Team Note of TTD the next generation

Le Kien Thanh, Ngo Huy Tin, Trinh Khanh Dung

Compiled on November 16, 2024

Contents

1 Misc

1.1 Generic Template

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

using namespace std;

typedef long long ll;
typedef unsigned int ul;
typedef unsigned long long ull;

#define MASK(i) (1ULL << (i))
#define GETBIT(mask, i) (((mask) >> (i)) & 1)
#define ALL(v) (v).begin(), (v).end()

ll gcd(ll a, ll b){return __gcd(a, b);}
ll max(ll a, ll b){return (a > b) ? a : b;}
ll min(ll a, ll b){return (a < b) ? a : b;}

ll LASTBIT(ll mask){return (mask) & (-mask);}
int pop_cnt(ll mask){return __builtin_popcountll(mask);}
int ctz(ull mask){return __builtin_ctzll(mask);}
int logOf(ull mask){return 63 - __builtin_clzll(mask);}

mt19937_64
rng(chrono::high_resolution_clock::now().time_since_epoch().count());
ll rngesus(ll l, ll r){return l + (ull) rng() % (r - l + 1);}

template <class T1, class T2>
    bool maximize(T1 &a, T2 b){
        if (a < b) {a = b; return true;}
        return false;
    }

template <class T1, class T2>
    bool minimize(T1 &a, T2 b){
        if (a > b) {a = b; return true;}
        return false;
    }

template <class T>
    void printArr(T the_array_itself, string separator = " ",
    string finish = "\n", ostream &out = cout){
        for(auto item: the_array_itself) out << item <<
        separator;
        out << finish;
    }

template <class T>
    void remove_dup(vector<T> &a){
        sort(ALL(a));
        a.resize(unique(ALL(a)) - a.begin());
    }


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

    clock_t start = clock();


    cerr << "Time elapsed: " << clock() - start << " ms!\n";

    return 0;
}
```

1.2 FastIO

```
/*
 Usage: FastIO in; in.init(); in >> n;
*/

struct FastIO{
    string buffer;
    vector<ll> king;
    void init(ll _n = 0){
        getline(cin, buffer, '\0');
        buffer += "\n";
        king.reserve(_n);
        ll cur = 0;
        for(char c: buffer){
            if (c <= '9' && c >= '0'){
                if (cur == -1) cur = 0;
                cur = cur * 10 + (c - '0');
            }
            else{
                if (cur != -1) king.push_back(cur);
                cur = -1;
            }
        }

        reverse(ALL(king));
    }

    template <class T>
    FastIO& operator >> (T &x){
        x = 0;
        if (king.empty()) return *this;
        x = king.back(); king.pop_back();
        return *this;
    }

    bool isEmpty(){return king.empty();}
};
```

1.3 Stress Tester

```
@echo off
title My futile attempt at writing a stress tester.

echo Compiling files, please wait...
g++ -std=c++14 -o test.exe test.cpp
echo Done compiling test generator!

g++ -std=c++14 -o brute.exe brute.cpp
echo Done compiling bruteforce submission!

g++ -std=c++14 -o sol.exe sol.cpp
echo Done compiling (possibly) faulty submission!

for /l %%x in (1, 1, 1000000) do (
  echo Testcase numero: %%x
  test > input.inp
  sol < input.inp > output.out
  brute < input.inp > output.ans
  fc output.out output.ans > log.txt || call :WA

)
echo Accepted!
pause
exit

:WA
echo Wrong Answer
echo Test case:
type input.inp
echo Participant answer:
type output.out
echo Jury answer:
type output.ans

pause
exit
```

1.4 Bench Marker

```
@echo off
title Bench Marker 6000
echo Compiling files, please wait...
g++ -O2 -Wl,--stack=268435456 -std=c++11   -o test.exe
test.cpp
echo Done compiling test generator!
g++ -O2 -Wl,--stack=268435456 -std=c++11   -o sol.exe sol.cpp
echo Done compiling (probably) faulty submission!

for /l %%x in (1, 1, 1000) do (
  echo Test case numero: %%x
  test > input.inp
  sol < input.inp > output.out
)

pause
exit
```

1.5 CP Sublime 3 Build

```
{
    "encoding": "utf-8",
    "working_dir": "$file_path",
    "shell_cmd": "g++ -Wall -O2 -std=c++14 -o
    ${file_base_name}.exe ${file_base_name}.cpp",
    "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
    "selector": "source.c++,source.c",
    "variants":
    [
        {
            "name": "Run",
            "shell_cmd": "g++ -Wall -O2 -std=c++14 \"${file}\"
            -o \"${file_base_name}\" && start cmd /c
            \"\"${file_path}/${file_base_name}\" & pause\""
        }
    ]
}
```

1.6 CP Sublime 3 Keymap

```
[

  { "keys": ["f9"], "command": "build" },

  { "keys": ["ctrl+shift+x"], "command": "toggle_comment",
  "args": { "block": false } },

  { "keys": ["ctrl+shift+c"], "command": "toggle_comment",
  "args": { "block": false } },

  { "keys": ["ctrl+f12"], "command": "sort_lines", "args":
  {"case_sensitive": false} },
  { "keys": ["ctrl+shift+f12"], "command": "sort_lines",
  "args": {"case_sensitive": true} },
]
```

2 Data Structure

2.1 Convex Hull Trick

```
/*
CHT template
*/

const ll INF = 1e18 + 69;

struct CHT{
    #define Node pair<ll, ll>
    vector<Node> a;
    vector<double> bubble;
    bool flag;

    // if flag = 0: if inserted in increasing order, return
    min, otherwise return max
    // flag = 1: the opposite

    CHT(bool _flag){
        flag = _flag;
    }
```

```
    double getInter(Node a, Node b){
        double x = (double) (b.second - a.second) / (a.first -
        b.first);
        return x;
    }

    void add(Node x, bool isMin = 1){
        if (a.empty()) {a.push_back(x); return;}
        if (a.back().first == x.first){
            if (isMin) minimize(a[a.size() - 1].second,
            x.second);
            else maximize(a[a.size() - 1].second, x.second);
            return;
        }
        while(a.size() >= 2){
            double x1 = getInter(a.back(), x), x2 =
            bubble.back();
            if (flag){
                if (x1 >= x2) break;
            }
            else{
                if (x1 <= x2) break;
            }
            a.pop_back(); bubble.pop_back();
        }
        bubble.push_back(getInter(a.back(), x));
        a.push_back(x);
    }

    ll get(ll x){
        if (a.empty()) return INF;
        long idx;
        if (flag) idx = lower_bound(ALL(bubble),x) -
        bubble.begin();
        else idx = lower_bound(ALL(bubble), x,
        greater<double>()) - bubble.begin();
        return a[idx].first * x + a[idx].second;
    }
};
```

2.2 Fast Segment Tree

```
/*
    Fast Seg Tree
*/

struct SegmentTree{
    int n;
    vector<int> a;

    SegmentTree(int _n){
        n = _n;
        a.resize(n * 2 + 2, -INF);
    }

    void update(int i, int v){
        i += n; if (!maximize(a[i], v)) return;
        while(i > 1){
            i >>= 1;
            a[i] = max(a[i * 2], a[i * 2 + 1]);
        }
    }

    int get(int l, int r){
        l += n; r += n + 1;
        int ans = -INF;
        while(l < r){
            if (l & 1) maximize(ans, a[l++]);
            if (r & 1) maximize(ans, a[--r]);
            l >>= 1; r >>= 1;
        }
        return ans;
    }
};
```

2.3 Fast Segment Tree Lazy

```
/*
  Fast segtree lazy
*/

struct SegmentTree{
    int n, h;
    vector<int> t, d;

    SegmentTree(int _n){
        n = _n;
        h = logOf(n);
        t.resize(n * 2 + 2), d.resize(n * 2 + 2);
        for(int i = 1; i <= n * 2 + 1; ++i) t[i] = d[i] = 0;
    }

    void apply(int p, int value) {
        t[p] += value;
        if (p < n) d[p] += value;
    }

    void build(int p) {
        while (p > 1) p >>= 1, t[p] = min(t[p<<1], t[p<<1|1])
        + d[p];
    }

    void push(int p) {
      for (int s = h; s > 0; --s) {
        int i = p >> s;
        if (d[i] != 0) {
          apply(i<<1, d[i]);
          apply(i<<1|1, d[i]);
          d[i] = 0;
        }
      }
    }

    void update(int l, int r, int value) {
      l += n, r += n+1;
      int l0 = l, r0 = r;
      for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) apply(l++, value);
        if (r&1) apply(--r, value);
      }
      build(l0);
      build(r0 - 1);
    }

    int get() {
        int l = 1, r = n;
      l += n, r += n+1;
      push(l);
      push(r - 1);
      int res = INF;
      for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) res = min(res, t[l++]);
        if (r&1) res = min(t[--r], res);
      }
      return res;
    }
};
```

2.4 Lichao Tree

```
/*
Lichao Tree
*/

const ll INF = 1e18;

struct LichaoTree{
    #define Node pair<ll, ll>

    long n;
    bool isMin;
    vector<Node> a;

    LichaoTree(long _n, bool flag = 1){
        n = _n;
        isMin = flag;
        if (flag) a.resize(n * 4 + 4, make_pair(0, INF));
        else a.resize(n * 4 + 4, make_pair(0, -INF));
    }
```

```
    ll f(Node t, ll x){return t.first * x + t.second;}

    void update(Node x, long l, long r, long id){
        bool check1 = f(a[id], l) >= f(x, l), check2 =
        f(a[id], r) >= f(x, r);
        bool check3 = f(a[id], l) <= f(x, l), check4 =
        f(a[id], r) <= f(x, r);
        if (check1 && check2){
            if (isMin) a[id] = x;
            return;
        }
        if (check3 && check4){
            if (!isMin) a[id] = x;
            return;
        }
        if (l == r) return;
        long mid = (l + r) >> 1;
        update(x, l, mid, id * 2);
        update(x, mid + 1, r, id * 2 + 1);
    }

    void update(Node x){update(x, 1, n, 1);}

    ll get(long i, long l, long r, long id){
        ll ans1 = f(a[id], i);
        if (l == r) return ans1;
        long mid = (l + r) >> 1;

        ll ans2;
        if (i <= mid) ans2 = get(i, l, mid, id * 2);
        else ans2 = get(i, mid + 1, r, id * 2 + 1);

        if (isMin) return min(ans1, ans2);
        else return max(ans1, ans2);
    }

    ll get(long i){return get(i, 1, n, 1);}
};
```

2.5 Persistent Segment Tree

```
class PersistentSegmentTree {
private:
    int n;
    vector<Node*> roots; // Store roots of different versions

