1 数据结构

1.1 区间增加区间求和

```
class Seg {
    BIT dif, pre;

    Seg(int n) {
        dif = new BIT(n);
        pre = new BIT(n);
    }

    void add(int s, int t, long v) {
        dif.add(s, v);
        dif.add(t, -v);
        pre.add(s, v * s);
        pre.add(t, -v * t);
    }

    long sum(int s, int t) {
        if (s > 0) return sum(0, t) - sum(0, s);
        return dif.sum(0, t) * t - pre.sum(0, t);
    }
}
```

1.2 线性变换线段树

```
class Seg {
    int N;
    long[] is, mul, add;

    Seg(int n) {
        N = Integer.highestOneBit(n) << 1;
        is = new long[N * 2];
        // 初始化过程 根据需要修改
        for (int i = 0; i < n; i++)
            is[N + i] = in.nextLong();
        for (int i = N - 1; i > 0; i--)
            is[i] = merge(is[i * 2], is[i * 2 + 1]);
        mul = new long[N * 2];
        add = new long[N * 2];
        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 mul[o], add[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
            is[o] = m * is[o] + a * (R - L); // 根据维护信息修改
        } 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 mul[o], add[o]
            mul[o] = 1;
            add[o] = 0;
            is[o] = merge(is[o * 2], is[o * 2 + 1]);
        }
    }

    long query(int s, int t) {
        update(s, t, 1, 0);
        long res = 0; // 初始化 根据维护信息修改
        while (0 < s && s + (s & -s) <= t) {
            int i = (N + s) / (s & -s);
            res = merge(res, is[i]);
            s += s & -s;
        }
        while (s < t) {
            int i = (N + t) / (t & -t) - 1;
            res = merge(res, is[i]);
            t -= t & -t;
        }
        return res;
    }

    long merge(long a, long b) {
        return a + b; // 根据维护信息修改
    }

    // 后面是另一种 update 实现，会慢一点
    void update(int o, int L, int R) {
        if (s <= L && R <= t) {
            push(m, a, o);
        } else {
            pushdown(o);
            int M = (L + R) / 2;
            if (s < M)
                update(o * 2, L, M);
            if (t > M)
                update(o * 2 + 1, M, R);
```

```
            is[o] = merge(is[o * 2], is[o * 2 + 1]);
        }
    }

    void pushdown(int o) {
        push(mul[o], add[o], o * 2);
        push(mul[o], add[o], o * 2 + 1);
        mul[o] = 1;
        add[o] = 0;
    }

    long size(int o) {
        return N / Integer.highestOneBit(o);
    }

    void push(long m, long a, int o) {
        is[o] = m * is[o] + size(o) * a; // 根据维护信息修改
        mul[o] *= m;
        add[o] = m * add[o] + a;
    }

}
```

1.3 Treap

```
class T {
    int key, size;
    double p;
    T left, right;

    public 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, random(), NULL, NULL);
    }
}

T change(T t, T left, T right) {
    t.size = left.size + right.size + 1;
    t.left = left;
    t.right = right;
    return t;
}

T[] splitSize(T t, int size) {
    T[] res;
    if (size <= 0) {
        res = new T[] { NULL, t };
    } else if (size <= t.left.size) {
        res = splitSize(t.left, size);
        res[1] = change(t, res[1], t.right);
    } else {
        res = splitSize(t.right, size - t.left.size - 1);
        res[0] = change(t, 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.key) {
        res = splitKey(t.left, key);
        res[1] = change(t, res[1], t.right);
    } else {
        res = splitKey(t.right, key);
        res[0] = change(t, t.left, res[0]);
    }
    return res;
}

void print(T t, String indent) {
    if (t != NULL) {
        print(t.right, indent + "     ");
        out.printf("%3d%3d%n", t.key, t.size);
        print(t.left, indent + "     ");
    }
    if (indent.length() == 0)
        out.println("----------------------------------");
}

T merge(T t1, T t2) {
    if (t1 == NULL) return t2;
    if (t2 == NULL) return t1;
    if (t1.p < t2.p)
        return change(t1, t1.left, merge(t1.right, t2));
    return change(t2, merge(t1, t2.left), t2.right);
}

T NULL = new T(0, 0, 0, null, null);
```

1.4 Hash

```
public class Hash {
    public static final long BASE = (long) (1e9 + 7);
    public static long[] ps;
    public Hash(int n) {
        ps = Num.powerTable(BASE, n + 1, -1);
    }
```

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

1.5 MatMin

```
public class MatMin {
    public SegMinC[] ss;
    public int N;
    public int M;
    public 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);
        }
    }
    public 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) {
            if (ss[x].is[M + y] > val) ss[x].update(y, 0, val);
            else break;
        }
        return val;
    }

    public 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;
    }
}
```

1.6 SegMinC

```
public class SegMinC {
    public int[] is;
    public int N;

    public SegMinC(int n) {
        N = Integer.highestOneBit(n) << 1;
        is = new int[N * 2];
        Arrays.fill(is, Integer.MAX_VALUE);
    }

    public 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) {
            if (is[k] > val) is[k] = val;
            else break;
        }
        return val;
    }

    public 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;
    }
}
```

1.7 MatSum

```
public class MatSum {
    BIT[] bs;
    public MatSum(int row, int col) {
        bs = new BIT[row + 1];
        for (int i = 0; i < bs.length; i++) {
            bs[i] = new BIT(col);
        }
    }

    public void add(int x, int y, int val) {
        for (int i = x + 1; i < bs.length; i += i & -i) {
            bs[i].add(y, val);
        }
    }

    public 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 数学

2.1 矩阵快速幂

