All pairs shortest path (Floyd-Warshall)

```
static int[][][] predecessor;
static double[][] graph;
static double[][][] distance;

for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
                dist[i][j][0] = rates[i][j];

for(int k = 1; k < size; k++)
        for(int i = 0; i < size; i++)
                for(int j = 0; j < size; j++) {
                        int best = 0;
                        for(int x = 1; x < size; x++) {
                                if(dist[i][x][k-1]+dist[x][j][k-1] <
                                   dist[i][best][k-1]+dist[best][j][k-1])
                                        best = x;
                                dist[i][j][k] = dist[i][best][k-1] +
                                                        dist[best][j][k-1];
                                path[i][j][k] = best;
                        }
                }

static void print(int i, int j, int k) {
        if(k==0) {
                System.out.print((i+1) + " " + (j+1));
                return;
        }
        print(i, path[i][j][k], k-1);
        System.out.print(" " + (j+1));
}
```

Maximum Sum

The following finds the maximum sum of paths in a graph

```
int[][] sums = new int[n][n];
for(int j = 0; j < n; j++)
        for(int i = 0; i < n; i++)
                sums[i][j] = (i==0)
                        ? matrix[i][j] : sums[i-1][j]+matrix[i][j];

int maximum = Integer.MIN_VALUE;
for(int i = 0; i < n; i++)
        for(int j = 0; j < i; j++) {
                int[] best = new int[n];
                for(int k = 0; k < n; k++) {
                        best[k] = sums[i][k]-sums[j][k];
                        if(k>0 && best[k]+best[k-1]>best[k])
                                best[k] = best[k]+best[k-1];
                        if(best[k] > maximum) maximum = best[k];
                }
        }
System.out.println(maximum);
```

Shortest Path ala Bellman Ford

Slower than Dijkstra but easier to implement. When complete, path resides in p[].

```
static void initializeSingleSource(int startNode) {
        d = new int[nnodes]; p = new int[nnodes];
        for(int i = 0; i < nnodes; i++) {
                d[i] = 999999;
                p[i] = -1;
        }
        d[startNode] = 0;
}
```

```
static void relax(int u, int v, int weight) {
	if(d[u] + weight < d[v]) {
		d[v] = d[u] + weight;
		p[v] = u;
	}
}

static void bellmanFord() {
	initializeSingleSource(startNode);

	for(int k = 0; k < nnodes-1; k++)
		for(int i = 0; i < nnodes; i++)
			for(int j = 0; j < nnodes; j++)
				relax(i, j, graph[i][j]);
}
```

Total Paths Between Nodes

```
for(int k = 0; k < maxNodes; k++)
	for(int i = 0; i < maxNodes; i++)
		for(int j = 0; j < maxNodes; j++)
			matrix[i][j] += matrix[i][k] * matrix[k][j];
```

To remove cycles (which would have infinite paths):

```
for(int k = 0; k < maxNodes; k++) if(matrix[k][k] > 0)
	for(int i = 0; i < maxNodes; i++)
		for(int j = 0; j < maxNodes; j++)
			if(matrix[i][k] != 0 && matrix[k][j] != 0)
				matrix[i][j] = -1;
```

Area of a Polygon

```
double calcArea() {
	Point[] tmp = new Point[npoints+2];
	for(int i = 0; i < npoints; i++) tmp[i] = points[i];
	tmp[npoints] = points[0]; tmp[npoints+1] = points[1];

	double area = 0.0;
	for(int i=1, j=2, k=0; i<=npoints; i++, j++, k++)
		area += (tmp[i].x * (tmp[j].y-tmp[k].y));

    return area / 2;
}
```

Point in Polygon

```
public boolean containsPoint(Point p) {
	Point[] tmp = new Point[npoints+1];
	for(int i = 0; i < npoints; i++) {
		if(points[i].equals(p)) return true;
		tmp[i] = points[i];
	}
	tmp[npoints] = points[0];

	int wn = 0;
	for(int i = 0; i < npoints; i++) {
		if(tmp[i].y <= p.y) {
			if(tmp[i+1].y > p.y)
				if(isLeft(tmp[i], tmp[i+1], p) > 0)
					wn++;
		} else {
			if(tmp[i+1].y <= p.y)
				if(isLeft(tmp[i], tmp[i+1], p) < 0)
					wn--;
		}
	}

	return (wn!=0);
}
```

```
private static double isLeft(Point p0, Point p1, Point p2) {
	return (p1.x-p0.x)*(p2.y-p0.y) - (p2.x-p0.x)*(p1.y-p0.y);
}
```

Angle Between Points

```
private static double getAngle(Point p1, Point p2)
	{return Math.atan2(p2.y-p1.y, p2.x-p1.x);}
```

Radial Sort of Points

```
private static Point[] radialSort(Point[] points, int npoints) {
	Point[] sorted = new Point[npoints];
	int lowest = 0;
	for(int i = 1; i < npoints; i++)
		if(points[i].y < points[lowest].y) lowest = i;
		else if(points[i].y == points[lowest].y)
			if(points[i].x > points[lowest].x) lowest = i;

	int count = 0;
	sorted[count++] = points[lowest];
	points[lowest] = null;

	double[] angles = new double[npoints];
	for(int i = 0; i < npoints; i++)
		if(points[i] != null) angles[i] = getAngle(sorted[0],
			points[i]);

	boolean done = false;
	while(!done) {
		done = true;
		int smallAngle = 0;
		for(int i = 0; i < npoints; i++)
			if(points[i] != null) {
				smallAngle = i;
				done = false;
				break;
			}

		if(done) continue;

		for(int i = 0; i < npoints; i++)
			if(points[i] != null)
				if(angles[i] < angles[smallAngle])
					smallAngle = i;

		sorted[count++] = points[smallAngle];
		points[smallAngle] = null;
	}
	return sorted;
}
```

Convex Hull of a Polygon

```
public static Polygon convexHull(Point[] points, int npoints) {
	Point[] sortedPoints = radialSort(points, npoints);

	Stack stack = new Stack(npoints);
	stack.push(sortedPoints[0]);
	stack.push(sortedPoints[1]);
	stack.push(sortedPoints[2]);

	int i = 3;
	while(i < npoints) {
		double left = isLeft((Point) stack.nextToTop(),
			(Point) stack.top(), sortedPoints[i]);

		if(left >= 0) stack.push(sortedPoints[i++]);
		else stack.pop();
```

```
        }

        int size = stack.size();
        Object[] tmp = stack.toArray();
        Point[] hull = new Point[size];
        for(i = 0; i < size; i++) hull[i] = (Point) tmp[i];

        return new Polygon(hull, size);
}
```

Euclid's GCD

```
int euclid(int a, int b) {
        while(a != b)
                if(a<b) b = b-a;
                else a = a-b;
        return a;
}
```

Another Pythagoras

```
x = b*b – a*a
y = 2*a*b
z = a*a + b*b
```

Now $x^2 + y^2 = z^2$.

