Team Note of ATRI

Ji Chan Kim, Seong Min Jeon, Myeong Geun Hwang

Compiled on October 10, 2025

Contents

1 Data Structure

1.1 DSU

Usage: call init() before use

```
// ACL(Atcoder Library) Template - Disjoint set
struct dsu {
  public:
    dsu() : _n(0) {}
    explicit dsu(int n) : _n(n), psz(n, -1) {}
    int merge(int a, int b) {
      int x = leader(a), y = leader(b);
      if (x == y) return x;
      if (-psz[x] < -psz[y]) swap(x, y);
      psz[x] += psz[y];
```

```
      psz[y] = x;
      return x;
    }
    bool same(int a, int b) { return leader(a) == leader(b); }
    int leader(int a) { return (psz[a] < 0 ? a : psz[a] = leader(psz[a])); }
    int size(int a) { return -psz[leader(a)]; }
    vector<vector<int>> groups() {
      vector<int> buf(_n), gsz(_n);
      for (int i = 0; i < _n; i++) {
        buf[i] = leader(i);
        gsz[buf[i]]++;
      }
      vector<vector<int>> result(_n);
      for (int i = 0; i < _n; i++) result[i].reserve(gsz[i]);
      for (int i = 0; i < _n; i++) result[buf[i]].push_back(i);
      result.erase(remove_if(result.begin(), result.end(), [&](const vector<int>& v) {
      return v.empty(); }), result.end());
      return result;
    }
  private:
    int _n;
    vector<int> psz;
};
```

1.2 Fenwick

```
ll fenwick[N+1];
void update(int i, ll diff) {
  while (i <= N) {
    a[i] += diff;
    i += (i & -i);
  }
}
// ex. sum
ll query(int i) {
  ll r = 0;
  while (i) {
    r += a[i];
    i -= (i & -i);
  }
  return r;
}
```

1.3 Line Container(cht)

```
// cht, MinLineContainer<ll> 처럼 사용, ax+b 꼴 점화식
enum Objective {
  MAXIMIZE = +1,
  MINIMIZE = -1,
};
template <typename T>
struct Line {
  mutable T k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(T x) const { return p < x; }
};
template <typename T>
T lc_inf() { return numeric_limits<T>::max(); }
template <> long double lc_inf<long double>() { return 1 / .0; }
template <typename T> T lc_div(T a, T b) { return a / b - ((a ^ b) < 0 and a % b); }
template <> long double lc_div(long double a, long double b) { return a / b; };
template <typename T, Objective objective>
struct LineContainer : multiset<Line<T>, less<>> {
  using super = multiset<Line<T>, less<>>;
  using super::begin, super::end, super::insert, super::erase;
  using super::empty, super::lower_bound;
  const T inf = lc_inf<T>();
  bool insect(typename super::iterator x, typename super::iterator y) {
    if (y == end()) return x->p = inf, false;
    if (x->k == y->k) x->p = (x->m > y->m ? inf : -inf);
    else x->p = lc_div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(T k, T m) {
    auto z = insert({k * objective, m * objective, 0}), y = z++, x = y;
    while (insect(y, z)) z = erase(z);
    if (x != begin() and insect(--x, y)) insect(x, y = erase(y));
    while ((y = x) != begin() and (--x)->p >= y->p) insect(x, erase(y));
  }
  T query(T x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return (l.k * x + l.m) * objective;
  }
};
template <typename T>
using MinLineContainer = LineContainer<T, Objective::MINIMIZE>;
template <typename T>
using MaxLineContainer = LineContainer<T, Objective::MAXIMIZE>;
```

1.4 Matrix

```
ll n, s, e;
using Matrix = vector<vector<ll>>;
Matrix matrix;
Matrix operator* (const Matrix &op1, const Matrix &op2) {
  Matrix result(op1.size(), vector<ll>(op1[0].size()));
  for (int i = 0; i < 5*n; i++) {
    for (int j = 0; j < 5*n; j++) {
      for (int k = 0; k < 5*n; k++) {
        result[i][j] += op1[i][k] * op2[k][j];
        result[i][j] %= DIV;
      }
    }
  }
  return result;
}
Matrix matrix_pow(ll k) {
  if (k == 1) return matrix;
  else if (k % 2) return matrix_pow(k-1) * matrix;
```

```
  else {
    Matrix _matrix = matrix_pow(k/2);
    return _matrix * _matrix;
  }
}
int main() {
  Matrix result;
  ll k;
  scanf("%lld %lld %lld %lld", &n, &s, &e, &k);
  for (int i = 0; i < n; i++) {
    vector<ll> v;
    int x;
    for (int j = 0; j < n; j++) {
      scanf("%1d", &x);
      for (int d = 5; d > 0; d--) {
        if (d == x) v.push_back(1);
        else v.push_back(0);
      }
    }
    for (int d = 0; d < 4; d++) {
      vector<ll> temp;
      for (int ti = 1; ti <= 5*n; ti++) {
        if (5*i + d + 1 == ti - 1) temp.push_back(1); else temp.push_back(0);
      }
      matrix.push_back(temp);
    }
    matrix.push_back(v);
  }
  result = matrix_pow(k);
}
```

1.5 PBDS(ordered set)

```
// 자료형만 수정, less_equal <-> less 변경시 주의
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type, less_equal<int>,
rb_tree_tag,tree_order_statistics_node_update>
// less_equal, greater_equal 사용시 필요
void m_erase(ordered_set &os, int val){
  int idx = osOS.order_of_key(val);
  ordered_set::iterator it = os.find_by_order(idx);
  if(*it == val) os.erase(it);
}
```

1.6 Rope and Crope

```
__gnu_cxx::crope s; // string
__gnu_cxx::rope<int> r; // template
insert(pos, string)
substr(pos, len)
erase(pos, len)
```

1.7 Lazy Reversible BBST(Splay)

```
// ref: nachia library, like ACL seg, lazy
struct SplayTreeByIdx{
  // ACL segと　ほぼ同じ
  struct S { ll l, r, ans, len; }; // affine monoid
  static S op(S l, S r) {
    ll nl = (l.l == l.len ? l.len+r.l : l.l);
    ll nr = (r.r == r.len ? r.len+l.r : r.r);
    ll len = l.len+r.len;
    return { nl, nr, max({nl, nr, l.ans, r.ans, l.r+r.l}), len };
  }
  static S e() { return { 0, 0, 0, 0 }; }
  static void reverse_prod(S& x) { swap(x.l, x.r); } // 反時に prod を更新する
  struct F { ll x; };
  static S mapping(F f, S x) { return x; }
  static F composition(F f, F x) { return x; }
  static F id() { return {0}; }
  struct Node{
    Node *l = 0, *r = 0, *p = 0;
    S a = e(); // 頂点が持つ値
    S prod = e(); // 集約( rev==1 のときでも reverse_prod 作用み)
    F f = id(); // 延(a,prod には作用み)
    int i = -1; // 配列の index
    int z = 0; // 頂点の重み ( NIL なら 0 、普通の頂点は 1 )
    int sumz = 0; // 部分木の重み
    int rev = 0; // 反の延
  };
  using pNode = unique_ptr<Node>;
  pNode pNIL;
  Node *NIL = nullptr;
  vector<pNode> A;
  Node *R;
  SplayTreeByIdx() {
    if(!pNIL){
      pNIL = make_unique<Node>();
      NIL = pNIL.get();
      NIL->l = NIL->r = NIL->p = NIL;
      R = NIL;
    }
  }
  // 播
  void prepareDown(Node* c){
    if(c->l != NIL) {
      // a,prod への作用を忘れずに
      c->l->a = mapping(c->f, c->l->a);
      c->l->prod = mapping(c->f, c->l->prod);
      c->l->f = composition(c->f, c->l->f);
    }
    if(c->r != NIL) {
      // a,prod への作用を忘れずに
      c->r->a = mapping(c->f, c->r->a);
      c->r->prod = mapping(c->f, c->r->prod);
      c->r->f = composition(c->f, c->r->f);
    }
```

```
  if(c->rev) {
    swap(c->l, c->r);
    if(c->l != NIL) {
      c->l->rev ^= 1; // 播
      reverse_prod(c->l->prod);
    }
    if(c->r != NIL) {
      c->r->rev ^= 1; // 播
      reverse_prod(c->r->prod);
    }
    c->rev = 0;
  }
  c->f = id(); // 播み
}
// 集約
void prepareUp(Node* c) {
  c->sumz = c->l->sumz + c->r->sumz + 1; // 部分木の重み
  c->prod = op(op(c->l->prod,c->a),c->r->prod); // 集約  c は播み
}
// (便利)
// p の親が、子として p を照するので、それを書き換えられるようにする
// 根の場合に張して R の照を返す。
Node*& parentchild(Node* p) {
  if(p->p == NIL) return R;
  if(p->p->l == p) return p->p->l;
  else return p->p->r;
}
// 左回
void rotL(Node* c) {
  Node* p = c->p;
  parentchild(p) = c;
  c->p = p->p;
  p->p = c;
  if(c->l != NIL) c->l->p = p; // 子が NIL かもしれない
  p->r = c->l;
  c->l = p;
}
// 右回
void rotR(Node* c) {
  Node* p = c->p;
  parentchild(p) = c;
  c->p = p->p;
  p->p = c;
  if(c->r != NIL) c->r->p = p; // 子が NIL かもしれない
  p->l = c->r;
  c->r = p;
}
// splay 後、c は播み
void splay(Node* c) {
  prepareDown(c); // ループが回らない時のために
  while(c->p != NIL) {
    Node* p = c->p;
    Node* pp = p->p;
    // 播は親から
    if (pp != NIL) prepareDown(pp);
    if (p != NIL) prepareDown(p);
    prepareDown(c);
    if (p->l == c) {
      if(pp == NIL) { rotR(c); }
      else if(pp->l == p) { rotR(p); rotR(c); }
      else if(pp->r == p) { rotR(c); rotL(c); }
    }
    else {
      if(pp == NIL) { rotL(c); }
      else if(pp->r == p) { rotL(p); rotL(c); }
      else if(pp->l == p) { rotL(c); rotR(c); }
    }
    // 集約は子から
    if(pp != NIL) prepareUp(pp);
    if(p != NIL) prepareUp(p);
    prepareUp(c);
  }
  prepareUp(c); // ループが回らない時のために
}
Node* kth_element(int k) {
  Node* c = R;
  while(true) {
    prepareDown(c);
    if(c->l->sumz == k) break;
    if(c->l->sumz > k){ c = c->l; continue; }
    k -= c->l->sumz + 1;
    c = c->r;
  }
  prepareDown(c);
  splay(c);
  return c;
}
void insert_at(int k, S x) {
  pNode pnx = make_unique<Node>(*NIL);
  Node* nx = pnx.get();
  nx->z = nx->sumz = 1;
  nx->i = A.size();
  nx->a = nx->prod = x;
  A.emplace_back(move(pnx));
  if(k == 0) {  // 左端
    nx->r = R;
    if(R != NIL) R->p = nx; // 元 0 頂点かもしれない
    R = nx;
    prepareUp(nx); // 入したら集約
    return;
  }
  if(k == R->sumz) { // 右端(左端と同)
    nx->l = R;
    if(R != NIL) R->p = nx;
    R = nx;
    prepareUp(nx);
    return;
  }
  auto p = kth_element(k);
  nx->l = p->l;
```

```
  nx->r = p;
  R = nx;
  p->l->p = nx;
  p->p = nx;
  p->l = NIL;
  prepareUp(p);  // split/merge の影響
  prepareUp(nx); //
}
void erase_at(int k) {
  auto p = kth_element(k);
  if(k == 0) { // 左端
    R = p->r;
    if(R != NIL) R->p = NIL; // 0 頂点になるかもしれない
  }
  else if(k == R->sumz-1) { // 右端
    R = p->l;
    if(R != NIL) R->p = NIL;
  }
  else {
    auto l = p->l;
    auto r = p->r;
    r->p = NIL;   // split
    R = r;        //
    kth_element(0);
    r = R;        // merge
    r->l = l;     //
    l->p = r;     //
    prepareUp(r); // split/merge の影響
  }
  swap(p->i,A.back()->i); // index が更新されるよ
  swap(A[p->i],A[A.back()->i]); // 後ろに移動して
  A.pop_back(); // 削除
}
Node* between(int l, int r) {
  if(l == 0 && r == R->sumz) return R; // 全域
  if(l == 0) return kth_element(r)->l; // 左端
  if(r == R->sumz) return kth_element(l-1)->r; // 右端
  auto rp = kth_element(r);
  auto lp = rp->l;
  R = lp;        // split
  lp->p = NIL;   //
  lp = kth_element(l-1);
  R = rp;        // merge
  rp->l = lp;    //
  lp->p = rp;    //
  prepareUp(rp); // split/merge の影響
  return lp->r;
}
void reverse(int l, int r) {
  auto c = between(l,r);
  c->rev ^= 1;
  reverse_prod(c->prod);
  splay(c);
}
void apply(int l, int r, F f) {
  auto c = between(l,r);
  c->a = mapping(f,c->a);
  c->prod = mapping(f,c->prod);
  c->f = composition(f,c->f);
  splay(c);
}
S prod(int l, int r) {
  return between(l,r)->prod;
}
}; // end of struct SplayTreeByIdx
SplayTreeByIdx splay; // insert_at := i-thへデータ投入(splay node)
```