```
long M = 1000000007;

long[][] mul(long[][] a, long[][] b) {
     int n = a.length;
     long[][] c = new long[n][n];
     for (int i = 0; i < n; i++) {
          for (int k = 0; k < n; k++) {
               for (int j = 0; j < n; j++) {
                    c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % M;
               }
          }
     }
     return c;
}

long[][] pow(long[][] a, long b) {
     int n = a.length;
     long[][] c = new long[n][n];
     for (int i = 0; i < n; i++)
          c[i][i] = 1;
     while (b > 0) {
          if ((b & 1) != 0)
               c = mul(c, a);
          a = mul(a, a);
          b >>>= 1;
     }
     return c;
}
```

2.2 无限精度分数类

```
import java.math.BigInteger;
import static java.math.BigInteger.*;

class Rational implements Comparable<Rational> {
     static final Rational R0 = new Rational(ZERO, ONE),
               R1 = new Rational(ONE,ONE);
     BigInteger num, 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.multiply(r.den).add(
                    r.num.multiply(den)), den.multiply(r.den));
     }

     Rational sub(Rational r) {
          return new Rational(num.multiply(r.den).subtract(
                    r.num.multiply(den)), den.multiply(r.den));
     }
```

```
    Rational mul(Rational r) {
        return new Rational(num.multiply(r.num),
                den.multiply(r.den));
    }

    Rational div(Rational r) {
        return new Rational(num.multiply(r.den),
                den.multiply(r.num));
    }

    int signum() {
        return num.signum();
    }

    Rational pow(int b) {
        BigInteger n = ONE, d = 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);
    }

    public int compareTo(Rational o) {
        return  (num.multiply(o.den).compareTo(
                o.num.multiply(den)));
    }
}
```

2.3 大数开方

```
// 传参要求是正数，返回的是它的算术平方根的整数部分
BigInteger sqrt(String theNumber) {
    int length = theNumber.length(), i;
    BigInteger res = BigInteger.ZERO;
    BigInteger twenty = BigInteger.valueOf(20);
    BigInteger t, x = BigInteger.ZERO, v, few = BigInteger.ZERO;
    BigInteger hg = BigInteger.valueOf(100);
    String tmpString = null;
    int pos = 2 - length % 2;
    tmpString = theNumber.substring(0, pos);
    while (true) {
        v = few.multiply(hg).add(
                BigInteger.valueOf(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.multiply(twenty).add(BigInteger.valueOf(i))
                    .multiply(BigInteger.valueOf(i));
            if (t.compareTo(v) <= 0) {
                x = t;
                break;
            }
        }
        res = res.multiply(BigInteger.TEN).add(
                BigInteger.valueOf(i));
        few = v.subtract(x);
        pos++;
        if (pos > length) break;
        tmpString = theNumber.substring(pos - 1, ++pos);
    }
    return res;
}
```

2.4 日期转天数

```
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);
}
```

3 图论

3.1 Dijkstra

```
void dijkstra(V s) {
    PriorityQueue<E> que = new PriorityQueue<E>();
    s.min = 0;
    que.offer(new E(s, 0));
    while (!que.isEmpty()) {
        E crt = que.poll();
        if (crt.cost > crt.to.min)
            continue;
        for (E e : crt.to.es) {
            if (crt.cost + e.cost < e.to.min) {
                e.to.min = crt.cost + e.cost;
                que.offer(new E(e.to, e.to.min));
            }
        }
    }
}

int INF = 1 << 29;

class V {
    ArrayList<E> es = new ArrayList<E>();
    int min = INF;
```

```
    void add(V to, int cost) {
        es.add(new E(to, cost));
    }
}

class E implements Comparable<E> {
    V to;
    int cost;

    E(V to, int cost) {
        this.to = to;
        this.cost = cost;
    }

    public int compareTo(E o) {
        return cost - o.cost;
    }
}
```

3.2 Spfa

```
void spfa(V s) {
    Queue<V> que = new LinkedList<V>();
    s.min = 0;
    que.offer(s);
    while (!que.isEmpty()) {
        V crt = que.poll();
        crt.inQue = false;
        for (E e : crt.es) {
            if (crt.min + e.cost < e.to.min) {
                e.to.min = crt.min + e.cost;
                if (!e.to.inQue) {
                    e.to.inQue = true;
                    que.offer(e.to);
                }
            }
        }
    }
}

int INF = 1 << 29;

class V {
    ArrayList<E> es = new ArrayList<E>();
    int min = INF;
    boolean inQue = false;

    void add(V to, int cost) {
        es.add(new E(to, cost));
    }
}

class E implements Comparable<E> {
    V to;
    int cost;

    E(V to, int cost) {
        this.to = to;
        this.cost = cost;
    }

    public int compareTo(E o) {
        return cost - o.cost;
    }
}
```

4 补充

4.1 后缀数组的 indexSort 函数

```
int[] indexSort(int[] is) {
    int[] c = new int[128];
    for (int i : is) c[i]++;
    for (int i = 1; i < 128; i++) c[i] += c[i - 1];
    int n = is.length;
    int[] si = new int[n];
    for (int i = n - 1; i >= 0; i--)
        si[--c[is[i]]] = i;
    return si;
}
```

4.2 模板

```
public class Main {
    Scanner in = new Scanner(System.in);
    PrintWriter out = new PrintWriter(System.out);

    public static void main(String[] args) {
        try {
            System.setIn(new FileInputStream("./../in"));
        } catch (Throwable e) {
        }
        Main main = new Main();
        main.run();
        main.out.close();
    }
}
```