

目次

1	Template	2
1.1	簡単なテンプレ	2
1.2	自作 Scanner	2
2	DataSetructure	2
2.1	Union-Find	2
2.2	重み付き Union-Find	3
2.3	SegmentTree	3
2.4	遅延評価 SegmentTree	4
2.5	BIT(Binary-Indexed-Tree)	4
3	Graph	5
3.1	隣接行列	5
3.2	隣接リスト	5
3.3	閉路検出	5
3.4	トポロジカルソート	6
4	Math	6
4.1	GCD, LCM	6
4.2	有理数	7
4.3	多倍長有理数	7
5	Mod	8
5.1	累乗 (mod m)	8
5.2	逆元 (mod p)	8
5.3	逆元列挙 (mod p)	8
5.4	逆元 (mod m)	9
5.5	中国剰余定理	9
5.6	Rolling Hash	9
6	2DimRealGeometry	10
6.1	基本データ構造	10
7	2DimIntGeometry	12
7.1	2次元の点	12

7.2	2次元の直線	12
8	String	13
8.1	Shift And	13
9	Puzzle	13
9.1	Nim(山 N 個, 制限無し)	13

1 Template

1.1 簡単なテンプレ

入力が少ない ($n < 100000$) 場合は Java 標準の Scanner で大丈夫.

Source Code

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void debug(Object ... objs){
5         System.out.println(Arrays.toString(objs));
6     }
7
8     public static void main(String[] args){
9         Scanner sc = new Scanner(System.in);
10        // input here
11        sc.close();
12    }
13 }
```

1.2 自作 Scanner

入が多い場合は, Java 標準の Scanner では間に合わない. なので, 正規表現を使わない Scanner を自作する必要がある.

Source Code

```
1 public static class Scanner {
2     private BufferedReader br;
3     private StringTokenizer tok;
4
5     public Scanner(InputStream is) throws IOException {
6         br = new BufferedReader(new InputStreamReader(is));
7     }
8
9     private void getLine() throws IOException {
10        while (!hasNext()) { tok = new StringTokenizer(br.readLine()); }
11    }
12
13    private boolean hasNext() {
14        return tok != null && tok.hasMoreTokens();
15    }
16
17    public String next() throws IOException {
18        getLine(); return tok.nextToken();
19    }
20
21    public int nextInt() throws IOException {
22        return Integer.parseInt(next());
23    }
24    // 他の nextXXXもXXX.parseXXX()メソッドを使って作れるので省略
25
26    public void close() throws IOException {
27        br.close();
28    }
29 }
```

2 DataStructure

2.1 Union-Find

素集合管理用のデータ構造. 超頻出データ構造.

計算量

経路圧縮 + ランク併合 で アッカーマン関数の逆関数 (大体 4 くらい)

Source Code

```
1 public static class UnionFind{
2     int[] par; //
3
4     public UnionFind(int n){
5         par = new int[n];
6         for(int i = 0; i < n; i++){ par[i] = -1; }
7     }
8
9     public int find(int x){
10        if(par[x] < 0){
11            return x;
12        }else{
13            return par[x] = find(par[x]);
14        }
15    }
16
17    public boolean union(int x, int y){
18        x = find(x);
19        y = find(y);
20
21        if(x != y){
22            if(par[y] < par[x]) { // 多い方が根になるようにスワップする.
23                int tmp = x; x = y; y = tmp;
24            }
25            par[x] += par[y];
26            par[y] = x;
27            return true;
28        }else{
29            return false;
30        }
31    }
32
33    public boolean same(int x, int y){
34        return find(x) == find(y);
35    }
36
37    public int size(int x){
38        return -par[find(x)];
39    }
40 }
```

2.2 重み付き Union-Find

UnionFind で同時にグループ内の重みを管理する。同一グループで比較可能な値がオンラインで与えられる場合に使える。

計算量

経路圧縮 + ランク併合 を使ってるため、アッカーマン関数の逆関数になる。

Source Code

