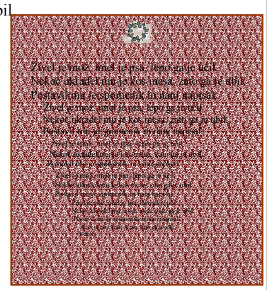
Pesmica

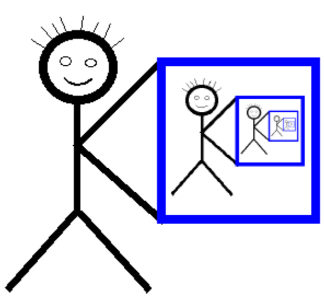
Živel je mož, imel je psa, lepo ga je učil.
Nekoč ukradel mu je kos mesa, zato ga je ubil.
Postavil mu je spomenik in nanj napisal:



Zgodba

Bila je temna, nevihtna noč ... Ladjo so valovi premetavali naprej in nazaj, veter je zavijal med jadri in dež se je zlival na palubo. Posadka je bila zbrana ob petrolejki. Vsi so se zavijali v odeje in trepetali, ko je kapitan pričel pripovedovati zgodbo:
"Bila je temna, nevihtna noč ..."

Slika v sliki

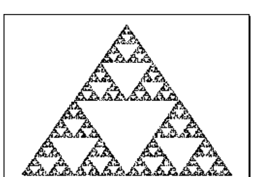


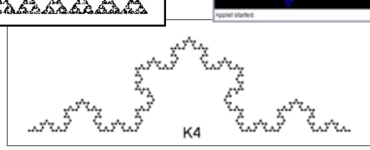
Rekurzija

- SSKJ:
  - rekurz: *knjiž.* vrnitev (na kako stvar, dejstvo)
- Postopek, ki je definiran (določen, opisan) sam s sabo.

Problemi

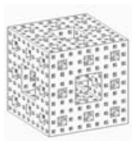
Problemi





Problemi

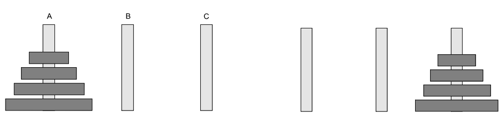
- Izračunaj volumen telesa, preluknjanega n-krat



- Poišči največje in najmanjše število v tabeli števil
- Uredi podatke po velikosti.
- Izračunaj produkt naravnih števil od 1 do n.

Problemi

- Izračunaj y^n s čim manj množenji.
- Hanoiski stolpiči



REKURZIJA

- Različni problemi
- Naloga:
  - Sestavi navodila (postopek) s katerim bi problem rešil
- Navkljub različnosti:
  - Skupni prijem: rekurzija

Rekurzija

- Rešitev problema – podana s samim problemom, le nad manjšim obsegom podatkov
- V opisu postopka rešitve uporabimo kar ta postopek
- Če želimo priti do rešitve, ne moremo nadaljevati v nedogled kot npr. pri pesmici
- ustavitveni pogoj:
  - Kdaj v postopku ne uporabimo istega postopka
  - Običajno: ko je problem "majhen" (enostaven)

Faktoriela

- 7! = 1 * 2* 3 * 4 * 5 * 6 * 7
- 3! = 6
- Zelo hitro naraščajoča zadeva
  - 3! = 6
  - 5! = 120
  - 42! = 1405006117752879898543142606244511569936384000000000
- Rekurzivna definicija: *n! = n * (n-1)!*
- n! bomo izračunali, če bomo poznali (n-1)!
- 3! = 3 * 2!
  - 2! = 2 * 1! =
  - 1! = 1 * 0! =
  - 0! = 0 * (-1)! = ???
- 0! = 1

Faktoriela - postopek

Faktoriela(n):
  Če je n = 0, je rezultat 1
  sicer pa
    rezultat = n * faktoriela(n – 1)

Faktoriela

- 0! = 1
- n! = n * (n-1)!

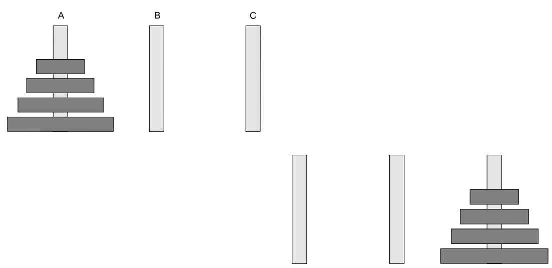
```
public static long faktoriela(int n) {
  if (n == 0) return 1;
  else return n * faktoriela(n-1);
}
```

