

Team Notebook

ntTas

April 19, 2019

Contents

1 Data Structure	2	3.11 lengthUnionSegment	7	5.2.2 InverseModulo	13
1.1 BIT	2	3.12 lineIntersection	7	5.2.3 PrimeFactor	13
1.2 BITRange	2	3.13 twoSegmentIntersect	7	5.2.4 Sieve	14
1.3 Binary Trie	2	4 Graph	7	5.2.5 extendedEuclid	14
1.4 LazyPropagation	2	4.1 Flow	7	5.3 Polynomial	14
1.5 MapComparator	3	4.1.1 Bipartite Matching	7	5.3.1 FFT mod	14
1.6 SegmentTree	3	4.1.2 Dinic	8	5.3.2 FFT	14
1.7 Trie	3	4.1.3 Edmond-Karp Max flow	8	5.3.3 FFT2	15
2 Dynamic Programming	4	4.1.4 Min-cost flow	9	6 Misc	16
2.1 Convex Hull Trick	4	4.2 LCABinaryLifting	9	6.1 Mo	16
3 Geometry	4	4.3 LCARmq	10	6.2 mt19937	16
3.1 2d	4	4.4 LCATarjan	10	6.3 ternary	16
3.2 3d	4	4.5 SCC	10	7 Setup	16
3.3 AreaPolygon	5	4.5.1 2SAT	10	7.1 BqTemplate	16
3.4 OrientationTriangle	5	4.5.2 Kosaraju	10	7.2 C++Template	17
3.5 circleLineIntersect	5	4.6 Shortest Path	11	7.3 FastScanner	17
3.6 convexhull	5	4.6.1 BellmanFord	11	7.4 sublimsetup	18
3.7 convexhullGrahamScan	5	4.6.2 Dijkstra	11	8 String	18
3.8 crossProduct	6	5 Math	11	8.1 Hashing	18
3.9 dotProduct	6	5.1 BigInt	11	8.2 KMP	18
3.10 findIntersectionTwoSegment	6	5.2 Number Theory	12	8.3 Manacher	18
		5.2.1 CRT	12	8.4 rabinkarp	18

1 Data Structure

1.1 BIT

```
template<class T, int SZ> struct BIT {
    T bit[SZ+1];
    BIT() { memset(bit,0,sizeof bit); }

    void upd(int k, T val) { // add val to index k
        for (;k <= SZ; k += (k&-k)) bit[k] += val;
    }

    T query(int k) {
        T temp = 0;
        for (;k > 0;k -= (k&-k)) temp += bit[k];
        return temp;
    }

    T query(int l, int r) { return query(r)-query(l-1); } //
        range query [l,r]
};
```

1.2 BITRange

```
/**
 * Description: 1D range update, range sum query
 * Alternative to lazy segment tree
 * Source: GeeksForGeeks?
 * Verification: ?
 */

template<class T, int SZ> struct BITrange {

    BIT<T,SZ> bit[2]; // sums piecewise linear functions

    void upd(int hi, T val) {
        bit[1].upd(1,val), bit[1].upd(hi+1,-val);
        bit[0].upd(hi+1,hi*val);
    }

    void upd(int lo, int hi, T val) {
        upd(lo-1,-val);
        upd(hi,val);
    }

    T query(int x) {
        return bit[1].query(x)*x+bit[0].query(x);
    }
};
```

```
T query(int x, int y) {
    return query(y)-query(x-1);
}
};
```

1.3 Binary Trie

```
//Binary Trie
//without deletion, comment all val change and trie.check
//Don't forget to check maxval
struct node{
    node *kiri = NULL;
    node *kanan = NULL;
    int val = 0; //Can be deleted
};

struct btrie{
    node *head = new node;
    node *cur = head;
    void insert(int x){
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                if(head->kanan == NULL)head->kanan = new node;
                head = head->kanan;
            } else{
                if(head->kiri == NULL)head->kiri = new node;
                head = head->kiri;
            }
            head->val += 1; // Can be deleted
        }
        head = cur;
    }
    void del(int x){
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                head = head->kanan;
            } else{
                head = head->kiri;
            }
            head->val -= 1; //Can be deleted
        }
        head = cur;
    }
    int max(int x){
        int res = 0;
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                if(check(head->kiri)){
```

```
                    res += (1<<i);
                    head = head->kiri;
                } else if(check(head->kanan)){
                    head = head->kanan;
                } //Break can be placed here
            } else {
                if(check(head->kanan)){
                    head = head->kanan;
                } else if(check(head->kiri)){
                    head = head->kiri;
                    res += (1<<i);
                } //Break can be placed here
            }
        }
        head = cur;
        return (res^x);
    }
};

private:
    bool check(node *x){
        if(x != NULL and x->val > 0)return true; //Condition
            may be changed
        else return false;
    }
};
```

1.4 LazyPropagation

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <string>

using namespace std;

typedef long long LL;
const int MAXN = 132000;

struct data {
    LL sum;
    LL delayedAdd;
};

data sTree[2*MAXN];

LL GET(int id, int l, int r) {
    return sTree[id].sum + sTree[id].delayedAdd*(r-l+1);
}
```

```

void LAZY_UPDATE(int id, int delta) {
    sTree[id].delayedAdd += delta;
}

void PROPAGATE(int id) {
    LAZY_UPDATE(2*id+1, sTree[id].delayedAdd);
    LAZY_UPDATE(2*id+2, sTree[id].delayedAdd);

    sTree[id].delayedAdd = 0;
}

void MERGE(int id, int l, int m, int r) {
    sTree[id].sum = GET(2*id+1, l, m) + GET(2*id+2, m+1, r);
}

void update(int id, int l, int r, int xa, int xb, int delta)
{
    if ((xa <= l) && (r <= xb)) {
        LAZY_UPDATE(id, delta);
    } else {
        int m = (l + r)/2;

        PROPAGATE(id);

        if (xa <= m) update(2*id+1, l, m, xa, xb, delta);
        if (xb > m) update(2*id+2, m+1, r, xa, xb, delta);

        MERGE(id, l, m, r);
    }
}

LL query(int id, int l, int r, int xa, int xb) {
    if ((xa <= l) && (r <= xb)) {
        return GET(id, l, r);
    } else {
        int m = (l + r)/2;

        PROPAGATE(id);

        LL ret = 0;
        if (xa <= m) ret += query(2*id+1, l, m, xa, xb);
        if (xb > m) ret += query(2*id+2, m+1, r, xa, xb);

        MERGE(id, l, m, r);
        return ret;
    }
}

int main() {
    int nTc;

```

```

scanf("%d", &nTc);

for (int jt = 0; jt < nTc; jt++) {
    int N, Q;
    scanf("%d%d", &N, &Q);

