TÖL101G - Tölvunarfræði 1

Vikublað 9

Lausn

Æfingar

3.2.8

```
public class Interval {
    private final double left,right;
    // táknar bilið [left,right] ef left <= right, annars tóma mengið

    // Notkun:  a = new Interval(left,right)
    // Fyrir:   ekkert
    // Eftir:   a er bilið [left,right] ef left <= right, annars tóma mengið
    public Interval(double left, double right) {
        this.left = left;
        this.right = right;
    }

    // Notkun: c = a.contains(x)
    // Fyrir:  ekkert
    // Eftir:  c er satt ef x er í bilinu a
    public boolean contains(double x) {
        return (this.left <= x) && (x <= this.right);
    }

    // Notkun: c = a.intersects(b)
    // Fyrir:  ekkert
    // Eftir:  c er satt ef a og b innihalda sameiginlegan punkt
    public boolean intersects(Interval b) {
        return this.proper() && b.proper() &&
            (this.left <= b.right) && (this.right >= b.left);
    }

    public String toString() {
        if (this.proper()) {
            return "["+this.left+","+this.right+"]";
        } else {
            return "[]";
        }
    }

    // Notkun: c = a.contains(b)
    // Fyrir:  ekkert
    // Eftir:  c er satt ef a inniheldur bilið b eða b er tómt
    public boolean contains(Interval b) {
```

```
        return !b.proper() || this.contains(b.left) && this.contains(b.right);
    }

    // Notkun: c = a.proper()
    // Fyrir:  ekkert
    // Eftir:  c er true ef bilið a er ekki tómt
    private boolean proper() {
        return this.left <= this.right;
    }
}
```

3.2.10

```
public class Rectangle  {
    private final double x,y,h,w;
    // x,y eru hnitin fyrir miðjuna
    // hæðin er h > 0,  breiddin er w > 0

    // Notkun: r = new Rectangle(x,y,w,h)
    // Fyrir:  w > 0, h > 0
    // Eftir:  r er ferhyrningur með miðju í x,y, hæð h og breidd w
    public Rectangle(double x, double y, double w, double h) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }

    // Notkun: x = a.area()
    // Fyrir:  ekkert
    // Eftir:  x er flatarmál a
    public double area() {
        return w*h;
    }

    // Notkun: x = a.perimeter()
    // Fyrir:  ekkert
    // Eftir:  x er ummál a
    public double perimeter() {
        return 2.0*(w+h);
    }

    // Notkun: c = a.intersects(b)
    // Fyrir:  ekkert
    // Eftir:  c er true ef a og b skarast
    public boolean intersects(Rectangle b) {
        // a og b skarast ef x og y bilin skarast
        boolean r = true;
```

```
        r = r && this.getXInterval().intersects(b.getXInterval());
        r = r && this.getYInterval().intersects(b.getYInterval());
        return r;
    }

    // Notkun: c = a.contains(b)
    // Fyrir:  ekkert
    // Eftir:  c er true ef b er inni í a
    public boolean contains(Rectangle b) {
        boolean r = true;
        r = r && this.getXInterval().contains(b.getXInterval());
        r = r && this.getYInterval().contains(b.getYInterval());
        return r;
    }


    public String toString() {
        return "(x,y) = (" + this.x+","+this.y+"), w="+this.w+", h="+this.h;
    }

    // Notkun: xb = a.getXInterval()
    // Fyrir:  ekkert
    // Eftir:  xb er bilið fyrir x-ásinn
    private Interval getXInterval() {
        return new Interval(x-w/2,x+w/2);
    }

    // Notkun: yb = a.getYInterval()
    // Fyrir:  ekkert
    // Eftir:  yb er bilið fyrir y-ásinn
    private Interval getYInterval() {
        return new Interval(y-h/2,y+h/2);
    }
}
```

3.2.13

Stopwatch klasinn er ekki nógu nákvæmur til að mæla tímamismuninn fyrir eina keyrslu, þar sem þetta tekur ca. 0.0 sek per aðgerð. Til að fá áreiðanlegar tölur er hægt að endurtaka aðgerðina 100000 sinnum og reikna út tíma fyrir hverja aðgerð. Þetta er gert í lausninni að neðan og niðurstöðurnar eru eftirfarandi.

```
Recursive: Time per operation for N = 10: 1.97E-7
Serial: Time per operation for N = 10: 1.5E-8
N = 10, serial is 13.133333333333335 times faster than recursive
Recursive: Time per operation for N = 100: 2.267E-6
Serial: Time per operation for N = 100: 0.0
N = 100, serial is Infinity times faster than recursive
Recursive: Time per operation for N = 1000: 2.3465E-5
```

```
Serial: Time per operation for N = 1000: 0.0
N = 1000, serial is Infinity times faster than recursive
Not measuring recursive solution
Serial: Time per operation for N = 10000: 0.0
Not measuring recursive solution
Serial: Time per operation for N = 100000: 1.0E-9
```