Finding Roots

The following finds if x is a whole power of i.

```
static int isPower(int x, int i) {
        int count = 0;
        while(x%i == 0) {
                x /= i; count++;
        }
        return (x==1) ? count : 0;
}
```

Misc. Stuff

Value of π = 3.14159265358979323846264338327950

Ad Hoc HashTable

```
static String[] nameTable = new String[1000];
static int getNameIndex(String s) {
        for(int i = 0; i < nnodes; i++)
                if(nameTable[i].equals(s)) return i;

        nameTable[nnodes] = s;
        return nnodes++;
}
```

Printing a tree ala BFS

```
String output = new String();
int maxLevel = 10;
while(maxLevel > -1)
        output = printTree(root, maxLevel--) + output;

static String printTree(Tree leaf, int level) {
        if(level == 0) {return leaf.value + " ";}

        String ret = new String();
        if(leaf.left != null) ret += printTree(leaf.left, level-1);
        if(leaf.right != null) ret += printTree(leaf.right, level-1);
```

```
        return ret;
}
```

Permutation Engine

```
Object[] objects = {whatever}

int nperms = factorial(n);
for(int i = 0; i < nperms; i++) {
        int[] perm = numberToPermutation(i, n);
        for(int j = 0; j < n; j++)
                objectPerm[j] = objects[perm[j]];
}

static int[] numberToPermutation(int value, int size) {
        int[] permutationSet = new int[size];
        int[] offsets = new int[size];

        int nperms = 0, noffsets = 0;
        int remainder = value, curSize = size, segment;
        for(int i = 0; i < size; i++) offsets[i] = i;
        while(curSize-- > 1) {
                segment = (remainder / factorial(curSize)) % size;
                permutationSet[nperms++] = offsets[segment];
                offsets = removeElement(segment, offsets);
                remainder = remainder % factorial(curSize);
        }
        permutationSet[nperms++] = offsets[0];
        return permutationSet;
}

static int[] removeElement(int pos, int[] arr) {
        int[] tmp = new int[arr.length-1];
        for(int i = 0; i < pos; i++) tmp[i] = arr[i];
        for(int i = pos+1; i < arr.length; i++) tmp[i-1] = arr[i];
        return tmp;
}

static int factorial(int n)
        {int r=n;while(--n>0)r*=n;return r;}
}
```

The Knapsack Problem

The knapsack problem is a particular type of integer program with just one constraint. Each item that can go into the knapsack has a size and a benefit. knapsack has a certain capacity. What should go into the knapsack so as to maximize the total benefit? As an example, suppose we have three items a Table 4, and suppose the capacity of the knapsack is 5.

Item (j)	Weight (w_j)	Benefit(b_j)
1	2	65
2	3	80
3	1	30

Table 4: Knapsack Items

The stages represent the items: we have three stages j=1,2,3. The state y_j at stage j represents the total weight of items j and all following items in the l The decision at stage j is how many items j to place in the knapsack. Call this value k_j.

This leads to the following recursive formulas: Let $f_j(y_j)$ be the value of using y_j units of capacity for items j and following. Let $\lfloor a \rfloor$ represent the larg less than or equal to a.

- $f_3(y_j) = 30y_j$
- $f_j(y_j) = \max_{k_j \le \lfloor y_j / w_j \rfloor} \{b_j k_j + f_{j+1}(y_j - w_j k_j)\}$.

Making change

Input: coin denominations d[0], d[1],..., d[n-1], an amount a
Output: the number of coins needed to total a exactly.
Consider Freedonia: we have pennies, 4-cent pieces, and nickels; and we have a=8. The output should be 2 (since we could use two four-cent pieces), even though the greedy method would output 4 (a nickel and three pennies).
Formulating a solution
Think of a recursive solution:

$$\text{MakeChange}(a) = 1 + \min_{0 \le i < n} \text{MakeChange}(a - d[i])$$

Compute bottom up

```
Algorithm DynamicMakeChange(amt, d[0], d[1],..., d[n-1]):
Let coins[0] be 0.
for each a from 1 to amt, do:
  coins[a] <- infinity // an upper bound
  for each j from 0 to n - 1, do:
    if d[j] <= a and 1 + coins[a - d[j]] < coins[a], then:
      Let coins[a] be 1 + coins[a - d[j]].
    end of if
  end of loop
end of loop
return coins[amt].
```

With our earlier example, coins would look like
| 0 | 1 | 2 | 3 | 1 | 1 | 2 | 3 | 2 |

Baskets

```
  public static void main(String args[]) throws IOException {
        BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));
        String line;
        while(true) {
            int n = Integer.parseInt(bfr.readLine()), i, max, j;
            if (n < 0) break;
            int[] baskets = new int[n];
            int[] table = new int[n];
            for(i = 0; i < n; i++) {
                baskets[i] = Integer.parseInt(bfr.readLine());
                table[i] = 0;
            }
            for(i = 0; i < n; i++) {
                if (i == 0) table[i] = 1;
                else {
                    max = 0;
                    for(j = (i-1); j >= 0; j--) {
                        if (baskets[j] < baskets[i] && table[j] > max)
                            max = table[j];
                    }
                    table[i] = max+1;
                }
            }
            max = 0;
             for(i = 0; i < n; i++) {
            if (table[i] > max)
                max = table[i];
              }
      System.out.println(max);
    }
  }
```

