目次

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

```java
import java.util.Scanner;

public class Main {
    public static void debug(Object ... objs){
        System.out.println(Arrays.toString(objs));
    }

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        // input here
        sc.close();
    }
}
```

1.2 自作 Scanner

入力が多い場合は, 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

```java
public static class Scanner {
    private BufferedReader br;
    private StringTokenizer tok;

    public Scanner(InputStream is) throws IOException {
        br = new BufferedReader(new InputStreamReader(is));
    }

    private void getLine() throws IOException {
        while (!hasNext()) { tok = new StringTokenizer(br.readLine()); }
    }

    private boolean hasNext() {
        return tok != null && tok.hasMoreTokens();
    }

    public String next() throws IOException {
        getLine(); return tok.nextToken();
    }

    public int nextInt() throws IOException {
        return Integer.parseInt(next());
    }
    // 他のnextXXXもXXX.parseXXX()メソッドを使って作れるので省略

    public void close() throws IOException {
        br.close();
    }
}
```

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

```java
public static class UnionFind{
    int[] par; //

    public UnionFind(int n){
        par = new int[n];
        for(int i = 0; i < n; i++){ par[i] = -1; }
    }

    public int find(int x){
        if(par[x] < 0){
            return x;
        }else{
            return par[x] = find(par[x]);
        }
    }

    public boolean union(int x, int y){
        x = find(x);
        y = find(y);

        if(x != y){
            if(par[y] < par[x]) {  // 多い方が根になるようにスワップする.
                int tmp = x; x = y; y = tmp;
            }
            par[x] += par[y];
            par[y] = x;
            return true;
        }else{
            return false;
        }
    }

    public boolean same(int x, int y){
        return find(x) == find(y);
    }

    public int size(int x){
        return -par[find(x)];
    }
}
```

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

```java
public static class WeightedUnionFind{
    int[] par; // 親の番号
    int[] ws;  // 親との重みの差

    public WeightedUnionFind(int n){
        par = new int[n];
        ws  = new int[n];
        for(int i = 0; i < n; i++){ par[i] = -1; }
    }

    public int find(int x){
        if(par[x] < 0){
            return x;
        }else{
            final int parent = find(par[x]);
            ws[x] += ws[par[x]];
            return par[x] = parent;
        }
    }

    public int weight(int x){
        find(x);
        return ws[x];
    }

    public boolean union(int x, int y, int w){ // x <-(w)- y (x + w = y)
        w += weight(x);
        w -= weight(y);
        x = find(x); y = find(y);

        if(x != y){
            if(par[y] < par[x]) {  // 多い方が根になるようにスワップする.
                int tmp = x; x = y; y = tmp; w = -w;
            }
            par[x] += par[y]; par[y] = x;
            ws[y] = w;
            return true;
        }else{
            return false;
        }
    }

    public boolean same(int x, int y){
        return find(x) == find(y);
    }

    public Integer diff(int x, int y){ // x - y を求める. 比較不能ならnull.
        if(!same(x, y)){ return null; }
        return this.weight(x) - this.weight(y);
    }
    // size()はUnionFindと同じなので省略.
}
```

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

```java
public static class SegTree{
    int n;
    long[] dat;

    public SegTree(int n_) {
        int n = 1;
        while(n < n_){
            n *= 2;
        }

        this.n = n;
        dat = new long[this.n * 2 - 1];
        for(int i = 0; i < this.n * 2 - 1 ; i++){
            dat[i] = 0;
        }
    }

    public long calc(long fst, long snd){
        return fst + snd;
    }

    public void update(int k, long a){
        k += n - 1;
        dat[k] = a;

        while(k > 0){
            k = (k - 1) / 2;
            dat[k] = calc(dat[k * 2 + 1], dat[k * 2 + 2]);
        }
    }

    public long query(int a, int b, int k, int l, int r){
        if(r <= a || b <= l){
            return 0;
        }else if(a <= l && r <= b){
            return dat[k];
        }else {
            return calc(query(a, b, k * 2 + 1, l, (l + r) / 2), query(a, b, k * 2 + 2 , (l + r) / 2, r));
        }
    }

    public long query(int a, int b){
        return query(a, b, 0, 0, n);
    }
}
```

