Team Contest Reference

Team Integer

TU Delft

Formulas for Geometric Shapes

$$\begin{aligned}
\text{oppervlakte cirkel} &: \pi r^2 \\
\text{omtrek cirkel} &: \pi d \\
\text{oppervlakte ellips} &: \pi ab \\
\text{oppervlakte kegel} &: \pi r^2 + \pi r\sqrt{r^2 h^2} \\
\text{inhoud kegel} &: \frac{1}{3}\pi r^2 h \\
\text{oppervlakte bol} &: 4\pi r^2 \\
\text{inhoud bol} &: \frac{4}{3}\pi r^3 \\
\text{oppervlakte cillinder} &: 2\pi rh + 2\pi r^2 \\
\text{inhoud cilinder} &: \pi r^2 h
\end{aligned}$$

More Formulas

$$\text{least common multiple} : \text{lcm}(m,n) = \frac{|m \cdot n|}{\text{gcd(m, n)}}$$

$$\text{Catalan number} : C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n} \frac{n+k}{k}$$

$$\text{Catalan numbers} : C = \{1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796\}$$

$$\text{Triangle numbers} : T = \{1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136\}$$

$$\text{Triangle numbers} : T_n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \binom{n+1}{2}$$

Fibonacci Numbers

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

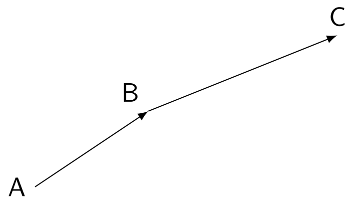
- When we take a pairs of large consecutive Fibonacci numbers, we can approximate the golden ratio by dividing them.
- The sum of any ten consecutive Fibonacci numbers is divisible by 11.
- Two consecutive Fibonacci numbers are co-prime.
- The Fibonacci numbers in the composite-number (i.e. non-prime)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

Computational Geometry

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



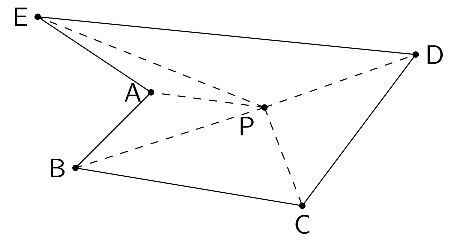
$$\overrightarrow{AB} = \begin{bmatrix} p \\ q \end{bmatrix} \qquad \overrightarrow{n} = \begin{bmatrix} q \\ -p \end{bmatrix}$$

$$\overrightarrow{n} \cdot \overrightarrow{BC} < 0 \Rightarrow \text{linksaf}$$
$$\overrightarrow{n} \cdot \overrightarrow{BC} > 0 \Rightarrow \text{rechtsaf}$$

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.

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

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

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

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:

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

Primes

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

```
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;
        }
    }
}
```

Greatest Common Divisor

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

```
def extended_gcd(a, b):
    s := 0;    old_s := 1
    t := 1;    old_t := 0
    r := b;    old_r := a
    while r != 0 :
        quotient := old_r div r
        (old_r, r) := (r, old_r - quotient * r)
        (old_s, s) := (s, old_s - quotient * s)
        (old_t, t) := (t, old_t - quotient * t)
    print "B zout coefficients:", (old_s, old_t)
    print "greatest common divisor:", old_r
    print "quotients by the gcd:", (t, s)
```

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];
}
```

Max Flow

```
public int maxFlow() {
    int output = 0;
    while (true) {
        int pathCap = findPath();
        if (pathCap == 0) { break; }
        output += pathCap;
    }
    return output;
}
private int findPath() {
    Queue<Integer> q = new LinkedList<Integer>();
    boolean[] visited = new boolean[nrNodes + 2];
    int[] from = new int[nrNodes + 2];
    Arrays.fill(from, -1);
    q.add(0); visited[0] = true;
    while (!q.isEmpty()) {
        int cur = q.poll();
        List<Edge> adjc = graph.get(cur);
        if (cur == nrNodes + 1)
            break;
        for (Edge e : adjc) {
            if (!visited[e.b] && e.getCapacity() > 0) {
                q.add(e.b);
                visited[e.b] = true;
                from[e.b] = cur;
            }
        }
    }
    if (from[nrNodes + 1] == -1) { return 0; }
    int pathCap = Integer.MAX_VALUE;
    int cur = nrNodes + 1;
    while (from[cur] > -1) {
        int prev = from[cur];
        pathCap = Math.min(pathCap, getEdge(prev, cur).getCapacity());
        cur = prev;
    }
    cur = nrNodes + 1;
    while (from[cur] > -1) {
        int prev = from[cur];
        Edge e = getEdge(prev, cur);
        e.addFlow(pathCap);
        cur = prev;
    }
    return pathCap;
}
```

