Team Contest Reference

Team Ortec 1

Contents

Formulas for Geometric Shapes

$$\text{oppervlakte cirkel} : \pi r^2$$
$$\text{omtrek cirkel} : \pi d$$
$$\text{oppervlakte ellips} : \pi ab$$
$$\text{oppervlakte kegel} : \pi r^2 + \pi r\sqrt{r^2+h^2}$$
$$\text{inhoud kegel} : \frac{1}{3}\pi r^2 h$$
$$\text{oppervlakte bol} : 4\pi r^2$$
$$\text{inhoud bol} : \frac{4}{3}\pi r^3$$
$$\text{oppervlakte cillinder} : 2\pi rh + 2\pi r^2$$
$$\text{inhoud cilinder} : \pi r^2 h$$
$$\text{nr regions by } n \text{ lines} = \frac{1}{2}n(n+1)+1$$
$$\text{nr bounded regions by } n \text{ lines} = \frac{1}{2}(n^2-3n+2)$$

More Formulas

$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$$
$$\sum_{i=1}^{n} i^3 = \frac{1}{4}n^2(n+1)^2$$
$$\text{least common multiple} : \text{lcm}(m,n) = \frac{|m\cdot n|}{\text{gcd(m, n)}}$$
$$\text{Catalan number} : C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n}\frac{n+k}{k}$$
$$\text{Catalan numbers} : C = \{1,1,2,5,14,42,132,429,1430,4862,16796\}$$
$$\text{Triangle numbers} : T = \{1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136\}$$
$$\text{Triangle numbers} : T_n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \binom{n+1}{2}$$
$$\text{digits in } n! : \lceil \log_{10}(n!)\rceil = \lceil \log_{10}(n\cdot(n-1)\cdot\ldots\cdot 1)\rceil$$
$$= \lceil \log_{10}(n) + \log_{10}(n-1) + \ldots + \log_{10}(1)\rceil$$

Lagrange Polynomial

Given a set of $(k+1)$ data points $(x_0,y_0),(x_1,y_1),...,(x_k,y_k)$ where no two x_j are the same. Find a polynomial of degree k that has all $(k+1)$ points.

$$L(x) = \sum_{j=0}^{k} y_j \ell_j(x)$$
$$\ell(x) = \prod_{\substack{0\le m\le k \\ m\neq j}} \frac{x-x_m}{x_j-x_m}$$

Example

We wish to interpolate $f(x)=x^2$ over the range $1\le x\le 3$, given these three points:

$$x_0 = 1 \qquad f(x_0) = 1$$
$$x_1 = 2 \qquad f(x_1) = 4$$
$$x_2 = 3 \qquad f(x_2) = 9$$

Solution:
$$L(x) = 1\cdot\frac{x-2}{1-2}\cdot\frac{x-3}{1-3} + 4\cdot\frac{x-1}{2-1}\cdot\frac{x-3}{2-3} + 9\cdot\frac{x-1}{3-1}\cdot\frac{x-2}{3-2} = x^2$$

Computational Geometry

Triangles

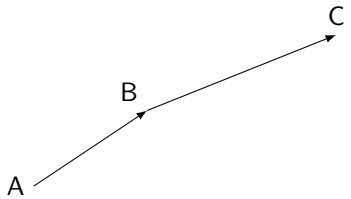
For any triangle with angles α,β,γ and opposing edges a,b,c:

$$a^2 = b^2 + c^2 - 2bc\cdot\cos(\alpha)$$

Cross product

$$a\times b = \begin{bmatrix} a_x \\ a_b \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Links of rechts ombuigen



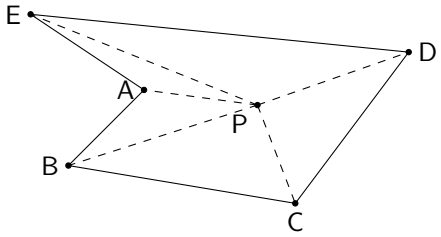
$$\overrightarrow{AB} = \begin{bmatrix} p \\ q \end{bmatrix}$$
$$\vec{n} = \begin{bmatrix} q \\ -p \end{bmatrix}$$

$\vec{n}\cdot\overrightarrow{BC} < 0 \Rightarrow$ linksaf

$\vec{n}\cdot\overrightarrow{BC} > 0 \Rightarrow$ rechtsaf

Punt in concaaf/convex polygon test

Tel het aantal doorsnijdingen van polygon met lijn P naar oneindig. Als het aantal doorsnijdingen oneven is, dan $P \in ABCDE$.



$\alpha = \angle APB + ... + \angle DPE + \angle EPA$
$\alpha = 0 \Rightarrow P \notin ABCDE$
$\alpha = 2\pi \Rightarrow P \in ABCDE$

Centroid of polygon

The centroid or geometric center of a plane figure is the arithmetic mean ("average") position of all the points in the shape. Informally, it is the point at which an infinitesimally thin cutout of the shape could be perfectly balanced on the tip of a pin.

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$A = \frac{1}{2}\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

Bit Operations

```
int number = 191;        // bit representatie:    10111111
number |= 1 << 6;        // nieuwe representatie: 11111111
number &= ~(1 << 5);     // nu:                   11101111
number ^= 1 << 1;        // nu:                   11101101
number >>= 3;            // nu:                      11011
number <<= 3;            // nu:                   11011000
```

```
public static int BitCount(uint x) {
    int c = 0;
    while (x != 0) c++; x &= x - 1; // clear the least significant bit
    return c;
}
```

Raaklijn aan cirkel

```
public class Circle{
    long x,y,r;
    public Circle(long x, long y, long r){
        this.x = x;
        this.y = y;
        this.r = r;
    }
    public Line2D.Double tangent1(long xp, long yp){
        xp -= x; yp -= y;
        Line2D.Double line = new Line2D.Double();
        double a = (xp*xp)+(yp*yp);
        double b = -2*r*r*xp;
        double c = r*r*r*r-yp*yp*r*r;
        double d = b*b-4*a*c;
        double x1 = (-b+Math.sqrt(d))/(2*a);
        double y1 = Math.sqrt(r*r-x1*x1);
        line.setLine(xp+x,yp+y,x1+x,y1+y);
        return line;
    }
    public Line2D.Double tangent2(long xp, long yp){
        xp-=x; yp-=y;
        Line2D.Double line = new Line2D.Double();
        double a = (xp*xp)+(yp*yp);
        double b = -2*r*r*xp;
        double c = r*r*r*r-yp*yp*r*r;
        double d = b*b-4*a*c;
        double x1 = (-b-Math.sqrt(d))/(2*a);
        double y1 = Math.sqrt(r*r-x1*x1);
        line.setLine(xp+x,yp+y,x1+x,y1+y);
        return line;
    }
}
```

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$
$$A = (b\cos\gamma, b\cos\gamma)$$
$$B = (a, 0)$$
$$C = (0, 0)$$

Point to line distance:

$$distance(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

```
static final double EPS = 1e-10;
public static int sign(double a) {
  return a < -EPS ? -1 : a > EPS ? 1 : 0;
}
```

```
public static class Point implements Comparable<Point> {
    public double x, y;
    public Point(double x, double y) { this.x = x; this.y = y; }
    public Point minus(Point b) { return new Point(x - b.x, y - b.y); }
    public double cross(Point b) { return x * b.y - y * b.x; }
    public double dot(Point b) { return x * b.x + y * b.y; }
    public Point rotateCCW(double angle) {
        return new Point(x * Math.cos(angle) - y * Math.sin(angle),
                         x * Math.sin(angle) + y * Math.cos(angle));
    }
    @Override public int compareTo(Point o) {
        return Double.compare(x, o.x) != 0 ? Double.compare(x, o.x) :
    Double.compare(y, o.y);
    }
}
```

```
public static class Line {
    public double a, b, c;
    public Line(double _a, double _b, double _c) {a = _a;b = _b;c = _c;}
    public Line(Point p1, Point p2) {
        a = +(p1.y - p2.y);
        b = -(p1.x - p2.x);
        c = p1.x * p2.y - p2.x * p1.y;
    }
    public Point intersect(Line line) {
        double d = a * line.b - line.a * b;
        if (sign(d) == 0) return null;
        double x = -(c * line.b - line.c * b) / d;
        double y = -(a * line.c - line.a * c) / d;
        return new Point(x, y);
    }
}
```

```
public static class Triangle {
    Point a, b, c;
    public Triangle(Point _a, Point _b, Point _c) {a = _a;b = _b;c = _c;}
    public boolean pointInside(Point p) { // Precondition: a, b, c CCW
        return ccw(a, b, p) && ccw(b, c, p) && ccw(c, a, p);
    }
}
```

```
// Returns -1 for cw, 0 for straight line, 1 for counter-cw order
static int orientation(Point a, Point b, Point c) {
    Point AB = b.minus(a);
    Point AC = c.minus(a);
    return sign(AB.cross(AC));
}
static boolean cw(Point a, Point b, Point c) {
    return orientation(a, b, c) < 0;
}
static boolean ccw(Point a, Point b, Point c) {
    return orientation(a, b, c) > 0;
}
static boolean isCrossIntersect(Point a, Point b, Point c, Point d) {
    return orientation(a, b, c) * orientation(a, b, d) < 0 &&
           orientation(c, d, a) * orientation(c, d, b) < 0;
}
static boolean isCrossOrTouchIntersect(Point a, Point b, Point c, Point d)
     {
    if (Math.max(a.x, b.x) < Math.min(c.x, d.x) - EPS ||
        Math.max(c.x, d.x) < Math.min(a.x, b.x) - EPS ||
        Math.max(a.y, b.y) < Math.min(c.y, d.y) - EPS ||
        Math.max(c.y, d.y) < Math.min(a.y, b.y) - EPS) { return false; }
    return orientation(a, b, c) * orientation(a, b, d) <= 0 &&
           orientation(c, d, a) * orientation(c, d, b) <= 0;
}
static double pointToLineDistance(Point p, Line line) {
    return Math.abs(line.a * p.x + line.b * p.y + line.c) /
                                    fastHypot(line.a, line.b);
}
static double fastHypot(double x, double y) {
    return Math.sqrt(x * x + y * y);
}
static double angleBetween(Point a, Point b) {
    return Math.atan2(a.cross(b), a.dot(b));
}
static double angle(Line line) {
    return Math.atan2(-line.a, line.b);
}
static double signedArea(Point[] points) {
    int n = points.length; double area = 0;
    for (int i = 0, j = n - 1; i < n; j = i++)
        area += (points[i].x - points[j].x) * (points[i].y + points[j].y);
    return area / 2;
}
```