Faktoriela.java

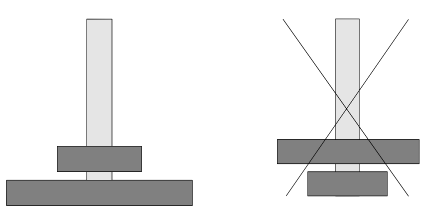
Fak.java

Hanoiski stolpiči

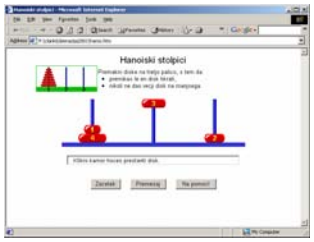
- Problem Hanoiskih stolpičev:



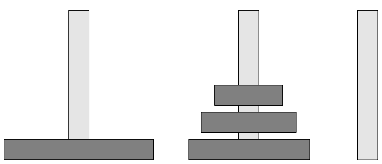
Hanoiski stolpiči



Hanoiski stolpiči - prikaz

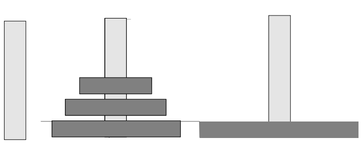


Hanoiski stolpiči -ideja



prestavi n-1 obročev z A na B (s pomočjo C)

Hanoiski stolpiči -ideja



Prestavi obroč z A na C

Prestavi n –1 obročev z B na C (s pomočjo A)

Hanoiski stolpiči

- Preloži n obročev z A na C s pomočjo B
  - Preloži n-1 obročev z A na B s pomočjo C
  - Preloži obroč z A na C
  - Preloži n-1 obročev z B na C s pomočjo A
- ustavi ?

Hanoiski stolpiči

Preloži n obročev z A na C s pomočjo B
  Če je n = 1 potem preloži obroč z A na C
  sicer pa
    Preloži n-1 obročev z A na B s pomočjo C
    Preloži obroč z A na C
    Preloži n-1 obročev z B na C s pomočjo A

Hanoiski stolpiči

/ n obročev z A na C s pomočjo B */*
Hanoi(n, A, B, C)
  Če je n = 1 potem preloži obroč z A na C
  sicer pa
    Hanoi(n-1, A, C, B)
    Preloži obroč z A na C
    Hanoi(n-1, B, A, C)

Hanoiski stolpiči

```
public static void hanoi(int n, char st1,
         char st2,char st3)
{
  if (n == 1)
  {
    System.out.println("Prelozi z " + st1
      + " na " + st3);
  }
  else
  {
    hanoi(n-1, st1, st3, st2);
    System.out.println("Prelozi z " + st1
      + " na " + st3);
    hanoi(n-1, st2, st1, st3);
  }
}
```

HanoiskiStolp.java

y^n

- n je potenca števila 2 (denimo 16)
- $y^{16} = y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y\,y$ .... 15 množenj
- $y^{16} = y^8\,y^8$
- $y^8 = y^4\,y^4$
- $y^4 = y^2\,y^2$
- $y^2 = y\,y$
- .... 4 množenja

y^n

- Kaj pa, če n npr.14
- $y^{14} = y^7\,y^7$
- $y^7 = y\,y^6$
- $y^6 = y^3\,y^3$
- $y^3 = y\,y^2$
- $y^2 = y\,y$ .... 5 množenj

y^n

- `if (n == 2) return y * y;`
```
else {
   pom = pot(y, n / 2);
   return pom*pom;
}
```
- če ni potenca 2
- `if (n == 1) return y;`
```
else
  { pom = pot(y, n / 2);
    if (n % 2 == 0)return pom * pom;
    else return y * pom * pom;
  }
```


Kaj je torej rekurzija

- Kako je v slovarju definirana beseda 'rekurzivno' ?
- *Piše:* Glej 'rekurzivno'.

Fibbonacijevo zaporedje

- Fibonaccijeva števila so zaporedje števil
- 1 1 2 3 5 8 13 21 34 55 ...
- Kot vidimo, sta prvi dve števili v zaporedju enaki 1, vsako naslednje število pa dobimo tako, da seštejmo prejšnji dve.