```

1 public static class WeightedUnionFind{
2     int[] par; // 親の番号
3     int[] ws; // 親との重みの差
4
5     public WeightedUnionFind(int n){
6         par = new int[n];
7         ws = new int[n];
8         for(int i = 0; i < n; i++){ par[i] = -1; }
9     }
10
11     public int find(int x){
12         if(par[x] < 0){
13             return x;
14         }else{
15             final int parent = find(par[x]);
16             ws[x] += ws[parent];
17             return par[x] = parent;
18         }
19     }
20
21     public int weight(int x){
22         find(x);
23         return ws[x];
24     }
25
26     public boolean union(int x, int y, int w){ // x <- (w) - y (x + w = y)
27         w += weight(x);
28         w -= weight(y);
29         x = find(x); y = find(y);
30
31         if(x != y){
32             if(par[y] < par[x]) { // 多い方が根になるようにスワップする。
33                 int tmp = x; x = y; y = tmp; w = -w;
34             }
35             par[x] += par[y]; par[y] = x;
36             ws[y] = w;
37             return true;
38         }else{
39             return false;
40         }
41     }
42
43     public boolean same(int x, int y){
44         return find(x) == find(y);
45     }
46
47     public Integer diff(int x, int y){ // x - y を求める。比較不能なら null.
48         if(!same(x, y)){ return null; }
49         return this.weight(x) - this.weight(y);
50     }
51     // size() は UnionFind と同じなので省略。
52 }

```

2.3 SegmentTree

範囲に関わるクエリを高速に処理するデータ構造。

計算量

- add(k) : $O(\log n)$ ※単一の値の更新
- get(s,t) : $O(\log n)$ ※範囲の値の計算

Source Code

```

1 public static class SegTree{
2     int n;
3     long[] dat;
4
5     public SegTree(int n_) {
6         int n = 1;
7         while(n < n_){
8             n *= 2;
9         }
10
11         this.n = n;
12         dat = new long[this.n * 2 - 1];
13         for(int i = 0; i < this.n * 2 - 1; i++){
14             dat[i] = 0;
15         }
16     }
17
18     public long calc(long fst, long snd){
19         return fst + snd;
20     }
21
22     public void update(int k, long a){
23         k += n - 1;
24         dat[k] = a;
25
26         while(k > 0){
27             k = (k - 1) / 2;
28             dat[k] = calc(dat[k * 2 + 1], dat[k * 2 + 2]);
29         }
30     }
31
32     public long query(int a, int b, int k, int l, int r){
33         if(r <= a || b <= l){
34             return 0;
35         }else if(a <= l && r <= b){
36             return dat[k];
37         }else {
38             return calc(query(a, b, k * 2 + 1, l, (l + r) / 2), query(a, b, k
39                             * 2 + 2, (l + r) / 2, r));
40         }
41     }
42
43     public long query(int a, int b){
44         return query(a, b, 0, 0, n);
45     }
46 }

```

2.4 遅延評価 SegmentTree

SegmentTree では区間に対する更新に $O(n \log n)$ かかってしまう。

計算量

- add(s,t): $O(\log n)$ ※範囲の値の更新
- get(s,t): $O(\log n)$ ※範囲の値の取得

Source Code

```

1 public static class SegTree{
2     int n;
3     long[] dat, lazy;
4
5     public SegTree(int n_) {
6         int n = 1;
7         while(n < n_){ n *= 2;} this.n = n;
8         dat = new long[this.n * 2 - 1];
9         lazy = new long[this.n * 2 - 1];
10        for(int i = 0; i < this.n * 2 - 1 ; i++){
11            dat[i] = 0; lazy[i] = 0;
12        }
13    }
14
15    private void lazy_evaluate_node(int k, int a, int b){
16        dat[k] += lazy[k] * (b - a);
17        if(k < n - 1){
18            lazy[2 * k + 1] += lazy[k]; lazy[2 * k + 2] += lazy[k];
19        }
20        lazy[k] = 0;
21    }
22
23    public void update_node(int k){
24        dat[k] = dat[2 * k + 1] + dat[2 * k + 2];
25    }
26
27    public void update(long v, int a, int b, int k, int l, int r){
28        lazy_evaluate_node(k, l, r);
29
30        if(r <= a || b <= l){ return; }
31        }else if(a <= l && r <= b){
32            lazy[k] += v; lazy_evaluate_node(k, l, r);
33        }else {
34            update(v, a, b, k * 2 + 1, l, (l + r) / 2);
35            update(v, a, b, k * 2 + 2, (l + r) / 2, r);
36            update_node(k);
37        }
38    }
39
40    public long get(int a, int b, int k, int l, int r){
41        lazy_evaluate_node(k, l, r);
42
43        if(r <= a || b <= l){ return 0; }
44        }else if(a <= l && r <= b){ return dat[k]; }
45        }else {
46            long v1 = get(a, b, k * 2 + 1, l, (l + r) / 2);
47            long v2 = get(a, b, k * 2 + 2, (l + r) / 2, r);
48            update_node(k); return v1 + v2;
49        }
50    }
51
52    public int size(){ return this.n; }
53 }
```

2.5 BIT(Binary-Indexed-Tree)

累積値を計算する事に特化したデータ構造. 空間計算量 $O(N)$ であり, 元のデータと同じ量のメモリ量で構築出来る.

計算量

- add(k) : $O(\log n)$ ※単一の値の更新
- sum(s,t): $O(\log n)$ ※範囲の和の計算

Source Code

