1 テンプレート

```
import static java.lang.Math.*;
import static java.util.Arrays.*;
import java.io.*;
import java.util.*;

public class Main {
  static boolean LOCAL = System.getSecurityManager() == null;
  Scanner sc = new Scanner(System.in);

  void run() {
  }

  void debug(Object...os) {
    System.err.println(deepToString(os));
  }

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

2 データ構造

2.1 BIT

```
class BIT {
  int[] vs;
  BIT(int n) {
    vs = new int[n + 1];
  }
  void add(int k, int a) {
    for (int i = k + 1; i < vs.length; i += i & -i) {
      vs[i] += a;
    }
  }
  int sum(int s, int t) {
    if (s > 0) return sum(0, t) - sum(0, s);
    int res = 0;
    for (int i = t; i > 0; i -= i & -i) {
      res += vs[i];
    }
    return res;
  }
  //[0,i] の和が k より大きくなる最小の i を求める
  int get(int k) {
    int p = Integer.highestOneBit(vs.length - 1);
    for (int q = p; q > 0; q >>= 1, p |= q) {
      if (p >= vs.length || k < vs[p]) p ^= q;
      else k -= vs[p];
    }
    return p;
  }
}
```

2.2 RMQ

```
class RMQ {
  int[] vs;
  int[][] min;
  RMQ(int[] vs) {
    int n = vs.length, m = log2(n) + 1;
    this.vs = vs;
    min = new int[m][n];
    for (int i = 0; i < n; i++) min[0][i] = i;
    for (int i = 1, k = 1; i < m; i++, k <<= 1) {
      for (int j = 0; j + k < n; j++) {
        min[i][j] = vs[min[i - 1][j]] <= vs[min[i - 1][j + k]] ?
            min[i - 1][j] : min[i - 1][j + k];
      }
    }
  }
  //最小値が複数存在する場合は一番最初のを返す
  int query(int from, int to) {
    int k = log2(to - from);
    return vs[min[k][from]] <= vs[min[k][to - (1 << k)]] ?
        min[k][from] : min[k][to - (1 << k)];
  }
  int log2(int b) {
    return 31 - Integer.numberOfLeadingZeros(b);
  }
}
```

2.3 範囲の更新

```
class Intervals {
  TreeMap<Integer, Integer> map = new TreeMap<Integer, Integer>();
  Intervals() {
    map.put(Integer.MIN_VALUE, -1);
    map.put(Integer.MAX_VALUE, -1);
  }
  void paint(int s, int t, int c) {
    int p = get(t);
    map.subMap(s, t).clear();
    map.put(s, c);
    map.put(t, p);
  }
  int get(int k) {
    return map.floorEntry(k).getValue();
  }
}
```

2.4 Treap

副作用なしの実装

```
class T {
  final int key, val;
  final double p;
  final T left, right;
  T(int key, int val, double p, T left, T right) {
    this.key = key;
    this.val = val;
    this.p = p;
    this.left = left;
    this.right = right;
  }
  T change(T left, T right) {
    return new T(key, val, p, left, right);
  }
  T normal() {
    if (left != null && left.p < p && (right == null || left.p < right.p)) {
      return left.change(left.left, change(left.right, right));
    } else if (right != null && right.p < p) {
      return right.change(change(left, right.left), right.right);
    }
    return this;
  }
}
T put(T t, int key, int val) {
  if (t == null) return new T(key, val, random(), null, null);
  if (key < t.key) return t.change(put(t.left, key, val), t.right).normal();
  if (key > t.key) return t.change(t.left, put(t.right, key, val)).normal();
  return new T(key, val, t.p, t.left, t.right);
}
T remove(T t, int key) {
  if (t == null) return null;
  if (key < t.key) return t.change(remove(t.left, key), t.right);
  if (key > t.key) return t.change(t.left, remove(t.right, key));
  return merge(t.left, t.right);
}
T merge(T t1, T t2) {
  if (t1 == null) return t2;
  if (t2 == null) return t1;
  if (t1.p < t2.p) return t1.change(t1.left, merge(t1.right, t2));
  return t2.change(merge(t1, t2.left), t2.right);
}
```

2.5 セグメント木

配列とループを用いた高速な実装

$parent(i) = i/2$

$left(i) = 2i$

$size(i) = N/highest(i)$

$s(i) = size(i)i - N$

```
class Seg {
  int N;
  Seg(int n) {
    N = Integer.highestOneBit(n) << 1;
  }
  //点を含む区間に対する処理
  void query(int k) {
    for (int i = N + k; i > 0; i >>= 1) {
      //...
    }
  }
  //カバーする区間に対する処理
  void query(int s, int t) {
    while (0 < s && s + (s & -s) <= t) {
      int i = (N + s) / (s & -s);
      //...
      s += s & -s;
    }
    while (s < t) {
      int i = (N + t) / (t & -t) - 1;
```

```
21        //...
22        t -= t & -t;
23      }
24    }
25  }
```

3 グラフ

3.1 公式集

3.1.1 オイラーの多面体定理

連結な平面グラフについて、頂点数 V, 辺数 E, 面数 F の関係は外面も含めて $V - E + F = 2$

3.1.2 MaxFlow-MinCut 定理

最大 s-t-フローの流量=最小 s-t-カットの容量

グラフ G の最小カットの辺集合は残容量グラフ G_f で s から到達可能な点の集合を X とすると $\delta_G^+(X)$

3.1.3 マッチングなどの関係式

| 最大マッチング | $\leq$ | 最小点カバー | (二部グラフでは等号成立)

| 最大安定集合 | + | 最小点カバー | $= V$

孤立点のないグラフについて | 最大マッチング | + | 最小辺カバー | $= V$

二部グラフについて | 最小辺カバー | = | 最大安定集合 |

$X \subseteq V(G)$ が G の点カバー $\Leftrightarrow V(G) \setminus X$ は G の安定集合

3.1.4 行列木定理

グラフ G の全域木の総数は (G の次数行列 $-$ G の隣接行列) の任意の余因子に等しい

3.1.5 次数列木の総数

K_n の全域木で、頂点 i の次数が d_i であるようなものの総数は、

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$$

3.1.6 Kasteleyn の公式

全ての偶数長閉路を奇数的に向きづけらようなグラフについて、その隣接行列 (逆向きの辺は-1 とする) の行列式は完全マッチングの個数の二乗に等しい

平面グラフには必ずこのような向き付けが存在する

二部グラフの場合は、二部隣接行列の行列式が完全マッチングの個数と等しくなる

3.1.7 Tutte 行列

Tutte 行列とは $V \times V$ の行列で、グラフ G の各辺を勝手に向き付けて得られる有向グラフ G' の辺 (v,u) に対し、変数 x_{vu} を割り当て、$t_{vu} = x_{vu}, t_{uv} = -x_{vu}$ としたもの

一般のグラフに対して、Tutte 行列のランクはマッチングの大きさの二倍と等しい

x をランダムにとれば十分な確率で正しい解が得られる

3.2 トポロジカルソート

トポロジカル順序とは v[i] から v[j] に辺がある $\Rightarrow$ i < j が成立するもののことをいう

深さ優先探索を行い、帰りがけに番号を振ると、それが逆トポロジカル順序になる

```
1 V[] topologicalSort(V[] vs) {
2   n = vs.length;
3   us = new V[n];
4   for (V v : vs) {
5     if (v.state == 0 && !dfs(v)) return null;
6   }
7   return us;
8 }
9 boolean dfs(V v) {
10  v.state = 1;
11  for (V u : v) {
12    if (u.state == 1 || u.state == 0 && !dfs(u)) return false;
13  }
14  us[--n] = v;
15  v.state = 2;
16  return true;
17 }
```

3.3 強連結成分分解

まずトポロジカル順序を求め (この際に閉路は無視する)、その順に逆辺を深さ優先探索する

このときに得られた極大な探索木が強連結成分である

強連結成分の個数 k が返り、comp にはトポロジカル順序 [0,k) が入る

```
1 int scc(V[] vs) {
2   n = vs.length;
3   us = new V[n];
4   for (V v : vs) if (!v.visit) dfs(v);
5   for (V v : vs) v.visit = false;
6   for (V u : us) if (!u.visit) dfsrev(u, n++);
7   return n;
8 }
9 void dfs(V v) {
10  v.visit = true;
11  for (V u : v.fs) if (!u.visit) dfs(u);
12  us[--n] = v;
13 }
14 void dfsrev(V v, int k) {
15  v.visit = true;
16  for (V u : v.rs) if (!u.visit) dfsrev(u, k);
17  v.comp = k;
18 }
```

3.4 関節点・橋

任意に根を選んで DFS を行い、行きがけに番号 num を振る

以下のうちの最小値として low を計算する

・num[v]

・直前の辺以外で v に隣接するすでに訪れた頂点 u における num[u]

・v の子供 u における low[u]

このとき、

根: r が関節点 $\Leftrightarrow$ r が 2 つ以上の子供を持つ

根以外: v が関節点 ⇔ v の子供 u に対して num[v] ≤ low[u]
(v,u) が橋 ⇔ num[v] < low[u]
count にはその頂点が取り除かれたときにできる分裂の個数が入り、橋集合が返る

```
ArrayList<E> connection(V[] vs) {
  bridge = new ArrayList<E>();
  for (V v : vs) if (v.num < 0) {
    dfs(v, 0);
    if (v.count > 0) v.count--;
  }
  return bridge;
}
int dfs(V v, int c) {
  v.num = c;
  int low = c;
  boolean rev = false;
  for (V u : v) {
    if (u.num < 0) {
      int t = dfs(u, c + 1);
      low = min(low, t);
      if (v.num <= t) v.count++;
      if (v.num < t) bridge.add(new E(v, u));
    } else if (u.num != v.num - 1 || rev) low = min(low, u.num);
    else rev = true;
  }
  return low;
}
```

3.5 最小平均長閉路

負の辺があってもよい
prev をたどって得られるパスに含まれる任意の閉路が最小平均長の閉路である
$O(VE)$

