

TDA 545: Objektorienterad programmering

Föreläsning 11:
Rekursion

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

Idag

Läsanvisning: kap 19, men bara t.o.m. sida 812

- ▶ rekursion
 - ▶ fakteteten
 - ▶ exponentiering
 - ▶ binary search
 - ▶ Fibonacci
 - ▶ 91 funktionen

men också:

- ▶ jämförelse med iteration
- ▶ att mäta tid i Java
- ▶ roliga problem

Nästa gång: problemlösning

Att tänka rekursivt...

Hur långt är tåget?

Rekursivt svar:

om det inte finns tågvarnar: då är dess **längd 0**

i övriga fall: **längden är 1 + längden av resten av tåget**

samma fråga som ursprungligen,
men med mindre indata (input)





metro

Woman spotted yesterday reading today's paper

Worth 1000 "yeww" says president



Fakulteten

Iterativt

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times 3 \times \dots \times n & \text{if } 1 \leq n \end{cases}$$

```
public static int facIt(int n) {  
    int v = 1;  
    for (int i=1; i<=n; i++) {  
        v = v * i;  
    }  
    return v;  
}
```

Rekursivt

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n - 1)! & \text{if } 1 \leq n \end{cases}$$

Induktiva definitioner

använder självreferens, t.ex.

En kö är antingen tom
eller en person följt av en kö.

$n!$ är 1, om n är 0 eller 1,
eller $n \times (n-1)!$ om n är större.

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n-1)! & \text{if } 1 \leq n \end{cases}$$

Induktiva definitioner består av

1. Ett basfall
2. Ett fall där definitionen (vanligen) minskar problemets storlek och refererar till sig själv.

Induktionsbevis och rekursion

Induction poof:

to prove that a statement $P(n)$ holds for all integers n

Base case: Prove $P(0)$ (or $P(\text{small number})$)

Assumption:

assume that $P(n-1)$ is true (or $P(n), P(n/2)\dots$ is)

Induction step:

prove that $P(n-1)$ implies $P(n)$ for all $n \geq 0$

Recursive computations:

Base case: Solve the problem on inputs of size 0 (or 1 or some small instance)

Assumption: assume you can solve the problem on input size less than n ($n-1, n/2, n/4\dots$)

Induction step: Show how you can solve it also on inputs of size n for all $n \geq 1$.

Fakulteten (forts)

Iterativt

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times 3 \times \dots \times n & \text{if } 1 \leq n \end{cases}$$

```
public static int facIt(int n) {  
    int v = 1;  
    for (int i=1; i<=n; i++) {  
        v = v * i;  
    }  
    return v;  
}
```

Rekursivt

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n - 1)! & \text{if } 1 \leq n \end{cases}$$

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluating

```
System.out.println(fac(4));
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluating

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluating

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```


Evaluating

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluating

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```



```
if (1 < 1) {  
    return 1;  
} else {  
    return 1 * fac(0);  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluating

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```



```
if (1 < 1) {  
    return 1;  
} else {  
    return 1 * fac(0);  
}
```



1

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```



```
if (1 < 1) {  
    return 1;  
} else {  
    return 1 * fac(0);  
}
```



1

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen går ner

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```



```
if (1 < 1) {  
    return 1;  
} else {  
    return 1 * fac(0);  
}
```



1

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * fac(1);  
}
```



```
if (1 < 1) {  
    return 1;  
} else {  
    return 1 * 1;  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * fac(2);  
}
```



```
if (2 < 1) {  
    return 1;  
} else {  
    return 2 * 1;  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * fac(3);  
}
```



```
if (3 < 1) {  
    return 1;  
} else {  
    return 3 * 2;  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Evaluering

```
System.out.println(fac(4));
```



```
if (4 < 1) {  
    return 1;  
} else {  
    return 4 * 6;  
}
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Evaluering

```
System.out.println( 24 );
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

rekursionen kommer upp

Sammanfattning av beräkningen

```
System.out.println(fac(4));
```

```
fac(4) =  
4 * fac(3) =  
4 * 3 * fac(2) =  
4 * 3 * 2 * fac(1) =  
4 * 3 * 2 * 1 * fac(0) =  
4 * 3 * 2 * 1 * 1 =  
4 * 3 * 2 * 1 =  
4 * 3 * 2 =  
4 * 6 =  
24
```

```
public static int fac(int n) {  
    if (n < 1) {  
        return 1;  
    } else {  
        return n * fac(n-1);  
    }  
}
```