```

1 public static class BIT{
2     int[] dat;
3
4     public BIT(int n){
5         dat = new int[n + 1];
6     }
7
8     public void add(int k, int a){ // k : 0-indexed
9         for(int i = k + 1; i < dat.length; i += i & -i){
10             dat[i] += a;
11         }
12     }
13
14     public int sum(int s, int t){ // [s, t)
15         if(s > 0) return sum(0, t) - sum(0, s);
16
17         int ret = 0;
18         for(int i = t; i > 0; i -= i & -i) {
19             ret += dat[i];
20         }
21         return ret;
22     }
23
24     public int get(int k){ // k : 0-indexed
25         int p = Integer.highestOneBit(dat.length - 1);
26         for(int q = p; q > 0; q >>= 1, p |= q){
27             if( p >= dat.length || k < dat[p]) p ^= q;
28             else k -= dat[p];
29         }
30         return p;
31     }
32 }
```

3 Graph

3.1 隣接行列

Source Code

```
1 public static final long INF = Long.MAX_VALUE / 2 - 1;
2
3 public static long[][] init_adj(final int n){
4     long[][] ret = new long[n][n];
5     for(int i = 0; i < n; i++){
6         for(int j = 0; j < n; j++){
7             ret[i][j] = i == j ? 01 : INF;
8         }
9     }
10    return ret;
11 }
```

3.2 隣接リスト

Source Code

```
1 public static ArrayList<Integer, LinkedHashMap<Integer, Long>> init_adj(final
2     int n){
3     ArrayList<Integer, LinkedHashMap<Integer, Long>> ret =
4         new ArrayList<Integer, LinkedHashMap<Integer, Long>>();
5     for(int i = 0; i < n; i++){
6         ret.add(new LinkedHashMap<Integer, Long>());
7     }
8     return ret;
9 }
```

3.3 閉路検出

閉路検出は、DFS の探索で探索中に二度探索する部分があるかで判定する。

計算量

Source Code

```
1 public static final int unvisited = 0;
2 public static final int visiting = 1;
3 public static final int visited = 2;
4
5 public static boolean dfs(int node, boolean[][] adj, int[] state){
6     state[node] = visiting;
7
8     for(int i = 0; i < adj.length; i++){
9         if(!adj[node][i]){ continue; }
10        else if(state[i] == unvisited){
11            if(!dfs(i, adj, state)){ //
12                state[node] = visited; return false;
13            }
14        }else if(state[i] == visiting){ //cycle!
15            return false;
16        }
17    }
18
19    state[node] = visited;
20    return true;
21 }
22
23 public static boolean find_cycle(boolean[][] adj){
24     int[] state = new int[adj.length];
25     for(int i = 0; i < adj.length; i++){
26         state[i] = unvisited;
27     }
28     for(int i = 0; i < adj.length; i++){
29         if(state[i] == unvisited){
30             if(!dfs(i, adj, state)){
31                 return false;
32             }
33         }
34     }
35     return true;
36 }
```

3.4 トポロジカルソート

閉路検出は, DFS の探索で帰りがけ順に見れば良い.

計算量

Source Code

```
1 public static final int unvisited = 0;
2 public static final int visiting = 1;
3 public static final int visited = 2;
4
5 public static boolean dfs(int node, boolean[][] adj, int[] state, LinkedList<
6     Integer> list){
7     state[node] = visiting;
8
9     for(int i = 0; i < adj.length; i++){
10         if(!adj[node][i]){ continue; }
11         else if(state[i] == unvisited){
12             if(!dfs(i, adj, state)){ //
13                 state[node] = visited; return false;
14             }
15         }else if(state[i] == visiting){ //cycle!
16             return false;
17         }
18     }
19     state[node] = visited;
20     list.addFront(node);
21     return true;
22 }
23
24 public static boolean topological_sort(long[][] ad, LinedList<Integer> list){
25     int[] state = new int[ad.length];
26     for(int i = 0; i < ad.length; i++){
27         state[i] = unvisited;
28     }
29     for(int i = 0; i < ad.length; i++){
30         if(state[i] == unvisited){
31             if(!dfs(i, ad, state, list)){
32                 return false;
33             }
34         }
35     }
36     return true;
37 }
```

4 Math

4.1 GCD, LCM

基本的な算術用の関数. いつもお世話になる.

計算量

軽い. $O(\log b)$ くらい

Source Code

```
1 public static long gcd(long a, long b){
2     return b == 0 ? a : gcd(b, a % b);
3 }
4 public static long lcm(long a, long b){
5     return a / gcd(a, b) * b;
6 }
```

4.2 有理数

有理数を表現するクラス. long なので int の範囲では誤差無しとして扱える.

Source Code

