模板1118补

目录

三维计算几何

```
const double eps = 1e-8;
const double PI = acos(-1.0);

struct Matrix
{
    double mat[4][4];

    Matrix()
    {
        memset(mat, 0, sizeof(mat));
        mat[3][3] = 1;
    }

    Matrix operator*(const Matrix& m) const
    {
        Matrix res;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                res.mat[i][j] = 0;
                for (int k = 0; k < 4; k++)
                {
                    res.mat[i][j] += mat[i][k] * m.mat[k][j];
                }
            }
        }
        return res;
    }
    Matrix operator+(const Matrix& m) const
    {
        Matrix res;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                res.mat[i][j] = mat[i][j] + m.mat[i][j];
            }
        }
        return res;
    }
    Matrix rotate(double x, double y, double z, double c, double s)
    {
        Matrix res;
        res.mat[0][0] = c + (1 - c) * x * x;
        res.mat[0][1] = (1 - c) * x * y - s * z;
        res.mat[0][2] = (1 - c) * x * z + s * y;
        res.mat[1][0] = (1 - c) * y * x + s * z;
        res.mat[1][1] = c + (1 - c) * y * y;
        res.mat[1][2] = (1 - c) * y * z - s * x;
        res.mat[2][0] = (1 - c) * z * x - s * y;
        res.mat[2][1] = (1 - c) * z * y + s * x;
        res.mat[2][2] = c + (1 - c) * z * z;
        return res;
    }
    Matrix rotate(double x, double y, double z, double theta)
```

```
    {
        double c = cos(theta);
        double s = sin(theta);
        return rotate(x, y, z, c, s);
    }
    Matrix translate(double x, double y, double z)
    {
        Matrix res;
        res.mat[0][0] = 1;
        res.mat[1][1] = 1;
        res.mat[2][2] = 1;
        res.mat[3][0] = x;
        res.mat[3][1] = y;
        res.mat[3][2] = z;
        return res;
    }
    Matrix scale(double x, double y, double z)
    {
        Matrix res;
        res.mat[0][0] = x;
        res.mat[1][1] = y;
        res.mat[2][2] = z;
        return res;
    }
};

int sig(double t)
{
    return (t > eps) - (t < -eps);
}

inline double sqr(double t)
{
    return t * t;
}

struct Point
{
    double x, y, z;

    Point()
    {
    }
    Point(double p, double q, double r) :
            x(p), y(q), z(r)
    {
    }

    double cross2(const Point& p) const
    {
        return x * p.y - y * p.x;
    }
    double mod2() const
    {
        return sqrt(x * x + y * y);
    }
    double dot(const Point& p) const
    {
```

```
        return x * p.x + y * p.y + z * p.z;
    }
    double dist(const Point& p) const
    {
        double a = x - p.x;
        double b = y - p.y;
        double c = z - p.z;
        return sqrt(a * a + b * b + c * c);
    }
    double mod() const
    {
        return sqrt(x * x + y * y + z * z);
    }
    double rad(const Point& p) const // cos (point to point)
    {
        return dot(p) / mod() / p.mod();
    }
    Point cross(const Point& p) const
    {
        return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y
* p.x);
    }
    double mix(const Point& b, const Point& c) const
    {
        return this->cross(b).dot(c);
    }
    int sameFace(const Point& p, const Point& q, const Point& r)
const
    {
        return sig((p - *this).mix(q - *this, r - *this)) == 0;
        //return sig(this->mix(p,q))==0;
    }

    Point operator-(const Point& p) const
    {
        return Point(x - p.x, y - p.y, z - p.z);
    }
    Point operator+(const Point& p) const
    {
        return Point(x + p.x, y + p.y, z + p.z);
    }
    Point operator*(double p) const
    {
        return Point(x * p, y * p, z * p);
    }
    Point operator/(double p) const
    {
        return Point(x / p, y / p, z / p);
    }
    bool operator<(const Point& p) const
    {
        double t = this->cross2(p);
        if (sig(t))
            return sig(t) > 0;
        return sig(this->mod2() - p.mod2()) > 0;
    }
    Point unit() const
    {
```

```
        double m = mod();
        return Point(x / m, y / m, z / m);
    }
    Point perform(const Matrix& m) const
    {
        Point delta = Point(m.mat[3][0], m.mat[3][1], m.mat[3][2]);
        return Point(x * m.mat[0][0] + y * m.mat[1][0] + z * m.mat[2]
[0],
                x * m.mat[0][1] + y * m.mat[1][1] + z * m.mat[2][1],
                x * m.mat[0][2] + y * m.mat[1][2] + z * m.mat[2][2]) +
delta;
    }
};

const Point ZERO(0, 0, 0);

struct Line
{
    Point n;
    Point m;
    Line()
    {
    }
    Line(const Point& p, const Point& q) :
           n(p), m(q)
    {
    }

    double rad(const Line& p) const // cos (line to line)
    {
        return fabs(n.rad(p.n));
    }
    double dist(const Point& p) const // line to point
    {
        return n.cross(p - m).mod() / n.mod();
    }
    int isPar(const Line& p) const
    {
        double t = n.x * p.n.y - n.y * p.n.x;
        double s = n.y * p.n.z - n.z * p.n.y;
        return sig(t - s) == 0;
    }
    int isVert(const Line& p) const
    {
        return sig(n.dot(p.n)) == 0;
    }
    Line perform(const Matrix& mat) const
    {
        return Line(n.perform(mat), m.perform(mat));
    }
    int containsPoint(const Point& p) const // TEST NEEDED!
    {
        return sig((p - m).cross(n).mod()) == 0;
    }
    int sameFace(const Line& p) const
    {
        return m.sameFace(m + n, p.m, p.m + p.n);
    }
```

```
    Point project(const Point& p) const
    {
        double l = this->n.x;
        double m = this->n.y;
        double n = this->n.z;

        double down = m * m + n * n + l * l;
        down /= l;
        double x = this->m.x;
        double y = this->m.y;
        double z = this->m.z;

        double up = this->n.dot(p);

        return Point(up + (sqr(m) + sqr(n)) / l * x - m * y - n * z,
               up + (sqr(l) + sqr(n)) / l * y - l * x - n * z,
               up + (sqr(l) + sqr(m)) / l * z - l * x - m * y);
    }
};

struct Face
{
    Point n;
    Point m;
    double d;

    Face()
    {
    }
    Face(const Point& p, double q) :
           n(p), d(q)
    {
        m = getPoint();
    }
    Face(const Point& p, const Point& q) :
           n(p), m(q)
    {
        d = -n.dot(m);
    }

    double rad(const Face& p) const // cos (face to face)
    {
        return fabs(n.rad(p.n));
    }
    double rad(const Line& p) const // cos (face to line)
    {
        return fabs(n.rad(p.n));
    }
    double dist(const Point& p) const
    {
        return fabs(p.dot(n) + d) / n.mod();
    }
    int isVert(const Face& p) const
    {
        return sig(n.dot(p.n)) == 0;
    }
    int isPar(const Face& p) const
    {
```

```
        double t = n.x * p.n.y - n.y * p.n.x;
        double s = n.y * p.n.z - n.z * p.n.y;
        return sig(t - s) == 0;
    }
    int isPar(const Line& p) const
    {
        return sig(n.dot(p.n)) == 0;
    }
    int isVert(const Line& p) const
    {
        double t = n.x * p.n.y - n.y * p.n.x;
        double s = n.y * p.n.z - n.z * p.n.y;
        return sig(t - s) == 0;
    }
    Point getPoint() const
    {
        if (sig(n.x))
            return Point(-d / n.x, 0, 0);
        if (sig(n.y))
            return Point(0, -d / n.y, 0);
        return Point(0, 0, -d / n.z);
    }
    Point intersect(const Line& p) const // assume p is not PAR to
face
    {
        double A = n.x, B = n.y, C = n.z, D = d;
        double l = p.n.x, m = p.n.y, n = p.n.z;
        double x0 = p.m.x, y0 = p.m.y, z0 = p.m.z;
        double down = this->n.dot(p.n);
        double x = (B * m + C * n) * x0 - l * (B * y0 + C * z0 + D);
        double y = (A * l + C * n) * y0 - m * (A * x0 + C * z0 + D);
        double z = (A * l + B * m) * z0 - n * (A * x0 + B * y0 + D);
        return Point(x, y, z) / down;
    }
};

Line commonVert(const Line& p, const Line& q)
{
    Line res;
    res.n = p.n.cross(q.n);
    Face tmp(res.n.cross(p.n), p.m);
    res.m = tmp.intersect(q);
    return res;
}

Point intersect(const Line& p, const Line& q)
{
    Face f(p.n.cross(q.n).cross(p.n), p.m);
    return f.intersect(q);
}

Matrix buildRotate(double a, double b, double c)
{
    Matrix toX;
    if (sig(a * a + b * b))
        toX = toX.rotate(0, 0, 1, b / sqrt(a * a + b * b),
               -a / sqrt(a * a + b * b));
    else
```

```
        toX.mat[0][0] = toX.mat[1][1] = toX.mat[2][2] = 1;
    double x = sqrt(a * a + b * b), y = c;
    Matrix toZ = toX
           * toX.rotate(1, 0, 0, y / sqrt(x * x + y * y),
                  -x / sqrt(x * x + y * y));
    return toZ;
}
```

