0; i < n; i += len) {
        w = 1;
        for (int j = 0; j < len/2; j++) {
          u = a[i+j], v = a[i+j+len/2] * w;
          a[i + j] = u + v;
          a[i + j + len/2] = u - v;
          w *= wlen;
        }
      }
    }

    if (invert) for (int i = 0; i < n; i++) a[i] /= n;
  }

  inline vector<ll> multiply(vector<ll> const& a, vector<ll> const& b) {
    vector<base> ca(a.begin(), a.end()), cb(b.begin(), b.end());
    int n = 1;
    while(n < (int) (a.size() + b.size())) n <<= 1;
    ca.resize(n); cb.resize(n);
    fft(ca, false); fft(cb, false);
    for (int i = 0; i < n; i++) ca[i] *= cb[i];
    fft(ca, true);
    vector<ll> ans(n);
    for (int i = 0; i < n; i++) ans[i] = llround(ca[i].real); // if using complex<double>, use .real
    () instead of .real
    return ans;
  }
}

int main() {
  int t; scanf("%d", &t);
  vector<ll> a, b, ans;
  while(t--) {
    int n; scanf("%d", &n);
    a.resize(n + 1), b.resize(n + 1);
    for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
    for (int i = 0; i <= n; i++) scanf("%lld", &b[i]);
    ans = FFT::multiply(a, b);
    for (int i = 0; i < n + n + 1; i++)
      printf("%lld ", ans[i]);
    printf("\n");
  }
}
```

10.16 Matrix

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

using namespace std;
typedef long long ll;

/**
 * If T is floating point, remove all of the occurences of the exact string "% mod" and you are safe
 * If T is int and you require negative values, do the same as above.
 * Otherwise, don't forget to set a mod, cause probably the problem asks you to.
 */
template<typename T, int mod = 1234567890>
struct Matrix {
  vector<vector<T>> mat;
  int n, m;

  Matrix(int n_, int m_): n(n_), m(m_), mat(n_, vector<T>(m_)) {}
  Matrix(vector<vector<T>> v): n(v.size()), m(v[0].size()), mat(v) {}
  Matrix(initializer_list<vector<T>> l): mat(l), n(l.size()), m(l.begin()->size()) {}

  vector<T>& operator[](int i) const {
    return const_cast<vector<T>&>(mat[i]);
  }

  // Sum two matrices
  Matrix operator+(const Matrix& b) const {
    if(n != b.n or m != b.m) throw runtime_error("Matrixes have different dimensions");

    Matrix ans(n, m);
    for(int i = 0; i < n; ++i) {
```

```
30      for(int j = 0; j < m; ++j) {
31        ans[i][j] = (mat[i][j] + b[i][j]) % mod;
32      }
33    }
34    return ans;
35  }
36
37  // Sum two matrices
38  Matrix operator-(const Matrix& b) const {
39    if(n != b.n or m != b.m) throw runtime_error("Matrixes have different dimensions");
40
41    Matrix ans(n, m);
42    for(int i = 0; i < n; ++i) {
43      for(int j = 0; j < m; ++j) {
44        ans[i][j] = mat[i][j] - b[i][j];
45        if(mod != 1234567890 and ans[i][j] < 0) ans[i][j] = ans[i][j] + mod;
46      }
47    }
48    return ans;
49  }
50
51  // Multiply two matrices
52  Matrix operator*(const Matrix& b) const {
53    if(m != b.n) throw runtime_error("Can't multiply: matrixes' dimensions are not compatible");
54
55    Matrix ans(n, b.m);
56    for(int i = 0; i < n; ++i) {
57      for(int j = 0; j < b.m; ++j) {
58        for(int k = 0; k < m; ++k) {
59          ans[i][j] = (ans[i][j] + 1ll * mat[i][k] * b[k][j] % mod) % mod;
60        }
61      }
62    }
63    return ans;
64  }
65
66  // multiply by column vector
67  vector<T> operator*(const vector<T>& col) const {
68    if(m != col.size()) throw runtime_error("Can't multiply: matrixes' dimensions are not compatible
    ");
69    vector<T> ans(m);
70    for(int i = 0; i < n; ++i) {
71      ans[i] = T(0);
72      for(int j = 0; j < m; ++j) {
73        ans[i] = (ans[i] + 1ll * mat[i][j] * col[j] % mod) % mod;
74      }
75    }
76    return ans;
77  }
78
79  // multiply by scalar
80  Matrix operator*(const T& x) const {
81    Matrix ans(n, m);
82    for(int i = 0; i < n; ++i) {
83      for(int j = 0; j < m; ++j) {
84        ans[i][j] = 1ll * mat[i][j] * x % mod;
85      }
86    }
87    return ans;
88  }
89