```

1 // equalsメソッドは eclipse等で生成するなりして作ること
2 public static class Rational implements Comparable<Rational> {
3     public static final Rational ZERO = new Rational(0);
4     public static final Rational ONE = new Rational(1);
5     public static final Rational NaN = new Rational(1, 0){
6         public String toString(){ return "NaN"; }};
7
8     private long nom, denom;
9
10    public Rational(long nom, long denom) {
11        if(nom == 0){ this.nom = 0; this.denom = 1; }
12        else{ final long gcd = inner_gcd(nom, denom);
13            this.nom = nom / gcd; this.denom = denom / gcd;
14            if(this.nom * this.denom < 0){
15                this.nom = -Math.abs(this.nom);
16                this.denom = Math.abs(this.denom);
17            }}
18    public Rational(long num) { this(num, 1); }
19    public long get_nom(){ return this.nom; }
20    public long get_denom(){ return this.denom; }
21    private static long inner_gcd(long a, long b){
22        return b == 0 ? a : inner_gcd(b, a % b); }
23    private static long inner_lcm(long a, long b){
24        return a / inner_gcd(a, b) * b; }
25    public Rational minus(){ return new Rational(-this.nom, this.denom); }
26    public Rational inv(){ return new Rational(this.denom, this.nom); }
27    public long sign(){
28        return this.nom == 0 ? 0 : inner_lcm(this.nom, this.denom) < 0 ? -1 : 1; }
29    public Rational abs(){
30        return new Rational(Math.abs(this.nom), Math.abs(this.denom)); }
31
32    public Rational add(Rational o){
33        final long lcm = inner_lcm(this.denom, o.denom);
34        return new Rational(lcm / this.denom * this.nom + lcm / o.denom * o.nom, lcm); }
35    public Rational sub(Rational o){ return this.add(o.minus()); }
36    public Rational mul(Rational o){ return new Rational(this.nom * o.nom,
37        this.denom * o.denom); }
38    public Rational div(Rational o){ return this.mul(o.inv()); }
39    public Rational gcd(Rational o){
40        return new Rational(inner_gcd(this.nom, o.nom), inner_lcm(this.denom,
41            o.denom)); }
42    public Rational lcm(Rational o){
43        return new Rational(inner_lcm(this.nom, o.nom), inner_gcd(this.denom,
44            o.denom)); }
45    public Rational pow(long p){
46        if(p == 0){ return Rational.ONE;
47        }else if(p % 2 != 0){ return this.mul(pow(p - 1));
48        }else{ Rational ret = pow(p / 2); return ret.mul(ret); } }
49
50    @Override
51    public int compareTo(Rational o){
52        final long det = this.nom * o.denom - this.denom * o.nom;
53        if(det < 0){ return -1;
54        }else if(det > 0){ return 1;
55        }else{ return 0; } }
56 }
```

4.3 多倍長有理数

有理数を表現するクラス. 多倍長なので安心して使える.

Source Code

```

1 // import java.math.BigInteger;
2 public static class BigRational implements Comparable<BigRational>{
3     public static final BigRational ZERO = new BigRational(0, 1);
4     public static final BigRational ONE = new BigRational(1, 1);
5     public static final BigRational NaN = new BigRational(1, 0);
6
7     public final BigInteger nom, denom;
8
9     public BigRational(BigInteger nom, BigInteger denom){
10        this.nom = calc_nom(nom, denom); this.denom = calc_denom(nom, denom); }
11
12    public BigRational(BigInteger nom){ this(nom, BigInteger.valueOf(1)); }
13    public BigRational(long nom, long denom){
14        this(BigInteger.valueOf(nom), BigInteger.valueOf(denom)); }
15    public BigRational(long nom){ this(nom, 1); }
16
17    private static BigInteger inner_lcm(BigInteger a, BigInteger b){
18        return a.multiply(b).divide(a.gcd(b)); }
19
20    private static BigInteger calc_nom(BigInteger nom, BigInteger denom){
21        nom = nom.divide(nom.gcd(denom));
22        if(nom.signum() * denom.signum() < 0){ nom = nom.abs().negate(); }
23        return nom; }
24    private static BigInteger calc_denom(BigInteger nom, BigInteger denom){
25        denom = denom.divide(nom.gcd(denom));
26        if(nom.equals(BigInteger.ZERO)) { return BigInteger.ONE; }
27        if(nom.signum() * denom.signum() < 0){ denom = denom.abs(); }
28        return denom; }
29
30    public BigRational minus(){ return new BigRational(nom.negate(), denom); }
31    public BigRational inv() { return new BigRational(denom, nom); }
32    public int sign() { return nom.signum(); }
33    public BigRational abs() { return new BigRational(nom.abs(), denom); }
34    public BigRational add(BigRational o) {
35        final BigInteger lcm = inner_lcm(this.denom, o.denom);
36        return new BigRational(lcm.divide(this.denom).multiply(this.nom)
37            .add(lcm.divide(o.denom).multiply(o.nom)), denom); }
38    public BigRational sub(BigRational o){ return this.add(o.minus()); }
39    public BigRational mul(BigRational o){
40        return new BigRational(this.nom.multiply(o.nom), this.denom.multiply(o
41            .denom)); }
42    public BigRational div(BigRational o){ return this.mul(o.inv()); }
43    public BigRational gcd(BigRational o){
44        return new BigRational(inner_gcd(this.nom, o.nom), this.denom.lcm(o
45            .denom)); }
46    public BigRational lcm(BigRational o){
47        return new BigRational(inner_lcm(this.nom, o.nom), this.denom.gcd(o
48            .denom)); }
49    public BigRational pow(int p){ return new BigRational(nom.pow(p), denom
50        .pow(p)); }
51
52    @Override
53    public int compareTo(BigRational arg0){
54        return this.nom.multiply(arg0.denom).compareTo(this.denom.multiply(
55            arg0.nom)); }
56 }
```

5 Mod

5.1 累乗 (mod m)

累乗 $a^e \pmod{m}$ に関しては、バイナリ法で高速に計算できる。

計算量

$O(\log e)$ ※ $\log(\text{指数})$ オーダー

Source Code

再帰関数で計算するコードはこちら。