点在多边形内

```
int isin(Point point[], int N, const Point& p)
{
    int cnt = 0;
    for (int i = 1; i <= N; i++)
    {
        if (sig((p - point[i]).cross2(p - point[i + 1])) == 0
                && sig((p - point[i]).dot2(p - point[i + 1])) <= 0)
            return 1;
        if (sig(point[i].y - point[i + 1].y) == 0)
            continue;
        Point tmp = getSol(p, Point(p.x + 1, p.y, 0), point[i],
point[i + 1]);
        if (sig((tmp - p).dot2(Point(1, 0, 0))) < 0)
            continue;
        double x = min(point[i].y, point[i + 1].y);
        double y = max(point[i].y, point[i + 1].y);
        if (sig(x - p.y) == 0)
            continue;
        if (sig((p.y - x) * (p.y - y)) <= 0)
            cnt++;
    }
    if (cnt % 2 == 0)
        return 0;
    return 1;
}
```

光线反射

```
struct Point
{
    double x, y, z;

    Point()
    {
    }
    Point(double p, double q, double r) :
           x(p), y(q), z(r)
    {
    }

    double dot(const Point& p) const
    {
        return x * p.x + y * p.y + z * p.z;
    }
    double dist(const Point& p) const
    {
        double a = x - p.x;
        double b = y - p.y;
        double c = z - p.z;
        return sqrt(a * a + b * b + c * c);
    }
    double mod() const
    {
        return sqrt(x * x + y * y + z * z);
    }
    double mod2() const
    {
        return x * x + y * y + z * z;
    }
    double rad(const Point& p) const // cos (point to point)
    {
        return dot(p) / mod() / p.mod();
    }
    Point cross(const Point& p) const
    {
        return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y
* p.x);
    }
    double mix(const Point& b, const Point& c) const
    {
        return this->cross(b).dot(c);
    }
    int sameFace(const Point& p, const Point& q, const Point& r)
const
    {
        return sig((p - *this).mix(q - *this, r - *this)) == 0;
        //return sig(this->mix(p,q))==0;
    }

    Point operator-(const Point& p) const
    {
        return Point(x - p.x, y - p.y, z - p.z);
    }
    Point operator+(const Point& p) const
```

```
    {
        return Point(x + p.x, y + p.y, z + p.z);
    }
    Point operator*(double p) const
    {
        return Point(x * p, y * p, z * p);
    }
    Point operator/(double p) const
    {
        return Point(x / p, y / p, z / p);
    }
    Point unit() const
    {
        double m = mod();
        return Point(x / m, y / m, z / m);
    }
    double project(const Point& p) const
    {
        return this->dot(p) / p.mod();
    }
};

struct Sphere
{
    Point center;
    double r;

    Sphere()
    {
    }
    Sphere(const Point& p, double q) :
            center(p), r(q)
    {
    }
};

struct Line
{
    Point a, b;

    double dist(const Point& p) const // line to point
    {
        Point n = b - a;
        Point m = a;
        return n.cross(p - m).mod() / n.mod();
    }
    Point project(const Point& p) const
    {
        double prj = (p - a).project(b - a);
        return (b - a).unit() * prj + a;
    }
    int intersect(const Sphere& p, Point& res) const
    {
        double prj = (p.center - a).project(b - a);
        if (sig(prj) < 0)
            return 0;
        double d = this->dist(p.center);
        if (sig(d - p.r) > 0)
```

```
            return 0;
        d = sqrt(p.r * p.r - d * d);
        res = (b - a).unit() * (prj - d) + a;
        return 1;
    }
    Line reflect(const Sphere& p, const Point &pnt) const
    {
        Line tmp;
        tmp.a = p.center;
        tmp.b = pnt;
        Point mid = tmp.project(a);
        Line res;
        res.b = mid * 2 - a;
        res.a = pnt;
        return res;
    }
};

int N;

Sphere sphere[100];

Line source;

void init()
{
    scanf("%d", &N);
    for (int i = 1; i <= N; i++)
    {
        scanf("%lf%lf%lf%lf", &sphere[i].center.x,
&sphere[i].center.y,
              &sphere[i].center.z, &sphere[i].r);
    }
    scanf("%lf%lf%lf", &source.a.x, &source.a.y, &source.a.z);
    scanf("%lf%lf%lf", &source.b.x, &source.b.y, &source.b.z);
}

int result[100];

void work()
{
    int last = 0;
    Line current = source;
    int cnt = 0;
    while (1)
    {
        int flag = 0;
        Point pnt;
        int id;
        for (int i = 1; i <= N; i++)
        {
            if (i == last)
                continue;
            Point tmp;
            if (current.intersect(sphere[i], tmp) == 0)
                continue;
            if (!flag)
            {
```

```
                pnt = tmp;
                id = i;
            }
            else
            {
                if (sig(tmp.dist(current.a) - pnt.dist(current.a)) < 0)
                {
                    pnt = tmp;
                    id = i;
                }
            }
            flag = 1;
        }
        if (!flag)
            break;
        last = id;
        current = current.reflect(sphere[id], pnt);
        cnt++;
        result[cnt] = id;
        if (cnt > 12)
            break;
    }
    if (cnt <= 10)
    {
        for (int i = 1; i <= cnt; i++)
        {
            printf("%d", result[i]);
            if (i == cnt)
                puts("");
            else
                putchar(' ');
        }
    }
    else
    {
        for (int i = 1; i <= 10; i++)
        {
            printf("%d", result[i]);
            if (i == 10)
                puts(" etc.");
            else
                putchar(' ');
        }
    }
}
```

最小包围矩形

```
struct Point
{
    double x, y;

    Point()
    {
    }
    Point(double p, double q) :
            x(p), y(q)
    {
    }

    double cross(const Point& p) const
    {
        return x * p.y - y * p.x;
    }
    double dot(const Point& p) const
    {
        return x * p.x + y * p.y;
    }
    double dist(const Point& p) const
    {
        double a = x - p.x;
        double b = y - p.y;
        return sqrt(a * a + b * b);
    }
    Point operator-(const Point& p) const
    {
        return Point(x - p.x, y - p.y);
    }
    Point operator+(const Point& p) const
    {
        return Point(x + p.x, y + p.y);
    }
    int id;
    bool operator<(const Point& p) const
    {
        const Point zero(0, 0);
        double t = this->cross(p);
        if (sig(t))
            return sig(t) > 0;
        return sig(this->dist(zero) - p.dist(zero)) > 0;
    }
    Point trans() const
    {
        return Point(-y, x);
    }
};

Point getSol(Point a, Point b, Point c, Point d)
{
    double s = (d.y - b.y) * (b.x - a.x) * (d.x - c.x)
            + (b.y - a.y) * b.x * (d.x - c.x) - (d.y - c.y) * (b.x -
a.x) * d.x;
    double t = (b.y - a.y) * (d.x - c.x) - (d.y - c.y) * (b.x - a.x);
    double x = s / t;
```

```
    double s1 = (d.x - b.x) * (b.y - a.y) * (d.y - c.y)
            + (b.x - a.x) * b.y * (d.y - c.y) - (d.x - c.x) * (b.y -
a.y) * d.y;
    double t1 = (b.x - a.x) * (d.y - c.y) - (d.x - c.x) * (b.y -
a.y);
    double y = s1 / t1;
    Point res(x, y);
    return res;
}

int N;

struct Poly
{
    Point point[maxn], stack[2 * maxn];
    int N, top;

    void init()
    {
        N = 0;
    }
    void add(const Point& p)
    {
        point[++N] = p;
    }
    void convexHull()
    {
        for (int i = 1; i <= N; i++)
        {
            if (sig(point[1].y - point[i].y) > 0
                    || (sig(point[1].y - point[i].y) == 0
                            && sig(point[1].x - point[i].x) > 0))
                swap(point[1], point[i]);
        }
        for (int i = 2; i <= N; i++)
            point[i] = point[i] - point[1];
        sort(point + 2, point + 1 + N);
        int n = N;
        N = 1;
        const Point zero(0, 0);
        point[1] = zero;
        for (int i = 2; i <= n;)
        {
            Point current = point[i];
            point[++N] = current;
            for (; i <= n && !sig(current.cross(point[i])); i++)
                ;
        }
        top = 0;
        stack[top++] = point[1];
        stack[top++] = point[2];
        for (int i = 3; i <= N; i++)
        {
            while (top >= 2
                    && sig(
                            (stack[top - 1] - stack[top - 2]).cross(
                                    point[i] - stack[top - 2])) < 0)
                top--;
```

```
            stack[top++] = point[i];
        }
    }
    void finish()
    {
        for (int i = 0; i < top; i++)
        {
            stack[i + top] = stack[i];
        }
    }
};

Poly poly;

const Point zero(0, 0);

struct Line
{
    Point a, b;
    int current;

    double cosValue() const
    {
        Point p = poly.stack[current + 1] - poly.stack[current];
        Point q = b - a;
        return p.dot(q) / p.dist(zero) / q.dist(zero);
    }
    void adjust()
    {
        if (sig((b - a).cross(poly.stack[current + 1] -
poly.stack[current]))
                == 0)
            current++;
    }
};

void init()
{
    poly.init();
    for (int i = 1; i <= N; i++)
    {
        Point tmp;
        scanf("%lf%lf", &tmp.x, &tmp.y);
        poly.add(tmp);
    }
    poly.convexHull();
    poly.finish();
}

Line line[4];

double S()
{
    Point point[5];
    for (int i = 0; i < 4; i++)
    {
        int next = (i + 1) % 4;
        Point p1 = poly.stack[line[i].current];
```

```
        Point p2 = p1 + line[i].b - line[i].a;
        Point p3 = poly.stack[line[next].current];
        Point p4 = p3 + line[next].b - line[next].a;
        point[i] = getSol(p1, p2, p3, p4);
    }
    point[4] = point[0];
    double res = 0;
    for (int i = 0; i < 4; i++)
    {
        res += point[i].cross(point[i + 1]);
    }
    res = fabs(res);
    return res;
}

void buildLine()
{
    //bottom ymin
    line[0].a = poly.stack[0];
    for (int i = 0; i < poly.top; i++)
    {
        if (line[0].a.y >= poly.stack[i].y)
        {
            line[0].a = poly.stack[i];
            line[0].current = i;
        }
    }
    line[0].b = line[0].a;
    line[0].b.x += 1;
    //right xmax
    line[1].a = poly.stack[0];
    for (int i = 0; i < poly.top; i++)
    {
        if (line[1].a.x <= poly.stack[i].x)
        {
            line[1].a = poly.stack[i];
            line[1].current = i;
        }
    }
    line[1].b = line[1].a;
    line[1].b.y += 1;
    //up ymax
    line[2].a = poly.stack[0];
    for (int i = 0; i < poly.top; i++)
    {
        if (line[2].a.y <= poly.stack[i].y)
        {
            line[2].a = poly.stack[i];
            line[2].current = i;
        }
    }
    line[2].b = line[2].a;
    line[2].b.x -= 1;
    //left xmin
    line[3].a = poly.stack[0];
    for (int i = 0; i < poly.top; i++)
    {
        if (line[3].a.x >= poly.stack[i].x)
```

```
        {
            line[3].a = poly.stack[i];
            line[3].current = i;
        }
    }
    line[3].b = line[3].a;
    line[3].b.y -= 1;
}

void work()
{
    if (poly.top <= 2)
    {
        puts("0.0000");
        return;
    }
    double result = 1e15;
    buildLine();
    do
    {
        result = min(result, S());
        double arc[4];
        for (int i = 0; i < 4; i++)
        {
            arc[i] = line[i].cosValue();
        }
        double mx = arc[0];
        int idx = 0;
        for (int i = 1; i < 4; i++)
        {
            if (mx < arc[i])
            {
                mx = arc[i];
                idx = i;
            }
        }
        Point vec = poly.stack[line[idx].current + 1]
                - poly.stack[line[idx].current];
        for (int i = idx, j = 0; j < 4; i = (i + 1) % 4, j++)
        {
            line[i].b = line[i].a + vec;
            line[i].adjust();
            vec = vec.trans();
        }
    } while (sig(
            atan2(line[0].b.y - line[0].a.y, line[0].b.x -
line[0].a.x)
                    - PI / 2) <= 0);
    printf("%.4lf\n", result / 2);
}
```