Þegar N er nógu stórt (5000-6000, fer eftir uppsetningu Java) fáum við `StackOverflowError` þegar það eru of mörg endurkvæm köll. Við endurkvæmni þarf að passa að ekki sé kallað of oft á fallið endurkvæmt.

```
public class LogFactMeasure {
    public static void main(String[] args) {

        int[] Ns = {10, 100, 1000, 10000, 100000};
        int reps = Integer.parseInt(args[0]);
        for (int i = 0; i < Ns.length; i++) {
            int N = Ns[i];
            Stopwatch s;
            double t1 = 0.0,t2 = 0.0;
            if (N < 5000) { // else stackoverflow!
                s = new Stopwatch();
                for (int j = 0; j < reps; j++) {
                    double x = LogFact.logFact(N);
                }
                t1 = s.elapsedTime();
                System.out.println("Recursive: Time per operation for + N = "
                                   + N +": " + t1/((double) reps));
            } else {
                System.out.println("Not measuring recursive solution");
            }

            s = new Stopwatch();
            for (int j = 0; j < reps; j++) {
                double x = LogFact2.logFact(N);
            }
            t2 = s.elapsedTime();
            System.out.println("Serial: Time per operation for N = "
                               + N +": " + t2/((double) reps));

            if (N <  5000) {
                System.out.println("N = " + N + ", serial is "
                                   + t1/t2 + " times faster than recursive");
            }
        }
    }
}
```

Verkefni

Í lausninni á verkefninu er mikilvægast að negla niður hvaða tilviksbreytur eru notaðar og hvað þær þýða með fastayrðingu gagna. Fyrir neðan eru tvær lausnir á verkefninu, annars vegar með því að nota $y = mx + c$ framsetningu á línu og hins vegar með jöfnunni $mx + nx = c$.

Í báðum tilfellum þarf að leysa jöfnur með 2 óþekktum stærðum, þetta er einfaldað fyrst á blaði og skrifað síðan beint í kóðanum. Til að bera saman fleytitölur sem geta verið misnákvæmar þá skilgreinum við fastann $\varepsilon = 10^{-15}$ sem `private static final double` (static þýðir að það er ein breyta sameiginleg öllum hlutum) og við segjum að $x \approx y$ ef $|x - y| < \varepsilon$, eða `Math.abs(x-y) < eps`.

Lausn 1

Hér notum við framsetninguna $y = mx + c$ fyrir línu þar sem m er hallatalan og c er skurðpunktur og allir punktar (x, y) á línunni uppfylla jöfnuna. Því miður er ekki hægt að lýsa lóðréttum línum með þessari jöfnu, fyrir þau tilvik notum við jöfnuna $x = d$, þ.e. allir punktar (x, y) sem uppfylla $x = d$ eru á línunni. Til að ná þessu fram notum við `boolean` tilviksbreytu til að halda utan um hvort línan sé lóðrétt eða ekki. Þessi ákvörðun flækir því miður allan kóða.

```
public class Line2D {
    // Tilviksbreytur
    private final double m,c,d;
    private final boolean vertical;
    // ef this.vertical er true er línan lóðrétt með x-hnit d
    // annars er hún með hallatölu m og skurðpunkt c

    private final static double eps = 1e-15;
    // notað fyrir nákvæmni

    // Notkun: x = new Line2D(a,b)
    // Fyrir:  a og b eru ekki sami punkturinn
    // Eftir:  x er línan sem fer í gegnum a og b
    public Line2D(Point2D a, Point2D b) {
        if (a.getX() == b.getX()) {
            this.vertical = true;
            this.d = a.getX();
            this.m = Double.NaN;
            this.c = Double.NaN;
        } else {
            this.vertical = false;
            this.m = (b.getY()-a.getY())/(b.getX()-a.getX());
            this.c = a.getY() - m*a.getX();
            this.d = Double.NaN;
        }
    }

    // Notkun: c = a.intersects(b)
```

```
// Fyrir:  ekkert
// Eftir:  c er true ef punkturinn b liggur á
//         línunni a, false annars
public boolean intersects(Point2D b) {
    if (this.vertical) {
        return Math.abs(b.getX() - this.d) < this.eps;
    } else {
        return Math.abs(this.m * b.getX() + this.c -b.getY()) < this.eps;
    }
}

// Notkun: c = a.isParallel(b)
// Fyrir:  ekkert
// Eftir:  c er true ef a og b eru samsíða, false annars
public boolean isParallel(Line2D b) {
    if (this.vertical) {
        return b.vertical;
    } else {
        return Math.abs(this.m-b.m) < this.eps;
    }
}

// Notkun: x = a.intersection(b)
// Fyrir:  a.isParallel(b) == false
// Eftir:  x er nýr punktur þar sem línurnar a og b skerast
public Point2D intersection(Line2D b) {
    double x,y;
    if (this.vertical) {
        x = this.d;
        y = b.m*x+b.c; // b er ekki lóðrétt
    } else if (b.vertical) {
        x = b.d;
        y = this.m*x+this.c;
    } else {
        y = (b.m*this.c - this.m*b.c)/(b.m-this.m);
        x = -(b.c-this.c)/(b.m-this.m);
    }
    return new Point2D(x,y);
}

// Notkun: c = a.equals(b)
// Fyrir:  ekkert
// Eftir:  c er true ef að a og b eru sama línan
public boolean equals(Line2D b) {
    if (this.vertical) {
        if (b.vertical) {
            return Math.abs(this.d - b.d) < this.eps;
        } else {
```

```
                return false;
            }
        } else {
            return  (Math.abs(this.m-b.m) < eps)
                &&  (Math.abs(this.c-b.c) < eps);
        }
    }
}
```