Van Gogh Triangulate

```
public static ArrayList<Triangle> trian(List<Point> p, boolean isCW) {
    ArrayList<Triangle> result = new ArrayList<>();
    while (p.size() > 3) {
        boolean isEar; int li = -1, tryings = 0;
        Point l, v, r; Triangle tr;
        do {
            tryings++;
            if (tryings >= p.size()) return null;
            li++;
            l = p.get(li % p.size());
            v = p.get((li + 1) % p.size());
            r = p.get((li + 2) % p.size());
            tr = new Triangle(l, v, r);
            isEar = ccw(l, v, r) ^ isCW;
            if (isEar) // Further analysis required
                for (int i = 0; i < p.size(); i++)
                    if (tr.pointInside(p.get(i))) {
                        isEar = false;
                        break;
                    }
        } while (!isEar); // Until we've discovered an ear
        result.add(tr); // Guaranteed that we got an ear here
        p.remove((li + 1) % p.size());
    }
    if (p.size() == 3) // The last triangle
        result.add(new Triangle(p.get(0), p.get(1), p.get(2)));
    return result;
}
```

Convex Cut (Cut Part of Polygon by line)

```
// cuts right part of poly (returns left part)
public static Point[] convexCut(Point[] poly, Point p1, Point p2) {
    int n = poly.length;
    List<Point> res = new ArrayList<>();
    for (int i = 0, j = n - 1; i < n; j = i++) {
        int d1 = orientation(p1, p2, poly[j]);
        int d2 = orientation(p1, p2, poly[i]);
        if (d1 >= 0) res.add(poly[j]);
        if (d1 * d2 < 0)
            res.add(new Line(p1, p2).intersect(new Line(poly[j], poly[i]))
    );
    }
    return res.toArray(new Point[res.size()]);
}
```

Classify position

Classifies position of point p against vector a.

```
public static enum Pos {
    LEFT, RIGHT, BEHIND, BEYOND, ORIGIN, DEST, BETWEEN
}

public static Pos classify(Point p, Point a) {
    int s = sign(a.cross(p));
    if (s > 0) return Pos.LEFT;
    if (s < 0) return Pos.RIGHT;
    if (sign(p.x) == 0 && sign(p.y) == 0) return Pos.ORIGIN;
    if (sign(p.x - a.x) == 0 && sign(p.y - a.y) == 0) return Pos.DEST;
    if (a.x * p.x < 0 || a.y * p.y < 0) return Pos.BEYOND;
    if (a.x * a.x + a.y * a.y < p.x * p.x + p.y * p.y) return Pos.BEHIND;
    return Pos.BETWEEN;
}
```

Segment Tree

```
public class SegTree {
    static int[] t;
    static int n, q;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt(); q = sc.nextInt();
        t = new int[2 * n];
        for (int i = 0; i < n; i++) { t[n + i] = sc.nextInt(); }
        build();
        for (int i = 0; i < q; i++) {
            System.out.println(query(sc.nextInt(), sc.nextInt()));
        }
    }
    static void build() {
        for (int i = n - 1; i >= 0; i--) t[i] = t[i<<1] + t[i<<1|1];
    }
    static void modify(int p, int value) {
        for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
    }
    static int query(int l, int r) { // sum on interval [l, r)
        int res = 0;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if ((l&1) > 0) res += t[l++];
            if ((r&1) > 0) res += t[--r];
        }
        return res;
    }
}
```

Binomials and Factorials

$\binom{a}{b} = \frac{a!}{(a-b)!\cdot b!)}$

```
public static long binomial(long n, long m) {
  m = Math.min(m, n - m); long res = 1;
  for (long i = 0; i < m; i++)
    res = res * (n - i) / (i + 1);
  return res;
}

public static long[][] binomialTable(int n) {
  long[][] c = new long[n + 1][n + 1];
  for (int i = 0; i <= n; i++)
    for (int j = 0; j <= i; j++)
      c[i][j] = (j == 0) ? 1 : c[i - 1][j - 1] + c[i - 1][j];
  return c;
}

public static int binomial(int n, int m, int mod) {
  m = Math.min(m, n - m); long res = 1;
  for (int i = n - m + 1; i <= n; i++)
    res = res * i % mod;
  return (int) (res * BigInteger.valueOf(factorial(m,
      mod)).modInverse(BigInteger.valueOf(mod)).intValue() % mod);
}
```

```
// n! % mod
public static int factorial(int n, int mod) {
  long res = 1;
  for (int i = 2; i <= n; i++)
    res = res * i % mod;
  return (int) (res % mod);
}

// n! mod p, p - prime, O(p*log(n)) complexity
public static int factorial2(int n, int p) {
  int res = 1;
  while (n > 1) {
    res = (res * ((n / p) % 2 == 1 ? p - 1 : 1)) % p;
    for (int i = 2; i <= n % p; ++i)
      res = (res * i) % p;
    n /= p;
  }
  return res % p;
}
```

Fibonacci Numbers

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584$$

- When we take a pairs of large consecutive Fibonacci numbers, we can approximate the golden ratio by dividing them.
- The sum of any ten consecutive Fibonacci numbers is divisible by 11.
- Two consecutive Fibonacci numbers are co-prime.
- The Fibonacci numbers in the composite-number (i.e. non-prime) positions are also composite numbers.

Fibbonaci $\mathcal{O}(\log n)$ (use BigInteger)

```
static long fibbo(int n) {
    long i = 1, h = 1, j = 0, k = 0, t;
    while (n > 0) {
        if (n % 2 == 1) {
            t = j * h;
            j = i * h + j * k + t;
            i = i * k + t;
        }
        t = h * h;
        h = 2 * k * h + t;
        k = k * k + t;
        n /= 2;
    }
    return j;
}
```

Primes

```
boolean isPrime(int n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (int i = 3; i * i <= n; i+=2) {
        if (n % i == 0) return false;
    }
    return true;
}
```

Sieve of Eratosthenes

```
int range = 100000000;
boolean zeef[] = new boolean[range];
Arrays.fill(zeef, true); zeef[0] = false; zeef[1] = false;
for(int i=2; i<range; i++){
    if(zeef[i]){
        for(int k = 2*i; k < range; k += i) {
            zeef[k] = false;
        }
    }
}
```

Prime Factorization / Get All Divisors

```
static Map<Long, Integer> factorize(long n) {
    Map<Long, Integer> factors = new LinkedHashMap<>(); // prime_divisor
    -> power
    for (long d = 2; n > 1;) {
        int power = 0;
        while (n % d == 0) {
            ++power;
            n /= d;
        }
        if (power > 0) factors.put(d, power);
        ++d;
        if (d * d > n) d = n;
    }
    return factors;
}

static int[] getAllDivisors(int n) {
    List<Integer> divisors = new ArrayList<>();
    for (int d = 1; d * d <= n; d++) {
        if (n % d == 0) {
            divisors.add(d);
            if (d * d != n) divisors.add(n / d);
        }
    }
    int[] res = new int[divisors.size()];
    for (int i = 0; i < res.length; i++)
        res[i] = divisors.get(i);
    Arrays.sort(res);
    return res;
}
```

Greatest Common Divisor / Least Common Multiple

```
static long gcd(long a, long b) {
    while(b != 0) {
        long c = a % b;
        a = b; b = c;
    }
    return a;
}

static long lcm(long a, long b) {
    return Math.abs(a / gcd(a, b) * b);
}
```