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

```java
public static class SegTree{
    int n;
    long[] dat, lazy;

    public SegTree(int n_) {
        int n = 1;
        while(n < n_){ n *= 2;} this.n = n;
        dat = new long[this.n * 2 - 1];
        lazy = new long[this.n * 2 - 1];
        for(int i = 0; i < this.n * 2 - 1 ; i++){
            dat[i] = 0; lazy[i] = 0;
        }
    }

    private void lazy_evaluate_node(int k, int a, int b){
        dat[k] += lazy[k] * (b - a);
        if(k < n - 1){
            lazy[2 * k + 1] += lazy[k]; lazy[2 * k + 2] += lazy[k];
        }
        lazy[k] = 0;
    }

    public void update_node(int k){
        dat[k] = dat[2 * k + 1] + dat[2 * k + 2];
    }

    public void update(long v, int a, int b, int k, int l, int r){
        lazy_evaluate_node(k, l, r);

        if(r <= a || b <= l){ return;
        }else if(a <= l && r <= b){
            lazy[k] += v; lazy_evaluate_node(k, l, r);
        }else {
            update(v, a, b, k * 2 + 1, l , (l + r) / 2);
            update(v, a, b, k * 2 + 2, (l + r) / 2, r);
            update_node(k);
        }
    }

    public long get(int a, int b, int k, int l, int r){
        lazy_evaluate_node(k, l, r);

        if(r <= a || b <= l){ return 0;
        }else if(a <= l && r <= b){ return dat[k];
        }else {
            long v1 = get(a, b, k * 2 + 1, l , (l + r) / 2);
            long v2 = get(a, b, k * 2 + 2, (l + r) / 2, r);
            update_node(k); return v1 + v2;
        }
    }

    public int size(){ return this.n; }
}
```

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

```java
public static class BIT{
    int[] dat;

    public BIT(int n){
        dat = new int[n + 1];
    }

    public void add(int k, int a){ // k : 0-indexed
        for(int i = k + 1; i < dat.length; i += i & -i){
            dat[i] += a;
        }
    }

    public int sum(int s, int t){ // [s, t)
        if(s > 0) return sum(0, t) - sum(0, s);

        int ret = 0;
        for(int i = t; i > 0; i -= i & -i) {
            ret += dat[i];
        }
        return ret;
    }

    public int get(int k){ // k : 0-indexed
        int p = Integer.highestOneBit(dat.length - 1);
        for(int q = p; q > 0; q >>= 1, p |= q){
            if( p >= dat.length || k < dat[p]) p ^= q;
            else k -= dat[p];
        }
        return p;
    }
}
```

3 Graph

3.1 隣接行列

Source Code

```
public static final long INF = Long.MAX_VALUE / 2 - 1;

public static long[][] init_adj(final int n){
    long[][] ret = new long[n][n];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            ret[i][j] = i == j ? 0l : INF;
        }
    }
    return ret;
}
```

3.2 隣接リスト

Source Code

```
public static ArrayList<Integer,LinkedHashMap<Integer,Long>> init_adj(final
     int n){
    ArrayList<Integer, LinkedHashMap<Integer, Long>> ret =
        new ArrayList<Integer, LinkedHashMap<Integer, Long>>();
    for(int i = 0; i < n; i++){
        ret.add(new LinkedHashMap<Integer, Long>());
    }
    return ret;
}
```

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

```
public static final int unvisited = 0;
public static final int visiting = 1;
public static final int visited = 2;

public static boolean dfs(int node, boolean[][] adj, int[] state){
    state[node] = visiting;

    for(int i = 0; i < adj.length; i++){
        if(!adj[node][i]){ continue; }
        else if(state[i] == unvisited){
            if(!dfs(i, adj, state)){ //
                state[node] = visited; return false;
            }
        }else if(state[i] == visiting){ //cycle!
            return false;
        }
    }

    state[node] = visited;
    return true;
}

public static boolean find_cycle(boolean[][] adj){
    int[] state = new int[adj.length];
    for(int i = 0; i < adj.length; i++){
        state[i] = unvisited;
    }
    for(int i = 0; i < adj.length; i++){
        if(state[i] == unvisited){
            if(!dfs(i, adj, state)){
                return false;
            }
        }
    }
    return true;
}
```