```
1 public static long mod_pow(long a, long e, long m){
2     if(e == 0){
3         return 1;
4     }else if(e % 2 == 0){
5         long ret = mod_pow(a, e / 2, m);
6         return (ret * ret) % m;
7     }else{
8         return (mod_pow(a, e - 1, m) * a) % m;
9     }
10 }
```

ループで下位のケタから計算するコードはこちら。

```
1 public static long mod_pow(long a, long e, long m){
2     long ret = 1;
3     for(; e > 0; e /= 2){
4         if (e % 2 != 0) ret = (ret * a) % m;
5         a = (a * a) % m;
6     }
7     return ret;
8 }
```

Verified

する予定

5.2 逆元 (mod p)

a の $\text{mod } p$ での逆元は、フェルマーの小定理より、 $a^{-1} = a^{p-2} \pmod{p}$

計算量

$O(\log p)$ ※累乗にバイナリ法が使える

5.2.1 依存

- 累乗 (mod m) p.8

Source Code

```
1 public static long mod_inv(long a, long p){
2     return mod_pow(a, p-2, p);
3 }
```

Verified

する予定

5.3 逆元列挙 (mod p)

$\text{mod } p$ の逆元は 1 から N まで $O(N)$ で計算できる。

計算量

$O(N)$ ※列挙する最大の数

Source Code

```
1 //long[] inv = new long[MAX]; //1-indexed
2 public static void mod_inv(long[] inv, long p){
3     inv[1] = 1;
4     for(int i = 2; i <= N; i++){
5         inv[i] = p - (p / i) * inv[p % i] % p;
6     }
7 }
```

Verified

する予定

5.4 逆元 (mod m)

a の $\text{mod } m$ での逆元は, a と m が互いに素であれば拡張ユークリッドの互助法で求められる.

計算量

$O(\log am)$ ※最悪の場合. 平均はかなり早いはず.

Source Code

```
1 // a and m must be co-prime.
2 public static long mod_inv(long a, long m){
3     return (a == 1 ? 1 : (1 - m*mod_inv(m%a, a)) / a + m);
4 }
```

5.5 中国剰余定理

$x = a_i \pmod{m_i}$ という条件から, 条件を満たす $x \pmod{\prod m_i}$ を求める.

計算量

$O(\text{条件の数} * \log \sum m_i)$ ※計算量よりオーバーフローに気をつけること

依存

- 逆元 (mod m) p.9

Source Code

```
1 public static long chinese_remainder(long[] as, long[] ms){
2     long prod = 1;
3     for(long m : ms){ prod *= m; }
4
5     long ret = 0;
6     for(int i = 0; i < ms.length; i++){
7         final long M = prod / ms[i];
8         final long inv = mod_inv(M % ms[i], ms[i]);
9
10        long a = as[i] - as[i] / prod * prod;
11        if(a < 0){ a += prod; }
12
13        ret = (ret + M * inv * a % prod) % prod;
14    }
15
16    return ret;
17 }
```

5.6 Rolling Hash

文字列をハッシュする関数. 部分文字列に対するハッシュを定数時間で行える. 文字列 $s[0, n)$ に対して

$$\text{hash}(s) = (s[n-1] + s[n-2] * b + \dots + s[0] * b^{n-1}) \bmod M \quad (b, M \text{ は互いに素})$$

計算量

$O(N) \dots n$ は文字列の長さ

Source Code