Extended Euclidian Algorithm

Wederzijdse wet met behulp van de uitgebreide Euclidische formule
Retourneert $(gcd(a,b), x, y)$ zodanig dat $ax + by = gcd(x, y)$
(a,b) algemene oplossing van $(a + d\frac{y}{c}, b - d\frac{x}{c})$

```
public static long[] euclid(long a, long b) {
    long x = 1, y = 0, x1 = 0, y1 = 1, t;
    while (b != 0) {
        long q = a / b;
        t = x; x = x1; x1 = t - q * x1;
        t = y; y = y1; y1 = t - q * y1;
        t = b; b = a - q * b; a = t;
    }
    return a > 0 ? new long[]{a, x, y} : new long[]{-a, -x, -y};
}
```

Euler Phi

```
// phi(n) is the amount of numbers less than n rel. prime to n
static long phi(long n) {
    double res = n;
    HashSet<Long> factors = primeFactors(n); // Use Set instead of List!
    for (long f : factors) res *= (1.-1./f);
    return Math.round(res);
}
```

Math

```
int ceildiv(int a, int b) {
    return (a + b - 1) / b;
}
```

Euler Totient Function (aantal coprimes $\leq n$)

```
public int totient (int n) {
    int ans = n;
    for (int i = 2 ; i * i <= n ; i++) {
        if (n % i == 0) ans -= ans / i;
        while (n % i == 0) n /= i;
    }
    if (n > 1) ans -= ans / n;
    return ans;
}
```

Discrete logaritme $a^x \equiv b \pmod m$, retourneert de kleinste x die hieraan voldoet anders -1 (maakt gebruik van egcd).

```
public 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;
}
```

Rekent $(a^b) \mod c$ uit:

```
int modpow(int a, int b, int c){
  long x=1,y=a; // long is taken to avoid overflow of intermediate results
  while(b > 0){
    if(b%2 == 1){
      x=(x*y)%c;
    }
    y = (y*y)%c; // squaring the base
    b /= 2;
  }
  return x%c;
}
```

Rekent $(a \cdot b) \mod c$ uit:

```
long mulmod(long a, long b, long c){
    long x = 0, y = a % c;
    while (b > 0) {
        if(b % 2 == 1){
            x = (x + y) % c;
        }
        y = (y * 2) % c;
        b /= 2;
    }
    return x % c;
}
```

Aantal mogelijke manieren om een nummer te splitsen in positieve getallen. Bijvoorbeeld: $f(4) = \{4, 3+1, 2+2, 2+1+1, 1+1+1+1\}$.

```
int partition(int n) {
    int[] dp = new int[n + 1];
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1, r = 1; i - (3*j * j - j) / 2 >= 0; j++, r *= -1) {
            dp[i] += dp[i - (3 * j * j - j) / 2] * r;
            if (i - (3 * j * j + j) / 2 >= 0) {
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
            }
        }
    }
    return dp[n];
}
```

More GCD stuff

```
// java.math.BigInteger contains gcd functionality
// solve a*x + b*y = c for integers, returns { x, y, gcd(a,b) }
public static long[] diophantine(long a, long b, long c) {
    long[] eucl = euclid(a, b);
    long d = eucl[0];     long x = eucl[1];     long y = eucl[2];
    return (c % d == 0) ? new long[]{c / d * x, c / d * y, d} : null;
}

// solve a*x = b mod m <=> a*x + m*y = b
// returns { x, d }, all solutions given by
// x + i * m / d, i = 0, ..., d - 1
public static long[] linearCongruence(long a, long b, long m) {
    long[] diop = diophantine(a, m, b);
    return (diop == null) ? null : new long[]{diop[0], diop[2]};
}

public static long[] linearCongruenceAll(long a, long b, long m) {
    long[] diop = diophantine(a, m, b);
    if (diop == null) return null;
    long[] x = new long[(int) diop[2]];
    for (int i = 0; i < diop[2]; i++)
        x[i] = diop[0] + i * (m / diop[2]);
    return x;
}

// solve x = a[i] (mod p[i]), where gcd(p[i], p[j]) == 1 (Chinese
    Remainder Theorem)
public static int simpleRestore(int[] a, int[] p) {
    int res = a[0], m = 1;
    for (int i = 1; i < a.length; i++) {
        m *= p[i - 1];
        while (res % p[i] != a[i]) res += m;
    }
    return res;
}
```

Switch Base of Numbers

```
// for radix check, Character.MAX_RADIX
Integer.parseInt(String s, int radix);
Integer.toString(int i, int radix);
```

Flow Problems - Edge Class

```
static void addEdge(List<Edge>[] g, int s, int t, int cap, int cost) {
    g[s].add(new Edge(t, cap, cost, g[t].size()));
    g[t].add(new Edge(s, 0, -cost, g[s].size() - 1));
}
static class Edge {
    int to, f, cap, cost, rev;
    Edge(int to, int cap, int cost, int rev) {
        this.to = to; this.cap = cap; this.cost = cost; this.rev = rev;
    }
}
```

Max Flow - Ford Fulkerson $\mathcal{O}(|V^2 \log(f)|)$ (Bipartite Matching)

Konig's theorem: In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.

```
static int augment(boolean[] done, int from, int to, int add, List<Edge>[]
     g, int maxup) {
    if (done[from]) return -1;
    done[from] = true;
    if (from == to) return maxup;
    int maxdown;
    for (Edge e : g[from])
        if (e.cap - e.f >= add && (maxdown = augment(done, e.to, to, add,
    g, Math.min(maxup, e.cap - e.f))) > -1) {
            e.f += maxdown;
            g[e.to].get(e.rev).f -= maxdown;
            return maxdown;
        }
    return -1;
}
static int maxFlow(int from, int to, List<Edge>[] g, int ub) {
    int M = 1;
    while (2 * M <= ub) M *= 2;
    int n = g.length, flow = 0, aug;
    for (int add = M; add >= 1; add /= 2) {
        boolean[] done = new boolean[n];
        while ((aug = augment(done, from, to, add, g, Integer.MAX_VALUE))
    > -1) {
            flow += aug;
            done = new boolean[n];
        }
    }
    return flow;
}
```

Minimum Cost (Max) Flow (Weighted Bipartite Matching)

```
static void bellmanFord(List<Edge>[] graph, int s, int[] dist, int[]
    prevnode, int[] prevedge, int[] curflow) {
    Arrays.fill(dist, Integer.MAX_VALUE);
    dist[s] = 0; curflow[s] = Integer.MAX_VALUE;
    boolean[] inqueue = new boolean[graph.length];
    Queue<Integer> q = new ArrayDeque<>();
    q.add(s);
    while (!q.isEmpty()) {
        int u = q.poll(); inqueue[u] = false;
        for (int i = 0; i < graph[u].size(); i++) {
            Edge e = graph[u].get(i);
            if (e.f >= e.cap) continue;
            int v = e.to, ndist = dist[u] + e.cost;
            if (dist[v] > ndist) {
                dist[v] = ndist;
                prevnode[v] = u;
                prevedge[v] = i;
                curflow[v] = Math.min(curflow[u], e.cap - e.f);
                if (!inqueue[v]) {
                    inqueue[v] = true;
                    q.add(v);
                }
            }
        }
    }
}
static int[] minCostFlow(List<Edge>[] graph, int s, int t, int maxf) {
    int n = graph.length;
    int[] dist = new int[n], curflow = new int[n];
    int[] prevedge = new int[n], prevnode = new int[n];
    int flow = 0, flowCost = 0;
    while (flow < maxf) {
        bellmanFord(graph, s, dist, prevnode, prevedge, curflow);
        if (dist[t] == Integer.MAX_VALUE) break;
        int df = Math.min(curflow[t], maxf - flow);
        flow += df;
        for (int v = t; v != s; v = prevnode[v]) {
            Edge e = graph[prevnode[v]].get(prevedge[v]);
            e.f += df;
            graph[v].get(e.rev).f -= df;
            flowCost += df * e.cost;
        }
    }
    return new int[]{flow, flowCost};
}
```

Dijkstra

```
static Class Edge { int t; int c; }
static class Q implements Comparable<Q> {
    int node; long prio;
    public Q(int n, long p){ node = n; prio = p; }
    public int compareTo(Q o) { return Long.compare(prio, o.prio); }
}
static void dijkstra(List<Edge>[] edges, int s, long[] prio, int[] pred) {
    Arrays.fill(pred, -1);
    Arrays.fill(prio, Long.MAX_VALUE);
    prio[s] = 0;
    PriorityQueue<Q> q = new PriorityQueue<>();
    q.add(new Q(s, 0));
    while (!q.isEmpty()) {
        Q cur = q.remove();
        if (cur.prio != prio[cur.node]) continue;
        for (Edge e : edges[cur.node]) {
            long nprio = prio[cur.node] + e.c;
            if (prio[e.t] > nprio) {
                prio[e.t] = nprio;
                pred[e.t] = cur.node;
                q.add(new Q(e.t, nprio));
            }
        }
    }
}
```