1.8 Segment Tree

```
template <class S, auto op, auto e> struct segtree {
  public:
    segtree(int n = 0) : segtree(vector<S>(n, e())) {}
    segtree(const vector<S>& v) : _n(v.size()) {
      sz = 1;
      while (sz < _n) sz <<= 1;
      d.assign(2 * sz, e());
      for (int i = 0; i < _n; i++) d[sz + i] = v[i];
      for (int i = sz - 1; i >= 1; i--) update(i);
    }
    void set(int p, S x) {
      p += sz;
      d[p] = x;
      for (int i = 1; i <= __builtin_ctz(sz); i++) update(p >> i);
    }
    S get(int p) { return d[p + sz]; }
    S prod(int l, int r) {
      S sml = e(), smr = e();
      l += sz;
      r += sz;
      while (l < r) {
        if (l & 1) sml = op(sml, d[l++]);
        if (r & 1) smr = op(d[--r], smr);
        l >>= 1;
        r >>= 1;
      }
      return op(sml, smr);
    }
    S all_prod() { return d[1]; }
  private:
    int _n, sz;
    vector<S> d;
    void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
};
```

1.9 Lazy Segment Tree

```
template <class S, auto op, auto e, class F, auto mapping, auto composition, auto id>
struct lazy_segtree {
  public:
    lazy_segtree(int n = 0) : lazy_segtree(vector<S>(n, e())) {}
```

```
  lazy_segtree(const vector<S>& v) : _n(v.size()) {
    sz = 1;
    while (sz < _n) sz <<= 1;
    lg = __builtin_ctz(sz);
    d.assign(2 * sz, e());
    lz.assign(sz, id());
    for (int i = 0; i < _n; i++) d[sz + i] = v[i];
    for (int i = sz - 1; i >= 1; i--) update(i);
  }
  void set(int p, S x) {
    p += sz;
    for (int i = lg; i >= 1; i--) push(p >> i);
    d[p] = x;
    for (int i = 1; i <= lg; i++) update(p >> i);
  }
  S get(int p) {
    p += sz;
    for (int i = lg; i >= 1; i--) push(p >> i);
    return d[p];
  }
  S prod(int l, int r) {
    if (l == r) return e();
    l += sz, r += sz;
    for (int i = lg; i >= 1; i--) {
      if (((l >> i) << i) != l) push(l >> i);
      if (((r >> i) << i) != r) push((r - 1) >> i);
    }
    S sml = e(), smr = e();
    while (l < r) {
      if (l & 1) sml = op(sml, d[l++]);
      if (r & 1) smr = op(d[--r], smr);
      l >>= 1, r >>= 1;
    }
    return op(sml, smr);
  }
  S all_prod() { return d[1]; }
  void apply(int p, F f) {
    p += sz;
    for (int i = lg; i >= 1; i--) push(p >> i);
    d[p] = mapping(f, d[p]);
    for (int i = 1; i <= lg; i++) update(p >> i);
  }
  void apply(int l, int r, F f) {
    if (l == r) return;
    l += sz, r += sz;
    for (int i = lg; i >= 1; i--) {
      if (((l >> i) << i) != l) push(l >> i);
      if (((r >> i) << i) != r) push((r - 1) >> i);
    }
    int l2 = l, r2 = r;
    while (l < r) {
      if (l & 1) all_apply(l++, f);
      if (r & 1) all_apply(--r, f);
      l >>= 1, r >>= 1;
    }
    l = l2, r = r2;
    for (int i = 1; i <= lg; i++) {
      if (((l >> i) << i) != l) update(l >> i);
      if (((r >> i) << i) != r) update((r - 1) >> i);
    }
  }
 private:
  int _n, sz, lg;
  vector<S> d;
  vector<F> lz;
  void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
  void all_apply(int k, F f) {
    d[k] = mapping(f, d[k]);
    if (k < sz) lz[k] = composition(f, lz[k]);
  }
  void push(int k) {
    all_apply(2 * k, lz[k]);
    all_apply(2 * k + 1, lz[k]);
    lz[k] = id();
  }
};
```

2 Monoid

2.1 RMQ and RNQ

```
S op(S a, S b){ return min(a, b); } // or max
S e(){ return INF; }
S mapping(F f, S x){ return f+x; }
F composition(F f, F g){ return f+g; }
F id(){ return 0; }
```

2.2 RMQ and RUQ

```
S op(S a, S b){ return min(a, b); } // or max
S e(){ return INF; }
S mapping(F f, S x){ return (f == ID ? x : f); }
F composition(F f, F g){ return (f == ID ? g : f); }
F id(){ return ID; }
```

2.3 RSQ and RNQ

```
struct S { ll x; int sz; };
S op(S a, S b){ return {a.x+b.x, a.sz+b.sz}; }
S e(){ return {0, 0}; }
S mapping(F f, S x){ return {x.x + f*x.x, x.sz}; }
F composition(F f, F g){ return f+g; }
F id(){ return 0; }
```

2.4 RSQ and RUQ

```
struct S { ll x; int sz; };
S op(S a, S b) { return {a.x+b.x, a.sz+b.sz}; }
S e() { return {0, 0}; }
S mapping(F f, S x) { if(f != ID) x.x = f*x.sz; return x; }
F composition(F f, F g) { return (f == ID ? g : f); }
F id() { return ID; }
```

2.5 Maximum Array(Struct Segment)

```
struct S {
  bool check;
  int l, r, sum, len;
  S(int x = 0, int c = false): sum(0) { l = r = x; len = 1; check = c; }
  S(int l, int r, int sum, int len, int c): l(l), r(r), sum(sum), len(len) { check = c; }
};
using F = int;
const F ID = -1;
S op(S a, S b) {
  if (!a.check || !b.check) {
    if (!a.check && !b.check) return S();
    if (!a.check) return b;
    else if (!b.check) return a;
  }
  return S(a.l, b.r, a.sum+b.sum+(a.r==b.l), a.len+b.len, true);
}
S e() { return S(); }
S mapping(F f, S x) {
  if (f == ID) return x;
  return S(f, f, x.len-1, x.len, true);
}
F composition(F f, F g) { return (f == ID ? g : f); }
F id() { return ID; }
```

3 DP

3.1 Doubling(Sparse Table)

```
// Tree 형태에서, 어떤 값의 조상...
int n, f[200000], dp[32][200000];
void doubling() {
  for (int j = 0; j < n; j++) dp[0][j] = f[j];
  for (int i = 1; i < 31; i++) {
    for (int j = 0; j < n; j++) {
      dp[i][j] = dp[i-1][dp[i-1][j]];
    }
  }
}
```