Lausn 2

Hér notum við framsetninguna $mx + ny = c$ þar sem m og n eru ekki bæði 0. Kóðinn verður einfaldari en það þarf oftar að leysa jöfnuhneppi. Þar sem margar (m, n, c) þrenndir lýsa línunni fullkomlega t.d. $(2m, 2n, 2c)$ lýsir sömu línu og (m, n, c) þá skölum við niður þannig að $m^2 + n^2 = 1$. Þetta er reyndar ekki nóg þar sem $(-m, -n, -c)$ lýsir sömu línu, því bætum við auka skilyrði að $n \geq 0$ og ef $n = 0$ þá er $m = 1$. Þetta er gert í smiðnum og tekið fram í fastayrðingu gagna.

```
public class Line2D{
    // Tilviksbreytur
    private final double m,n,c;
    // línan er safn allra punkta x,y sem uppfylla
    // mx+nx = c og þar að auki er m^2+n^2=1.0, n >=0
    // og ef n=0 þá er m=1.

    private static final double eps = 1e-15;
    // notað fyrir nákvæmni

    // Notkun: x = new Line2D(a,b)
    // Fyrir:  a og b eru ekki sami punkturinn
    // Eftir:  x er línan sem fer í gegnum a og b
    public Line2D(Point2D a, Point2D b) {
        double m0,n0,c0;

        if (b.getX()==a.getX()) {
            // vertical line
            m0 = 1.0;
            n0 = 0.0;
            c0 = a.getX();
        } else {
            // now n != 0.0, so we can rescale
            // n = 1.0, then m0*x+y = c
            n0 = 1.0;
            m0 = -(b.getY()-a.getY())/(b.getX()-a.getX());
            c0 = m0*a.getX() + a.getY();
        }

        double scale = Math.sqrt(m0*m0+n0*n0);
        this.m = m0/scale;
```

```
        this.n = n0/scale;
        this.c = c0/scale;
    }

    // Notkun: c = a.intersects(b)
    // Fyrir:  ekkert
    // Eftir:  c er true ef punkturinn b liggur á
    //         línunni a, false annars
    public boolean intersects(Point2D b) {
        return Math.abs(this.m*b.getX() + this.n*b.getY() - this.c) < eps;
    }

    // Notkun: c = a.isParallel(b)
    // Fyrir:  ekkert
    // Eftir:  c er true ef a og b eru samsíða, false annars
    public boolean isParallel(Line2D b) {
        return Math.abs(b.m-this.m) < eps
            && Math.abs(b.n-this.n) < eps;
    }

    // Notkun: x = a.intersection(b)
    // Fyrir:  a.isParallel(b) == false
    // Eftir:  x er nýr punktur þar sem línurnar a og b skerast
    public Point2D intersection(Line2D b) {

        double det = this.m*b.n - this.n*b.m;
        double x = (b.n*this.c - this.n*b.c)/det;
        double y = (-b.m*this.c + this.m*b.c)/det;

        return new Point2D(x,y);
    }

    // Notkun: c = a.equals(b)
    // Fyrir:  ekkert
    // Eftir:  c er true ef að a og b eru sama línan
    public boolean equals(Line2D b) {
        return this.isParallel(b) && (Math.abs(b.c-this.c) < eps);
    }
}
```