大数开平方

```
import java.math.*;
import java.util.*;

public class Solution {
    String source;
    int pos;

    BigInteger nextPair() {
        BigInteger res;
        if (source.length() % 2 != 0 && pos == 0) {
            res = new BigInteger(source.substring(pos, pos + 1));
            pos++;
        } else {
            res = new BigInteger(source.substring(pos, pos + 2));
            pos += 2;
        }
        return res;
    }

    void init() {
        Scanner scan = new Scanner(System.in);
        source = scan.next();
    }

    void work() {
        pos = 0;
        BigInteger left = BigInteger.ZERO, current = BigInteger.ZERO;
        BigInteger result = BigInteger.ZERO;
        while (true) {
            int last = left.mod(BigInteger.TEN).intValue();
            left = left.subtract(BigInteger.valueOf(last)).add(
                    BigInteger.valueOf(last * 2));
            left = left.multiply(BigInteger.TEN);
            BigInteger r = BigInteger.ZERO;
            for (int i = 0; i <= 9; i++) {
                BigInteger tmp = BigInteger.valueOf(i);
                if (left.add(tmp).multiply(tmp).compareTo(current) <=
0)
                    r = tmp;
                else
                    break;
            }
            left = left.add(r);
            result = result.multiply(BigInteger.TEN);
            result = result.add(r);
            current = current.subtract(left.multiply(r));
            current = current.multiply(BigInteger.TEN.pow(2));
            if (pos == source.length())
                break;
            current = current.add(nextPair());
        }
        System.out.println(result);
    }
}
```

快速傅里叶

```
const double pi = acos(-1);
const complex<double> I(0, 1);
const double eps = 1e-6;
void fft(int n, int sig, complex<double> a[])
{
    for (int j = 1; j < n - 1; j++)
    {
        int i = 0;
        for (int k = 1, tmp = j; k < n;
                i = (i << 1) | (tmp & 1), k <<= 1, tmp >>= 1)
            ;
        if (j < i)
            swap(a[i], a[j]);
    }
    for (int m = 2; m <= n; m <<= 1)
    {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++)
        {
            complex<double> w = exp(sig * i * pi / mh * I);
            for (int j = i; j < n; j += m)
            {
                int k = j + mh;
                complex<double> u = a[j];
                a[j] = u + a[k] * w;
                a[k] = u - a[k] * w;
            }
        }
    }
}
complex<double> a[200000], b[200000];
char a1[100010], a2[100010];
int ans[200010];
int main()
{
    while (scanf("%s%s", a1, a2) != EOF)
    {
        int l1 = strlen(a1);
        int l2 = strlen(a2);
        int l = 1;
        while (1)
        {
            if (l >= l1 && l >= l2)
                break;
            l <<= 1;
        }
        l <<= 1;
        for (int i = 0; i < l1; ++i)
            a[i] = complex<double>(a1[l1 - 1 - i] - '0', 0);
        for (int i = 0; i < l2; ++i)
            b[i] = complex<double>(a2[l2 - 1 - i] - '0', 0);
        for (int i = l1; i < l; ++i)
            a[i] = complex<double>(0, 0);
        for (int i = l2; i < l; ++i)
            b[i] = complex<double>(0, 0);
        fft(l, 1, a);
```

```
        fft(l, 1, b);
        for (int i = 0; i < l; ++i)
            a[i] *= b[i];
        fft(l, -1, a);
        int k = 0, tmp = 0;
        ans[0] = 0;
        for (int i = 0; i < l; ++i)
        {
            tmp = (int) (a[i].real() / l + eps);
            ans[i] += tmp;
            if (ans[i])
                k = i;
            ans[i + 1] = ans[i] / 10;
            ans[i] %= 10;
        }
        for (int i = k; i >= 0; --i)
            printf("%d", ans[i]);
        printf("\n");
    }
    return 0;
}
```

矩阵类

```
struct Matrix
{
    //init
    //toStage
    //solve, det...
    double mat[maxn][maxn];
    double ext[maxn];
    int m, n;
    int change;

    Matrix()
    {
    }
    Matrix(int p, int q) :
           m(p), n(q)
    {
    }

    void init()
    {
        change = 0;
    }

    void init(double e[])
    {
        change = 0;
        memcpy(ext, e, sizeof(ext));
    }

    Matrix operator+(const Matrix &p) const
    {
        Matrix res(m, n);
        for (int i = 1; i <= m; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                res.mat[i][j] = mat[i][j] + p.mat[i][j];
            }
        }
        return res;
    }
    Matrix operator*(const Matrix& p) const
    {
        Matrix res(m, p.n);
        for (int i = 1; i <= m; i++)
        {
            for (int j = 1; j <= p.n; j++)
            {
                res.mat[i][j] = 0;
                for (int k = 1; k <= n; k++)
                {
                    res.mat[i][j] = mat[i][k] * p.mat[k][j];
                }
            }
        }
        return res;
```

```
}
void swapLine(int p, int q)
{
    double tmp[maxn];
    memcpy(tmp, mat[p], sizeof(tmp));
    memcpy(mat[p], mat[q], sizeof(tmp));
    memcpy(mat[q], tmp, sizeof(tmp));
    swap(ext[p], ext[q]);
    if (p != q)
        change++;
}
int toStage()
{
    int current = 1;
    for (int i = 1; i <= n; i++)
    {
        int idx = current;
        for (int j = current; j <= m; j++)
        {
            if (fabs(mat[j][i]) - fabs(mat[idx][i]) > 0)
                idx = j;
        }
        int j = idx;
        if (mat[j][i])
        {
            swapLine(current, j);
            for (int k = current + 1; k <= m; k++)
            {
                if (!mat[k][i])
                    continue;
                double fac = mat[k][i] / mat[current][i];
                for (int l = 1; l <= n; l++)
                {
                    mat[k][l] -= fac * mat[current][l];
                }
                ext[k] -= fac * ext[current];
            }
            current++;
        }
    }
    int rank = 0;
    for (int i = 1; i <= m; i++)
    {
        int flag = 1;
        for (int j = 1; j <= n; j++)
        {
            if (sig(mat[i][j]))
            {
                flag = 0;
                break;
            }
        }
        if (!flag)
            rank++;
    }
    return rank;
}
int solve(double result[])
```

```
    {
        int rank = 0;
        for (int i = 1; i <= m; i++)
        {
            int flag = 1;
            for (int j = 1; j <= n; j++)
            {
                if (sig(mat[i][j]))
                {
                    flag = 0;
                    break;
                }
            }
            if (flag && sig(ext[i]))
            {
                return -1;
            }
            if (!flag)
                rank++;
        }
        if (rank < n)
            return 0;
        for (int i = n; i >= 1; i--)
        {
            double s = 0;
            for (int j = i + 1; j <= n; j++)
            {
                s += result[j] * mat[i][j];
            }
            s = ext[i] - s;
            result[i] = s / mat[i][i];
        }
        return 1;
    }
    double det()
    {
        double res = change % 2 == 0 ? 1 : -1;
        for (int i = 1; i <= n; i++)
            res *= mat[i][i];
        return res;
    }
};
```