```
1 public static long rolling_hash(String s, final long b, final long m){
2     long ret = 0;
3     for(char c : s.toCharArray()){
4         ret *= b; ret %= m;
5         ret += c; ret %= m;
6     }
7     return ret;
8 }
```

6 2DimRealGeometry

6.1 基本データ構造

Source Code

```

1 public static class Point2D {
2     public double x;
3     public double y;
4
5     public static final double EPS = 1e-9;
6
7     public Point2D(double x, double y) {
8         this.x = x;
9         this.y = y;
10    }
11
12    public Point2D(Point2D point) {
13        this.x = point.x;
14        this.y = point.y;
15    }
16
17    public String toString() {
18        return x + "," + y;
19    }
20
21    @Override
22    public boolean equals(Object o) {
23        if (o instanceof Point2D) {
24            Point2D another = (Point2D) o;
25
26            if(Point2D.eq(this.x, another.x) && Point2D.eq(this.y, another.y)
27                ){
28                return true;
29            }
30
31            return false;
32        }
33        return false;
34    }
35
36    public Point2D add(double x, double y) {
37        return new Point2D(this.x + x, this.y + y);
38    }
39
40    public Point2D sub(double x, double y) {
41        return add(-x, -y);
42    }
43
44    public Point2D add(Point2D another) {
45        return add(another.x, another.y);
46    }
47
48    public Point2D sub(Point2D another) {
49        return sub(another.x, another.y);
50    }
51
52    public Point2D mul(double d) {
53        return new Point2D(this.x * d, this.y * d);
54    }
55
56    public Point2D div(double d) {
57        return new Point2D(this.x / d, this.y / d);
58    }
59
60    public double dot(double x, double y) {
61        return this.x * x + this.y * y;

```

```

62
63    public double dot(Point2D another) {
64        return dot(another.x, another.y);
65    }
66
67    public double cross(double x, double y) {
68        return this.x * y - this.y * x;
69    }
70
71    public double cross(Point2D another) {
72        return cross(another.x, another.y);
73    }
74
75    public double dist(double x, double y) {
76        return Math.sqrt((this.x - x) * (this.x - x) + (this.y - y)
77            * (this.y - y));
78    }
79
80    public double dist(Point2D another) {
81        return dist(another.x, another.y);
82    }
83
84    public double dist_o() {
85        return dist(0, 0);
86    }
87
88    public Point2D unit() {
89        return div(dist_o());
90    }
91
92    public boolean pol(Point2D start, Point2D end) {
93        return end.sub(start).cross(this.sub(start)) < EPS;
94    }
95
96    public boolean pos(Point2D start, Point2D end) {
97        return (start.dist(this) + this.dist(end) < start.dist(end) + EPS);
98    }
99
100    public double pld(Point2D start, Point2D end) {
101        return Math.abs((end.sub(start).cross(this.sub(start)))
102            / end.sub(start).dist_o());
103    }
104
105    public double psd(Point2D start, Point2D end) {
106        if (end.sub(start).dot(this.sub(start)) < EPS) {
107            return this.dist(start);
108        } else if (start.sub(end).dot(this.sub(end)) < EPS) {
109            return this.dist(end);
110        } else {
111            return Math.abs(end.sub(start).cross(this.sub(start)) / end.dist(
112                start));
113        }
114    }
115
116    public static int signum(double x){
117        return Math.abs(x) < EPS ? 0 : x > 0 ? 1 : -1;
118    }
119
120    public static boolean eq(double x, double y){
121        return signum(x - y) == 0;
122    }
123
124    public static int ccw(Point2D p, Point2D r, Point2D s){
125        Point2D a = r.sub(p);
126        Point2D b = s.sub(p);
127
128        final int sgn = Point2D.signum(a.cross(b));
129        if(sgn != 0){
130            return sgn;
131        }else if(a.x * b.x < -EPS && a.y * b.y < -EPS){
132            return -1;

```

