

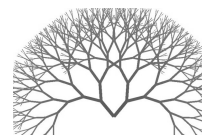
Programozás III

REKURZIÓ

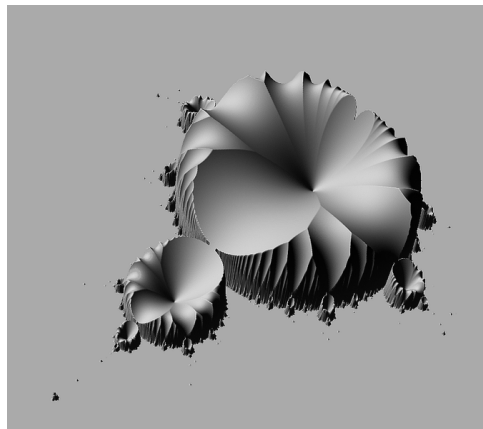
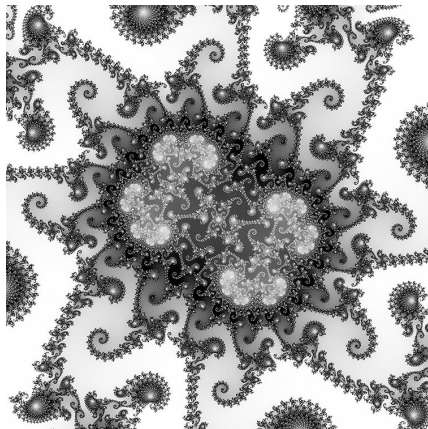
(M.C. Escher: Rajzoló kezek)

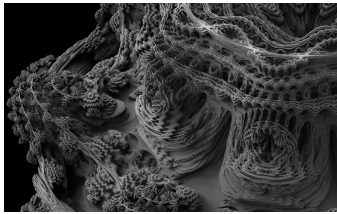
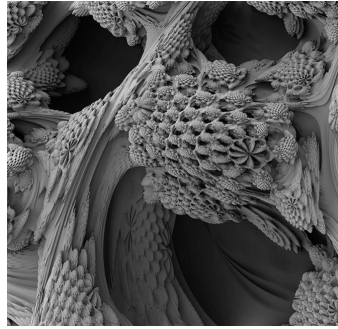
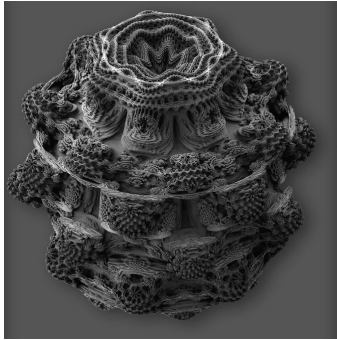


REKURZIÓ



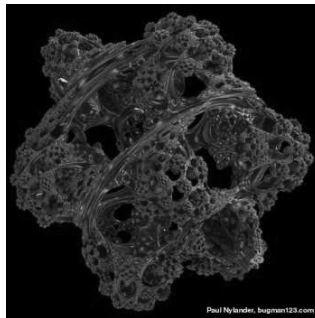
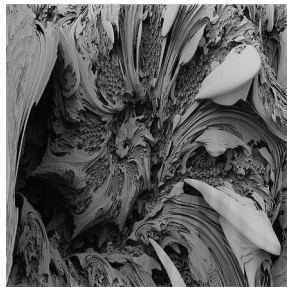
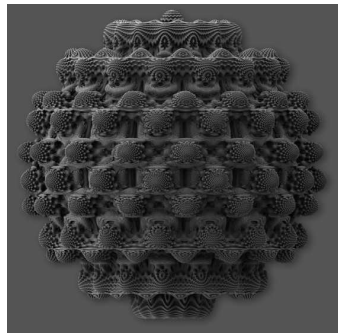
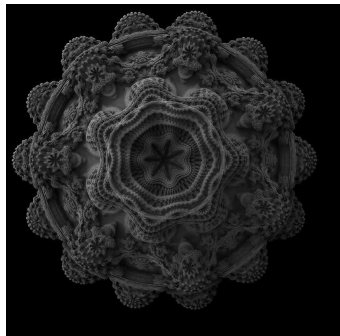
„Ahhoz, hogy megértsd a rekurziót, meg kell értened a rekurziót.”