```
double mmc(int[] ss, int[] ts, double[] cs, int n) {
  int m = ss.length;
  double[][] dp = new double[n + 1][n];
  for (int i = 0; i < n; i++) {
    fill(dp[i + 1], INF);
    for (int j = 0; j < m; j++) {
      dp[i + 1][ts[j]] = min(dp[i + 1][ts[j]], dp[i][ss[j]] + cs[j]);
    }
  }
  double res = INF;
  for (int i = 0; i < n; i++) if (dp[n][i] < INF) {
    double max = -INF;
    for (int j = 0; j < n; j++) {
      max = max(max, (dp[n][i] - dp[j][i]) / (n - j));
    }
    res = min(res, max);
  }
  return res;
}
```

3.6 最小根指定有向木

根以外の各点に入る最小の辺を貪欲に選び、閉路ができたら圧縮して再帰的に処理している
有向木自体を求めるには、圧縮時にもともとの辺を覚えておけばよい
根が指定されないときは、十分大きな値をを各辺からから引き、根から他の全ての頂点にコスト 0 の辺を加えればよい
最大有向森を求めるには辺の重みを反転し、根から他の全ての頂点にコスト 0 の辺を加えればよい
$O(VE)$

```
int arborescence(V[] vs, V r) {
  int res = 0;
  for (V v : vs) for (E e : v.es) e.to.min = min(e.to.min, e.cost);
  for (V v : vs) if (v != r) {
    if (v.min == INF) return INF;
    res += v.min;
  }
  for (V v : vs) for (E e : v.es) if (e.to != r) {
    e.cost -= e.to.min;
    if (e.cost == 0) {
      v.fs.add(e.to);
      e.to.rs.add(v);
    }
  }
  int m = scc(vs);
  if (m == vs.length) return res;
  V[] us = new V[m];
  for (int i = 0; i < m; i++) us[i] = new V();
  for (V v : vs) for (E e : v.es) {
    if (v.comp != e.to.comp) us[v.comp].add(us[e.to.comp], e.cost);
  }
  return min(INF, res + arborescence(us, us[r.comp]));
}
```

3.7 シュタイナー木

非負の無向グラフに対し、ターミナル ts の要素を全て含む最小の木のコストを求める
$dp[S][v] := S \subset T$ と $v \in V$ を含む最小シュタイナー木のコスト
$O(3^t n + 2^t n^2 + n^3)$

```
int steiner(int[][] g, int[] ts) {
  int n = g.length, m = ts.length;
```

```
  if (m < 2) return 0;
  int[][] dp = new int[1 << m][n];
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
      }
    }
  }
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
      dp[1 << i][j] = g[ts[i]][j];
    }
  }
  for (int i = 1; i < 1 << m; i++) if (((i - 1) & i) != 0) {
    for (int j = 0; j < n; j++) {
      dp[i][j] = INF;
      for (int k = (i - 1) & i; k > 0; k = (k - 1) & i) {
        dp[i][j] = min(dp[i][j], dp[k][j] + dp[i ^ k][j]);
      }
    }
    for (int j = 0; j < n; j++) {
      for (int k = 0; k < n; k++) {
        dp[i][j] = min(dp[i][j], dp[i][k] + g[k][j]);
      }
    }
  }
  return dp[(1 << m) - 1][ts[0]];
}
```

3.8 Dinic

```
int dinic(V s, V t) {
  int flow = 0;
  for (int p = 1; ; p++) {
    Queue<V> que = new LinkedList<V>();
    s.level = 0;
    s.p = p;
    que.offer(s);
    while (!que.isEmpty()) {
      V v = que.poll();
      v.iter = v.es.size() - 1;
      for (E e : v.es) if (e.to.p < p && e.cap > 0) {
        e.to.level = v.level + 1;
        e.to.p = p;
        que.offer(e.to);
      }
    }
    if (t.p < p) return flow;
    for (int f; (f = dfs(s, t, INF)) > 0; ) flow += f;
  }
}
int dfs(V v, V t, int f) {
  if (v == t) return f;
  for (; v.iter >= 0; v.iter--) {
    E e = v.es.get(v.iter);
    if (v.level < e.to.level && e.cap > 0) {
      int d = dfs(e.to, t, min(f, e.cap));
      if (d > 0) {
        e.cap -= d;
        e.rev.cap += d;
        return d;
      }
    }
  }
  return 0;
}
class V {
  ArrayList<E> es = new ArrayList<E>();
  int level, p, iter;
  void add(V to, int cap) {
    E e = new E(to, cap), rev = new E(this, 0);
    e.rev = rev; rev.rev = e;
    es.add(e); to.es.add(rev);
  }
}
class E {
  V to;
  E rev;
  int cap;
  E(V to, int cap) {
    this.to = to;
    this.cap = cap;
  }
}
```

3.9 最小全域カット

非負の重みであること

$O(n^3)$

```
int minCut(int[][] c) {
```

```
  int n = c.length, cut = INF;
  int[] id = new int[n], b = new int[n];
  for (int i = 0; i < n; i++) id[i] = i;
  for ( ; n > 1; n--) {
    fill(b, 0);
    for (int i = 0; i + 1 < n; i++) {
      int p = i + 1;
      for (int j = i + 1; j < n; j++) {
        b[id[j]] += c[id[i]][id[j]];
        if (b[id[p]] < b[id[j]]) p = j;
      }
      swap(id, i + 1, p);
    }
    cut = min(cut, b[id[n - 1]]);
    for (int i = 0; i < n - 2; i++) {
      c[id[i]][id[n - 2]] += c[id[i]][id[n - 1]];
      c[id[n - 2]][id[i]] += c[id[n - 1]][id[i]];
    }
  }
  return cut;
}
```

3.10 最小費用流

全ての頂点を渡すのを忘れないこと

```
int minCostFlow(V[] vs, V s, V t, int flow) {
  int res = 0;
  while (flow > 0) {
    for (V v : vs) v.min = INF;
    PriorityQueue<E> que = new PriorityQueue<E>();
    s.min = 0;
    que.offer(new E(s, 0, 0));
    while (!que.isEmpty()) {
      E crt = que.poll();
      if (crt.cost == crt.to.min) {
        for (E e : crt.to.es) {
          int tmp = crt.cost + e.cost + crt.to.h - e.to.h;
          if (e.cap > 0 && e.to.min > tmp) {
            e.to.min = tmp;
            e.to.prev = e;
            que.offer(new E(e.to, 0, e.to.min));
          }
        }
      }
    }
    if (t.min == INF) return -1;
    int d = flow;
    for (E e = t.prev; e != null; e = e.rev.to.prev) {
      d = min(d, e.cap);
    }
    for (E e = t.prev; e != null; e = e.rev.to.prev) {
      res += d * e.cost;
      e.cap -= d;
      e.rev.cap += d;
    }
    flow -= d;
    for (V v : vs) v.h += v.min;
  }
  return res;
}
class V {
  ArrayList<E> es = new ArrayList<E>();
  E prev;
  int min, h;
  void add(V to, int cap, int cost) {
    E e = new E(to, cap, cost), rev = new E(this, 0, -cost);
    e.rev = rev; rev.rev = e;
    es.add(e); to.es.add(rev);
  }
}
class E implements Comparable<E> {
  V to;
  E rev;
  int cap, cost;
  E(V to, int cap, int cost) {
    this.to = to;
    this.cap = cap;
    this.cost = cost;
  }
  public int compareTo(E o) {
    return cost - o.cost;
  }
}
```

3.11 二部マッチング

最小点被覆は、左側の到達不能な頂点と右側の到達可能な頂点を合わせたもの

$O(VE)$

```
int bipartiteMatching(V[] vs) {
  int match = 0;
```

```
  for (V v : vs) if (v.pair == null) {
    for (V u : vs) u.used = false;
    if (dfs(v)) match++;
  }
  return match;
}
boolean dfs(V v) {
  v.used = true;
  for (V u : v) {
    V w = u.pair;
    if (w == null || !w.used && dfs(w)) {
      v.pair = u;
      u.pair = v;
      return true;
    }
  }
  return false;
}
class V extends ArrayList<V> {
  V pair;
  boolean used;
  void connect(V v) {
    add(v);
    v.add(this);
  }
}
```

$O(\sqrt{V}E)$

```
int hopcroftKarp(V[] vs) {
  for (int match = 0;;) {
    Queue<V> que = new LinkedList<V>();
    for (V v : vs) v.level = -1;
    for (V v : vs) if (v.pair == null) {
      v.level = 0;
      que.offer(v);
    }
    while (!que.isEmpty()) {
      V v = que.poll();
      for (V u : v) {
        V w = u.pair;
        if (w != null && w.level < 0) {
          w.level = v.level + 1;
          que.offer(w);
        }
      }
    }
    for (V v : vs) v.used = false;
    int d = 0;
    for (V v : vs) if (v.pair == null && dfs(v)) d++;
    if (d == 0) return match;
    match += d;
  }
}
boolean dfs(V v) {
  v.used = true;
  for (V u : v) {
    V w = u.pair;
    if (w == null || !w.used && v.level < w.level && dfs(w)) {
      v.pair = u;
      u.pair = v;
      return true;
    }
  }
  return false;
}
```

3.12 安定マッチング

男性最良の安定マッチングを求め方

・男性を一人選び、その男性がまだプロポーズしていない女性の中で優先度の一番高い女性にプロポーズする

・その女性にとってこの男性が現在の相手よりも好ましいか、相手がいない場合、この男性と新たなペアを作り、以前のペアを解消する

・振られた男性はすぐにまたほかの女性にプロポーズする

$O(n^2)$

男 i のペアを res[i] として返す

orderM[i][j] は男 i が j 番目に好む女

preferW[i][j] は女 i の男 j に対する選好順位