```

132     }else if(a.dist_o() < b.dist_o() - EPS){
133         return 1;
134     }else{
135         return 0;
136     }
137 }
138
139 public static boolean intersect_s(Point2D a1, Point2D a2, Point2D b1,
140     Point2D b2) {
141     return (Point2D.ccw(a1, a2, b1) * Point2D.ccw(a1, a2, b2) <= 0)
142         && (Point2D.ccw(b1, b2, a1) * Point2D.ccw(b1, b2, a2) <= 0);
143 }
144
145 public static boolean insersect_l(Point2D a1, Point2D a2, Point2D b1,
146     Point2D b2) {
147     return a1.sub(a2).cross(b1.sub(b2)) < EPS;
148 }
149
150 public static Point2D interpoint_s(Point2D a1, Point2D a2, Point2D b1,
151     Point2D b2) {
152     Point2D b = b2.sub(b1);
153     double d1 = Math.abs(b.cross(a1.sub(b1)));
154     double d2 = Math.abs(b.cross(a2.sub(b1)));
155     double t = d1 / (d1 + d2);
156     Point2D a = a2.sub(a1), v = a.mul(t);
157     return a1.add(v);
158 }
159
160 public static Point2D interpoint_l(Point2D a1, Point2D a2, Point2D b1,
161     Point2D b2) {
162     Point2D a = a2.sub(a1);
163     Point2D b = b2.sub(b1);
164     double t = b.cross(b1.sub(a1)) / b.cross(a);
165     Point2D v = a.mul(t);
166     return a1.add(v);
167 }
168
169 public static Point2D[] cross_ss(Point2D p1, double r1, Point2D p2,
170     double r2) {
171     double dis = p1.dist(p2);
172
173     if (r1 + EPS > r2 && r1 - EPS < r2 && dis < EPS) {
174         return new Point2D[0]; // same
175     }
176
177     if (dis - EPS < r1 + r2 && dis + EPS > r1 + r2) {
178         Point2D tmp = p2.sub(p1);
179         tmp = tmp.mul(r1 / tmp.dist_o());
180         Point2D ret[] = new Point2D[1];
181         ret[0] = p1.add(tmp);
182         return ret;
183     } else if (dis + EPS > r1 + r2) {
184         return new Point2D[0]; // out
185     }
186
187     double dis_m = Math.abs(r1 - r2);
188
189     if (dis_m + EPS > dis && dis_m - EPS < dis) {
190         Point2D tmp = null;
191         if (r1 > r2) {
192             tmp = p2.sub(p1);
193         } else {
194             tmp = p1.sub(p2);
195         }
196
197         double min = Math.min(r1, r2);
198
199         tmp = tmp.mul((min + tmp.dist_o()) / tmp.dist_o());
200
201         Point2D ret[] = new Point2D[1];
202         ret[0] = p1.add(tmp);

```

```

203         return ret;
204     } else if (dis_m + EPS > dis) {
205         return new Point2D[0]; // inner
206     } else {
207         Point2D ret[] = new Point2D[2];
208
209         double theta = Math.acos((dis * dis + r1 * r1 - r2 * r2)
210             / (2 * dis * r1));
211         double a = Math.atan2(p2.y - p1.y, p2.x - p1.x);
212
213         ret[0] = new Point2D(r1 * Math.cos(a + theta) + p1.x, r1
214             * Math.sin(a + theta) + p1.y);
215         ret[1] = new Point2D(r1 * Math.cos(a - theta) + p1.x, r1
216             * Math.sin(a - theta) + p1.y);
217         return ret;
218     }
219 }
220
221 public static double ss_dist(Point2D start1, Point2D end1, Point2D start2,
222     Point2D end2){
223     if(Point2D.intersect_s(start1, end1, start2, end2)){
224         return 0;
225     }else{
226         return Math.min(Math.min(Math.min(start1.psd(start2, end2), end1.
227             psd(start2, end2)), start2.psd(start1, end1)), end2.psd(start1
228             , end1));
229     }
230 }
231
232 public void interpoint_lc(Point2D start, Point2D end, Point2D c, double r,
233     Point2D ans[]) {
234     if (c.pld(start, end) > r + EPS)
235         return;
236     Point2D v = end.sub(start).unit();
237     double delta = v.dot(start.sub(c)) * v.dot(start.sub(c))
238         - start.dist(c) * start.dist(c) + r * r;
239     double t = -v.dot(start.sub(c));
240     double s = Math.sqrt(delta);
241     ans[0] = start.add(v.mul(t + s));
242     ans[1] = start.add(v.mul(t - s));
243 }
244
245 public Point2D normal_vector(Point2D p, Point2D a, Point2D b) {
246     Point2D v = b.sub(a).unit();
247     v = v.cross(p.sub(a)) > 0 ? new Point2D(v.y, (-1) * v.x) : new Point2D
248     (
249         (-1) * v.y, v.x);
250     return v.mul(p.pld(a, b));
251 }
252
253 public double area(Point2D a, Point2D b, Point2D c) {
254     return Math.abs((c.sub(a).cross(b.sub(a))) * 0.5);
255 }
256 }

```

7 2DimIntGeometry

7.1 2次元の点

依存

- 有理数 p.7 or 多倍長有理数 p.7

Source Code