Fibbonaci1.java

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


Fibbonaci1

```
> java Fibonacci 4
Exception in thread "main"
  java.lang.StackOverflowError
      at Fibonacci.fib(Fibonacci.java:4)
      at Fibonacci.fib(Fibonacci.java:8)
      at Fibonacci.fib(Fibonacci.java:8)
      at Fibonacci.fib(Fibonacci.java:8)
```


Fibbonaci2

- Rekurzija se ni ustavila
- Vstavimo `println` stavek

```
public class Fibonacci {
    public static int fib(int n) {
       System.out.println("Izvajam fib(" + n + ")");
       if (n == 1)
       {
         return 1;
      }
      else
       {
        return fib(n - 2) + fib(n - 1);
          }
    }
}
```

Fibbonaci2

```
> java Fibonacci2 4
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(0)
Izvajam fib(-2)
Izvajam fib(-4)
Izvajam fib(-6)
Izvajam fib(-8)
Izvajam fib(-10)
Izvajam fib(-12)
...
```


Fibbonaci3

- Pozabili na primer n = 2!

```
public class Fibonacci {
    public static int fib(int n)
    {
      System.out.println("Izvajam fib(" + n + ")");
       if (n == 1 || n == 2)
       {
         return 1;
      }
      else
       {
        return fib(n - 2) + fib(n - 1);
          }
    }
}
```


Fibbonaci3

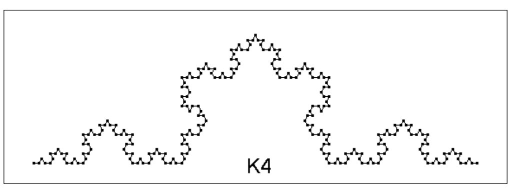
```
> java Fibonacci3 4
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
3
> java Fibonacci3 8
Izvajam fib(8)
Izvajam fib(6)
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(5)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(7)
Izvajam fib(5)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(6)
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(5)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
Izvajam fib(4)
Izvajam fib(2)
Izvajam fib(3)
Izvajam fib(1)
Izvajam fib(2)
21
```

Končna oblika

```
public class Fibonacci4 {
  public static int fib(int n)
  {
     if (n == 1 || n == 2) {
       return 1;
     }
     else
      { return fib(n - 2) + fib(n - 1);
     }
  }

  public static void main(String[] args) {
     int k = Integer.parseInt(args[0]);
    System.out.println(fib(k));
  }
}
```


Kochova črta



Kochova črta

- Kochova črta stopnje 0: daljica od A do B.

K0

- Kochova črta stopnje 1: daljico od A do B razdelimo na tretjine in srednjo tretjino nadomestimo z dvema v obliki trikotnika

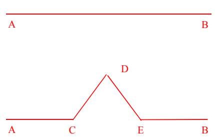

K1

Kochova črta

- Kochova črta stopnje 1 je sestavljena iz štirih Kochovih črt stopnje 0
- Kochova črta stopnje 2: isti postopek (srednjo tretjino nadomestiti z dvema enako dolgima daljicama) ponovimo na vseh črtah, ki sestavljajo Kochovo črto stopnje 1

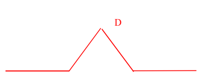
Kochova črta stopnje n od A do B

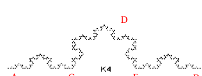
- Daljico AB spremenimo po ustreznem postopku v 4 dele: od A do C, od C do D, od D do E in od E do B.



Kochova črta stopnje n od A do B

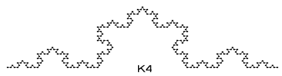
- Nariši 4 Kochove črte stopnje n – 1 – od A do C, od C do D, od D do E in od E do B





- Seveda to velja le, če ne rišemo Kochove črte stopnje 0
- Ustavitveni pogoj?
  - Stopnja črte = 0
  - Kochova črta stopnje 0: daljica od A do B

Kochova črta stopnje n od A do B



K4

- Kochova črta stopnje n od A do B:

```
Če (n = 0) nariši daljico AB
sicer
    izračunamo C, D, E
    nariši Kochovo črto st. n – 1 od A do C
    nariši Kochovo črto st. n – 1 od C do D
    nariši Kochovo črto st. n – 1 od D do E
    nariši Kochovo črto st. n – 1 od E do B
```


Koch

```
public void koch(Graphics g, int n, int ax, int ay, int bx, int by)
  {
     // narišemo Kochovo črto stopnje n od A do B

     if (n == 0)
     { // črta od A do B
        g.drawLine(ax, ay, bx, by);
     }
     else
     {
        int cx = ax + (bx - ax)/3;
        int cy = ay + (by - ay)/3;
        int ex = ax + 2 * (bx - ax)/3;
        int ey = ay + 2 * (by - ay)/3;
        int dx = (int) ((ax + bx) / 2.0 + Math.sqrt(3)/6 * (ay - by));
        int dy = (int) ((ay + by) / 2.0 + Math.sqrt(3)/6 * (bx - ax));
        koch(g, n - 1, ax, ay, cx, cy);
        koch(g, n - 1, cx, cy, dx, dy);
        koch(g, n - 1, dx, dy, ex, ey);
        koch(g, n - 1, ex, ey, bx, by);
     }
  } // ==================== Koch ====================
```

Lomljenka.java

Lomljenka1.java