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

```
public static final int unvisited = 0;
public static final int visiting = 1;
public static final int visited = 2;

public static boolean dfs(int node, boolean[][] adj, int[] state, LinkedList<
     Integer> list){
    state[node] = visiting;

    for(int i = 0; i < adj.length; i++){
        if(!adj[node][i]){ continue; }
        else if(state[i] == unvisited){
            if(!dfs(i, adj, state)){ //
                state[node] = visited; return false;
            }
        }else if(state[i] == visiting){ //cycle!
            return false;
        }
    }

    state[node] = visited;
    list.addFront(node);
    return true;
}

public static boolean topological_sort(long[][] ad, LinedList<Integer> list){
    int[] state = new int[adj.length];
    for(int i = 0; i < adj.length; i++){
        state[i] = unvisited;
    }
    for(int i = 0; i < adj.length; i++){
        if(state[i] == unvisited){
            if(!dfs(i, adj, state, list)){
                return false;
            }
        }
    }
    return true;
}
```

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

```
public static long gcd(long a, long b){
    return b == 0 ? a : gcd(b, a % b);
}
public static long lcm(long a, long b){
    return a / gcd(a, b) * b;
}
```

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

```java
// equalsメソッドは eclipse等で生成するなりして作ること
public static class Rational implements Comparable<Rational> {
    public static final Rational ZERO = new Rational(0);
    public static final Rational ONE  = new Rational(1);
    public static final Rational NaN  = new Rational(1, 0){
        public String toString(){ return "NaN"; }};

    private long nom, denom;

    public Rational(long nom, long denom) {
        if(nom == 0){ this.nom = 0; this.denom = 1; }
        else{ final long gcd = inner_gcd(nom, denom);
            this.nom = nom / gcd; this.denom = denom / gcd;
            if(this.nom * this.denom < 0){
                this.nom = -Math.abs(this.nom);
                this.denom = Math.abs(this.denom);
            }}}
    public Rational(long num) { this(num, 1); }
    public long get_nom(){ return this.nom; }
    public long get_denom(){ return this.denom; }
    private static long inner_gcd(long a, long b){
        return b == 0 ? a : inner_gcd(b, a % b); }
    private static long inner_lcm(long a, long b){
        return a / inner_gcd(a, b) * b; }
    public Rational minus(){ return new Rational(-this.nom, this.denom); }
    public Rational inv(){ return new Rational(this.denom, this.nom); }
    public long sign(){
        return this.nom ==0 ? 0 : inner_lcm(this.nom,this.denom) < 0 ?-1:1;}
    public Rational abs(){
        return new Rational(Math.abs(this.nom), Math.abs(this.denom));}

    public Rational add(Rational o){
        final long lcm = inner_lcm(this.denom, o.denom);
        return new Rational(lcm/this.denom*this.nom + lcm/o.denom*o.nom,lcm);}
    public Rational sub(Rational o){ return this.add(o.minus()); }
    public Rational mul(Rational o){ return new Rational(this.nom * o.nom, this.denom * o.denom); }
    public Rational div(Rational o){ return this.mul(o.inv()); }
    public Rational gcd(Rational o){
        return new Rational(inner_gcd(this.nom, o.nom), inner_lcm(this.denom, o.denom)); }
    public Rational lcm(Rational o){
        return new Rational(inner_lcm(this.nom, o.nom), inner_gcd(this.denom, o.denom)); }
    public Rational pow(long p){
        if(p == 0){ return Rational.ONE;
        }else if(p % 2 != 0){ return this.mul(pow(p - 1));
        }else{ Rational ret = pow(p / 2); return ret.mul(ret); }}

    @Override
    public int compareTo(Rational o){
        final long det = this.nom * o.denom - this.denom * o.nom;
        if(det < 0){ return -1;
        }else if(det > 0){ return 1;
        }else{ return 0; }}
}
```