```
class Edge {
    int a, b, cap, flow;
    public Edge(int a, int b, int cap) {
        this.a = a; this.b = b; this.cap = cap; this.flow = 0;
    }
    public int getCapacity() { return cap - flow; }
    public void addFlow(int n) { flow += n; }
}
```

Dijkstra

```
PriorityQueue<Integer> qu = new PriorityQueue<Integer>();
qu.add(s);
distances.remove(s);
distances.put(s, 0);
while (!qu.isEmpty()) {
    int current = qu.poll();
    for (int i = 0; i < m; i++) {
        if (maze[i].start == current) {
            int newDistance = distances.get(maze[i].start);
            if (newDistance >= 5) {
                newDistance += 2 * maze[i].weight;
            } else {
                newDistance += maze[i].weight;
            }
            if (newDistance < distances.get(maze[i].end)) {
                qu.add(maze[i].end);
                distances.remove(maze[i].end);
                distances.put(maze[i].end, newDistance);
            }
        }
    }
}
int output = distances.get(t);
```

Floyd-Warshall

```
for(int k = 0; k < n; k++)
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            if(dist[i][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];
```

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];
}
```

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
```

Kruskal's Algorithm (Minimum Spanning Tree)

```
static int[] parent;
static int[] rank;

static void solve(Scanner sc) throws Exception {
    int n = sc.nextInt();        // number of nodes
    int m = sc.nextInt(), s=0; // number of edges (in graph and in tree)
    Edge[] edges = new Edge[m];
    Edge[] tree = new Edge[n-1];
    for (int i = 0; i < m; i++)
        edges[i] = new Edge(sc.nextInt(), sc.nextInt(), sc.nextInt());
    parent = new int[n]; // disjoint-set
    rank = new int[n];   // datastructure
    for (int i = 0; i < n; i++) { parent[i] = i; }
    Arrays.sort(edges);
    for (Edge e : edges) { if (join(e.x, e.y)) { tree[s++] = e; }}
    for (Edge e : tree)
        System.out.println("("+e.x+", "+e.y+", ["+e.l+"])");
}

static int find(int x) {
    return parent[x]==x ? x : (parent[x] = find(parent[x]));
}

static boolean join(int x, int y) { // false if x & y are in same set
    int xrt = find(x);
    int yrt = find(y);
    if (rank[xrt] > rank[yrt]) {
        parent[yrt] = xrt;
    } else if (rank[xrt] < rank[yrt]) {
        parent[xrt] = yrt;
    } else if (xrt != yrt) {
        rank[parent[yrt]=xrt]++;
    }
    return xrt != yrt;
}

static class Edge implements Comparable<Edge> {
    int x, y, l; // from, to and length
    public Edge(int from, int to, int length){x=from; y=to; l=length;}
    public int compareTo(Edge e) {return l - e.l;}
}
```

Bellman Ford

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

```java
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]);
System.out.println();
```

Prime Factorization

```java
List<Integer> primeFactors(int n) {
    List<Integer> factors = new ArrayList<>();
//  for (int i = 2; i * i <= n; ++i) {   <-- WRONG
    for (int i = 2; i <= n / i; ++i) {
        while (n % i == 0) {
            factors.add(i);
            n /= i;
        }
    }
    if (n > 1) { factors.add(n); }
    return factors;
}
```

Convex Hull

```java
class ConvexHull {
    static final double EPSILON = 0.0001;
    public static Point[] getConvexHull(Point[] p) {
        Arrays.sort(p);
        List<Point> hull = new ArrayList<Point>();
        hull.add(p[0]);
        for (int i = 1, min = hull.size() + 1; i < p.length; i++) {
            hull.add(p[i]);
            while (hull.size() > min && !endsWithRightTurn(hull))
                hull.remove(hull.size() - 2);
        }
        for (int i = p.length - 2, min = hull.size() + 1; i >= 0; i--) {
            hull.add(p[i]);
            while (hull.size() > min && !endsWithRightTurn(hull))
                hull.remove(hull.size() - 2);
        }
        hull.remove(hull.size() - 1);
        return hull.toArray(new Point[hull.size()]);
    }