90  // return k-th power
91  Matrix operator^(ll k) {
92    if(n != m) throw runtime_error("Cant exponentiate a non-square matrix");
93
94    Matrix ans = identity(n);
95    Matrix aux(mat);
96    while(k) {
97      if(k&1) ans = ans * aux;
98      aux = aux * aux;
99      k >>= 1;
100   }
101   return ans;
102 }
103
104 bool operator== (const Matrix& a) const {
105   if(n != a.n or m != a.m) return false;
106   bool ans = true;
107   for(int i = 0; i < n && ans; ++i) ans &= mat[i] == a[i];
```

```
108     return ans;
109   }
110
111   // Get transposed
112   Matrix transposed() {
113     Matrix ans(m, n);
114     for(int i = 0; i < n; ++i) {
115       for(int j = 0; j < m; ++j) {
116         ans[j][i] = mat[i][j];
117       }
118     }
119     return ans;
120   }
121
122   /** For double, if you're using Z/mod you have to customize
123     * Returns the rank of the matrix (the number of column vectors that are linearly independent)
124     * O(N*M*M), I think
125     */
126   int rank() {
127     const double EPS = 1e-9;
128     int rank = 0;
129     vector<bool> row_selected(n, false);
130
131     Matrix A = *this;
132
133     for (int i = 0; i < m; ++i) {
134       int j;
135       for (j = 0; j < n; ++j) {
136         if (!row_selected[j] && abs(A[j][i]) > EPS)
137           break;
138       }
139
140       if (j != n) {
141         ++rank;
142         row_selected[j] = true;
143         for (int p = i + 1; p < m; ++p)
144           A[j][p] = A[j][p] / A[j][i];
145         for (int k = 0; k < n; ++k) {
146           if (k != j && abs(A[k][i]) > EPS) {
147             for (int p = i + 1; p < m; ++p)
148               A[k][p] = A[k][p] - A[j][p] * A[k][i];
149           }
150         }
151       }
152     }
153     return rank;
154   }
155
156   friend ostream& operator<<(ostream& os, const Matrix& mat) {
157     for(int i = 0; i < mat.n; i++) {
158       for(int j = 0; j < mat.m; ++j) {
159         cout << mat[i][j] << ' ';
160       }
161       cout << '\n';
162     }
163     return os;
164   }
165
166   /* -------------------- Static functions ------------------- */
167   static Matrix identity(int n) {
168     Matrix m(n, n);
169     for(int i = 0; i < n; ++i) m[i][i] = 1;
170     return m;
171   }
172
173   static Matrix zeros(int n, int m) {
174     return Matrix(n, m);
175   }
176
177   /** Freivalds computes if a * b = c, with probability at least (1 - (1/2)^k) of certainty
178     * a and b must be square matrices of same dimension
179     * Complexity: O(k * n^2, where k = number of steps
180     */
181   static bool freivald(const Matrix& a, const Matrix& b, const Matrix& c, int k = 50) {
182     if(a.n != a.m or b.n != b.m or c.n != c.m) throw runtime_error("All matrices must be square
      matrices");
183     if(a.n != b.n or b.n != c.n) throw runtime_error("All matrices must have the same dimension");
184
185     int n = a.n;
```

```
    bool ans = true;
    while(k-- && ans) {
      vector<T> aux(n);
      for(int i = 0; i < n; ++i) {
        aux[i] = T(rand()%2);
      }

      vector<T> ret1 = a * (b * aux);
      vector<T> ret2 = c * aux;
      ans &= ret1 == ret2;
    }
    return ans;
  }
};
```

10.17 LagrangeInterpolation

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

#define pb push_back

typedef long long ll;
using namespace std;

/**
 * Faz a interpola  o de lagrange de um polinomio P(x), por m  n o encontra os coeficientes do polinomio
 * Se P(x) tem grau deg, e tenho P(x_0), P(x_1), ..., P(x_deg), consigo calcular um P(x) arbitrario em O(deg^2 lg(deg))
 * Se x_0, x_1, ..., x_deg formarem uma P.A., d  pra calcular em O(deg)
 * x_0, x_1, ..., x_deg devem ser todos distintos
 * Repare que deg    o grau do polinomio, mas o polinomio tem deg+1 termos!!
 * Fonte: https://www.notion.so/turci/Lagrange-Interpolation-2c1b2465899146d2a9dd17bf45f16e11
 */
struct Lagrange {
  int deg;
  vector<ll> X, Y;