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n - 1)! & \text{if } 1 \leq n \end{cases}$$

Ett test program

```
public class Fac {  
  
    public static int fac(int n) {  
        if (n == 0) {  
            return 1;  
        } else {  
            return n * fac(n-1) ;  
        }  
    }  
}  
  
public static void main(String[] args) {  
    System.out.println("fac(1)  = " + fac(1));  
    System.out.println("fac(4)  = " + fac(4));  
    System.out.println("fac(20) = " + fac(20));  
}  
}
```

Utskrift:

```
fac(1)  = 1  
fac(4)  = 24  
fac(20) = -2102132736
```

umm... Vad hände här?

svar: värdet blev för stort.

Obs. overflow har inget med rekursionen att göra.

Vi kan använda BigInteger

```
import java.math.BigInteger;
```

```
public class BigFac {
```

```
    public static BigInteger fac(BigInteger n) {  
        if (n.equals(BigInteger.ZERO)) {  
            return BigInteger.ONE;  
        } else {  
            return n.multiply(fac(n.subtract(BigInteger.ONE)));  
        }  
    }  
}
```

```
    public static void main(String[] args) {  
        System.out.println("fac(1) = " + fac(new BigInteger("1")));  
        System.out.println("fac(4) = " + fac(new BigInteger("4")));  
        System.out.println("fac(20) = " + fac(new BigInteger("20")));  
        System.out.println("fac(50) = " + fac(new BigInteger("50")));  
    }  
}
```

Utskrift:

```
fac(1) = 1  
fac(4) = 24  
fac(20) = 2432902008176640000  
fac(50) = 30414093201713378043612608166064768844377641568960512000000000000
```

Uppgift

Skriv ett program som beräknar **hur många nollor** det finns i ett decimal tal.

Idé: I ett **negativ tal** finns det like många nollor som i motsvarande **positiva tal**.

Om talet är **0**, finns det **en nolla** i talet.

Om talet är **positivt och < 10** , finns det **ingen nolla**.

I övriga fall, dvs. **positivt och > 10**

om talet **mod 10 är 0** , då finns det **en nolla + nollorna i talet / 10**

annars: **nollorna i talet / 10**

Lösning

```
public class Z {  
  
    public static int k(int n) {  
        if (n < 0) {  
            return k(0 - n);  
        } else if (n == 0) {  
            return 1;  
        } else if (n < 10) {  
            return 0;  
        } else if (n % 10 == 0) {  
            return 1 + k(n / 10);  
        } else {  
            return k(n / 10);  
        }  
    }  
}  
  
public static void main(String[] args) {  
    System.out.println("nollor(-1050) = " + k(-1050));  
}  
}
```

Uppgift

Skriv en *rekursiv* metod som vänder om en sträng, dvs `rev("Hello")` bör bli `"olleH"`.

```
public static String rev(String str) {
    if (str.length() == 0) {
        return str;
    } else {
        return rev(str.substring(1)) + str.substring(0,1);
    }
}
```

Ofta går det att skriva om rekursionen som en iterativa metod ... *men inte alltid*.

Några animationer

<http://impedagogy.com/wp/wp-content/uploads/2014/08/Recursionelevators.gif>

<http://i.imgur.com/P1Chi6u.gif>

<http://media.giphy.com/media/VLNG6LOKeURfG/giphy.gif>

Roligt med rekursion...

Vad räknar McCarthy's funktion m ?

$$m(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ m(m(n + 11)) & \text{if } n \leq 100 \end{cases}$$

Samma i Java:

```
public static int m(int n) {
    if (n > 100) {
        return n - 10;
    } else {
        return m(m(n + 11));
    }
}
```

McCarthy's funktion

```
public class M {  
  
    public static int m(int n) {  
        if (n > 100) {  
            return n - 10;  
        } else {  
            return m(m(n + 11));  
        }  
    }  
  
    public static void main(String[] args) {  
        for (int j=0; j<100; j++) {  
            System.out.println("m(" + j + ") = " + m(j));  
        }  
    }  
}
```

Detta är McCarthy's **91 funktion**.

http://en.wikipedia.org/wiki/McCarthy_91_function

Varför blir det 91?

Utskrift:

```
m(0) = 91  
m(1) = 91  
m(2) = 91  
m(3) = 91  
m(4) = 91  
m(5) = 91  
m(6) = 91  
m(7) = 91  
m(8) = 91  
m(9) = 91  
m(10) = 91  
m(11) = 91  
m(12) = 91  
m(13) = 91  
m(14) = 91  
m(15) = 91  
m(16) = 91  
m(17) = 91  
m(18) = 91  
...  
m(99) = 91
```

Slutar beräkningen?

