1. tpl

1.1. Algo

```
int[] unique(int[] is, int b, int e) {
    if (b == e) return new int[0];
    int count = 1;
    for (int i = b + 1; i < e; i++) {
        if (is[i] != is[i - 1])
            count++;
    }
    int[] res = new int[count];
    res[0] = is[b];
    int id = 1;
    for (int i = b + 1; i < e; i++) {
        if (is[i] != is[i - 1])
            res[id++] = is[i];
    }
    return res;
}
int log2(int b) {
    return 31 - Integer.numberOfLeadingZeros(b);
}
```

2. ds

2.1. LCA

```
class LCA {
    List<Integer>[] vs;
    int root;
    int[] depth;
    int[][] pre;
    LCA(List<Integer>[] vs, int root) {
        this.vs = vs;
        this.root = root;
        int n = vs.length;
        depth = new int[n];
        pre = new int[Algo.log2(n) + 1][n];
        dfs(root, -1, 0);
        for (int k = 0; k + 1 < pre.length; k++) {
            for (int v = 0; v < n; v++) {
                if (pre[k][v] < 0) pre[k + 1][v] = -1;
                else pre[k + 1][v] = pre[k][pre[k][v]];
            }
        }
    }
    void dfs(int v, int p, int d) {
        pre[0][v] = p;
        depth[v] = d;
        for (int u : vs[v]) if (u != p) {
            dfs(u, v, d + 1);
        }
    }
    int lca(int u, int v) {
        if (depth[u] > depth[v]) return lca(v, u);
        v = climb(v, depth[v] - depth[u]);
        if (u == v) return u;
        for (int k = pre.length - 1; k >= 0; k--) {
            if (pre[k][u] != pre[k][v]) {
                u = pre[k][u];
                v = pre[k][v];
            }
        }
        return pre[0][u];
    }
    int climb(int v, int d) {
        for (int k = 0; k < pre.length; k++) {
            if ((d >> k & 1) != 0) v = pre[k][v];
        }
        return v;
    }
}
```

2.2. LCT

```
class LCT {
    class T {
        int id;
        boolean rev;
        double p;
        T pre;
        T left;
        T right;
        T(int id, boolean rev, double p, T pre, T left, T right) {
            this.id = id;
            this.rev = rev;
            this.p = p;
            this.pre = pre;
            this.left = left;
            this.right = right;
        }
        T(int id) {
            this(id, false, Math.random(), NULL, NULL, NULL);
        }
        T change(T left, T right) {
            this.left = left; left.pre = this;
            this.right = right; right.pre = this;
            return this;
        }
        T setRev() {
            if (this == NULL) return NULL;
            rev ^= true;
            T t = left; left = right; right = t;
            return this;
        }
        T push() {
            if (rev) {
                left.setRev();
                right.setRev();
                rev ^= true;
            }
            return this;
        }
    }
    T merge(T t1, T t2) {
        if (t1 == NULL) return t2;
        if (t2 == NULL) return t1;
        if (t1.p < t2.p) return t1.push().change(t1.left,
                                 merge(t1.right, t2));
        return t2.push().change(merge(t1, t2.left), t2.right);
    }
    T[] split(T t) {
        pushDownAllMark(t);
        T[] res = new T[2];
        res[1] = t.right;
        res[0] = t.change(t.left, NULL);
        T tcp = t;
        for (;;) {
            if (t.pre.left == t) {
```

```
                t = t.pre;
                res[1] = t.change(res[1], t.right);
            } else if (t.pre.right == t) {
                t = t.pre;
                res[0] = t.change(t.left, res[0]);
            } else {
                res[0].pre = t.pre;
                res[1].pre = tcp;
                return res;
            }
        }
    }
    T access(T t) {
        T last = NULL;
        while (t != NULL) {
            T[] ss = split(t);
            t = ss[0].pre;
            last = merge(ss[0], last);
        }
        last.pre = NULL;
        return last;
    }
    T makeRoot(T t) {
        return access(t).setRev();
    }
    T getRoot(T t) {
        t = access(t);
        while (t.push().left != NULL) t = t.left;
        return t;
    }
    void link(T x, T y) {
        makeRoot(x).pre = y;
    }
    void cut(T x, T y) {
        makeRoot(y);
        access(y);
        while (x.pre.left == x || x.pre.right == x) x = x.pre;
        x.pre = NULL;
    }
    void pushDownAllMark(T t) {
        if (t.pre.left == t || t.pre.right == t)
```

```
            pushDownAllMark(t.pre);
        t.push();
    }
    T NULL = new T(0);
}
```

2.3. Seg

```
abstract class Seg {
    int N;
    long[] is;
    long[] ds;
    final long I = 1;
    final long D = 1;
    Seg(int n) {
        N = Integer.highestOneBit(n) << 1;
        is = new long[N * 2];
        ds = new long[N * 2];
        Arrays.fill(ds, D);
        for (int i = N; i < N * 2; i++) {
            is[i] = I;
        }
        for (int i = N - 1; i > 0; i--) {
            pushUp(i);
        }
    }
    abstract long mergeInfo(long a, long b);
    void pushUp(int o) {
        is[o] = mergeInfo(is[o * 2], is[o * 2 + 1]);
    }
    abstract void push(int o, int l, int r, long d);
    long query(int s, int t) {
        return query(1, 0, N, s, t);
    }
    long query(int o, int l, int r, int s, int t) {
        if (s <= l && r <= t) {
            // 如果 [l, r) 和 [s, t) 不同，需要修改
            return is[o];
        } else {
            pushDown(o, l, r);
            int m = (l + r) / 2;
            if (t <= m) return query(o * 2, l, m, s, t);
            if (s >= m) return query(o * 2 + 1, m, r, s, t);
            return mergeInfo(query(o * 2, l, m, s, m),
                    query(o * 2 + 1, m, r, m, t));
        }
    }
    void update(int s, int t, long d) {
        update(1, 0, N, s, t, d);
    }
    void update(int o, int l, int r, int s, int t, long d) {
        if (s <= l && r <= t) {
            // 如果 [l, r) 和 [s, t) 不同，需要修改
            push(o, l, r, d);
        } else {
            pushDown(o, l, r);
            int m = (l + r) / 2;
            if (s < m) update(o * 2, l, m, s, Math.min(m, t), d);
            if (t > m) update(o * 2 + 1, m, r, Math.max(s, m), t,
d);
            pushUp(o);
        }
    }
    void pushDown(int o, int l, int r) {
        if (ds[o] != D) {
            int m = (l + r) / 2;
            push(o * 2, l, m, ds[o]);
            push(o * 2 + 1, m, r, ds[o]);
            ds[o] = D;
        }
    }
}
```