    // Reset
    for (int i = 0; i < 2*MAXN; i++) {
        sTree[i].sum = 0;
        sTree[i].delayedAdd = 0;
    }

    for (int i = 0; i < Q; i++) {
        int a, b, delta, jq;
        scanf("%d", &jq);

        if (jq == 0) {
            scanf("%d%d%d", &a, &b, &delta);
            update(0, 0, N-1, a-1, b-1, delta);
        } else {
            scanf("%d%d", &a, &b);
            printf("%lld\n", query(0, 0, N-1, a-1, b-1));
        }
    }

    return 0;
}

```

1.5 MapComparator

```

/**
 * Description: Custom comparator for map / set
 * Source: StackOverflow
 * Verification: ?
 */

struct cmp {
    bool operator()(const int& l, const int& r) const {
        return l > r;
    }
};

set<int, cmp> s;
map<int, int, cmp> m;

```

1.6 SegmentTree

```

/* RMQ */
class SegmentTree{
private:
    vector<int> st, A;
    int n;
public:
    int left(int p){
        return (p << 1);
    }
    int right(int p){
        return (p << 1) + 1;
    }
    void build(int p, int L, int R){
        if (L == R){
            st[p] = L;
        } else {
            int mid = (L + R) >> 1;
            build(left(p), L, mid);
            build(right(p), mid+1, R);
            int p1 = st[left(p)];
            int p2 = st[right(p)];
            st[p] = (A[p1] <= A[p2] ? p1 : p2);
        }
    }

    int rmq(int p, int L, int R, int i, int j){
        if (i > R || j < L) return -1; //outside range
        if (L >= i && R <= j) return st[p];
        int mid = (L + R) >> 1;
        int p1 = rmq(left(p), L, mid, i, j);
        int p2 = rmq(right(p), mid+1, R, i, j);
        if (p1 == -1) return p2;
        if (p2 == -1) return p1;
        return (A[p1] <= A[p2] ? p1 : p2);
    }
}

```

```
};
```

1.7 Trie

```

struct node2{
    node2 *children[26] = {NULL};
};

struct trie{
    node2 *head = new node2;
    node2 *cur = head;
}

```

```

void insert(string x){
    head = cur;
    for(int i = 0; i < x.size(); i++){
        int val = x[i]-'a';
        if(head->children[val] == NULL){
            head->children[val] = new node2;
        }
        head = head->children[val];
    }

    bool find(string x){
        head = cur;
        for(int i = 0; i < x.size(); i++){
            int val = x[i]-'a';
            if(head->children[val] == NULL) return false;
            head = head->children[val];
        }
        return true;
    }
};

```

2 Dynamic Programming

2.1 Convex Hull Trick

```

//DP convex hull trick (Beware on overflow)
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will
    maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);

```

```

        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (long double)(x->b - y->b)*(z->m - y->m) >= (
            long double)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(
            y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};

```

3 Geometry

3.1 2d

```

/* ftype = int, double, long long , dll */
struct point2d {
    ftype x, y;
    point2d() {}
    point2d(ftype x, ftype y): x(x), y(y) {}
    point2d& operator+=(const point2d &t) {
        x += t.x;
        y += t.y;
        return *this;
    }
    point2d& operator-=(const point2d &t) {
        x -= t.x;
        y -= t.y;
        return *this;
    }
    point2d& operator*=(ftype t) {
        x *= t;
        y *= t;
        return *this;
    }
    point2d& operator/=(ftype t) {
        x /= t;
        y /= t;
        return *this;
    }
};

```

```

point2d operator+(const point2d &t) const {
    return point2d(*this) += t;
}
point2d operator-(const point2d &t) const {
    return point2d(*this) -= t;
}
point2d operator*(ftype t) const {
    return point2d(*this) *= t;
}
point2d operator/(ftype t) const {
    return point2d(*this) /= t;
}
};
point2d operator*(ftype a, point2d b) {
    return b * a;
}

```

3.2 3d

```

/* ftype = int, double, long long , dll */
struct point3d {
    ftype x, y, z;
    point3d() {}
    point3d(ftype x, ftype y, ftype z): x(x), y(y), z(z) {}
    point3d& operator+=(const point3d &t) {
        x += t.x;
        y += t.y;
        z += t.z;
        return *this;
    }
    point3d& operator-=(const point3d &t) {
        x -= t.x;
        y -= t.y;
        z -= t.z;
        return *this;
    }
    point3d& operator*=(ftype t) {
        x *= t;
        y *= t;
        z *= t;
        return *this;
    }
    point3d& operator/=(ftype t) {
        x /= t;
        y /= t;
        z /= t;
        return *this;
    }
};
point3d operator+(const point3d &t) const {

```

```

        return point3d(*this) += t;
    }
    point3d operator-(const point3d &t) const {
        return point3d(*this) -= t;
    }
    point3d operator*(ftype t) const {
        return point3d(*this) *= t;
    }
    point3d operator/(ftype t) const {
        return point3d(*this) /= t;
    }
};
point3d operator*(ftype a, point3d b) {
    return b * a;
}

```

3.3 AreaPolygon

// Let a simple polygon (i.e. without self intersection, not necessarily convex) be given. It is required to calculate its area given its vertices.

```

double area(const vector<point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        point p = i ? fig[i - 1] : fig.back();
        point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}

```

3.4 OrientationTriangle

// Given three points p1, p2 and p3, calculate an oriented (signed) area of a triangle formed by them. The sign of the area is determined in the following way: imagine you are standing in the plane at point p1 and are facing p2. You go to p2 and if p3 is to your right (then we say the three vectors turn "clockwise"), the sign of the area is positive, otherwise it is negative. If the three points are collinear, the area is zero.

```

int signed_area_parallellogram(point2d p1, point2d p2,
    point2d p3) {
    return cross(p2 - p1, p3 - p2);
}

double triangle_area(point2d p1, point2d p2, point2d p3) {
    return abs(signed_area_parallellogram(p1, p2, p3)) / 2.0;
}

bool clockwise(point2d p1, point2d p2, point2d p3) {
    return signed_area_parallellogram(p1, p2, p3) < 0;
}

bool counter_clockwise(point2d p1, point2d p2, point2d p3) {
    return signed_area_parallellogram(p1, p2, p3) > 0;
}

```

3.5 circleLineIntersect

// Given the coordinates of the center of a circle and its radius, and the equation of a line, you're required to find the points of intersection.