Ibland blir input större inför rekursionen.

$$t(n) = \begin{cases} 1 & \text{if } n = 1 \\ t(3 \times n + 1) & \text{if } n \text{ is odd} \\ t(n/2) & \text{if } n \text{ is even} \end{cases}$$

Samma i Java:

```
public class T {
    public static int t(int n) {
        System.out.print(n + " ");
        if (n < 1) { System.exit(1); }
        if (n == 1) {
            return 1;
        } else if (n % 2 == 1) {
            return t(3 * n + 1);
        } else {
            return t(n / 2);
        }
    }
    public static void main (String[] args) {
        t(Integer.parseInt(args[0]));
    }
}
```

Utskrift:

```
$ javac T.java
$ java T 125
125 376 188 94 47 142 71 214 107
322 161 484 242 121 364 182 91
274 137 412 206 103 310 155 466
395
58
566
283 850 425 1276 638 199 958 479
1438 719 2158 1079 3238 1619 4858
2429 7288 3644 1822 911 2734 1367
4102 2051 6154 3077 9232 4616
2308 1154 577 1732 866 433 1300
650 325 976 488 244 122 61 184 92
46 23 70 35 106 53 160 80 40 20
10 5 16 8 4 2 1
```

Koden var i upp vid 9232!

Slutar beräkningen?

Ibland blir input större inför rekursionen.

$$t(n) = \begin{cases} 1 & \text{if } n = 1 \\ t(3 \times n + 1) & \text{if } n \text{ is odd} \\ t(n/2) & \text{if } n \text{ is even} \end{cases}$$

Slutar beräkningen för alla utgångsvärden av n ?

Svar: ingen vet! Det är en *open research question*...

Vi kan testa litegrann... men hur vet vi att räkningen slutar för *alla* input?
(Bör göras med BigInteger.)

Flera animationer

<https://i.chzbgr.com/maxW500/7732713984/h2E032A91/>

<http://d5lx5634mkgoi.cloudfront.net/wp-content/uploads/2010/02/Slide190.gif>

http://33.media.tumblr.com/142251408fcoa2446612ef96e7cffbe7/tumblr_moltlz3OwO1q

Lite mera seriöst igen...

Beräkna exp

Att beräkna x^n :

Iterativt, idé $x^n = x \times x \times \dots \times x$

```
public static double power1(double x, int n) {  
    double tmp = 1.0;  
    for (int i=1; i<=n; i++) {  
        tmp = tmp * x;  
    }  
    return tmp;  
}
```

multiplicerar x in i tmp, n gånger.

Rekursivt, idé $x^n = x \times x^{(n-1)}$

```
public static double power2(double x, int n) {  
    if (n == 0) {  
        return 1.0;  
    } else {  
        return x * power2(x,n-1);  
    }  
}
```

... gör samma sak.

Samma algoritm!

Beräkna exp (forts)

Bättre rekursion?

Obs: $x^n = x^{n/2} \times x^{n/2}$ **eller ännu bättre:** $x^n = (x \times x)^{n/2}$

Mera precis:

$$\begin{aligned}x^{2 \times n} &= (x \times x)^n \\x^{2 \times n + 1} &= (x \times x)^n \times x\end{aligned}$$

I Java:

```
public static double power3(double x, int n) {
    if (n == 0) {
        return 1.0;
    } else {
        if (n % 2 == 0) {
            return power3(x * x, n / 2);
        } else {
            return power3(x * x, n / 2) * x;
        }
    }
}
```

heltalsdivision,
dvs blir heltal

Jämförelse

$\text{power2}(9, 3.0) =$
 $3.0 * \text{power2}(8, 3.0) =$
 $3.0 * 3.0 * \text{power2}(7, 3.0) =$
 $3.0 * 3.0 * 3.0 * \text{power2}(6, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(5, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(4, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(3, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(2, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(1, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * \text{power2}(0, 3.0) =$
 $3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 3.0 * 1.0 =$
19683.0

$\text{power3}(9, 3.0) =$
 $\text{power3}(4, 9.0) * 3.0 =$
 $\text{power3}(2, 81.0) * 3.0 =$
 $\text{power3}(1, 6561.0) * 3.0 =$
 $\text{power3}(0, 6561.0) * 6561.0 * 3.0 =$
 $1.0 * 6561.0 * 3.0 =$
19683.0

bättre algoritm!