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

```java
// import java.math.BigInteger;
public static class BigRational implements Comparable<BigRational>{
    public static final BigRational ZERO = new BigRational(0, 1);
    public static final BigRational ONE  = new BigRational(1, 1);
    public static final BigRational NaN  = new BigRational(1, 0);

    public final BigInteger nom, denom;

    public BigRational(BigInteger nom, BigInteger denom){
        this.nom = calc_nom(nom,denom); this.denom = calc_denom(nom,denom); }

    public BigRational(BigInteger nom){ this(nom, BigInteger.valueOf(1)); }
    public BigRational(long nom, long denom){
        this(BigInteger.valueOf(nom), BigInteger.valueOf(denom)); }
    public BigRational(long nom){ this(nom, 1); }

    private static BigInteger inner_lcm(BigInteger a, BigInteger b){
        return a.multiply(b).divide(a.gcd(b));}

    private static BigInteger calc_nom(BigInteger nom, BigInteger denom){
        nom = nom.divide(nom.gcd(denom));
        if(nom.signum() * denom.signum() < 0){ nom = nom.abs().negate(); }
        return nom; }
    private static BigInteger calc_denom(BigInteger nom, BigInteger denom){
        denom = denom.divide(nom.gcd(denom));
        if(nom.equals(BigInteger.ZERO)) { return BigInteger.ONE; }
        if(nom.signum() * denom.signum() < 0){ denom = denom.abs(); }
        return denom; }

    public BigRational minus(){ return new BigRational(nom.negate(), denom); }
    public BigRational inv()  { return new BigRational(denom, nom); }
    public int sign() { return nom.signum(); }
    public BigRational abs()  { return new BigRational(nom.abs(), denom); }
    public BigRational add(BigRational o)  {
        final BigInteger lcm = inner_lcm(this.denom, o.denom);
        return new BigRational(lcm.divide(this.denom).multiply(this.nom)
                .add(lcm.divide(o.denom).multiply(o.nom)), denom); }
    public BigRational sub(BigRational o){ return this.add(o.minus()); }
    public BigRational mul(BigRational o){
        return new BigRational(this.nom.multiply(o.nom), this.denom.multiply(o.denom)); }
    public BigRational div(BigRational o){ return this.mul(o.inv()); }
    public BigRational gcd(BigRational o){
        return new BigRational(inner_gcd(this.nom, o.nom), this.denom.lcm(o.denom)); }
    public BigRational lcm(BigRational o){
        return new BigRational(inner_lcm(this.nom, o.nom), this.denom.gcd(o.denom)); }
    public BigRational pow(int p){ return new BigRational(nom.pow(p), denom.pow(p)); }

    @Override
    public int compareTo(BigRational arg0){
        return this.nom.multiply(arg0.denom).compareTo(this.denom.multiply(arg0.nom)); }
}
```

5 Mod

5.1 累乗 (mod m)

累乗 $a^e \ (mod \, m)$ 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(指数) オーダー

Source Code

再帰関数で計算するコードはこちら.

```java
public static long mod_pow(long a, long e, long m){
    if(e == 0){
        return 1;
    }else if(e % 2 == 0){
        long ret = mod_pow(a, e / 2, m);
        return (ret * ret) % m;
    }else{
        return (mod_pow(a, e - 1, m) * a) % m;
    }
}
```

ループで下位のケタから計算するコードはこちら.

```java
public static long mod_pow(long a, long e, long m){
    long ret = 1;
    for(; e > 0; e /= 2){
        if (e % 2 != 0) ret = (ret * a) % m;
        a = (a * a) % m;
    }
    return ret;
}
```

Verified

する予定

5.2 逆元 (mod p)

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

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

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

Verified

する予定

5.3 逆元列挙 (mod p)

$mod \, p$ 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

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

Verified

する予定

5.4 逆元 (mod m)

a の $mod\, m$ 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

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

5.5 中国剰余定理

$x = a_i\ (mod\ m_i)$ という条件から, 条件を満たす $x\ (mod \prod m_i)$ 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

```
public static long chinese_remainder(long[] as, long[] ms){
    long prod = 1;
    for(long m : ms){ prod *= m; }

    long ret = 0;
    for(int i = 0; i < ms.length; i++){
        final long M = prod / ms[i];
        final long inv = mod_inv(M % ms[i], ms[i]);

        long a = as[i] - as[i] / prod * prod;
        if(a < 0){ a += prod; }

        ret = (ret + M * inv * a % prod) % prod;
    }

    return ret;
}
```