```

double r, a, b, c; // given as input
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
if (c*c > r*r*(a*a+b*b)+EPS)
    puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
    puts ("1 point");
    cout << x0 << ' ' << y0 << '\n';
}
else {
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
}
}

```

3.6 convexhull

```

/**
 * Description: Top-bottom convex hull
 * Source: Wikibooks
 * Verification:
 * https://open.kattis.com/problems/convexhull
 */

ll cross(pi O, pi A, pi B) {
    return (ll)(A.f-O.f)*(B.s-O.s) -(ll)(A.s-O.s)*(B.f-O.f);
}

vpi convex_hull(vpi P) {
    sort(all(P)); P.erase(unique(all(P)),P.end());
    int n = sz(P);
    if (n == 1) return P;

    vpi bot = {P[0]};
    FOR(i,1,n) {
        while (sz(bot) > 1 && cross(bot[sz(bot)-2],
            bot.back(), P[i]) <= 0) bot.pop_back();
        bot.pb(P[i]);
    }
    bot.pop_back();

    vpi up = {P[n-1]};
    FORd(i,n-1) {
        while (sz(up) > 1 && cross(up[sz(up)-2],
            up.back(), P[i]) <= 0) up.pop_back();
        up.pb(P[i]);
    }
    up.pop_back();

    bot.insert(bot.end(),all(up));
    return bot;
}

```

3.7 convexhullGrahamScan

```

struct pt {
    double x, y;
};

bool cmp(pt a, pt b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
}

```

```

}

bool ccw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
}

void convex_hull(vector<pt>& a) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), &cmp);
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[
                up.size()-1], a[i]))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2)) {
            while (down.size() >= 2 && !ccw(down[down.size()
                -2], down[down.size()-1], a[i]))
                down.pop_back();
            down.push_back(a[i]);
        }
    }

    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}

```

3.8 crossProduct

```

/* ftype = int, double, long long , dll */

point3d cross(point3d a, point3d b) {
    return point3d(a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x);
}

ftype triple(point3d a, point3d b, point3d c) {
    return dot(a, cross(b, c));
}

```

```

ftype cross(point2d a, point2d b) {
    return a.x * b.y - a.y * b.x;
}

3.9 dotProduct

/* ftype = int, double, long long , dll */
ftype dot(point2d a, point2d b) {
    return a.x * b.x + a.y * b.y;
}

ftype dot(point3d a, point3d b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

ftype norm(point2d a) {
    return dot(a, a);
}

double abs(point2d a) {
    return sqrt(norm(a));
}

double proj(point2d a, point2d b) {
    return dot(a, b) / abs(b);
}

double angle(point2d a, point2d b) {
    return acos(dot(a, b) / abs(a) / abs(b));
}

```

3.10 findIntersectionTwoSegment

```

// You are given two segments AB and
// CD, described as pairs of their endpoints. Each segment
// can
// be a single point if its endpoints are the same. You have
// to
// find the intersection of these segments, which can be
// empty
// (if the segments don't intersect),
// a single point or a segment (if the given segments
// overlap).

const double EPS = 1E-9;

struct pt {
    double x, y;

    bool operator<(const pt& p) const
    {

```

```

        return x < p.x - EPS || (abs(x - p.x) < EPS && y < p.
            y - EPS);
    }
};

struct line {
    double a, b, c;

    line() {}
    line(pt p, pt q)
    {
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }

    void norm()
    {
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS)
            a /= z, b /= z, c /= z;
    }

    double dist(pt p) const { return a * p.x + b * p.y + c; }
};

double det(double a, double b, double c, double d)
{
    return a * d - b * c;
}

inline bool betw(double l, double r, double x)
{
    return min(l, r) <= x + EPS && x <= max(l, r) + EPS;
}

inline bool intersect_1d(double a, double b, double c,
    double d)
{
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}

bool intersect(pt a, pt b, pt c, pt d, pt& left, pt& right)
{

```

```

if (!intersect_1d(a.x, b.x, c.x, d.x) || !intersect_1d(a.
    y, b.y, c.y, d.y))
    return false;
line m(a, b);
line n(c, d);
double zn = det(m.a, m.b, n.a, n.b);
if (abs(zn) < EPS) {
    if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
        return false;
    if (b < a)
        swap(a, b);
    if (d < c)
        swap(c, d);
    left = max(a, c);
    right = min(b, d);
    return true;
} else {
    left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
    left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
    return betw(a.x, b.x, left.x) && betw(a.y, b.y, left.
        y) &&
        betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y
        );
}
}
}

```

3.11 lengthUnionSegment

```

// Given n segments on a line, each described
// by a pair of coordinates (ai1,ai2). We have
// to find the length of their union
int length_union(const vector<pair<int, int>> &a) {
    int n = a.size();
    vector<pair<int, bool>> x(n*2);
    for (int i = 0; i < n; i++) {
        x[i*2] = {a[i].first, false};
        x[i*2+1] = {a[i].second, true};
    }

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

    int result = 0;
    int c = 0;
    for (int i = 0; i < n * 2; i++) {
        if (i > 0 && x[i].first > x[i-1].first && c > 0)
            result += x[i].first - x[i-1].first;
        if (x[i].second)
            c++;
        else

```

```

--c;
    }
    return result;
}

```

3.12 lineIntersection

```

point2d intersect(point2d a1, point2d d1, point2d a2,
    point2d d2) {
    return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
}

point3d intersect(point3d a1, point3d n1, point3d a2,
    point3d n2, point3d a3, point3d n3) {
    point3d x(n1.x, n2.x, n3.x);
    point3d y(n1.y, n2.y, n3.y);
    point3d z(n1.z, n2.z, n3.z);
    point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
    return point3d(triple(d, y, z),
        triple(x, d, z),
        triple(x, y, d)) / triple(n1, n2, n3);
}

```

3.13 twoSegmentIntersect

```

struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator-(const pt& p) const { return pt(x - p.x, y -
        p.y); }
    long long cross(const pt& p) const { return x * p.y - y *
        p.x; }
    long long cross(const pt& a, const pt& b) const { return
        (a - *this).cross(b - *this); }
};

int sgn(const long long& x) { return x >= 0 ? x ? 1 : 0 :
    -1; }

bool inter1(long long a, long long b, long long c, long long
    d) {
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d);
}

```

```

}

bool check_inter(const pt& a, const pt& b, const pt& c,
    const pt& d) {
    if (c.cross(a, d) == 0 && c.cross(b, d) == 0)
        return inter1(a.x, b.x, c.x, d.x) && inter1(a.y, b.y,
            c.y, d.y);
    return sgn(a.cross(b, c)) != sgn(a.cross(b, d)) &&
        sgn(c.cross(d, a)) != sgn(c.cross(d, b));
}

```

4 Graph

4.1 Flow

4.1.1 Bipartite Matching

```

//To handle some corner cases, don't forget to randomize the
//edge order
struct BipartiteMatcher {
    vector<vector<int>> G;
    vector<int> L, R, Viz;

    BipartiteMatcher(int n, int m) :
        G(n, vector<int>(-1, R(m, -1), Viz(n) {}

    void AddEdge(int a, int b) {
        G[a].push_back(b);
    }

    bool Match(int node) {
        if (Viz[node])
            return false;
        Viz[node] = true;

        for (auto vec : G[node]) {
            if (R[vec] == -1) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }

        for (auto vec : G[node]) {
            if (Match(R[vec])) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
    }
}

```

```

    }
}

return false;
}

int Solve() {
    int ok = true;
    while (ok--) {
        fill(Viz.begin(), Viz.end(), 0);
        for (int i = 0; i < (int)L.size(); ++i)
            if (L[i] == -1)
                ok |= Match(i);
    }

    int ret = 0;
    for (int i = 0; i < L.size(); ++i)
        ret += (L[i] != -1);
    return ret;
}
};

```

4.1.2 Dinic