模组合数

```
LL fastPow(LL a, LL n, LL m)
{
    if (n == 0)
        return 1;
    if (n == 1)
        return a % m;
    if (n == 2)
        return a % m * a % m;
    if (n & 1)
        return fastPow(a, n - 1, m) * a % m;
    return fastPow(fastPow(a, n / 2, m), 2, m);
}

LL c(LL n, LL k, LL m)
{
    if (n < k)
        return 0;
    if (k > n - k)
        k = n - k;
    LL res = 1;
    for (int i = 1; i <= k; i++)
    {
        res = res * (n - i + 1) % m;
        res = res * fastPow(i, m - 2, m) % m;
    }
    return res;
}

LL lucas(LL n, LL k, LL m)
{
    LL r = 1;
    while (n && k && r)
    {
        r = r * c(n % m, k % m, m) % m;
        n /= m;
        k /= m;
    }
    return r;
}
```

Rho-Pollard

```
#include <algorithm>

using namespace std;

typedef long long LL;

const int TIME = 11;
LL factor[1000000];
int fac_top;

LL abs(LL n)
{
    if (n < 0)
        return -n;
    return n;
}

LL gcd(LL small, LL big)
{
    while (small)
    {
        swap(small, big);
        small %= big;
    }
    return abs(big);
}

//ret = (a*b)%n (n<2^62)
LL muti_mod(LL a, LL b, LL n)
{
    LL exp = a % n, res = 0;
    while (b)
    {
        if (b & 1)
        {
            res += exp;
            if (res > n)
                res -= n;
        }
        exp <<= 1;
        if (exp > n)
            exp -= n;

        b >>= 1;
    }
    return res;
}

// ret = (a^b)%n
LL mod_exp(LL a, LL p, LL m)
{
    LL exp = a % m, res = 1; //
    while (p > 1)
    {
        if (p & 1) //
            res = muti_mod(res, exp, m);
```

```
        exp = muti_mod(exp, exp, m);
        p >>= 1;
    }
    return muti_mod(res, exp, m);
}

bool miller_rabin(LL n, int times)
{
    if (n == 2)
        return 1;
    if (n < 2 || !(n & 1))
        return 0;

    LL a, u = n - 1, x, y;
    int t = 0;
    while (u % 2 == 0)
    {
        t++;
        u /= 2;
    }

    srand(time(0));
    for (int i = 0; i < times; i++)
    {
        a = rand() % (n - 1) + 1;
        x = mod_exp(a, u, n);
        for (int j = 0; j < t; j++)
        {
            y = muti_mod(x, x, n);
            if (y == 1 && x != 1 && x != n - 1)
                return false; //must not
            x = y;
        }
        if (y != 1)
            return false;
    }
    return true;
}

LL pollard_rho(LL n, int c)
{
    LL x, y, d, i = 1, k = 2;
    srand(time(0));

    x = rand() % (n - 1) + 1;
    y = x;
    while (true)
    {
        i++;
        x = (muti_mod(x, x, n) + c) % n;
        d = gcd(y - x, n);
        if (1 < d && d < n)
            return d;
        if (y == x)
            return n;
        if (i == k)
        {
            y = x;
```

```
            k <<= 1;
        }
    }
}

void findFactor(LL n, int k)
{
    if (n == 1)
        return;

    if (miller_rabin(n, TIME))
    {
        factor[++fac_top] = n;
        return;
    }
    LL p = n;
    while (p >= n)
        p = pollard_rho(p, k--);
    findFactor(p, k);
    findFactor(n / p, k);
}

LL N, K;

void init()
{
    fac_top = 0;
    scanf("%I64u%I64u", &N, &K);
    if (K != 1)
        findFactor(K, 107);
    sort(factor + 1, factor + 1 + fac_top);
    int c = fac_top;
    fac_top = 0;
    for (int i = 1; i <= c;)
    {
        LL current = factor[i];
        factor[++fac_top] = current;
        for (; i <= c && current == factor[i]; i++)
            ;
    }
}
```

生成树计数_基尔霍夫矩阵-树定理

```
LL det(int N)
{
    LL ans = 1;
    for (LL i = 1; i <= N; i++)
    {
        for (LL j = i + 1; j <= N; j++)
        {
            while (mat[i][j])
            {
                LL t = mat[i][i] / mat[i][j];
                for (int k = 1; k <= N; k++)
                    mat[k][i] -= t * mat[k][j];
                for (LL k = 1; k <= N; k++)
                {
                    LL tmp = mat[k][i];
                    mat[k][i] = mat[k][j];
                    mat[k][j] = tmp;
                }
                ans = -ans;
            }
            if (mat[i][i] == 0)
                return 0;
        }
    }
    for (int i = 1; i <= N; i++)
        ans *= mat[i][i];
    return ans;
}

void work()
{
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            mat[i][j] = -1;
        }
        mat[i][i] = N - 1;
    }
    for (int i = 1; i <= M; i++)
    {
        int p, q;
        scanf("%d%d", &p, &q);
        if (mat[p][q] == 0)
            continue;
        mat[p][q] = 0;
        mat[q][p] = 0;
        mat[p][p]--;
        mat[q][q]--;
    }
    cout << det(N - 1) << endl;
}
```

O(N)最长回文子串

```
struct SString
{
    int F[100005];
    int Palindrome(char *s)
    {
        int ans = 0;
        int n = strlen(s);
        s--;
        for (int i = 0; i <= n; i++)
            F[i] = 0;
        // odd Palindrome
        int Max = 0;
        int Maxi = 0;
        for (int i = 1; i <= n; i++)
        {
            if (i > Max)
            {
                int k = 0;
                while (i - k >= 1 && i + k <= n && s[i - k] == s[i +
k])
                    k++;
                F[i] = k;
            }
            else
            {
                int p = Maxi - (i - Maxi);
                int k = F[p];
                if (i + k - 1 > Max)
                    k = Max - i + 1;
                while (i - k >= 1 && i + k <= n && s[i - k] == s[i +
k])
                    k++;
                F[i] = k;
            }
            if (i + F[i] - 1 > Max)
            {
                Max = i + F[i] - 1;
                Maxi = i;
            }
            ans = max(ans, F[i] * 2 - 1);
        }
        //even Palindrome
        for (int i = 0; i <= n; i++)
            F[i] = 0;
        Max = 0;
        Maxi = 0;
        for (int i = 2; i <= n; i++)
        {
            if (i > Max)
            {
                int k = 1;
                while (i - k >= 1 && i + k - 1 <= n && s[i - k] == s[i
+ k - 1])
                    k++;
                F[i] = k - 1;
            }
```

```
            else
            {
                int p = Maxi - (i - Maxi);
                int k = F[p];
                if (i + k - 1 > Max)
                    k = Max - i + 1;
                while (i - k >= 1 && i + k - 1 <= n && s[i - k] == s[i
+ k - 1])
                    k++;
                F[i] = k - 1;
            }
            if (F[i] + i - 1 > Max)
            {
                Max = F[i] + i - 1;
                Maxi = i;
            }
            ans = max(ans, F[i] * 2);
        }
        return ans;
    }
} String;
```