... men vilken tar längre tid att beräkna?

Att mäta tid i Java

I `java.lang.System` finns metoder:

```
static long currentTimeMillis()
```

Returns the current time in milliseconds.

```
static long nanoTime()
```

Returns the current value of the *most precise available* system timer, in nanoseconds.

Exempel användning:

```
long start = System.currentTimeMillis();  
gör arbete  
long duration = System.currentTimeMillis() - start;  
System.out.println("Duration: " + duration + " ms");
```

Att mäta tid på power funktionerna

```
public static void main(String[] args) {  
  
    int reps = 10000;  
    long start, duration;  
  
    // test power1  
    start = System.currentTimeMillis();  
    for (int i = 0; i < reps; i++) {  
        power1(2.0, 5000);  
    }  
    duration = System.currentTimeMillis() - start;  
    System.out.println("Duration1: " + duration + " ms");  
  
    // test power3  
    start = System.currentTimeMillis();  
    for (int i = 0; i < reps; i++) {  
        power3(2.0, 5000);  
    }  
    duration = System.currentTimeMillis() - start;  
    System.out.println("Duration3: " + duration + " ms");  
  
}
```

```
$ java Pow  
Duration1: 77 ms  
Duration3: 2 ms
```

Hitta element i array

Hur kollar man om ett element finns i en array?

Enkel kod:

```
public static boolean lookup(int k,int[] arr) {  
    for (int i=0; i<arr.length; i++) {  
        if (arr[i] == k) {  
            return true;  
        }  
    }  
    return false;  
}
```

... om arrayn sorterad går det att göra **mycket snabbare!**

Binärsökning

Exempel: Kolla om värdet 5 finns i arrayn?

0 | 1 | 2 | 5 | 8 | 11 | 12 | 14 | 16 | 18 | 22 | 24

Idé: Inspektera elementet i mitten av arrayn:

0 | 1 | 2 | 5 | 8 | 11 | 12 | 14 | 16 | 18 | 22 | 24

Är mitten elementet det vi söker?

Om mitten elementet är mindre kan vi “kasta bort” den högra sidan.

0 | 1 | 2 | 5 | 8 | 11

vi vet ju att arrayn är sorterad.

... och så kan vi upprepa sägen ovan tills arrayn är tom.

Binärsökning (forts)

Som Java kod:

```
public static boolean bsearch1(int k,int[] arr,int b,int e) {
    // System.out.println("b,e = " + b + ", " + e);
    if (e <= b) { return false; }
    int i = (b + e) / 2; // mitten
    if (arr[i] == k) {
        return true;
    } else if (arr[i] < k) {
        return bsearch1(k,arr,i+1,e);
    } else /* k < arr[i] */ {
        return bsearch1(k,arr,b,i);
    }
}

public static boolean bsearch(int k,int[] arr) {
    return bsearch1(k,arr,0,arr.length);
}
```

Binärsökning (forts)

Samma iterativt:

```
public static boolean bsearchIt(int k,int[] arr) {
    int b = 0;
    int e = arr.length;
    while (b < e) {
        int i = (b + e) / 2; // mitten
        if (arr[i] == k) {
            return true;
        } else if (arr[i] < k) {
            b = i+1;
        } else /* k < arr[i] */ {
            e = i;
        }
    }
    return false;
}
```