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

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

計算量

$O(N)...n$ は文字列の長さ

Source Code

```
public static long rolling_hash(String s, final long b, final long m){
    long ret = 0l;
    for(char c : s.toCharArray()){
        ret *= b; ret %= m;
        ret += c; ret %= m;
    }
    return ret;
}
```

6 2DimRealGeometry

6.1 基本データ構造

Source Code

```java
public static class Point2D {
    public double x;
    public double y;

    public static final double EPS = 1e-9;

    public Point2D(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public Point2D(Point2D point) {
        this.x = point.x;
        this.y = point.y;
    }

    public String toString() {
        return x + "," + y;
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof Point2D) {
            Point2D another = (Point2D) o;

            if(Point2D.eq(this.x, another.x) && Point2D.eq(this.y, another.y
                )){
                return true;
            }

            return false;
        }
        return false;
    }

    public Point2D add(double x, double y) {
        return new Point2D(this.x + x, this.y + y);
    }

    public Point2D sub(double x, double y) {
        return add(-x, -y);
    }

    public Point2D add(Point2D another) {
        return add(another.x, another.y);
    }

    public Point2D sub(Point2D another) {
        return sub(another.x, another.y);
    }

    public Point2D mul(double d) {
        return new Point2D(this.x * d, this.y * d);
    }

    public Point2D div(double d) {
        return new Point2D(this.x / d, this.y / d);
    }

    public double dot(double x, double y) {
        return this.x * x + this.y * y;
    }

    public double dot(Point2D another) {
        return dot(another.x, another.y);
    }

    public double cross(double x, double y) {
        return this.x * y - this.y * x;
    }

    public double cross(Point2D another) {
        return cross(another.x, another.y);
    }

    public double dist(double x, double y) {
        return Math.sqrt((this.x - x) * (this.x - x) + (this.y - y)
                * (this.y - y));
    }

    public double dist(Point2D another) {
        return dist(another.x, another.y);
    }

    public double dist_o() {
        return dist(0, 0);
    }

    public Point2D unit() {
        return div(dist_o());
    }

    public boolean pol(Point2D start, Point2D end) {
        return end.sub(start).cross(this.sub(start)) < EPS;
    }

    public boolean pos(Point2D start, Point2D end) {
        return (start.dist(this) + this.dist(end) < start.dist(end) + EPS);
    }

    public double pld(Point2D start, Point2D end) {
        return Math.abs((end.sub(start).cross(this.sub(start)))
                / end.sub(start).dist_o());
    }

    public double psd(Point2D start, Point2D end) {
        if (end.sub(start).dot(this.sub(start)) < EPS) {
            return this.dist(start);
        } else if (start.sub(end).dot(this.sub(end)) < EPS) {
            return this.dist(end);
        } else {
            return Math.abs(end.sub(start).cross(this.sub(start)) / end.dist(
                start));
        }
    }

    public static int signum(double x){
        return Math.abs(x) < EPS ? 0 : x > 0 ? 1 : -1;
    }

    public static boolean eq(double x, double y){
        return signum(x - y) == 0;
    }

    public static int ccw(Point2D p, Point2D r, Point2D s){
        Point2D a = r.sub(p);
        Point2D b = s.sub(p);

        final int sgn = Point2D.signum(a.cross(b));
        if(sgn != 0){
            return sgn;
        }else if(a.x * b.x < -EPS && a.y * b.y < -EPS){
            return -1;
        }else if(a.dist_o() < b.dist_o() - EPS){
            return 1;
        }else{
            return 0;
        }
    }

    public static boolean intersect_s(Point2D a1, Point2D a2, Point2D b1,
            Point2D b2) {