<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.: $n! = n \cdot (n-1)!$ 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.: $n! = n \cdot (n-1)!$

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: $n! = n \cdot (n-1)!$ 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:

```
SWI-Prolog -- c:/temp/fakt.pl
File Edit Settings Run Debug Help
1 ?- fakt(5000,F).
F = 4228577926605543522201064200233584405390786674626646748849782402181358052708108200690899047871
70638753708474665730068544587848606668381273633721089377278763127939036305846216064390447898698223
98719297088962116126529683217755003992421968370314690726447287878979040475488416221522667192841096
92369104495659717363529484002238403811206448202308576711045023061748947554283097617817240408053248
0992780932878405548619936454829121187625824880218917397900050213212598043639244626460770511358846
59510867547058583392465522558903547443598834738317898803463300845863151020909150993565382001093304
79657425567419309170551728052002360750859911976352287559079020433697431235069168312119244959715562
67407521462198986233088625998302859864857578749445963115286970886710046268423648178989905454690861
39161321834417414880718623444811483120949036119654687276775561788682872026910481409245641034183597
5604276458165131785759016610717825441569808833593727299956033713712004710494376562911424886053352
99499642300699972204918120100819059439140675053265004775533850899097945101551091486907004407119572
33602624336813233021870928769919680665656975279042225826784156108337642578103262920268721107027468
13943511286015023261906499591718973641763784364912197091098409445148953589591038041769419566578348
2207174910551275263914838117205260482696516264271009491939332661030104360530459117014557209584714
35372194824668679346737590487226813341020786090365710880637661624974950741310707740168218058594552
64451714092774692300626975113460441745679467358287822616295842486751573791729427241787831054298582
45117575511884506574424827574660800238588378492396247368761507015767725898321128632295537044902516
387925127590841791744464046691353104734798446499615459554201399631735747630174003679619291994219076
28954456562617670417995381611333873128235115341525813090879158836383516647972259129442706535571425
11737323807232632958121797916679692329687096923901003255574789055099807487061047230646195984955239
65761220867386651417169930755769189790267515734207586479634533844683508596549072732632191050406428
97130962245051620646694680988699171221274045040206849232662417601329102278666872703052847094525268
25496617772499645206699836925910690894082637401043498371591126455822280606361394115344316771769934
35366428492829443641476961588199366138825557748770993700459475390784514903443452117456059403991626
8444697661821387470705325595779331964609966621453775649354741697085623892147732228665071824904300
16186142192760452307670621142961767274704123616107220009743758647492753665149532164780849075146330
07101669131342066288256261828386583698363210876071042751607334834778841479673242708041086076184128
18883071150989821353384066106521470870468747609954274736735094515535997690403673533855510525716826
50317682405743993414862392331981432579182193321898940450865013610998098383993110996355981328001049
73158859631213185380120504678764291066936560043730563343198487904898952470129330078934453286815667
97628804955328463660201334802652798369463933849956750499937078147465615434389304313842378789618478
0288600997108869563298834771186312238278596365311513237931373647397429369411499028751972227999545
18261548829895115192668211245135531847220999043535594988729992203506203981601108637623653978217238
```

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)

MÁSİK 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

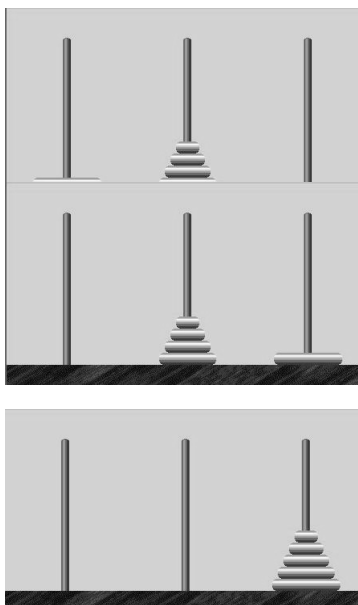
- Ha n korong van, $2^n - 1$ lépés kell.
- Minden páratlanadik lépést a legkisebb koronggal kell megtenni.
- Ha n páratlan, A-B-C-A sorrendben lép, ha n páros, A-C-B-A sorrendben lép.
- Ha a következő lépést ugyanarra kell megtenni, oda kell lépni, ahonnan léptünk.
- Ha a következő lépést ellenkező irányban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