  // deg = grau do polinomio. O numero de termos do polinomio (e o numero de samples necessario) eh deg + 1
  Lagrange(int deg_ = 0): deg(deg_) {
    X.reserve(deg_ + 1);
    Y.reserve(deg_ + 1);
  }

  /**
   * Adiciona o sample (x, y), onde P(x) = y
   * Todos os x devem ser distintos
   * Assume que 0 <= x < mod e 0 <= y < mod
   */
  void add_sample(ll x, ll y) {
    X.pb(x), Y.pb(y);
  }

  ll fast_pow(ll a, ll b, int mod) {
    ll ans = 1;
    while(b) {
      if(b&1) ans = ans * a % mod;
      b >>= 1;
      a = a * a % mod;
    }
    return ans;
  }

  // Calcula P(x) em O(deg^2 * lg(mod))
  ll solve(ll x, int mod) {
    if(X.size() != deg + 1) throw runtime_error("Sem samples suficientes (deg + 1 != #samples)");

    ll ans = 0;
    for(int i = 0; i <= deg; ++i) {
      ll p = Y[i];
      for(int j = 0; j <= deg; ++j) {
        if(j == i) continue;
        p = p * (x - X[j] + mod) % mod;
        p = p * fast_pow(X[i] - X[j] + mod, mod - 2, mod) % mod;
      }
      ans = (ans + p) % mod;
    }
```

```
    return ans;
  }

  // Calcula P(x) em O(deg), se os X[0], X[1], ... X[deg] formarem uma progressao aritmetica
  ll solve_pa(ll x, int mod) {
    if(X.size() != deg + 1) throw runtime_error("Sem samples suficientes (deg + 1 != #samples)");
    if(deg == 0) return X[0];

    const ll d = (X[1] - X[0] + mod) % mod;
    const ll d_n = fast_pow(d, 1LL * deg * (mod - 2), mod);

    // calcula fatorial inverso
    vector<ll> inv_fat(deg+1);
    inv_fat[0] = 1;
    for(int i = 1; i <= deg; ++i) inv_fat[i] = inv_fat[i-1] * i % mod;
    inv_fat[deg] = fast_pow(inv_fat[deg], mod - 2, mod);
    for(int i = deg-1; i >= 0; --i) inv_fat[i] = inv_fat[i+1] * (i+1) % mod;

    vector<ll> pref(deg + 1), suf(deg + 1);

    ll cur = 1;
    for(int i = 0; i <= deg; ++i) {
      cur = cur * (x - X[i] + mod) % mod;
      pref[i] = cur;
    }

    cur = 1;
    for(int i = deg; i >= 0; --i) {
      cur = cur * (x - X[i] + mod) % mod;
      suf[i] = cur;
    }

    ll ans = 0;
    for(int i = 0; i <= deg; ++i) {
      ll aux = (i? pref[i-1] : 1) * (i < deg? suf[i + 1] : 1) % mod;
      aux = aux * d_n % mod;
      aux = aux * inv_fat[i] % mod;
      aux = aux * inv_fat[deg - i] % mod;
      aux = aux * Y[i] % mod;
      if((deg - i)%2) aux = mod-aux;
      ans = (ans + aux) % mod;
    }

    return ans;
  }
};
```

10.18 PointsUnderLine

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

using namespace std;

/*
  PointsUnderLine::calc(p, q, n) calcula:
    Soma { floor(p * i / q) } para i=1..n

  Complexidade: log(min(p, q))
  Ideia: Pensando no resultado de calc(p, q, n) como a area hachurada de um retangulo de largura n e
     altura p * n,
  escreva calc(p, q, n) como  f u n   o de g(p, q, n), em que g(p, q, n) calcula a  r e a hachurada se
    este retangulo
  for rotacionado em 90   no sentido anti-horario
*/

namespace PointsUnderLine {
  ll g(ll p, ll q, ll n) {
      return p == 0? 0 : (n * gcd(p, q))/q;
  }

  ll calc(ll p, ll q, ll n) {
      if(p == 0 or q == 0 or n == 0) return 0;
      return (p/q) * ((n *(n+1))/2) + (((p%q)*n)/q)*n - calc(q, p%q, ((p%q)*n)/q) + g(p%q, q, n);
  }
};
```