```

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
        cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
    }
};

```

```

adj[v].push_back(m);
adj[u].push_back(m + 1);
m += 2;
}

bool bfs() {
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap - edges[id].flow < 1)
                continue;
            if (level[edges[id].u] != -1)
                continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}

long long dfs(int v, long long pushed) {
    if (pushed == 0)
        return 0;
    if (v == t)
        return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
        int id = adj[v][cid];
        int u = edges[id].u;
        if (level[v] + 1 != level[u] || edges[id].cap -
            edges[id].flow < 1)
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap -
            edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}

long long flow() {
    long long f = 0;
    while (true) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
    }
};

```

```

        if (!bfs())
            break;
        fill(ptr.begin(), ptr.end(), 0);
        while (long long pushed = dfs(s, flow_inf)) {
            f += pushed;
        }
    }
    return f;
}
};
};

```

4.1.3 Edmond-Karp Max flow

```

#define MAXNODE 1000
#define INF 1000000007
int capacity[MAXNODE+5][MAXNODE+5];
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(MAXNODE);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
    }
};

```



```

        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }
    return flow;
}

```

4.1.4 Min-cost flow

```

struct Edge
{
    int from, to, capacity, cost;
    Edge(int from, int to, int capacity, int cost){
        this->from = from;
        this->to = to;
        this->capacity = capacity;
        this->cost = cost;
    }
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<
    int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<int> m(n, 2);
    deque<int> q;
    q.push_back(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        m[u] = 0;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (m[v] == 2) {

```

```

                    m[v] = 1;
                    q.push_back(v);
                } else if (m[v] == 0) {
                    m[v] = 1;
                    q.push_front(v);
                }
            }
        }
    }

    int min_cost_flow(int N, vector<Edge> edges, int K, int s,
        int t) {
        adj.assign(N, vector<int>());
        cost.assign(N, vector<int>(N, 0));
        capacity.assign(N, vector<int>(N, 0));
        for (Edge e : edges) {
            adj[e.from].push_back(e.to);
            adj[e.to].push_back(e.from);
            cost[e.from][e.to] = e.cost;
            cost[e.to][e.from] = -e.cost;
            capacity[e.from][e.to] = e.capacity;
        }

        int flow = 0;
        int cost = 0;
        vector<int> d, p;
        while (flow < K) {
            shortest_paths(N, s, d, p);
            if (d[t] == INF)
                break;

            // find max flow on that path
            int f = K - flow;
            int cur = t;
            while (cur != s) {
                f = min(f, capacity[p[cur]][cur]);
                cur = p[cur];
            }

            // apply flow
            flow += f;
            cost += f * d[t];
            cur = t;
            while (cur != s) {
                capacity[p[cur]][cur] -= f;
                capacity[cur][p[cur]] += f;
                cur = p[cur];
            }
        }

        return cost;
    }
}

```

```

        if (flow < K)
            return 1;
        else
            return cost;
    }
}

```

4.2 LCABinaryLifting

```

int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

```

```
void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

4.3 LCARmq

```
/**
 * Description: Euler Tour LCA w/ O(1) query
 * Source: own
 * Verification: Debug the Bugs
 * Dependency: Range Minimum Query
 */
```

```
template<int SZ> struct LCA {
    vi adj[SZ];
    RMQ<pi, 2*SZ> r;
    vpi tmp;
    int depth[SZ], pos[SZ];
    int N, R = 1;
```

```
void addEdge(int u, int v) {
    adj[u].pb(v), adj[v].pb(u);
}
```

```
void dfs(int u, int prev){
    pos[u] = sz(tmp); depth[u] = depth[prev]+1;
    tmp.pb({depth[u], u});
    for (int v: adj[u]) if (v != prev) {
        dfs(v, u);
        tmp.pb({depth[u], u});
    }
}
```

```
void init(int _N) {
    N = _N;
    dfs(R, 0);
    r.build(tmp);
}
```

```
int lca(int u, int v){
    u = pos[u], v = pos[v];
    if (u > v) swap(u, v);
    return r.query(u, v).s;
}
```

```
int dist(int u, int v) {
    return depth[u]+depth[v]-2*depth[lca(u, v)];
}
};
```

4.4 LCATarjan

```
vector<vector<int>> adj;
vector<vector<int>> queries;
vector<int> ancestor;
vector<bool> visited;
```

```
void dfs(int v)
{
    visited[v] = true;
    ancestor[v] = v;
    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
            union_sets(v, u);
            ancestor[find_set(v)] = v;
        }
    }
    for (int other_node : queries[v]) {
        if (visited[other_node])
            cout << "LCA of " << v << " and " << other_node
                << " is " << ancestor[find_set(other_node)]
                << ".\n";
    }
}
```

```
void compute_LCAs() {
    // initialize n, adj and DSU
    // for (each query (u, v)) {
    //     queries[u].push_back(v);
    //     queries[v].push_back(u);
    // }
```

```
    ancestor.resize(n);
    visited.assign(n, false);
    dfs(0);
}
```

4.5 SCC

4.5.1 2SAT

```
/*
 * Description: Solves 2SAT
 * Source: ?
 * Verification: https://www.spoj.com/problems/BUGLIFE/
 * Also useful: at most one
    http://codeforces.com/contest/1007
    /submission/40284510
 */
```

```
// struct scc
```

```
template<int SZ> struct twosat {
    scc<2*SZ> S;
    int N;
```

```
void OR(int x, int y) {
    S.addEdge(x^1, y);
    S.addEdge(y^1, x);
}
```

```
int tmp[2*SZ];
bitset<SZ> ans;
```

```
bool solve() {
    S.N = 2*N; S.genSCC();
    for (int i = 0; i < 2*N; i += 2)
        if (S.comp[i] == S.comp[i^1])
            return 0;
    reverse(all(S.allComp));
    for (int i: S.allComp) if (tmp[i] == 0)
        tmp[i] = 1, tmp[S.comp[i^1]] = -1;
    FOR(i, N) if (tmp[S.comp[2*i]] == 1) ans[i] = 1;
    return 1;
}
};
```

4.5.2 Kosaraju

```
/**
 * Source: Wikipedia
 * Description: generates SCC in topological order
 * Verification: POI 8 peaceful commission
 */
```

```
template<int SZ> struct scc {
    vi adj[SZ], radj[SZ], todo, allComp;
    int N, comp[SZ];
    bitset<SZ> visit;
```

```
void dfs(int v) {
    visit[v] = 1;
    for (int w: adj[v]) if (!visit[w]) dfs(w);
    todo.pb(v);
}
```

```
void dfs2(int v, int val) {
    comp[v] = val;
    for (int w: radj[v]) if (comp[w] == -1)
        dfs2(w, val);
}
```

```
void addEdge(int a, int b) {
    adj[a].pb(b), radj[b].pb(a);
}
```

```
void genSCC() {
    FOR(i,N) comp[i] = -1, visit[i] = 0;
    FOR(i,N) if (!visit[i]) dfs(i);
    reverse(all(todo)); // toposort
    for (int i: todo) if (comp[i] == -1)
        dfs2(i, i), allComp.pb(i);
}
};
```

