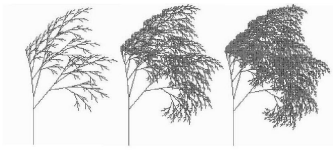
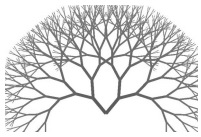
Programozás III

REKURZIÓ

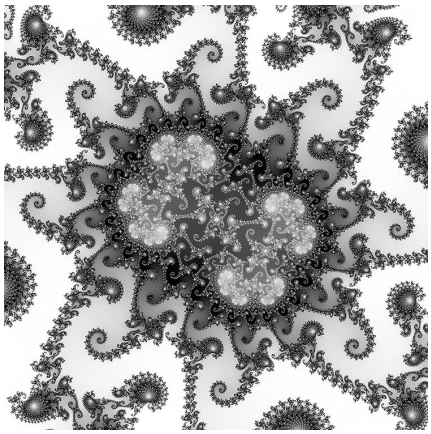
(M.C. Escher: Rajzoló kezek)

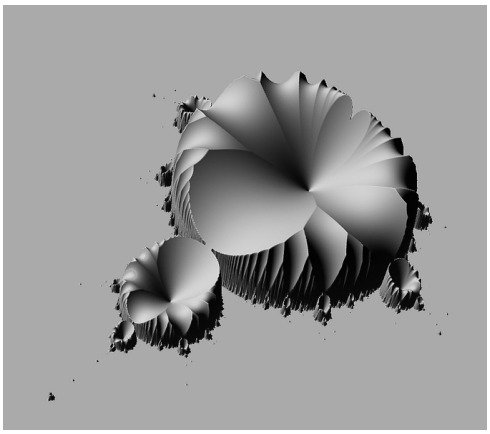


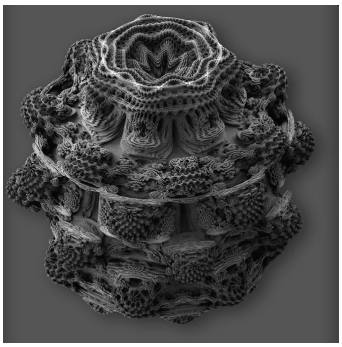
REKURZIÓ

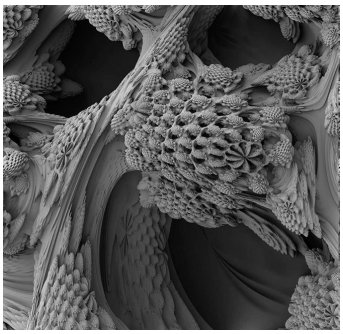


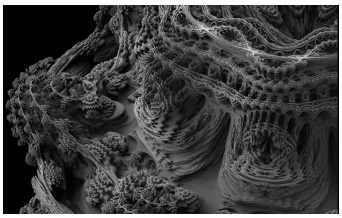
„Ahhoz, hogy megértsd a rekurziót, meg kell értened a rekurziót.”





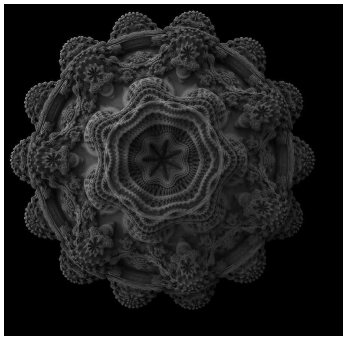


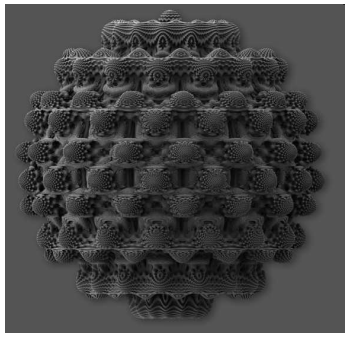


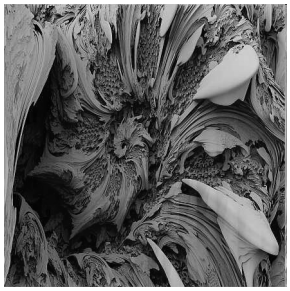


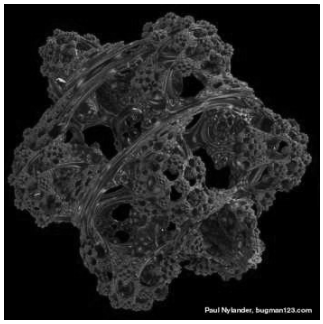
http://www.wired.com/wiredscience/2009/12/mandelbulb-gallery/2/