3.2 Streetlight(prefix sum + dp)

```
int n, m;
ll x[1002], w[1002], wsum[1002], dp[1002][1002][2];
int main() {
  cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    cin >> x[i] >> w[i];
    wsum[i] = wsum[i-1] + w[i];
  }
  for (int i = 0; i <= 1001; i++) for (int j = 0; j <= 1001; j++) dp[i][j][0] = dp[i][j][1]
  = 1'000'000'000'000'000LL;
  dp[m][m][0] = dp[m][m][1] = 0;
  for (int len = 1; len <= n; len++) {
    for (int l = max(1, m-len+1); l <= m; l++) {
      int r = l+len-1;
      if (r > n) continue;
      if (l-1 >= 1) dp[l-1][r][0] = min({dp[l-1][r][0],
      dp[l][r][0]+(wsum[n]-wsum[r]+wsum[l-1])*(x[l]-x[l-1]),
      dp[l][r][1]+(wsum[n]-wsum[r]+wsum[l-1])*(x[r]-x[l-1])});
      if (r+1 <= n) dp[l][r+1][1] = min({dp[l][r+1][1],
      dp[l][r][0]+(wsum[n]-wsum[r]+wsum[l-1])*(x[r+1]-x[l]),
      dp[l][r][1]+(wsum[n]-wsum[r]+wsum[l-1])*(x[r+1]-x[r])});
    }
  }
  cout << min(dp[1][n][0], dp[1][n][1]);
}
```

4 Geometry

4.1 vector2

```
double pi = acos(-1);
template<class T>
struct vector2 {
  T x, y; // double or ll, dot, cross, fix req
  explicit vector2(T _x = 0, T _y = 0): x(_x), y(_y)
  {}
  vector2 operator + (const vector2 &rhs) const {
    return vector2(x + rhs.x, y + rhs.y);
  }
  vector2 operator - (const vector2 &rhs) const {
    return vector2(x - rhs.x, y - rhs.y);
  }
  vector2 operator * (const double k) const {
    return vector2(k*x, k*y);
  }
  bool operator == (const vector2 &rhs) const {
    return x == rhs.x && y == rhs.y;
  }
  bool operator < (const vector2 &rhs) const {
    return x != rhs.x ? x < rhs.x : y < rhs.y;
  }
  T dot(const vector2 &rhs) const {
    return x*rhs.x + y*rhs.y;
  }
  T cross(const vector2 &rhs) const {
    return x * rhs.y - y * rhs.x;
  }
  double norm() const {
    return hypot(x, y);
  }
  T norm2() const {
    return x*x+y*y;
  }
  vector2 basis() const {
    double d = norm(); // req T=double
    return vector2(x/d, y/d);
  }
  vector2 reverse() const {
    return vector2(-1*x, -1*y);
  }
};
int doubleCompare(double x, double y, double eps = 1e-8) {
  if (fabs(x-y) <= eps) return 0; // 0 is true
  return (x-y > 0) ? 1 : -1;
}
ll ccw(vector2 v1, vector2 v2, vector2 v3) {
  if (doubleCompare((v2-v1).cross(v3-v2), 0) == 0) return 0;
  else return (v2-v1).cross(v3-v2); // 1(>0), -1(<0)
}
bool pointYCompare(const vector2 &a, const vector2 &b) {
  return a.x != b.x ? a.x < b.x : a.y < b.y;
```

```
}
bool pointThetaCompare(const vector2 &a, const vector2 &b) {
  if (ccw(vector2(), a, b) == 0) return a.norm() < b.norm();
  else return ccw(vector2(), a, b) > 0;
}
double safe_acos(double a) {
  if (a <= -1.0) return pi; // req acos(-1) = pi
  else if (a >= 1.0) return 0;
  else return acos(a);
}
// like cross
double f(vector2 p1, vector2 q1, vector2 p2, vector2 q2) {
  return ((p2-p1).cross(q2-p2) / (q1-p1).cross(q2-p2));
}
double k = f(p1, q1, p2, q2); // k배
double cx = p1.x + k*(p2.x-p1.x), cy = p1.y + k*(p2.y-p1.y); // p1->p1, p2->q2 교차점
```

4.2 vector3

```
// fix vector2 -> 3, this template only v3
vector3 cross(const vector3 &rhs) const {
  return vector3(y*rhs.z-z*rhs.y, z*rhs.x-x*rhs.z, x*rhs.y-y*rhs.x);
}
double norm() const {
  return sqrt(x*x + y*y + z*z);
}
```

4.3 Line Intersection

```
// req vector2
struct line2 {
  vector2 p1, p2;

  void setLeftRight() {
    if (p2 < p1) {
      vector2 temp = p2;
      p2 = p1;
      p1 = temp;
    }
  }
};
int line_intersection(vector2 s1, vector2 e1, vector2 s2, vector2 e2) {
  if (e1 < s1) swap(s1, e1);
  if (e2 < s2) swap(s2, e2);
  int c1s = iccw(s1, e1, s2), c1e = iccw(s1, e1, e2),
    c2s = iccw(s2, e2, s1), c2e = iccw(s2, e2, e1);

  if (c1s * c1e == -1 && c2s * c2e == -1) return 2;
  else if (c1s == 0 && c1e == 0 && c2s == 0 && c2e == 0) {
    if (s1.x == e1.x) {
      // 세로 두 직선
      if (s1.y > s2.y) swap(s1, s2), swap(e1, e2);
      if (e1.y < s2.y) return 0;
      if (doubleCompare(e1.y, s2.y) == 0) return 1;
    } else {
      // 가로 두 직선
      if (s1.x > s2.x) swap(s1, s2), swap(e1, e2);
      if (e1.x < s2.x) return 0;
      if (doubleCompare(e1.x, s2.x) == 0) return 1;
    }
    return 3;
    if (s1.x > s2.x) swap(s1, s2), swap(e1, e2);
    if (e1.x < s2.x) return 0;
    if (s1.y > s2.y) swap(s1, s2), swap(e1, e2);
    if (e1.y < s2.y) return 0;
    if (e1.y == s2.y) return 1;
    if (s1.x > s2.x) swap(s1, s2), swap(e1, e2);
    if (e1.x == s2.x) return 1;
    return 3;
  } else if (c1s * c1e * c2s * c2e == 0) {
    if ((c1s * c1e == 0 && c2s * c2e == 1)
      || (c1s * c1e == 1 && c2s * c2e == 0)
    ) return 0; else return 1;
  } else return 0;
}
double f(vector2 p1, vector2 q1, vector2 p2, vector2 q2) {
  return ((p2-p1).cross(q2-p2) / (q1-p1).cross(q2-p2));
}
// 교차점 계산
double w = f(v[i], v[j], v[k], v[nk]);
double cx = v[i].x + w*(v[j].x-v[i].x), cy = v[i].y + w*(v[j].y-v[i].y);
// 반사
vector2 p, np, fp;
p.input();
double k = (e-s).dot(p-s)/(e-s).get_size();
np = s + (e-s).basis()*k;
fp = p+(np-p)*2; // x1의 경우 사영
cout << fp.x << " " << fp.y << "\n";
```

4.4 Convex Hull

```
// cps: 컨헐집합, [0]이 제일 외곽, O(nlogn)
vector<vector2> p;
vector<vector<vector2>> cps;
int n, check[1050], ci = 1, cnt = 0;
double oa = 0, ia = 0;
int main () {
  cin >> n;
  p = vector<vector2>(n);
  for (int i = 0; i < n; i++) {
    check[i] = 0;
    cin >> p[i].x >> p[i].y;
    p[i].i = i;
  }
  while (p.size() >= 3) {
    sort(p.begin(), p.end(), pointYCompare);
    vector2 save = p.front();
    for (int i = 1; i < p.size(); i++) {
      p[i].x = p[i].x - p[0].x;
      p[i].y = p[i].y - p[0].y;
```

```
    }
    sort(next(p.begin()), p.end(), pointThetaCompare);
    vector<vector2> np, cp;
    p[0] = vector2();
    p[0].i = save.i;
    cp.push_back(p[0]);
    cp.push_back(p[1]);
    for (int i = 2; i < p.size(); i++) {
      while (cp.size() > 1) {
        if (ccw(cp[cp.size()-2], cp[cp.size()-1], p[i]) <= 0) {
          np.push_back(cp.back());
          cp.pop_back();
        } else break;
      }
      cp.push_back(p[i]);
    }
    if (cp.size() < 3) break;
    for (auto &e: cp) check[e.i] = ci, e.x += save.x, e.y += save.y;
    for (auto &e: np) e.x += save.x, e.y += save.y;
    swap(p, np);
    cps.push_back(cp);
    ci++;
  }
}
```

4.5 Rotating Callipers

```
// 가장 먼 두 점, O(n)
int i2 = 1, p1, p2;
ll maxdist = 0;
for (int i1 = 0; i1 < cp.size();) {
  auto dist = (cp[i2]-cp[i1]).norml1();
  if (maxdist <= dist) {
    maxdist = dist;
    p1 = i1;
    p2 = i2;
  }

  auto ccwv = ccw(cp[i1], cp[(i1+1)%cp.size()], cp[i2], cp[(i2+1)%cp.size()]);
  if (ccwv > 0) i2 = (i2+1)%cp.size();
  else i1++;
}
```

4.6 Angle Sort

```
sort(a, a+n, [&](const vector2 &a, const vector2 &b){
  if((tie(a.x, a.y) > tie(0, 0)) ^ (tie(b.x, b.y) > tie(0, 0))) return tie(a.x, a.y) >
  tie(b.x, b.y);
  if(ccw(a, b) != 0) return ccw(a, b) > 0; // ccw 잘 수정하셈
  return hypot(a) < hypot(b);
});
```

5 Graph

5.1 Floyd warshall

Time Complexity: $O(n^3)$

```
int citys[101][101];
int main() {
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= 100; i++)
    for (int j = 1; j <= 100; j++)
      citys[i][j] = 1e6 * 100;
  int a, b, c;
  for (int i = 0; i < m; i++) {
    cin >> a >> b >> c;
    citys[a][b] = min(citys[a][b], c);
  }
  for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        if (i != j)
          citys[i][j] = min(citys[i][j], citys[i][k]+citys[k][j]);
      }
    }
  }
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      cout << (citys[i][j] != 1e6 * 100 ? citys[i][j] : 0) << (j != n ? ' ' : '\n');
}
```