4.6 Shortest Path

4.6.1 BellmanFord

```
/**
 * Description: Shortest Path w/ negative edge weights
 * Can be useful with linear programming
 * Constraints of the form  $x_i - x_j < k$ 
 * Source: Own
 * Verification: https://open.kattis.com/problems/shortestpath3
 */
```

```
template<int SZ> struct BellmanFord {
    int n;
    bool bad[SZ];
    vector<pair<pi, int>> edge;

    ll dist[SZ];

    ll query(int x){
        if (bad[x]) return -INF;
        return dist[x];
    }
};
```

```
}

void gen(int s) {
    FOR(i,n) dist[i] = INF, bad[i] = 0;
    dist[s] = 0;
    FOR(i,n) for (auto a: edge)
        if (dist[a.f.f] < INF) dist[a.f.s] =
            min(dist[a.f.s], dist[a.f.f]+a.s);
    for (auto a: edge) if (dist[a.f.f] < INF)
        if (dist[a.f.s] > dist[a.f.f]+a.s)
            bad[a.f.s] = 1;
    FOR(i,n) for (auto a: edge)
        if (bad[a.f.f]) bad[a.f.s] = 1;
}
};
```

4.6.2 Dijkstra

```
/**
 * Description: shortest path!
 * Works with negative edge weights (aka SPFA?)
 */
```

```
template<class T> using pqg = priority_queue<T, vector<T>,
    greater<T>>;
```

```
template<class T> T poll(pqg<T>& x) {
    T y = x.top(); x.pop();
    return y;
}
```

```
template<int SZ> struct Dijkstra {
    ll dist[SZ];
    vpi adj[SZ];
    pqg<pi> q;
```

```
void addEdge(int A, int B, int C) {
    adj[A].pb({B,C}), adj[B].pb({A,C});
}
```

```
void gen(int st) {
    fill_n(dist, SZ, INF);
    q = pqg<pi>(); q.push({dist[st] = 0, st});
    while (sz(q)) {
        auto x = poll(q);
        if (dist[x.s] < x.f) continue;
        for (auto y: adj[x.s]) if (x.f+y.s < dist[y.f])
            q.push({dist[y.f] = x.f+y.s, y.f});
    }
}
```

```
}
};
```

5 Math

5.1 BigInt

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;
```

```
string int_to_string(int i, int width, bool zero) {
    string res = "";
    while (width-->0) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}
```

```
struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }
```

```
    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {
            s[len] = num % BASE;
            num /= BASE;
            len++;
        }
        s[len++] = num;
    }
```

```
    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -= BASE_LENGTH) {
            int tmp = 0;
            int k = i - BASE_LENGTH + 1;
            if (k < 0) k = 0;
```

```

        for (int j = k; j <= i; j++) {
            tmp = tmp*10 + num[j] - '0';
        }
        s[index++] = tmp;
    }
}

void clean() {
    while(len > 1 && !s[len-1]) len--;
}

string str() const {
    string ret = "";
    if (len == 1 && !s[0]) return "0";
    for(int i = 0; i < len; i++) {
        if (i == 0) {
            ret += int_to_string(s[len - i - 1],
                                BASE_LENGTH, false);
        } else {
            ret += int_to_string(s[len - i - 1],
                                BASE_LENGTH, true);
        }
    }
    return ret;
}

unsigned long long ll() const {
    unsigned long long ret = 0;
    for(int i = len-1; i >= 0; i--) {
        ret *= BASE;
        ret += s[i];
    }
    return ret;
}

bigint operator + (const bigint& b) const {
    bigint c = b;
    while (c.len < len) c.s[c.len++] = 0;
    c.s[c.len++] = 0;
    bool r = 0;
    for (int i = 0; i < len || r; i++) {
        c.s[i] += (i < len) * s[i] + r;
        r = c.s[i] >= BASE;
        if (r) c.s[i] -= BASE;
    }
    c.clean();
    return c;
}

bigint operator - (const bigint& b) const {

```

```

    if (operator < (b)) throw "cannot do subtract";
    bigint c = *this;
    bool r = 0;
    for (int i = 0; i < b.len || r; i++) {
        c.s[i] -= b.s[i];
        r = c.s[i] < 0;
        if (r) c.s[i] += BASE;
    }
    c.clean();
    return c;
}

bigint operator * (const bigint& b) const {
    bigint c;
    c.len = len + b.len;
    for(int i = 0; i < len; i++)
        for(int j = 0; j < b.len; j++)
            c.s[i+j] += s[i] * b.s[j];
    for(int i = 0; i < c.len-1; i++){
        c.s[i+1] += c.s[i] / BASE;
        c.s[i] %= BASE;
    }
    c.clean();
    return c;
}

bigint operator / (const int b) const {
    bigint ret;
    int down = 0;
    for (int i = len - 1; i >= 0; i--) {
        ret.s[i] = (s[i] + down * BASE) / b;
        down = s[i] + down * BASE - ret.s[i] * b;
    }
    ret.len = len;
    ret.clean();
    return ret;
}

bool operator < (const bigint& b) const {
    if (len < b.len) return true;
    else if (len > b.len) return false;
    for (int i = 0; i < len; i++)
        if (s[i] < b.s[i]) return true;
        else if (s[i] > b.s[i]) return false;
    return false;
}

bool operator == (const bigint& b) const {
    return !(*this < b) && !(b < (*this));
}

```

```

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};

```

5.2 Number Theory

5.2.1 CRT

```

// A C++ program to demonstrate working of Chinese remainder
// Theorem
#include <bits/stdc++.h>
using namespace std;

// Returns modulo inverse of a with respect to m using
// extended
// Euclid Algorithm. Refer below post for details:
// https://www.geeksforgeeks.org/multiplicative-inverse-
// under-modulo-m/
int inv(int a, int m)
{
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;

    if (m == 1)
        return 0;

    // Apply extended Euclid Algorithm
    while (a > 1)
    {
        // q is quotient
        q = a / m;

        t = m;

        // m is remainder now, process same as
        // euclid's algo
        m = a % m, a = t;

        t = x0;

        x0 = x1 - q * x0;

        x1 = t;
    }

    // Make x1 positive
    if (x1 < 0)

```

```

        x1 += m0;

    return x1;
}

// k is size of num[] and rem[]. Returns the smallest
// number x such that:
// x % num[0] = rem[0],
// x % num[1] = rem[1],
// .....
// x % num[k-2] = rem[k-1]
// Assumption: Numbers in num[] are pairwise coprime
// (gcd for every pair is 1)
int findMinX(int num[], int rem[], int k)
{
    // Compute product of all numbers
    int prod = 1;
    for (int i = 0; i < k; i++)
        prod *= num[i];

    // Initialize result
    int result = 0;