Subset Sum (closest to a given sum)

```
dp = new int[arr.length+1][target+1];
contains = new int[arr.length+1][target+1];
// initialiseer onderste rij
for(int i = 0; i <= target; i++)
    dp[arr.length][i] = i;
for(int i = arr.length - 1; i >= 0; i--) {
    for(int bestsum = 0; bestsum <= target; bestsum++) {
        if(i == notIn) {
            dp[i][bestsum] = dp[i+1][bestsum];
        } else if(bestsum + arr[i] >= target) {
            dp[i][bestsum] = dp[i+1][bestsum];
        } else {
            dp[i][bestsum] = Math.max(dp[i+1][bestsum],
                dp[i+1][bestsum + arr[i]]);
        }
    }
}
```

Union Find

```
int find(int n) {
    if(parent[n] == n) return n;
    return parent[n] = find(parent[n]);
}
void merge(int x, int y) {
    int rx = find(x), ry = find(y);
    if(rx == ry) return;
    size[ry] += size[rx];
    parent[rx] = parent[ry];
}
```

Floyd-Warshall

Transitive hull/all pairs reachability: `boolean d[i][j] = d[i][j] || (d[i][k] && d[k][j])`

All pairs MinMax (reverse for MaxMin): `d[i][j] = min(d[i][j], max(d[i][k], d[k][j]))`

Safest path: `p[i][j] = max(p[i][j], p[i][k] * p[k][j])`

```
public static boolean floydWarshall(int[][] distance) {
    int[][] next = new int[distance.length][distance.length];
    for (int i = 0; i < distance.length; i++)
        for (int j = 0; j < distance.length; j++)
            next[i][j] = (distance[i][j] != Integer.MAX_VALUE) ? j : -1;
    for (int i = 0; i < distance.length; i++)
        for (int j = 0; j < distance.length; j++)
            for (int k = 0; k < distance.length; k++)
                if (distance[i][k] + distance[k][j] < distance[i][j]) {
                    distance[i][j] = distance[i][k] + distance[k][j];
                    next[i][j] = next[i][k];
                }
    for (int i = 0; i < distance.length; i++)
        if (distance[i][i] < 0) return false;
    return true;
}
```

Bellman Ford

Bellman-Ford's algorithm is usefull to detect negative cycles and able to report them.

```
int n = sc.nextInt(); // nr vertices
int m = sc.nextInt(); // nr edges
Edge[] edges = new Edge[m]; // Edge(source, destination, weight)
for (int i = 0; i < m; i++)
    edges[i] = new Edge(sc.nextInt(), sc.nextInt(), sc.nextInt());

int[] d = new int[n];
Arrays.fill(d, Integer.MAX_VALUE); d[0] = 0;
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < m; j++) {
        Edge e = edges[j];
        if (d[e.d] > d[e.s] + e.w) d[e.d] = d[e.s] + e.w;
    }
}
boolean flag = false;
for (int j = 0; j < m; j++) {
    Edge e = edges[j];
    if (d[e.d] > d[e.s] + e.w) {
        System.out.println("graph has a negative cycle.");
        flag = true;
    }
}
for (int i = 0; !flag && i < n; i++)
    System.out.printf("%d --> %d : %d\n", 0, i, d[i]);
```

Knapsack 0-1 with integer weights

```
int[] wi = new int[N + 1], pi = new int[N + 1]; // Weights and profits, 1-
    indexed
int[][] C = new int[N + 1][K + 1];
for (int i = 0; i <= N; i++) C[i][0] = 0;
for (int w = 0; w <= K; w++) C[0][w] = 0;
for (int i = 1; i <= N; i++)
    for (int w = 1; w <= K; w++) {
        if (wi[i] > w)
            C[i][w] = C[i-1][w];
        else
            C[i][w] = Math.max(C[i-1][w] , C[i-1][w-wi[i]] + pi[i]);
    }
```

Prim's Algorithm (Minimum Spanning Tree)

```
static class Edge implements Comparable<Edge> {
    int t, c;
    public Edge(int t, int c) { this.t = t; this.c = c; }
    public int compareTo(Edge o) { return c - o.c; }
}

// graph should contain each edge in both directions
// edge should implement compareTo on edge.c
public static Edge[] prim(List<Edge>[] graph, int start) {
    PriorityQueue<Edge> front = new PriorityQueue<>();
    int n = graph.length;
    int cursor = 0;
    boolean[] done = new boolean[n];
    done[start] = true;
    Edge[] tree = new Edge[n - 1];
    for (Edge e : graph[start]) {
        front.add(e);
    }
    while (!front.isEmpty() && cursor < n - 1) {
        Edge e = front.poll();
        if (done[e.t]) {
            continue;
        }
        done[e.t] = true;
        for (Edge e2 : graph[e.t]) {
            if (!done[e2.t]) {
                front.add(e2);
            }
        }
        tree[cursor++] = e;
    }
    return tree;
}
```

Convex Hull

```
static class Point implements Comparable<Point> {
    final int x, y, index;
    public Point(int x, int y, int index) {
        this.x = x; this.y = y; this.index = index;
    }
    public int compareTo(Point other) {
        return (x == other.x) ? (y - other.y) : (x - other.x);
    }
}
static long cross(Point O, Point A, Point B) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}
static Point[] convexHull(Point[] P) { // Returns hull in ccw order
    if (P.length <= 1)
        return Arrays.copyOf(P, P.length);
    int n = P.length, k = 0;
    Point[] H = new Point[2 * n];
    Arrays.sort(P);
    for (int i = 0; i < n; ++i) { // Build lower hull
        while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) <= 0)
            k--;
        H[k++] = P[i];
    }
    for (int i = n - 2, t = k + 1; i >= 0; i--) { // Build upper hull
        while (k >= t && cross(H[k - 2], H[k - 1], P[i]) <= 0)
            k--;
        H[k++] = P[i];
    }
    return Arrays.copyOf(H, k - 1);
}
```

Coin Change

```
// S[] the coins available
// m   the number of coins
// n   the total amount
public static long count(int S[], int m, int n) {
    long[][] table = new long[n + 1][m];
    for (int i = 0; i < m; i++) {
        table[0][i] = 1;
    }
    for (int i = 1; i < n + 1; i++) {
        for (int j = 0; j < m; j++) {
            long x = (i - S[j] >= 0) ? table[i - S[j]][j] : 0;
            long y = (j >= 1) ? table[i][j - 1] : 0;
            table[i][j] = x + y;
        }
    }
    return table[n][m - 1];
}
```

Levenshtein Distance (edit distance of strings)

```
public static int levenshtein_distance(String a, String b) {
    int[] costs = new int[b.length() + 1];
    for (int j = 0; j < costs.length; j++)
        costs[j] = j;
    for (int i = 1; i <= a.length(); i++) {
        costs[0] = i;
        int nw = i - 1;
        for (int j = 1; j <= b.length(); j++) {
            int x = 1 + costs[j];
            int y = 1 + costs[j - 1];
            int z = a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw + 1;
            nw = costs[j];
            costs[j] = Math.min(x, Math.min(y, z));
        }
    }
    return costs[b.length()];
}
```

Convert date

```
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;
}
```

Binary Search

```
public static int binary_search(int x, int[] array) {
    int low = 0;
    int high = array.length - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        if (x > array[mid]) { low = mid + 1; }
        else { high = mid; }
    }
    if(array[low] == x) return low;
    return -1;
}
```

Ternary Search

```
// Finds maximum of concave function, for minimum use -eval
static int ternarySearch(int lb, int ub) {
    int first, second, valFirst = eval(lb), valSecond = eval(ub);
    while (ub - lb > 2) {
        first = (ub + 2 * lb) / 3; second = (2 * ub + lb) / 3;
        valFirst = eval(first); valSecond = eval(second);
        if (valFirst >= valSecond)
            ub = second;
        if (valFirst <= valSecond)
            lb = first;
    }
    return Math.max(eval(lb + 1), Math.max(valFirst, valSecond));
}
```

Permutations

```
public static boolean nextPerm(int[] perm) {
    int n = perm.length, j = n - 2;
    while (j >= 0 && perm[j] >= perm[j + 1]) j--;
    if (j < 0) return false;
    int temp = perm[j], k = n - 1;
    while (perm[k] <= temp) k--;
    perm[j] = perm[k]; perm[k] = temp;
    int r = j + 1, s = n - 1;
    while (r < s) {
        temp = perm[r]; perm[r++] = perm[s]; perm[s--] = temp;
    }
    return true;
}
```

Longest Common Subsequence

```
public static int longest_common_subsequence(String a, String b) {
    int m = a.length();
    int n = b.length();
    int[][] dp = new int[m + 1][n + 1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) dp[i][j] = 0;
            else if (a.charAt(i - 1) == b.charAt(j - 1))
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[m][n];
}
```

Convex Diameter

The diameter of the convex polygon. To determine the distance between the farthest point of the convex polygon substrates using calipers method.

```
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;
}
```