5.2 scc

```
// 1-index, initialize: scc_graph(int size), necessary make_graph
struct scc_graph {
  int n, id = 0;
  vector<int> ids, finished;
  vector<vector<int>> path, scc;
  stack<int> stk;

  scc_graph(int _n) {
    n = _n;
    ids = finished = vector<int>(n+1);
    path = vector<vector<int>>(n+1);
  }

  void make_graph() { for (int i = 1; i <= n; i++) if (!ids[i]) dfs(i); }
  void add_edge(int u, int v) { path[u].push_back(v); }

  int dfs(int s) {
    ids[s] = ++id;
    stk.push(s);

    int parent = ids[s];
    for (auto &i: path[s]) {
      if (!ids[i]) parent = min(parent, dfs(i));
```

```
      else if (!finished[i]) parent = min(parent, ids[i]);
    }

    if (parent == ids[s]) {
      vector<int> findset;
      while (true) {
        auto x = stk.top();
        stk.pop();

        findset.push_back(x);
        finished[x] = 1;
        if (s == x) break;
      }
      sort(findset.begin(), findset.end());
      scc.push_back(findset);
    }

    return parent;
  }
};
```

5.3 2-sat

```
struct csr {
  vector<int> s, el;
  csr(int n, vector<pii>& edges): s(n + 1), el(edges.size()) {
    for (auto e: edges) s[e.first+1]++;
    for (int i = 1; i <= n; i++) s[i] += s[i-1];
    auto cnt = s;
    for (auto e : edges) el[cnt[e.first]++] = e.second;
  }
};

struct scc_graph {
  scc_graph(int n) : _n(n) {}
  void add(int from, int to) { edges.push_back({from, to}); }
  pair<int, vector<int>> scc_ids() {
    auto g = csr(_n, edges);
    int cord = 0, gnum = 0;
    vector<int> vis, low(_n), ord(_n, -1), ids(_n);
    vis.reserve(_n);
    auto dfs = [&](auto self, int v) -> void {
      low[v] = ord[v] = cord++;
      vis.push_back(v);
      for (int i = g.s[v]; i < g.s[v + 1]; i++) {
        auto to = g.el[i];
        if (ord[to] == -1) {
          self(self, to);
          low[v] = min(low[v], low[to]);
        } else low[v] = min(low[v], ord[to]);
      }
      if (low[v] == ord[v]) {
        while (true) {
          int u = vis.back();
          vis.pop_back();
          ord[u] = _n, ids[u] = gnum;
          if (u == v) break;
        }
        gnum++;
      }
    };
    for (int i = 0; i < _n; i++) if (ord[i] == -1) dfs(dfs, i);
    for (auto& x : ids) x = gnum - 1 - x;
    return {gnum, ids};
  }

  vector<vector<int>> scc() {
    auto ids = scc_ids();
    int gnum = ids.first;
    vector<int> cnt(gnum);
    for (auto x : ids.second) cnt[x]++;
    vector<vector<int>> g(ids.first);
    for (int i = 0; i < gnum; i++) g[i].reserve(cnt[i]);
    for (int i = 0; i < _n; i++) g[ids.second[i]].push_back(i);
    return g;
  }

  int _n;
  vector<pii> edges;
};

struct two_sat {
  int n;
  vector<bool> ans;
  scc_graph scc;

  explicit two_sat(int _n): n(_n), ans(_n), scc(2*_n) {}
  void add(int i, bool f, int j, bool g) {
    scc.add(2 * i + (f ? 0 : 1), 2 * j + (g ? 1 : 0));
    scc.add(2 * j + (g ? 0 : 1), 2 * i + (f ? 1 : 0));
  }

  bool satisfiable() {
    auto id = scc.scc_ids().second;
    for (int i = 0; i < n; i++) {
      if (id[2 * i] == id[2 * i + 1]) return false;
      ans[i] = id[2 * i] < id[2 * i + 1];
    }
    return true;
  }
  vector<bool> answer() { return ans; }
};
```

5.4 hld

```
// require ACL seg
template <typename T = int>
struct Edge {
  int from, to;
  T cost;
```

```
  Edge(int _to, T _cost = 1): from(-1), to(_to), cost(_cost) {} // from is array index
  Edge(int _from, int _to, T _cost): from(_from), to(_to), cost(_cost) {} // only edge
  save, ex) use in prim
  Edge &operator= (const int &x) {
    to = x;
    return *this;
  }
  operator int() const { return to; }
};
// weighted graph
template <typename T>
using WeightedEdges = vector<Edge<T>>;
template <typename T>
using WeightedFIDXEdges = vector<vector<pair<int, T>>>;
using UnweightedEdges = vector<vector<int>>;
template <typename T>
pair<WeightedEdges<T>, WeightedFIDXEdges<T>> w_fidx_graph(int n, int m = -1, bool
is_directed = false, bool is_1idx = true) {
  WeightedEdges<T> wg;
  WeightedFIDXEdges<T> g(n), c(n);
  if (m == -1) m = n-1;
  int ui = 1;
  while (m--) {
    int u, v;
    T w;
    cin >> u >> v;
    // cin >> v;
    // u = ++ui;
    w=0;
    if (is_1idx) u--, v--;
    wg.emplace_back(u, v, w);
    g[u].push_back({v, w});
    if (!is_directed) g[v].push_back({u, w});
  }
  queue<pair<int,int>> q;
  q.push({0, -1});
  while (q.size()) {
    auto [v, pv] = q.front();
    q.pop();
    for (auto &[e, w]: g[v]) {
      if (e != pv) {
        c[v].emplace_back(e, w);
        q.push({e, v});
      }
    }
  }
  return {wg, c};
}
// basic HLD
template <typename T, class S, auto Sop, auto Se>
struct HeavyLightDecomposition {
  // private:
  public:
    int root, eid, fid;
    vector<int> sz, d, p, top, in, out, vis, id;
    vector<T> x;
    WeightedFIDXEdges<T> c;
    atcoder::segtree<S, Sop, Se> seg;
    void dfs1(int v, int pv = -1, int cid = -1) {
      vis[v] = 1;
      sz[v] = 1;
      id[v] = cid;
      for (auto &e: c[v]) {
        if (e.first == pv) continue;
        d[e.first] = d[v]+1;
        p[e.first] = v;
        dfs1(e.first, v, cid);
        sz[v] += sz[e.first];
        if (sz[e.first] > sz[c[v][0].first]) swap(e, c[v][0]);
      }
    }
    void dfs2(int v, int pv = -1) {
      vis[v] = 2;
      in[v] = eid++;
      for (auto &[e, w]: c[v]) {
        if (e == pv) continue;
        top[e] = e == c[v][0].first ? top[v] : e;
        dfs2(e, v);
        x[e] = w;
      }
      out[v] = eid;
    }
  // public:
    HeavyLightDecomposition(WeightedFIDXEdges<T> g, int _root = 0)
      : c(g),
        root(_root),
        eid(0),
        fid(0),
        sz(g.size(), 0),
        d(g.size(), 0),
        x(g.size(), 0),
        in(g.size(), -1),
        out(g.size(), -1),
        vis(g.size(), 0),
        id(g.size(), 0),
        seg(atcoder::segtree<S, Sop, Se>(g.size()+1)),
        top(g.size(), root),
        p(g.size(), root) {
      for (int i = 0; i < g.size(); i++) if (vis[i] != 1) dfs1(i, -1, fid++);
      for (int i = 0; i < g.size(); i++) if (vis[i] != 2) dfs2(i);
      // init_seg();
    }

    void init_seg(vector<T> v) {
      assert(x.size() == v.size());
      for (int i = 0; i < x.size(); i++) seg.set(in[i], x[i] = v[i]);
    }

    int lca(int u, int v) {
      while (top[u] != top[v]) {
```

```
      if (d[top[u]] < d[top[v]]) swap(u, v);
      int st = top[u];
      u = p[st];
    }
    if (d[u] > d[v]) swap(u, v);
    return u;
  }

  S query(int u, int v, int inc_start = 0) {
    S r = Se();
    while (top[u] != top[v]) {
      if (d[top[u]] < d[top[v]]) swap(u, v);
      int st = top[u];
      r = Sop(r, seg.prod(in[st], in[u]+1));
      u = p[st];
    }
    if (d[u] > d[v]) swap(u, v);
    r = Sop(r, seg.prod(in[u]+inc_start, in[v]+1));
    return r;
  }
};
// lazy hld
template <typename T, class S, auto Sop, auto Se, class F, auto mapping, auto composition,
auto id>
struct LazyHeavyLightDecomposition {
  // private:
  public:
    int root, eid;
    vector<int> sz, d, p, top, in, out, vis;
    vector<T> x;
    WeightedFIDXEdges<T> c;
    atcoder::lazy_segtree<S, Sop, Se, F, mapping, composition, id> seg;
    void dfs1(int v, int pv = -1) {
      vis[v] = 1;
      sz[v] = 1;
      for (auto &e: c[v]) {
        if (e.first == pv) continue;
        d[e.first] = d[v]+1;
        p[e.first] = v;
        dfs1(e.first, v);
        sz[v] += sz[e.first];
        if (sz[e.first] > sz[c[v][0].first]) swap(e, c[v][0]);
      }
    }
    void dfs2(int v, int pv = -1) {
      vis[v] = 2;
      in[v] = eid++;
      for (auto &[e, w]: c[v]) {
        if (e == pv) continue;
        top[e] = e == c[v][0].first ? top[v] : e;
        dfs2(e, v);
        x[e] = w;
      }
      out[v] = eid;
    }
// public:
  LazyHeavyLightDecomposition(WeightedFIDXEdges<T> g, int _root = 0)
    : c(g),
      root(_root),
      eid(0),
      sz(g.size(), 0),
      d(g.size(), 0),
      x(g.size(), 0),
      in(g.size(), -1),
      out(g.size(), -1),
      vis(g.size(), 0),
      seg(atcoder::lazy_segtree<S, Sop, Se, F, mapping, composition, id>(g.size()+1)),
      top(g.size(), root),
      p(g.size(), root) {
    for (int i = 0; i < g.size(); i++) if (vis[i] != 1) dfs1(i);
    for (int i = 0; i < g.size(); i++) if (vis[i] != 2) dfs2(i);
    // init_seg();
  }

  void init_seg(vector<S> v) {
    assert(x.size() == v.size());
    for (int i = 0; i < x.size(); i++) seg.set(in[i], v[i]);
  }

  int lca(int u, int v) {
    while (top[u] != top[v]) {
      if (d[top[u]] < d[top[v]]) swap(u, v);
      int st = top[u];
      u = p[st];
    }
    if (d[u] > d[v]) swap(u, v);
    return u;
  }

  S query(int u, int v, int inc_start = 0) {
    S l = Se();
    S r = Se();
    while (top[u] != top[v]) {
      // if (d[top[u]] < d[top[v]]) swap(u, v);
      if (d[top[u]] > d[top[v]]) {
        int st = top[u];
        l = Sop(seg.prod(in[st], in[u]+1), l);
        u = p[st];
      } else {
        int st = top[v];
        r = Sop(seg.prod(in[st], in[v]+1), r);
        v = p[st];
      }
    }
    if (d[u] < d[v]) r = Sop(seg.prod(in[u]+inc_start, in[v]+1), r);
    else l = Sop(seg.prod(in[v]+inc_start, in[u]+1), l);
    swap(l.l, l.r);
    return Sop(l, r);
  }
```

```
    void apply(int u, int v, F x, int inc_start = 0) {
      while (top[u] != top[v]) {
        if (d[top[u]] < d[top[v]]) swap(u, v);
        int st = top[u];
        seg.apply(in[st], in[u]+1, x);
        u = p[st];
      }
      if (d[u] > d[v]) swap(u, v);
      seg.apply(in[u]+inc_start, in[v]+1, x);
    }
};
```