    // Apply above formula
    for (int i = 0; i < k; i++)
    {
        int pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) * pp;
    }

    return result % prod;
}

// Driver method
int main(void)
{
    int num[] = {3, 4, 5};
    int rem[] = {2, 3, 1};
    int k = sizeof(num)/sizeof(num[0]);
    cout << "x is " << findMinX(num, rem, k);
    return 0;
}

```

5.2.2 InverseModulo

```

/**
 * Description : find x such that ax = 1 mod m
 */

```

```

/* case 1 : when(gcd(a,m) = 1) */
/* use extended euclid : find x such that ax + my = 1 */

/* store x, y, and d as global variables */
void extendedEuclid(int a, int b) {
    if (b == 0) { x = 1; y = 0; d = a; return; }
    /* base case */
    extendedEuclid(b, a % b);
    /* similar as the original gcd */
    int x1 = y;
    int y1 = x - (a / b) * y;
    x = x1;
    y = y1;
}

/* compute the first case inverse modulo*/
int firstInverseModulo(int a, int m){
    /* produces x and y, such that ax + my = 1 */
    extendedEuclid(a, m);
    return (x + m)%m;
}

/* case 2 : m is prime */
/* a^(m-1) = 1 mod m */
/* a^(m-2) = a^-1 mod m */

int power(int a,int b){
    int res = 1;
    while (b > 0){
        if (b%2 == 1)
            res *= a;
        b /= 2;
        a *= a;
    }
    return res;
}

int secondInverseModulo(int a,int m){
    return power(a, m-2);
}

```

5.2.3 PrimeFactor

```

/**
 * Description : some function that have relation with prime
                factor
 */

/* find prime factor */

```

```

vector<long long> primefactor(long long N){
    vector<long long> factors;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        while (N%PF == 0){
            N /= PF;
            factors.push_back(PF);
        }
        PF = primes[++idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}

/* number of divisor */
long long numDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        ans *= (power + 1);
        PF = primes[++idx];
    }
    if (N != 1) ans *= 2;
    return ans;
}

/* sum of divisor */
long long sumDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        /* 1 + PF + PF^2 + PF^3 + ... + PF^pow = (a.r^n - 1)
           / (r-1) */
        ans *= ((long long)pow((double)PF, power + 1.0) - 1)
            / (PF - 1);
        PF = primes[++idx];
    }
}

```

```

    if (N != 1) ans *= ((long long)pow((double)N, 2.0) - 1) /
        (N - 1);
    return ans;
}

/* Euler Phi */
long long eulerPhi(long long N){
    long long idx = 0;
    long long PF = primes[idx];
    long long ans = N;
    while (PF <= (long long)sqrt(N)){
        if (N%PF == 0) ans -= ans / PF;
        while (N%PF == 0) N /= PF;
        PF = primes[++idx];
    }
    if (N != 1) ans -= ans / N;
    return ans;
}

```

5.2.4 Sieve

```

/**
 * Description :Test Primality up to n in O(nlog(logn))
 */

const int SZ = 1e7;
bitset<SZ> bs;
vector<long long> primes;

void sieve(){
    bs.set();
    bs[0] = false; bs[1] = false;
    for (long long i = 2; i <= SZ; i++){
        if (bs[i]){
            primes.push_back(i);
            for (long long j = i * i; j <= SZ; j+=i)
                bs[j] = false;
        }
    }
}

```

5.2.5 extendedEuclid

```

/**
 * Description : find x and y such that ax + by = 1
 */

/* store x, y, and d as global variables */

```

```

void extendedEuclid(int a, int b) {
    if (b == 0) { x = 1; y = 0; d = a; return; }
    /* base case */
    extendedEuclid(b, a % b);
    /* similar as the original gcd */
    int x1 = y;
    int y1 = x - (a / b) * y;
    x = x1;
    y = y1;
}

```

5.3 Polynomial

5.3.1 FFT mod

/*
Description: Allows multiplication of polynomials in
general moduli.
Verification:
<http://codeforces.com/contest/960/submission/37085144>
*/

```

namespace FFTmod {
    int get(int s) {
        return s > 1 ? 32 - __builtin_clz(s - 1) : 0;
    }
    void fft(vcd& a, bool inv){
        int n = sz(a), j = 0;
        vcd roots(n/2);
        FOR(i,1,n) {
            int bit = (n >> 1);
            while (j >= bit){
                j -= bit;
                bit >>= 1;
            }
            j += bit;
            if(i < j) swap(a[i], a[j]);
        }

```

```

        ld ang = 2 * M_PI / n * (inv ? -1 : 1);
        FOR(i,n/2) roots[i] = cd(cos(ang * i), sin(ang * i));

```

```

        for (int i=2; i<=n; i<=1){
            int step = n / i;
            for(int j=0; j<n; j+=i){
                for(int k=0; k<i/2; k++){
                    cd u = a[j+k], v = a[j+k+i/2] * roots[step * k];
                    a[j+k] = u+v;
                    a[j+k+i/2] = u-v;
                }
            }
        }
    }
}

```

```

    }
}
}
if (inv) FOR(i,n) a[i] /= n;
}

v1 conv(v1 a, v1 b, ll mod){
    int s = sz(a)+sz(b)-1, L = get(s), n = 1<<L;
    vcd v1(n), v2(n), r1(n), r2(n);
    FOR(i,sz(a)) v1[i] = cd(a[i] >> 15, a[i] & 32767);
    FOR(i,sz(b)) v2[i] = cd(b[i] >> 15, b[i] & 32767);
    fft(v1, 0); fft(v2, 0);

    FOR(i,n) {
        int j = (i ? (n - i) : i);
        cd ans1 = (v1[i] + conj(v1[j])) * cd(0.5, 0);
        cd ans2 = (v1[i] - conj(v1[j])) * cd(0, -0.5);
        cd ans3 = (v2[i] + conj(v2[j])) * cd(0.5, 0);
        cd ans4 = (v2[i] - conj(v2[j])) * cd(0, -0.5);
        r1[i] = (ans1 * ans3) + (ans1 * ans4) * cd(0, 1);
        r2[i] = (ans2 * ans3) + (ans2 * ans4) * cd(0, 1);
    }
    fft(r1, 1); fft(r2, 1);
    v1 ret(n);
    FOR(i,n) {
        ll av = (ll)round(r1[i].real());
        ll bv = (ll)round(r1[i].imag()) + (ll)round(r2[i].real());
        ;
        ll cv = (ll)round(r2[i].imag());
        av %= mod, bv %= mod, cv %= mod;
        ret[i] = (av << 30) + (bv << 15) + cv;
        ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
    }
    ret.resize(s);
    return ret;
}
}

```