Word Sums:

```
class E {
   static int[] assign = new int[27];
   public static void main(String[] args) throws IOException {
         BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));
         String line;
         while((line = bfr.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(line);
            String w1 = st.nextToken();
            String w2 = st.nextToken();
            String r = st.nextToken();
            int i = Math.abs(w1.length()-w2.length());
            String pad = "", t1 = w1, t2 = w2;
            while(i-- > 0) pad += (char)('z'+1);
            if (w1.length() > w2.length()) w2 = pad + w2;
            else w1 = pad + w1;
            for(i = 0; i < 26; i++) assign[i] = -1;
            assign[26] = 10;
            if (checkResult(w1, w2, r, 0)) {
                  System.out.println(t1 + " + " + t2 + " = " + r + " -> " +
                                          wordVal(t1) + " + " + wordVal(t2) + " = " + wordVal(r));
            } else {
                  System.out.println(t1 + " + " + t2 + " = " + r + " -> No solution");
            }
         }
   }
   static boolean checkResult(String w1, String w2, String r, int carry) {
         if (w1.length() == 0) {
            if (carry == 0 && r.length() > 0) return false;
            if (carry > 0) {
                  if (r.length() == 1) {
                     int v = assign[r.charAt(0)-'a'];
                     if ((v >= 0 &&  v != 1) || (v < 0 && !check(1))) return false;
                     assign[r.charAt(0)-'a'] = 1;
                  } else {
                     return false;
                  }
            }
            return true;
         }
         char c1 = w1.charAt(w1.length()-1);
         char c2 = w2.charAt(w2.length()-1);
         char cr = r.charAt(r.length()-1);
         boolean b1 = assign[c1-'a'] >= 0, b2 = assign[c2-'a'] >= 0, br = assign[cr-'a'] >= 0;
         int v1 = assign[c1-'a']%10, v2 = assign[c2-'a']%10, vr = assign[cr-'a'];
         for(int i = 0; i < 10; i++) {
            for(int j = 0; j < 10; j++) {
                  if (!b1 && check(i)) {v1 = i; assign[c1-'a'] = i;}
                  if (!b2 && check(j)) {v2 = j; assign[c2-'a'] = j;}
                  if (c1 == c2) v1 = v2;
                  if (((w1.length() > 1 && v1 >= 0 && v2 >= 0) || (c1 > 'z')) ||
                     ((w1.length() == 1 && v1 > 0 && v2 > 0) || (c2 > 'z'))) {
                     int newVal = v1+v2+carry;
                     if (!br) {vr = newVal%10;}
                     if (newVal%10 == vr) {
```

```
                                        if (br || check(vr) || cr == c1 || cr == c2) {
                                           if (!((cr == c1 && v1 != vr) || (cr == c2 && v2 != vr))) {
                                                assign[cr-'a'] = newVal%10;
                                                boolean b = checkResult(w1.substring(0, w1.length()-1),
w2.substring(0, w2.length()-1),
                                                                                     r.substring(0, r.length()-1),
(newVal>=10)?1:0);
                                                if (b) return true;
                                           }
                                        }
                                   }
                              }
                              if (!b1) {assign[c1-'a'] = -1; v1 = -1;}
                              if (!b2) {assign[c2-'a'] = -1; v2 = -1;}
                              if (!br) assign[cr-'a'] = -1;
                    }
               }
               return false;
   }
   static boolean check(int val) {
               for(int i = 0; i < 26; i++) if (assign[i] == val) return false;
               return true;
   }
   static String wordVal(String w) {
               String val = "";
               for(int i = 0; i < w.length(); i++) {
                   val += ""+assign[w.charAt(i)-'a'];
               }
               return val;
   }
}
```

Max Flow:

```
import java.util.*;
import java.io.*;
class flow {
   static class Val {
               Val(int c) {this.c = c;}
               int c;
               int f = 0;
   }
   static class Node {
               String name;
               int d;
               Node parent;
               Vector n = new Vector();
               Vector v = new Vector();
   }
   public static void main(String[] args) {
               Vector g = new Vector();
               Node s, v1, v2, v3, v4, t;
               s = new Node();
               s.name = "s";
               v1 = new Node();
               v1.name = "v1";
               v2 = new Node();
```

```
        v2.name = "v2";
        v3 = new Node();
        v3.name = "v3";
        v4 = new Node();
        v4.name = "v4";
        t = new Node();
        t.name = "t";
        s.n.add(v1); s.n.add(v2); s.v.add(new Val(16)); s.v.add(new Val(13));
        v1.n.add(v2); v1.n.add(v3); v1.v.add(new Val(10)); v1.v.add(new Val(12));
        v2.n.add(v1); v2.n.add(v4); v2.v.add(new Val(4)); v2.v.add(new Val(14));
        v3.n.add(v2); v3.n.add(t); v3.v.add(new Val(9)); v3.v.add(new Val(20));
        v4.n.add(v3); v4.n.add(t); v4.v.add(new Val(7)); v4.v.add(new Val(4));
        g.add(s); g.add(v1); g.add(v2); g.add(v3); g.add(v4); g.add(t);
        maxFlow(g, s, t);
        for(int i = 0; i < g.size(); i++) {
           Node u = (Node)g.get(i);
           for(int j = 0; j < u.n.size(); j++) {
                Node v = (Node)u.n.get(j);
                Val val = (Val)u.v.get(j);
                System.out.println(u.name + " " + v.name + " " + val.c + "/" + val.f);
           }
        }
}
static int maxFlow (Vector g, Node s, Node t) {
        int max = 0;
        Val val;
        while (dijkstra(g, s, t)) {
           int increment = 99999999;
           Node n = t;
           while(n.parent != null) {
                val = pred(n);
                increment = Math.min(increment, val.c - val.f);
                n = n.parent;
           }
           n = t;
           while(n.parent != null) {
                val = pred(n);
                if (val != null) val.f += increment;
                val = succ(n, n.parent);
                if (val != null) val.f -= increment;
                n = n.parent;
           }
           max += increment;
        }
        return max;
}
static Val pred(Node u) {
        int i = u.parent.n.indexOf(u);
        if (i >= 0) {
           return (Val)u.parent.v.get(i);
        }
        return null;
}
static Val succ(Node u, Node v) {
        int i = u.n.indexOf(v);
        if (i >= 0) {
```

```
            return (Val)u.v.get(i);
        }
        return null;
    }
    static boolean dijkstra(Vector g, Node s, Node t) {
        initSingleSource(g, s);
        Vector q = (Vector)g.clone();
        while(q.size() > 0) {
            int min = 99999999+1;
            Node u = null;
            for(int i = 0; i < q.size(); i++) {
                Node m = (Node)q.get(i);
                if (m.d < min) {
                    min = m.d;
                    u = m;
                }
            }
            q.removeElement(u);
            for(int i = 0; i < u.n.size(); i++) {
                Node v = (Node)u.n.get(i);
                relax(u, v);
            }
        }
        return (t.parent != null);
    }
    static void initSingleSource(Vector g, Node s) {
        for(int i = 0; i < g.size(); i++) {
            Node n = (Node)g.get(i);
            n.parent = null;
            n.d = 99999999;
        }
        s.d = 0;
    }
    static void relax(Node u, Node v) {
        Val val = succ(u, v);
        if (val != null) {
            int dist = val.c - val.f;
            if (dist > 0) {
                if (v.d > (u.d + dist)) {
                    v.parent = u;
                    v.d = u.d + dist;
                }
            }
        }
    }
}
```