5.5 Bellman Ford

```
// require HLD-WeightedEdges,
template <typename T = int>
vector<T> bellman_ford(int n, kENN3N::WeightedEdges<T> edges, int sv = 0, int ev = -1) {
  T MAX = numeric_limits<T>::max() / 2;
  vector<T> d(n, MAX);
  d[sv] = 0;
  for (int i = 0; i < n; i++) {
    bool upd = false;
    for (auto &e: edges) {
      if (d[e.from] == MAX) continue;
      if (d[e.to] > d[e.from] + e.cost) upd = true, d[e.to] = d[e.from] + e.cost;
    }
    if (!upd) return d;
  }
  if (ev == -1) return vector<T>();
  vector<bool> nega(n, false);
  for (int i = 0; i < n; i++) {
    for (auto &e: edges) {
      if (d[e.from] == MAX) continue;
      if (d[e.to] > d[e.from] + e.cost) nega[e.to] = true, d[e.to] = d[e.from] + e.cost;
      if (nega[e.from]) nega[e.to] = true;
    }
  }
  if (nega[ev]) return vector<T>();
  else return d;
}
/*
벨만 포드 부등식
방법은, xj - xi <= T일때, i -> j로 가는 가중치 T의 간선을 만들어주는 것이다.
이러한 일련의 작업이 끝났다면, 임의의 정점 i에서 벨만 포드를 돌렸을 때, xj = xi + ShortestPath(i,
j) 라는 값을 대입했을 때 저것을 만족함을 보장할 수 있다.
*/
WeightedEdges<ll> g;
for (int i = 0; i < q; i++) {
  int a, b, x;
  string op;
  cin >> a >> b >> op >> x;
  // a--, b--;
  if (op == "<=") g.push_back(Edge<ll>(b+n, a, x));
  else g.push_back(Edge<ll>(a, b+n, -x));
}
for (int i = 1; i <= n+m; i++) g.push_back(Edge<ll>(0, i, 0));
auto y = bellman_ford(n+m+1, g);
cout << (y.size() ? "Possible" : "Impossible") << "\n";
```

5.6 Maxflow

Time Complexity: $O(n+m)\sqrt{n}$, $O(n^2m)$

```
template <class Cap> struct mf_graph {
  mf_graph(int n = 0) : _n(n), g(n) {}
  void add_edge(int from, int to, Cap cap) {
    int from_id = g[from].size();
    int to_id = g[to].size();
    g[from].push_back({to, to_id, cap});
    g[to].push_back({from, from_id, 0});
  }
  Cap flow(int s, int t, Cap limit = numeric_limits<Cap>::max()) {
    Cap flow = 0;
    vector<int> level(_n), iter(_n);
    auto bfs = [&]() {
      fill(level.begin(), level.end(), -1);
      level[s] = 0;
      queue<int> q;
      q.push(s);
      while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (auto& e : g[v]) {
          if (e.cap > 0 && level[e.to] < 0) {
            level[e.to] = level[v] + 1;
            q.push(e.to);
          }
        }
      }
      return level[t] != -1;
    };
    auto dfs = [&](auto self, int v, Cap up) -> Cap {
      if (v == t) return up;
      Cap res = 0;
      for (int& i = iter[v]; i < g[v].size(); i++) {
        _edge &e = g[v][i], &re = g[e.to][e.rev];
        if (e.cap > 0 && level[v] < level[e.to]) {
          Cap d = self(self, e.to, min(up - res, e.cap));
          if (d > 0) {
            e.cap -= d;
            re.cap += d;
            res += d;
            if (res == up) break;
          }
        }
      }
      return res;
    };
    while (flow < limit && bfs()) {
      fill(iter.begin(), iter.end(), 0);
```

```
      Cap f = dfs(dfs, s, limit - flow);
      if (!f) break;
      flow += f;
    }
    return flow;
  }

private:
  struct _edge { int to, rev; Cap cap; };
  int _n;
  vector<vector<_edge>> g;
};
mf_graph<int> mf;
```

5.7 Min cost Max Flow

Time Complexity: $O(F(n+m)log(n+m))$

```
template <class Cap, class Cost> struct mcf_graph {
  public:
    mcf_graph(int n = 0) : _n(n), g(n) {}
    void add_edge(int from, int to, Cap cap, Cost cost) {
      g[from].push_back({to, (int)g[to].size(), cap, cost});
      g[to].push_back({from, (int)g[from].size() - 1, 0, -cost});
    }
    pair<Cap, Cost> flow(int s, int t, Cap flow_limit = numeric_limits<Cap>::max()) {
      Cap flow = 0;
      Cost cost = 0;
      vector<Cost> h(_n, 0);
      vector<int> prev_v(_n), prev_e(_n);

      while (flow < flow_limit) {
        vector<Cost> dist(_n, numeric_limits<Cost>::max());
        dist[s] = 0;
        priority_queue<pair<Cost, int>, vector<pair<Cost, int>>, greater<pair<Cost, int>>>
        que;
        que.push({0, s});

        while (!que.empty()) {
          auto [d, v] = que.top();
          que.pop();
          if (dist[v] < d) continue;
          for (int i = 0; i < g[v].size(); i++) {
            _edge& e = g[v][i];
            if (e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
              dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
              prev_v[e.to] = v;
              prev_e[e.to] = i;
              que.push({dist[e.to], e.to});
            }
          }
        }

        if (dist[t] == numeric_limits<Cost>::max()) break;
        for (int i = 0; i < _n; i++) h[i] += dist[i];

        Cap d = flow_limit - flow;
        for (int v = t; v != s; v = prev_v[v]) d = min(d, g[prev_v[v]][prev_e[v]].cap);
        flow += d;
        cost += d * h[t];
        for (int v = t; v != s; v = prev_v[v]) {
          _edge& e = g[prev_v[v]][prev_e[v]];
          e.cap -= d;
          g[v][e.rev].cap += d;
        }
      }
      return {flow, cost};
    }
  private:
    struct _edge { int to, rev; Cap cap; Cost cost; };
    int _n;
    vector<vector<_edge>> g;
};
```

6 Math

6.1 Combination

```
// ncr
struct Combination {
  const ll MAX, MOD;
  vector<ll> inv, fac, fi;
  Combination(ll max, ll mod): MAX(max), MOD(mod) {
    inv = vector<ll>(max+1);
    fac = vector<ll>(max+1);
    fi = vector<ll>(max+1);
    fac[1] = inv[1] = fi[1] = 1;
    for (ll i = 2; i <= max; i++) {
      fac[i] = (fac[i-1]*i) % mod;
      inv[i] = (mod - mod/i) * inv[mod%i] % mod;
      fi[i] = fi[i-1]*inv[i]  % mod;
    }
  }
  ll ncr(ll n, ll r) {
    if (r == 0 || n == r) return 1;
    return (fac[n]*fi[r]%MOD)*fi[n-r]%MOD;
  }
};
```