Counting Squares

Draw a simple polygon and counts the number of squares that are within the figure. Remember that the boundary of a simple polygon does not cross itself. Drawing is done by either moving up, down, right or left.

```
private void solve(String line) {
    int lineLength = line.length();
    Map<Integer, ArrayList<Integer>> polygon = new HashMap<>();
    int currentX = 0; int currentY = 0;
    for(int i = 0; i < lineLength; i++){
        char operator = line.charAt(i);
        if(operator == 'U'){
            if(!polygon.containsKey(currentY)){
                ArrayList<Integer> temp = new ArrayList<Integer>();
                temp.add(currentX);
                polygon.put(currentY, temp);
            } else {
                ArrayList<Integer> temp = polygon.get(currentY);
                temp.add(currentX);
            }
            currentY++;
        } else if(operator == 'D'){
            currentY--;
            if(!polygon.containsKey(currentY)){
                ArrayList<Integer> temp = new ArrayList<Integer>();
                temp.add(currentX);
                polygon.put(currentY, temp);
            } else {
                ArrayList<Integer> temp = polygon.get(currentY);
                temp.add(currentX);
            }
        } else if(operator == 'R') { currentX++; }
        else if(operator == 'L') { currentX--; }
    }
    Iterator<Integer> allY = polygon.keySet().iterator();
    int nrBlocks = 0;
    while(allY.hasNext()){
        int yPos = allY.next();
        ArrayList<Integer> xValues = polygon.get(yPos);
        Collections.sort(xValues);
        for(int i=0; i<xValues.size()-1; i+=2){
            nrBlocks += xValues.get(i+1) - xValues.get(i);
        }
    }
    System.out.println(nrBlocks);
}
```

Area of Overlapping Rectangles

```
public int area(Rectangle[] rects) {
    int result = 0;
    for (int code = 1; code < (1 << rects.length); code++) {
        boolean used[] = decode(code, rects.length);
        int sign = -1;
        Rectangle intersected = new Rectangle(Integer.MIN_VALUE,
                Integer.MIN_VALUE, Integer.MAX_VALUE, Integer.MAX_VALUE);
        for (int i = 0; i < rects.length; i++) {
            if (used[i]) {
                intersected = Rectangle.intersect(intersected, rects[i]);
                if (intersected == null) { break; }
                sign = -sign;
            }
        }
        if (intersected != null) {
            result += sign * intersected.getArea();
        }
    }
    return result;
}
boolean[] decode(int code, int size) {
    boolean used[] = new boolean[size];
    for (int i = 0; i < used.length; i++) {
        used[i] = (code % 2 == 1);
        code /= 2;
    }
    return used;
}
class Rectangle {
    int minX, minY, int maxX, int maxY;
    public Rectangle(int minX, int minY, int maxX, int maxY) {
        this.minX=minX; this.minY=minY; this.maxX=maxX; this.maxY=maxY;
    }
    int getArea() { return (maxX - minX) * (maxY - minY); }
    static Rectangle intersect(Rectangle r1, Rectangle r2) {
        int iMinX = Math.max(r1.minX, r2.minX);
        int iMinY = Math.max(r1.minY, r2.minY);
        int iMaxX = Math.min(r1.maxX, r2.maxX);
        int iMaxY = Math.min(r1.maxY, r2.maxY);
        if (iMinX >= iMaxX || iMinY >= iMaxY) { return null; }
        else { return new Rectangle(iMinX, iMinY, iMaxX, iMaxY); }
    }
}
```

Knuth-Morris-Pratt $\mathcal{O}(n)$ (string search)

```
public class KMPplus {
    private String pattern;
    private int[] next;

    // create Knuth-Morris-Pratt NFA from pattern
    public KMPplus(String pattern) {
        this.pattern = pattern;
        int M = pattern.length();
        next = new int[M];
        int j = -1;
        for (int i = 0; i < M; i++) {
            if (i == 0) next[i] = -1;
            else if (pattern.charAt(i) != pattern.charAt(j)) next[i] = j;
            else next[i] = next[j];
            while (j >= 0 && pattern.charAt(i) != pattern.charAt(j)) {
                j = next[j];
            }
            j++;
        }

        for (int i = 0; i < M; i++)
            System.out.println("next[" + i + "] = " + next[i]);
    }

    // return offset of first occurrence of text in pattern
    // (or N if no match). Simulate the NFA to find match
    public int search(String text) {
        int M = pattern.length();
        int N = text.length();
        int i, j;
        for (i = 0, j = 0; i < N && j < M; i++) {
            while (j >= 0 && text.charAt(i) != pattern.charAt(j))
                j = next[j];
            j++;
        }
        if (j == M) return i - M;
        return N;
    }

    // substring search
    KMPplus kmp = new KMPplus(pattern);
    int offset = kmp.search(text);
}
```

Maximum matching general graph $\mathcal{O}(|V|^3)$

```
static int lca(int[] match, int[] base, int[] p, int a, int b) {
    boolean[] used = new boolean[match.length];
    for(;;) {
        a = base[a]; used[a] = true;
        if (match[a] == -1) break; a = p[match[a]];
    }
    for(;;) { b = base[b]; if (used[b]) return b; b = p[match[b]];}
}
static void markPath(int[] match, int[] base, boolean[] blossom,
                     int[] p, int v, int b, int children) {
    for (; base[v] != b; v = p[match[v]]) {
        blossom[base[v]] = blossom[base[match[v]]] = true;
        p[v] = children; children = match[v];
    }
}
static int findPath(List<Integer>[] graph, int[] match, int[] p,
        int root) { int n = graph.length;
    boolean[] used = new boolean[n]; Arrays.fill(p, -1);
    int[] base = new int[n]; for (int i = 0; i < n; ++i) base[i] = i;
    used[root] = true; int qh = 0; int qt = 0;
    int[] q = new int[n]; q[qt++] = root;
    while (qh < qt) {
        int v = q[qh++];
        for (int to : graph[v]) {
            if (base[v] == base[to] || match[v] == to) continue;
            if (to == root || match[to] != -1 && p[match[to]] != -1) {
                int curbase = lca(match, base, p, v, to);
                boolean[] blossom = new boolean[n];
                markPath(match, base, blossom, p, v, curbase, to);
                markPath(match, base, blossom, p, to, curbase, v);
                for (int i = 0; i < n; ++i)
                    if (blossom[base[i]]) {
                        base[i] = curbase;
                        if (!used[i]) { used[i] = true; q[qt++] = i; }
                    }
            } else if (p[to] == -1) {
                p[to] = v; if (match[to] == -1) return to;
                to = match[to]; used[to] = true; q[qt++] = to;
            }
        }
    }
    return -1;
}
static int maxMatching(List<Integer>[] graph) {
    int[] match = new int[graph.length]; Arrays.fill(match, -1);
    int[] p = new int[graph.length];
    for (int i = 0; i < graph.length; ++i) {
        if (match[i] == -1) {
            int v = findPath(graph, match, p, i);
            while (v != -1) {
                int pv = p[v]; int ppv = match[pv]; match[v] = pv;
                match[pv] = v; v = ppv;
            }
        }
    }
    int matches = 0;
    for (int i = 0; i < n; ++i) if (match[i] != -1) ++matches;
    return matches / 2;
}

static void main(String[] args) { // Usage example
    int n = 4; List<Integer>[] g = new List[n];
    for (int i = 0; i < n; i++) g[i] = new ArrayList<>();
    g[0].add(1); g[1].add(0); g[1].add(2); g[2].add(1);
    g[2].add(3); g[3].add(2); g[0].add(3); g[3].add(0);
    System.out.println(2 == maxMatching(g));
}
```

Topological Sort

```
static void dfs(List<Integer>[] graph, boolean[] used,
            List<Integer> res, int u) {
    used[u] = true;
    for (int v : graph[u])
        if (!used[v]) dfs(graph, used, res, v);
    res.add(u);
}
public static List<Integer> topologicalSort(List<Integer>[] graph) {
    int n = graph.length;
    boolean[] used = new boolean[n];
    List<Integer> res = new ArrayList<>();
    for (int i = 0; i < n; i++)
        if (!used[i]) dfs(graph, used, res, i);
    Collections.reverse(res);
    return res;
}
static void main(String[] args) {
    int n = 3; List<Integer>[] g = new List[n];
    for (int i = 0; i < n; i++) { g[i] = new ArrayList<>(); }
    g[2].add(0); g[2].add(1); g[0].add(1); // add three edges
    List<Integer> res = topologicalSort(g);
    System.out.println(res);
}
```

Regex

`s.matches("regex")`	Evaluates if "regex" matches "s". Returns only true if the WHOLE string can be matched
`s.split("regex")`	Creates array with substrings of s divided at occurrence of "regex". "regex" is not included in the result.
`s.replace("a", "b")`	Replaces regex "a" with replacement "b"
`.`	Matches any character
`^regex`	regex must match at the beginning of the line
`regex$`	regex must match at the beginning of the line
`[abc]`	Can match the letter 'a' or 'b' or 'c'
`[abc][vz]`	Can match the letter 'a' or 'b' or 'c' followed by either 'v' or 'z'
`[^abc]`	When a '^' appears as the first character inside [] it negates the pattern. This can match any character except 'a' or 'b' or 'c'
`[a-d1-7]`	Ranges, letter between a and d and figures from 1 to 7, will not match d1
`X|Z`	Finds X or Z
`XZ`	Finds X directly followed by Z
`$`	checks if a line end follows
`\d`	Any digit, short for [0-9]
`\D`	A non-digit, short for [^0-9]
`\s`	A whitespace character, short for [\t\n\x0b \r \f]
`\S`	A non-whitespace character, short for [^\s]
`\w`	A word character, short for [a-zA-Z_0-9]
`\W`	A non-word character [^\w]
`\S+`	Several non-whitespace characters
`\b`	Matches a word boundary. A word character is [a-zA-Z_0-9] and \b matches its bounderies.
`*`	Occurs zero or more times, is short for {0,}
`+`	Occurs one or more times, is short for {1,}
`?`	Occurs no or one times, ? is short for {0,1}
`{X}`	Occurs X number of times, {} describes the order of the preceding liberal
`{X,Y}`	Occurs between X and Y times
`*?`	? after a quantifier makes it a reluctant quantifier, it tries to find the smallest match.