栈扫描求不短于K公共子串对数

```
int A[maxn], B[maxn], C[maxn], D[maxn], *sa = D + 1, *rank, *height;

long long left[maxn], right[maxn];

void sortAndRank(int *a1, int *a2, int n, int &m, int j)
{
    int i;
    memset(C, 0, sizeof(C));
    for (i = 0; i < n; i++)
        C[a1[i]]++;
    for (i = 1; i <= m; i++)
        C[i] += C[i - 1];
    for (i = n - 1; i >= 0; i--)
        sa[--C[a1[a2[i]]]] = a2[i];
    a2[sa[0]] = m = 0;
    for (i = 1; i < n; i++)
        a2[sa[i]] =
               a1[sa[i - 1]] == a1[sa[i]]
                      && a1[sa[i - 1] + j] == a1[sa[i] + j] ? m : ++m;
}

void da(char *str, int n, int m)
{
    int *a1 = A, *a2 = B, *tmp;
    int i, j, p;
    for (i = 0; i < n; i++)
    {
        a1[i] = i;
        a2[i] = str[i];
    }
    a1[n] = a2[n] = -1;
    sortAndRank(a2, a1, n, m, 0);
    for (j = 1; m < n - 1; j <<= 1)
    {
        p = 0;
        for (i = n - j; i < n; i++)
            a2[p++] = i;
        for (i = 0; i < n; i++)
            if (sa[i] >= j)
                a2[p++] = sa[i] - j;
        sortAndRank(a1, a2, n, m, j);
        tmp = a1;
        a1 = a2;
        a2 = tmp;
    }
    rank = a1;
    height = a2;
}

void calHeight(char *str, int n)
{
    int i, j, k;
    sa[-1] = n;
    for (height[0] = k = i = 0; i < n; i++)
    {
        for (k ? k-- : 0, j = sa[rank[i] - 1]; str[i + k] == str[j +
```

```
k]; k++)
            ;
        height[rank[i]] = k;
    }
}

char tmp[maxn];

char source[maxn];

int Len, K;
int La;

void init()
{
    Len = 0;
    La = 0;
    scanf("%s", tmp);
    for (int i = 0; tmp[i]; i++)
    {
        source[Len++] = tmp[i];
        La++;
    }
    source[Len++] = 1;
    scanf("%s", tmp);
    for (int i = 0; tmp[i]; i++)
        source[Len++] = tmp[i];
    source[Len] = 0;
    da(source, Len, 300);
    calHeight(source, Len);
    for (int i = 0; i < Len; i++)
    {
        height[i] -= K - 1;
        if (height[i] < 0)
            height[i] = 0;
    }
}

int stack[maxn];

int sumB[maxn], sumA[maxn];

void buildSum()
{
    if (sa[0] > La)
        sumB[0] = 1;
    else
        sumB[0] = 0;
    for (int i = 1; i < Len; i++)
    {
        sumB[i] = sumB[i - 1] + (sa[i] > La);
    }
}

long long getSum(int from, int to)
{
    if (from > to)
        return 0;
```

```
    from--;
    if (from < 0)
        return sumB[to];
    return sumB[to] - sumB[from];
}

void print()
{
    source[La] = '|';
    for (int i = 0; i < Len; i++)
    {
        puts(source + sa[i]);
    }
    source[La] = 1;
}

void buildLeft()
{
    int top = 0;
    for (int i = 0; i < Len; i++)
    {
        for (int j = top - 1; j >= 0; j--)
        {
            if (height[stack[j]] >= height[i])
                top--;
            else
                break;
        }
        if (top)
            left[i] = left[stack[top - 1]]
                    + (height[i]) * getSum(stack[top - 1], i - 1);
        else
            left[i] = (height[i]) * getSum(0, i - 1);
        stack[top++] = i;
    }
}

void buildRight()
{
    int top = 0;
    right[Len - 1] = 0;
    right[Len] = 0;
    for (int i = Len - 1; i >= 0; i--)
    {
        for (int j = top - 1; j >= 0; j--)
        {
            if (height[stack[j]] >= height[i])
                top--;
            else
                break;
        }
        if (top)
        {
            right[i] = right[stack[top - 1]]
                    + (height[i]) * getSum(i, stack[top - 1] - 1);
        }
        else
            right[i] = height[i] * getSum(i, Len - 1);
```

```
        stack[top++] = i;
    }
}

void work()
{
    buildSum();
    buildLeft();
    buildRight();
    long long result = 0;
    right[Len] = 0;
    for (int i = 0; i < Len; i++)
    {
        if (sa[i] < La)
        {
            result += left[i] + right[i + 1];
        }
    }
    printf("%I64d\n", result);
}
```

栈扫描求不同回文子串数

```
template<class T>
struct RMQ
{
    int N;
    T val[maxn];
    int rmq[20][maxn];
    int qry[maxn];

    void init(T arr[], int n)
    {
        N = n;
        for (int i = 1; i <= N; i++)
            val[i] = arr[i];
        for (int i = 1, cnt = 0; i <= N; i <<= 1, cnt++)
        {
            for (int j = i; j < (i << 1) && j <= N; j++)
            {
                qry[j] = cnt;
            }
        }
        val[0] = 0;
        build();
    }
    void build()
    {
        for (int i = 1; i <= N; i++)
            rmq[0][i] = i;
        for (int j = 1; (1 << j) <= N; j++)
        {
            for (int i = 1; i <= N; i++)
            {
                int p = rmq[j - 1][i];
                int q = 0;
                if (i + (1 << (j - 1)) <= N)
                    q = rmq[j - 1][i + (1 << (j - 1))];
                if (val[p] < val[q])
                {
                    rmq[j][i] = p;
                }
                else
                    rmq[j][i] = q;
            }
        }
    }
    int query(int p, int q)
    {
        if (p > q)
            return 0;
        int k = qry[q - p + 1];
        int i = p;
        int j = q - (1 << k) + 1;
        if (val[rmq[k][i]] < val[rmq[k][j]])
            return rmq[k][i];
        else
            return rmq[k][j];
    }
```

```
};

int A[maxn], B[maxn], C[maxn], D[maxn], *sa = D + 1, *rank, *height;

void sortAndRank(int *a1, int *a2, int n, int &m, int j)
{
    int i;
    for (i = 0; i <= m; i++)
        C[i] = 0;
    for (i = 0; i < n; i++)
        C[a1[i]]++;
    for (i = 1; i <= m; i++)
        C[i] += C[i - 1];
    for (i = n - 1; i >= 0; i--)
        sa[--C[a1[a2[i]]]] = a2[i];
    a2[sa[0]] = m = 0;
    for (i = 1; i < n; i++)
        a2[sa[i]] =
               a1[sa[i - 1]] == a1[sa[i]]
                      && a1[sa[i - 1] + j] == a1[sa[i] + j] ? m : ++m;
}

void da(char *str, int n, int m)
{
    int *a1 = A, *a2 = B, *tmp;
    int i, j, p;
    for (i = 0; i < n; i++)
    {
        a1[i] = i;
        a2[i] = str[i];
    }
    a1[n] = a2[n] = -1;
    sortAndRank(a2, a1, n, m, 0);
    for (j = 1; m < n - 1; j <<= 1)
    {
        p = 0;
        for (i = n - j; i < n; i++)
            a2[p++] = i;
        for (i = 0; i < n; i++)
            if (sa[i] >= j)
                a2[p++] = sa[i] - j;
        sortAndRank(a1, a2, n, m, j);
        tmp = a1;
        a1 = a2;
        a2 = tmp;
    }
    rank = a1;
    height = a2;
}

void calHeight(char *str, int n)
{
    int i, j, k;
    sa[-1] = n;
    for (height[0] = k = i = 0; i < n; i++)
    {
        for (k ? k-- : 0, j = sa[rank[i] - 1]; str[i + k] == str[j +
k]; k++)
```

```
            ;
        height[rank[i]] = k;
    }
}

RMQ<int> maxQ, lcpQ;

int getLCP(int p, int q)
{
    if (p > q)
        swap(p, q);
    return lcpQ.val[lcpQ.query(p + 1, q)];
}

char str[maxn], source[maxn];
int Len, N;

int odd[maxn], even[maxn];
int oddTmp[maxn], evenTmp[maxn];

int hash[maxn];

void init()
{
    scanf("%s", source);
    N = strlen(source);
    str[N] = 1;
    for (int i = 0; i < N; i++)
    {
        str[i] = source[i];
        str[i + N + 1] = source[N - 1 - i];
        hash[N - 1 - i] = i + N + 1;
    }
    Len = 2 * N + 1;
    str[Len] = 0;
    da(str, Len, 150);
    calHeight(str, Len);
    lcpQ.init(height, Len);
    for (int i = 0; i < N; i++)
        oddTmp[i + 1] = -getLCP(rank[i], rank[hash[i]]);
    even[0] = 0;
    for (int i = 1; i < N; i++)
        evenTmp[i + 1] = -getLCP(rank[i], rank[hash[i - 1]]);
    da(source, N, 150);
    calHeight(source, N);
    for (int i = 0; i < N; i++)
        odd[rank[i] + 1] = oddTmp[i + 1];
    for (int i = 0; i < N; i++)
        even[rank[i] + 1] = evenTmp[i + 1];
}

void print()
{
    for (int i = 0; i < N; i++)
    {
        puts(source + sa[i]);
    }
}
```

```
int mx[maxn];
int stack[maxn];

long long solveOdd()
{
    long long result = 0;
    maxQ.init(odd, N);
    int top = 0;
    for (int i = 0; i < N; i++)
    {
        for (int j = top - 1; j >= 0; j--)
        {
            if (height[stack[j]] >= height[i])
                top--;
            else
                break;
        }
        if (top)
        {
            int p = (mx[stack[top - 1]]);
            int q = min(height[i],
                    -maxQ.val[maxQ.query(stack[top - 1] + 1, i)]);
            mx[i] = max(p, q);
        }
        else
            mx[i] = min(-maxQ.val[maxQ.query(stack[top - 1] + 1, i)],
                    height[i]);
        stack[top++] = i;
        result += max(-odd[i + 1] - mx[i], 0);
    }
    return result;
}

long long solveEven()
{
    long long result = 0;
    maxQ.init(even, N);
    int top = 0;
    mx[0] = 0;
    for (int i = 0; i < N; i++)
    {
        for (int j = top - 1; j >= 0; j--)
        {
            if (height[stack[j]] >= height[i])
                top--;
            else
                break;
        }
        if (top)
        {
            int p = mx[stack[top - 1]];
            int q = min(height[i],
                    -maxQ.val[maxQ.query(stack[top - 1] + 1, i)]);
            mx[i] = max(p, q);
        }
        else
            mx[i] = min(-maxQ.val[maxQ.query(stack[top - 1] + 1, i)],
```

```
                height[i]);
        stack[top++] = i;
        result += max(0, -even[i + 1] - mx[i]);
    }
    return result;
}

void work()
{
    long long result = solveOdd() + solveEven();
    printf("%I64d\n", result);
}
```