http://www.origo.hu/tudomany/komment/20100103-mandelbulb-matematika-a-mandelbrot-halmaz-terbeli-megfeleloje.html?cmnt_page=1









http://www.skytopia.com/project/fractal/2mandelbulb.html

REKURZIÓ

Rekurzív az az eljárás vagy függvény, amely közvetve vagy közvetlenül önmagára hivatkozik.

Pl.: $n! = n*(n-1)!$ ha $n>1$ és $1! = 1$

A rekurzió mindig két részből áll:

1. statikus rész: pl.: $1! = 1$

Célja: a megállás biztosítása

2. dinamikus rész: pl.: $n! = n*(n-1)!$

Célja: a továbblépés biztosítása

REKURZIÓ

Másképp fogalmazva: egy rekurzió definíciója két részből áll:

1. Az **alapesetek** meghatározása.

2. **Redukciós szabályok** megadása.
A redukciós szabály segítségével a problémát visszavezetjük egy kisebb méretű probléma megoldására.
A redukciós szabály megadásakor felhasználjuk a definiálandó fogalmat is.

Példa: n! rekurzív definíciója:

1. Alapeset:	$n! = 1$	ha $n = 1$
2. Redukciós szabály:	$n! = n * (n-1)!$	ha $n > 1$

REKURZIÓ A JAVA-BAN

```
class RekurzivFakt{
   public static void main(String args[]){
      System.out.print("n értéke: ");
      int n = Input.readInt();
      double f = fakt(n);
      if(f == 0){
       System.out.println("Ennek a számnak nincs faktoriálisa.");
      }
      else System.out.println(n + " faktoriálisa: " + fakt(n));
   }

   public static double fakt(int n){
      if (n<0) return 0;
      if(n<=1) return 1;
      return n*fakt(n-1);
   }
}
```

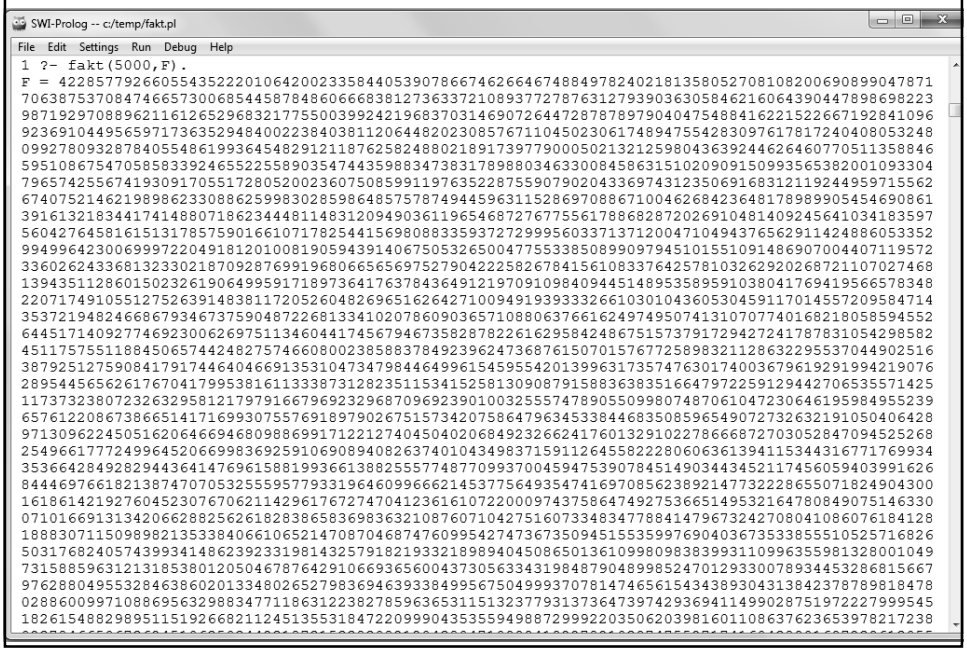
Rekurzió a Java-ban

Az eredmény meghatározása:

☺
1 <= 1
fakt(1) = 1
2 <= 1
fakt(2) = 2
3 <= 1
fakt(3) = 6

```
static double fakt(int n){
   if (n<=1) return 1;
   else return n*fakt(n-1);
}
```

KITÉRŐ:



REKURZIÓ A JAVA-BAN

Fibonacci sorozat: $a_1 = 1$, $a_2 = 1$, $a_n = a_{n-1} + a_{n-2}$

```
public class Fibonacci{

   public static void main(String args[]){
       System.out.print("Meddig számoljak? :");
       int n =Input.readInt();
       System.out.printf("Az első %d Fibonacci szám: \n", n);
       for(int i = 1; i<= n; i++)
           System.out.println(fib(i));
   }

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

HF: Javítani, hogy n <=0-ra hibát jelezzen.

Megjegyzés: Van hatékonyabb megoldás is.

REKURZIÓ A JAVA-BAN

Mi lehet a rekurzió előnye, hátránya?

Előny:

viszonylag egyszerű

Hátrány:

túl nagy memóriaigény (verem memória)

MÁSIK REKURZIÓS PÉLDA

Hanoi torony:

adott három rúd: A, B és C. A rúdon induláskor N darab különböző átmérőjű lyukas korong helyezkedik el egymás fölött csökkenő sorrendben.
Feladat: a korongok áthelyezése az alábbi szabályok alapján:
– a B rúd közbenső tárolásra használható
– minden lépésben csak egy korong mozgatható
– minden korong csak nála nagyobb átmérőjű korongra helyezhető.

(Legenda: Hanoi közelében egy kolostorban 64 korongból álló tornyot raknak át a szerzetesek a fenti szabályok betartásával, minden nap egyetlen korongot mozgatva. A legenda szerint akkor lesz vége a világnak, ha az átrakást befejezik).

1 korong/sec, 64 korong $\Rightarrow$ 585 milliárd év

HANOI TORONY CIKLUSSAL

Hanoi tornyai

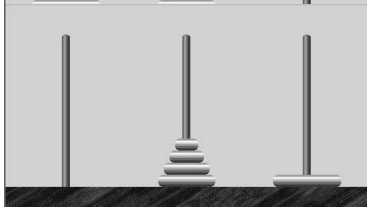
Megvalósítás ciklussal

- Ha n korong van, 2^n-1 lépés kell.
- Minden páratlanadik lépést a legkisebb koronggal kell megtenni.
- Ha n páratlan, A-B-C-A sorrendben lép, ha n páros, A-C-B-A sorrendben lép.
- Ha a következő lépést ugyanarra kell megtenni, oda kell lépni, ahonnan léptünk.
- Ha a következő lépést ellenkező irányban kell megtenni, onnan kell lépni, ahonnan léptünk.

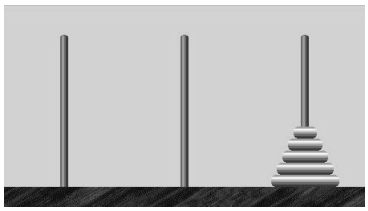
HANOI TORONY – MEGOLDÁS

1. Rakj n-1 korongot A rúdról B rúdra

2. Rakj 1 korongot A rúdról C rúdra



3. Rakj n-1 korongot B rúdról C rúdra



```
public class Hanoi
{
  public static void main (String args[])
  {
    String bal = "bal rúd";
    String jobb = "jobb rúd";
    String kozep = "középső rúd";