6.2 FFT - int

```
using pll = pair<ll, ll>;
using ull = unsigned long long;
// X.v 와 같이 접근해야함. int 범위 내 NTT 전용
namespace NTT {
  constexpr int MOD = 998244353;
  constexpr int G = 3; // Primitive root for MOD
  struct mint {
    int v;
    mint(ll x = 0) : v(x % MOD) { if (v < 0) v += MOD; }
    mint& operator+=(const mint& o) { v += o.v; if (v >= MOD) v -= MOD; return *this; }
    mint& operator-=(const mint& o) { v -= o.v; if (v < 0) v += MOD; return *this; }
    mint& operator*=(const mint& o) { v = (int)((ll)v * o.v % MOD); return *this; }
    mint operator+(const mint& o) const { return mint(*this) += o; }
    mint operator-(const mint& o) const { return mint(*this) -= o; }
    mint operator*(const mint& o) const { return mint(*this) *= o; }
    mint pow(ll n) const {
      mint res = 1, x = *this;
      while (n > 0) {
        if (n & 1) res *= x;
        x *= x;
        n >>= 1;
      }
      return res;
    }
    mint inv() const { return pow(MOD - 2); }
  };
  int ceil_pow2(int n) {
    int x = 0;
    while ((1U << x) < (unsigned int)(n)) x++;
    return x;
  }
  int bsf(unsigned int n) { return __builtin_ctz(n); }
  void butterfly(vector<mint>& a) {
    int n = a.size();
    int h = ceil_pow2(n);
    static bool first = true;
    static vector<mint> sum_e;
    if (first) {
      first = false;
      sum_e.resize(30);
      mint es[30], ies[30];
      int cnt2 = bsf(MOD - 1);
      mint e = mint(G).pow((MOD - 1) >> cnt2), ie = e.inv();
      for (int i = cnt2; i >= 2; i--) {
        es[i - 2] = e;
        ies[i - 2] = ie;
        e *= e;
        ie *= ie;
      }
      mint now = 1;
      for (int i = 0; i < cnt2 - 2; i++) {
        sum_e[i] = es[i] * now;
        now *= ies[i];
      }
    }
    for (int ph = 1; ph <= h; ph++) {
      int w = 1 << (ph - 1), p = 1 << (h - ph);
      mint now = 1;
      for (int s = 0; s < w; s++) {
        int offset = s << (h - ph + 1);
        for (int i = 0; i < p; i++) {
          auto l = a[i + offset];
          auto r = a[i + offset + p] * now;
          a[i + offset] = l + r;
          a[i + offset + p] = l - r;
        }
        now *= sum_e[bsf(~(unsigned int)(s))];
      }
    }
  }
  void butterfly_inv(vector<mint>& a) {
    int n = a.size();
    int h = ceil_pow2(n);
    static bool first = true;
    static vector<mint> sum_ie;
    if (first) {
      first = false;
      sum_ie.resize(30);
      mint es[30], ies[30];
      int cnt2 = bsf(MOD - 1);
      mint e = mint(G).pow((MOD - 1) >> cnt2), ie = e.inv();
      for (int i = cnt2; i >= 2; i--) {
        es[i - 2] = e; ies[i - 2] = ie;
```

```
        e *= e; ie *= ie;
      }
      mint now = 1;
      for (int i = 0; i < cnt2 - 2; i++) { sum_ie[i] = ies[i] * now; now *= es[i]; }
    }
    for (int ph = h; ph >= 1; ph--) {
      int w = 1 << (ph - 1), p = 1 << (h - ph);
      mint inow = 1;
      for (int s = 0; s < w; s++) {
        int offset = s << (h - ph + 1);
        for (int i = 0; i < p; i++) {
          auto l = a[i + offset];
          auto r = a[i + offset + p];
          a[i + offset] = l + r;
          a[i + offset + p] = (l - r) * inow;
        }
        inow *= sum_ie[bsf(~(unsigned int)(s))];
      }
    }
  }
  vector<mint> convolution(vector<mint> a, vector<mint> b) {
    int n = a.size(), m = b.size();
    if (!n || !m) return {};
    if (min(n, m) <= 60) {
      if (n < m) { swap(n, m); swap(a, b); }
      vector<mint> ans(n + m - 1);
      for (int i = 0; i < n; i++) for (int j = 0; j < m; j++) ans[i + j] += a[i] * b[j];
      return ans;
    }
    int z = 1 << ceil_pow2(n + m - 1);
    a.resize(z); b.resize(z);
    butterfly(a); butterfly(b);
    for (int i = 0; i < z; i++) a[i] *= b[i];
    butterfly_inv(a);
    a.resize(n + m - 1);
    mint iz = mint(z).inv();
    for (int i = 0; i < n + m - 1; i++) a[i] *= iz;
    return a;
  }
} // namespace NTT
vector<NTT::mint> a;
```

6.3 FFT - ll (Dnc + CRT)

```
using ull = unsigned long long;
// fft ll = DnC + CRT
// ================= AtCoder Convolution Library =================
namespace atcoder {
namespace internal {
  int ceil_pow2(int n) { int x = 0; while ((1U << x) < (ui)(n)) x++; return x; }
  int bsf(ui n) { return __builtin_ctz(n); }
  constexpr ll safe_mod(ll x, ll m) { x %= m; if (x < 0) x += m; return x; }
  constexpr pair<ll, ll> inv_gcd(ll a, ll b) {
    a = safe_mod(a, b);
    if (a == 0) return {b, 0};
    ll s = b, t = a;
    ll m0 = 0, m1 = 1;
    while (t) {
      ll u = s / t;
      s -= t * u;
      m0 -= m1 * u;
      auto tmp = s; s = t; t = tmp;
      tmp = m0; m0 = m1; m1 = tmp;
    }
    if (m0 < 0) m0 += b / s;
    return {s, m0};
  }
  constexpr ll pow_mod(ll x, ll n, int m) {
    if (m == 1) return 0;
    ui _m = (ui)(m);
    ull r = 1, y = safe_mod(x, m);
    while (n) { if (n & 1) r = (r * y) % _m; y = (y * y) % _m; n >>= 1; }
    return r;
  }
  constexpr bool is_prime(int n) {
    if (n <= 1) return false;
    if (n == 2 || n == 7 || n == 61) return true;
    if (n % 2 == 0) return false;
    ll d = n - 1;
    while (d % 2 == 0) d /= 2;
    constexpr ll bases[3] = {2, 7, 61};
    for (ll a : bases) {
      ll t = d, y = pow_mod(a, t, n);
      while (t != n - 1 && y != 1 && y != n - 1) { y = y * y % n; t <<= 1; }
      if (y != n - 1 && t % 2 == 0) return false;
    }
    return true;
  }
  constexpr int primitive_root(int m) {
    if (m == 998244353) return 3;
    if (m == 2) return 1;
    int divs[20] = {2}, cnt = 1;
    int x = (m - 1) / 2;
    while (x % 2 == 0) x /= 2;
    for (int i = 3; (ll)(i) * i <= x; i += 2) {
      if (x % i == 0) { divs[cnt++] = i; while (x % i == 0) x /= i; }
    }
    if (x > 1) divs[cnt++] = x;
    for (int g = 2; ; g++) {
      bool ok = true;
      for (int i = 0; i < cnt; i++) {
        if (pow_mod(g, (m - 1) / divs[i], m) == 1) { ok = false; break; }
      }
      if (ok) return g;
    }
  }
  template <class T> using is_integral = typename std::is_integral<T>;
} // namespace internal
template <int m>
struct static_modint {
```

```
  using mint = static_modint;
  static constexpr int mod() { return m; }
  int v;
  static_modint(ll x = 0) : v(x % m) { if (v < 0) v += m; }
  mint& operator+=(const mint& o) { v += o.v; if (v >= m) v -= m; return *this; }
  mint& operator-=(const mint& o) { v -= o.v; if (v < 0) v += m; return *this; }
  mint& operator*=(const mint& o) { v = (int)((ll)v * o.v % m); return *this; }
  mint& operator/=(const mint& o) { return *this = *this * o.inv(); }
  mint operator+(const mint& o) const { return mint(*this) += o; }
  mint operator-(const mint& o) const { return mint(*this) -= o; }
  mint operator*(const mint& o) const { return mint(*this) *= o; }
  mint operator/(const mint& o) const { return mint(*this) /= o; }
  mint pow(ll n) const {
    mint res = 1, x = *this;
    while (n > 0) { if (n & 1) res *= x; x *= x; n >>= 1; }
    return res;
  }
  mint inv() const {
    if (prime) { assert(v); return pow(m - 2); }
    else { auto eg = internal::inv_gcd(v, m); assert(eg.first == 1); return eg.second; }
  }
  static constexpr bool prime = internal::is_prime(m);
};
namespace internal {
  template <class mint> void butterfly(vector<mint>& a) {
    constexpr int g = primitive_root(mint::mod());
    int n = a.size(), h = ceil_pow2(n);
    static vector<mint> sum_e;
    if (sum_e.empty()) {
      sum_e.resize(30);
      mint es[30], ies[30];
      int cnt2 = bsf(mint::mod() - 1);
      mint e = mint(g).pow((mint::mod() - 1) >> cnt2), ie = e.inv();
      for (int i = cnt2; i >= 2; i--) {
        es[i - 2] = e; ies[i - 2] = ie; e *= e; ie *= ie;
      }
      mint now = 1;
      for (int i = 0; i < cnt2 - 2; i++) {
        sum_e[i] = es[i] * now; now *= ies[i];
      }
    }
    for (int ph = 1; ph <= h; ph++) {
      int w = 1 << (ph - 1), p = 1 << (h - ph);
      mint now = 1;
      for (int s = 0; s < w; s++) {
        int offset = s << (h - ph + 1);
        for (int i = 0; i < p; i++) {
          auto l = a[i + offset], r = a[i + offset + p] * now;
          a[i + offset] = l + r; a[i + offset + p] = l - r;
        }
        now *= sum_e[bsf(~(ui)(s))];
      }
    }
  }
  template <class mint> void butterfly_inv(vector<mint>& a) {
    constexpr int g = primitive_root(mint::mod());
    int n = a.size(), h = ceil_pow2(n);
    static vector<mint> sum_ie;
    if (sum_ie.empty()) {
      sum_ie.resize(30);
      mint es[30], ies[30];
      int cnt2 = bsf(mint::mod() - 1);
      mint e = mint(g).pow((mint::mod() - 1) >> cnt2), ie = e.inv();
      for (int i = cnt2; i >= 2; i--) {
        es[i - 2] = e; ies[i - 2] = ie; e *= e; ie *= ie;
      }
      mint now = 1;
      for (int i = 0; i < cnt2 - 2; i++) {
        sum_ie[i] = ies[i] * now; now *= es[i];
      }
    }
    for (int ph = h; ph >= 1; ph--) {
      int w = 1 << (ph - 1), p = 1 << (h - ph);
      mint inow = 1;
      for (int s = 0; s < w; s++) {
        int offset = s << (h - ph + 1);
        for (int i = 0; i < p; i++) {
          auto l = a[i + offset], r = a[i + offset + p];
          a[i + offset] = l + r; a[i + offset + p] = (l - r) * inow;
        }
        inow *= sum_ie[bsf(~(ui)(s))];
      }
    }
  }
} // namespace internal
template <class mint>
vector<mint> convolution(vector<mint> a, vector<mint> b) {
  int n = a.size(), m = b.size();
  if (!n || !m) return {};
  int z = 1 << internal::ceil_pow2(n + m - 1);
  a.resize(z); b.resize(z);
  internal::butterfly(a); internal::butterfly(b);
  for (int i = 0; i < z; i++) a[i] *= b[i];
  internal::butterfly_inv(a);
  a.resize(n + m - 1);
  mint iz = mint(z).inv();
  for (int i = 0; i < n + m - 1; i++) a[i] *= iz;
  return a;
}
template <ui mod = 998244353, class T, enable_if_t<internal::is_integral<T>::value>* =
nullptr>
vector<T> convolution_int(const vector<T>& a, const vector<T>& b) {
  int n = a.size(), m = b.size();
  if (!n || !m) return {};
  using mint = static_modint<mod>;
  vector<mint> a2(n), b2(m);
  for (int i = 0; i < n; i++) a2[i] = mint(a[i]);
  for (int i = 0; i < m; i++) b2[i] = mint(b[i]);
  auto c2 = convolution(move(a2), move(b2));
  vector<T> c(n + m - 1);
```

```
  for (int i = 0; i < n + m - 1; i++) c[i] = c2[i].v;
  return c;
}
vector<ll> convolution_ll(const vector<ll>& a, const vector<ll>& b) {
  int n = a.size(), m = b.size();
  if (!n || !m) return {};
  static constexpr ui MOD1 = 754974721; // 2^24 * 45 + 1
  static constexpr ui MOD2 = 167772161; // 2^25 * 5 + 1
  static constexpr ui MOD3 = 469762049; // 2^26 * 7 + 1
  auto c1 = convolution_int<MOD1>(a, b);
  auto c2 = convolution_int<MOD2>(a, b);
  auto c3 = convolution_int<MOD3>(a, b);
  ll m1_m2 = (ll)MOD1 * MOD2;
  ll m1_inv_m2 = internal::inv_gcd(MOD1, MOD2).second;
  ll m1_m2_inv_m3 = internal::inv_gcd(m1_m2, MOD3).second;
  vector<ll> c(n + m - 1);
  for (int i = 0; i < n + m - 1; i++) {
    ll v1 = c1[i], v2 = c2[i], v3 = c3[i];
    ll x = v1;
    ll y = (v2 - x) * m1_inv_m2 % MOD2;
    if (y < 0) y += MOD2;
    x += y * MOD1;
    ll z = (v3 - x % MOD3) * m1_m2_inv_m3 % MOD3;
    if (z < 0) z += MOD3;
    c[i] = x + z * m1_m2;
  }
  return c;
}} // namespace atcoder
```