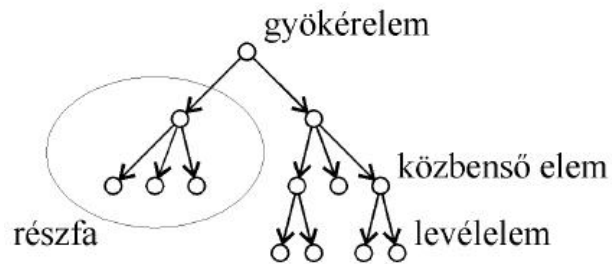
1. Néhány alapvető kérdés a JRE, JVM, JDK környékéről. Ezek többnyire egyszerűek, bár előfordulhat nehezebb memóriakezeléssel kapcsolatos is.

2. Algoritmus-elméletet nem mindenhol, de kérdezhetnek: Rendezések, keresések stb. **Egy bináris keresést elmagyarázni angolul, illetve megírni Javában 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.

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

HIERARCHIKUS ADATSZERKEZETEK – BINÁRIS FA

Bejárások:

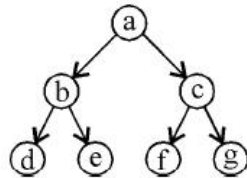
Preorder bejárás: Ha a fa üres, akkor vége a bejárásnak, egyébként vegyük a **gyökérelemet** és dolgozzuk fel. Ezután járjuk be preorder módon a **baloldali**, majd a **jobboldali** részfat.

Inorder bejárás: Ha a fa üres, akkor a bejárás befejeződik. Egyébként járjuk be inorder módon a **baloldali** részfat, majd dolgozzuk fel a **gyökérelemet**, és járjuk be ugyanezen módszerrel a **jobboldali** részfat.

Postorder bejárás: Üres fánál vége, egyébként járjuk be postorder módon a **baloldali** majd a **jobboldali** részfat, végül dolgozzuk fel a **gyökérelemet**.

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

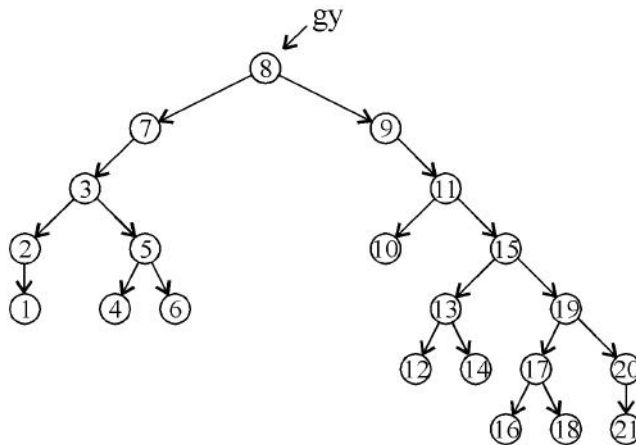
KERESŐ-FA:

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 **keresőfáról** (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

```
Process completed.  
A számok eredeti sorrendje:  
5  
1  
8  
6  
3  
9  
Beszúrta 1-t 5 bal oldalára  
Beszúrta 8-t 5 jobb oldalára  
Beszúrta 6-t 8 bal oldalára  
Beszúrta 3-t 1 jobb oldalára  
Beszúrta 9-t 8 jobb oldalára  
A fa létrehozása utáni sorrend:  
1  
3  
5  
6  
8  
9  
Process completed.
```

BINÁRIS FA