    // Build initial segment tree
    Node* build(int start, int end) {
        if (start == end) return new Node(0); // Initialize
        with zero; customize as needed
        int mid = (start + end) / 2;
        return new Node(0, build(start, mid), build(mid + 1,
        end));
    }

    // Create new version with point update
    Node* update(Node* prev, int start, int end, int idx, int
    val) {
        if (start == end) return new Node(prev->value + val);
        // Update node value
        int mid = (start + end) / 2;
        if (idx <= mid)
            return new Node(prev->value + val,
            update(prev->left, start, mid, idx, val),
            prev->right);
        else
            return new Node(prev->value + val, prev->left,
            update(prev->right, mid + 1, end, idx, val));
    }

    // Query for a range [L, R]
    int query(Node* node, int start, int end, int L, int R) {
        if (start > R || end < L) return 0; // Return 0 for
        sum; adjust for other ops
        if (L <= start && end <= R) return node->value;
        int mid = (start + end) / 2;
        return query(node->left, start, mid, L, R) +
        query(node->right, mid + 1, end, L, R);
    }

public:
    PersistentSegmentTree(int size) : n(size) {
        roots.push_back(build(0, n - 1)); // Initial version
    }

    // Update tree version by creating a new version
    void update(int idx, int val) {
        roots.push_back(update(roots.back(), 0, n - 1, idx,
        val));
    }

    // Query a specific version
    int query(int version, int L, int R) {
        return query(roots[version], 0, n - 1, L, R);
    }
};
```

2.6 Treap

```
/*
Treap template
*/

struct BST{
    struct Node{
        int val, priority;
        Node* child[2];
        Node(int val): val(val), priority(rngesus(0, MASK(30)
        - 1)){
            child[0] = child[1] = nullptr;
        }
    };

    Node* root;

    BST(){
        root = nullptr;
    }

    void split(Node* cur, int x, Node* & L, Node*& R){ //
    split up to a point x
        if (cur == nullptr) {L = R = nullptr;return;}
        if (x < cur -> val){
            split(cur -> child[0], x, L, cur -> child[0]);
            R = cur;
        }
        else{
            split(cur -> child[1], x, cur -> child[1], R);
            L = cur;
        }
    }

    void merge(Node*& cur, Node* L, Node* R){
        if (L == nullptr) cur = R;
        else if (R == nullptr) cur = L;
        else{
            if (R -> priority > L -> priority) {
                cur = R;
                merge(cur -> child[0], L, R -> child[0]);
            }
            else{
                cur = L;
                merge(cur -> child[1], L -> child[1], R);
            }
        }
    }

    void insert(int x){
        Node *L, *R;
        split(root, x, L, R);
        merge(root, L, new Node(x));
        merge(root, root, R);
    }

    void erase(int x){
        Node *L = nullptr, *mid = nullptr, *R = nullptr;
        split(root, x-1, L, mid);
        split(mid, x, mid, R);
```

```
        merge(root, L, R);
    }

    bool find(int x){
        Node *id = root;
        while(true){
            if (id == nullptr) return false;
            if (id -> val == x) return true;
            else if (x < id -> val) id = id -> child[0];
            else id = id -> child[1];
        }
    }

    void clear(){
        while(root != nullptr){
            int val = root -> val;
            erase(val);
        }
    }
};
```

3 Geometry

3.1 Boiler Plate

```
/*
  Just some good old boiler plate code
*/

struct PointRect{
    double x, y;
    PointRect(){}
    PointRect(double _x, double _y){
        x = _x, y = _y;
    }
};

struct PointPolar{
    double theta, len;
    PointPolar(){}
    PointPolar(double _theta, double _len){
        theta = _theta; len = _len;
    }
};

const double PI = acos(0) * 2;

PointPolar toPolar(PointRect p){
    double x = p.x, y = p.y;
    double len = sqrt(p.x * p.x + p.y * p.y);
    if (x== 0){
        if (y>= 0) return PointPolar(PI / 2, sqrt(p.x * p.x +
        p.y * p.y));
        return PointPolar(PI * 3 / 2, sqrt(p.x * p.x + p.y *
        p.y));
    }
    if (y == 0){
        if (x > 0) return PointPolar(0, sqrt(p.x * p.x + p.y *
        p.y));
        return PointPolar(PI,sqrt(p.x * p.x + p.y * p.y));
    }

    double more = 0;
    while(x < 0 || y < 0){
        more += PI / 2;
        swap(x, y);
        y = -y;
    }
    return PointPolar(more + atan(y / x), len);
}

PointRect toRect(PointPolar p){
    double theta = p.theta, len = p.len;
    int cnt = 0;
    while(theta >= PI / 2) {
        theta -= PI / 2; cnt++;
    }
    PointRect v = PointRect(cos(theta) * len, sin(theta) *
    len);
    while(cnt--){
        v.y = -v.y;
        swap(v.x, v.y);
    }
    return v;
}

double getArea(PointRect a, PointRect b, PointRect c){
    return a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y
    - b.y);
}


array<double, 3> getFunction(pair<PointRect, PointRect> x){
    array<double, 3> ans;
    ans[0] = x.first.y - x.second.y;
    ans[1] = x.second.x - x.first.x;
    ans[2] = -(ans[0] * x.first.x + ans[1] * x.first.y);
    return ans;
}

PointRect getIntersection(pair<PointRect, PointRect> line1,
pair<PointRect, PointRect> line2){
    array<double, 3> slope1 = getFunction(line1), slope2
    =getFunction(line2);
    double delta = (slope1[1] * slope2[0] - slope2[1] *
    slope1[0]);
    if (abs(delta) <= EPS) return PointRect(INF * 2, INF * 2);
    double y = -(slope1[2] * slope2[0] - slope2[2] *
    slope1[0]) / delta;
    double x = (slope1[2] * slope2[1] - slope2[2] * slope1[1])
    / delta;
    return PointRect(x, y);
}

void find_convex_hull(vector<PointRect> &a){
    if (a.size() <= 2) return;
    sort(ALL(a), [] (PointRect x, PointRect y){
        if (x.x != y.x) return x.x < y.x;
        return x.y < y.y;
    });

    PointRect l = a[0], r = a.back();
    vector<PointRect> up, down;
    up.push_back(l); down.push_back(l);
    for(long i = 1; i<a.size(); ++i){
        if (getArea(l, r, a[i]) >= 0){
            while(up.size() > 1 && getArea(up[up.size() - 2],
            up[up.size() - 1], a[i]) > 0) up.pop_back();
            up.push_back(a[i]);
        }
        if (getArea(l, r, a[i]) <= 0){
            while(down.size() > 1 && getArea(down[down.size()
            - 2], down[down.size() - 1], a[i]) < 0)
            down.pop_back();
            down.push_back(a[i]);
        }
    }
    a = up;
    for(long i = down.size() - 2; i>=1; --i)
    a.push_back(down[i]);
}
```

3.2 Manhattan Minimum Spanning Tree

```
struct Point{
    ll x, y;
    Point(){}
    Point(ll x, ll y): x(x), y(y){}

    Point& operator -= (Point a){
        x -= a.x;
        y -= a.y;
        return *this;
    }
    Point operator - (Point x) const {
        Point ans = *this;
```

```
        return ans -= x;
    }
};


struct Path{
    int u, v; ll w;
    Path(){}
    Path(int _u, int _v, ll _w){u = _u, v  =_v, w = _w;}
};

const ll INF = 1e18;
namespace ManhattanSpanningTreeSolver {
    #define sz(v) ((int)v.size())

    vector<Path> solve(vector<Point> ps) {
        vector<int> id(sz(ps));
        iota(ALL(id), 0);
        vector<Path> edges;
        for(int k = 0; k < 4; ++k) {
            sort(ALL(id), [&](int i, int j) {
                return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
            map<ll, int> sweep;
            for (int i : id) {
                for (auto it = sweep.lower_bound(-ps[i].y);
                            it != sweep.end();
                            sweep.erase(it++)) {
                    int j = it->second;
                    Point d = ps[i] - ps[j];
                    if (d.y > d.x) break;
                    edges.push_back(Path(i, j, d.y + d.x));
                }
                sweep[-ps[i].y] = i;
            }
            for (Point& p : ps) if (k & 1) p.x = -p.x; else
            swap(p.x, p.y);
        }
        return edges;
    }
};
```

3.3 Smallest Enclosing Circle

```
struct Point {
    double x, y;
    Point(double _x, double _y) : x(_x), y(_y) {}
};

struct Circle {
    Point center;
    double radius;
    Circle(Point c, double r) : center(c), radius(r) {}
};

// Function to calculate the Euclidean distance between two
points
double distance(const Point& p1, const Point& p2) {
    return hypot(p1.x - p2.x, p1.y - p2.y);
}

// Check if a point is inside a given circle
bool isPointInsideCircle(const Point& p, const Circle& c) {
    return distance(p, c.center) <= c.radius;
}

// Create a circle from two points
Circle circleFromTwoPoints(const Point& p1, const Point& p2) {
    Point center((p1.x + p2.x) / 2.0, (p1.y + p2.y) / 2.0);
    double radius = distance(p1, p2) / 2.0;
    return Circle(center, radius);
}