2.4. Hash

```
class Hash {
    final long BASE = (long) (1e9 + 7);
    long[] ps;
    Hash(int n) {
        ps = new long[n + 1];
        for (int i = 0; i <= n; i++)
            ps[i] = (i == 0 ? 1 : ps[i - 1] * BASE);
    }
    long[] build(char[] cs) {
        int n = cs.length;
        long[] hs = new long[n + 1];
        for (int i = 0; i < n; i++)
            hs[i + 1] = hs[i] * BASE + cs[i];
        return hs;
    }
    long[] build(int[] is) {
        int n = is.length;
        long[] hs = new long[n + 1];
        for (int i = 0; i < n; i++)
            hs[i + 1] = hs[i] * BASE + is[i];
        return hs;
    }
    long getHash(char[] cs) {
        return getHash(cs, 0, cs.length);
    }
    long getHash(char[] cs, int b, int e) {
        long h = 0;
        for (int i = b; i < e; i++) {
            h = h * BASE + cs[i];
        }
        return h;
    }
    long getHash(int[] is) {
        return getHash(is, 0, is.length);
    }
    long getHash(int[] is, int b, int e) {
        long h = 0;
        for (int i = b; i < e; i++) {
            h = h * BASE + is[i];
        }
        return h;
    }
    long get(long[] hs, int b, int e) {
        return hs[e] - hs[b] * ps[e - b];
    }
}
```

2.5. Hash2

```
class Hash2 {
    final long BL;
    final long BR;
    final long ML;
    final long MR;
    final long[] psl;
    final long[] psr;
    Hash2(int n) {
        Random r = new Random(System.nanoTime());
        BL = (long) (1e9 + r.nextInt((int) 1e9));
        BR = (long) (1e9 + r.nextInt((int) 1e9));
        ML = (long) (1e9 + r.nextInt((int) 1e9));
        MR = (long) (1e9 + r.nextInt((int) 1e9));
        psl = new long[n + 1];
        psr = new long[n + 1];
        for (int i = 0; i <= n; i++)
            psl[i] = (i == 0 ? 1 : psl[i - 1] * BL) % ML;
        for (int i = 0; i <= n; i++)
            psr[i] = (i == 0 ? 1 : psr[i - 1] * BR) % MR;
    }
    long[] build(char[] cs) {
        int n = cs.length;
        long[] hs = new long[n + 1];
        long l = 0, r = 0;
        for (int i = 0; i < n; i++) {
            l = (l * BL + cs[i]) % ML;
            r = (r * BR + cs[i]) % MR;
            if (l < 0) l += ML;
            if (r < 0) r += MR;
            hs[i + 1] = (l << 32) | r;
        }
        return hs;
    }
    long[] build(int[] is) {
        int n = is.length;
        long[] hs = new long[n + 1];
        long l = 0, r = 0;
        for (int i = 0; i < n; i++) {
            l = (l * BL + is[i]) % ML;
            r = (r * BR + is[i]) % MR;
            if (l < 0) l += ML;
            if (r < 0) r += MR;
            hs[i + 1] = (l << 32) | r;
        }
        return hs;
    }
    long getHash(char[] cs) {
        return getHash(cs, 0, cs.length);
    }
    long getHash(char[] cs, int b, int e) {
        long l = 0, r = 0;
        for (int i = b; i < e; i++) {
            l = (l * BL + cs[i]) % ML;
            r = (r * BR + cs[i]) % MR;
        }
        if (l < 0) l += ML;
        if (r < 0) r += MR;
        return (l << 32) | r;
    }
    long getHash(int[] is) {
        return getHash(is, 0, is.length);
    }
    long getHash(int[] is, int b, int e) {
        long l = 0, r = 0;
        for (int i = b; i < e; i++) {
            l = (l * BL + is[i]) % ML;
            r = (r * BR + is[i]) % MR;
        }
        if (l < 0) l += ML;
        if (r < 0) r += MR;
        return (l << 32) | r;
    }
    long get(long[] hs, int b, int e) {
        long el = hs[e] >>> 32;
        long er = hs[e] & 0xffffffffL;
        long bl = hs[b] >>> 32;
        long br = hs[b] & 0xffffffffL;
        long l = el - bl * psl[e - b] % ML;
        long r = er - br * psr[e - b] % MR;
        if (l < 0) l += ML;
        if (r < 0) r += MR;
        return (l << 32) | r;
    }
}
```

2.6. Treap

```
class Treap {
    class T {
        int key, size;
        double p;
        T left, right;
        T(int key, int size, double p, T left, T right) {
            this.key = key;
            this.size = size;
            this.p = p;
            this.left = left;
            this.right = right;
        }
        T(int key) {
            this(key, 1, Math.random(), NULL, NULL);
        }
        T change(T left, T right) {
            size = left.size + right.size + 1;
            this.left = left;
            this.right = right;
            return this;
        }
        T push() {
            if (this != NULL) {
            }
            return this;
        }
    }
    T NULL = new T(0, 0, 0, null, null);
    T[] splitSize(T t, int size) {
        T[] res;
        if (size <= 0) {
            res = new T[] { NULL, t };
        } else if (size <= t.push().left.size) {
            res = splitSize(t.left, size);
            res[1] = t.change(res[1], t.right);
        } else {
            res = splitSize(t.right, size - t.left.size - 1);
            res[0] = t.change(t.left, res[0]);
        }
        return res;
    }
    T[] splitKey(T t, int key) {
        T[] res;
        if (t == NULL) {
            res = new T[] { NULL, NULL };
        } else if (key < t.push().key) {
            res = splitKey(t.left, key);
            res[1] = t.change(res[1], t.right);
        } else {
            res = splitKey(t.right, key);
            res[0] = t.change(t.left, res[0]);
        }
        return res;
    }
    void print(T t, String indent) {
        if (t != NULL) {
            print(t.push().right, indent + "      ");
            System.err.printf("%s%3d%3d%n", indent, t.key, t.size);
            print(t.left, indent + "      ");
        }
        if (indent.length() == 0)
            System.err.println("--------------------------------");
    }
    T merge(T t1, T t2) {
        if (t1 == NULL) return t2;
        if (t2 == NULL) return t1;
        if (t1.p < t2.p) return t1.push().change(t1.left,
                                 merge(t1.right, t2));
        return t2.push().change(merge(t1, t2.left), t2.right);
    }
}
```