Negative Lookahead provide the possibility to exclude a pattern. With this you can say that a string should not be followed by another string. Negative Lookaheads are defined via `(?!pattern)`. For example the following will match a if a is not followed by b.

```
a(?!b)
```

Tree Isomorphism

Checkt of twee bomen isomorphic zijn (zelfde door wat kinderen te swappen).

```
class TreeIsomorphism { //ProblemE_ekp2003

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int cases = in.nextInt();
        while (cases-- > 0) {
            char[] s = in.next().toCharArray();
            char[] t = in.next().toCharArray();
            Node[] t1 = new Node[s.length/2 + 1];
            Node[] t2 = new Node[t.length/2 + 1];
            makeTree(t1, s);
            makeTree(t2, t);
            if (isIsomorph(t1, t2)) System.out.println("same");
            else System.out.println("different");
        }
    }

    static boolean isIsomorph(Node[] s, Node[] t) {
        setRoot(s[0], s.length);
        for (int i = 0; i < t.length; i++)
            if (isomorph(s[0], setRoot(t[i], t.length))) return true;
        return false;
    }

    static boolean isomorph(Node n, Node m) {
        if (n.successors != m.successors || n.children.length != m.
    children.length)
            return false;
        if (n.successors == 0)
            return true;
        boolean found = true;
        boolean[] used = new boolean[m.children.length];
        for (int i = 0; i < n.children.length && found; i++) {
            if (n.children[i] == n.parent) continue;
            found = false;
            for (int j = 0; j < m.children.length && !found; j++) {
                if (used[j] || m.children[j] == m.parent) continue;
                used[j] = found = isomorph(n.children[i], m.children[j]);
            }
        }
        return found;
    }

    static void makeTree(Node[] tree, char[] s) {
```

```
        int size = 0;
        Node root = tree[size++] = new Node(null);
        for (int i = 0; i < s.length; i++)
            if (s[i] == '0') {
                tree[size] = new Node(root);
                root.childrenArray.add(tree[size]);
                root = tree[size++];
            } else {
                int successors = root.successors + 1;
                root.makeChildren();
                root = root.parent;
                root.successors += successors;
            }
        tree[0].makeChildren();
    }
    static Node setRoot(Node node, int n) {
        Node newParent = null;
        // walk up to the old root to set the new parent
        while (node != null) {
            Node next = node.parent;    node.parent = newParent;
            newParent = node;           node = next;
        }
        // walk back to the new root to set the number of successors
        node = newParent;
        while (node.parent != null) {
            node.successors = n - node.parent.successors - 2;
            node = node.parent;
        }
        node.successors = n - 1;
        return node;
    }
    static class Node {
        public ArrayList<Node> childrenArray = new ArrayList<Node>();
        public Node[] children; public Node parent;
        public int successors = 0;
        public Node(Node parent) {
            this.parent = parent;
            if (parent != null) childrenArray.add(parent);
        }
        public void makeChildren() {
            children = new Node[childrenArray.size()];
            childrenArray.toArray(children);
            childrenArray = null;
        }
    }
}
```

Hopcroft-Karp

HopcroftKarp algorithm is an algorithm that takes as input a bipartite graph and produces as output a maximum cardinality matching - a set of as many edges as possible with the property that no two edges share an endpoint. - 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;
}
```

Roman Numerals

```
// converts in range 1- 3999
string arabicToRoman(int num) {
    string uni[10] = {"","I","II","III","IV","V","VI","VII","VIII","IX"};
    string deci[10] = {"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};
    string cen[10] ={"","C","CC", "CCC", "CD","D","DC","DCC","DCCC","CM"};
    string mil[4] = {"", "M", "MM", "MMM"};
    int nUni = num % 10;
    int nDec = (num / 10) % 10;
    int nCen = ((num / 10) / 10) % 10;
    int nMil = (((num / 10) / 10) / 10) % 10;
    string ans = mil[nMil];
    ans += cen[nCen];
    ans += deci[nDec];
    ans += uni[nUni];
    return ans;
}
```

```
// converts in range 1 - 3999
int romanToArabic(string num) {
    map <char, int> RtoA;
    RtoA['I'] = 1;  RtoA['V'] = 5;  RtoA['X'] = 10; RtoA['L'] = 50;
    RtoA['C'] = 100;RtoA['D'] = 500;RtoA['M'] = 1000;

    int value = 0;
    for (int i = 0; num[i]; i++) {
        if (num[i+1] && RtoA[num[i]] < RtoA[num[i+1]]) {
            value += RtoA[num[i+1]] - RtoA[num[i]];
            i++;
        } else value += RtoA[num[i]];
    }
    return value;
}
```

Josephus Problem

n prisoners are standing on a circle, sequentially numbered from 0 to $(n-1)$. An executioner walks along the circle, starting from prisoner 0, removing every k^{th} prisoner and killing him. Given any $n, k > 0$, find out which prisoner will be the final survivor.

```
int josephus(int n, int k) {
    if (n == 1) return 1;
    else return (josephus(n - 1, k) + k - 1) % n + 1;
}
```

Repeating Float to Fraction

```
// input:   0.1(6)
// output:  1/6
public static void solve(Scanner sc) {
    char[] in = sc.next().toCharArray();
    String fixed = "";
    String seq = "";
    boolean isSeq = false;
    for (int i = 2; i < in.length; i++) {
        if (in[i] == '(') isSeq = true;
        else if (in[i] == ')') isSeq = false;
        else if (isSeq) seq += in[i];
        else fixed += in[i];
    }
    long a = 0, b = 0;
    if (fixed.length() > 0) a += Long.parseLong(fixed);
    if (seq.length() > 0) b += Long.parseLong(seq);
    String divA = "1";
    String divB = "";
    for (int i = 0; i < seq.length(); i++) {
        divB += "9";
    }
    for (int i = 0; i < fixed.length(); i++) {
        divA += "0";
        divB += "0";
    }
    long c = 0, d = 0;
    if (divA.length() > 0) c = Long.parseLong(divA);
    if (divB.length() > 0) d = Long.parseLong(divB);
    BigInteger bigA;
    BigInteger bigB;
    if (b > 0) {
        bigA = BigInteger.valueOf(a).multiply(BigInteger.valueOf(d)).
               add(BigInteger.valueOf(c).multiply(BigInteger.valueOf(b)));
        bigB = BigInteger.valueOf(c).multiply(BigInteger.valueOf(d));
    } else {
        bigB = BigInteger.valueOf(c);
        bigA = BigInteger.valueOf(a);
    }
    BigInteger biggcd = bigA.gcd(bigB);
    System.out.println(bigA.divide(biggcd) + "/" + bigB.divide(biggcd));
}
```

Closest-pair Problem