6.4 Pollard Rho (with fast prime check)

```
// time: O(n^(1/4))
using i64 = __int128_t;
i64 my_abs(i64 x) { return (x < 0 ? -1*x : x); }
i64 primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 };
// i64 primes[] = { 2, 7, 61 }; // int only
i64 f(i64 x, i64 p, i64 mod) {
  i64 r = 1;
  x %= mod;
  while (p) {
    if (p%2) r *= x, r %= mod;
    p /= 2, x *= x, x %= mod;
  }
  return r;
}
// a is test number, a^d = 1 (MOD N), a^(d*2^r) = -1 (MOD N), d is odd
bool miller_rabin(i64 n, i64 a) {
  if (n == a) return true;
  if (a % n == 0) return false;
  i64 k = n-1;
  while (true) {
    i64 t = f(a, k, n);
    if (t == n-1) return true;
    if (k%2) return (t == 1 || t == n-1);
    k /= 2;
  }
}
bool isp(i64 x) {
  bool p = true;
  for (auto &e: primes) {
    p &= miller_rabin(x, e);
    if (!p) return false;
  }
  return true;
}
void g(i64 x, vector<i64> &v) {
  if (x == 1) return;
  if (x%2 == 0) { v.push_back(2); g(x/2, v); return; }
  if (isp(x)) { v.push_back(x); return; }
  i64 a, b, c, w = x;
  auto h = [&](i64 y) { return (c + y*y%x) % x; };
  do {
    if (w == x) a = b = rand()%(x-2) + 2, c = rand() % 20 + 1;
    a = h(a), b = h(h(b)), w = __gcd(my_abs(a-b), x);
  } while (w == 1);
  g(w, v);
  g(x/w, v);
}
// source: jinhan, factorize code
vector<pair<i64, int>> pollard_rho(i64 x) {
  vector<i64> v;
  vector<pair<i64, int>> r;
  g(x, v);
  sort(v.begin(), v.end());
  for (auto e: v) {
    if (r.size() && r.back().first == e) r.back().second++;
    else r.push_back({ e, 1 });
  }
  return r;
}
int main() {
  ll x;
  while (cin >> x) {
    ll r = 1;
    for (auto [e, c]: pollard_rho(x)) r *= (c+1);
    cout << r << "\n";
  }
}
```

6.5 Permutation and Combination

```
'''
중복 순열
서로 다른 n개의 원소 중에서 중복을 허용하면서 r개를 순서있게 뽑는 방법의 수
즉, 한 원소를 여러번 뽑을 수 있고, 뽑은 순서를 고려함

print(n**r)

========================================================================
```

```
중복 조합
서로 다른 n개의 원소 중에서 중복을 허용하면서 r개를 순서 없이 뽑는 방법의 수
즉, 한 원소를 여러번 뽑을 수 있지만, 순서는 고려하지 않음

math.comb(n+r-1, r)

======================================================================

중복 원소가 있는 순열
동일한 원소가 여러 번 있는 집합(예: AAB)에서 원소들을 모두 사용하여 순열을 만드는 방법의 수
전체 원소의 개수가 n이고, 중복된 원소의 개수가 n1,n2,...,nk일 때

math.factorial(n) / (math.factorial(n1) * math.factorial(n2) * ...)

======================================================================

완전 순열
순열의 일종으로, 일렬로 배열한 대상들의 위치를 재조정했을 때,
모든 대상이 자기 위치에 있지 않도록 하는 배열 방법

D[n] = (n-1)(D[n-1]+D[n-2])

D[0] = 1
D[1] = 0
D[2] = 1
D[3] = 2
D[4] = 9
D[5] = 44
'''
```

6.6 Pairwise Product Sum

```python
# 모든 i<j에 대해 arr[i]*arr[j]의 합
# 1*2 + 1*3 + 1*4 + 2*3 + 2*4 + 3*4
arr = [1,2,3,4]

S = sum(arr)
sum_sq = sum(x*x for x in arr)

result = (S*S - sum_sq) // 2
print(result)  # 35
```

6.7 Modular Divide

```python
# (a/b)%M => (a%M * pow(b,M-2,M)) % M

M = 1000000007
a=7
b=3
print((a%M * pow(b,M-2,M)) % M)
```

7 String

7.1 KMP

```cpp
vector<int> get_pi(string p) {
  vector<int> pi(p.length(), 0);

  for (int i = 1, j = 0; i < p.length(); i++) {
    while (j && p[i] != p[j]) j = pi[j-1];
    if (p[i] == p[j]) pi[i] = ++j;
  }
  return pi;
}
int kmp(string s, string p) {
  auto pi = get_pi(p);
  int lj = 0;

  for (int i = 0, j = 0; i < s.length(); i++) {
    lj = j;
    while (j && s[i] != p[j]) j = pi[j-1];
    if (s[i] == p[j]) {
      if (j == p.length() - 1) {
        // result.push_back(i-p.length()+1); if want result string
        j = pi[j];
      } else j++;
    }
  }
  return lj;
}
```

7.2 Z

```cpp
// 접미사와 문자열전체접두사와 가장 긴 공통접두사
template <class T> vector<int> z_algorithm(const vector<T>& s) {
  int n = int(s.size());
  if (n == 0) return {};
  vector<int> z(n);
  z[0] = 0;
  for (int i = 1, j = 0; i < n; i++) {
    int& k = z[i];
    k = (j + z[j] <= i) ? 0 : std::min(j + z[j] - i, z[i - j]);
    while (i + k < n && s[k] == s[i + k]) k++;
    if (j + z[j] < i + z[i]) j = i;
  }
  z[0] = n;
  return z;
}
vector<int> z_algorithm(const string& s) {
  int n = int(s.size());
  vector<int> s2(n);
  for (int i = 0; i < n; i++) s2[i] = s[i];
  return z_algorithm(s2);
}
```

7.3 Trie and Ahocorasick

```
// include Trie
struct Trie {
  map<char, Trie*> ch;
  Trie* fail;
  bool end;
  Trie() { end = false; }
  ~Trie() {
    for (auto trie: ch) delete trie.second;
    ch.clear();
  }
  void insert(string s, int i = 0) {
    if (i == s.length()) {
      this->end = true;
      return;
    }
    if (!ch[s[i]]) ch[s[i]] = new Trie();
    ch[s[i]]->insert(s, i+1);
  }
};
Trie trie;
void set_fail() {
  queue<Trie*> q;
  trie.fail = &trie;
  q.push(&trie);
  while (q.size()) {
    Trie* cur = q.front();
    q.pop();
    for (auto [c, nxt]: cur->ch) {
      if (cur == &trie) nxt->fail = &trie;
      else {
        auto dest = cur->fail;
        while (dest != &trie && dest->ch.find(c) == dest->ch.cend()) dest = dest->fail;
        if (dest->ch.find(c) != dest->ch.cend()) dest = dest->ch[c];
        nxt->fail = dest;
      }
      if (nxt->fail->end) nxt->end = true;
      q.push(nxt);
    }
  }
}
int main() {
  // put all string to trie
  set_fail();
  cin >> n;
  while (n--) {
    cin >> s;
    bool result = false;
    Trie* cur = &trie;
    for (auto c: s) {
      while (cur != &trie && cur->ch.find(c) == cur->ch.cend()) cur = cur->fail;
      if (cur->ch.find(c) != cur->ch.cend()) cur = cur->ch[c];
      if (cur->end) result = true;
    }
    cout << (result ? "YES" : "NO") << (n ? "\n" : "");
  }
}
```