```
using namespace FFTmod;
```

5.3.2 FFT

```

#include <algorithm>
#include <cstdio>
#include <ctime>
#include <vector>
#include <complex>

using namespace std;

```

```

typedef complex<double> cd;
typedef vector<cd> vcd;

vcd fft(const vcd &as) {
    int n = as.size();
    int k = 0; // n
    while ((1 << k) < n) k++;
    vector<int> rev(n);
    rev[0] = 0;
    int high1 = -1;
    for (int i = 1; i < n; i++) {
        if ((i & (i - 1)) == 0) //
            i = i - 1;

        high1++;
        rev[i] = rev[i ^ (1 << high1)]; //

        rev[i] |= (1 << (k - high1 - 1)); //
    }

    vcd roots(n);
    for (int i = 0; i < n; i++) {
        double alpha = 2 * M_PI * i / n;
        roots[i] = cd(cos(alpha), sin(alpha));
    }

    vcd cur(n);
    for (int i = 0; i < n; i++)
        cur[i] = as[rev[i]];

    for (int len = 1; len < n; len <= 1) {
        vcd ncur(n);
        int rstep = roots.size() / (len * 2);
        for (int pdest = 0; pdest < n; pdest++) {
            int p1 = pdest;
            for (int i = 0; i < len; i++) {
                cd val = roots[i * rstep] * cur[p1 + len];
                ncur[pdest] = cur[p1] + val;
                ncur[pdest + len] = cur[p1] - val;
                pdest++, p1++;
            }
            pdest += len;
        }
        cur.swap(ncur);
    }
    return cur;
}

```

```

vcd fft_rev(const vcd &as) {
    vcd res = fft(as);
    for (int i = 0; i < (int)res.size(); i++) res[i] /= as.size();
    reverse(res.begin() + 1, res.end());
    return res;
}

int main() {
    int n;
    scanf("%d", &n);
    vcd as(n);
    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        as[i] = x;
    }

    clock_t stime = clock();
    vcd res = fft(as);
    fprintf(stderr, "%d\n", (int)(clock() - stime));
    for (int i = 0; i < n; i++)
        printf("%.4lf %.4lf\n", res[i].real(), res[i].imag());

    stime = clock();
    vcd as2 = fft_rev(res);
    fprintf(stderr, "%d\n", (int)(clock() - stime));
    for (int i = 0; i < n; i++)
        printf("%.4lf %.4lf\n", as2[i].real(), as2[i].imag());
    return 0;
}

```

5.3.3 FFT2

```

**
* Description:
* Source: KACTL, https://pastebin.com/3Tnj5mRu
* Verification: SPOJ polymul, CSA manhattan
*/

```

```

namespace FFT {
    int get(int s) {
        return s > 1 ? 32 - __builtin_clz(s - 1) : 0;
    }

    vcd fft(vcd& a) {
        int n = sz(a), x = get(n);
        vcd res, RES(n), roots(n);
        FOR(i,n) roots[i] = cd(cos(2*M_PI*i/n), sin(2*M_PI*i/n));
    }
}

```

```

    res = a;
    FOR(i,1,x+1) {
        int inc = n>>i;
        FOR(j,inc) for (int k = 0; k < n; k += inc){
            int t = 2*k%n+j;
            RES[k+j] = res[t]+roots[k]*res[t+inc];
        }
        swap(res,RES);
    }

    return res;
}

vcd fft_rev(vcd& a) {
    vcd res = fft(a);
    FOR(i,sz(res)) res[i] /= sz(a);
    reverse(res.begin() + 1, res.end());
    return res;
}

vcd brute(vcd& a, vcd& b) {
    vcd c(sz(a)+sz(b)-1);
    FOR(i,sz(a)) FOR(j,sz(b)) c[i+j] += a[i]*b[j];
    return c;
}

vcd conv(vcd a, vcd b) {
    int s = sz(a)+sz(b)-1, L = get(s), n = 1<<L;
    if (s <= 0) return {};
    if (s <= 200) return brute(a,b);
    a.resize(n); a = fft(a);
    b.resize(n); b = fft(b);
    FOR(i,n) a[i] *= b[i];
    a = fft_rev(a);

    a.resize(s);

    return a;
}

vl convll(vl a, vl b) {
    vcd A(sz(a)); FOR(i,sz(a)) A[i] = a[i];
    vcd B(sz(b)); FOR(i,sz(b)) B[i] = b[i];
    vcd X = conv(A,B);
    vl x(sz(X)); FOR(i,sz(X)) x[i] = round(X[i].real());
    return x;
}
}

```

6 Misc

6.1 Mo

```
bool comp(query a, query b){
    if (a.L / block == b.L/block)
        return a.R < b.R;
    return a.L/block < b.L/block;
}
```

```
void add(int x){
    cnt[x]++;
    if (cnt[x] == 1) distinct++;
}
```

```
void del(int x){
    cnt[x]--;
    if (cnt[x] == 0) distinct--;
}
```

```
int main(){
    OPTIMATION
    cin >> N;
    for (int i = 0; i < N; i++){
        cin >> arr[i];
        block = (int)sqrt(N) + 1;
        cin >> Q;
        for (int i = 0; i < Q; i++){
            int tl, tr;
            cin >> tl >> tr;
            tl--; tr--;
            q[i].L = tl;
            q[i].R = tr;
            q[i].no = i;
        }
        sort(q, q+Q, comp);
        currL = 0;
        currR = 0;
        for (int i = 0; i < Q; i++){
            int L = q[i].L;
            int R = q[i].R;
            while (currL < L) {
                del(arr[currL]);
                currL++;
            }
            while (currL > L){
                add(arr[currL-1]);
                currL--;
            }
            while (currR <= R) {
```

```
                add(arr[currR]);
                currR++;
            }
            while (currR > R+1){
                del(arr[currR-1]);
                currR--;
            }
            ans[q[i].no] = distinct;
        }
        for (int i = 0; i < Q; i++){
            cout << ans[i] << '\n';
        }
        return 0;
    }
}
```

6.2 mt19937

```
#include <bits/stdc++.h>
using namespace std;
//Application of mt19937

const int N = 3000000;

double average_distance(const vector<int> &permutation) {
    double distance_sum = 0;

    for (int i = 0; i < N; i++)
        distance_sum += abs(permutation[i] - i);

    return distance_sum / N;
}

int main() {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch
        ().count());
    vector<int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    shuffle(permutation.begin(), permutation.end(), rng);
    cout << average_distance(permutation) << '\n';

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[
            uniform_int_distribution<int>(0, i)(rng)]);
```

```
    cout << average_distance(permutation) << '\n';
}
```

6.3 ternary

/*We are given a function $f(x)$ which is unimodal on an interval $[l;r]$. By unimodal function, we mean one of two behaviors of the function:

The function strictly increases first, reaches a maximum (at one point or at a segment), and then strictly decreases.

The function strictly decreases first, reaches a minimum, and then strictly increases.

In this article we will assume the first scenario, and the second is be completely symmetrical to it.

The task is to find the maximum of function $f(x)$ on the interval $[l;r]$.*/