Permutation:

```
    static Vector numberToPermutation(int value, int size) {
        int i;
        Vector permutationSet = new Vector();
        Vector offsets = new Vector();
        int remainder = value;
        int curSize = size;
        int segment;
        for(i = 1; i <= size; i++) offsets.add(new Integer(i));
        while(curSize > 1) {
```

```
        curSize--;
        segment = (remainder/factorial(curSize)) % size;
        permutationSet.add(offsets.elementAt(segment));
        offsets.removeElementAt(segment);
        remainder = remainder % factorial(curSize);
            }
      permutationSet.add(offsets.elementAt(0));
      return permutationSet;
   }
   static int factorial(int n) {
            int ret = n;
            while(--n > 0) {
               ret *= n;
            }
            return ret;
   }
```

Point in a Polygon:

```
class polygon1 {
   static class Line {
            public double a;
            public double b;
            public boolean p;
            public double xmin;
            public double xmax;
   }
   public static void main(String args[]) {
            BufferedReader bfr;
            StringTokenizer st;
            String line;
            int vertices, i, j, k, oddCount;
            double x1=0 , y1=0, x2=0, y2=0, x0=0, y0=0;
            Line v;
            Line[] vp;
            try {
               bfr = new BufferedReader(new FileReader(args[0]));
               line = bfr.readLine();
               vertices = Integer.parseInt(line.trim());
               vp = new Line[vertices-1];
               for(i = 0; i < vertices; i++) {
                        line = bfr.readLine();
                        st = new StringTokenizer(line);
                        x1 = Integer.parseInt(st.nextToken());
                        y1 = Integer.parseInt(st.nextToken());
                        if (i != 0) {
                           v = new Line();
                           if (x1 == x2) {
                                    v.p = true;
                                    v.a = x1;
                                    v.xmax = Math.max(y1, y2);
                                    v.xmin = Math.min(y1, y2);
                           } else {
                                    v.p = false;
                                    v.a = (y2-y1)/(x2-x1);
                                    v.b = y1-x1*v.a;
                                    v.xmax = Math.max(x1, x2);
                                    v.xmin = Math.min(x1, x2);
```

```
            }
            vp[i-1] = v;
            d(v.a + " " + v.b + " " + v.xmin + " " + v.xmax);
        } else {
            x0 = x1;
            y0 = y1;
        }
        if (i == (vertices-1)) {
            v = new Line();
            if (x1 == x0) {
                    v.p = true;
                    v.a = x1;
                    v.xmax = Math.max(y1, y2);
                    v.xmin = Math.min(y1, y2);
            } else {
                    v.p = false;
                    v.a = (y0-y1)/(x0-x1);
                    v.b = y1-x1*v.a;
                    v.xmax = Math.max(x1, x0);
                    v.xmin = Math.min(x1, x0);
            }
            vp[i-1] = v;
            d(v.a + " " + v.b + " " + v.xmin + " " + v.xmax);
        }
        x2 = x1;
        y2 = y1;

}
while((line = bfr.readLine()) != null) {
        st = new StringTokenizer(line);
        x1 = Integer.parseInt(st.nextToken());
        y1 = Integer.parseInt(st.nextToken());
        oddCount = 0;
        for(i = 0; i < (vertices-1); i++) {
            if (!vp[i].p && (y1 == vp[i].a*x1 + vp[i].b)) {
                    oddCount = 0;
                    break;
            } else if (vp[i].p && (vp[i].xmin <= y0) && (vp[i].xmax >= y0)) {
                    oddCount = 0;
                    break;
            }
            if ((x1 > vp[i].xmin) && (x1 < vp[i].xmax)) {
                    d("c:" + x1 + " " + vp[i].xmin + " " + vp[i].xmax);
                    y2 = vp[i].a*x1 + vp[i].b;
                    if (y2 < y1) {
                        oddCount++;
                        v = vp[i];
                        d("x:" + v.a + " " + v.b + " " + v.xmin + " " + v.xmax);
                    }
            }
        }
        d("" + oddCount);
        if ((oddCount % 2) == 0) {
            s("NO");
        } else {
            s("YES");
```

```
                }
            }

        } catch (IOException ex) {
            s(ex.getMessage());
        }
    }
    static void s(String p) {
        System.out.println(p);
    }
    static void d(String p) {
        //s(p);
    }
}
```

Parsing:

```
class pdas1 {
    static Vector rules = new Vector();
    static Vector vars = new Vector();
    static boolean accept = false;
    public static void main(String[] args) {
        vars.addElement("E");
        rules.addElement("T");
        vars.addElement("E");
        rules.addElement("T<E");
        vars.addElement("T");
        rules.addElement("F");
        vars.addElement("T");
        rules.addElement("F*T");
        vars.addElement("F");
        rules.addElement("x");
        vars.addElement("F");
        rules.addElement("(E)");
        vars.addElement("F");
        rules.addElement("x<T>");
        parse(args[0], "E");
        if (accept) {
            s("yes");
        } else {
            s("no");
        }
    }
    static void parse(String word, String stack) {
        s(word + ":" + stack);
        int i;
        char c;
        if (accept || !validate(word, stack)) {
            return;
        }
        if (stack.length() == 0) {
            if (word.length() == 0) {
                accept = true;
                d("success");
                return;
            } else {
                return;
            }
```

```
        }
        if (isRule(c = stack.charAt(0))) {
          for(i = 0; i < vars.size(); i++) {
                if (c == ((String)vars.get(i)).charAt(0)) {
                  parse(word, (String)rules.get(i) + stack.substring(1));
                }
          }
        } else {
          if ((word.length() == 0) || (word.charAt(0) != c)) {
                return;
          } else {
                parse(word.substring(1), stack.substring(1));
          }
        }
  }
  static boolean validate(String word, String stack) {
        if (word.length() < (stack.length() - 2)) {
          return false;
        }
        return true;
  }
  static boolean isRule(char c) {
        if ((c >= 'A') && (c <= 'Z')) {
          return true;
        }
        return false;
  }
```