// Create a circle from three points using the circumcircle
formula
Circle circleFromThreePoints(const Point& p1, const Point& p2,
const Point& p3) {
    double ax = p1.x, ay = p1.y;
    double bx = p2.x, by = p2.y;
    double cx = p3.x, cy = p3.y;

    double d = 2 * (ax * (by - cy) + bx * (cy - ay) + cx * (ay
    - by));
    if (d == 0) throw runtime_error("Collinear points");

    double ux = ((ax * ax + ay * ay) * (by - cy) + (bx * bx +
    by * by) * (cy - ay) + (cx * cx + cy * cy) * (ay - by)) /
    d;
    double uy = ((ax * ax + ay * ay) * (cx - bx) + (bx * bx +
    by * by) * (ax - cx) + (cx * cx + cy * cy) * (bx - ax)) /
    d;

    Point center(ux, uy);
    double radius = distance(center, p1);
    return Circle(center, radius);
}

// Welzl's algorithm to find the minimum enclosing circle
Circle welzlAlgorithm(vector<Point>& points, vector<Point>
boundary = {}) {
    if (points.empty() || boundary.size() == 3) {
        if (boundary.empty()) return Circle(Point(0, 0), 0);
        if (boundary.size() == 1) return Circle(boundary[0],
        0);
        if (boundary.size() == 2) return
        circleFromTwoPoints(boundary[0], boundary[1]);
        return circleFromThreePoints(boundary[0], boundary[1],
        boundary[2]);
    }

    Point p = points.back();
    points.pop_back();

    Circle d = welzlAlgorithm(points, boundary);

    if (isPointInsideCircle(p, d)) {
        points.push_back(p);
        return d;
    }

    boundary.push_back(p);
    Circle result = welzlAlgorithm(points, boundary);
    points.push_back(p);
    return result;
}

Circle findMinimumEnclosingCircle(vector<Point>& points) {
    srand(time(0));
    random_shuffle(points.begin(), points.end());
    return welzlAlgorithm(points);
}
```

3.4 Circle vs CCW Polygon Intersection

```
/*
 * Usage :
 * Returns the area of the intersection of a circle with a CCW
 polygon.
 * Time complexity : O(n)
*/

typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b =
    (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
```

```
  return sum;
}
```

3.5 Circle vs Line Intersect

```
/*
 * Usage :
 * Finds the intersection between a circle and a line.
 * Returns a vector of either 0, 1, or 2 intersection points.
 * P is intended to be Point<double>.
 */

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

3.6 Circles Intersect

```
/*
 * Usage :
 * Computes the pair of points at which two circles intersect.
 -> out
 * Returns false in case of no intersection.
 */

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out)
{
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) /
  d2);
  *out = {mid + per, mid - per};
  return true;
}
```

3.7 Hull Diameter

```
/*
 * Usage :
 * Returns the two points with max distance on a convex hull
 (CCW, no duplicate/collinear points).
 * Time complexity : O(n)
*/

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

3.8 Point in Poly

```
/*
 * Usage :
 * Returns true if p lies within the polygon. If strict is
 true,
 * it returns false for points on the boundary.
 * The algorithm uses products in intermediate steps so watch
 out for overflow.
 * Time complexity : O(n)
*/

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

4 Graph

4.1 Block Cut Tree

```
/*
Usage:
- generate_block_cut_tree(): duh, also return number of
vertices
- cc[u]: vertices inside block u
- sz[u]: number of vertices, -1 if is joint
*/


const int N = 1e5 + 69;
int n, m;
vector<int> graph[N], dfs_tree[N];

int num[N], low[N];
bool isJoint[N];
vector<int> st;
int dfs_cnt;
vector<vector<int>> bbc;

void tarjan(int u, int p){
    num[u] = low[u] = ++dfs_cnt;

    vector<int> child;
    for(int v: graph[u]) {
        if (v == p) {p = -1; continue;}
        if (num[v]){
            minimize(low[u], num[v]);
        }
        else{
            child.push_back(v);
            tarjan(v, u);
            minimize(low[u], low[v]);
            if (low[v] >= num[u]){
                isJoint[u] = true;
            }
        }
    }
    if (u == p) isJoint[u] = isJoint[u] && (child.size() >=
    2);
    for(int v: child){
        if (isJoint[u] && low[v] >= num[u])
        dfs_tree[u].push_back(-v);
        else dfs_tree[u].push_back(v);
    }
}

void get_block(int u){
    st.push_back(u);
    for(int v: dfs_tree[u]){
        get_block(abs(v));
        if (v < 0){
            bbc.push_back({});
            while(bbc.back().empty() || bbc.back().back() !=
            abs(v)){
                bbc.back().push_back(st.back());
                st.pop_back();
            }
            bbc.back().push_back(u);
        }
    }
}

vector<int> bc_tree[2*N];
int joint_cnt = 0;
int pos[2*N];
```

```
int sz[2 * N];
vector<int> cc[2 * N];

int generate_block_cut_tree(){
    dfs_cnt = 0;
    for(int i= 1; i<=n; ++i) if (num[i] == 0){
        tarjan(i, i);
        get_block(i);
        if (st.size()) {
            bbc.push_back(st);
            st.clear();
        }
    }

    int idx = 0;
    for(int i= 1; i<=n; ++i){
        if (!isJoint[i]) continue;
        joint_cnt++;
        pos[i] = ++idx;
        cc[idx].push_back(-i);
        sz[idx] = 1;
    }

    for(vector<int> i: bbc){
        ++idx;
        sz[idx] = i.size();
        cc[idx] = i;
        for(int j: i){
            if (isJoint[j]){
                int u = idx, v = pos[j];
                sz[v]--;
                bc_tree[u].push_back(v);
                bc_tree[v].push_back(u);
            }
        }
    }
    return idx;
}
```

4.2 Eulerian Path

```
/*
Usage:
- dfs(u, p): dfs, what the fuck
- Return the order of edge you enter (in reverse)
*/

vector<int> ans;
void find_eulerian_path(int u){
    while(graph[u].size()){
        pair<int, int> v = graph[u].back();
        graph[u].pop_back();
        if (used[v.second]) continue;
        used[v.second] = true;
        find_eulerian_path(v.first);
        ans.push_back(v.first);
    }
}
```

4.3 FLow With Demand

```
/*
Usage:
 - Dependency: Dinic template.
 - FlowWithDemand(n, s, t): initialize a graph of size n, with
 s and t being the source and sink nodes, respectively
 - add_edge(u, v, d, w): add an edge from u to v, with
 capacity w, and demand d.
 - try_edge(capacity): verify whether there exists a flow,
 such that the total residue is <= capacity.
*/

struct FlowWithDemand {
    int n, s, t, fake_s, fake_t;
    vector<long long> balance;
    Dinic dinic;

    FlowWithDemand(int n, int source, int sink)
        : n(n), s(source), t(sink), fake_s(n), fake_t(n + 1),
        balance(n, 0), dinic(n + 2, n, n + 1) {}

    void add_edge(int u, int v, long long demand, long long
    capacity) {
        balance[u] -= demand;
        balance[v] += demand;
        dinic.add_edge(u, v, capacity - demand);
    }

    bool try_flow(long long capacity) {
        vector<int> extras;
        for (int i = 0; i < n; ++i) {
            if (balance[i] > 0) {
                dinic.add_edge(fake_s, i, balance[i]);
                extras.push_back(fake_s);
                extras.push_back(i);
            } else if (balance[i] < 0) {
                dinic.add_edge(i, fake_t, -balance[i]);
                extras.push_back(fake_t);
                extras.push_back(i);
            }
        }
        extras.push_back(s); extras.push_back(t);
        dinic.add_edge(t, s, capacity);

        bool valid = dinic.max_flow() >=
        accumulate(balance.begin(), balance.end(), 0LL,
        [](long long sum, long long x) {
            return sum + max(0LL, x);
        });

        for(int i: extras) {
            dinic.adj[i].pop_back();
        }

        return valid;
    }
};
```

4.4 General Max Matching

```
/*
Usage:
 - Blossom(n): initialize a graph of size n
 - add_edge(u, v): add an edge from u to v
 - max_matching(): compute max matching
*/

struct Blossom {
    int N;
    vector<int> match, par, base, vis;
    vector<vector<int>> adj;
    queue<int> Q;

    Blossom(int n) : N(n), match(n, -1), par(n, -1), base(n),
    vis(n, -1), adj(n) {
        iota(base.begin(), base.end(), 0);
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int find_lca(int u, int v) {
        vector<bool> visited(N);
        while (1) {
            u = base[u];
            visited[u] = true;
            if (match[u] == -1) break;
            u = par[match[u]];
        }
        while (1) {
            v = base[v];
            if (visited[v]) return v;
            v = par[match[v]];
        }
    }

    void blossom(int u, int v, int lca) {
```

```
        while (base[u] != lca) {
            par[u] = v;
            v = match[u];
            if (vis[v] == 1) Q.push(v), vis[v] = 0;
            base[u] = base[v] = lca;
            u = par[v];
        }
    }

    bool bfs(int root) {
        fill(vis.begin(), vis.end(), -1);
        Q = queue<int>();
        Q.push(root), vis[root] = 0;

        while (!Q.empty()) {
            int u = Q.front();
            Q.pop();
            for (int v : adj[u]) {
                if (vis[v] == -1) {
                    vis[v] = 1, par[v] = u;
                    if (match[v] == -1) {
                        while (u != -1) {
                            int next_u = match[u];
                            match[u] = v, match[v] = u;
                            v = next_u, u = (v == -1) ? -1 :
                            par[v];
                        }
                        return true;
                    }
                    Q.push(match[v]), vis[match[v]] = 0;
                } else if (vis[v] == 0 && base[u] != base[v])
                {
                    int lca = find_lca(u, v);
                    blossom(u, v, lca);
                    blossom(v, u, lca);
                }
            }
        }
        return false;
    }

    int max_matching() {
        int res = 0;
        for (int u = 0; u < N; ++u)
            if (match[u] == -1 && bfs(u))
                ++res;
        return res;
    }
};
```