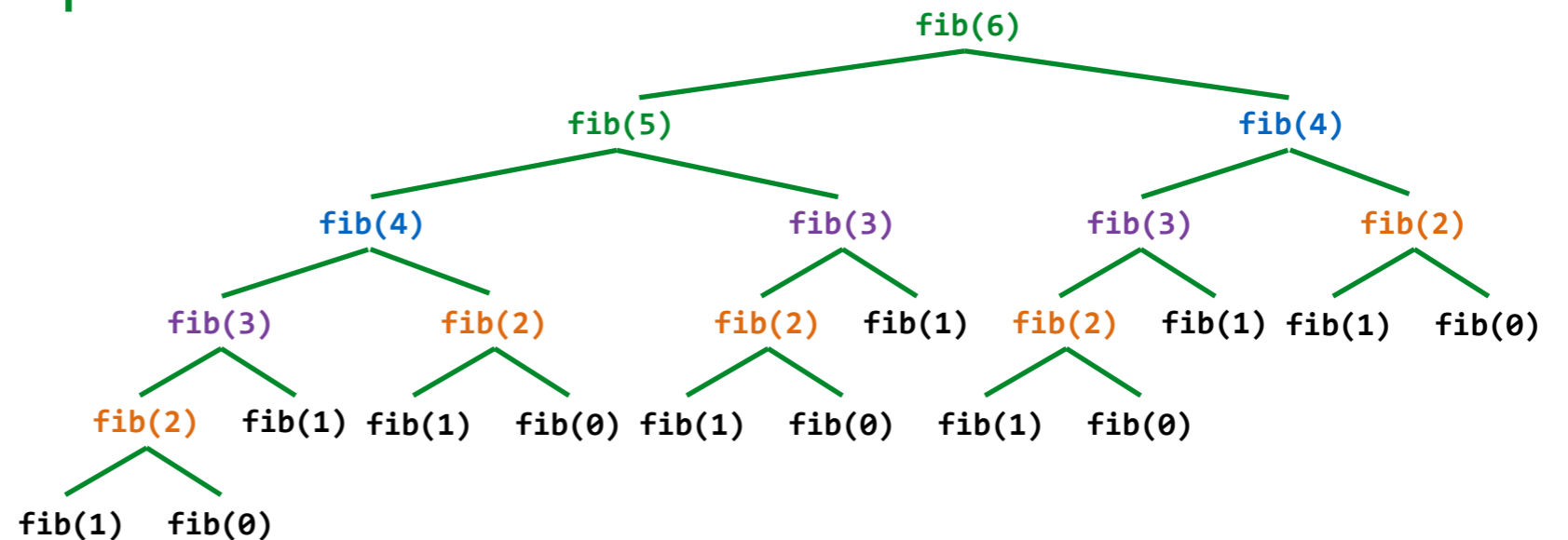

Fibonacci

Rekursion leder **inte alltid** till **bra kod!**

Java:

```
public static int fib(int n) {  
    if (n < 2) {  
        return n;  
    } else {  
        return fib(n-1) + fib(n-2);  
    }  
}
```

Flera rekursiva anrop **beräknar samma sak:**

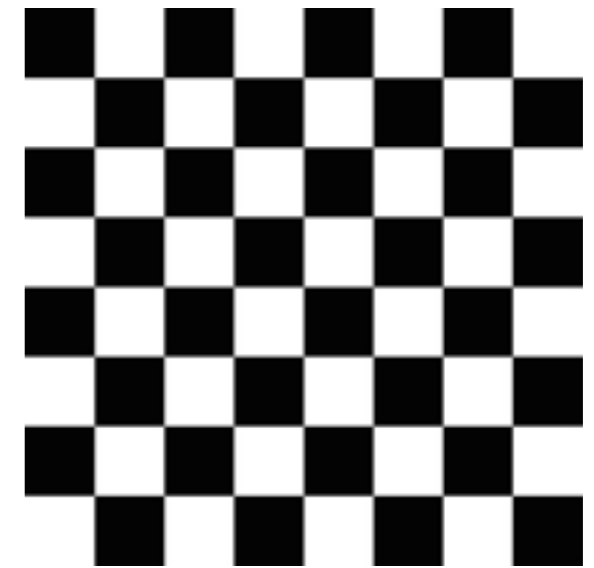


Hästen på schackbräde

Utmaning:

Kan en schackhäst gå till varje ruta på ett schackbräde utan att besöka en enda ruta mera än en gång?

Tips: en stilig lösning med rekursion?



```

public class Horse {

    public static String board[][] = new String[8][8];

    public static int totalVisited = 0;
    public static int currentlyVisited = 0;

    // L = left
    // R = right
    // U = up
    // D = down

    public static final String LU = "LU";
    public static final String LD = "LD";
    public static final String DL = "DL";
    public static final String DR = "DR";
    public static final String UR = "UR";
    public static final String UL = "UL";
    public static final String RU = "RU";
    public static final String RD = "RD";
    public static final String XX = "XX";

    public static void printState() {
        System.out.println();
        System.out.println("Total visited: " + totalVisited);
        System.out.println("Curr visited: " + currentlyVisited);
        for (int y=0; y<8; y++) {
            for (int x=0; x<8; x++) {
                if (board[x][y] == null) {
                    System.out.print("-- ");
                } else {
                    System.out.print(board[x][y] + " ");
                }
            }
            System.out.println();
        }
    }

    // nånting här

    public static void main(String[] args) {
        // nånting här
    }
}

```