2.7. MatMin

```
class MatMin {
    SegMinC[] ss;
    int N;
    int M;
    MatMin(int row, int col) {
        N = Integer.highestOneBit(row) << 1;
        M = Integer.highestOneBit(col) << 1;
        ss = new SegMinC[N * 2];
        for (int i = 0; i < N * 2; i++) {
            ss[i] = new SegMinC(col);
        }
```

```
    }
    int update(int x, int y, int m, int a) {
        x += N;
        int val = ss[x].update(y, m, a);
        for (x >>= 1; x > 0; x >>= 1) {
            ss[x].update(y, 0, Math.min(ss[x * 2].is[M + y],
                                ss[x * 2 + 1].is[M + y]));
        }
        return val;
    }
    int query(int x0, int y0, int x1, int y1) {
        int res = Integer.MAX_VALUE;
        while (0 < x0 && x0 + (x0 & -x0) <= x1) {
            int i = (N + x0) / (x0 & -x0);
            res = Math.min(res, ss[i].query(y0, y1));
            x0 += x0 & -x0;
        }
        while (x0 < x1) {
            int i = (N + x1) / (x1 & -x1) - 1;
            res = Math.min(res, ss[i].query(y0, y1));
            x1 -= x1 & -x1;
        }
        return res;
    }
}
```

2.8. MatSum

```
class MatSum {
    BIT[] bs;
    MatSum(int row, int col) {
        bs = new BIT[row + 1];
        for (int i = 0; i < bs.length; i++) {
            bs[i] = new BIT(col);
        }
    }
    void add(int x, int y, int val) {
        for (int i = x + 1; i < bs.length; i += i & -i) {
            bs[i].add(y, val);
        }
    }
    int sum(int x0, int y0, int x1, int y1) {
        if (x0 != 0)
```

```
            return sum(0, y0, x1, y1) - sum(0, y0, x0, y1);
        int res = 0;
        for (int i = x1; i > 0; i -= i & -i) {
            res += bs[i].sum(y0, y1);
        }
        return res;
    }
}
```

2.9. SegMinC

```
class SegMinC {
    int[] is;
    int N;
    SegMinC(int n) {
        N = Integer.highestOneBit(n) << 1;
        is = new int[N * 2];
        Arrays.fill(is, Integer.MAX_VALUE);
    }
    int update(int k, int m, int a) {
        k += N;
        int val = is[k] = is[k] * m + a;
        for (k >>= 1; k > 0; k >>= 1) {
            is[k] = Math.min(is[k * 2], is[k * 2 + 1]);
        }
        return val;
    }
    int query(int s, int t) {
        int res = Integer.MAX_VALUE;
        while (0 < s && s + (s & -s) <= t) {
            int i = (N + s) / (s & -s);
            res = Math.min(res, is[i]);
            s += s & -s;
        }
        while (s < t) {
            int i = (N + t) / (t & -t) - 1;
            res = Math.min(res, is[i]);
            t -= t & -t;
        }
        return res;
    }
}
```

2.10. Rational

```
class Rational implements Comparable<Rational> {
    final Rational ZERO = new Rational(B.ZERO, B.ONE);
    final Rational ONE = new Rational(B.ONE,B.ONE);
    BigInteger num;
    BigInteger den;
    Rational(BigInteger num, BigInteger den) {
        this.num = num;
        this.den = den;
        red();
    }
    void red() {
        BigInteger gcd = num.gcd(den);
        if (gcd.signum() != 0) {
            num = num.divide(gcd);
            den = den.divide(gcd);
        }
        if (den.signum() < 0) {
            num = num.negate();
            den = den.negate();
        }
    }
    Rational add(Rational r) {
        return new Rational(num.mul(r.den).add(r.num.mul(den)),
                den.mul(r.den));
    }
    Rational sub(Rational r) {
        return new Rational(num.mul(r.den).sub(r.num.mul(den)),
                den.mul(r.den));
    }
    Rational mul(Rational r) {
        return new Rational(num.mul(r.num), den.mul(r.den));
    }
    Rational div(Rational r) {
        return new Rational(num.mul(r.den), den.mul(r.num));
    }
    int signum() {
        return num.signum();
    }
    Rational pow(int b) {
        BigInteger n = B.ONE, d = B.ONE, an = num, ad = den;
        while (b > 0) {
            if ((b & 1) == 1) {
                n = n.multiply(an);
                d = d.multiply(ad);
            }
            an = an.multiply(an);
            ad = ad.multiply(ad);
            b >>>= 1;
        }
        return new Rational(n, d);
    }
    @Override
    int compareTo(Rational o) {
        return num.multiply(o.den).compareTo(o.num.multiply(den));
    }
    @Override
    String toString() {
        return num + "/" + den;
    }
}
```

2.11. Intervals

```
class Intervals<K, V> {
    TreeMap<K, V> map = new TreeMap<K, V>();
    Intervals(K min, K max, V ini) {
        map.put(min, ini);
        map.put(max, ini);
    }
    void paint(K s, K t, V c) {
        V p = get(t);
        map.subMap(s, t).clear();
        if (!get(s).equals(c)) map.put(s, c);
        if (!get(t).equals(p)) map.put(t, p);
        if (p.equals(c)) map.remove(t);
    }
    V get(K k) {
        return map.floorEntry(k).getValue();
    }
}
```

2.12. LongSegSum

```
class LongSegSum {
    int N;
    long[] ls, mul, add;
```

```
    LongSegSum(int n) {
        N = Integer.highestOneBit(n) << 1;
        ls = new long[N * 2];
        mul = new long[N * 2];
        add = new long[N * 2];
        Arrays.fill(mul, 1);
    }
    int s, t;
    long m, a;
    void update(int s, int t, long m, long a) {
        this.s = s;
        this.t = t;
        this.m = m;
        this.a = a;
        update(1, 0, N, 1, 0);
    }
    void update(int o, int L, int R, long m, long a) {
        if (s <= L && R <= t) {
            // push this.m, this.a to m, a
            m = this.m * m;
            a = this.m * a + this.a;
        }
        // push m, a to L[o], R[o]
        mul[o] = m * mul[o];
        add[o] = m * add[o] + a;
        if (t <= L || R <= s || s <= L && R <= t) {
            // maintain is[o] for m, a
            ls[o] = m * ls[o] + a * (R - L); // need change
        } else {
            int M = (L + R) / 2;
            update(o * 2, L, M, mul[o], add[o]);
            update(o * 2 + 1, M, R, mul[o], add[o]);
            // init L[o], R[o]
            mul[o] = 1;
            add[o] = 0;
            ls[o] = ls[o * 2] + ls[o * 2 + 1];
        }
    }
    long query2(int s, int t) {
        this.s = s;
        this.t = t;
        return query(1, 0, N, 1, 0);
    }
    long query(int o, int L, int R, long m, long a) {
        // push m, a to L[o], R[o]
        mul[o] = m * mul[o];
        add[o] = m * add[o] + a;
        if (t <= L || R <= s || s <= L && R <= t) {
            // maintain is[o] for m, a
            ls[o] = m * ls[o] + a * (R - L); // need change
            if (t <= L || R <= s) return 0;
            return ls[o];
        } else {
            int M = (L + R) / 2;
            long res = 0;
            res += query(o * 2, L, M, mul[o], add[o]);
            res += query(o * 2 + 1, M, R, mul[o], add[o]);
            // init L[o], R[o]
            mul[o] = 1;
            add[o] = 0;
            ls[o] = ls[o * 2] + ls[o * 2 + 1];
            return res;
        }
    }
    long query(int s, int t) {
        update(s, t, 1, 0);
        long res = 0; // need change
        while (0 < s && s + (s & -s) <= t) {
            int i = (N + s) / (s & -s);
            res = res + ls[i];
            s += s & -s;
        }
        while (s < t) {
            int i = (N + t) / (t & -t) - 1;
            res = res + ls[i];
            t -= t & -t;
        }
        return res;
    }
    void pushDownMark() {
        pushDown(1, 0, N, 1, 0);
    }
```

```
    void pushDown(int o, int L, int R, long m, long a) {
        ls[o] = ls[o] * m + a * (R - L);
        mul[o] = mul[o] * m;
        add[o] = add[o] * m + a;
        if (o >= N) return ;
        int M = (L + R) / 2;
        pushDown(o * 2, L, M, mul[o], add[o]);
        pushDown(o * 2 + 1, M, R, mul[o], add[o]);
        mul[o] = 1;
        add[o] = 0;
    }
}
```