4.5 Max Flow

```
/*
Usage:
 - Dinic(n, s, t): initialize a graph of size n, with s and t
 being the source and sink nodes, respectively
 - add_edge(u, v, w): add an edge from u to v, with capacity
 w.
 - max_flow(): compute maximum flow
*/
struct Dinic {
    struct Edge { int to, rev; ll cap, flow; };
    int n, s, t;
    vector<vector<Edge>> adj;
    vector<int> level, ptr;

    Dinic(int n, int source, int sink) : n(n), s(source),
    t(sink), adj(n), level(n), ptr(n) {}

    void add_edge(int u, int v, ll cap) {
        adj[u].push_back({v, (int)adj[v].size(), cap, 0});
        adj[v].push_back({u, (int)adj[u].size() - 1, 0, 0});
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        queue<int> q({s});
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto &e : adj[u]) {
                if (e.cap - e.flow > 0 && level[e.to] == -1) {
                    level[e.to] = level[u] + 1;
                    q.push(e.to);
                }
            }
        }
        return level[t] != -1;
    }

    ll dfs(int u, ll flow) {
        if (u == t || flow == 0) return flow;
        for (; ptr[u] < (int)adj[u].size(); ++ptr[u]) {
            Edge &e = adj[u][ptr[u]];
            if (level[e.to] == level[u] + 1 && e.cap - e.flow
            > 0) {
                ll pushed = dfs(e.to, min(flow, e.cap -
                e.flow));
                if (pushed > 0) {
                    e.flow += pushed;
                    adj[e.to][e.rev].flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
    }

    void reset_flows() {
        for (auto &u_edges : adj) {
            for (auto &e : u_edges) {
                e.flow = 0; // Reset flow to 0
            }
        }
    }

    ll max_flow() {
        ll flow = 0;
        while (bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
            while (ll pushed = dfs(s, LLONG_MAX)) {
                flow += pushed;
            }
        }
        reset_flows();
        return flow;
    }
};
```

4.6 Max Matching

```
/*
Usage:
 - Matching(n, s, t): initialize a graph of size n
 - add_edge(u, v): add an edge from u to v
 - max_matching(): return maximum matching
*/

struct Matching {
    int n; vector<vector<int>> adj; vector<int> pairU, pairV,
    dist, color;
    Matching(int n) : n(n), adj(n + 1), pairU(n + 1), pairV(n
    + 1), dist(n + 1), color(n + 1, -1) {}

    void add_edge(int u, int v) { adj[u].push_back(v),
    adj[v].push_back(u); }

    bool bfs() {
        queue<int> q;
        for (int u = 1; u <= n; ++u)
            if (!pairU[u] && color[u] == 0) dist[u] = 0,
            q.push(u);
            else dist[u] = INT_MAX;
        dist[0] = INT_MAX;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v : adj[u])
                if (dist[pairV[v]] == INT_MAX)
                    dist[pairV[v]] = dist[u] + 1,
                    q.push(pairV[v]);
```

```
        }
        return dist[0] != INT_MAX;
    }

    bool dfs(int u) {
        if (!u) return true;
        for (int v : adj[u])
            if (dist[pairV[v]] == dist[u] + 1 &&
            dfs(pairV[v]))
                return pairV[v] = u, pairU[u] = v, true;
        dist[u] = INT_MAX;
        return false;
    }

    vector<pair<int, int>> max_matching() {
        function<void(int)> color_graph = [&](int u) {
            for (int v : adj[u]) if (color[v] == -1) color[v]
            = 1 - color[u], color_graph(v);
        };
        for (int u = 1; u <= n; ++u) if (color[u] == -1)
        color[u] = 0, color_graph(u);
        while (bfs()) for (int u = 1; u <= n; ++u) if
        (!pairU[u] && color[u] == 0) dfs(u);
        vector<pair<int, int>> res;
        for (int u = 1; u <= n; ++u) if (pairU[u])
        res.emplace_back(u, pairU[u]);
        return res;
    }
};
```

4.7 Min Cost Flow

```
/*
Usage:
 - MCMF(n, s, t): initialize a graph of size n, with s and t
 being the source and sink nodes, respectively
 - add_edge(u, v, c, w): add an edge from u to v, with
 capacity c and cost w.
 - minCostFlow(): compute {flow, min_cost}
*/

struct MCMF {
    struct Edge { int v, cap, cost, rev; };
    int N, s, t;
    vector<vector<Edge>> adj;
    vector<int> dist, parent, parentEdge, inQueue;

    MCMF(int n, int s, int t) : N(n), s(s), t(t), adj(n),
    dist(n), parent(n), parentEdge(n), inQueue(n) {}

    void addEdge(int u, int v, int cap, int cost) {
        adj[u].push_back({v, cap, cost, (int)adj[v].size()});
        adj[v].push_back({u, 0, -cost, (int)adj[u].size() -
        1});
    }

    bool spfa(int s, int t) {
        fill(dist.begin(), dist.end(), INT_MAX);
        queue<int> q;
        dist[s] = 0; q.push(s); inQueue[s] = 1;
        while (!q.empty()) {
            int u = q.front(); q.pop(); inQueue[u] = 0;
            for (int i = 0; i < adj[u].size(); ++i) {
                Edge &e = adj[u][i];
                if (e.cap > 0 && dist[e.v] > dist[u] + e.cost)
                {
                    dist[e.v] = dist[u] + e.cost;
                    parent[e.v] = u; parentEdge[e.v] = i;
                    if (!inQueue[e.v]) q.push(e.v),
                    inQueue[e.v] = 1;
                }
            }
        }
        return dist[t] != INT_MAX;
    }

    pair<int, int> minCostFlow(int k) {
        int flow = 0, cost = 0;
        while (flow < k && spfa(s, t)) {
            int augFlow = k - flow;
            for (int u = t; u != s; u = parent[u])
                augFlow = min(augFlow,
                adj[parent[u]][parentEdge[u]].cap);

            for (int u = t; u != s; u = parent[u]) {
                Edge &e = adj[parent[u]][parentEdge[u]];
                e.cap -= augFlow;
                adj[u][e.rev].cap += augFlow;
                cost += augFlow * e.cost;
            }

            flow += augFlow;
        }
        return flow == k ? make_pair(flow, cost) :
        make_pair(-1, -1);
    }
};
```

4.8 Tarjan

```
/*
Usage:
- dfs(u, p): dfs, what the fuck
- topo[u]: the strongly connected component that u belongs to,
sorted topologically.
*/


int n, m;
vector<int> graph[N];
int num[N], low[N], topo[N];
deque<int> st;
int dfs_cnt = 0, topo_cnt = 0;

void dfs(int u, int p){
    num[u] = low[u] = ++dfs_cnt;
    st.push_back(u);
    int cnt = 0;
    for(int v: graph[u]){
        if (v == p && cnt == 0){cnt++; continue;}
        if (!num[v]){
            dfs(v, u);
            minimize(low[u], low[v]);
        }
        else minimize(low[u], num[v]);
    }

    if (num[u] == low[u]){
        topo_cnt++;
        while(st.back() != u){
            topo[st.back()] = topo_cnt;
            st.pop_back();
        }
        topo[u] = topo_cnt; st.pop_back();
    }
}
```

4.9 Two Sat

```
/*
Usage:
 - TwoSAT(n): initialize a graph of size n
 - addDisjunction(u, v): u and v = true
 - solve(): check if it's correct
*/

struct TwoSAT {
    int n, time = 0, sccCount = 0;
    vector<vector<int>> adj;
    vector<int> low, ids, scc;
    vector<bool> onStack;
    vector<int> assignment, st;

    TwoSAT(int n) : n(n) {
        adj.resize(2 * n);
        low.assign(2 * n, -1);
        ids.assign(2 * n, -1);
        scc.assign(2 * n, -1);
        onStack.assign(2 * n, false);
        assignment.resize(n);
```

```
        st.reserve(2 * n);
    }

    inline void addDisjunction(int u, int v) {
        adj[u ^ 1].push_back(v);
        adj[v ^ 1].push_back(u);
    }

    void tarjanDFS(int v) {
        ids[v] = low[v] = time++;
        st.push_back(v);
        onStack[v] = true;

        for (int u : adj[v]) {
            if (ids[u] == -1)
                tarjanDFS(u);
            if (onStack[u])
                low[v] = min(low[v], low[u]);
        }

        if (ids[v] == low[v]) {
            while (true) {
                int u = st.back();
                st.pop_back();
                onStack[u] = false;
                scc[u] = sccCount;
                if (u == v) break;
            }
            ++sccCount;
        }
    }

    bool solve() {
        for (int i = 0; i < 2 * n; ++i)
            if (ids[i] == -1)
                tarjanDFS(i);
         for (int i = 0; i < n; ++i) {
            if (scc[2 * i] == scc[2 * i + 1])
                return false;
            assignment[i] = (scc[2 * i] < scc[2 * i + 1]);
        }
        return true;
    }
};
```

5 String

5.1 Aho Corasick

```
/*
 * Usage :
 * addPattern : Adds patterns to the trie structure.
 * build : Sets up the fail links via BFS.
 * search : Performs the search and returns the indices of
 patterns found in the text.
 * Time complexity : O(n + S(Ai))
*/

class AhoCorasick {
public:
    AhoCorasick() { edges.push_back({}); fail.push_back(-1);
    output.push_back({}); }

    void addPattern(const string& pattern, int idx) {
        int node = 0;
        for (char c : pattern) {
            if (!edges[node].count(c)) {
                edges[node][c] = edges.size();
                edges.push_back({}); fail.push_back(-1);
                output.push_back({});
            }
            node = edges[node][c];
        }
        output[node].insert(idx);
    }

    void build() {
        queue<int> q;
        for (auto& p : edges[0]) {
            fail[p.second] = 0;
            q.push(p.second);
        }
        while (!q.empty()) {
            int state = q.front(); q.pop();
            for (auto& p : edges[state]) {
                char c = p.first;
                int next = p.second, f = fail[state];
                while (!edges[f].count(c)) f = fail[f];
                fail[next] = edges[f][c];

                output[next].insert(output[fail[next]].begin(),
                output[fail[next]].end());
                q.push(next);
            }
        }
    }

    vector<int> search(const string& text) {
        vector<int> result(text.size(), -1);
        int node = 0;
        for (int i = 0; i < text.size(); ++i) {
            while (!edges[node].count(text[i])) node =
            fail[node];
            node = edges[node][text[i]];
            for (int idx : output[node]) result[i] = idx;
        }
        return result;
    }

private:
    vector<unordered_map<char, int>> edges;
    vector<int> fail;
    vector<set<int>> output;
};
```