    System.out.print("Hány koronggal dolgozzam?: ");
    int N = Input.readInt();
    while( N<= 0){
        System.out.println("A darabszám nem lehet negatív.");
        System.out.print("Hány koronggal dolgozzam?: ");
        N = Input.readInt();
    }
    pakol(N, bal, jobb, kozep);
  }

  static void pakol(int n, String bal, String kozep, String jobb)
  {
    if (n > 0) {
      pakol(n-1, bal, kozep, jobb);
      atrak(bal, jobb);
      pakol(n-1, kozep, jobb, bal);
    }
  }

  static void atrak(String innen, String oda)
  {
    System.out.print("rakj egy korongot ");
    System.out.print(innen);
    System.out.print(" --> ");
    System.out.println(oda);
  }
}
```

HANOI TORONY – JAVA RÉSZLET

```
public static void main (String args[])
{
  String bal = "bal rúd";
  String jobb = "jobb rúd";
  String kozep = "középső rúd";

  System.out.print("Hány koronggal dolgozzam?: ");
  int N = Input.readInt();
  while( N<= 0){
      System.out.println("A darabszám nem lehet negatív.");
      System.out.print("Hány koronggal dolgozzam?: ");
      N = Input.readInt();
  }
  pakol(N, bal, jobb, kozep);
}
```

HANOI TORONY – JAVA RÉSZLET

```
  static void pakol(int n, String bal, String kozep, String jobb)
  {
    if (n > 0) {
      pakol(n-1, bal, kozep, jobb);
      atrak(bal, jobb);
      pakol(n-1, kozep, jobb, bal);
    }
  }

  static void atrak(String innen, String oda)
  {
    System.out.print("rakj egy korongot ");
    System.out.print(innen);
    System.out.print(" --> ");
    System.out.println(oda);
  }
}
```

HANOI TORONY – FUTÁSI EREDMÉNY