```
int[] stableMatching(int[][] orderM, int[][] preferW) {
  int n = orderM.length;
  int[] pairM = new int[n], pairW = new int[n], p = new int[n];
  fill(pairM, -1);
  fill(pairW, -1);
  for (int i = 0; i < n; i++) {
    while (pairM[i] < 0) {
      int w = orderM[i][p[i]++], m = pairW[w];
      if (m == -1) {
        pairM[i] = w;
        pairW[w] = i;
      } else if (preferW[w][i] < preferW[w][m]) {
```

```
        pairM[m] = -1;
        pairM[i] = w;
        pairW[w] = i;
        i = m;
      }
    }
  }
  return pairM;
}
```

3.13 オイラー閉路

連結チェックを忘れないこと

無向グラフの場合は rev も調べる

```
V[] eulerianWalk(V v) {
  Stack<V> res = new Stack<V>(), stack = new Stack<V>();
  stack.push(v);
  while (!stack.isEmpty()) {
    v = stack.pop();
    while (v.p < v.size()) {
      stack.push(v);
      v = v.get(v.p++);
    }
    res.push(v);
  }
  return res.toArray(new V[0]);
}
```

3.14 LCA

ルートから DFS したときに訪れるノードの順を e[2*n-1]、その深さを v[2*n-1]、各ノードが初めて現れる e のインデックスを a[n] として、e[RMQv([a[u], a[v]])] を求めればよい

3.15 2SAT

各 $(y \vee z)$ に対し、$\neg y \to z$ と $\neg z \to y$ の 2 つの辺をもつグラフを作成する。

このグラフを強連結成分分解し、ある強連結成分に x と $\neg x$ が同時に含まれることは、元の論理式が充足不能であることと同値である。

x を含む強連結成分のトポロジカル順序が $\neg x$ を含む強連結成分のトポロジカル順序よりも後ろ $\Leftrightarrow x$ が真

4 平面幾何

4.1 交点など

```
//線分と点の距離
double disSP(P p1, P p2, P q) {
  if (p2.sub(p1).dot(q.sub(p1)) < EPS) return q.sub(p1).abs();
  if (p1.sub(p2).dot(q.sub(p2)) < EPS) return q.sub(p2).abs();
  return disLP(p1, p2, q);
}
//直線と点の距離
double disLP(P p1, P p2, P q) {
  return abs(p2.sub(p1).det(q.sub(p1))) / p2.sub(p1).abs();
}
//線分と線分の交差判定
boolean crsSS(P p1, P p2, P q1, P q2) {
  if (max(p1.x, p2.x) + EPS < min(q1.x, q2.x)) return false;
  if (max(q1.x, q2.x) + EPS < min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) + EPS < min(q1.y, q2.y)) return false;
  if (max(q1.y, q2.y) + EPS < min(p1.y, p2.y)) return false;
  return sig(p2.sub(p1).det(q1.sub(p1))) * sig(p2.sub(p1).det(q2.sub(p1))) < EPS
    && sig(q2.sub(q1).det(p1.sub(q1))) * sig(q2.sub(q1).det(p2.sub(q1))) < EPS;
}
//円と線分の交差判定
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);
}
//円と円の交差判定
boolean crsCC(P c1, double r1, P c2, double r2) {
  double dis = c1.sub(c2).abs();
  return dis < r1 + r2 + EPS && abs(r1 - r2) < dis + EPS;
}
//四点の同一円周上判定 (一直線上は true)
boolean onCir(P p1, P p2, P p3, P p4) {
  if (abs(p2.sub(p1).det(p3.sub(p1))) < EPS) return true;
  P c = ccenter(p1, p2, p3);
  return abs(c.sub(p1).abs2() - c.sub(p4).abs2()) < EPS;
}
//点から直線に下ろした垂線の足
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()));
}
//直線と直線の交点
P isLL(P p1, P p2, P q1, P q2) {
  double d = q2.sub(q1).det(p2.sub(p1));
  if (abs(d) < EPS) return null;
  return p1.add(p2.sub(p1).mul(q2.sub(q1).det(q1.sub(p1)) / d));
}
//直線と円の交点 (p1 に近い順)
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 = x * x - y * (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(sqrt(d) / y);
  return new P[]{q1.sub(q2), q1.add(q2)};
}
//二円の交点
P[] isCC(P c1, double r1, P c2, double r2) {
  double x = c1.sub(c2).abs2();
  double y = ((r1 * r1 - r2 * r2) / x + 1) / 2;
  double d = 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(sqrt(d)).rot90();
  return new P[]{q1.sub(q2), q1.add(q2)};
}
//点 p をから引いた接線の接点
P[] tanCP(P c, double r, P p) {
  double x = p.sub(c).abs2();
  double d = 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 * sqrt(d) / x).rot90();
  return new P[]{c.add(q1.sub(q2)), c.add(q1.add(q2))};
}
//二円の共通接線
P[][] tanCC(P c1, double r1, P c2, double r2) {
  List<P[]> list = new ArrayList<P[]>();
  if (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][]);
}
//２円の共通部分面積
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 < abs(r1 - r2) + EPS) {
    double r = min(r1, r2);
    return r * r * PI;
  }
  double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
  double t1 = acos(x / r1);
  double t2 = acos((d - x) / r2);
  return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * sin(t1);
}
//原点中心半径 r の円と原点と p1,p2 からなる三角形の共通部分面積
double areaCT(double r, P p1, P p2) {
  P[] qs = isCL(O, 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;
  }
}
```

4.2 凸包

反時計回りの凸包を作る

辺上の点を含まない

不等号を変えることで辺上の点の選択が可能

辺上の点を含む場合、点の重複があってはいけない

```
P[] convexHull(P[] ps) {
  int n = ps.length, k = 0;
  if (n <= 1) return ps;
  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;
}
```

4.3 凸多角形の切断

$O(n)$

凸多角形を直線 p1p2 で切断して左側を残す

```
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]);
}
```

4.4 線分アレンジメント

線分の端点と交点を頂点とし、二点が同一線分上に乗っていて間に他の点がない時に、二点間の距離を重みとする辺をもつグラフを作成する

$O(n^3)$

```
V[] segmentArrangement(P[] ps1, P[] ps2) {
  int n = ps1.length;
  TreeSet<V> set = new TreeSet<V>();
  for (int i = 0; i < n; i++) {
    set.add(new V(ps1[i]));
    set.add(new V(ps2[i]));
    for (int j = 0; j < i; j++) {
      //線分交差判定のうち、端点が他の線分上の条件は抜くこと
      if (crsSS(ps1[i], ps2[i], ps1[j], ps2[j])) {
        set.add(new V(isLL(ps1[i], ps2[i], ps1[j], ps2[j])));
      }
    }
  }
  V[] vs = set.toArray(new V[0]);
  for (int i = 0; i < n; i++) {
    ArrayList<V> list = new ArrayList<V>();
    for (V v : vs) {
      if (crsSP(ps1[i], ps2[i], v.p)) list.add(v);
    }
    V[] us = list.toArray(new V[0]);
    sort(us);
    for (int j = 0; j + 1 < us.length; j++) {
      us[j].connect(us[j + 1]);
    }
  }
  return vs;
}
```

4.5 点の多角形に対する内外判定

$O(n)$

内部のときは 1、辺上のときは 0、外部のときは-1 を返す

```
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 (abs(a.det(b)) < EPS && a.dot(b) < EPS) return 0;
  }
  return res;
}
```

4.6 凸多角形とその外部の点の距離

凸多角形の辺の法線によって平面を分割し、どの領域に含まれるかを二分探索により求める

$O(\log n)$

凸多角形は反時計回りとすること

```
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;
}
```

4.7 凸多角形の直径

キャリパー法を用いて凸多角形上の最遠点間距離を求める

$O(n)$

```
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 = max(maxd, ps[i].sub(ps[j]).abs());
  } while (i != is || j != js);
  return maxd;
}
```

4.8 公式集

4.8.1 三角形の内心

$$\frac{a\overrightarrow{A}+b\overrightarrow{B}+c\overrightarrow{C}}{a+b+c}$$

4.8.2 三角形の外心

$$\frac{\overrightarrow{A}+\overrightarrow{B}-\frac{\overrightarrow{BC}\cdot\overrightarrow{CA}}{\overrightarrow{AB}\times\overrightarrow{BC}}\overrightarrow{AB}^T}{2}$$

4.8.3 三角形の垂心

$$\overrightarrow{H}=3\overrightarrow{G}-2\overrightarrow{O}$$

4.8.4 三角形の傍心

$$\frac{-a\overrightarrow{A}+b\overrightarrow{B}+c\overrightarrow{C}}{-a+b+c}$$ (残りの二点も同様)

4.8.5 三角形の外接円の半径

$$R=\frac{abc}{4S}$$

4.8.6 ヘロンの公式

$$2s := a+b+c$$
$$S = \sqrt{s(s-a)(s-b)(s-c)}$$

4.8.7 ピックの公式

頂点が全て格子点上にある単純多角形の面積 S は、辺上の格子点数を B、内部の格子点数を I とすると

$$S=\frac{B}{2}+I-1$$

5 空間幾何

5.1 交点など

```
//直線と点の距離
double disLP(P p1, P p2, P q) {
  return p2.sub(p1).det(q.sub(p1)).abs() / p2.sub(p1).abs();
}
//直線と直線の距離
double disLL(P p1, P p2, P q1, P q2) {
  P p = q1.sub(p1);
  P u = p2.sub(p1);
  P v = q2.sub(q1);
  double d = u.abs2() * v.abs2() - u.dot(v) * u.dot(v);
  if (abs(d) < EPS) return disLP(q1, q2, p1);
  double s = (p.dot(u) * v.abs2() - p.dot(v) * u.dot(v)) / d;
  return disLP(q1, q2, p1.add(u.mul(s)));
}
//平面と直線の交点
P isFL(P p, P o, P q1, P q2) {
```