自顶向下伸展树

```
template<class T>
struct SplayNode
{
    SplayNode<T> *L, *R, *P;
    int size;
    T key;
    bool rev;
};

#define keyTree (root->R->L)
template<class T>
struct SplayTree
{
    int C, Top, count;
    SplayNode<T> *root, *null, nodes[maxn];
    int stack[maxn];

    SplayNode<T>* newNode(const T& c)
    {
        count++;
        int x;
        if (Top)
            x = stack[--Top];
        else
            x = ++C;
        SplayNode<T> *res = nodes + x;
        res->L = res->R = res->P = null;
        res->size = 1;
        res->key = c;
        res->rev = 0;
        res->hval = res->rhval = c;
        return res;
    }

    void push_up(SplayNode<T>* x)
    {
        if (x == null)
            return;
        push_down(x->L);
        push_down(x->R);
        x->size = x->L->size + x->R->size + 1;
    }
    void push_down(SplayNode<T> *x)
    {
        if (x == null)
            return;
        if (x->rev)
        {
            x->L->rev ^= 1;
            x->R->rev ^= 1;
            swap(x->L, x->R);

            x->rev = 0;
        }
    }
```

```
void init(int l, int r, T value[])
{
    C = Top = count = 0;
    null = &nodes[++C];
    root = null;
    root = newNode(value[0]);
    root->R = newNode(value[0]);

    makeTree(keyTree, l, r, value);

    push_up(root->R);
    push_up(root);
}
void makeTree(SplayNode<T>* &x, int l, int r, T value[])
{
    if (l > r)
        return;
    int m = (l + r) >> 1;
    x = newNode(value[m]);
    makeTree(x->L, l, m - 1, value);
    makeTree(x->R, m + 1, r, value);
    push_up(x);
}
void rightRotate(SplayNode<T>* &x)
{
    SplayNode<T> *y = x->L;
    x->L = y->R;
    y->R = x;
    push_up(x);
    x = y;
}
void leftRotate(SplayNode<T>* &x)
{
    SplayNode<T> *y = x->R;
    x->R = y->L;
    y->L = x;
    push_up(x);
    x = y;
}
void leftLink(SplayNode<T>* &t, SplayNode<T>* &l)
{
    SplayNode<T> *tmp = t;
    t = t->R;
    tmp->R = l;
    l = tmp;
}
void rightLink(SplayNode<T>* &t, SplayNode<T>* &r)
{
    SplayNode<T> *tmp = t;
    t = t->L;
    tmp->L = r;
    r = tmp;
}
void leftFinish(SplayNode<T> *l, SplayNode<T> *p)
{
    while (1)
    {
        SplayNode<T> *tmp = l;
```

```
        l = l->R;
        tmp->R = p;
        push_up(tmp);
        p = tmp;
        if (tmp == null)
            break;
    }
}
void rightFinish(SplayNode<T> *l, SplayNode<T> *p)
{
    while (1)
    {
        SplayNode<T> *tmp = l;
        l = l->L;
        tmp->L = p;
        push_up(tmp);
        p = tmp;
        if (tmp == null)
            break;
    }
}
void splay(SplayNode<T>* &t, int k)
{
    SplayNode<T> *l = null, *r = null;
    null->L = null->R = null;
    push_down(t);
    while (k != t->L->size + 1)
    {
        push_down(t->L);
        if (k <= t->L->size)
        {
            push_down(t->L->L);
            if (k == t->L->L->size + 1)
                rightLink(t, r);
            else if (k <= t->L->L->size)
            {
                rightRotate(t);
                rightLink(t, r);
            }
            else
            {
                k -= t->L->L->size + 1;
                rightLink(t, r);
                leftLink(t, l);
            }
        }
        else
        {
            push_down(t->R);
            push_down(t->R->L);
            k -= t->L->size + 1;
            if (k == t->R->L->size + 1)
                leftLink(t, l);
            else if (k > t->R->L->size + 1)
            {
                k -= t->R->L->size + 1;
                leftRotate(t);
                leftLink(t, l);
```

```
            }
            else
            {
                leftLink(t, l);
                rightLink(t, r);
            }
        }
        push_down(t);
    }
    push_down(t);
    leftFinish(l, t->L);
    rightFinish(r, t->R);
    t->L = null->R;
    t->R = null->L;
    push_up(t);
}

void visit(SplayNode<T> *root)
{
    if (root != null)
    {
        push_down(root);
        visit(root->L);
        visit(root->R);
    }
}

void flip(int a, int b)
{
    splay(root, a);
    splay(root->R, b - a + 2);

    SplayNode<T> *idx = keyTree;
    idx->rev ^= 1;
    push_down(idx);
    push_up(root->R);
    push_up(root);
}
void modify(int p, int c)
{
    splay(root, p);
    splay(root->R, 2);

    keyTree->key = c;
    keyTree->hval = keyTree->rhval = c;
    push_up(root->R);
    push_up(root);
}
void get()
{
    splay(root, 1);
    splay(root->R, count - 1);
    visit(keyTree);
}

LL getHash(int from, int to)
{
    splay(root, from);
```

```
        splay(root->R, to - from + 2);
        push_down(keyTree);
        return keyTree->hval;
    }
};
```

动态树高效版

```
template<class T>
struct SplayNode
{
    SplayNode<T> *L, *R, *P;
    T key;
    int size;
    T add;
    T mx;
    bool rev;
};

template<class T>
struct SplayTree
{
    int C, Top, count;
    SplayNode<T> *root, *null, nodes[maxn], *stack[maxn];

    SplayNode<T>* newNode(const T& c)
    {
        count++;
        SplayNode<T> *res;
        if (Top)
            res = stack[--Top];
        else
            res = &nodes[++C];
        res->size = 1;
        res->L = res->R = res->P = null;
        res->add = 0;
        res->rev = 0;
        res->key = res->mx = c;
        return res;
    }
    void push_up(SplayNode<T>* x)
    {
        if (x == null)
            return;
        push_down(x->L);
        push_down(x->R);
        x->size = x->L->size + x->R->size + 1;
        x->mx = max(x->key, max(x->L->mx, x->R->mx));
    }
    void push_down(SplayNode<T>* x)
    {
        if (x == null)
            return;
        if (x->add)
        {
            x->L->add += x->add;
            x->R->add += x->add;
            x->mx += x->add;
            x->key += x->add;

            x->add = 0;
        }
        if (x->rev)
        {
```

```
        x->L->rev ^= 1;
        x->R->rev ^= 1;
        swap(x->L, x->R);

        x->rev = 0;
    }
}
void init(int N)
{
    C = count = Top = 0;
    null = &nodes[++C];
    null->L = null->R = null->P = null;
    null->mx = -INF;
    for (int i = 1; i <= N; i++)
        newNode(0);
}
void leftRotate(SplayNode<T>* y)
{
    SplayNode<T>* x = y->R, *z = y->P;
    push_down(y);
    push_down(x);
    if (z != null)
    {
        if (z->L == y)
            z->L = x;
        else if (z->R == y)
            z->R = x;
    }
    y->R = x->L;
    x->L = y;
    x->P = z;
    y->P = x;
    if (y->R != null)
        y->R->P = y;
    push_up(y);
}
void rightRotate(SplayNode<T>* y)
{
    SplayNode<T>* x = y->L, *z = y->P;
    push_down(y);
    push_down(x);
    if (z != null)
    {
        if (z->R == y)
            z->R = x;
        else if (z->L == y)
            z->L = x;
    }
    y->L = x->R;
    x->R = y;
    x->P = z;
    y->P = x;
    if (y->L != null)
        y->L->P = y;
    push_up(y);
}
void splay(SplayNode<T>* x)
{
```

```
    if (x == null)
        return;
    null->L = null->R = null->P = null;
    push_down(x);
    while (x->P->L == x || x->P->R == x)
    {
        if (x == x->P->L)
        {
            if (x->P->P->L != x && x->P->P->R != x)
                rightRotate(x->P);
            else if (x->P == x->P->P->L)
            {
                rightRotate(x->P->P);
                rightRotate(x->P);
            }
            else if (x->P == x->P->P->R)
            {
                rightRotate(x->P);
                leftRotate(x->P);
            }
        }
        else
        {
            if (x->P->P->L != x && x->P->P->R != x)
                leftRotate(x->P);
            else if (x->P == x->P->P->R)
            {
                leftRotate(x->P->P);
                leftRotate(x->P);
            }
            else if (x->P == x->P->P->L)
            {
                leftRotate(x->P);
                rightRotate(x->P);
            }
        }
    }
    push_up(x);
}
void access0(SplayNode<T> *v)
{
    if (v->P != null)
        access0(v->P);
    push_down(v);
}
void access(SplayNode<T> *x)
{
    access0(x);
    for (SplayNode<T> *v = null, *u = x; u != null; u = u->P)
    {
        splay(u);
        u->L = v;
        v->P = u;
        push_up(v = u);
    }
    splay(x);
}
SplayNode<T>* findRoot(SplayNode<T> *x)
```

```
{
    access(x);
    while (x->R != null)
        x = x->R;
    return x;
}
void updateLCA(SplayNode<T> *x, SplayNode<T> *y, T w)
{
    access(x);
    for (SplayNode<T> *v = null, *u = y; u != null; u = u->P)
    {
        splay(u);
        if (u->P == null) //u is LCA
        {
            u->key += w;
            u->L->add += w;
            v->add += w;
        }
        u->L = v;
        v->P = u;
        push_up(v = u);
    }
}
T queryLCA(SplayNode<T> *x, SplayNode<T> *y)
{
    T res;
    access(x);
    for (SplayNode<T> *v = null, *u = y; u != null; u = u->P)
    {
        splay(u);
        if (u->P == null)
        {
            push_down(u->L);
            push_down(v);
            res = max(u->key, max(u->L->mx, v->mx));
        }
        u->L = v;
        v->P = u;
        push_up(v = u);
    }
    return res;
}
void cut(SplayNode<T> *x)
{
    access(x);
    SplayNode<T> *r = x->R;
    r->P = null;
    x->R = null;
    push_up(x);
}
void link(SplayNode<T> *son, SplayNode<T> *parent)
{
    access(son);
    access(parent);
    son->P = parent;
}
void changeRoot(SplayNode<T> *x)
{
```

```
        access(x);
        x->L = null;
        push_up(x);
        x->rev ^= 1;
        push_down(x);
    }
};
```