```
double ternary_search(double l, double r) {
    double eps = 1e-9; //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1); //evaluates the function at m1
        double f2 = f(m2); //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l); //return the maximum of f(x)
    in [l, r]
}
```

7 Setup

7.1 BqTemplate

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
```



```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/rope>

using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx

typedef long long ll;
typedef long double ld;
typedef complex<ld> cd;

typedef pair<int, int> pi;
typedef pair<ll,ll> pl;
typedef pair<ld,ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
typedef vector<cd> vcd;

template <class T> using Tree = tree<T, null_type, less<T>,
rb_tree_tag,tree_order_statistics_node_update>;

#define FOR(i, a, b) for (int i=a; i<(b); i++)
#define FOR(i, a) for (int i=0; i<(a); i++)
#define FORd(i,a,b) for (int i = (b)-1; i >= a; i--)
#define FORd(i,a) for (int i = (a)-1; i >= 0; i--)

#define sz(x) (int)(x).size()
#define mp make_pair
#define pb push_back
#define f first
#define s second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()

const int MOD = 1000000007;
const ll INF = 1e18;
const int MX = 100001;

int main(){
}
```

7.2 C++Template

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

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

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

typedef long long LL;
typedef vector<int> vi;
typedef pair<int,int> ii;

const int MOD = 1e9 + 7;
const LL INF = 1e18;

void fastscan(int &number) {
    //variable to indicate sign of input number
    bool negative = false;
    register int c;

    number = 0;

    // extract current character from buffer
    c = getchar();
    if (c=='-')
    {
        // number is negative
        negative = true;

        // extract the next character from the buffer
        c = getchar();
    }

    // Keep on extracting characters if they are integers
    // i.e ASCII Value lies from '0'(48) to '9' (57)
    for (; (c>47 && c<58); c=getchar())
        number = number *10 + c - 48;

    // if scanned input has a negative sign, negate the
    // value of the input number
    if (negative)
        number *= -1;
}

int main(){
    //cin / cout user
```

```
//ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0)

return 0;
}
```

7.3 FastScanner

```
class FastScanner {
private:
    Inputstream stream;
    byte[] buf = new byte[1024];
    int curChar;
    int numChars;

public:
    FastScanner(Inputstream stream) {
        this.stream = stream;
    }

    int read() {
        if (numChars == -1)
            throw new InputMismatchException();
        if (curChar >= numChars) {
            curChar = 0;
            try {
                numChars = stream.read(buf);
            } catch (IOException e) {
                throw new InputMismatchException();
            }
            if (numChars <= 0) return -1;
        }
        return buf[curChar++];
    }

    boolean isSpaceChar(int c) {
        return c == ' ' || c == '\n' || c == '\r' || c
            == '\t' || c == -1;
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }

    public long nextLong() {
        return Long.parseLong(next());
    }

    public double nextDouble() {
        return Double.parseDouble(next());
    }
}
```

```
public String next() {
    int c = read();
    while (isSpaceChar(c)) c = read();
    StringBuilder res = new StringBuilder();
    do {
        res.appendCodePoint(c);
        c = read();
    } while (!isSpaceChar(c));
    return res.toString();
}
```

```
public String nextLine() {
    int c = read();
    while (isEndline(c))
        c = read();
    StringBuilder res = new StringBuilder();
    do {
        res.appendCodePoint(c);
        c = read();
    } while (!isEndline(c));
    return res.toString();
}
```

7.4 sublimesetup

```
{
    "cmd": ["g++", "-std=c++11", "$file_name", "-o", "${
        file_base_name}.exe", "&&", "start", "cmd", "/k" , "
        $file_base_name"],
    "selector": "source.cpp",
    "file_regex": "^(..[^:]*):([0-9]+)?(?:[0-9]+)?(?:\\.*)$",
    "working_dir": "${file_path}",
    "shell": true
}
```

8 String

8.1 Hashing

```
/*use double hashing */
```

```
long long compute_hash(string const& s) {
    const int p = 31; //another good option : p = 53
    const int m = 1e9 + 9;
```

```
long long hash_value = 0;
long long p_pow = 1;
for (char c : s) {
    hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
    p_pow = (p_pow * p) % m;
}
return hash_value;
}
```

8.2 KMP

```
#define HHH 10003

int ne[HHH]; // next[], if par[i] not matched, jump to i = ne[i]
int kmp(string& par, string& ori) {
    ne[0] = -1;
    for (int p = ne[0], i = 1; i < par.length(); i++) {
        while (p >= 0 && par[p+1] != par[i])
            p = ne[p];
        if (par[p+1] == par[i])
            p++;
        ne[i] = p;
    }

    int match = 0;
    for (int p = -1, q = 0; q < ori.length(); q++) {
        while (p >= 0 && par[p+1] != ori[q])
            p = ne[p];
        if (par[p+1] == ori[q])
            p++;
        if (p + 1 == par.length()) { // match!
            p = ne[p];
            match++;
        }
    }

    return match; // return number of occurrence
}
```

```
int main () {
    int n; cin >> n;
    string par, ori;
    while (cin >> par >> ori)
        cout << kmp(par, ori) << endl;
    return 0;
}
```

8.3 Manacher

```
int dp[HHH];
int lengthLongestPalindromSubstring(string& s) {
    memset(dp, 0, sizeof(dp));
    int ans = 0;
    int pivot = 1;
    int len = s.length() * 2; // _s0_s1_s2 = 2 * length
    for (int i = 1; i < len; i++) {
        int pBorder = pivot + dp[pivot];
        int iBorder = i;
        if (iBorder < pBorder && 2 * pivot - i > 0) {
            dp[i] = dp[2*pivot-i];
            iBorder = min(pBorder, i + dp[i]);
        }

        if (iBorder >= pBorder) {
            int j = iBorder + (iBorder % 2 ? 2 : 1);
            for (; j < len && 2*i-j > 0 && s[j/2] == s[(2*i-j)/2]; j += 2)
                ;
            iBorder = j - 2;
            dp[i] = iBorder - i;
            pivot = i;
        }
        ans = max(ans, dp[i] + 1);
    }

    return ans;
}
```

```
int main () {
    int n; cin >> n;
    string s;
    while (cin >> s)
        cout << lengthLongestPalindromSubstring(s) << endl;
    return 0;
}
```

8.4 rabinkarp

```
/* Problem: Given two strings - a pattern s and a text t,
determine if the pattern appears in the text and if it does,
enumerate all its occurrences in O(|s|+|t|) time.*/
vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();
```

```

vector<long long> p_pow(max(S, T));
p_pow[0] = 1;
for (int i = 1; i < (int)p_pow.size(); i++)
    p_pow[i] = (p_pow[i-1] * p) % m;

vector<long long> h(T + 1, 0);
for (int i = 0; i < T; i++)

```

```

    h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
long long h_s = 0;
for (int i = 0; i < S; i++)
    h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

vector<int> occurrences;
for (int i = 0; i + S - 1 < T; i++) {

```

```

    long long cur_h = (h[i+S] + m - h[i]) % m;
    if (cur_h == h_s * p_pow[i] % m)
        occurrences.push_back(i);
}
return occurrences;
}

```