```
  double a = o.dot(q2.sub(p));
  double b = o.dot(q1.sub(p));
  double d = a - b;
  if (abs(d) < EPS) return null;
  return q1.mul(a).sub(q2.mul(b)).div(d);
}
//平面と平面の交線
P[] isFF(P p1, P o1, P p2, P o2) {
  P e = o1.det(o2);
  P v = o1.det(e);
  double d = o2.dot(v);
  if (abs(d) < EPS) return null;
  P q = p1.add(v.mul(o2.dot(p2.sub(p1)) / d));
  return new P[]{q, q.add(e)};
}
```

5.2 空間凸包

各面を外から見て反時計回りになるように返す
同一平面上に 4 点が乗らないよう摂動すること
$O(n^2)$

```
P[][] convexHull(P[] ps) {
  int n = ps.length;
  //vs[i][j]=辺 i->j をこの向きに含む三角形が、まだ調べてない:0、残った:-1、取り除かれた:1
  int[][] vs = new int[n][n];
  List<int[]> crt = new ArrayList<int[]>();
  crt.add(new int[]{0, 1, 2});
  crt.add(new int[]{2, 1, 0});
  for (int i = 3; i < n; i++) {
    List<int[]> next = new ArrayList<int[]>();
    for (int[] t : crt) {
      int v = ps[t[1]].sub(ps[t[0]]).det(ps[t[2]].sub(ps[t[0]])).dot(
          ps[i].sub(ps[t[0]])) < 0 ? -1 : 1;
      if (v < 0) next.add(t);
      for (int j = 0; j < 3; j++) {
        if (vs[t[(j + 1) % 3]][t[j]] == 0) {
          vs[t[j]][t[(j + 1) % 3]] = v;
        } else {
          if (vs[t[(j + 1) % 3]][t[j]] != v) {
            if (v > 0) next.add(new int[]{t[j], t[(j + 1) % 3], i});
            else next.add(new int[]{t[(j + 1) % 3], t[j], i});
          }
          vs[t[(j + 1) % 3]][t[j]] = 0;
        }
      }
    }
    crt = next;
  }
  P[][] pss = new P[crt.size()][3];
  for (int i = 0; i < pss.length; i++) {
    for (int j = 0; j < 3; j++) pss[i][j] = ps[crt.get(i)[j]];
  }
  return pss;
}
```

5.3 凸多面体の切断

$O(n \log n)$
凸多面体を平面 po で切断して裏側を残す
凸多面体は外から見た時に反時計回りになるようにすること

```
P[][] convexCut(P[][] pss, P p, P o) {
  ArrayList<P[]> res = new ArrayList<P[]>();
  ArrayList<P> sec = new ArrayList<P>();
  for (P[] ps : pss) {
    int n = ps.length;
    ArrayList<P> qs = new ArrayList<P>();
    boolean dif = false;
    for (int i = 0; i < n; i++) {
      int d1 = sig(o.dot(ps[i].sub(p)));
      int d2 = sig(o.dot(ps[(i + 1) % n].sub(p)));
      if (d1 <= 0) qs.add(ps[i]);
      if (d1 * d2 < 0) {
        P q = isFL(p, o, ps[i], ps[(i + 1) % n]);
        qs.add(q);
        sec.add(q);
      }
      if (d1 == 0) sec.add(ps[i]);
      else dif = true;
      dif |= o.dot(ps[(i + 1) % n].sub(ps[i]).det(ps[(i + 2) % n].sub(ps[i]))) < -EPS;
    }
    if (qs.size() > 0 && dif) res.add(qs.toArray(new P[0]));
  }
  //2D の convexHull に.dot(o) をつけたもの
  if (sec.size() > 0) res.add(convexHull2D(sec.toArray(new P[0]), o));
  return res.toArray(new P[0][]);
}
//空間の初期化
P[][] init() {
  P[][] pss = new P[6][4];
  pss[0][0] = pss[1][0] = pss[2][0] = new P(-INF, -INF, -INF);
```

```
31   pss[0][3] = pss[1][1] = pss[5][2] = new P(-INF, -INF,  INF);
32   pss[0][1] = pss[2][3] = pss[4][2] = new P(-INF,  INF, -INF);
33   pss[0][2] = pss[5][3] = pss[4][1] = new P(-INF,  INF,  INF);
34   pss[1][3] = pss[2][1] = pss[3][2] = new P( INF, -INF, -INF);
35   pss[1][2] = pss[5][1] = pss[3][3] = new P( INF, -INF,  INF);
36   pss[2][2] = pss[4][3] = pss[3][1] = new P( INF,  INF, -INF);
37   pss[5][0] = pss[4][0] = pss[3][0] = new P( INF,  INF,  INF);
38   return pss;
39 }
```

5.4 公式集

5.4.1 オイラーの四面体公式

六辺の長さが a,b,c,d,e,f の四面体の体積

四面体を $O-ABC$ とすると $a=AB, b=BC, c=CA, d=OC, e=OA, f=OB$

$(12V)^2 = a^2d^2(b^2+c^2+e^2+f^2-a^2-d^2)+b^2e^2(c^2+a^2+f^2+d^2-b^2-e^2)+c^2f^2(a^2+b^2+d^2+e^2-c^2-f^2)-a^2b^2c^2-a^2e^2f^2-d^2b^2f^2-d^2e^2c^2$

5.5 変換

5.5.1 双対変換

■変換方法

双対変換は d 次元の点 $p=(p_1,p_2,\ldots,p_d)$ を d 次元の超平面 $p^*: x_d = p_1x_1+p_2x_2+\ldots+p_{d-1}x_{d-1}-p_d$ に写す

また d 次元の超平面 $h: x_d = a_1x_1+a_2x_2+\ldots+a_d$ は d 次元の点 $h^*=(a_1,a_2,\ldots,a_{d-1},-a_d)$ に写される

■性質

この変換により接続関係と順序関係は保存される

つまり、

$p \in h \Leftrightarrow h^* \in p^*$

p が h の上にある $\Leftrightarrow$ h^*が p^*の上にある

■エンベロープと凸包

超平面 h の集合 H における上側エンベロープを求める問題は双対変換を行うことにより、点 h^* の集合 H^* における下側凸包を求める問題になる

5.5.2 反転変換 1

■変換方法

二次元の点 $p=(x,y)$ を三次元の点 $p'=(x,y,x^2+y^2)$ に写す

また、二次元の円 $C: (x-a)^2+(y-b)^2=r^2$ を三次元の平面 $C': z=a(x-a)+b(y-b)+r^2$ に写す

■性質

二次元における円の内外判定を三次元における平面の上下判定に変換する

つまり、

p が C の内部に含まれる $\Leftrightarrow$ p'が C'の下にある

■ドロネー分割とボロノイ図

二次元の点集合 P に対して反転変換を行ったものを P' とすると、

P' における下側凸包を $z=0$ に射影したものは P のドロネー分割になり、P'^* における上側エンベロープを $z=0$ に射影したものは P のボロノイ図になる

5.5.3 反転変換 2

■変換方法

二次元の点 $p=(r,\theta)$ を二次元の点 $p'=(\frac{1}{r},\theta)$ に写す

原点を通る円は直線に、他の円は円に写される

■性質

原点を通る円に対する内外判定が直線の上下判定になる

6 数学

6.1 数え上げ

6.1.1 カタラン数

$$C(n) = \frac{{}_{2n}C_n}{n+1}$$

ノード数 n に対してバイナリツリーを構成する方法は $C(n)$ 通り

正 n 角形を $n-2$ 個の三角形に分ける方法は $C(n-2)$ 通り

6.1.2 重複組合せ

i 番目のものが $m[i]$ 個ある n 種類のものの中から k 個選ぶ組合せの総数

$x[i,j] = x[i-1,j]+x[i,j-1]-x[i-1,j-m[i]-1]$

6.1.3 分割数

n を k 個の非負整数に分割する仕方の総数

自然数への分割は $P(n-k,k)$、任意個数の自然数への分割は $P(n,n)$

$P[i,j] = P[i-1,j]+P[i,j-i]$

任意個数の自然数への分割はもっと高速に計算できる

$O(n^{\frac{3}{2}})$