Rectangle area:

```
/* Calculate the area covered by a set of rectangles. The area of regions
 * covered by more than rectangle should only be counted once.
 * You may assume that the standard input file contains four integer values
 * x1, y1, x2, y2 on each line where (x1,y1) and (x2,y2) are diagonally opposite
 * points defining some rectangle.
 * There may be any number of rectangles followed by end-of-file. */
import java.io.*; import java.util.*;
class Rectangle { int x1, y1, x2, y2; // Rectangle with top-left bottom-right
 public Rectangle(int a, int b, int c, int d){ // a,b and c,d are diagonally opposite
   x1 = Math.min(a,c); x2 = Math.max(a,c);
   y1 = Math.min(b,d); y2 = Math.max(b,d);
 }
}
class RectList {  public Rectangle head; public RectList tail;
 public RectList ( Rectangle r, RectList list) { head = r; tail = list; }
}
public class Rect{
 static RectList union ( Rectangle r, RectList list) {
  // list is a list of disjoint Rects
  // breaks r into smaller Rects which do not overlap any Rect in the list
  // returns combined list of disjoint Rects
  if ( list == null ) return new RectList(r, null);
  for (RectList p = nonOverlap( r, list.head ); p != null; p = p.tail)
    list.tail = union( p.head, list.tail );
  return list;
 }
 static RectList nonOverlap( Rectangle r, Rectangle s ) {
  if( r.x2 <= s.x1 || r.x1 >= s.x2 || r.y2 <= s.y1 || r.y1 >= s.y2 )
```

```
   return new RectList( r, null ); // no part of r overlaps s
 RectList list = null; // list of subrectangles of r which don't overlap s
 if ( r.x1 < s.x1 ) // calculate part of r to the left of s
   list = new RectList( new Rectangle(r.x1, r.y1, s.x1, r.y2), list );
 if ( r.x2 > s.x2 ) // calculate part of r to the right of s
   list = new RectList( new Rectangle(s.x2, r.y1, r.x2, r.y2), list );
 int a = Math.max( r.x1, s.x1 ), b = Math.min( r.x2, s.x2);
 if ( r.y1 < s.y1 ) // part of r strictly above s
   list = new RectList( new Rectangle(a, r.y1, b, s.y1), list);
 if ( r.y2 > s.y2 ) // part of r strictly below s
   list = new RectList( new Rectangle(a, s.y2, b, r.y2), list);
 return list;
}
public static void main( String args[] ) throws IOException {
 BufferedReader br = new BufferedReader( new InputStreamReader(System.in));
 String line;
 RectList disjoint = null;
 while ((line = br.readLine()) != null ) {
   StringTokenizer st = new StringTokenizer( line );
   int a = Integer.parseInt( st.nextToken()), b = Integer.parseInt( st.nextToken()),
       c = Integer.parseInt( st.nextToken()), d = Integer.parseInt( st.nextToken());
   disjoint = union( new Rectangle(Math.min(a,c), Math.min(b,d),
                              Math.max(a,c), Math.max(b,d)), disjoint);
 }
 int area = 0;
 for (RectList p = disjoint; p != null; p = p.tail )
   area += (p.head.x2 - p.head.x1) * (p.head.y2 - p.head.y1);
 System.out.println("The area is " + area );
 }
}
```

Triangles:

Triangle

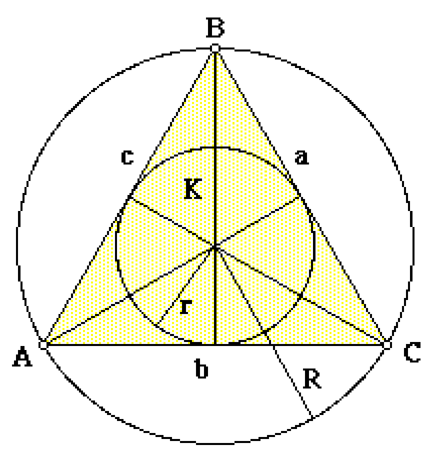
A polygon (plane figure) with 3 angles and 3 sides.

Sides: a, b, c
Opposite angles: A, B, C

Altitudes: h_a , h_b , h_c
Medians: m_a , m_b , m_c
Angle bisectors: t_a , t_b , t_c

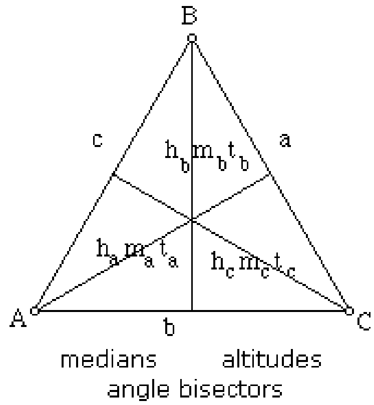
Perimeter: P
Semiperimeter: s
Area: K

Radius of circumscribed circle: R
Radius of inscribed circle: r



Equilateral Triangle

A triangle with all three sides of equal length.



$a = b = c.$
$A = B = C = Pi/3$ radians $= 60^o$

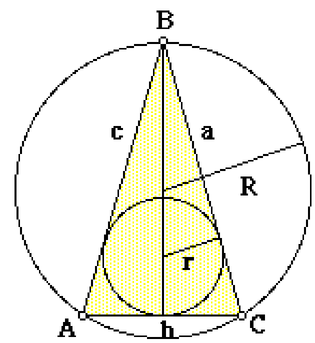
$P = 3a$
$s = 3a/2$
$K = a^2 sqrt(3)/4$

$h_a = m_a = t_a = a\ sqrt(3)/2$

$R = a\ sqrt(3)/3$
$r = a\ sqrt(3)/6$

Isosceles Triangle

A triangle with two sides of equal length.