二维RMQ

```
template<class T>
struct RMQ2D
{
    int N, M; //N*M
    T val[maxn][maxn];
    int qry[maxn];
    T rmq[9][9][maxn][maxn];

    void init(T arr[maxn][maxn], int n, int m)
    {
        N = n;
        M = m;
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= M; j++)
            {
                val[i][j] = arr[i][j];
            }
        }
        int mx = max(M, N);
        for (int i = 1, cnt = 0; i <= mx; i <<= 1, cnt++)
        {
            for (int j = i; j < (i << 1) && j <= mx; j++)
            {
                qry[j] = cnt;
            }
        }
        for (int i = 1; i <= M; i++)
            val[0][i] = INF;
        for (int i = 1; i <= N; i++)
            val[i][0] = INF;
        build();
    }
    void build()
    {
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= M; j++)
            {
                rmq[0][0][i][j] = val[i][j];
            }
        }
        for (int i = 1; (1 << i) <= N; i++)
        {
            for (int j = 1; j <= M; j++)
            {
                for (int k = 1; k <= N; k++)
                {
                    rmq[i][0][k][j] = rmq[i - 1][0][k][j];
                    if (k + (1 << (i - 1)) <= N)
                        rmq[i][0][k][j] = min(rmq[i][0][k][j],
                                rmq[i - 1][0][k + (1 << (i - 1))][j]);
                }
            }
        }
        for (int i = 1; (1 << i) <= M; i++)
```

```
        {
            for (int j = 1; j <= N; j++)
            {
                for (int k = 1; k <= M; k++)
                {
                    rmq[0][i][j][k] = rmq[0][i - 1][j][k];
                    if (k + (1 << (i - 1)) <= M)
                        rmq[0][i][j][k] = min(rmq[0][i][j][k],
                               rmq[0][i - 1][j][k + (1 << (i - 1))]);
                }
            }
        }
        for (int i = 1; (1 << i) <= N; i++)
        {
            for (int j = 1; (1 << j) <= M; j++)
            {
                for (int k = 1; k <= N; k++)
                {
                    for (int l = 1; l <= M; l++)
                    {
                        rmq[i][j][k][l] = rmq[i - 1][j - 1][k][l];
                        int x = k + (1 << (i - 1));
                        int y = l + (1 << (j - 1));
                        if (x <= N)
                            rmq[i][j][k][l] = min(rmq[i][j][k][l],
                                   rmq[i - 1][j - 1][x][l]);
                        if (y <= M)
                            rmq[i][j][k][l] = min(rmq[i][j][k][l],
                                   rmq[i - 1][j - 1][k][y]);
                        if (x <= N && y <= M)
                            rmq[i][j][k][l] = min(rmq[i][j][k][l],
                                   rmq[i - 1][j - 1][x][y]);
                    }
                }
            }
        }
    }
    T query(int x1, int y1, int x2, int y2)
    {
        if (x1 > x2)
            swap(x1, x2);
        if (y1 > y2)
            swap(y1, y2);
        int p = qry[x2 - x1 + 1];
        int q = qry[y2 - y1 + 1];
        int x = x2 - (1 << p) + 1;
        int y = y2 - (1 << q) + 1;
        return min(min(rmq[p][q][x1][y1], rmq[p][q][x][y]),
               min(rmq[p][q][x1][y], rmq[p][q][x][y1]));
    }
};

RMQ2D<int> rmq;

int N;
int mat[maxn][maxn];

void init()
```

```
{
    scanf("%d", &N);
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            scanf("%d", mat[i] + j);
        }
    }
    rmq.init(mat, N, N);
}

void work()
{
    int M;
    scanf("%d", &M);
    for (int i = 1; i <= M; i++)
    {
        int x1, y1, x2, y2;
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        printf("%d\n", rmq.query(x1, y1, x2, y2));
    }
}
```

恰好覆盖K次矩形面积并

```
struct Rect
{
    double x1, y1;
    double x2, y2;
};

struct Seg
{
    int x, y, z;
    int value;

    bool operator<(const Seg& p) const
    {
        return x < p.x;
    }
};

double val[50010];
int C;

int N, K;

struct Node
{
    int left, right;
    double sum[10010];
    int count[210];
    int cover;

    void init()
    {
        memset(count, 0, sizeof(count));
        memset(sum, 0, sizeof(sum));
        sum[0] = val[right + 1] - val[left];
        cover = 0;
    }

    void update(int from, int to, int v)
    {
        from -= left;
        to -= left;
        for (int i = from; i <= to;)
        {
            int current = count[i];
            int begin = i;
            for (; i <= to && current == count[i]; i++)
                count[i] += v;
            sum[current] -= (val[i + left] - val[begin + left]);
            sum[current + v] += (val[i + left] - val[begin + left]);
        }
    }

    double query()
    {
        int t = K - cover;
        if (t < 0)
```

```
            return 0.0;
        else
        {
            return sum[t];
        }
    }
};

Node node[210];

Rect rect[10010];
Seg seg[50010];
int S;

int L, M;

int BS(double key)
{
    int low = 1, high = C;
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (val[mid] == key)
            return mid;
        else if (val[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

void init()
{
    C = 0;
    S = 0;

    int n = N;
    N = 0;

    for (int i = 1; i <= n; i++)
    {
        double x, y, z, l;
        scanf("%lf%lf%lf%lf", &x, &y, &z, &l);
        if (sig(l) == 0)
            continue;
        if (sig(2 * z - l) > 0)
            continue;
        ++N;
        l *= 0.5;
        rect[N].x1 = x - l;
        rect[N].y1 = y - l;
        rect[N].x2 = x + l;
        rect[N].y2 = y + l;

        val[++C] = rect[N].x1;
        val[++C] = rect[N].y1;
        val[++C] = rect[N].x2;
```

```
        val[++C] = rect[N].y2;
    }

    sort(val + 1, val + 1 + C);
    int c = C;
    C = 0;
    for (int i = 1; i <= c;)
    {
        double current = val[i];
        val[++C] = current;
        for (; i <= c && current == val[i]; i++)
            ;
    }
    for (int i = 1; i <= N; i++)
    {
        ++S;
        seg[S].x = BS(rect[i].x1);
        seg[S].y = BS(rect[i].y1);
        seg[S].z = BS(rect[i].y2);
        seg[S].value = 1;

        ++S;
        seg[S].x = BS(rect[i].x2);
        seg[S].y = seg[S - 1].y;
        seg[S].z = seg[S - 1].z;
        seg[S].value = -1;
    }
    sort(seg + 1, seg + 1 + S);

    for (L = 0; L * L < C - 1; L++)
        ;
    M = 0;
    for (int i = 1; i <= C - 1; i += L)
    {
        ++M;
        node[M].left = i;
        node[M].right = min(i + L - 1, C - 1);
        node[M].init();
    }

    scanf("%d", &K);
}

void update(int from, int to, int v)
{
    for (int i = 1; i <= M; i++)
    {
        if (from >= node[i].left && from <= node[i].right)
        {
            if (to >= node[i].left && to <= node[i].right)
            {
                node[i].update(from, to, v);
                return;
            }
            else
            {
                node[i].update(from, node[i].right, v);
                for (int j = i + 1; j <= M; j++)
```

```
            {
                if (to >= node[j].left && to <= node[j].right)
                {
                    node[j].update(node[j].left, to, v);
                    break;
                }
                else
                {
                    node[j].cover += v;
                }
            }
        }
        return;
    }
}

void work()
{
    double result = 0;
    for (int i = 1; i <= S;)
    {
        int last = seg[i].x;
        int current = seg[i].x;
        for (; i <= S && current == seg[i].x; i++)
        {
            update(seg[i].y, seg[i].z - 1, seg[i].value);
        }
        if (i <= S)
        {
            double sum = 0;
            for (int j = 1; j <= M; j++)
            {
                sum += node[j].query();
            }
            result += sum * (val[seg[i].x] - val[last]);
        }
    }
    printf("%.3f\n", result);
}
```