```
--------------------Configuration: <Default>--
Hány koronggal dolgozzam?: 4
rakj egy korongot bal rúd --> középső rúd
rakj egy korongot bal rúd --> középső rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot bal rúd --> középső rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot középső rúd --> jobb rúd
rakj egy korongot bal rúd --> középső rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot középső rúd --> jobb rúd
rakj egy korongot jobb rúd --> bal rúd
rakj egy korongot középső rúd --> jobb rúd
rakj egy korongot középső rúd --> jobb rúd
rakj egy korongot bal rúd --> középső rúd
```

KÖZISMERT(?) PÉLDA

MOTIVÁCIÓ-FÉLE

Részlet egy állásinterjúkról szóló levélből:

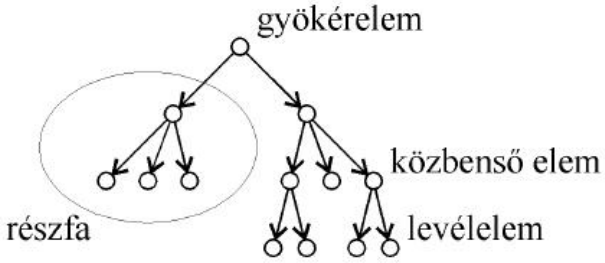
A programozós kérdések rendszerint ugyanazokra a területekre térnek ki:

1. Néhány alapvető kérdés a JRE, JVM, JDK környékéről. Ezek többnyire egyszerűek, bár előfordulhat nehezebb memóriakezeléssel kapcsolatos is.
2. Algoritmus-elméletet nem mindenhol, de kérdezhetnek: Rendezések, keresések stb. **Egy bináris keresést elmagyarázni angolul, illetve megírni Javaban nem is olyan könnyű...**

HIERARCHIKUS ADATSZERKEZETEK

FA (tree)

Dinamikus, homogén adatszerkezet, amelyben minden elem megmondja a rákövetkezőjét.



HIERARCHIKUS ADATSZERKEZETEK – BINÁRIS FA

BINÁRIS FA:

A számítástechnikában kitüntetett szerepe van a bináris fának.
A bináris fa olyan fa, amelyben minden elemnek legfeljebb két rákövetkezője lehet.
Szigorú értelemben vett bináris fáról akkor beszélünk, amikor minden elemnek 0 vagy pontosan 2 rákövetkező eleme van.

Létrehozása:
Létrehozzuk az üres fát, majd létrehozzuk a gyökérelemet, ezután bővítjük a fát.

HIERARCHIKUS ADATSZERKEZETEK – BINÁRIS FA

Speciális faművelet a **bejárás**:
A bejárás az a tevékenység, amikor a fát, mint hierarchikus adatszerkezet egy lineáris adatszerkezetre képezzük le.
A fa elemeit a bejárás folyamán egyszer, és pontosan egyszer érintjük, az elemek között valamilyen sorrendet állapítunk meg, attól függően, hogy hogyan járjuk be a fát.

Bejárási módok megkülönböztetése: attól függően, hogy mikor kerül sorra a gyökérelem.

HIERARCHIKUS ADATSZERKEZETEK – BINÁRIS FA

Bejárások:

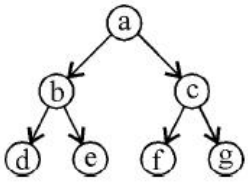
Preorder bejárás: Ha a fa üres, akkor vége a bejárásnak, egyébként vegyük a **gyökér**elemet és dolgozzuk fel. Ezután járjuk be preorder módon a **bal**oldali, majd a **jobb**oldali részfát.

Inorder bejárás: Ha a fa üres, akkor a bejárás befejeződik. Egyébként járjuk be inorder módon a **bal**oldali részfát, majd dolgozzuk fel a **gyökér**elemet, és járjuk be ugyanezen módszerrel a **jobb**oldali részfát.

Postorder bejárás: Üres fánál vége, egyébként járjuk be postorder módon a **bal**oldali majd a **jobb**oldali részfát, végül dolgozzuk fel a **gyökér**elemet.

Megvalósítás: rekurzióval

HIERARCHIKUS ADATSZERKEZETEK – BINÁRIS FA



preorder: (gyökér, bal, jobb) – abdecfg

inorder: (bal, gyökér, jobb) – dbeafcg

postorder: (bal, jobb, gyökér) – debfgca

Bináris rendezés: ügyes faépítés + megfelelő bejárás

HIERARCHIKUS ADATSZERKEZETEK – FA

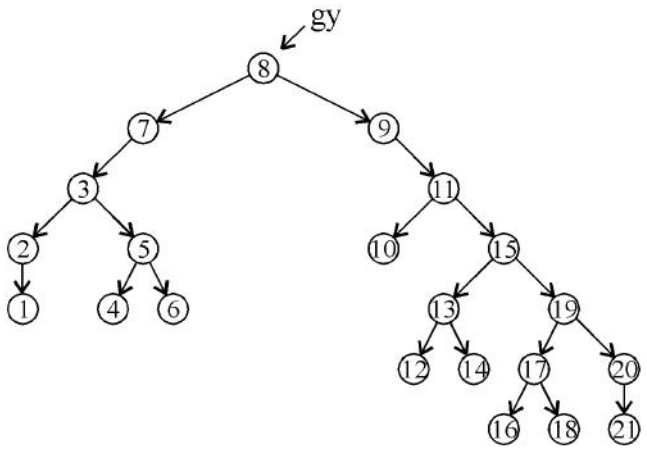
KERESŐ-FA:

A bináris fákat gyakran használják olyan adathalmaz feldolgozására, amikor az adatelemeknek van egy kulcsrészük (táblázatok) vagy maguk az adatelemek különböző értékűek. Ilyenkor a kulcs a feldolgozás alapja.

Ha adott elemszám mellett úgy építjük fel a fát, hogy bármely elemére igaz, hogy az elem baloldali részfájában az összes elem kulcsa kisebb, a jobboldali részfájában az összes elem kulcsa pedig nagyobb az adott elem kulcsánál, akkor **keresőfá**ról (vagy rendezőfáról) beszélünk.

HIERARCHIKUS ADATSZERKEZETEK – FA

Kereső-fa – példa:



HIERARCHIKUS ADATSZERKEZETEK – FA

A keresőfa jelentősége:
Ha egy adott elemet meg akarunk keresni a fában, akkor a gyökértől kiindulva bármelyik kulcsú elem megkereshető úgy, hogy vizsgáljuk: a gyökérelem kulcsa megegyezik-e a keresett elem kulcsával.
Ha igen, megállunk, ha nem, megnézzük, hogy annál kisebb-e vagy nagyobb.
Ha nagyobb, akkor a jobboldali részfán, ha kisebb, akkor a baloldali részfán haladunk tovább ugyanezzel a módszerrel mindaddig, míg az elemet meg nem találjuk, vagy nem jutunk el egy olyan elemhez, amelyiknél az adott irányban nincs több elem, ekkor a keresett kulcsú elem nem szerepel a fán.

HIERARCHIKUS ADATSZERKEZETEK – FA

A keresés gyors, mert maximum a fa magassága+1 hasonlítással vagy megtaláljuk az elemet, vagy az elem nincs a fában.
A keresőfában az elemek olyan sorrendben vannak, hogy ha inorder módon járjuk be a fát, akkor az elemek egy rendezett sorozatát kapjuk.

HIERARCHIKUS ADATSZERKEZETEK – FA

A keresőfa létrehozása:

Vegyük az elemeket az adott sorrendben.
Az első elem lesz a fa gyökere.
A második elemet véve nézzük meg, szerepel-e már a fában. Ha igen, akkor hiba történt (kétszer nem lehet benne ugyanaz az elem), ha nem, akkor döntünk, hogy az hol helyezkedjen el az előzőhöz képest (kisebb vagy nagyobb a gyökérelemnél). Ha nagyobb, akkor a jobboldali részfának lesz az eleme, ha pedig kisebb, akkor a baloldalinak.
Beillesztjük az új elemet a részfába: addig megyünk, amíg levélelemet nem találunk, mert mindig levélelemmel bővítünk.

BINÁRIS FA KEZELÉSE – REKURZIÓVAL

Bináris fa definíciója:

```
public class BinarisFa {
    private BinarisFa bal;
    private BinarisFa jobb;
    private int gyoker;

    public BinarisFa(int gyoker) {
        this.gyoker = gyoker;
    }
```

+ set/get

```
/* Példa fa-struktúra felépítésére, bejárására */

public class FaDemo {

    public void indit() {

        int i;
        int tomb[] = new int[]{5, 1, 8, 6, 3, 9};
        System.out.println("A számok eredeti sorrendje: ");

        for (i = 0; i < tomb.length; i++) {
            System.out.println(tomb[i]);
        }

        BinarisFa fa = new BinarisFa(tomb[0]);

        for (i = 1; i < tomb.length; i++) {
            beszur(fa, tomb[i]);
        }

        System.out.println("\nA fa létrehozása utáni sorrend:");
        bejar(fa);
    }
```

BINÁRIS FA KEZELÉSE – REKURZIÓVAL

```
public void beszur(BinarisFa fa, int adat) {
    if (adat < fa.getGyoker()) {
        if (fa.getBal() != null) {
            beszur(fa.getBal(), adat);
        } else {
            System.out.println("Beszúrtam " + adat + "-t "
                    + fa.getGyoker() + " bal oldalára");
            fa.setBal(new BinarisFa(adat));
        }
    } else {
        if (fa.getJobb() != null) {
            beszur(fa.getJobb(), adat);
        } else {
            System.out.println("Beszúrtam " + adat + "-t "
                    + fa.getGyoker() + " jobb oldalára");
            fa.setJobb(new BinarisFa(adat));
        }
    }
}
```

BINÁRIS FA KEZELÉSE – REKURZIÓVAL

```
public void bejar(BinarisFa fa) {
    if (fa != null) {
        bejar(fa.getBal());
        System.out.println(fa.getGyoker());
        bejar(fa.getJobb());
    }
}


public class BinarisRendezes {

    /**...*/
    public static void main(String[] args) {
        new FaDemo().indit();
    }
}
```

BINÁRIS FA KEZELÉSE – REKURZIÓVAL

```
A számok eredeti sorrendje:
5
1
8
6
3
9
Beszúrtam 1-t 5 bal oldalára
Beszúrtam 8-t 5 jobb oldalára
Beszúrtam 6-t 8 bal oldalára
Beszúrtam 3-t 1 jobb oldalára
Beszúrtam 9-t 8 jobb oldalára

A fa létrehozása utáni sorrend:
1
3
5
6
8
9

Process completed.
```

BINÁRIS FA