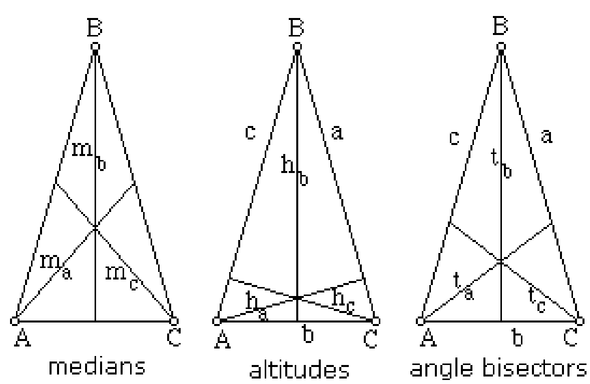


$a = c$
$A = C$

$B + 2A = Pi$ radians $= 180^o$
$P = 2a + b$
$s = a + b/2$
$K = b\ sqrt(4a^2\text{-}b^2)/4 = a^2\ sin(B)/2 = ab\ sin(A)/2$

$h_a = b\ sqrt(4a^2\text{-}b^2)/(2a) = a\ sin(B) = b\ sin(A)$
$m_a = sqrt(a^2+2b^2)/2$
$t_a = b\ sqrt(a[2a+b])/(a+b) = b\ sin(A)/sin(3A/2)$
$h_b = m_b = t_b = sqrt(4a^2\text{-}b^2)/2 = a\ cos(B/2)$

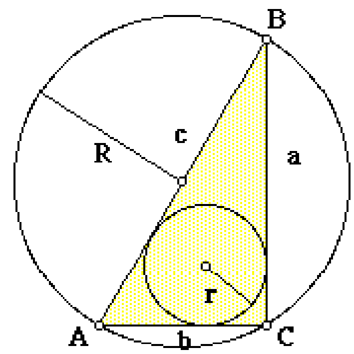
$R = a^2b/4K = a/[2\ sin(A)] = b/[2\ sin(B)]$
$r = K/s = b\ sqrt[(2a\text{-}b)/(2a+b)]/2$



A triangle with one right angle.

$C = A + B = Pi/2$ radians $= 90^o$

$c^2 = a^2 + b^2$
(Pythagorean Theorem)



$P = a + b + c$
$s = (a+b+c)/2$
$K = ab/2$

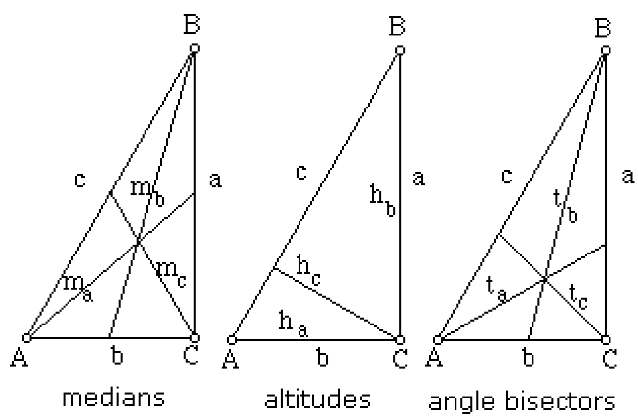
$h_a = b$
$h_b = a$
$h_c = ab/c$
$m_a = sqrt(4b^2+a^2)/2$
$m_b = sqrt(4a^2+b^2)/2$
$m_c = c/2$

$t_a = 2bc\ cos(A/2)/(b+c) =$
$sqrt[bc(1-a^2/[b+c]^2)]$

$t_b = 2ac\ cos(B/2)/(a+c) =$
$sqrt[ac(1-b^2/[a+c]^2)]$

$t_c = ab\ sqrt(2)/(a+b)$

$R = c/2$
$r = ab/(a+b+c) = s - c$



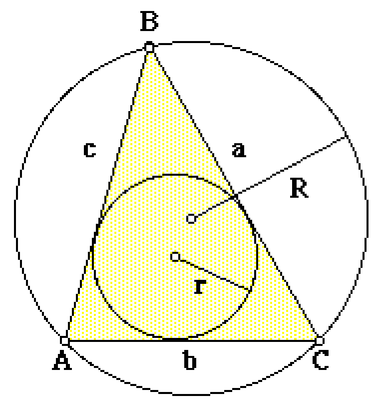
Rotate:
x' = cos(a) x – sin(a) y
y' = cos(a) x + sin(a) y
BFS:

```
private static void BFS(Node[] graph, Node start) {
        int i;
        Vector q = new Vector();
        Node u, v;

        for(i = 0 ; i < graph.length; i++) {
                graph[i].color = 0;
                graph[i].distance = -1;
                graph[i].pred = null;
        }
        start.color = 1;
        start.distance = 0;
        q.add(start);
```

```
        while(q.size() != 0) {
            u = (Node)q.elementAt(0);
            for(i = 0; i < u.adjList.size(); i++) {
                v = (Node)u.adjList.elementAt(i);
                if (0 == v.color) {
                    v.color = 1;
                    v.distance = u.distance+1;
                    v.pred = u;
                    q.add(v);
                }
            }
            q.removeElementAt(0);
            u.color = 2;
        }
    }
```

(Note that the following formulas work with all triangles, not just scalene triangles.)