    private static boolean endsWithRightTurn(List<Point> hull) {
        Point a = hull.get(hull.size() - 3);
        Point b = hull.get(hull.size() - 2);
        Point c = hull.get(hull.size() - 1);
        return area(a, b, c) < -EPSILON;
    }

    private static double area(Point a, Point b, Point c) {
        return (a.x*b.y-a.y*b.x+a.y*c.x-a.x*c.y+b.x*c.y-c.x*b.y)/2;
    }

    public static class Point implements Comparable<Point> {
        public double x, y;
        public Point(double x, double y) { this.x = x; this.y = y; }

        @Override
        public int compareTo(Point that) {
            if (this.x < that.x) return -1;
            if (this.x > that.x) return 1;
            if (this.y < that.y) return -1;
            if (this.y > that.y) return 1;
            return 0;
        }
    }
}
```

Coin Change

```java
// 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

```java
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()];
}
```

Switch Base of Numbers

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

Binary Search

```java
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

```python
# Find maximum of unimodal function f() within [left, right]
# For the minimum, revert the if/else statement or revert the comparison.
def ternarySearch(f, left, right, absolutePrecision):
    while True:
    # left and right are the current bounds; the maximum is between them
        if abs(right - left) < absolutePrecision:
            return (left + right)/2
        leftThird = left + (right - left)/3
        rightThird = right - (right - left)/3
        if f(leftThird) < f(rightThird):
            left = leftThird
        else:
            right = rightThird
```

Permutations

```java
void permutation(String str) { permutation("", str); }

void permutation(String prefix, String str) {
    int n = str.length();
    if (n == 0) System.out.println(prefix);
    else {
        for (int i = 0; i < n; i++)
            permutation(prefix + str.charAt(i), str.substring(0, i) + str.
    substring(i+1, n));
    }
}
```

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 Diamter

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

```java
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

```java
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);
}
```

Polygon Area

```
double polygonArea(double[] X, double[] Y) {
    double area = 0;
    j = X.length - 1;
    for (i = 0; i < X.lengtjh; i++) {
        area = area +  (X[j]+X[i]) * (Y[j]-Y[i]);
        j = i;
    }
    return area / 2;
}
```

Topological Sort

```
public class TopologicalSort {
    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;
    }
    public 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).

```java
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;
}
```

Binom

```
// Guaranteed not to overflow, as longas the answer fits into 53 bits.
public static long binom (int n , int k) {
    if (k > n / 2)
        k = n     k;
    double ans = 1;
    for (; k > 0 ; k      , n      )
        ans *= (double) n / k;
    return Math.round(ans);
}
```

Repeating Float to Fraction

```java
// 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));
}
```

Segment Tree

```java
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;
    }
}
```

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.

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

Closest-pair Problem

```java
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;
    }
}
```

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.

$$L(x) = \sum_{j=0}^{k} y_j \ell_j(x)$$

$$\ell(x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$$

Example

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

$$\begin{aligned} x_0 &= 1 & f(x_0) &= 1 \\ x_1 &= 2 & f(x_1) &= 4 \\ x_2 &= 3 & f(x_2) &= 9 \end{aligned}$$

Solution:

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

Triangles

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

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

More Math

$$\text{nr regions by } n \text{ lines} = \frac{1}{2}n(n+1) + 1$$

$$\text{nr bounded regions by } n \text{ lines} = \frac{1}{2}(n^2 - 3n + 2)$$

$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{i=1}^{n} i^3 = \frac{1}{4}n^2(n+1)^2$$

Strongly Connected Components

A directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph. In a directed graph G that may not itself be strongly connected, a pair of vertices u and v are said to be strongly connected to each other if there is a path in each direction between them.