        return (Point2D.ccw(a1, a2, b1) * Point2D.ccw(a1, a2, b2) <= 0)
                && (Point2D.ccw(b1, b2, a1) * Point2D.ccw(b1, b2, a2) <= 0);
    }

    public static boolean insersect_l(Point2D a1, Point2D a2, Point2D b1,
            Point2D b2) {
        return a1.sub(a2).cross(b1.sub(b2)) < EPS;
    }

    public static Point2D interpoint_s(Point2D a1, Point2D a2, Point2D b1,
            Point2D b2) {
        Point2D b = b2.sub(b1);
        double d1 = Math.abs(b.cross(a1.sub(b1)));
        double d2 = Math.abs(b.cross(a2.sub(b1)));
        double t = d1 / (d1 + d2);
        Point2D a = a2.sub(a1), v = a.mul(t);
        return a1.add(v);
    }

    public static Point2D interpoint_l(Point2D a1, Point2D a2, Point2D b1,
            Point2D b2) {
        Point2D a = a2.sub(a1);
        Point2D b = b2.sub(b1);
        double t = b.cross(b1.sub(a1)) / b.cross(a);
        Point2D v = a.mul(t);
        return a1.add(v);
    }

    public static Point2D[] cross_ss(Point2D p1, double r1, Point2D p2,
            double r2) {
        double dis = p1.dist(p2);

        if (r1 + EPS > r2 && r1 - EPS < r2 && dis < EPS) {
            return new Point2D[0]; // same
        }

        if (dis - EPS < r1 + r2 && dis + EPS > r1 + r2) {
            Point2D tmp = p2.sub(p1);
            tmp = tmp.mul(r1 / tmp.dist_o());
            Point2D ret[] = new Point2D[1];
            ret[0] = p1.add(tmp);
            return ret;
        } else if (dis + EPS > r1 + r2) {
            return new Point2D[0]; // out
        }

        double dis_m = Math.abs(r1 - r2);

        if (dis_m + EPS > dis && dis_m - EPS < dis) {
            Point2D tmp = null;
            if (r1 > r2) {
                tmp = p2.sub(p1);
            } else {
                tmp = p1.sub(p2);
            }

            double min = Math.min(r1, r2);

            tmp = tmp.mul((min + tmp.dist_o()) / tmp.dist_o());

            Point2D ret[] = new Point2D[1];
            ret[0] = p1.add(tmp);
            return ret;
        } else if (dis_m + EPS > dis) {
            return new Point2D[0]; // inner
        } else {
            Point2D ret[] = new Point2D[2];

            double theta = Math.acos((dis * dis + r1 * r1 - r2 * r2)
                    / (2 * dis * r1));
            double a = Math.atan2(p2.y - p1.y, p2.x - p1.x);

            ret[0] = new Point2D(r1 * Math.cos(a + theta) + p1.x, r1
                    * Math.sin(a + theta) + p1.y);
            ret[1] = new Point2D(r1 * Math.cos(a - theta) + p1.x, r1
                    * Math.sin(a - theta) + p1.y);
            return ret;
        }
    }

    public static double ss_dist(Point2D start1, Point2D end1, Point2D start2,
          Point2D end2){
        if(Point2D.intersect_s(start1, end1, start2, end2)){
            return 0;
        }else{
            return Math.min(Math.min(Math.min(start1.psd(start2, end2), end1.
                psd(start2, end2)), start2.psd(start1, end1)), end2.psd(start1
                , end1));
        }
    }

    public void interpoint_lc(Point2D start, Point2D end, Point2D c, double r,
            Point2D ans[]) {
        if (c.pld(start, end) > r + EPS)
            return;
        Point2D v = end.sub(start).unit();
        double delta = v.dot(start.sub(c)) * v.dot(start.sub(c))
                - start.dist(c) * start.dist(c) + r * r;
        double t = -v.dot(start.sub(c));
        double s = Math.sqrt(delta);
        ans[0] = start.add(v.mul(t + s));
        ans[1] = start.add(v.mul(t + s));
    }

    public Point2D normal_vector(Point2D p, Point2D a, Point2D b) {
        Point2D v = b.sub(a).unit();
        v = v.cross(p.sub(a)) > 0 ? new Point2D(v.y, (-1) * v.x) : new Point2D
            (
                (-1) * v.y, v.x);
        return v.mul(p.pld(a, b));
    }

    public double area(Point2D a, Point2D b, Point2D c) {
        return Math.abs((c.sub(a).cross(b.sub(a))) * 0.5);
    }
}
```