$P = a + b + c$

$s = (a+b+c)/2$

$K = ah_a/2 = ab\ \sin(C)/2 =$
$\quad a^2 \sin(B) \sin(C)/[2 \sin(A)] =$
$\quad \mathrm{sqrt}[s(s-a)(s-b)(s-c)]$
(Heron's or Hero's Formula)

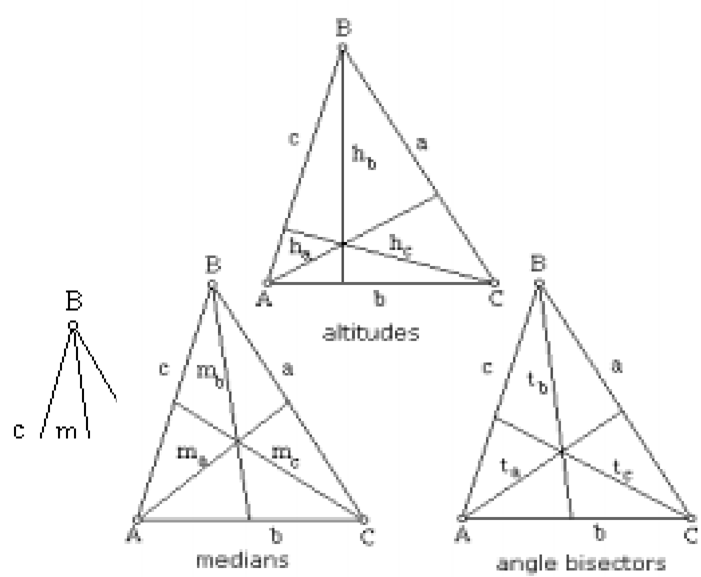
$h_a = c \sin(B) = b \sin(C) = 2K/a$

$m_a = \mathrm{sqrt}(2b^2+2c^2-a^2)/2$

$t_a = 2bc \cos(A/2)/(b+c) =$
$\quad \mathrm{sqrt}[bc(1-a^2/[b+c]^2)]$

$R = abc/4K =$
$\quad a/[2 \sin(A)] =$
$\quad b/[2 \sin(B)] =$
$\quad c/[2 \sin(C)]$

$r = 2K/P = K/s =$
$\quad \mathrm{sqrt}[(s-a)(s-b)(s-c)/s] =$
$\quad c \sin(A/2)\sin(B/2)/\cos(C/2) =$
$\quad ab \sin(C)/(2 s) =$
$\quad (s-c)\tan(C/2)$



Combination:

```
  static boolean nextCombination(int n, int k, int[] data) {
        boolean found = false;
        int i, j;
        if (data[0] == 0) {
           for(i = 0; i < k; i++) data[i] = i+1;
           return true;
        }
        for(i = 0; i < k; i++) {
           if ((i < k-1 && data[i] < (data[i+1]-1)) || (i == (k-1) && data[i] < n)) {
                data[i]++;
                found = true;
                break;
           }
        }
        if (found) {
           for(j = 0; j < i; j++) data[j] = j+1;
        }
        return found;
   }
```

Prime factorization:

```
static Vector getPrimes(int x) {
    Vector v = new Vector();
    int c = x;
    while((c%2)==0) {
       v.add(new Integer(2));
       c /= 2;
    }
    int i = 3;
    while(i <= (Math.sqrt(c)+1)) {
       if ((c%i)==0) {
          v.add(new Integer(i));
          c /= i;
       } else {
          i += 2;
       }
    }
    if (c > 1)
       v.add(new Integer(c));
    return v;
 }
```