```
1  int partition(int n) {
2    int[] dp = new int[n + 1];
3    dp[0] = 1;
4    for (int i = 1; i <= n; i++) {
5      for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1) {
6        dp[i] += dp[i - (3 * j * j - j) / 2] * r;
7        if (i - (3 * j * j + j) / 2 >= 0) {
8          dp[i] += dp[i - (3 * j * j + j) / 2] * r;
9        }
10     }
11   }
12   return dp[n];
13 }
```

6.1.4 ベル数

互いに異なる n 個のものを分割する方法の総数

$B[n] = \sum_{i=0}^{n-1} C[n-1,i]B[i]$

6.1.5 第一種スターリング数

互いに異なる n 個のものを空でない k 個のサイクルに分割する方法の総数

また、上昇階乗べきを通常べきの和で表す際の係数でもある

$S[n,k] = (n-1)S[n-1,k]+S[n-1,k-1]$

6.1.6 第二種スターリング数

互いに異なる n 個のものを空でない k 組に分割する方法の総数

また、通常べきを下降階乗べきの和で表す際の係数でもある
$B[n] = \sum_{i=1}^{n} S[n,i]$ が成り立つ
$S[n,k] = kS[n-1,k] + S[n-1,k-1]$
r 個以上の要素数の組に分割する場合は
$S[n,k] = kS[n-1,k] + C[n-1,r-1]S[n-r,k-1]$
となる

6.1.7 オイラー数

$\{1,\dots,n\}$ 上の置換のうちで、k 個の上昇を持つものの総数
$E[n,k] = (k+1)E[n-1,k] + (n-k)E[n-1,k-1]$

6.1.8 包除原理

$$\left|\bigcup_{P\in A} P\right| = \sum_{\emptyset\subset S\subseteq A} (-1)^{|S|-1} \left|\bigcap_{P\in S} P\right|$$

$$g(A) = \sum_{S\subseteq A} f(S) \Longleftrightarrow f(A) = \sum_{S\subseteq A} (-1)^{|A|-|S|} g(S)$$

6.1.9 メビウス関数

$$\mu(n) = \begin{cases} 0 & (n \text{ が平方因子を持つとき}) \\ (-1)^k & (n \text{ が異なる } k \text{ 個の素数の積のとき}) \end{cases}$$

$$g(n) = \sum_{d|n} f(d) \Longleftrightarrow f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$$

$$g(x) = \sum_{n=1}^{\lfloor x \rfloor} f(\frac{x}{n}) \Longleftrightarrow f(x) = \sum_{n=1}^{\lfloor x \rfloor} \mu(n) g(\frac{x}{n})$$

6.1.10 旧称バーンサイドの補題

$$|X/G| = \frac{1}{|G|} \sum_{g\in G} |X_g|$$

6.2 数論

6.2.1 公式集

■フェルマーの小定理

$a^p \equiv a \pmod p$
$(a,p) = 1$ ならば $a^{p-1} \equiv 1 \pmod p$
よって $a^{-1} \equiv a^{p-2} \pmod p$

■オイラー関数

$\phi(n)$ は 1 から n までの自然数のうち n と互いに素なものの個数を表す
$\phi(n) = n\prod_{p|n}(1-\frac{1}{p})$
$(a,n) = 1$ ならば $a^{\phi(n)} \equiv 1 \pmod n$

■位数

$a^x \equiv 1 \pmod m$ となる最小の x を $a \pmod m$ の位数という
$a \pmod m$ の位数が$\phi(m)$ のとき、a は m の原始根という
素数 p の原始根は$\phi(p-1)$ 個ある
$x^d \equiv 1 \pmod p$ は、$d \mid (p-1)$ のとき最大限の根を持ち (d 個)、そのうち$\phi(d)$ 個の位数が d

■ウィルソンの定理

$(p-1)! \equiv -1 \pmod p$

■平方剰余

奇素数 p に対し、$x^2 \equiv a \pmod p$ が解を持つ $\Longleftrightarrow a^{\frac{p-1}{2}} \equiv 1 \pmod p$

■Stern-Brocot 木

有理数の区間を頂点とする二分探索木で、次により定義される
根は $\left(\frac{0}{1}, \frac{1}{0}\right)$
頂点 $\left(\frac{a}{b}, \frac{c}{d}\right)$ の子供は $\left(\frac{a}{b}, \frac{a+c}{b+d}\right)$ と $\left(\frac{a+c}{b+d}, \frac{c}{d}\right)$
この木には次のような特徴がある
各頂点の端点は既約分数
全ての既約分数が一度だけ現れる
木の深さに対して分子・分母は単調に増加する
頂点 $\left(\frac{a}{b}, \frac{c}{d}\right)$ について、$ad - bc = 1$

■スターリングの公式

$n! \approx \sqrt{2\pi n}\frac{n^n}{e^n}$

■nCk mod p

${}_nC_k$が素数 p で割りきれる $\Longleftrightarrow k + (n-k)$ を p 進数で表現した時にキャリーが発生する
また $n = \sum n_i p^i, k = \sum k_i p^i$とすると、
${}_nC_k \equiv \prod {}_{n_i}C_{k_i} \pmod p$

6.2.2 階乗剰余

素数 p に対し、$n! = ap^k$ とあらわした時の $a \pmod p$ の値を求める
$O(p \log n)$

```
int modFact(int n, int p) {
  int res = 1;
  while (n > 0) {
    for (int i = 1, m = n % p; i <= m; i++) res = (res * i) % p;
    if ((n /= p) % 2 > 0) res = p - res;
  }
  return res;
}
```

6.2.3 拡張ユークリッドの互助法

$ax + by = \gcd(x, y)$ となるような数の組 $\{a, b, c = \gcd(x, y)\}$ を一組求める

(a, b) の一般解は $(a + d\frac{y}{c}, b - d\frac{x}{c})$

```
int[] exgcd(int x, int y) {
  int a0 = 1, a1 = 0, b0 = 0, b1 = 1, t;
  while (y != 0) {
    t = a0 - x / y * a1; a0 = a1; a1 = t;
    t = b0 - x / y * b1; b0 = b1; b1 = t;
    t = x % y; x = y; y = t;
  }
  if (x < 0) {
    a0 = -a0; b0 = -b0; x = -x;
  }
  return new int[] {a0, b0, x};
}
```

6.2.4 線形連立合同式

線形連立合同式 $Ax \equiv B \pmod M$ の解とその法の組 $\{x, m\}$ を返す

解が存在しない場合は null を返す

```
BigInt[] congruence(BigInt[] A, BigInt[] B, BigInt[] M) {
  BigInt x = 0, m = 1;
  for (int i = 0; i < A.length; i++) {
    BigInt a = A[i] * m, b = B[i] - A[i] * x, d = a.gcd(M[i]);
    if (b % d != 0) return null;
    x += m * (b / d * (a / d).modInv(M[i] / d) % (M[i] / d));
    m *= M[i] / d;
  }
  return new BigInt[] {x % m, m};
}
```

6.2.5 離散対数

$a^x \equiv b \pmod m$ となる最小の x を返す

解が存在しない場合は -1 を返す

```
long modLog(long a, long b, long m) {
  if (b % exgcd(a, m)[2] != 0) return -1;
  if (m == 0) return 0;
  long n = (long)sqrt(m) + 1;
  Map<Long, Long> map = new HashMap<Long, Long>();
  long an = 1;
  for (long j = 0; j < n; j++) {
    if (!map.containsKey(an)) map.put(an, j);
    an = an * a % m;
  }
  long ain = 1, res = Long.MAX_VALUE;
  for (long i = 0; i < n; i++) {
    long[] is = congruence(ain, b, m);
    for (long aj = is[0]; aj < m; aj += is[1]) if (map.containsKey(aj)) {
      long j = map.get(aj);
      res = min(res, i * n + j);
    }
    if (res < Long.MAX_VALUE) return res;
    ain = ain * an % m;
  }
  return -1;
}
```

6.2.6 合成数を法とする合同式の解法

■一次式

$ax + by = n$ が解を持つ $\iff (a, b) \mid n$

$ka \equiv kb \pmod m \iff a \equiv b \pmod{\frac{m}{(k, m)}}$

$kx \equiv l \pmod m$ は $(k, m) \mid l$ のとき (k, m) 個の解を持つ

■多項式

$f(x) \equiv 0 \pmod m$ の根を求めたい

$m = \prod_{i=1}^{k} p_i^{a_i}$ とすると、

各 $f(x) \equiv 0 \pmod{p_i^{a_i}}$ の根の組 $\{x_i\}_{i=1}^{k}$ に対して、1 つ根が定まる

$f(x) \equiv 0 \pmod{p^a}$ の根は、$f(x) \equiv 0 \pmod{p^{a-1}}$ の根 x' に対し、

$f'(x') \equiv 0 \pmod p$ かつ $f(x') \equiv 0 \pmod{p^a} \Rightarrow x' + dp^{a-1} (d = 0, \ldots, p-1)$

$f'(x') \not\equiv 0 \pmod p \Rightarrow x' - \frac{f(x')}{f'(x')}$

6.3 線形代数

6.3.1 公式集

■クラメルの公式

$Ax = b$ について A の第 i 列を b で置き換えた行列を A_i とすると、$x_i = \frac{|A_i|}{|A|}$

三元一次連立方程式はこれで解くべし

■双対変換

$\max\{cx \mid Ax \le b, x \ge 0\} = \min\{yb \mid yA \ge c, y \ge 0\}$

$\max\{cx \mid Ax = b, x \ge 0\} = \min\{yb \mid yA \ge c\}$

$\max\{cx \mid Ax = b\} = \min\{yb \mid yA = c\}$

6.3.2 連立方程式

■$Ax = 0$ の解空間 $X = \{x_0 + \sum a_i x_i\}$ を求める　$O(nmr)$

```
double[][] solutionSpace(double[][] A, double[] b) {
  int n = A.length, m = A[0].length;
  double[][] a = new double[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];//単位行列の成分が (i,id[i]) になる
  fill(id, -1);
  int pi = 0;//ランクに相当
  for (int pj = 0; pi < n && pj < m; pj++) {
    for (int i = pi + 1; i < n; i++) {
      if (abs(a[i][pj]) > abs(a[pi][pj])) {
        double[] t = a[i]; a[i] = a[pi]; a[pi] = t;
      }
    }
    if (abs(a[pi][pj]) < EPS) continue;
    double inv = 1 / a[pi][pj];
    for (int j = 0; j <= m; j++) a[pi][j] *= inv;
    for (int i = 0; i < n; i++) if (i != pi) {
      double d = a[i][pj];
      for (int j = 0; j <= m; j++) a[i][j] -= d * a[pi][j];
    }
    id[pi++] = pj;
  }
  for (int i = pi; i < n; i++) if (abs(a[i][m]) > EPS) return null;
  double[][] X = new double[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;
}
```

■Toeplitz 行列 A に対して、$Ax = b$ を解く　$O(n^2)$

$A[i, j] = A[i - j + ZERO]$

$A[0, 0] \neq 0$