回路插头DP四进制括号状压必走不走分开转移

```
const int HSIZE = 4001;
const int QSIZE = 100010;

struct Queue
{
    int state;
    LL sum;
};

struct ListNode
{
    ListNode *next;
    int hval;
    int index;
};

ListNode nodes[50010];
int C;

struct Hash
{
    ListNode *hash[HSIZE];

    void clear()
    {
        C = 0;
        for (int i = 0; i < HSIZE; i++)
        {
            hash[i] = &nodes[C++];
            hash[i]->next = NULL;
        }
    }
    int add(int k, int id)
    {
        int hval = k % HSIZE;
        for (ListNode *ite = hash[hval]->next; ite; ite = ite->next)
        {
            if (ite->hval == k)
                return ite->index;
        }
        ListNode *t = &nodes[C++];
        t->next = hash[hval]->next;
        hash[hval]->next = t;
        t->index = id;
        t->hval = k;
        return -1;
    }
};

int M, N;

char mat[20][20];

int destx, desty;

int vis[20][20];
```

```
int dx[] =
{ 0, 0, 1, -1 };
int dy[] =
{ 1, -1, 0, 0 };

int mask[15];

Hash S;

Queue queue[QSIZE];

LL result;

int zip(int a[])
{
    int res = 0;
    for (int i = N; i >= 0; i--)
    {
        res <<= 2;
        res += a[i];
    }
    return res;
}

void unzip(int s, int a[])
{
    for (int i = 0; i < N + 1; i++)
    {
        a[i] = s & 3;
        s >>= 2;
    }
}

void push(int s, int &tail, LL sum)
{
    int id = S.add(s, tail);
    if (id != -1)
        queue[id].sum += sum;
    else
    {
        queue[tail].state = s;
        queue[tail].sum = sum;
        tail++;
        if (tail == QSIZE)
            tail = 0;
    }
}

void transCant(int &tail, int px, int py, int p, int q, Queue
current)
{
    if (p || q)
        return;
    push(current.state, tail, current.sum);
}

void transMust(int &tail, int px, int py, int p, int q, Queue
```

```
current)
{
    int tmp[20];
    if (p && q)
    {
        if (p == 1 && q == 2)
        {
            if (!((px == destx && py >= desty) || px > destx))
                return;
            if (current.state & mask[py - 1] & mask[py])
                return;
            result += current.sum;
            return;
        }
        else if (p == 2 && q == 1)
            ;
        else
        {
            unzip(current.state, tmp);
            int stack[20];
            int top = 0;
            if (p == 1 && q == 1)
            {
                for (int j = py; j < N + 1; j++)
                {
                    if (tmp[j] == 0)
                        continue;
                    if (tmp[j] == 1)
                        stack[top++] = j;
                    else
                    {
                        if (stack[top - 1] == py)
                        {
                            current.state -= 1 << (2 * j);
                            break;
                        }
                        top--;
                    }
                }
            }
            else
            {
                for (int j = py - 1; j >= 0; j--)
                {
                    if (tmp[j] == 0)
                        continue;
                    if (tmp[j] == 2)
                        stack[top++] = j;
                    else
                    {
                        if (stack[top - 1] == py - 1)
                        {
                            current.state += 1 << (2 * j);
                            break;
                        }
                        top--;
                    }
                }
```

```
            }
        }
        current.state = current.state & mask[py - 1] & mask[py];
        push(current.state, tail, current.sum);
    }
    else if (!p && !q)
    {
        current.state += (1 << (2 * (py - 1))) + (2 << (2 * py));
        push(current.state, tail, current.sum);
    }
    else
    {
        int t = p + q;
        current.state = current.state & mask[py - 1] & mask[py];
        push(current.state + (t << (2 * (py - 1))), tail,
current.sum);
        push(current.state + (t << (2 * py)), tail, current.sum);
    }
}

void solve()
{
    result = 0;

    int head = 0, tail = 0;
    int px = 1, py = 1;
    queue[tail].state = 0;
    queue[tail].sum = 1;
    tail++;
    for (; (px != M + 1 && head != tail);)
    {
        S.clear();
        int hd = head, tl = tail;
        for (int i = hd; i != tl; i = (i + 1) % QSIZE)
        {
            Queue current = queue[head++];
            if (head == QSIZE)
                head = 0;
            if (py == 1)
            {
                if (current.state >> (2 * N))
                    continue;
                current.state <<= 2;
            }
            int p = (current.state >> (2 * py - 2)) & 3;
            int q = (current.state >> (2 * py)) & 3;
            if (mat[px][py] == 'X')
            {
                transCant(tail, px, py, p, q, current);
            }
            else if (mat[px][py] == 'O')
            {
                transMust(tail, px, py, p, q, current);
            }
            else
            {
                transMust(tail, px, py, p, q, current);
                transCant(tail, px, py, p, q, current);
```

```
            }
        }
        py++;
        if (py == N + 1)
        {
            px++;
            py = 1;
        }
    }
    cout << result << endl;
}

void init()
{
    destx = desty = 0;
    memset(mat, 'X', sizeof(mat));
    scanf("%d%d", &M, &N);
    int flag = 1;
    for (int i = 1; i <= M; i++)
    {
        scanf("%s", mat[i] + 1);
        mat[i][N + 1] = 'X';
    }
    for (int i = 1; i <= M; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            if (mat[i][j] == 'O')
            {
                destx = i;
                desty = j;
            }
        }
    }
}

int main()
{
    int t;
    scanf("%d", &t);
    for (int i = 0; i < 15; i++)
    {
        mask[i] = ((1 << 30) - 1) ^ (3 << (2 * i));
    }
    for (int i = 1; i <= t; i++)
    {
        printf("Case %d: ", i);
        init();
        solve();
    }
    return 0;
}
```

Can You Answer These Queries II

```
const LL INF = 1LL << 62;

template<class T>
struct SegNode
{
    int left, right;
    T add, mxadd;
    T mx, mxmx;

    int mid()
    {
        return (left + right) >> 1;
    }
};
template<class T>
struct SegTree
{
    SegNode<T> tree[7 * maxn];
    void init(int left, int right, int idx)
    {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].add = tree[idx].mxadd = tree[idx].mx =
tree[idx].mxmx = 0;
        if (left == right)
        {
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1);
        init(mid + 1, right, (idx << 1) + 1);
        push_up(idx);
    }
    void update(int left, int right, int idx, T value)
    {
        push_down(idx);
        if (left <= tree[idx].left && right >= tree[idx].right)
        {
            tree[idx].add += value;
            tree[idx].mxadd = max(tree[idx].mxadd, tree[idx].add);
            return;
        }
        int mid = tree[idx].mid();
        if (left <= mid)
            update(left, right, idx << 1, value);
        if (mid < right)
            update(left, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    T query(int left, int right, int idx)
    {
        push_down(idx);
        if (left == tree[idx].left && right == tree[idx].right)
        {
            return tree[idx].mxmx;
        }
```

```
        int mid = tree[idx].mid();
        if (right <= mid)
            return query(left, right, idx << 1);
        else if (left > mid)
            return query(left, right, (idx << 1) + 1);
        else
        {
            return max(query(left, mid, idx << 1),
                   query(mid + 1, right, (idx << 1) + 1));
        }
    }
    void push_down(int idx)
    {
        T add = tree[idx].add;
        tree[idx].mxmx = max(tree[idx].mxmx, tree[idx].mx +
tree[idx].mxadd);
        tree[idx].mx += add;

        tree[idx << 1].mxadd = max(tree[idx << 1].mxadd,
               tree[idx << 1].add + tree[idx].mxadd);
        tree[idx << 1].add += add;

        tree[(idx << 1) + 1].mxadd = max(tree[(idx << 1) + 1].mxadd,
               tree[idx].mxadd + tree[(idx << 1) + 1].add);
        tree[(idx << 1) + 1].add += add;

        tree[idx].add = 0;
        tree[idx].mxadd = 0;
    }
    void push_up(int idx)
    {
        push_down(idx << 1);
        push_down((idx << 1) + 1);
        tree[idx].mx = max(tree[idx << 1].mx, tree[(idx << 1) +
1].mx);
        tree[idx].mxmx = max(tree[idx].mxmx,
               max(tree[idx << 1].mxmx, tree[(idx << 1) + 1].mxmx));
    }
};

struct Query
{
    int x, y;
    int id;

    bool operator<(const Query& p) const
    {
        return y < p.y;
    }
};

Query query[maxn];

int arr[maxn];
int val[maxn];
int last[maxn];
int pre[maxn];
int C;
```

```
SegTree<LL> tree;

int N, Q;

int BS(int k)
{
    int low = 1, high = C;
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (val[mid] == k)
            return mid;
        else if (val[mid] < k)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

void build()
{
    for (int i = 1; i <= N; i++)
    {
        pre[i] = 0;
        last[i] = 0;
    }
    for (int i = 1; i <= N; i++)
    {
        pre[i] = last[arr[i]];
        last[arr[i]] = i;
    }
}

void init()
{
    scanf("%d", &N);
    C = 0;
    for (int i = 1; i <= N; i++)
    {
        scanf("%d", arr + i);
        val[++C] = arr[i];
    }
    sort(val + 1, val + 1 + C);
    int c = C;
    C = 0;
    for (int i = 1; i <= c;)
    {
        int current = val[i];
        val[++C] = current;
        for (; i <= c && current == val[i]; i++)
            ;
    }
    for (int i = 1; i <= N; i++)
    {
        arr[i] = BS(arr[i]);
    }
```

```
    tree.init(1, N, 1);
    build();
    scanf("%d", &Q);
    for (int i = 1; i <= Q; i++)
    {
        scanf("%d%d", &query[i].x, &query[i].y);
        query[i].id = i;
    }
    sort(query + 1, query + 1 + Q);
}

LL result[maxn];

void work()
{
    for (int i = 1, j = 1; i <= Q; i++)
    {
        for (; j <= N && j <= query[i].y; j++)
        {
            tree.update(pre[j] + 1, j, 1, val[arr[j]]);
        }
        result[query[i].id] = tree.query(query[i].x, query[i].y, 1);
    }
    for (int i = 1; i <= Q; i++)
    {
        printf("%lld\n", result[i]);
    }
}
```