2.13. LongSegMin

```
class LongSegMin {
    int N;
    long[] ls, mul, add;
    LongSegMin(int n) {
        N = Integer.highestOneBit(n) << 1;
        ls = new long[N * 2];
        Arrays.fill(ls, Long.MAX_VALUE);
        mul = new long[N * 2];
        add = new long[N * 2];
        Arrays.fill(mul, 1);
    }
    int s, t;
    long m, a;
    void update(int s, int t, long m, long a) {
        this.s = s;
        this.t = t;
        this.m = m;
        this.a = a;
        update(1, 0, N, 1, 0);
    }
    void update(int o, int L, int R, long m, long a) {
        if (s <= L && R <= t) {
            // push this.m, this.a to m, a
            m = this.m * m;
            a = this.m * a + this.a;
        }
        // push m, a to L[o], R[o]
        mul[o] = m * mul[o];
        add[o] = m * add[o] + a;
        if (t <= L || R <= s || s <= L && R <= t) {
            // maintain is[o] for m, a
            ls[o] = m * ls[o] + a; // need change
        } else {
            int M = (L + R) / 2;
            update(o * 2, L, M, mul[o], add[o]);
            update(o * 2 + 1, M, R, mul[o], add[o]);
            // init L[o], R[o]
            mul[o] = 1;
            add[o] = 0;
            ls[o] = Math.min(ls[o * 2], ls[o * 2 + 1]);
        }
    }
    long query(int s, int t) {
        update(s, t, 1, 0);
        long res = Long.MAX_VALUE; // need change
        while (0 < s && s + (s & -s) <= t) {
            int i = (N + s) / (s & -s);
            res = Math.min(res, ls[i]);
            s += s & -s;
        }
        while (s < t) {
            int i = (N + t) / (t & -t) - 1;
            res = Math.min(res, ls[i]);
            t -= t & -t;
        }
        return res;
    }
}
```