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

```java
public static class Point2D implements Comparable<Point2D>{
    public static final Point2D NaN = new Point2D(Rational.NaN, Rational.NaN);

    private Rational x, y;

    public Point2D(Rational x, Rational y){
        assert(x != null && y != null); // nullを入れたら殺す!
        this.x = x; this.y = y;
    }

    public Rational get_x(){ return x; }
    public Rational get_y(){ return y; }
    public Rational norm(){ return this.x.pow(2).add(this.y.pow(2)); }

    public Point2D add(Point2D o){return new Point2D(x.add(o.x),y.add(o.y)); }
    public Point2D sub(Point2D o){return new Point2D(x.sub(o.x),y.sub(o.y)); }
    public Point2D mul(Rational r){ return new Point2D(x.mul(r), y.mul(r)); }
    public Point2D div(Rational r){ return new Point2D(x.div(r), y.div(r)); }

    public Rational dot(Point2D o)  { return x.mul(o.x).add(y.mul(o.y)); }
    public Rational cross(Point2D o){ return x.mul(o.y).sub(y.mul(o.x)); }
    public Rational dist(Point2D o) { return o.sub(this).norm(); }

    public long ccw(Point2D r, Point2D s){
        final Point2D a = r.sub(this), b = s.sub(this);
        final long sign = a.cross(b).sign();

        if(sign != 0){ return sign;
        }else if(a.x.mul(b.x).sign() < 0 || a.y.mul(b.y).sign() < 0){
            return -1;
        }else if(a.norm().compareTo(b.norm()) < 0){
            return 1;
        }else{ return 0; }
    }

    @Override
    public boolean equals(Object obj) { //必要ならHashCodeも生成する事
        if(!(obj instanceof Point2D)){ return false; }
        Point2D o = (Point2D) obj;
        if(!this.x.equals(o.x) || !this.y.equals(o.y)){ return false; }
        return true;
    }

    @Override
    public int compareTo(Point2D o) {
        if(this.x.compareTo(o.x) != 0){ return this.x.compareTo(o.x); }
        else if(this.y.compareTo(o.y) != 0){ return this.y.compareTo(o.y); }
        else { return 0; }
    }
}
```

7.2 2 次元の直線

依存

- 2 次元の点 (有理数) p.12

Source Code

```java
public static class Line2D{
    private Point2D begin, end;

    public Line2D(Point2D begin, Point2D end){
        assert(begin != null && end != null);
        this.begin = begin; this.end = end;
    }

    public Point2D get_begin(){ return begin; }
    public Point2D get_end(){ return end; }
    public Point2D get_dir(){ return this.end.sub(this.begin); }

    public boolean is_orthogonal(Line2D o){
        return Rational.ZERO.equals(this.get_dir().dot(o.get_dir()));
    }
    public boolean is_parallel(Line2D o){
        return Rational.ZERO.equals(this.get_dir().cross(o.get_dir()));
    }

    public Point2D line_corss(Line2D o){
        if(is_parallel(o)) { return Point2D.NaN; }
        final Point2D this_dir = this.get_dir();
        final Point2D o_dir = o.get_dir();

        return this.begin.add(this_dir.mul(o_dir.cross(o.begin.sub(this.begin
            ))).div(o_dir.cross(this_dir)));
    }

    public boolean ss_intersects(Line2D o){
        return this.begin.ccw(this.end, o.begin) * this.begin.ccw(this.end, o.
            end) <= 0
                && o.begin.ccw(o.end, this.begin) * o.begin.ccw(o.end, this.
                    end) <= 0;
    }

    @Override
    public String toString(){ return this.begin + " -> " + this.end; }
}
```

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

```java
//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.
public static int shift_and(String t, String p){ //pの長さはbit幅依存
    int[] M = new int[Character.MAX_VALUE]; // alphabet全体分の長さが必要.
    int count = 0;
    for(int i = 0; i < p.length(); i++){
        M[p.charAt(i)] |= (1 << i);
    }

    for(int i = 0, S = 0; i < t.length(); i++){
        S = ((S << 1) | 1) & M[t.charAt(i)];

        if((S & (1 << (p.length() - 1))) != 0){
            count++; // t[i - p.length() + 1, i]
        }
    }

    return count;
}
```

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