5.2 KMP

```
/*
    KMP template
*/

struct KMP{
    struct Node{
        array<int, 2> nxt;
        int pi;
        Node(){nxt[0] = nxt[1] = 0;pi = -1;}
    };

    int n;
    vector<Node> a;

    KMP(string s){
        n = s.size();
        a.resize(n+1);
        s = "#" + s;
        for(int i = 0; i<n; ++i){
            if (i < n){
                a[i + 1].pi = a[i].nxt[s[i+1] - '0'];
                a[i].nxt[s[i+1] - '0'] = i + 1;
                a[i+1].nxt = a[a[i + 1].pi].nxt;
            }
        }
    }

    int count_matches(string s){
        if (s.size() < n) return 0;
        int x = 0;
        int ans = 0;
        for(char c: s){
            x = a[x].nxt[c - '0'];
            if (x == n) ans++;
        }
        return ans;
    }
};
```

5.3 Palindrome Tree

```
/* THIS SHIT RUN IN O(N) TOO */
```

```
struct PalindromeTree{
    const static int K = 26;
    struct Node{
        long child[K], layer, pi;
        Node(long _layer = 0){
            memset(child, -1, sizeof child);
            layer = _layer;
            pi = -1;
        }
    };

    vector<Node> a;

    PalindromeTree(){
        a.push_back(Node(-1)); a.push_back(Node(0));
        a[1].pi = a[0].pi = 0;
    }

    void add_child(long &x, long layer){
        x = a.size();
        a.push_back(Node(layer));
    }

    void build(string s){
        long id = 0;
        s = "#" + s;
        for(long i = 1; i<s.size(); ++i){
            long digit = s[i] - '0';
            long j = id;
            while(j >= 0){
                if (s[i] == s[i - a[j].layer - 1]) break;
                j = a[j].pi;
            }

            if (a[j].child[digit] != -1) {
                id = a[j].child[digit];
                continue;
            }

            add_child(a[j].child[digit], a[j].layer + 2);
            long v = a[j].child[digit];

            if (a[v].layer == 1) j = 1;
            else if (a[v].layer == 2) j = a[0].child[digit];
            else{
                j = a[j].pi;
                while(j >= 0){
                    if (j == 0 || (s[i] == s[i - a[j].layer -
                    1])) break;
                    j = a[j].pi;
                }
                j = a[j].child[digit];
            }
            a[v].pi = j;
            id = v;
        }
        cout << a.size() - 2 << "\n";
    }
};
```

5.4 Suffix Array

```
/* I HAVE NO THING TO SAY. IT REALLY O(nlog2). */


const int LOG_N = 17, N = 1e5 + 69; // modify this

int spare_table[LOG_N][N];
vector<int> gen_sa(string s){
    int n = s.size();
    vector<int> perm(n);
    for(int i = 0; i<n; ++i) perm[i] = i;

    vector<int> bruh(n);
    for(int i = 0; i<n; ++i) bruh[i] = spare_table[0][i] =
    s[i] - 'a' + 1;

    for(int j = 1; j < LOG_N; ++j){
        vector<ll> cost(n);
        for(int i = 0; i<n; ++i){
            int _i = (i + MASK(j-1)) % n;
            cost[i] = 1LL * bruh[i] * N + bruh[_i];
        }
        if (j == 1)sort(ALL(perm), [&cost](int x, int
        y){return cost[x] < cost[y];});
        else{
            int last = 0;
            for(int i = 0; i<perm.size(); ++i){
                if (i + 1 == perm.size() || bruh[perm[i]] !=
                bruh[perm[i+1]]){
                    sort(perm.begin() + last, perm.begin() + i
                    + 1, [&cost](int x, int y){return cost[x]
                    < cost[y];});
                    last = i + 1;
                }
            }
        }

        int cnt = 0;
        for(int i = 0; i<n; ++i){
            if (i >= 1){
                int x = perm[i-1], y = perm[i];
                if (cost[x] != cost[y]) cnt++;
            }
            bruh[perm[i]] = spare_table[j][perm[i]] = cnt;
        }
    }
    return perm;
}
int lcp(int u,int v){
    int ans = 0;
    for(int j = LOG_N - 1; j>=0; --j) if (spare_table[j][u] ==
    spare_table[j][v]){
        ans += MASK(j);
        u += MASK(j); v += MASK(j);
        if (u > n) u -= n+1;
        if (v > n) v -= n+1;
    }
    return ans;
}
```

5.5 Suffix Automaton

```
/* IT IS THE SHORTEST VER I CAN FIND */
// Time complexity : O(N)

struct SuffixAutomaton {
    vector<vector<int>> next;
    vector<int> len, link;
    int last, size;

    SuffixAutomaton(const string &s) {
        size = 1; last = 0;
        len.push_back(0); link.push_back(-1);
        next.push_back(vector<int>(26, -1));

        for (char c : s) {
            int cur = size++;
            len.push_back(len[last] + 1);
            next.push_back(vector<int>(26, -1));
            link.push_back(-1);

            int p = last;
            while (p != -1 && next[p][c - 'a'] == -1) {
                next[p][c - 'a'] = cur;
                p = link[p];
            }

            if (p == -1) link[cur] = 0;
            else {
                int q = next[p][c - 'a'];
                if (len[p] + 1 == len[q]) link[cur] = q;
                else {
                    int clone = size++;
                    len.push_back(len[p] + 1);
```

```
                    next.push_back(next[q]);
                    link.push_back(link[q]);
                    while (p != -1 && next[p][c - 'a'] == q) {
                        next[p][c - 'a'] = clone;
                        p = link[p];
                    }
                    link[q] = link[cur] = clone;
                }
            }
            last = cur;
        }
    }
};
```

5.6 Z Function

```
/* YEP! IT IS JUST A Z FUNCTION */
//    Time complexity : O(N)

vector<int> ZFunction(const string &S) {
    int n = S.length();
    vector<int> Z(n);
    for (int L = 0, R = 0, i = 1; i < n; ++i) {
        if (i <= R) Z[i] = min(R - i + 1, Z[i - L]);
        while (i + Z[i] < n && S[Z[i]] == S[i + Z[i]]) Z[i]++;
        if (i + Z[i] - 1 > R) L = i, R = i + Z[i] - 1;
    }
    Z[0] = n;
    return Z;
}
```

6 Tree

6.1 Tree Line

```
/*
    Usage: that's the dsu, push the edges in, and get the
    chains. Now you have the treeline
*/


struct DSU{
    int n;
    vector<int> parent, sz;
    vector<vector<pair<int, int>>> chain;

    DSU(int _n){
        n = _n;
        parent.resize(n+1); sz.resize(n+1, 1);
        chain.resize(n+1);
        for(int i = 0; i<=n; ++i) {
            parent[i] = i;
            chain[i] = {{i, 0}};
        }
    }

    int find_set(int u){return (u == parent[u]) ? u :
    (parent[u] = find_set(parent[u]));}
    bool same_set(int u, int v){return find_set(u) ==
    find_set(v);}

    bool join_set(int u, int v, int w){
        u = find_set(u), v = find_set(v);
        if (u != v){
            if (sz[u] < sz[v]) swap(u, v);
            parent[v] = u;
            sz[u] += sz[v];
            chain[v][0].second = w;
            for(pair<int, int> i: chain[v])
            chain[u].push_back(i);
            return true;
        }
        return false;
    }

    int get_size(int u){return sz[find_set(u)];}
};
```

7 Arithmetic

7.1 Extended GCD

```
int extendedGCD(int a, int b, int &x, int &y){
    x = 1, y= 0;
    int _x = 0, _y = 1;
    while(b > 0){
        int q = a / b;
        tie(x, _x) = make_tuple(_x, x - q * _x);
        tie(y, _y) = make_tuple(_y, y - q * _y);
        tie(a, b) = make_tuple(b, a - q * b);
    }
    return a;
}
```

7.2 Chinese Remainder Theorem

```
tuple<int64_t, int64_t, int64_t> extended_gcd(int64_t a,
int64_t b) {
    if (b == 0) {
        return {a, 1, 0};
    }
    int64_t gcd, x1, y1;
    tie(gcd, x1, y1) = extended_gcd(b, a % b);
    return {gcd, y1, x1 - (a / b) * y1};
}

pair<int64_t, int64_t> chinese_remainder_theorem(int64_t a1,
int64_t m1, int64_t a2, int64_t m2) {
    int64_t gcd, x, y;
    std::tie(gcd, x, y) = extended_gcd(m1, m2);

    if ((a2 - a1) % gcd != 0) {
        return {0, -1};
    }

    int64_t lcm = (m1 / gcd) * m2;
    int64_t combined_solution = a1 + ((a2 - a1) / gcd * x %
    (m2 / gcd)) * m1;
    combined_solution = (combined_solution % lcm + lcm) % lcm;

    return {combined_solution, lcm};
}
```