3. geo

3.1. P

```
class P implements Comparable<P> {
    static final double EPS = 1e-8;
    static double add(double a, double b) {
        if (Math.abs(a + b) < EPS * (Math.abs(a) + Math.abs(b)))
            return 0;
        return a + b;
    }
    final double x, y;
    P(double x, double y) {
        this.x = x;
        this.y = y;
    }
    P sub(P p) {
        return new P(add(x, -p.x), add(y, -p.y));
    }
    P add(P p) {
        return new P(add(x, p.x), add(y, p.y));
    }
    P mul(double k) {
        return new P(x * k, y * k);
    }
    P div(double k) {
        return new P(x / k, y / k);
    }
    double det(P p) {
        return add(x * p.y, -y * p.x);
    }
    double dot(P p) {
        return add(x * p.x, y * p.y);
    }
    double abs() {
        return Math.sqrt(abs2());
    }
    double abs2() {
        return dot(this);
    }
    //饶原点旋转角度 B（弧度值）产生的新点
    P rot(double rad) {
        return new P(add(x * Math.cos(rad), -y * Math.sin(rad)),
                add(x * Math.sin(rad), y * Math.cos(rad)));
    }
    P rot90() {
        return new P(-y, x);
    }
    @Override
    String toString() {
        return "(" + x + ", " + y + ")";
    }
    @Override
    boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        return compareTo((P) obj) == 0;
    }
    @Override
    int compareTo(P p) {
        int b = sig(x - p.x);
        if (b != 0) return b;
        return sig(y - p.y);
    }
    int sig(double x) {
        if (Math.abs(x) < EPS) return 0;
        return x < 0 ? -1 : 1;
    }
    //线段相交判定
    boolean crsSS(P p1, P p2, P q1, P q2) {
        if (Math.max(p1.x, p2.x) + EPS < Math.min(q1.x, q2.x))
            return false;
        if (Math.max(q1.x, q2.x) + EPS < Math.min(p1.x, p2.x))
            return false;
        if (Math.max(p1.y, p2.y) + EPS < Math.min(q1.y, q2.y))
            return false;
        if (Math.max(q1.y, q2.y) + EPS < Math.min(p1.y, p2.y))
            return false;
        return p2.sub(p1).det(q1.sub(p1)) *
                p2.sub(p1).det(q2.sub(p1)) <= 0 &&
                q2.sub(q1).det(p1.sub(q1)) *
                q2.sub(q1).det(p2.sub(q1)) <= 0;
    }
    //直线和线段的相交判定
    boolean crsLS(P l1, P l2, P s1, P s2) {
        return s1.sub(l2).det(l1.sub(l2)) *
                s2.sub(l2).det(l1.sub(l2)) <= 0;
    }
    //直线相交判定
    //返回-1表示重合，为0表示平行，为1表示相交
    int crsLL(P p1, P p2, P q1, P q2) {
```

```
        if (sig(p1.sub(p2).det(q1.sub(q2))) != 0) return 1;
        if (sig(p1.sub(q2).det(q1.sub(p2))) != 0) return 0;
        return -1;
    }
    //直线和直线的交点
    P isLL(P p1, P p2, P q1, P q2) {
        double d = q2.sub(q1).det(p2.sub(p1));
        if (sig(d) == 0) return null;
        return p1.add(
                p2.sub(p1).mul(q2.sub(q1).det(q1.sub(p1)) / d));
    }
    //点到直线的垂足
    P proj(P p1, P p2, P q) {
        return p1.add(p2.sub(p1).mul(p2.sub(p1).dot(q.sub(p1))
                / p2.sub(p1).abs2()));
    }
    //计算多边形的有向面积
    //点不需要有顺序
    double directedArea(P... ps) {
        double res = 0;
        for (int i = 0; i < ps.length; i++) {
            res += ps[i].det(ps[(i + 1) % ps.length]);
        }
        return res / 2;
    }
    //计算多边形的面积
    //点不需要有顺序
    double area(P... ps) {
        return Math.abs(directedArea(ps));
    }
    //线段到点的距离
    double disSP(P p1, P p2, P q) {
        if (p2.sub(p1).dot(q.sub(p1)) <= 0) return q.sub(p1).abs();
        if (p1.sub(p2).dot(q.sub(p2)) <= 0) return q.sub(p2).abs();
        return disLP(p1, p2, q);
    }
    //直线到点的距离
    double disLP(P p1, P p2, P q) {
        return Math.abs(p2.sub(p1).det(q.sub(p1)))
                / p2.sub(p1).abs();
    }
    //圆和线段的相交判定
    boolean crsCS(P c, double r, P p1, P p2) {
        return disSP(p1, p2, c) < r + EPS &&
                (r < c.sub(p1).abs() + EPS ||
                r < c.sub(p2).abs() + EPS);
    }
    //圆和圆的相交判定
    boolean crsCC(P c1, double r1, P c2, double r2) {
        double dis = c1.sub(c2).abs();
        return dis < r1 + r2 + EPS &&
                Math.abs(r1 - r2) < dis + EPS;
    }
    //四点共圆判定
    boolean onC(P p1, P p2, P p3, P p4) {
        P c = CCenter(p1, p2, p3);
        if (c == null) return false; //有三点共线，返回 false
        return add(c.sub(p1).abs2(), -c.sub(p4).abs2()) == 0;
    }
    //三点共圆的圆心
    P CCenter(P p1, P p2, P p3) {
        if (disLP(p1, p2, p3) < EPS) return null; // 三点共线
        P q1 = p1.add(p2).mul(0.5);
        P q2 = q1.add(p1.sub(p2).rot90());
        P s1 = p3.add(p2).mul(0.5);
        P s2 = s1.add(p3.sub(p2).rot90());
        return isLL(q1, q2, s1, s2);
    }
    //直线和圆的交点
    P[] isCL(P c, double r, P p1, P p2) {
        double x = p1.sub(c).dot(p2.sub(p1));
        double y = p2.sub(p1).abs2();
        double d = add(x * x,
                -y * (add(p1.sub(c).abs2(), -r * r)));
        if (d < -EPS) return new P[0];
        if (d < 0) d = 0;
        P q1 = p1.sub(p2.sub(p1).mul(x / y));
        P q2 = p2.sub(p1).mul(Math.sqrt(d) / y);
        return new P[]{q1.sub(q2), q1.add(q2)};
    }
    //两圆的交点
    P[] isCC(P c1, double r1, P c2, double r2) {
        double x = c1.sub(c2).abs2();
        double y = (add(r1 * r1, -r2 * r2) / x + 1) / 2;
        double d = add(r1 * r1 / x, -y * y);
        if (d < -EPS) return new P[0];
        if (d < 0) d = 0;
        P q1 = c1.add(c2.sub(c1).mul(y));
        P q2 = c2.sub(c1).mul(Math.sqrt(d)).rot90();
        return new P[]{q1.sub(q2), q1.add(q2)};
    }
    //点和圆的两个切点
    P[] tanCP(P c, double r, P p) {
        double x = p.sub(c).abs2();
        double d = add(x, -r * r);
        if (d < -EPS) return new P[0];
        if (d < 0) d = 0;
        P q1 = p.sub(c).mul(r * r / x);
        P q2 = p.sub(c).mul(-r * Math.sqrt(d) / x).rot90();
        return new P[]{c.add(q1.sub(q2)), c.add(q1.add(q2))};
    }
    //两圆的公切线
    //返回的是切点对
    P[][] tanCC(P c1, double r1, P c2, double r2) {
        List<P[]> list = new ArrayList<P[]>();
        if (Math.abs(r1 - r2) < EPS) {
            P dir = c2.sub(c1);
            dir = dir.mul(r1 / dir.abs()).rot90();
            list.add(new P[]{c1.add(dir), c2.add(dir)});
            list.add(new P[]{c1.sub(dir), c2.sub(dir)});
        } else {
            P p = c1.mul(-r2).add(c2.mul(r1)).div(r1 - r2);
            P[] ps = tanCP(c1, r1, p);
            P[] qs = tanCP(c2, r2, p);