```
double[] levinson(double[] A, double[] b) {
  int n = (A.length + 1) / 2, ZERO = n - 1;
  double[] x = new double[n], fs = new double[n], bs = new double[n];
  fs[0] = bs[0] = 1 / A[ZERO];
  x[0] = b[0] / A[ZERO];
  for (int i = 1; i < n; i++) {
    double ef = 0, eb = 0, ex = 0;
    for (int j = 0; j < i; j++) {
      ef += fs[j] * A[i - j + ZERO];
      eb += bs[j] * A[-1 - j + ZERO];
      ex += x[j] * A[i - j + ZERO];
    }
    if (abs(ef * eb - 1) < EPS) return null;
    for (int j = i; j >= 0; j--) {
      double af = (j < i ? fs[j] : 0), ab = (j != 0 ? bs[j - 1] : 0);
      fs[j] = af / (1 - ef * eb) - ef * ab / (1 - ef * eb);
      bs[j] = ab / (1 - ef * eb) - eb * af / (1 - ef * eb);
    }
    for (int j = 0; j <= i; j++) x[j] += (b[i] - ex) * bs[j];
  }
  return x;
}
```

6.3.3 単体法

$\max\{cx \mid Ax \leq b, x \geq 0\}$ を解く

解が存在しない場合は null を返す

```
double[] simplex(double[][] A, double[] b, double[] c) {
  int n = A.length, m = A[0].length + 1, r = n, s = m - 1;
  double[][] D = new double[n + 2][m + 1];
  int[] ix = new int[n + m];
  for (int i = 0; i < n + m; i++) ix[i] = i;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m - 1; j++) D[i][j] = -A[i][j];
    D[i][m - 1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; j++) D[n][j] = c[j];
  D[n + 1][m - 1] = -1;
  for (double d;;) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      D[r][s] = 1.0 / D[r][s];
      for (int j = 0; j <= m; j++) if (j != s) D[r][j] *= -D[r][s];
      for (int i = 0; i <= n + 1; i++) if (i != r) {
        for (int j = 0; j <= m; j++) if (j != s) D[i][j] += D[r][j] * D[i][s];
        D[i][s] *= D[r][s];
      }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; j++) if (s < 0 || ix[s] > ix[j]) {
```

```
      if (D[n + 1][j] > EPS || D[n + 1][j] > -EPS && D[n][j] > EPS) s = j;
    }
    if (s < 0) break;
    for (int i = 0; i < n; i++) if (D[i][s] < -EPS) {
      if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS
          || d < EPS && ix[r + m] > ix[i + m]) r = i;
    }
    if (r < 0) return null;//非有界
  }
  if (D[n + 1][m] < -EPS) return null;//実行不能
  double[] x = new double[m - 1];
  for (int i = m; i < n + m; i++) if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
  return x;//値は D[n][m]
}
```

6.3.4 漸化式

漸化式 $a_{i+n} = \sum k_j a_{i+j} + d$ に対し、$a_m = \sum c_i a_i + c_n d$ と表した時の $\{c_i\}$ を求める

$O(n^2 \log m)$

$\{k_i\}$ が疎な場合畳込みに FFT を用いることで $O(n \log n \log m)$ となる

```
double[] recFormula(double[] k, long m) {
  int n = k.length;
  double[] c = new double[n + 1];
  if (m < n) c[(int)m] = 1;
  else {
    double[] b = recFormula(k, m >>> 1);
    double[] a = new double[n * 2];
    int s = (int)(m & 1);
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        a[i + j + s] += b[i] * b[j];
      }
      c[n] += b[i];
    }
    c[n] = (c[n] + 1) * b[n];
    for (int i = n * 2 - 1; i >= n; i--) {
      for (int j = 0; j < n; j++) {
        a[i - n + j] += k[j] * a[i];
      }
      c[n] += a[i];
    }
    System.arraycopy(a, 0, c, 0, n);
  }
  return c;
}
```

6.3.5 整数解

$O(n^5)$ くらい

解がない場合は null, 複数ある場合はそのうちの 1 つを返す

```
BigInteger[] intSolve(BigInteger[][] A, BigInteger[] b) {
  int n = A.length, m = A[0].length;
  BigInteger[][] a = new BigInteger[n][m + 1];
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) a[i][j] = A[i][j];
    a[i][m] = b[i];
  }
  Stack<Trans> trans = new Stack<Trans>();
  for (int p = 0; p < m && p < n; p++) {
    for (;;) {
      int pi = p, pj = p;
      for (int i = p; i < n; i++) {
        for (int j = p; j < m; j++) {
          if (a[i][j].signum() != 0 && (a[pi][pj].signum() == 0
              || a[pi][pj].abs().compareTo(a[i][j].abs()) > 0)) {
            pi = i;
            pj = j;
          }
        }
      }
      swap(a, pi, p);
      for (int i = p; i < n; i++) swap(a[i], pj, p);
      if (pj != p) trans.push(new Trans(0, pj, p, null));
      if (a[p][p].signum() == 0) break;
      boolean end = true;
      for (int i = p + 1; i < n; i++) {
        BigInteger d = a[i][p].divide(a[p][p]);
        end = end && a[i][p].signum() == 0;
        for (int j = p; j <= m; j++) {
          a[i][j] = a[i][j].subtract(a[p][j].multiply(d));
        }
      }
      for (int j = p + 1; j < m; j++) {
        BigInteger d = a[p][j].divide(a[p][p]);
        end = end && a[p][j].signum() == 0;
        trans.push(new Trans(1, j, p, d));
        for (int i = p; i < n; i++) {
          a[i][j] = a[i][j].subtract(a[i][p].multiply(d));
        }
      }
```

```
      if (end) break;
    }
  }
  BigInteger[] res = new BigInteger[m];
  fill(res, ZERO);
  for (int i = 0; i < m && i < n; i++) {
    if (a[i][i].signum() < 0) {
      a[i][i] = a[i][i].negate();
      a[i][m] = a[i][m].negate();
    }
    if (a[i][i].signum() == 0) {
      if (a[i][m].signum() != 0) return null;
    } else if (a[i][m].mod(a[i][i]).signum() == 0) {
      res[i] = a[i][m].divide(a[i][i]);
    } else return null;
  }
  for (int i = min(m, n); i < n; i++) if (a[i][m].signum() != 0) return null;
  while (!trans.isEmpty()) {
    Trans t = trans.pop();
    if (t.type == 0) swap(res, t.a, t.b);
    else res[t.b] = res[t.b].subtract(res[t.a].multiply(t.c));
  }
  return res;
}
```

6.4 解析

6.4.1 高速フーリエ変換

■ $O(n \log n)$

長さは 2 のべき乗であること

順変換のときは sign=1, 逆変換のときは sign=-1 を指定する

逆変換の際には結果を 1/n 倍すること

```
void fft(int sign, double[] real, double[] imag) {
  int n = real.length, d = Integer.numberOfLeadingZeros(n) + 1;
  double theta = sign * 2 * PI / n;
  for (int m = n; m >= 2; m >>= 1, theta *= 2) {
    for (int i = 0, mh = m >> 1; i < mh; i++) {
      double wr = cos(i * theta), wi = sin(i * theta);
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        double xr = real[j] - real[k], xi = imag[j] - imag[k];
        real[j] += real[k];
        imag[j] += imag[k];
        real[k] = wr * xr - wi * xi;
        imag[k] = wr * xi + wi * xr;
      }
    }
  }
  for (int i = 0; i < n; i++) {
    int j = Integer.reverse(i) >>> d;
    if (j < i) {
      double tr = real[i]; real[i] = real[j]; real[j] = tr;
      double ti = imag[i]; imag[i] = imag[j]; imag[j] = ti;
    }
  }
}
```

6.4.2 テイラー展開

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$\log(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n \ (|x| < 1)$$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \ (|x| < 1)$$

6.4.3 数値積分

■シンプソンの公式

関数 $f(x)$ を二次関数で近似することにより数値積分を行う

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f(x_n)\right]$$

ただし n は $[a,b]$ を均等に偶数個に分割した際の部分区間の個数、

$h = \frac{b-a}{n}$、$x_i = a + ih$

誤差は分割幅 h に対して $O(h^4)$

7 文字列

7.1 KMP

テキスト t 中からパターン p を探す

fail[i] は prefix と suffix が一致する最大の長さ $L \in [0, i)$ になる

また、n-fail[n] は文字列の周期を表す

文字列 s の接尾辞のうち回文になるものを列挙するには、rev(s) を s から検索し、i=n-1 になったら fail をたどって i-j の値を列挙すればよい

```
class KMP {
  int m;
  char[] p;
  int[] fail;
  KMP(char[] p) {
    m = p.length;
    this.p = p;
    fail = new int[m + 1];
    int crt = fail[0] = -1;
    for (int i = 1; i <= m; i++) {
      while (crt >= 0 && p[crt] != p[i - 1]) crt = fail[crt];
      fail[i] = ++crt;
    }
  }
  int searchFrom(char[] t) {
    int n = t.length, count = 0;
    for (int i = 0, j = 0; i < n; i++) {
      while (j >= 0 && t[i] != p[j]) j = fail[j];
      if (++j == m) {
        count++;
        j = fail[j];
      }
    }
    return count;
  }
}
```

7.2 AhoCorasick

テキスト t 中から複数のパターン ps を探す