7.3 Big Num Short

```
/*------------------------------ Big num template start
-------------------------------*/

string add(string a, string b){
    if (a.size() > b.size()) b = string(a.size() - b.size(),
    '0') + b;
    if (a.size() < b.size()) a = string(b.size() - a.size(),
    '0') + a;
    long carry = 0;
    for(long i = a.size() - 1; i>=0; --i){
        a[i] += b[i] - '0' + carry;
        if (a[i] > '9'){
            a[i] -= 10;
            carry = 1;
        }
        else carry = 0;
    }
    if (carry) a = "1" + a;
    return a;
}

string sub(string a, string b){
    if (a.size() > b.size()) b = string(a.size() - b.size(),
    '0') + b;
    if (a.size() < b.size()) a = string(b.size() - a.size(),
    '0') + a;
    long carry = 0;
    for(long i = a.size() - 1; i>=0; --i){
        a[i] -= b[i] - '0' + carry;
        if (a[i] < '0'){
            a[i] += 10;
            carry = 1;
```

```
        }
        else carry = 0;
    }
    for(long i = 0; i<a.size(); ++i) if (a[i] != '0') return
    a.substr(i);
    return "0";
}

string mul(string a, ll x){
    if (x == 0) return "0";
    string ans = "";
    ll carry = 0;
    reverse(ALL(a));
    for(auto c: a){
        long digit = c - '0';
        carry += x * digit;
        ans.push_back(carry % 10 + '0');
        carry /= 10;
    }
    while(carry > 0){
        ans.push_back(carry % 10 + '0');
        carry /= 10;
    }
    reverse(ALL(ans));
    return ans;
}

const ll BLOCK = 15;
string mul(string a, string b){
    if (a == "0" || b == "0") return "0";
    string ans = "0";
    for(long i = 0; i<(b.size() + BLOCK - 1) / BLOCK; ++i){
        long e = b.size() - i * BLOCK;
        long s = max(0, e - BLOCK);
        ll cur = 0;
        for(long j = s; j<e; ++j)
            cur = cur * 10 + (b[j] - '0');
        ans = add(ans, mul(a, cur) + string(i * BLOCK, '0'));
    }
    return ans;
}

string fast_pow(string a, ll n){
    if (n == 0) return "1";
    string tmp = fast_pow(a, n/2);
    tmp = mul(tmp, tmp);
    if (n % 2) return mul(tmp, a);
    return tmp;
}

string di(string a, ll x){
    string ans = "";
    ll carry = 0;
    for(char c: a){
        long digit = c - '0';
        carry = carry * 10 + digit;
        ans.push_back('0' + carry / x);
        carry %= x;
    }
    for(long i = 0; i<ans.size(); ++i) if (ans[i] != '0')
    return ans.substr(i);
    return "0";
}

/*------------------------------ Big num template end
------------------------------*/
```

7.4 Big Num Long

```
/* BIGNUM AGAIN!! BUT TESTED BY I_LOVE_HOANG_YEN */

// BigInt {{{
const int BASE_DIGITS = 9;
const int BASE = 1000000000;

struct BigInt {
    int sign;
    vector<int> a;

    // -------------------- Constructors --------------------
    // Default constructor.
    BigInt() : sign(1) {}

    // Constructor from long long.
    BigInt(long long v) {
        *this = v;
    }
    BigInt& operator = (long long v) {
        sign = 1;
        if (v < 0) {
            sign = -1;
            v = -v;
        }
        a.clear();
        for (; v > 0; v = v / BASE)
            a.push_back(v % BASE);
        return *this;
    }

    // Initialize from string.
    BigInt(const string& s) {
        read(s);
    }

    // -------------------- Input / Output
    --------------------
    void read(const string& s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' ||
        s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= BASE_DIGITS)
        {
            int x = 0;
            for (int j = max(pos, i - BASE_DIGITS + 1); j <=
            i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend istream& operator>>(istream &stream, BigInt &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream, const BigInt
    &v) {
        if (v.sign == -1 && !v.isZero())
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << setw(BASE_DIGITS) << setfill('0') <<
            v.a[i];
        return stream;
    }

    // -------------------- Comparison --------------------
    bool operator<(const BigInt &v) const {
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() * v.sign;
        for (int i = ((int) a.size()) - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] * sign;
        return false;
    }

    bool operator>(const BigInt &v) const {
        return v < *this;
    }
```

```
bool operator<=(const BigInt &v) const {
    return !(v < *this);
}
bool operator>=(const BigInt &v) const {
    return !(*this < v);
}
bool operator==(const BigInt &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const BigInt &v) const {
    return *this < v || v < *this;
}

// Returns:
// 0 if |x| == |y|
// -1 if |x| < |y|
// 1 if |x| > |y|
friend int __compare_abs(const BigInt& x, const BigInt& y)
{
    if (x.a.size() != y.a.size()) {
        return x.a.size() < y.a.size() ? -1 : 1;
    }

    for (int i = ((int) x.a.size()) - 1; i >= 0; --i) {
        if (x.a[i] != y.a[i]) {
            return x.a[i] < y.a[i] ? -1 : 1;
        }
    }
    return 0;
}

// -------------------- Unary operator - and operators +-
--------------------
BigInt operator-() const {
    BigInt res = *this;
    if (isZero()) return res;

    res.sign = -sign;
    return res;
}

// Note: sign ignored.
void __internal_add(const BigInt& v) {
    if (a.size() < v.a.size()) {
        a.resize(v.a.size(), 0);
    }
    for (int i = 0, carry = 0; i < (int) max(a.size(),
    v.a.size()) || carry; ++i) {
        if (i == (int) a.size()) a.push_back(0);

        a[i] += carry + (i < (int) v.a.size() ? v.a[i] :
        0);
        carry = a[i] >= BASE;
        if (carry) a[i] -= BASE;
    }
}

// Note: sign ignored.
void __internal_sub(const BigInt& v) {
    for (int i = 0, carry = 0; i < (int) v.a.size() ||
    carry; ++i) {
        a[i] -= carry + (i < (int) v.a.size() ? v.a[i] :
        0);
        carry = a[i] < 0;
        if (carry) a[i] += BASE;
    }
    this->trim();
}

BigInt operator += (const BigInt& v) {
    if (sign == v.sign) {
        __internal_add(v);
    } else {
        if (__compare_abs(*this, v) >= 0) {
            __internal_sub(v);
        } else {
            BigInt vv = v;
            swap(*this, vv);
            __internal_sub(vv);
        }
    }
    return *this;
}

BigInt operator -= (const BigInt& v) {
    if (sign == v.sign) {
        if (__compare_abs(*this, v) >= 0) {
            __internal_sub(v);
        } else {
            BigInt vv = v;
            swap(*this, vv);
            __internal_sub(vv);
            this->sign = -this->sign;
        }
    } else {
        __internal_add(v);
    }
    return *this;
}

template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value &&
        std::is_lvalue_reference<R&&>::value,
        BigInt>::type friend operator + (L&& l, R&& r) {
    BigInt result(std::forward<L>(l));
    result += r;
    return result;
}
template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value &&
        std::is_rvalue_reference<R&&>::value,
        BigInt>::type friend operator + (L&& l, R&& r) {
    BigInt result(std::move(r));
    result += l;
    return result;
}

template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value,
        BigInt>::type friend operator - (L&& l, R&& r) {
    BigInt result(std::forward<L>(l));
    result -= r;
    return result;
}

// -------------------- Operators * / %
--------------------
friend pair<BigInt, BigInt> divmod(const BigInt& a1, const
BigInt& b1) {
    assert(b1 > 0);  // divmod not well-defined for b < 0.

    long long norm = BASE / (b1.a.back() + 1);
    BigInt a = a1.abs() * norm;
    BigInt b = b1.abs() * norm;
    BigInt q = 0, r = 0;
    q.a.resize(a.a.size());

    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= BASE;
        r += a.a[i];
        long long s1 = r.a.size() <= b.a.size() ? 0 :
        r.a[b.a.size()];
        long long s2 = r.a.size() <= b.a.size() - 1 ? 0 :
        r.a[b.a.size() - 1];
        long long d = ((long long) BASE * s1 + s2) /
        b.a.back();
        r -= b * d;
        while (r < 0) {
            r += b, --d;
        }
        q.a[i] = d;
    }

    q.sign = a1.sign * b1.sign;
```

```
    r.sign = a1.sign;
    q.trim();
    r.trim();
    auto res = make_pair(q, r / norm);
    if (res.second < 0) res.second += b1;
    return res;
}
BigInt operator/(const BigInt &v) const {
    if (v < 0) return divmod(-*this, -v).first;
    return divmod(*this, v).first;
}

BigInt operator%(const BigInt &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    assert(v > 0);  // operator / not well-defined for v
    <= 0.
    if (llabs(v) >= BASE) {
        *this /= BigInt(v);
        return ;
    }
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i)
    {
        long long cur = a[i] + rem * (long long) BASE;
        a[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

BigInt operator/(int v) const {
    assert(v > 0);  // operator / not well-defined for v
    <= 0.

    if (llabs(v) >= BASE) {
        return *this / BigInt(v);
    }
    BigInt res = *this;
    res /= v;
    return res;
}
void operator/=(const BigInt &v) {
    *this = *this / v;
}

long long operator%(long long v) const {
    assert(v > 0);  // operator / not well-defined for v
    <= 0.
    assert(v < BASE);
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long) BASE) % v;
    return m * sign;
}

void operator*=(int v) {
    if (llabs(v) >= BASE) {
        *this *= BigInt(v);
        return ;
    }
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() ||
    carry; ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / BASE);
        a[i] = (int) (cur % BASE);
    }
    trim();
}

BigInt operator*(int v) const {
    if (llabs(v) >= BASE) {
        return *this * BigInt(v);
    }
    BigInt res = *this;
    res *= v;
    return res;
}

// Convert BASE 10^old --> 10^new.
static vector<int> convert_base(const vector<int> &a, int
old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back((long long)(cur %
            p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
        res.pop_back();
    return res;
}

void fft(vector<complex<double> > &x, bool invert) const {
    int n = (int) x.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(x[i], x[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * 3.14159265358979323846 / len *
        (invert ? -1 : 1);
        complex<double> wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            complex<double> w(1);
            for (int j = 0; j < len / 2; ++j) {
                complex<double> u = x[i + j];
                complex<double> v = x[i + j + len / 2] *
                w;
                x[i + j] = u + v;
                x[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (int i = 0; i < n; ++i)
            x[i] /= n;
}

void multiply_fft(const vector<int> &x, const vector<int>
&y, vector<int> &res) const {
    vector<complex<double> > fa(x.begin(), x.end());
    vector<complex<double> > fb(y.begin(), y.end());
    int n = 1;
    while (n < (int) max(x.size(), y.size()))
        n <<= 1;
    n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; ++i)
        fa[i] *= fb[i];
```

```
    fft(fa, true);

    res.resize(n);
    long long carry = 0;
    for (int i = 0; i < n; ++i) {
        long long t = (long long) (fa[i].real() + 0.5) +
        carry;
        carry = t / 1000;
        res[i] = t % 1000;
    }
}

BigInt mul_simple(const BigInt &v) const {
    BigInt res;
    res.sign = sign * v.sign;
    res.a.resize(a.size() + v.a.size());
    for (int i = 0; i < (int) a.size(); ++i)
        if (a[i])
            for (int j = 0, carry = 0; j < (int)
            v.a.size() || carry; ++j) {
                long long cur = res.a[i + j] + (long long)
                a[i] * (j < (int) v.a.size() ? v.a[j] : 0)
                + carry;
                carry = (int) (cur / BASE);
                res.a[i + j] = (int) (cur % BASE);
            }
    res.trim();
    return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

BigInt mul_karatsuba(const BigInt &v) const {
    vector<int> x6 = convert_base(this->a, BASE_DIGITS,
    6);
    vector<int> y6 = convert_base(v.a, BASE_DIGITS, 6);
    vll x(x6.begin(), x6.end());
    vll y(y6.begin(), y6.end());
    while (x.size() < y.size())
        x.push_back(0);
    while (y.size() < x.size())
        y.push_back(0);
    while (x.size() & (x.size() - 1))
        x.push_back(0), y.push_back(0);
    vll c = karatsubaMultiply(x, y);
    BigInt res;
    res.sign = sign * v.sign;
    long long carry = 0;
    for (int i = 0; i < (int) c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int) (cur % 1000000));
        carry = cur / 1000000;
    }
    res.a = convert_base(res.a, 6, BASE_DIGITS);
    res.trim();
    return res;
}

void operator*=(const BigInt &v) {
    *this = *this * v;
}
BigInt operator*(const BigInt &v) const {
    if (a.size() * v.a.size() <= 1000111) return
    mul_simple(v);
    if (a.size() > 500111 || v.a.size() > 500111) return
    mul_fft(v);
    return mul_karatsuba(v);
}

BigInt mul_fft(const BigInt& v) const {
    BigInt res;
    res.sign = sign * v.sign;
    multiply_fft(convert_base(a, BASE_DIGITS, 3),
    convert_base(v.a, BASE_DIGITS, 3), res.a);
    res.a = convert_base(res.a, 3, BASE_DIGITS);
    res.trim();
    return res;
}

// -------------------- Misc --------------------
BigInt abs() const {
    BigInt res = *this;
    res.sign *= res.sign;
    return res;
}
void trim() {
    while (!a.empty() && !a.back())
        a.pop_back();
    if (a.empty())
        sign = 1;
}

bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
}

friend BigInt gcd(const BigInt &x, const BigInt &y) {
    return y.isZero() ? x : gcd(y, x % y);
}
friend BigInt lcm(const BigInt &x, const BigInt &y) {
    return x / gcd(x, y) * y;
}

friend BigInt sqrt(const BigInt &a1) {
    BigInt a = a1;
    while (a.a.empty() || a.a.size() % 2 == 1)
        a.a.push_back(0);

    int n = a.a.size();

    int firstDigit = (int) sqrt((double) a.a[n - 1] * BASE
    + a.a[n - 2]);
    int norm = BASE / (firstDigit + 1);
    a *= norm;
    a *= norm;
    while (a.a.empty() || a.a.size() % 2 == 1)
        a.a.push_back(0);

    BigInt r = (long long) a.a[n - 1] * BASE + a.a[n - 2];
    firstDigit = (int) sqrt((double) a.a[n - 1] * BASE +
    a.a[n - 2]);
```

```
        int q = firstDigit;
        BigInt res;

        for(int j = n / 2 - 1; j >= 0; j--) {
            for(; ; --q) {
                BigInt r1 = (r - (res * 2 * BigInt(BASE) + q)
                * q) * BigInt(BASE) * BigInt(BASE) + (j > 0 ?
                (long long) a.a[2 * j - 1] * BASE + a.a[2 * j
                - 2] : 0);
                if (r1 >= 0) {
                    r = r1;
                    break;
                }
            }
            res *= BASE;
            res += q;

            if (j > 0) {
                int d1 = res.a.size() + 2 < r.a.size() ?
                r.a[res.a.size() + 2] : 0;
                int d2 = res.a.size() + 1 < r.a.size() ?
                r.a[res.a.size() + 1] : 0;
                int d3 = res.a.size() < r.a.size() ?
                r.a[res.a.size()] : 0;
                q = ((long long) d1 * BASE * BASE + (long
                long) d2 * BASE + d3) / (firstDigit * 2);
            }
        }

        res.trim();
        return res / norm;
    }
};
// }}}
```