            for (int i = 0; i < ps.length && i < qs.length; i++) {
                list.add(new P[]{ps[i], qs[i]});
            }
        }
        P p = c1.mul(r2).add(c2.mul(r1)).div(r1 + r2);
        P[] ps = tanCP(c1, r1, p);
        P[] qs = tanCP(c2, r2, p);
        for (int i = 0; i < ps.length && i < qs.length; i++) {
            list.add(new P[]{ps[i], qs[i]});
        }
        return list.toArray(new P[0][]);
    }
    //两圆公共部分的面积
    double areaCC(P c1, double r1, P c2, double r2) {
        double d = c1.sub(c2).abs();
        if (r1 + r2 < d + EPS) return 0;
        if (d < Math.abs(r1 - r2) + EPS) {
            double r = Math.min(r1, r2);
            return r * r * Math.PI;
        }
        double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
        double t1 = Math.acos(x / r1);
        double t2 = Math.acos((d - x) / r2);
        return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * Math.sin(t1);
    }
    //以 r 为半径的圆 O 与三角形 Op1p2 的公共面积
    //O 为坐标原点
    //注意返回值可能为负
    double areaCT(double r, P p1, P p2) {
        P[] qs = isCL(new P(0, 0), r, p1, p2);
        if (qs.length == 0) return r * r * rad(p1, p2) / 2;
        boolean b1 = p1.abs() > r + EPS, b2 = p2.abs() > r + EPS;
        if (b1 && b2) {
            if (p1.sub(qs[0]).dot(p2.sub(qs[0])) < EPS &&
                    p1.sub(qs[1]).dot(p2.sub(qs[1])) < EPS) {
                return (r * r * (rad(p1, p2) - rad(qs[0], qs[1])) +
                        qs[0].det(qs[1])) / 2;
            } else {
                return r * r * rad(p1, p2) / 2;
            }
        } else if (b1) {
            return (r * r * rad(p1, qs[0]) + qs[0].det(p2)) / 2;
        } else if (b2) {
            return (r * r * rad(qs[1], p2) + p1.det(qs[1])) / 2;
        } else {
            return p1.det(p2) / 2;
        }
    }
    //返回两点和原点形成的夹角
    //注意这两点都不能为原点
```

```
    double rad(P p1, P p2) {
        return Math.acos(p1.dot(p2) / p1.abs() / p2.abs());
    }
    //凸包
    //逆时针 不包含线上的点
    //如果需要包含线上的点 将 <= 0 改成 < 0
    //但是需要注意此时不能有重点
    P[] convexHull(P[] ps) {
        int n = ps.length, k = 0;
        if (n <= 1) return ps;
        Arrays.sort(ps);
        P[] qs = new P[n * 2];
        for (int i = 0; i < n; qs[k++] = ps[i++]) {
            while (k > 1 && qs[k - 1].sub(qs[k - 2]).det(
                    ps[i].sub(qs[k - 1])) < EPS) k--;
        }
        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--]) {
            while (k > t && qs[k - 1].sub(qs[k - 2]).det(
                    ps[i].sub(qs[k - 1])) < EPS) k--;
        }
        P[] res = new P[k - 1];
        System.arraycopy(qs, 0, res, 0, k - 1);
        return res;
    }
    // 按相对于 p0 的极角逆时针排序
    // 角度相同，则离 p0 距离更近的放在前面
    class CmpByAngle implements Comparator<P> {
        P p0;
        CmpByAngle(P p0) {
            this.p0 = p0;
        }
        @Override
        int compare(P o1, P o2) {
            double det = o1.sub(p0).det(o2.sub(p0));
            if (det != 0) return det > 0 ? -1 : 1;
            double dis = add(o1.sub(p0).abs2(),
                    -o2.sub(p0).abs2());
            if (dis != 0) return dis > 0 ? 1 : -1;
            return 0;
        }
    }
    P[] convexHullByAngle(P[] ps) {
        int n = ps.length, k = 0;
        if (n <= 1) return ps;
        for (int i = 1; i < n; i++) {
            if (ps[i].y < ps[0].y ||
                    ps[i].y == ps[0].y && ps[i].x < ps[0].x) {
                Algo.swap(ps, 0, i);
            }
        }
        Arrays.sort(ps, 1, n, new CmpByAngle(ps[0]));
        P[] qs = new P[n];
        for (int i = 0; i < n; qs[k++] = ps[i++]) {
            while (k > 1 && qs[k - 1].sub(qs[k - 2]).det(
                    ps[i].sub(qs[k - 1])) < EPS) k--;
        }
        return Arrays.copyOf(qs, k);
    }
    //凸多边形的切断
    //返回p1p2左侧凸包
    P[] convexCut(P[] ps, P p1, P p2) {
        int n = ps.length;
        ArrayList<P> res = new ArrayList<P>();
        for (int i = 0; i < n; i++) {
            int d1 = sig(p2.sub(p1).det(ps[i].sub(p1)));
            int d2 = sig(p2.sub(p1).det(ps[(i + 1) % n].sub(p1)));
            if (d1 >= 0) res.add(ps[i]);
            if (d1 * d2 < 0)
                res.add(isLL(p1, p2, ps[i], ps[(i + 1) % n]));
        }
        return res.toArray(new P[0]);
    }
    //点在多边形内外的判定
    //内部返回1，边上返回0，外部返回-1
    int contains(P[] ps, P q) {
        int n = ps.length;
        int res = -1;
        for (int i = 0; i < n; i++) {
            P a = ps[i].sub(q), b = ps[(i + 1) % n].sub(q);
            if (a.y > b.y) { P t = a; a = b; b = t; }
            if (a.y < EPS && b.y > EPS && a.det(b) > EPS) {
                res = -res;
```

```
            }
            if (Math.abs(a.det(b)) < EPS && a.dot(b) < EPS) return
0;
        }
        return res;
    }
    //凸多边形与外部点的距离
    double disConvexP(P[] ps, P q) {
        int n = ps.length;
        int left = 0, right = n;
        while (right - left > 1) {
            int mid = (left + right) / 2;
            if (in(ps[(left + n - 1) % n], ps[left], ps[mid],
                    ps[(mid + 1) % n], q)) {
                right = mid;
            } else {
                left = mid;
            }
        }
        return disSP(ps[left], ps[right % n], q);
    }
    boolean in(P p1, P p2, P p3, P p4, P q) {
        P o12 = p1.sub(p2).rot90();
        P o23 = p2.sub(p3).rot90();
        P o34 = p3.sub(p4).rot90();
        return in(o12, o23, q.sub(p2)) || in(o23, o34, q.sub(p3))
                || in(o23, p3.sub(p2), q.sub(p2))
                && in(p2.sub(p3), o23, q.sub(p3));
    }
    boolean in(P p1, P p2, P q) {
        return p1.det(q) > -EPS && p2.det(q) < EPS;
    }
    //凸多边形的直径
    //凸多边形上最远点的距离
    //O(n)
    double convexDiameter(P[] ps) {
        int n = ps.length;
        int is = 0, js = 0;
        for (int i = 1; i < n; i++) {
            if (ps[i].x > ps[is].x) is = i;
            if (ps[i].x < ps[js].x) js = i;
        }
        double maxD = ps[is].sub(ps[js]).abs();
        int i = is, j = js;
        do {
            if (ps[(i + 1) % n].sub(ps[i]).det(
                    ps[(j + 1) % n].sub(ps[j])) >= 0) {
                j = (j + 1) % n;
            } else {
                i = (i + 1) % n;
            }
            maxD = Math.max(maxD, ps[i].sub(ps[j]).abs());
        } while (i != is || j != js);
        return maxD;
    }
}
```