```
class AhoCorasick {
  int m;
  Node root;
  AhoCorasick(char[][] ps) {
    m = ps.length;
    root = new Node();
    for (int i = 0; i < m; i++) {
      Node t = root;
      for (char c : ps[i]) {
        if (!t.cs.containsKey(c)) t.cs.put(c, new Node());
        t = t.cs.get(c);
      }
      t.accept.add(i);
    }
    Queue<Node> que = new LinkedList<Node>();
    que.offer(root);
    while (!que.isEmpty()) {
      Node t = que.poll();
      for (Map.Entry<Character, Node> e : t.cs.entrySet()) {
        char c = e.getKey();
        Node u = e.getValue();
        que.offer(u);
        Node r = t.fail;
        while (r != null && !r.cs.containsKey(c)) r = r.fail;
        if (r == null) u.fail = root;
        else u.fail = r.cs.get(c);
        u.accept.addAll(u.fail.accept);
      }
    }
  }
  int[] searchFrom(char[] t) {
    int n = t.length;
    int[] count = new int[m];
    Node u = root;
    for (int i = 0; i < n; i++) {
      while (u != null && !u.cs.containsKey(t[i])) u = u.fail;
      if (u == null) u = root;
      else u = u.cs.get(t[i]);
      for (int j : u.accept) count[j]++;
    }
    return count;
  }
  class Node {
    Map<Character, Node> cs = new TreeMap<Character, Node>();
    List<Integer> accept = new ArrayList<Integer>();
    Node fail;
  }
}
```

7.3 SuffixArray

O(n log n)

```
class SuffixArray {
  int n;
  char[] cs;
  int[] si, is;//ソート後と元のインデックスの対応
  SuffixArray(char[] t) {
    n = t.length;
    cs = new char[n + 1];
    System.arraycopy(t, 0, cs, 0, n);
```

```
    cs[n] = 0;//番兵 (他の全ての値より小さくすること)
    is = new int[n + 1];
    for (int i = 0; i <= n; i++) is[i] = cs[i];
    si = indexSort(is);
    int[] a = new int[n + 1], b = new int[n + 1];
    for (int h = 0; ; ) {
      for (int i = 0; i < n; i++) {
        int x = si[i + 1], y = si[i];
        b[i + 1] = b[i];
        if (is[x] > is[y] || is[x + h] > is[y + h]) b[i + 1]++;
      }
      for (int i = 0; i <= n; i++) is[si[i]] = b[i];
      if (b[n] == n) break;
      h = max(1, h << 1);
      for (int k = h; k >= 0; k -= h) {
        fill(b, 0);
        b[0] = k;
        for (int i = k; i <= n; i++) b[is[i]]++;
        for (int i = 0; i < n; i++) b[i + 1] += b[i];
        for (int i = n; i >= 0; i--) {
          a[--b[si[i] + k > n ? 0 : is[si[i] + k]]] = si[i];
        }
        int[] tmp = si; si = a; a = tmp;
      }
    }
  }
  int[] hs;//hs[i]:=ソート後の suffix の i と i+1 の LCP
  void buildHs() {
    hs = new int[n + 1];
    for (int i = 0, h = 0; i < n; i++) {
      for (int j = si[is[i] - 1]; cs[i + h] == cs[j + h]; h++);
      hs[is[i] - 1] = h;
      if (h > 0) h--;
    }
  }
  RMQ rmq;
  void buildRMQ() {
    rmq = new RMQ(hs);//値を返す版
  }
  //位置 i,j から始まる部分列の LCP を求める
  int getLCP(int i, int j) {
    if (i == j) return n - i;
    return rmq.query(min(is[i], is[j]), max(is[i], is[j]));
  }
}
```

7.4 回文

回文の性質:$[a,b),[a,c),[b,d)$ がそれぞれ回文なら $[c,d)$ も回文

O(n) で回文の長さを求める

len[i]:i/2 の位置を中心とする回文の長さ

```
int[] palindrome(char[] cs) {
  int n = cs.length;
  int[] len = new int[n * 2];
  for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
    while (i - j >= 0 && i + j + 1 < n * 2
        && cs[(i - j) / 2] == cs[(i + j + 1) / 2]) j++;
    len[i] = j;
    for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++) {
      len[i + k] = min(len[i - k], j - k);
    }
  }
  return len;
}
```

8 その他

8.1 マトロイド

8.1.1 独立公理

$(M1)$ $\emptyset \in F$

$(M2)$ $X \subseteq Y \in F \Rightarrow X \in F$

$(M3)$ $X,Y \in F, |X| > |Y| \Rightarrow \exists x \in X \setminus Y.\ Y \cup \{x\} \in F$

$(M3')$ 各 $X \subseteq E$ に対して、X のどの基も元数は等しい

8.1.2 ランク公理

$(R1)$ $r(X) \leq |X|$

$(R2)$ $X \subseteq Y \Rightarrow r(X) \leq r(Y)$

$(R3)$ $r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y)$

8.1.3 マトロイド交差

二つのマトロイドの交差 (共通部分) はマトロイドにならないかもしれないが、以下のアルゴリズムで最大の独立集合が求まる。

1. $X = \emptyset$ と初期化
2. $E \cup \{s,t\}$ を頂点とし、以下のように辺を張ったグラフで s から t への最短路を求める
   (s,y) : X に y を追加しても F_1で独立
   (y,t) : X に y を追加しても F_2で独立
   (x,y) : X に y を追加すると F_1で独立でなくなるが、x を取り除けば独立に戻る
   (y,x) : X に y を追加すると F_2で独立でなくなるが、x を取り除けば独立に戻る
3. 最短路が存在しなければ、X が最大
   存在したら、$X := X \cup \{y_0,\cdots,y_m\} \setminus \{x_1,\cdots,x_m\}$ と更新して 2 へ

8.1.4 マトロイド分割

$X = X_1 \cup \cdots \cup X_k$で $X_i \in F_i$となる X の分割が存在 $\Leftrightarrow X \in F$ と定義すると、これはマトロイドになる。

最大の独立集合は、$E' = E \times \{1, \cdots, k\}$ に対する以下の 2 つのマトロイド交差によって求まる。

$F_1 = \{Q \subseteq E' : \text{全ての } i \text{ で } Q_i \in F_i\}$

$F_2 = \{Q \subseteq E' : \text{全ての } i \neq j \text{ で } Q_i \cap Q_j = \emptyset\}$

ただし、$Q_i = \{e \in E' : (e, i) \in Q\}$

8.2 組合せゲーム理論

8.2.1 勝てるか判定

勝てる ⇔ 負ける状態への手が存在する

負ける ⇔ 勝てる状態への手しか存在しない

8.2.2 Nim

複数ある山の一つから任意個を取り除き、取り除けなくなった人の負けというゲーム

山の大きさについて XOR を取り、0 ならば負け、0 以外なら勝ち

最善手は XOR の最上位ビットの位置が立っている山から XOR が 0 になるように取り除けばよい

最後に取り除いた人が負けの場合は、XOR が 0 以外かつ 0,1 以外の山がある、もしくは、XOR が 0 かつ全ての山が 0,1 ならば勝ち

8.2.3 Grundy 数

Grundy 数が a のゲームは山の大きさが a の Nim に等しい

現在の状態から進める全ての状態の Grundy 数の中に現れない最小の数が、その状態の Grundy 数となる

複数のゲームに分離する場合はそれらの Grundy 数の XOR を使えばよい

複数のゲームがあり、一度に一つのゲームを進める場合は XOR を取ればよい

一度に $1 \sim M$ 個のゲームを進める場合は 2 進数の各桁ごとに $\mod (M+1)$ で和を取ればよい

8.3 日付

8.3.1 日に換算

うるう年を考慮して日に換算する

```
int days(int y, int m, int d) {
  if (m < 3) {
    y--;
    m += 12;
  }
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 2) / 5 + d;
}
```

8.3.2 java.util.GregorianCalendar

曜日は [Sunday, Saturday] = [1, 7]

月は [January, December] = [0, 11]

1582 年 10 月 4 日の次は 1582 年 10 月 15 日になり、ここでユリウス歴からグレゴリオ歴になる

純粋なグレゴリオ暦を得るには、`setGregorianChange(new Date(Long.MIN_VALUE))` とする

純粋なユリウス歴を得るには、`setGregorianChange(new Date(Long.MAX_VALUE))` とする

値の変更により無効な値になってもちゃんと処理される

(例) 2008 年 6 月 31 日 = 2008 年 7 月 1 日, 2008 年 7 月 31 日 - 1 月 = 2008 年 6 月 30 日

その月の日数を調べるには `getActualMaximum(Calendar.DAY_OF_MONTH)`

うるう年か調べるには `isLeapYear(year)`

8.4 小物

8.4.1 Scanner

hasNext の仕様が違うので注意

```
class Scanner {
  BufferedReader br;
  StringTokenizer st;
  Scanner(InputStream in) {
    br = new BufferedReader(new InputStreamReader(in));
    eat("");
  }
  void eat(String s) {
    st = new StringTokenizer(s);
  }
  String nextLine() {
    try {
      return br.readLine();
    } catch (IOException e) {
      throw new IOError(e);
    }
  }
  boolean hasNext() {
    while (!st.hasMoreTokens()) {
      String s = nextLine();
      if (s == null) return false;
      eat(s);
    }
    return true;
  }
  String next() {
    hasNext();
    return st.nextToken();
  }
  int nextInt() {
    return Integer.parseInt(next());
  }
  long nextLong() {
    return Long.parseLong(next());
  }
  double nextDouble() {
    return Double.parseDouble(next());
  }
}
```

8.4.2 nCk のビットコンビネーション列挙

昇順に列挙する

```
for (int comb = (1 << k) - 1; comb < 1 << n;) {
  //...
```

```
  int x = comb & -comb, y = comb + x;
  comb = ((comb & ~y) / x >>> 1) | y;
}
```

8.4.3 順列生成

C++ の next_permutation

次の順列を生成し、存在するかどうかを返す

同じ値が複数ある場合も重複なく生成する

全順列を生成したいときは最初にソートすること

```
boolean nextPermutation(int[] is) {
  int n = is.length;
  for (int i = n - 1; i > 0; i--) {
    if (is[i - 1] < is[i]) {
      int j = n;
      while (is[i - 1] >= is[--j]);
      swap(is, i - 1, j);
      rev(is, i, n);
      return true;
    }
  }
  rev(is, 0, n);
  return false;
}
```

8.4.4 さいころ

立方体の全回転状態の生成を行う