```

1 public static class Point2D implements Comparable<Point2D>{
2     public static final Point2D NaN = new Point2D(Rational.NaN, Rational.NaN);
3
4     private Rational x, y;
5
6     public Point2D(Rational x, Rational y){
7         assert(x != null && y != null); // nullを入れたら殺す!
8         this.x = x; this.y = y;
9     }
10
11     public Rational get_x(){ return x; }
12     public Rational get_y(){ return y; }
13     public Rational norm(){ return this.x.pow(2).add(this.y.pow(2)); }
14
15     public Point2D add(Point2D o){return new Point2D(x.add(o.x),y.add(o.y)); }
16     public Point2D sub(Point2D o){return new Point2D(x.sub(o.x),y.sub(o.y)); }
17     public Point2D mul(Rational r){ return new Point2D(x.mul(r), y.mul(r)); }
18     public Point2D div(Rational r){ return new Point2D(x.div(r), y.div(r)); }
19
20     public Rational dot(Point2D o) { return x.mul(o.x).add(y.mul(o.y)); }
21     public Rational cross(Point2D o){ return x.mul(o.y).sub(y.mul(o.x)); }
22     public Rational dist(Point2D o) { return o.sub(this).norm(); }
23
24     public long ccw(Point2D r, Point2D s){
25         final Point2D a = r.sub(this), b = s.sub(this);
26         final long sign = a.cross(b).sign();
27
28         if(sign != 0){ return sign;
29         }else if(a.x.mul(b.x).sign() < 0 || a.y.mul(b.y).sign() < 0){
30             return -1;
31         }else if(a.norm().compareTo(b.norm()) < 0){
32             return 1;
33         }else{ return 0; }
34     }
35
36     @Override
37     public boolean equals(Object obj) { //必要ならHashCodeも生成する事
38         if(!(obj instanceof Point2D)){ return false; }
39         Point2D o = (Point2D) obj;
40         if(!this.x.equals(o.x) || !this.y.equals(o.y)){ return false; }
41         return true;
42     }
43
44     @Override
45     public int compareTo(Point2D o) {
46         if(this.x.compareTo(o.x) != 0){ return this.x.compareTo(o.x); }
47         else if(this.y.compareTo(o.y) != 0){ return this.y.compareTo(o.y); }
48         else { return 0; }
49     }
50 }

```

7.2 2次元の直線

依存

- 2次元の点 (有理数) p.12

Source Code

```

1 public static class Line2D{
2     private Point2D begin, end;
3
4     public Line2D(Point2D begin, Point2D end){
5         assert(begin != null && end != null);
6         this.begin = begin; this.end = end;
7     }
8
9     public Point2D get_begin(){ return begin; }
10    public Point2D get_end(){ return end; }
11    public Point2D get_dir(){ return this.end.sub(this.begin); }
12
13    public boolean is_orthogonal(Line2D o){
14        return Rational.ZERO.equals(this.get_dir().dot(o.get_dir()));
15    }
16    public boolean is_parallel(Line2D o){
17        return Rational.ZERO.equals(this.get_dir().cross(o.get_dir()));
18    }
19
20    public Point2D line_corss(Line2D o){
21        if(is_parallel(o)) { return Point2D.NaN; }
22        final Point2D this_dir = this.get_dir();
23        final Point2D o_dir = o.get_dir();
24
25        return this.begin.add(this_dir.mul(o_dir.cross(o.begin.sub(this.begin
26        )))div(o_dir.cross(this_dir)));
27    }
28
29    public boolean ss_intersects(Line2D o){
30        return this.begin.ccw(this.end, o.begin) * this.begin.ccw(this.end, o
31        .end) <= 0
32        && o.begin.ccw(o.end, this.begin) * o.begin.ccw(o.end, this
33        .end) <= 0;
34    }
35
36    @Override
37    public String toString(){ return this.begin + " -> " + this.end; }
38 }

```

8 String

8.1 Shift And

短いパターン文字列を bit 演算を使って高速に検索するアルゴリズム.

計算量

対象文字列の長さを n , パターンの長さを m とすると

- パターンの bit 配列の構築: $O(m)$
- 文字列検索: $O(n + m)$

Source Code

```
1 //Mのbit幅 >= パターンの文字列じゃないと死ぬ. 長いなら BitSet を使おう.
2 public static int shift_and(String t, String p){ //pの長さはbit幅依存
3     int[] M = new int[Character.MAX_VALUE]; // alphabet全体分の長さが要.
4     int count = 0;
5     for(int i = 0; i < p.length(); i++){
6         M[p.charAt(i)] |= (1 << i);
7     }
8
9     for(int i = 0, S = 0; i < t.length(); i++){
10        S = ((S << 1) | 1) & M[t.charAt(i)];
11
12        if((S & (1 << (p.length() - 1))) != 0){
13            count++; // t[i - p.length() + 1, i]
14        }
15    }
16
17    return count;
18 }
```

9 Puzzle

9.1 Nim(山 N 個, 制限無し)

全ての xor を取る. 0 だったら先手必負, それ以外なら先手必勝

計算量

Source Code