4. math

4.1. Num

```
class Num {
    Random rnd;
    List<Integer> primes;
    boolean[] isPrime;
    boolean millerRabin(BigInteger n, int times) {
        n = n.abs();
        if (n.compareTo(BigInteger.valueOf(2)) < 0) return false;
        if (n.equals(BigInteger.valueOf(2))) return true;
        if (!n.testBit(0)) return false;
        BigInteger q = n.subtract(BigInteger.ONE);
        int k = 0;
        while (q.mod(B.valueOf(2)).equals(Bi.ZERO)) {
            k++;
            q = q.shiftRight(1);
        }
        // n - 1 = 2^k * q (q は奇素数)
        // n が素数であれば、下記のいずれかを満たす
        // (i) a^q ≡ 1 (mod n)
        // (ii) a^q, a^2q,..., a^(k-1)q のどれかが n を法として-1
        // なので、逆に(i)(ii)いずれも満たしていない時は合成数と
        // 判定できる
        for (int i = 0; i < times; i++) {
            BigInteger a = new BigInteger(n.bitLength(),
                    rnd == null ? rnd = new Random() : rnd)
```

```
                    .abs().mod(n.sub(B.ONE)).add(B.ONE); //
                                   //1,..,n-1 からランダムに値を選ぶ
            BigInteger x = a.modPow(q, n);
            // (i)をチェック
            if (x.equals(BigInteger.ONE)) continue;
            // (ii)をチェック
            boolean found = false;
            for (int j = 0; j < k; j++) {
                if (x.equals(n.subtract(BigInteger.ONE))) {
                    found = true;
                    break;
                }
                x = x.multiply(x).mod(n);
            }
            if (found) continue;
            return false;
        }
        return true;
    }
    // ポラード・ロー素因数分解法
    BigInteger pollardRho(BigInteger n, BigInteger c) {
        BigInteger x = BigInteger.valueOf(2);
        BigInteger y = BigInteger.valueOf(2);
        BigInteger d = BigInteger.ONE;
        while (d.equals(BigInteger.ONE)) {
            x = x.multiply(x).mod(n).add(c);
            y = y.multiply(y).mod(n).add(c);
            y = y.multiply(y).mod(n).add(c);
            d = x.subtract(y).abs().gcd(n);
        }
        if (d.equals(n))
            return pollardRho(n, c.add(BigInteger.ONE));
        return d;
    }
    // 素数かどうか判定。大きければミラーラビンを使う
    boolean isPrime(BigInteger n) {
        if (isPrime != null && n.comTo(B.Of(isPrime.length)) < 0)
            return isPrime[n.intValue()];
        return millerRabin(n, 20);
    }
    boolean isPrime(long n) {
        return isPrime(BigInteger.valueOf(n));
    }
    // 素因数分解する。
    // 小さい数は用意した素数で試し割り、大きければポラード・ロー
    void factorize(B n, Map<B, Integer> factors) {
        if (isPrime(n)) {
            Num.inc(factors, n);
        } else {
            for (Integer prime : primes) {
                BigInteger p = BigInteger.valueOf(prime);
                while (n.mod(p).equals(BigInteger.ZERO)) {
                    Num.inc(factors, p);
                    n = n.divide(p);
                }
            }
            if (!n.equals(BigInteger.ONE)) {
                if (isPrime(n)) {
                    Num.inc(factors, n);
                } else {
                    BigInteger d = pollardRho(n, BigInteger.ONE);
                    factorize(d, factors);
                    factorize(n.divide(d), factors);
                }
            }
        }
    }
    boolean[] primeTable(int n, List<Integer> primes) {
        Num.primes = primes;
        isPrime = new boolean[n + 1];
        Arrays.fill(isPrime, true);
        isPrime[0] = isPrime[1] = false;
        /*
        for (int i = 2; i <= n; i++) {
            if (isPrime[i]) primes.R(i);
            for (int p : primes) {
                if (i > n / p) break;
                isPrime[i * p] = false;
                if (i % p == 0) break;
            }
        }*/
        for (int i = 2; i <= n; i++) {
```

```
            if (isPrime[i]) {
                primes.add(i);
                for (int j = i + i; j <= n; j += i) {
                    isPrime[j] = false;
                }
            }
        }
        return isPrime;
    }
    long phi(long n) {
        long ans = n;
        for (long i : primes) {
            if (i * i > n) break;
            if (n % i == 0) {
                ans = ans / i * (i - 1);
                while (n % i == 0) n /= i;
            }
        }
        if (n > 1) ans = ans / n * (n - 1);
        return ans;
    }
    int[] phiTable(int n) {
        int[] phi = new int[n + 1];
        phi[1] = 1;
        for (int i = 2; i <= n; i++)
            if (phi[i] == 0) {
                for (int j = i; j <= n; j += i) {
                    if (phi[j] == 0) phi[j] = j;
                    phi[j] = phi[j] / i * (i - 1);
                }
            }
        return phi;
    }
    long combination(int n, int m, long mod) {
        if (m < 0 || m > n) return 0;
        if (2 * m > n) m = n - m;
        long res = 1;
        for (int i = n - m + 1; i <= n; i++)
            res = res * i % mod;
        return res * B.Of(factorial(m, mod))
                .modInv(B.Of(mod)).longValue() % mod;
    }
    long[][] combinationTable(int n) {
        long[][] res = new long[n + 1][n + 1];
        for (int i = 0; i <= n; i++) {
            res[i][0] = 1;
            for (int j = 1; j <= i; j++)
                res[i][j] = res[i - 1][j - 1] + res[i - 1][j];
        }
        return res;
    }
    long[] combinationRowTable(int n, long mod) {
        long[] res = invFactorialTable(n, mod);
        res[0] = 1;
        for (int i = 1; i <= n; i++) {
            res[i] = res[i - 1] * (n - i + 1) % mod * res[i] % mod;
        }
        return res;
    }
    long[] invFactorialTable(int n, long mod) {
        long[] res = new long[n + 1];
        if (n >= 1) res[1] = 1;
        for (int i = 2; i <= n; i++)
            res[i] = (mod - mod / i * res[((int) (mod % i))] % mod)
                    % mod;
        return res;
    }
    long pow(long p, long e, long mod) {
        long res = 1;
        while (e != 0) {
            if ((e & 1L) != 0) res = res * p % mod;
            p = mul(p, p, mod);
//            p = p * p % mod;
            e >>= 1;
        }
        return res;
    }
    long invS(long a, long mod) {
        if (a == 1) return 1;
        return invS(mod % a, mod) * (mod - mod / a) % mod;
    }
    int gcd(int a, int b) {
```

```
        while (b != 0) {
            int c = a;
            a = b;
            b = c % b;
        }
        if (a < 0) a = -a;
        return a;
    }
    //把 n 的约数的莫比乌斯值用 map 形式的返回。O(sqrt n)
    Map<Long, Integer> moebius(long n) {
        Map<Long, Integer> res = new TreeMap<Long, Integer>();
        List<Long> primes = primeFactors(n);
        int m = primes.size();
        for (int i = 0; i < (1 << m); i++) {
            int mu = 1;
            long d = 1;
            for (int j = 0; j < m; j++) {
                if ((i & (1 << j)) != 0) {
                    mu *= -1;
                    d *= primes.get(j);
                }
            }
            res.put(d, mu);
        }
        return res;
    }
    int[] moebiusTable(int n) {
        boolean[] check = new boolean[n + 1];
        List<Integer> primes = new ArrayList<Integer>();
        int[] mu = new int[n + 1];
        mu[1] = 1;
        for (int i = 2; i <= n; i++) {
            if (!check[i]) {
                primes.add(i);
                mu[i] = -1;
            }
            for (int p : primes) {
                if (i * p > n) break;
                check[i * p] = true;
                if (i % p == 0) {
                    mu[i * p] = 0;
                    break;
                } else {
                    mu[i * p] = -mu[i];
                }
            }
        }
        return mu;
    }
    BigInteger sqrt(String theNumber) {
        int length = theNumber.length(), i;
        BigInteger res = BigInteger.ZERO;
        BigInteger twenty = BigInteger.valueOf(20);
        BigInteger t, x = B.ZERO, v, few = B.ZERO;
        BigInteger hg = BigInteger.valueOf(100);
        int pos = 2 - length % 2;
        String tmpString = theNumber.substring(0, pos);
        while (true) {
            v = few.mul(hg).add(B.Of(Integer.parseInt(tmpString)));
            if (res.compareTo(BigInteger.ZERO) == 0) i = 9;
            else i = v.divide(res.multiply(twenty)).intValue();
            for (; i >= 0; i--) {
                t = res.mul(twenty).add(B.Of(i)).mul(B.Of(i));
                if (t.compareTo(v) <= 0) {
                    x = t;
                    break;
                }
            }
            res = res.mul(B.TEN).add(B.Of(i));
            few = v.subtract(x);
            pos++;
            if (pos > length) break;
            tmpString = theNumber.substring(pos - 1, ++pos);
        }
        return res;
    }
    Map<Integer, int[]> fact;
    int e;
    int[] modFact(int n, int p) {
        return new int[] { modFactRec(n, p), e };
    }
    int modFactRec(int n, int p) {
```

```
        e = 0;
        if (n == 0) return 1;
        int res = modFactRec(n / p, p);
        e += n / p;
        if (n / p % 2 != 0) return res * (p - fact(n % p, p)) % p;
        return res * fact(n % p, p) % p;
    }
    int fact(int n, int p) {
        if (fact == null) fact = new HashMap<Integer, int[]>();
        if (!fact.containsKey(p)) {
            int[] f = new int[p];
            f[0] = 1;
            for (int i = 1; i < p; i++)
                f[i] = (int) ((long) f[i - 1] * i % p);
            fact.put(p, f);
        }
        return fact.get(p)[n];
    }
    // C(n, k) % p
    int modComb(int n, int k, int p) {
        if (n < 0 || k < 0 || n < k) return 0;
        int[] a1 = modFact(n, p), a2 = modFact(k, p),
              a3 = modFact(n - k, p);
        if (a1[1] > a2[1] + a3[1]) return 0;
        return a1[0] * (int) inv(a2[0] * a3[0] % p, p) % p;
    }
}
```