7.4 Suffix Automaton

```
struct Suffix_Automaton {
  struct Node {
    int len, link, last, cnt = 1;
    int minpos = 0, maxpos = 0;
    // bool isleaf = true;
    map<int, int> nxt;
    // int nxt[26];
    Node(int len = 0, int link = 0, int last = 0): len(len), link(link), last(last) {}
  };
  vector<Node> v;
  vector<pii> sv; // len / original i
  int total;
  // ll substr_cnt = 0;
  Suffix_Automaton() { total = 0; v.push_back(Node(0, -1)); }
  void add(int c, int last = 0) {
    // int c = _c - 'A';
    v.push_back(Node(v[total].len+1, 0, last));
    int p = total;
    v[p].minpos = v[p].maxpos = v[p].len-1;
    total = v.size()-1;
    // cout << "[debug]" << (char)c << ": " << v[p].len << "," << v[p].link << "," <<
    v[p].minpos << "," << v[p].maxpos << "\n";
    while (p != -1 && v[p].nxt.find(c) == v[p].nxt.cend()) {
      v[p].nxt.insert({ c, total });
      // v[p].nxt[c] = total;
      // if (v[p].link != -1) substr_cnt += v[p].len-v[v[p].link].len;
      // else substr_cnt++;
      p = v[p].link;
    }
    if (p != -1) {
      int q = v[p].nxt[c];
      int upd = q;
      if (v[p].len+1 < v[q].len) {
        upd = v.size();
        Node clone = v[q];
        clone.len = v[p].len + 1;
        clone.cnt = 0;
        v.push_back(clone);
        v[q].link = upd;
        for (int j = p; j != -1 && v[j].nxt[c] == q; j = v[j].link) v[j].nxt[c] = upd;
      }
      v[total].link = upd;
    }
  }
  // len, i
  pii lcs(string s) {
    int cur = 0;
    Node x = v[0];
    int l = 0, rl = 0, ri = 0;
```

```
  for (int i = 0; i < s.length(); i++) {
    x = v[cur];
    // cout << i << ": " << x.len << ", " << x.link << "\n";
    // cout << i << "(" << s[i] << "): " << x.len << " " << x.link << " => ";
    int idx = s[i] - 'a';
    // for (auto e: x.nxt) cout << e << " "; cout << " => ";
    while (x.len && x.nxt[idx] == -1) {
      // l -= (x.len-v[x.link].len);
      l = v[x.link].len;
      cur = x.link;
      x = v[x.link];
    }
    if (x.nxt[idx] != -1) cur = v[cur].nxt[idx], l++;
    if (l > rl) rl = l, ri = i;
    // cout << x.len << " " << x.link << "\n";
  }
  return { rl, ri };
}
void clear() {
  total = 0;
  v.clear();
  v.push_back(Node(0, -1, 0));
}
// vector<int> search(string t) {
//   vector<int> pos;
//   int cur = 0, l = 0;
//   for (int i = 0; i < t.length(); i++) {
//     while (v[cur].len && v[cur].nxt.find(t[i]) == v[cur].nxt.cend()) {
//       cur = v[cur].link;
//       l = v[cur].len;
//     }
//     if (v[cur].nxt.find(t[i]) != v[cur].nxt.cend()) cur = v[cur].nxt[t[i]], l++;
//     if (t.length() == l) pos.push_back(i-l+2);
//   }
//   return pos;
// }
// bool search(string t) {
//   int cur = 0, l = 0;
//   for (int i = 0; i < t.length(); i++) {
//     while (v[cur].len && v[cur].nxt.find(t[i]) == v[cur].nxt.cend()) {
//       cur = v[cur].link;
//       l = v[cur].len;
//     }
//     if (v[cur].nxt.find(t[i]) != v[cur].nxt.cend()) cur = v[cur].nxt[t[i]], l++;
//     if (t.length() == l) pos.push_back(i-l+2);
//   }
//   return pos;
// }
void sort() {
  sv = vector<pii>(v.size());
  for (int i = 0; i < v.size(); i++) sv[i] = { v[i].len, i };
  auto comp = [](pii a, pii b) -> bool {
    return a.first > b.first;
  };
    std::sort(sv.begin(), sv.end(), comp);
  }
};
Suffix_Automaton sa;
```

8 etc

8.1 Deque Trick

```
int n, l;
deque<pii> dq;
cin >> n >> l;
for (int i = 0; i < n; i++) {
  int x;
  cin >> x;
  while (dq.size() && dq.back().first >= x) dq.pop_back();
  dq.push_back({x, i});
  while (dq.size() && dq.front().second <= i-l) dq.pop_front();
  cout << dq.front().first << " ";
}
```

8.2 imos rectangle and diamond

```
const int FIX = 5, DIV = 1e9;
int ty, px, py, qx, qy, r;
for (int i = 0; i < k; i++) {
  cin >> ty;
  if (ty == 1) {
    cin >> px >> py >> qx >> qy;
    px += FIX;
    py += FIX;
    qx += FIX;
    qy += FIX;
    rect[py][px] += 1;
    rect[qy+1][px] -= 1;
    rect[py][qx+1] -= 1;
    rect[qy+1][qx+1] += 1;
  } else {
    cin >> px >> py >> r;
    px += FIX;
    py += FIX;
    dia[py-r][px] += 1;
    dia[py-r+1][px] += 1;
    dia[py+1][px-r-1] -= 1;
    dia[py+1][px-r] -= 1;
    dia[py+1][px+r+1] -= 1;
    dia[py+1][px+r] -= 1;
    dia[py+r+1][px] += 1;
    dia[py+r+2][px] -= 1;
  }
}
for (int i = 0; i < h+FIX; i++) for (int j = 1; j < w+FIX; j++) rect[i][j] += rect[i][j-1];
for (int i = 0; i < w+FIX; i++) for (int j = 1; j < h+FIX; j++) rect[j][i] += rect[j-1][i];
for (int i = 1; i < h+FIX; i++) for (int j = 1; j < w+FIX; j++) dia[i][j] += dia[i-1][j-1];
for (int i = 1; i < h+FIX; i++) for (int j = 0; j < w-1+FIX; j++) dia[i][j] +=
dia[i-1][j+1];
for (int i = 0+FIX; i < h+FIX; i++) {
  for (int j = 0+FIX; j < w+FIX; j++) {
    if ((rect[i][j] + dia[i][j]) % 2) cout << "#";
    else cout << ".";
  }
  cout << "\n";
}
```

8.3 imos triangle

```
int x, y, s; // (x, y), (x+s, y), (x+s, y+s)를 정점으로 하는 삼각형
for (int i = 0; i < m; i++) {
  cin >> x >> y >> s;
  pos[x][y]++;
  pos[x][y+1]--;
  pos[x+s+1][y]--;
  pos[x+s+2][y+1]++;
  pos[x+s+1][y+s+2]++;
  pos[x+s+2][y+s+2]--;
}
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++) pos[i][j] += pos[i][j-1];
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++) pos[i][j] += pos[i-1][j];
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++) pos[i][j] += pos[i-1][j-1];
int cnt = 0;
for (int i = 1; i <= n; i++) for (int j = 1; j <= i; j++) if (pos[i][j] > 0) cnt++;
```

8.4 mo's algorithm

```
// O(nsqrt(n))
struct Query {
  ll l, r;
  int i;
};
ll n, m, cnt[100001], cntn[100001], ans[100001], sqrtn, a[100001];
Query query[100000];
int main() {
  cin >> n;
  for (int i = 1; i <= n; i++)
    cin >> a[i];
  cin >> m;
  for (int i = 0; i < m; i++) {
    cin >> query[i].l >> query[i].r;
    query[i].i = i;
  }
  sqrtn = sqrt(n);
  sort(query, query+m, [](Query q1, Query q2){
    if (q1.l/sqrtn == q2.l/sqrtn) return q1.r < q2.r;
    else return q1.l/sqrtn < q2.l/sqrtn;
  });

  cntn[0] = n;

  ll mcnt = 0, s = query[0].l, e = query[0].r;
  for (int i = s; i <= e; i++) {
    cntn[cnt[a[i]]]--;
    cnt[a[i]]++;
    cntn[cnt[a[i]]]++;
    mcnt = max(mcnt, cnt[a[i]]);
  }
```

```
  ans[query[0].i] = mcnt;

  int temp;
  for (int i = 1; i < m; i++) {
    while (s < query[i].l) {
      cntn[cnt[a[s]]]--;
      cnt[a[s]]--;
      cntn[cnt[a[s]]]++;
      s++;
      while (cntn[mcnt] == 0) mcnt--;
    }
    while (query[i].l < s) {
      s--;
      cntn[cnt[a[s]]]--;
      if (cnt[a[s]] == mcnt) mcnt++;
      cnt[a[s]]++;
      cntn[cnt[a[s]]]++;
    }
    while (e < query[i].r) {
      e++;
      cntn[cnt[a[e]]]--;
      if (cnt[a[e]] == mcnt) mcnt++;
      cnt[a[e]]++;
      cntn[cnt[a[e]]]++;
    }
    while (query[i].r < e) {
      cntn[cnt[a[e]]]--;
      cnt[a[e]]--;
      cntn[cnt[a[e]]]++;
      e--;
      while (cntn[mcnt] == 0) mcnt--;
    }
    ans[query[i].i] = mcnt;
  }
}
```

8.5 random generator

```
// init
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<int> dis(0, n);
// use
int x = dis(gen);
```

9 Note and Checklist

9.1 etc formular

```
// Area = I + B/2 - 1, B: 경계격자점, I: 내부격자점
// C^2 = A^2 + B^2 - 2ABcos
// 카탈란수: Cn = { 1/(n+1) } * (2n C n)
```

9.2 pisano

```
// 주기의 길이를 P라고 하면 N번째 피보나치 수를 M으로 나눈 나머지는 N%P번째 피보나치 수를 M으로 나눈 나머지와 같다.
// 주기는 M = 10^k (k>2)일때, 항상 15 * 10^(k-1)이다.
```

9.3 Ternary Search

```
/*
while (l+2 < r)로 해두고
i = l ~ r까지 순회해서 찾기
*/
```

9.4 문제 풀이 체크리스트

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 구간을 통째로 가져간다 : 플로우 + 적당한 자료구조 $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- a = b : a만 움직이기, b만 움직이기, 두 개 동시에 움직이기, 반대로 움직이기
- 말도 안 되는 것들을 한 번은 생각해보기 / "당연하다고 생각한 것" 다시 생각해보기
- 확률 : DP, 이분 탐색(NYPC 2019 Finals C)
- 최대/최소 : 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)