```
class Dice {
  //top, front, left, right, back, bottom
  int[] is = new int[6];
  Dice(int...is) {
    this.is = is;
  }
  void rollX() {
    roll(0, 1, 5, 4);
  }
  void rollY() {
    roll(0, 3, 5, 2);
  }
  void rollZ() {
    roll(1, 3, 4, 2);
  }
  void roll(int a, int b, int c, int d) {
    int t = is[d];
    is[d] = is[c];
    is[c] = is[b];
    is[b] = is[a];
    is[a] = t;
  }
  Dice[] rollAll() {
    Dice[] dices = new Dice[24];
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < 4; j++) {
        dices[i * 4 + j] = new Dice(is.clone());
        rollZ();
      }
      if ((i & 1) > 0) rollX();
      else rollY();
    }
    return dices;
  }
}
```

8.4.5 ヨセフス数

n 人の輪から m 人ごとに殺していった時に k 番目に殺される人の番号 $[0, n-1]$ を求める

$J(n, m, 0) = -1$

$J(n, m, k) = (J(n-1, m, k-1) + k) \% n$

$O(k)$

```
int josephus(int n, int m, int k) {
  int x = -1;
  for (int i = n - k + 1; i <= n; i++) {
    x = (x + m) % i;
  }
  return x;
}
```

逆ヨセフス数

$O(n)$

n 人の輪から m 人ごとに殺していった時に x 番が殺されるターン数 $[1, n]$ を求める

```
int invJosephus(int n, int m, int x) {
  for (int i = n; ; i--) {
    if (x == i) return n - i;
    x = (x - m % i + i) % i;
  }
}
```

8.4.6 最大長方形

高さの配列が与えられるので、その中に含まれる最大長方形の面積を求める

$O(n)$

```
int maxRect(int[] _hs) {
  int n = _hs.length;
  int res = 0;
  int[] hs = new int[n + 1];//n 番に番兵 0 を置く
  System.arraycopy(_hs, 0, hs, 0, n);
  Stack<Integer> sx = new Stack<Integer>();
  Stack<Integer> sh = new Stack<Integer>();
  sh.push(0);
  for (int i = 0; i <= n; i++) {
    int x = i;
    while (sh.peek() > hs[i]) {
      res = max(res, sh.pop() * (i - (x = sx.pop())));
    }
    sx.push(x);
    sh.push(hs[i]);
  }
  return res;
}
```

9 探索

9.1 $\alpha\beta$探索

```
int alphaBeta(State s, int alpha, int beta) {
  if (s.finished()) return s.calcScore();
  for (State t : s.next()) {
    alpha = max(alpha, -alphaBeta(t, -beta, -alpha));
    if (alpha >= beta) break;
  }
  return alpha;
}
```

9.2 最小頂点彩色問題

隣接する頂点が同じ色にならないように最小の色数で彩色する

探索のアイディア

1. 連結成分に分解して解く

2.MA 順序でソートして探索開始

3. 色の数で枝狩りをした深さ優先探索

```
int[] coloring(boolean[][] g) {
  this.g = g;
  int n = g.length;
  res = new int[n];
  id = new int[n + 1];//元のインデックス (n 番は番兵)
  int[] b = new int[n + 1];//連結度
  for (int i = 0; i <= n; i++) id[i] = i;
  for (s = 0, t = 1; t <= n; t++) {//MA 順序でソート
    int p = t;//最大連結度の点
    for (int i = t; i < n; i++) {
      if (g[id[t - 1]][id[i]]) b[id[i]]++;
      if (b[id[p]] < b[id[i]]) p = i;
    }
    swap(id, t, p);
    if (b[id[t]] == 0) {//連結度 0 なので [s,t) が連結成分
      min = n + 1;
      dfs(new int[n], s, 0);
      s = t;
    }
  }
  return res;
}
void dfs(int[] is, int p, int k) {//is:現在の塗り方,p:次塗る点,k:使った色の数
  if (k >= min) return;
  if (p == t) {
    for (int i = s; i < t; i++) res[id[i]] = is[i];
    min = k;
  } else {
    boolean[] used = new boolean[k + 1];
    for (int i = 0; i < p; i++) if (g[id[p]][id[i]]) used[is[i]] = true;
    for (int i = 0; i <= k; i++) if (!used[i]) {//今までに使った色 +1 まで調べる
      int[] js = is.clone();
      js[p] = i;
      dfs(js, p + 1, max(k, i + 1));
    }
  }
}
```

9.3 完全マッチングの個数

完全マッチングの個数や、安定集合の個数など、ビット DP の問題を一般的に解く

```
long match(V[] vs) {
  int n = vs.length;
  for (int i = 0; i < n; i++) vs[i].id = i;
  for (V v : vs) for (V u : v) v.max = max(v.max, u.id);
  long[] crt = {1};
  int[] cvi = new int[n], civ = new int[n];
  int cn = 0;
  fill(cvi, -1);
  for (int k = 0; k < n; k++) {
    int nn = 0;
    int[] nvi = new int[n], niv = new int[n];
```

```
    fill(nvi, -1);
    for (int i = 0; i <= k; i++) if (vs[i].max > k) {
      niv[nn] = i;
      nvi[i] = nn++;
    }
    long[] next = new long[1 << nn];
    loop : for (int i = 0; i < 1 << cn; i++) if (crt[i] > 0) {
      int i2 = 0;
      boolean need = false;
      for (int j = 0; j < cn; j++) if ((i >> j & 1) != 0) {
        if (nvi[civ[j]] < 0) {
          if (need) continue loop;
          need = true;
        } else {
          i2 |= 1 << nvi[civ[j]];
        }
      }
      if (need) {
        next[i2] += crt[i];
      } else {
        if (nvi[k] >= 0) {
          next[i2 | 1 << nvi[k]] += crt[i];
        }
        for (V u : vs[k]) if (u.id < k && (i >> cvi[u.id] & 1) != 0) {
          next[i2 ^ 1 << nvi[u.id]] += crt[i];
        }
      }
    }
    crt = next;  cvi = nvi; civ = niv; cn = nn;
  }
  return crt[0];
}
class V extends ArrayList<V> {
  int id, max = -1;
}
```

10 Visualizer

```
class Vis extends JFrame {
  BufferedImage img;
  Graphics2D g;
  boolean stop;
  Vis() {
    img = new BufferedImage(500, 500, BufferedImage.TYPE_INT_RGB);
    g = (Graphics2D)img.getGraphics();
    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    clear();
    JComponent c = new JComponent() {
      protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(img, 0, 0, null);
      }
    };
    c.setPreferredSize(new Dimension(500, 500));
    add(c);
    addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e) {
        stop = false;
      }
    });
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
  void vis() {
    repaint();
    stop = true;
    try {
      while (stop) Thread.sleep(10);
    } catch (Exception e) {
    }
  }
  void clear() {
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, 500, 500);
    g.setColor(Color.BLACK);
  }
}
```

11 便利な表

11.1 書式

'-'	左揃え
'+'	常に符号を含む || ' '	非負の数の前に空白
'0'	左端を 0 で埋める
','	1000 ごとに区切る

11.2 正規表現

11.2.1 文字クラス

[abc]	a、b、または c (単純クラス)
[^abc]	a、b、c 以外の文字 (否定)
[a-zA-Z]	a ～ z または A ～ Z (範囲)
[a-d[m-p]]	a ～ d、または m ～ p:[a-dm-p] (結合)
[a-z&&[def]]	d、e、f (交差)
[a-z&&[^bc]]	b と c を除く a ～ z:[ad-z] (減算)
[a-z&&[^m-p]]	m ～ p を除く a ～ z:[a-lq-z] (減算)
.	任意の文字
\d	数字:[0-9]
\D	数字以外:[^0-9]
\s	空白文字:[\t\n\x0B\f\r]
\S	非空白文字:[^\s]
\w	単語構成文字:[a-zA-Z_0-9]
\W	非単語文字:[^\w]

11.2.2 境界

^	行の先頭
$	行の末尾
\b	単語境界
\B	非単語境界
\A	入力の先頭
\G	前回のマッチの末尾
\Z	最後の行末記号がある場合は、それを除く入力の末尾
\z	入力の末尾

11.2.3 数量子

なにも指定しないと最長一致

後ろに?をつけると最短一致

後ろに + をつけると強欲 (マッチが失敗しても戻らない) になる

X?	X、1 または 0 回
X*	X、0 回以上
X+	X、1 回以上
X{n}	X、n 回
X{n,}	X、n 回以上
X{n,m}	X、n 回以上、m 回以下

11.2.4 その他

キャプチャグループは 0 が全体を表し、カッコを開いた順に 1 から番号が付く

XY	X の直後に Y
X|Y	X または Y
(X)	X、キャプチャグループ
(?:X)	X、非キャプチャグループ
\n	マッチした n 番目のキャプチャグループ
(?i)	CASE_INSENSITIVE に設定
(?-i)	CASE_INSENSITIVE を解除

11.3 数表

n	$\log_{10} n$	$n!$	${}_nC_{(n/2)}$	$LCM(1 \ldots n)$	P_n	B_n
2	0.30	2	2	2	2	2
3	0.48	6	3	6	3	5
4	0.60	24	6	12	5	15
5	0.70	120	10	60	7	52
6	0.78	720	20	60	11	203
7	0.85	5040	35	420	15	877
8	0.90	40320	70	840	22	4140
9	0.95	362880	126	2520	30	21147
10	1.00	3628800	252	2520	42	115975
11		39916800	462	27720	56	678570
12		479001600	924	27720	77	4213597
15			6435	360360	176	1382958545
20			184756	232792560	627	
25			5200300		1958	
30			155117520		5604	
40					37338	
50					204226	
70					4087968	
100					190569292	