4.2. Matrix

```
class Matrix {
    int[][] mul(int[][] a, int[][] b, int mod) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int k = 0; k < n; k++) if (a[i][k] != 0) {
                for (int j = 0; j < n; j++) {
                    c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % mod;
                }
            }
        }
        return c;
    }
    int[][] pow(int[][] a, int b, int mod) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++)
            c[i][i] = 1;
        while (b > 0) {
            if ((b & 1) != 0)
                c = mul(c, a, mod);
            a = mul(a, a, mod);
            b >>>= 1;
        }
        return c;
    }
    long[][] solutionSpace(long[][] A, long[] b, long mod) {
        int n = A.length, m = A[0].length;
        BigInteger MOD = BigInteger.valueOf(mod);
        long[][] a = new long[n][m + 1];
        for (int i = 0; i < n; i++) {
            System.arraycopy(A[i], 0, a[i], 0, m);
            a[i][m] = b[i];
        }
        int[] id = new int[n + 1]; // 第 i 行的第一个非零元 1
                                   // 所在的位置是 id[i]
        Arrays.fill(id, -1);
        int pi = 0; // 矩阵 A 的秩
        for (int pj = 0; pi < n && pj < m; pj++) {
            for (int i = pi + 1; i < n; i++) {
                if (Math.abs(a[i][pj]) > Math.abs(a[pi][pj])) {
                    long[] t = a[i];
                    a[i] = a[pi];
                    a[pi] = t;
                }
            }
            if (Math.abs(a[pi][pj]) < EPS) // 当前列已经全零
                continue;
            long inv = B.Of(a[pi][pj]).modInv(MOD).longValue();
            for (int j = 0; j <= m; j++) // 化主元为 1，可以优化
                a[pi][j] = (a[pi][j] * inv) % mod;
            for (int i = 0; i < n; i++)
                if (i != pi) {
                    long d = a[i][pj];
                    for (int j = 0; j <= m; j++) // 化当前列为 0
                        a[i][j] = (a[i][j] - d * a[pi][j] % mod)
                                % mod;
                }
            id[pi++] = pj;
        }
        for (int i = pi; i < n; i++)
            if (Math.abs(a[i][m]) > EPS) // 增广矩阵的秩更大，无解
                return null;
        long[][] X = new long[1 + m - pi][m];
        for (int j = 0, k = 0; j < m; j++) {
            if (id[k] == j)
                X[0][j] = a[k++][m];
            else {
                for (int i = 0; i < k; i++)
                    X[1 + j - k][id[i]] = -a[i][j];
                X[1 + j - k][j] = 1;
            }
        }
        return X;
    }
    boolean[][] solutionSpace(boolean[][] A, boolean[] b) {
        int n = A.length, m = A[0].length;
        boolean[][] a = new boolean[n][m + 1];
        for (int i = 0; i < n; i++) {
            System.arraycopy(A[i], 0, a[i], 0, m);
            a[i][m] = b[i];
        }
        int[] id = new int[n + 1]; // 第 i 行的第一个非零元 1
                                   // 所在的位置是 id[i]
        Arrays.fill(id, -1);
        int pi = 0; // 矩阵 A 的秩
        for (int pj = 0; pi < n && pj < m; pj++) {
            for (int i = pi + 1; i < n; i++) {
                if (a[i][pj] && !a[pi][pj]) {
                    boolean[] t = a[i];
                    a[i] = a[pi];
                    a[pi] = t;
                }
            }
            if (!a[pi][pj]) // 当前列已经全零
```

```
                continue;
            for (int i = 0; i < n; i++)
                if (i != pi) {
                    boolean d = a[i][pj];
                    for (int j = 0; j <= m; j++) // 化当前列为 0
                        a[i][j] ^= d & a[pi][j];
                }
            id[pi++] = pj;
        }
        for (int i = pi; i < n; i++)
            if (a[i][m]) // 增广矩阵的秩更大，无解
                return null;
        boolean[][] X = new boolean[1 + m - pi][m];
        for (int j = 0, k = 0; j < m; j++) {
            if (id[k] == j)
                X[0][j] = a[k++][m];
            else {
                for (int i = 0; i < k; i++)
                    X[1 + j - k][id[i]] = a[i][j];
                X[1 + j - k][j] = true;
            }
        }
        return X;
    }
}
```

5. datetime

5.1. DateTime

```
class DateTime {
    int[] ds = {
            0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
    int days(int y, int m, int d) {
        m = (m + 9) % 12;
        y = y - m / 10;
        return 365 * y + y / 4 - y / 100 + y / 400 +
                (m * 306 + 5) / 10 + (d - 1);
    }
    int[] nextDay(int y, int m, int d) {
        if (d < ds[m]) return new int[] { y, m, d + 1 };
        if (d == 28 && m == 2 && isLeapYear(y))
            return new int[] { y, 2, 29 };
        m++;
```

```
        if (m == 13) {
            m = 1;
            y++;
        }
        return new int[] { y, m, 1 };
    }
    boolean isLeapYear(int year) {
        return new GregorianCalendar().isLeapYear(year);
    }
}
```