7.5 Gaussian Elimination (Float)

```
/* GAUSS ELIMINATION YAHOOO! */

void gauss(vector<vector<double>>& a, vector<double>& b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (a[j][i]) {
                double ratio = a[j][i] / a[i][i];
                for (int k = i; k < n; ++k) a[j][k] -= ratio *
                a[i][k];
                b[j] -= ratio * b[i];
            }
        }
    }
    for (int i = n - 1; i >= 0; --i) {
        for (int j = i + 1; j < n; ++j) b[i] -= a[i][j] *
        b[j];
        b[i] /= a[i][i];
    }
}
```

7.6 Gaussian Elimination (Integer)

```
const int MOD = 1e9 + 9;

ll fast_pow(ll a, ll n){
    ll ans = 1;
    while(n > 0){
        if (n & 1) ans = ans * a % MOD;
        a = a * a % MOD;
        n /= 2;
    }
    return ans;
}

ll inverse(ll a){return fast_pow(a, MOD - 2);}

ll normie(ll x){
    x %= MOD;
    if (x < 0) x += MOD;
    return x;
}

vector<ll> solve(vector<vector<ll>> sigma){
    int n = sigma.size(),m = sigma[0].size() - 1;
    vector<bool> banned(n);
    for(int j = 0; j<m; ++j){
        int idx = -1;
        for(int i = 0; i<n; ++i) if (!banned[i] &&
        sigma[i][j]) {
            idx = i;
            ll head = sigma[i][j];
            ll inv = inverse(head);
            for(int k = 0; k <= m; ++k) sigma[i][k] =
            (sigma[i][k] * inv) % MOD;
        }
        if (idx == -1) continue;
        banned[idx] = true;
        for(int i = 0; i<n; ++i) if (sigma[i][j] && i != idx){
            for(int k = 0; k <= m; ++k) sigma[i][k] =
            normie(sigma[i][k] - sigma[idx][k]);
        }
    }

    vector<int> deg(n);
    for(int i = 0; i<n; ++i){
        deg[i] = m;
        for(int j = 0; j < m; ++j) if (sigma[i][j]){
            deg[i] = j;
            break;
        }
    }
    for(int i = 0; i<n; ++i){
        if (deg[i] == m) continue;
        while(true){
            int v = m;
            for(int j = deg[i] + 1; j < m; ++j) if
            (sigma[i][j]){
                v = j;
                break;
            }
            if (v == m) break;
            for(int x = 0; x < n; ++x) if (deg[x] == v){
                ll shit = inverse(sigma[x][v]) * sigma[i][v] %
                MOD;
                for(int j = 0; j <= m; ++j) {
                    sigma[i][j] = normie(sigma[i][j] - shit *
                    sigma[x][j]);
                }
                break;
            }
        }
    }
    vector<ll> ans(m);
    for(int i = 0; i<n; ++i) {
        ans[deg[i]] = sigma[i].back();
    }
    return ans;
}
```

7.7 Miller Rabin

```
namespace MillerRabin{
    using u64 = uint64_t;
    using u128 = __uint128_t;

    u64 binpower(u64 base, u64 e, u64 mod) {
        u64 result = 1;
        base %= mod;
        while (e) {
            if (e & 1)
                result = (u128)result * base % mod;
            base = (u128)base * base % mod;
            e >>= 1;
        }
        return result;
    }

    bool check_composite(u64 n, u64 a, u64 d, int s) {
        u64 x = binpower(a, d, n);
```

```
        if (x == 1 || x == n - 1)
            return false;
        for (int r = 1; r < s; r++) {
            x = (u128)x * x % n;
            if (x == n - 1)
                return false;
        }
        return true;
    };

    bool MillerRabin(u64 n) { // returns true if n is prime,
    else returns false.
        if (n < 2)
            return false;

        int r = 0;
        u64 d = n - 1;
        while ((d & 1) == 0) {
            d >>= 1;
            r++;
        }

        for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37}) {
            if (n == a)
                return true;
            if (check_composite(n, a, d, r))
                return false;
        }
        return true;
    }
}
```

7.8 Pollard Rho

```
/* THIS IS POLLARD RHO AND I HAVE NO IDEA WHAT ITS USAGE IS */
\

unsigned long long mod_mul(unsigned long long a, unsigned long
long b, unsigned long long mod) {
    unsigned long long res = 0;
    a %= mod;
    while (b) {
        if (b & 1) res = (res + a) % mod;
        a = (2 * a) % mod;
        b >>= 1;
    }
    return res;
}

unsigned long long mod_exp(unsigned long long base, unsigned
long long exp, unsigned long long mod) {
    unsigned long long res = 1;
    base %= mod;
    while (exp) {
        if (exp & 1) res = mod_mul(res, base, mod);
        base = mod_mul(base, base, mod);
        exp >>= 1;
    }
    return res;
}

unsigned long long gcd(unsigned long long a, unsigned long
long b) {
    return b ? gcd(b, a % b) : a;
}

bool miller_rabin(unsigned long long n, int iterations = 5) {
    if (n < 4) return n == 2 || n == 3;
    unsigned long long d = n - 1;
    while (d % 2 == 0) d /= 2;
    for (int i = 0; i < iterations; ++i) {
        unsigned long long a = 2 + rand() % (n - 4);
        unsigned long long x = mod_exp(a, d, n);
        if (x == 1 || x == n - 1) continue;
        bool is_composite = true;
        for (unsigned long long temp = d; temp != n - 1; temp
        *= 2) {
            x = mod_mul(x, x, n);
            if (x == n - 1) {
                is_composite = false;
                break;
            }
        }
        if (is_composite) return false;
    }
    return true;
}

unsigned long long pollards_rho(unsigned long long n) {
    if (n % 2 == 0) return 2;
    unsigned long long x = 2, y = 2, d = 1, c = rand() % (n -
    1) + 1;
    while (d == 1) {
        x = (mod_mul(x, x, n) + c) % n;
        y = (mod_mul(mod_mul(y, y, n) + c, y, n) + c) % n;
        d = gcd(x > y ? x - y : y - x, n);
    }
    return (d == n) ? pollards_rho(n) : d;
}

void factorize(unsigned long long n, std::vector<unsigned long
long> &factors) {
    if (n == 1) return;
    if (miller_rabin(n)) {
        factors.push_back(n);
        return;
    }
    unsigned long long factor = pollards_rho(n);
    factorize(factor, factors);
    factorize(n / factor, factors);
}

std::vector<unsigned long long> get_prime_factors(unsigned
long long n) {
    std::vector<unsigned long long> factors;
    factorize(n, factors);
    std::sort(factors.begin(), factors.end());
    return factors;
}
```