```java
public final class Kosaraju { // for DirGraph see 2-SAT
    public static <T> Map<T, Integer> scc(DirGraph<T> g) {
        Stack<T> visitOrder = dfsVisitOrder(graphReverse(g));
        Map<T, Integer> result = new HashMap<T, Integer>();
        int iteration = 0;
        while (!visitOrder.isEmpty()) {
            T startPoint = visitOrder.pop();
            if (result.containsKey(startPoint)) continue;
            markRNodes(startPoint, g, result, iteration);
            ++iteration;
        }
        return result;
    }
    static <T> DirGraph<T> graphReverse(DirGraph<T> g) {
        DirGraph<T> result = new DirGraph<T>();
        for (T node: g) result.addNode(node);
        for (T node: g) { for (T endpoint: g.edgesFrom(node)) {
                result.addEdge(endpoint, node); }}
        return result;
    }
    static <T> Stack<T> dfsVisitOrder(DirGraph<T> g) {
        Stack<T> result = new Stack<T>();
        Set<T> visited = new HashSet<T>();
        for (T node: g) recExplore(node, g, result, visited);
        return result;
    }
    static <T> void recExplore(T node, DirGraph<T> g,
            Stack<T> result, Set<T> visited) {
        if (visited.contains(node)) return;
        visited.add(node);
        for (T endpoint: g.edgesFrom(node))
            recExplore(endpoint, g, result, visited);
        result.push(node);
    }
    static <T> void markRNodes(T node,DirGraph<T> g,Map<T,Int> r,int l) {
        if (r.containsKey(node)) return; r.put(node, l);
        for (T endpoint: g.edgesFrom(node)) markRNodes(endpoint, g, r, l);
    }
}
```

2-SAT

```java
public class TwoSat { // for Kosaraju.scc(DirGraph) see previous page
    public static <T> boolean isSatisfiable(List<Clause<T>> formula) {
        Set<T> vars = new HashSet<T>();
        for (Clause<T> clause: formula) {
            vars.add(clause.first.value);
            vars.add(clause.second.value);
        }
        DirectedGraph<Literal<T>> impls = new DirectedGraph<Literal<T>>();
        for (T variable: vars) {
            impls.addNode(new Literal<T>(variable, true));
            impls.addNode(new Literal<T>(variable, false));
        }
        for (Clause<T> clause: formula) {
            impls.addEdge(clause.first.negation(), clause.second);
            impls.addEdge(clause.second.negation(), clause.first);
        }
        Map<Literal<T>, Integer> scc = Kosaraju.scc(impls);
        for (T variable: vars) {
            if (scc.get(new Literal<T>(variable, true))
                    .equals(scc.get(new Literal<T>(variable, false))))
                return false;
        }
        return true;
    }
}
```

```java
public class Literal<T> {
    public final T value;
    public final boolean isPos;
    public Literal(T value, boolean isPos) {
        this.value = value; this.isPos = isPos;
    }
    public Literal<T> negation() { return new Literal<T>(value, !isPos); }
    public String toString() { return (isPos ? "" : "~") + value; }
    public boolean equals(Object obj) {
        if (!(obj instanceof Literal)) return false;
        Literal<?> realObj = (Literal) obj;
        return realObj.isPos == isPos && realObj.value.equals(value);
    }
    public int hashCode() {
        return (isPos ? 1 : 31) * value.hashCode();
    }
}
```

```java
public final class Clause<T> {
    public final Literal<T> first, second;
    public Clause(Literal<T> one, Literal<T> two) {
        first = one; second = two;
    }
    public String toString() {
        return "(" + first + " or " + second + ")";
    }
}
```

```java
public final class DirectedGraph<T> implements Iterable<T> {
    private final Map<T, Set<T>> mGraph = new HashMap<T, Set<T>>();
    public boolean addNode(T node) {
        if (mGraph.containsKey(node))
            return false;
        mGraph.put(node, new HashSet<T>());
        return true;
    }
    public void addEdge(T start, T dest) {
        if (!mGraph.containsKey(start) || !mGraph.containsKey(dest))
            throw new NoSuchElementException("Both must be in the graph");
        mGraph.get(start).add(dest);
    }
    public void removeEdge(T start, T dest) {
        if (!mGraph.containsKey(start) || !mGraph.containsKey(dest))
            throw new NoSuchElementException("Both must be in the graph");
        mGraph.get(start).remove(dest);
    }
    public boolean edgeExists(T start, T end) {
        if (!mGraph.containsKey(start) || !mGraph.containsKey(end))
            throw new NoSuchElementException("Both must be in the graph");
        return mGraph.get(start).contains(end);
    }
    public Set<T> edgesFrom(T node) {
        Set<T> arcs = mGraph.get(node);
        if (arcs == null)
            throw new NoSuchElementException("Source does not exist.");
        return Collections.unmodifiableSet(arcs);
    }
    public Iterator<T> iterator() { return mGraph.keySet().iterator(); }
}
```

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;
}
```

Problem: Solve It

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 $0 \le x \le 1$.

```
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;
        }
    }
}
```