Gauss:

```
class gauss {
        public static void main(String[] args) {
                double[][] a = {{1, 2, 3, 4}, {2, 3, 4, 6}, {3, 4, 2, 5}, {4, 6, 5, 7}};
                double[] b = {8, 6, 4, 2};
                double[] x = {0, 0, 0, 0};

                triangulate(a, b);
                backsusbst(a, x, b);

                for(int i= 0; i < a.length; i++) {
                        for(int j= 0; j < a.length; j++)
```

```
                        System.out.print(""+a[i][j] + " ");
                    System.out.println();
                }

                System.out.println();
                for(int i= 0; i < x.length; i++)
                    System.out.print(""+x[i] + " ");
                System.out.println();
            }
            static void triangulate(double[][] a, double[] b) {
                int n = a.length;
                for(int i = 0; i < n-1; i++) {
                    double diag = a[i][i];
                    for(int j = i+1; j < n; j++) {
                        double mult = a[j][i]/diag;
                        for(int k = 0; k < n; k++)
                            a[j][k] -= mult * a[i][k];

                        b[j] -= mult * b[i];
                    }
                }
            }

            static void backsusbst(double[][] a, double[] x, double[] b) {
                int n = a.length;
                for(int i = n-1; i >= 0; i--) {
                    double diag = a[i][i];
                    x[i] = (b[i] - scalar(a[i], x))/diag;
                }
            }
            static double scalar(double[] v1, double[] v2) {
                double t = 0;
                for(int i = 0; i < v1.length; i++)
                    t += v1[i] * v2[i];
                return t;
            }
}
public class Segment {
    public Point p1, p2, direction;
    public double mag;
    public Segment(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
        mag = Math.sqrt(
            Math.pow(p1.x-p2.x, 2) +
            Math.pow(p1.y-p2.y, 2) +
            Math.pow(p1.z-p2.z, 2)
        );

        direction = p2.subtract(p1);
    }
    public Point[] closestPoint(Segment s) {
        Point p13 = p1.subtract(s.p1);
        Point p43 = s.p2.subtract(s.p1);
        Point p21 = p2.subtract(p1);
```

```
        double d1343 = p13.x * p43.x + p13.y * p43.y + p13.z * p43.z;
        double d4321 = p43.x * p21.x + p43.y * p21.y + p43.z * p21.z;
        double d1321 = p13.x * p21.x + p13.y * p21.y + p13.z * p21.z;
        double d4343 = p43.x * p43.x + p43.y * p43.y + p43.z * p43.z;
        double d2121 = p21.x * p21.x + p21.y * p21.y + p21.z * p21.z;
        double denom = d2121 * d4343 - d4321 * d4321;
        double numer = d1343 * d4321 - d1321 * d4343;
        double mua = numer / denom;
        double mub = (d1343 + d4321 * mua) / d4343;
        Point pa = new Point(
            p1.x + mua * p21.x,
            p1.y + mua * p21.y,
            p1.z + mua * p21.z
        );
        Point pb = new Point(
            s.p1.x + mub * p43.x,
            s.p1.y + mub * p43.y,
            s.p1.z + mub * p43.z
        );

        Point[] p = {pa, pb};
        return p;
    }
    public Point intersect(Segment s) {
        Point[] intersect = closestPoint(s);
        if(!intersect[0].equals(intersect[1]))
            return Point.INVALID;
        double minx = Math.min(p1.x, p2.x);
        double maxx = Math.max(p1.x, p2.x);
        if(intersect[0].x < minx || intersect[0].x > maxx)
            return Point.INVALID;
        double miny = Math.min(p1.y, p2.y);
        double maxy = Math.max(p1.y, p2.y);
        if(intersect[0].y < miny || intersect[0].y > maxy)
            return Point.INVALID;
        double minz = Math.min(p1.z, p2.z);
        double maxz = Math.max(p1.z, p2.z);
        if(intersect[0].z < minz || intersect[0].z > maxz)
            return Point.INVALID;
        return intersect[0];
    }
}
class Line extends Object {
        public Line(Point p1, Point p2) {
                this.p1 = p1; this.p2 = p2;
                this.slope = getSlope(this);
                this.yIntercept = getYIntercept(this);
        }

        public Line(double slope, double yIntercept) {
                this.p1 = new Point(Double.NEGATIVE_INFINITY, Double.NEGATIVE_INFINITY);
                this.p2 = new Point(Double.POSITIVE_INFINITY, Double.POSITIVE_INFINITY);
                this.slope = new Double(slope);
                this.yIntercept = new Double(yIntercept);
        }
```

```
public static Double getSlope(Line line) {
		if(line.p2.x - line.p1.x == 0)
			return new Double(Double.POSITIVE_INFINITY);
		else
			return new Double(
				(line.p2.y - line.p1.y) / (line.p2.x - line.p1.x)
			);
}
public static Double getYIntercept(Line line) {
		if(line.slope.doubleValue() == 0.0)
			return new Double(line.p1.y);
		else if(line.slope.doubleValue() == Double.POSITIVE_INFINITY)
			return new Double(Double.NaN);
		else
			return new Double(
				line.p1.y - line.slope.doubleValue() * line.p1.x
			);
}
public Point intersection(Line line) {
		Double yIntersection = new Double(0);
		Double xIntersection = new Double(0);
		if(this.slope.doubleValue() == Double.POSITIVE_INFINITY) {
			if(line.slope.doubleValue() == Double.POSITIVE_INFINITY)
				return null;
			xIntersection = new Double(this.p1.x);
			if(line.slope.doubleValue() == 0.0)
				yIntersection = new Double(this.p1.x);
			else
				yIntersection = new Double(
					line.slope.doubleValue() * xIntersection.doubleValue() +
					line.yIntercept.doubleValue()
				);
		} else if(this.slope.doubleValue() == 0.0) {
			if(line.slope.doubleValue() == 0.0)
				return null;
			yIntersection = new Double(this.p1.y);
			if(line.slope.doubleValue() == Double.POSITIVE_INFINITY)
				xIntersection = new Double(line.p1.x);
			else
				xIntersection = new Double(
					1 / line.slope.doubleValue() * (yIntersection.doubleValue()
					- line.yIntercept.doubleValue())
				);
		} else {
			xIntersection = new Double(
				(this.yIntercept.doubleValue() - line.yIntercept.doubleValue())
				/ (line.slope.doubleValue() - this.slope.doubleValue())
			);
			yIntersection = new Double(
				this.slope.doubleValue() * xIntersection.doubleValue() +
				this.yIntercept.doubleValue()
			);
		}
		if(this.p1.x < this.p2.x) {
			if(xIntersection.doubleValue() < this.p1.x) return null;
			if(xIntersection.doubleValue() > this.p2.x) return null;
```

```
		} else {
			if(xIntersection.doubleValue() < this.p2.x) return null;
			if(xIntersection.doubleValue() > this.p1.x) return null;
		}
		if(this.p1.y < this.p2.y) {
			if(yIntersection.doubleValue() < this.p1.y) return null;
			if(yIntersection.doubleValue() > this.p2.y) return null;
		} else {
			if(yIntersection.doubleValue() < this.p2.y) return null;
			if(yIntersection.doubleValue() > this.p1.y) return null;
		}
		if(xIntersection.doubleValue() == Double.NaN ||
		  yIntersection.doubleValue() == Double.NaN)
			return null;
		return new Point(
			xIntersection.doubleValue(), yIntersection.doubleValue()
		);
	}
	public String toString() {
		return new String("Point 1: " + p1 + ",  Point 2: " + p2);
	}
	public Point p1, p2;
	public Double slope, yIntercept;
}
class Point extends Object {
	public Point(double x, double y) {
		this.x = x;		this.y = y;
	}
	public String toString() {
		return new String("(" + this.x + ", " + this.y + ")");
	}
	public double x, y;
}
public class Point {
   public int dim;
   public double x, y, z;

   public static Point INVALID = new Point(
      Double.NaN, Double.NaN, Double.NaN
   );

   public Point(int x, int y)
      {this((double) x, (double) y);}

   public Point(int x, int y, int z)
      {this((double) x, (double) y, (double) z);}

   public Point(double x, double y) {
      this.x = x;
      this.y = y;
      dim = 2;
   }

   public Point(double x, double y, double z) {
      this.x = x;
      this.y = y;
```

```
        this.z = z;
        dim = 3;
    }

    public boolean equals(Point p)
        {return x == p.x && y == p.y && z == p.z && dim == p.dim;}

    public boolean isInvalid()
        {return isInvalid(this);}

    public static boolean isInvalid(Point p)
        {return Double.isNaN(p.x) && Double.isNaN(p.y) && Double.isNaN(p.z);}

    public static boolean notZero(Point p)
        {return p.x != 0 && p.y != 0 && p.z != 0;}

    public double dist2(Point p) {
        double dx = x-p.x;
        double dy = y-p.y;
        double dz = z-p.z;
        return dx*dx + dy*dy + dz*dz;
    }

    public double magnitude()
        {return Math.sqrt(x*x + y*y + z*z);}

    public double dist(Point p)
        {return Math.sqrt(dist2(p));}

    public double isLeft(Point p1, Point p2)
        {return (x-p1.x)*(y-p2.y) - (x-p2.x)*(y-p1.y);}

    public Point subtract(Point p)
        {return new Point(x-p.x, y-p.y, z-p.z);}

    public Point divide(double n) {
        if(n == 0) return new Point(0, 0, 0);
        return new Point(x/n, y/n, z/n);
    }

    public Point crossProd(Point p) {
        return new Point(
            y*p.z - z*p.y,
            z*p.x - x*p.z,
            x*p.y - y*p.x
        );
    }

    public String toString()
        {return "(" + x + "," + y + (dim == 3 ? "," + z : "") + ")";}
}
```