7.9 Lagrange Interpol

```
typedef long long ll;

// Function to compute (base^exp) % mod
ll mod_exp(ll base, ll exp, ll mod) {
    ll res = 1;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) res = res * base % mod;
        base = base * base % mod;
        exp >>= 1;
    }
    return res;
}

// Function to compute modular inverse using Fermat's Little
Theorem
ll mod_inv(ll a, ll mod) {
    return mod_exp(a, mod - 2, mod);
}

// Function to add two polynomials
std::vector<ll> add_polynomials(const std::vector<ll>& a,
const std::vector<ll>& b) {
    std::vector<ll> result(std::max(a.size(), b.size()));
    for (size_t i = 0; i < result.size(); ++i) {
        if (i < a.size()) result[i] = (result[i] + a[i]) %
        MOD;
        if (i < b.size()) result[i] = (result[i] + b[i]) %
        MOD;
    }
    return result;
}

// Function to multiply a polynomial by a monomial (x - c)
```

```
std::vector<ll> multiply_polynomial(const std::vector<ll>&
poly, ll c) {
    std::vector<ll> result(poly.size() + 1);
    for (size_t i = 0; i < poly.size(); ++i) {
        result[i + 1] = poly[i];
        result[i] = (result[i] - c * poly[i] % MOD + MOD) %
        MOD;
    }
    return result;
}

// Function to multiply a polynomial by a constant
std::vector<ll> multiply_by_constant(const std::vector<ll>&
poly, ll constant) {
    std::vector<ll> result(poly.size());
    for (size_t i = 0; i < poly.size(); ++i) {
        result[i] = poly[i] * constant % MOD;
    }
    return result;
}

// Function for Lagrange interpolation to find the polynomial
std::vector<ll> lagrange_polynomial(const std::vector<ll>&
x_vals, const std::vector<ll>& y_vals) {
    int n = x_vals.size();
    std::vector<ll> poly(n, 0);

    for (int i = 0; i < n; ++i) {
        ll denom = 1;
        std::vector<ll> term = {y_vals[i]};
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                denom = denom * (x_vals[i] - x_vals[j] + MOD)
                % MOD;
                term = multiply_polynomial(term, x_vals[j]);
            }
        }
        ll inv_denom = mod_inv(denom, MOD);
        term = multiply_by_constant(term, inv_denom);
        poly = add_polynomials(poly, term);
    }

    return poly;
}
```

8 Combinatorics

8.1 FFT (Float)

```
/*
Usage:
- conv: multiply A and B modulo 998244353
*/



namespace PolyMul {

    using namespace std;
    using cd = complex<double>;
    const double PI = acos(-1);

    // Perform FFT or inverse FFT
    void fft(vector<cd>& 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);
            cd wlen(cos(ang), sin(ang));
            for (int i = 0; i < n; i += len) {
                cd w(1);
                for (int j = 0; j < len / 2; ++j) {
                    cd 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 (cd& x : a) x /= n;
    }

    // Multiply two polynomials
    vector<double> multiply(const vector<double>& a, const
    vector<double>& b) {
        int n = 1;
        int total_size = a.size() + b.size() - 1;
        while (n < total_size) n <<= 1;  // Ensure n is a
        power of two

        vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
        b.end());
        fa.resize(n);
        fb.resize(n);

        fft(fa, false);
        fft(fb, false);
        for (int i = 0; i < n; ++i) fa[i] *= fb[i];
        fft(fa, true);

        vector<double> result(total_size);
        for (int i = 0; i < total_size; ++i) result[i] =
        fa[i].real();
        return result;
    }
}
```

8.2 FFT (Integer)

```
/*
Usage:
- conv: multiply A and B modulo 998244353
*/

const int MOD = 998244353;

const int N = 2e5 + 69;

namespace NTT{
    const int mod = MOD;
    ll modpow(ll b, ll e) {
        ll ans = 1;
        for (; e; b = b * b % mod, e /= 2)
            if (e & 1) ans = ans * b % mod;
        return ans;
    }
    const int root = 62; // = 998244353
    // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479
    << 21
    // and 483 << 21 (same root). The last two are > 10^9.
    typedef vector<ll> vl;

    #define rep(i, a, b) for(int i = a; i < (b); ++i)
    #define all(x) begin(x), end(x)
    #define sz(x) (int)(x).size()
    typedef long long ll;
    typedef pair<int, int> pii;
    typedef vector<int> vi;
    void ntt(vl &a) {
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vl rt(2, 1);
        for (static int k = 2, s = 2; k < n; k *= 2, s++) {
            rt.resize(n);
            ll z[] = {1, modpow(root, mod >> s)};
            rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
        }
        vi rev(n);
        rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
        rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
            for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
```

```
                ll z = rt[j + k] * a[i + j + k] % mod, &ai =
                a[i + j];
                a[i + j + k] = ai - z + (z > ai ? mod : 0);
                ai += (ai + z >= mod ? z - mod : z);
            }
    }
    vl conv(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
            n = 1 << B;
        int inv = modpow(n, mod - 2);
        vl L(a), R(b), out(n);
        L.resize(n), R.resize(n);
        ntt(L), ntt(R);
        rep(i,0,n)
            out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv %
            mod;
        ntt(out);
        return {out.begin(), out.begin() + s};
    }
    #undef vl
}
```

8.3 Modular

```
/*
Usage: - use as if it was integers
*/

const int MOD = 998244353;
struct Modular{
    ll x;
    Modular(ll _x = 0){
        x = _x % MOD;
        if (x < 0) x += MOD;
    }
    Modular& operator += (Modular y){
        x += y.x;
        if (x >= MOD) x -= MOD;
        return *this;
    }
    Modular operator + (Modular y) {
        Modular tmp = *this;
        return tmp += y;
    }

    Modular& operator -= (Modular y){
        x -= y.x;
        if (x < 0) x += MOD;
        return *this;
    }
    Modular operator - (Modular y) {
        Modular tmp = *this;
        return tmp -= y;
    }

    Modular& operator *= (Modular y){
        x *= y.x;
        if (x >= MOD) x %= MOD;
        return *this;
    }
    Modular operator * (Modular y) {
        Modular tmp = *this;
        return tmp *= y;
    }

    // use at your own risk
    bool operator == (Modular y){
        return x == y.x;
    }
    bool operator != (Modular y){
        return x != y.x;
    }
};
ostream& operator << (ostream& out, Modular x){
    out << x.x;
    return out;
}

Modular fast_pow(Modular a, ll n){
    Modular ans = 1;
    while(n > 0){
        if (n & 1) ans *= a;
        a *= a;
        n >>= 1;
    }
    return ans;
}
Modular inverse(Modular a){return fast_pow(a, MOD - 2);}
```

8.4 Matrix

```
/*
Usage:
- Good old matrix
*/


struct Matrix{
    int n, m;
    vector<vector<Modular>> a;

    Matrix(int n, int m): n(n), m(m){
        a.resize(n, vector<Modular>(m));
    }

    Matrix operator * (Matrix b){
        if (m != b.n) assert(1);
        Matrix res(n, b.m);
        for(int i = 0; i<n; ++i)
        for(int j = 0; j<b.m; ++j)
        for(int k = 0; k<m; ++k)
            res.a[i][j] += a[i][k] * b.a[k][j];
        return res;
    }
};

Matrix fast_pow(Matrix a, ll n){
    if (n == 1) return a;
    Matrix t = fast_pow(a, n / 2);
    if (n % 2 == 0) return t * t;
    return t * t * a;
}
```

8.5 Xor Convolution

```
/*
Xor convolution:
- Usage: xormul(a, b, &c);
*/


const int mod = 998244353;

int inverse(int x, int mod) {
    return x == 1 ? 1 : mod - mod / x * inverse(mod % x, mod)
    % mod;
}

void xormul(vector<int> a, vector<int> b, vector<int> &c) {
    int m = (int) a.size();
    c.resize(m);
    for (int n = m / 2; n > 0; n /= 2)
        for (int i = 0; i < m; i += 2 * n)
            for (int j = 0; j < n; j++) {
                int x = a[i + j], y = a[i + j + n];
                a[i + j] = (x + y) % mod;
                a[i + j + n] = (x - y + mod) % mod;
            }
    for (int n = m / 2; n > 0; n /= 2)
        for (int i = 0; i < m; i += 2 * n)
            for (int j = 0; j < n; j++) {
                int x = b[i + j], y = b[i + j + n];
                b[i + j] = (x + y) % mod;
                b[i + j + n] = (x - y + mod) % mod;
            }
    for (int i = 0; i < m; i++)
        c[i] = a[i] * b[i] % mod;
    for (int n = 1; n < m; n *= 2)
```

```
        for (int i = 0; i < m; i += 2 * n)
            for (int j = 0; j < n; j++) {
                int x = c[i + j], y = c[i + j + n];
                c[i + j] = (x + y) % mod;
                c[i + j + n] = (x - y + mod) % mod;
            }
    int mrev = inverse(m, mod);
    for (int i = 0; i < m; i++)
        c[i] = c[i] * mrev % mod;
}
```