```
public class ClosestPair {
    public static class Point {
        public final double x, y;
        public Point(double x, double y) { this.x = x; this.y = y; }
        public String toString() {  return "(" + x + ", " + y + ")";  }
    }
    public static class Pair {
        public Point point1 = null, point2 = null;
        public double distance = 0.0;
        public Pair() {  }
        public Pair(Point point1, Point point2) {
            this.point1 = point1; this.point2 = point2;
            this.distance = distance(point1, point2);
        }
        public void update(Point point1, Point point2, double distance) {
            this.point1 = point1; this.point2 = point2;
            this.distance = distance;
        }
    }
    public static double distance(Point p1, Point p2) {
        return Math.hypot(p2.x - p1.x, p2.y - p1.y);
    }
    public static void sortByX(List<? extends Point> points) {
        public int compare(Point point1, Point point2) {
            if (point1.x < point2.x) return -1;
            if (point1.x > point2.x) return 1; return 0;
        }
    }
    public static void sortByY(List<? extends Point> points) {
        public int compare(Point point1, Point point2) {
            if (point1.y < point2.y) return -1;
            if (point1.y > point2.y) return 1; return 0;
        }
    }
    public static Pair bruteForce(List<? extends Point> points) {
        int numPoints = points.size();
        if (numPoints < 2) return null;
        Pair pair = new Pair(points.get(0), points.get(1));
        if (numPoints > 2) {
            for (int i = 0; i < numPoints - 1; i++) {
                Point point1 = points.get(i);
                for (int j = i + 1; j < numPoints; j++) {
                    Point point2 = points.get(j);
                    double distance = distance(point1, point2);
                    if (distance < pair.distance)
                        pair.update(point1, point2, distance);
                }
            }
        }
        return pair;
    }
    public static Pair divideAndConquer(List<Point> points) {
        List<Point> pSortX = new ArrayList<>(points); sortByX(pSortX);
        List<Point> pSortY = new ArrayList<>(points); sortByY(pSortY);
        return divideAndConquer(pSortX, pSortY);
    }
    static Pair divideAndConquer(List<Point> pSortX, List<Point> pSortY) {
        int numPoints = pSortX.size();
        if (numPoints <= 3) return bruteForce(pSortX);
        int divIndex = numPoints >>> 1;
        List<Point> leftOfCenter = pSortX.subList(0, divIndex);
        List<Point> rightOfCenter = pSortX.subList(divIndex, numPoints);
        List<Point> tempList = new ArrayList<>(leftOfCenter);
        sortByY(tempList);
        Pair closestPair = divideAndConquer(leftOfCenter, tempList);
        tempList.clear();
        tempList.addAll(rightOfCenter);
        sortByY(tempList);
        Pair closestPairRight = divideAndConquer(rightOfCenter, tempList);
        if (closestPairRight.distance < closestPair.distance)
            closestPair = closestPairRight;
        tempList.clear();
        double shortestDistance = closestPair.distance;
        double centerX = rightOfCenter.get(0).x;
        for (Point point : pSortY)
            if (Math.abs(centerX - point.x) < shortestDistance)
                tempList.add(point);
        for (int i = 0; i < tempList.size() - 1; i++) {
            Point point1 = tempList.get(i);
            for (int j = i + 1; j < tempList.size(); j++) {
                Point point2 = tempList.get(j);
                if ((point2.y - point1.y) >= shortestDistance) break;
                double distance = distance(point1, point2);
                if (distance < closestPair.distance) {
                    closestPair.update(point1, point2, distance);
                    shortestDistance = distance;
                }
            }
        }
        return closestPair;
    }
}
```

Strongly Connected Components

```
public class Tarjan {
    static List<Integer>[] graph;
    static boolean[] visited;
    static Stack<Integer> stack;
    static int time;
    static int[] lowlink;
    static List<List<Integer>> components;
    static List<List<Integer>> scc(List<Integer>[] g) {
        graph = g;
        int n = graph.length;
        visited = new boolean[n];
        stack = new Stack<>();
        time = 0;
        lowlink = new int[n];
        components = new ArrayList<>();
        for (int u = 0; u < n; u++)
            if (!visited[u]) dfs(u);
        return components;
    }
    static void dfs(int u) {
        lowlink[u] = time++;
        visited[u] = true;
        stack.add(u);
        boolean isComponentRoot = true;
        for (int v : graph[u]) {
            if (!visited[v]) dfs(v);
            if (lowlink[u] > lowlink[v]) {
                lowlink[u] = lowlink[v];
                isComponentRoot = false;
            }
        }
        if (isComponentRoot) {
            List<Integer> component = new ArrayList<>();
            while (true) {
                int x = stack.pop();
                component.add(x);
                lowlink[x] = Integer.MAX_VALUE;
                if (x == u) break;
            }
            components.add(component);
        }
    }
}
```

2-SAT

```
static void dfs1(List<Integer>[] graph, boolean[] used, List<Integer>
    order, int u) {
    used[u] = true;
    for (int v : graph[u])
        if (!used[v]) dfs1(graph, used, order, v);
    order.add(u);
}
static void dfs2(List<Integer>[] reverseGraph, int[] comp, int u, int
    color) {
    comp[u] = color;
    for (int v : reverseGraph[u])
        if (comp[v] == -1) dfs2(reverseGraph, comp, v, color);
}
public static boolean[] solve2Sat(List<Integer>[] graph) {
    int n = graph.length;
    boolean[] used = new boolean[n];
    List<Integer> order = new ArrayList<>();
    for (int i = 0; i < n; ++i)
        if (!used[i])
            dfs1(graph, used, order, i);
    List<Integer>[] reverseGraph = new List[n];
    for (int i = 0; i < n; i++)
        reverseGraph[i] = new ArrayList<>();
    for (int i = 0; i < n; i++)
        for (int j : graph[i]) reverseGraph[j].add(i);
    int[] comp = new int[n]; Arrays.fill(comp, -1);
    for (int i = 0, color = 0; i < n; ++i) {
        int u = order.get(n - i - 1);
        if (comp[u] == -1) dfs2(reverseGraph, comp, u, color++);
    }
    for (int i = 0; i < n; ++i)
        if (comp[i] == comp[i ^ 1]) return null;
    boolean[] res = new boolean[n / 2];
    for (int i = 0; i < n; i += 2)
        res[i / 2] = comp[i] > comp[i ^ 1];
    return res;
}
public static void main(String[] args) {
    List<Integer>[] g = new List[6];
    for (int i = 0; i < g.length; i++) g[i] = new ArrayList<>();
    // (a || b) && (b || !c) so !a => b, !b => a, !b => !c, c => b
    int a = 0, na = 1, b = 2, nb = 3, c = 4, nc = 5; // Node indices
    g[na].add(b); g[nb].add(a); g[nb].add(nc); g[c].add(b);
    System.out.println(Arrays.toString(solve2Sat(g)));
}
```

Longest Increasing Subsequence $\mathcal{O}(n^2)$

```
public class LongestIncreasingSubsequence {
    public Integer[] LIS(Integer[] A) {
        int[] m = new int[A.length];
        //Arrays.fill(m, 1); //not important here
        for (int x = A.length - 2; x >= 0; x--) {
            for (int y = A.length - 1; y > x; y--) {
                if (A[x] < A[y] && m[x] <= m[y]) { m[x]++; }
            }
        }
        int max = m[0];
        for (int i = 1; i < m.length; i++) {
            if (max < m[i]) { max = m[i]; }
        }
        List<Integer> result = new ArrayList<Integer>();
        for (int i = 0; i < m.length; i++) {
            if (m[i] == max) { result.add(A[i]); max--; }
        }
        return result.toArray(new Integer[0]);
    }
}
```

Longest Increasing Subsequence $\mathcal{O}(n \log n)$

```
static int CeilIndex(int A[], int l, int r, int key) {
    while (r - l > 1) {
        int m = l + (r - l)/2;
        if (A[m]>=key) r = m;
        else l = m;
    }
    return r;
}
static int LongestIncreasingSubsequenceLength(int A[], int size) {
    int[] tailTable   = new int[size];
    int len; // always points empty slot
    tailTable[0] = A[0];
    len = 1;
    for (int i = 1; i < size; i++) {
        if (A[i] < tailTable[0]) tailTable[0] = A[i];
        else if (A[i] > tailTable[len-1]) tailTable[len++] = A[i];
        else tailTable[CeilIndex(tailTable, -1, len-1, A[i])] = A[i];
    }
    return len;
}
```

Binary Search $f(n)$

Problem description: solve $p \cdot e^{-x} + q \cdot \sin(x) + r \cdot \cos(x) + s \cdot \tan(x) + t \cdot x^2 + u = 0$. For $0 \leq x \leq 1$.

```
void solve() {
    if (f(0) * f(1) > 0) { System.out.printf("No solution\n");
    } else { System.out.printf("%.4f\n", binarySearch()); }
}
static double binarySearch() {
    double low = 0.0, high = 1.0, mid = 0.5;
    while (Math.abs(high - low) > 0.00000001) {
        mid = (low + high) / 2;
        if (f(low) * f(mid) <= 0) { high = mid; }
        else { low = mid; }
    } return mid;
}
static double f(double x) {
    return p * Math.pow(Math.E, -x) + q * Math.sin(x) + r * Math.cos(x) +
           s * Math.tan(x) + t * x * x + u;
}
```

Fast Fourier Transformation

Lengte moet macht van 2 zijn. Als sign -1 is krijg je de inverse transformatie, bij 1 de normale.

```
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;
        }
    }
}
```

Gaussian elimination

Gaussian elimination (also known as row reduction) is an algorithm for solving systems of linear equations.

```
public static double[] gauss(double[][] a, double[] b) {
    int n = a.length;
    for (int row = 0; row < n; row++) {
        int best = row;
        for (int i = row + 1; i < n; i++)
            if (Math.abs(a[best][row]) < Math.abs(a[i][row]))
                best = i;
        double[] tt = a[row]; a[row] = a[best]; a[best] = tt; // swap
        double t = b[row]; b[row] = b[best]; b[best] = t; // swap
        for (int i = row + 1; i < n; i++)
            a[row][i] /= a[row][row];
        b[row] /= a[row][row];
        for (int i = 0; i < n; i++) {
            double x = a[i][row];
            if (i != row && x != 0) {
                for (int j = row + 1; j < n; j++)
                    a[i][j] -= a[row][j] * x;
                b[i] -= b[row] * x;
            }
        }
    }
    return b;
}

// example use:
public static void main(String[] args) {
    double[][] a = {{4, 2, -1}, {2, 4, 3}, {-1, 3, 5}};
    double[] b = {1, 0, 0};
    double[][] a1 = a.clone();
    for (int i = 0; i < a.length; i++) a1[i] = a[i].clone();
    double[] b1 = b.clone();
    double[] x = gauss(a, b);
    for (int i = 0; i < a.length; i++) {
        double y = 0;
        for (int j = 0; j < a[i].length; j++)
            y += a1[i][j] * x[j];
        if (Math.abs(b1[i] - y) > 1e-9) {
            System.err.println("error");
            return;
        }
    }
}
```

Maximum independent set / max clique

```
public class Clique {
    //max independent set is max clique in the complement, Gave
    stackoverflow at 70 nodes
    public static int BronKerbosch(BitSet[] g, BitSet cur, BitSet allowed,
     BitSet forbidden, int[]
            weights) {
        if (allowed.isEmpty() && forbidden.isEmpty()) {
            int res = 0;
            for (int u = cur.nextSetBit(0); u >= 0; u = cur.nextSetBit(u +
     1))
                res += weights[u];
            return res;
        }
        if (allowed.isEmpty()) return -1;
        int res = -1;
        BitSet temp = (BitSet) allowed.clone();
        temp.or(forbidden);
        int pivot = temp.nextSetBit(0);
        BitSet z = (BitSet) allowed.clone();
        z.andNot(g[pivot]);
        for (int u = z.nextSetBit(0); u >= 0; u = z.nextSetBit(u + 1)) {
            BitSet newCur = (BitSet) cur.clone();
            BitSet newAllowed = (BitSet) allowed.clone();
            BitSet newForbidden = (BitSet) forbidden.clone();
            newCur.set(u);
            newAllowed.and(g[u]);
            newForbidden.and(g[u]);
            res = Math.max(res, BronKerbosch(g, newCur, newAllowed,
    newForbidden, weights));
            allowed.flip(u);
            forbidden.set(u);
        }
        return res;
    }

    public static void main(String[] args) {
        Random rnd = new Random(1);
        int n = 65;
        long[] g = new long[n];
        int[] weights = new int[n];
        for (int i = 0; i < n; i++)
            weights[i] = rnd.nextInt(1000);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < i; j++)
                if (rnd.nextBoolean()) {
                    g[i] |= 1L << j;
```

```
                    g[j] |= 1L << i;
                }
            BitSet[] bsGraph = new BitSet[n];
            for (int i = 0; i < n; i++) {
                long[] l = {g[i]};
                bsGraph[i] = BitSet.valueOf(l);
            }
            BitSet full = new BitSet(n);
            full.set(0, n);
            long x = System.nanoTime();
            int res = BronKerbosch(bsGraph, new BitSet(n), full, new BitSet(n)
        , weights);
            long y = System.nanoTime();
            System.out.println(y - x);
            System.out.println(res);
        }
}
```

Fenwick tree 2D (sum of subsets)

```
public static void add(int[][] t, int r, int c, int value) {
    for (int i = r; i < t.length; i |= i + 1)
        for (int j = c; j < t[0].length; j |= j + 1)
            t[i][j] += value;
}
// sum[(0, 0), (r, c)]
public static int sum(int[][] t, int r, int c) {
    int res = 0;
    for (int i = r; i >= 0; i = (i & (i + 1)) - 1)
        for (int j = c; j >= 0; j = (j & (j + 1)) - 1)
            res += t[i][j];
    return res;
}
// sum[(r1, c1), (r2, c2)]
public static int sum(int[][] t, int r1, int c1, int r2, int c2) {
    return sum(t, r2, c2) - sum(t, r1 - 1, c2) - sum(t, r2, c1 - 1) + sum(
    t, r1 - 1, c1 - 1);
}

public static int get(int[][] t, int r, int c) {
    return sum(t, r, c, r, c);
}

public static void set(int[][] t, int r, int c, int value) {
    add(t, r, c, -get(t, r, c) + value);
}
```

Flood-fill BFS

```
// queue already contains the start location(s)
static int[][] floodFill(boolean[][] walls, Queue<Location> queue) {
    int[] dr = {-1, 0, 1, 0}, dc = {0, -1, 0, 1};
    // int[] dr = {-1, -1, -1,  0,  0, +1, +1, +1};
    // int[] dc = {+1,  0, -1, +1, -1, +1,  0, -1};
    int h = walls.length, w = walls[0].length;
    int[][] t = new int[h][w];
    for (int r = 0; r < h; r++) Arrays.fill(t[r], -1);
    for (Location l : queue) t[l.r][l.c] = 0;
    while( !queue.isEmpty()) {
        Location l = queue.poll();
        for( int d = 0; d < dr.length; d++) {
            if (l.r + dr[d] >= 0 && l.r + dr[d] < h &&
                l.c + dc[d] >= 0 && l.c + dc[d] < w &&
                   !walls[l.r + dr[d]][ l.c + dc[d]] &&
                   t[l.r + dr[d]][l.c + dc[d]] < 0) {
                t[l.r + dr[d]][l.c + dc[d]] =  t[l.r][l.c] + 1;
                queue.add(new Location(l.r + dr[d], l.c + dc[d]));
            }
        }
    }
    return t;
}
static class Location {
    public int r, c;
    Location(int r, int c) {
        this.r = r; this.c = c;
    }
}
```

Inverse

```
function inverse(a, p)
    t := 0;      newt := 1;
    r := p;      newr := a;
    while newr     0
        quotient := r div newr
        (r, newr) := (newr, r - quotient * newr)
        (t, newt) := (newt, t - quotient * newt)
    if degree(r) > 0 then
        return "Either p is not irreducible or a is a multiple of p"
    return (1/r) * t
```

Evaluate expressions

```
// sample in: "( ( 10 + 5 ) * 3 )"
Scanner sc = new Scanner(System.in);
Stack<String> ops = new Stack<>();
Stack<Double> vals = new Stack<>();

while (sc.hasNext()) {
    String s = sc.next();
    if (s.equals("(")) {}
    else if (s.equals("+")) ops.push(s);
    else if (s.equals("-")) ops.push(s);
    else if (s.equals("*")) ops.push(s);
    else if (s.equals("/")) ops.push(s);
    else if (s.equals("sqrt")) ops.push(s);
    else if (s.equals(")")) {
        String op = ops.pop();
        double v = vals.pop();
        if (op.equals("+")) v = vals.pop() + v;
        else if (op.equals("-")) v = vals.pop() - v;
        else if (op.equals("*")) v = vals.pop() * v;
        else if (op.equals("/")) v = vals.pop() / v;
        else if (op.equals("sqrt")) v = Math.sqrt(v);
        vals.push(v);
    } else vals.push(Double.parseDouble(s));
}
System.out.println(vals.pop());
```

Convert Infix to Postfix

```
public class InfixToPostfix {
    public static void main(String[] args) {
        Stack<String> stack = new Stack<String>();
        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            if      (s.equals("+")) stack.push(s);
            else if (s.equals("*")) stack.push(s);
            else if (s.equals(")")) System.out.print(stack.pop() + " ");
            else if (s.equals("(")) System.out.print("");
            else                    System.out.print(s + " ");
        }
        System.out.println();
    }
}
```

Fast input

```
import java.io.*; import java.util.*;
public class Template {
    static BufferedReader in; static String nextLine;
    public static void main(String[] args) throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));
        while (hasNextLine()) instance();
    }
    public static void instance() throws IOException { }
    public static boolean hasNextLine() throws IOException {
        if (nextLine == null) nextLine = in.readLine();
        return nextLine != null;
    }
    public static String readLine() throws IOException {
        String s = (nextLine != null) ? nextLine : in.readLine();
        nextLine = null;
        return s;
    }
    public static int readInt() throws IOException {
        return Integer.parseInt(readLine());
    }
    static int[] readInts() throws IOException {
        return readInts(0);
    }
    public static int[] readInts(int pad) throws IOException {
        String[] ss = readLine().split(" ");
        int[] s = new int[ss.length + pad];
        for (int i = 0; i < ss.length; i++)
            s[i + pad] = Integer.parseInt(ss[i]);
        return s;
    }
}
```

JAVA Stuff

```
static void Arrays.sort(T[] a, Comparator<? super T> c);
static String Arrays.deepToString(Object[] a);
static int binarySearch(int[] a, int key)
Polygon p; Area a; Line2D l;// java.awt.geom
TreeMap<Long, Integer> map = new TreeMap<Long, Integer>();
for(Map.Entry e : map.entrySet()){
    key = (long)e.getKey();
    val = (int)e.getValue();
}
```