Algoritmusok és adatszerkezetek II.

előadás

I404e-1

H[10-11:30] BE-002-3 minden héten

Szegedi Tudományegyetem - Természettudományi és Informatikai Kar - Informatikai Tanszékcsoport - Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék - Németh Tamás

Felelős tanszék:

Számítógépes algoritmusok és mesterséges intelligencia tanszék

Nappali tagozaton:

Gyakorlat: heti 1 óra / 1 kredit. Teljesítés módja: Gyakorlati jegy

Előadás: heti 2 óra / 3 kredit. Teljesítés módja: Kollokvium

Tematika
bevezetés, absztrakt adatszerkezetek és műveleteik, statikus és dinamikus memóriakezelés, verem és sor megvalósítása, bináris keresőfák (definíciók, keresés, beszúrás, törlés, tárolási módok), rendezett-minta fa, intervallumvák
bináris keresőfa magassága, véletlen építésű bináris keresőfa, optimális binnáris keresőfa építése önkiegyensúlyozó keresőfák: AVL fák (pont egyensúlyfaktora, forgatások, beszúrás utáni javítás, törlés utáni javítás, rendezettminta-fa tulajdonság fentartása)
általános keresőfák, keresés általános keresőfában. B-fák (beszúrás, törlés)
2-3 fák, piros-fekete fák (definíció, beszúrás utáni javítás, törlés utáni javítás)
amortizációs költségelemzés, önszervező bináris keresőfák (vágás, beszúrás, törlés, egyesítés)
binomiális és Fibonacci kupacok (vág, egyesít, beszúr, töröl műveletek)
halmazerdő, hasítótáblázatok, ugrólisták
geometriai algoritmusok (alapfogalmak, forgásirány, metszés eldöntése, pont helyzetének meghatározása, pontok összekötése zárt nem metsző poligonná, konvex burok meghatározása, legtávolabbi pontpár meghatározása)
geometriai algoritmusok (legközelebbi pontpár meghatározása, szakaszhalmazban metsző szakaszpár keresése, Delanuey háromszöglefedés, Voronoi diagramm, raszteres képek problémái, négyesfa modell), Mintaillesztés (mintaillesztés automatával, Knuth Morris Pratt algoritmus, Rabin-Karp algoritmus)
számelméleti algoritmusok, nyilvános kulcsú titkosítás, RSA algoritmus, véletlenített algoritmusok
megoldás szisztematikus keresése visszalépéssel, megoldás szisztematikus keresése elágazás-korlátozással;korlátozás-szétválasztás elve
online algoritmusok, adatszerkezetek tervezése és használata

Ajánlott irodalom:

- [1] T. H. Cormen, C. E. Leiserson, R.L. Rivest: Algoritmusok, Műszaki Könyvkiadó, 2003.
- **[2] T. H. Cormen, C. E. Leiserson, R.L. Rivest, C. Stein: Új algoritmusok, Scolar Kiadó, 2003.**
- [3] T. Ottman, P. Widmayer: Algoritmen und Datenstrukturen, Wissenschaftsverlag, 1990
- [4] E. Knuth: A számitógépprogramozás művészete, 3. Kötet, Műszaki Könyvkiadó, 1990.
- [5] A. V. Aho, J. E. Hopcroft, J. D. Ullman: Számítógép-algoritmusok tervezése és analízise,Műszaki Könyvkiadó, 1982.
- [6] G. Gonnet, R. Baeza-Yates: Handbook of algorithms and data structures. In Pascal and C. , Addison-Wesley. 1991.
- [7] R. Sedgewick: Algoritms in C++, Addison-Wesley. 1991.
- [8] E. Horowitz, S. Shani: Fundamentals of Computer Algorithms, Computer Science Press, 1998.
- [9] *Adonyi Róbert: Adatstruktúrák és algoritmusok (letölthető jegyzet)*
- [10] J. Hromkovic: Algoritmic Adventures From Knowledge to Magic, Springer

A kurzus teljesítésének feltételei:

Előfeltétel: gyakorlati jegy
Kollokvium: írásbeli vagy szóbeli, elérhető maximális pontszám: 100.

Az írásbeli kollokvium 3 részből áll:

- A. 4 kérdés a kurzus anyagát lefedő témakörökből.
  - Elérhető maximális pontszám: 40
  - Teljesítendő minimális pontszám: 15

- B. Teljesen kidolgozandó tétel.
  - Elérhető maximális pontszám: 30
  - Teljesítendő minimális pontszám: 10

- C. Gyakorlati feladatok megoldása.
  - Elérhető maximális pontszám: 30
  - Teljesítendő minimális pontszám: 15

- A+B+C
  - Elérhető maximális pontszám: 100
  - Teljesítendő minimális pontszám: 50

A szóbeli kollokvium egy részből áll:

- A szóbeli kollokviumon a hallgatóknak kérdésekre kell válaszolnia a tananyaggal kapcsolatban, várhatóan 15-30 percig, felkészülési idő nincs, viszont bármilyen papír alapú segédanyag használható.

- A szóbeli felelet értékelése a teljesítés %-ában mérve történik, az elért % a kollokvium pontszáma.

Érdemjegy:

- 50-től elégséges (2)
- 60-tól közepes (3)
- 72-től jó (4)
- 85-től jeles (5)

Optimalizálás a szoftverfejlesztésben

* futási idő
* tárigény
* továbbított adatmennyiség
* SQL lekérdezések erőforrásigénye
* egyéb felhasznált erőforrások (energia…)

Statikus adatszerkezetek

* Feltétel: tudjuk előre az elemszámot!
* Tömb
* Rekord
* Rekord elemű tömb….

Dinamikus Adatszerkezetek

Pointerek, dinamikus memóriafoglalás, objektumok

Absztrakt adatszerkezetek

- Verem, Sor
- Halmaz, Függvény, Rendezett minta
- Prioritási sor, Egyesíthető prioritási sor
- Egyedi adatszerkezetek (pl. halmazerdő)

Alapműveletek

* adatot felvesz az adatszerkezetbe
* adatot kulcs alapján keres
* adatot kivesz/töröl az adatszerkezetből
* 2 adatszerkezetet egyesít (unió, metszet)
* adatszerkezetet adott kulcs mentén kettévág

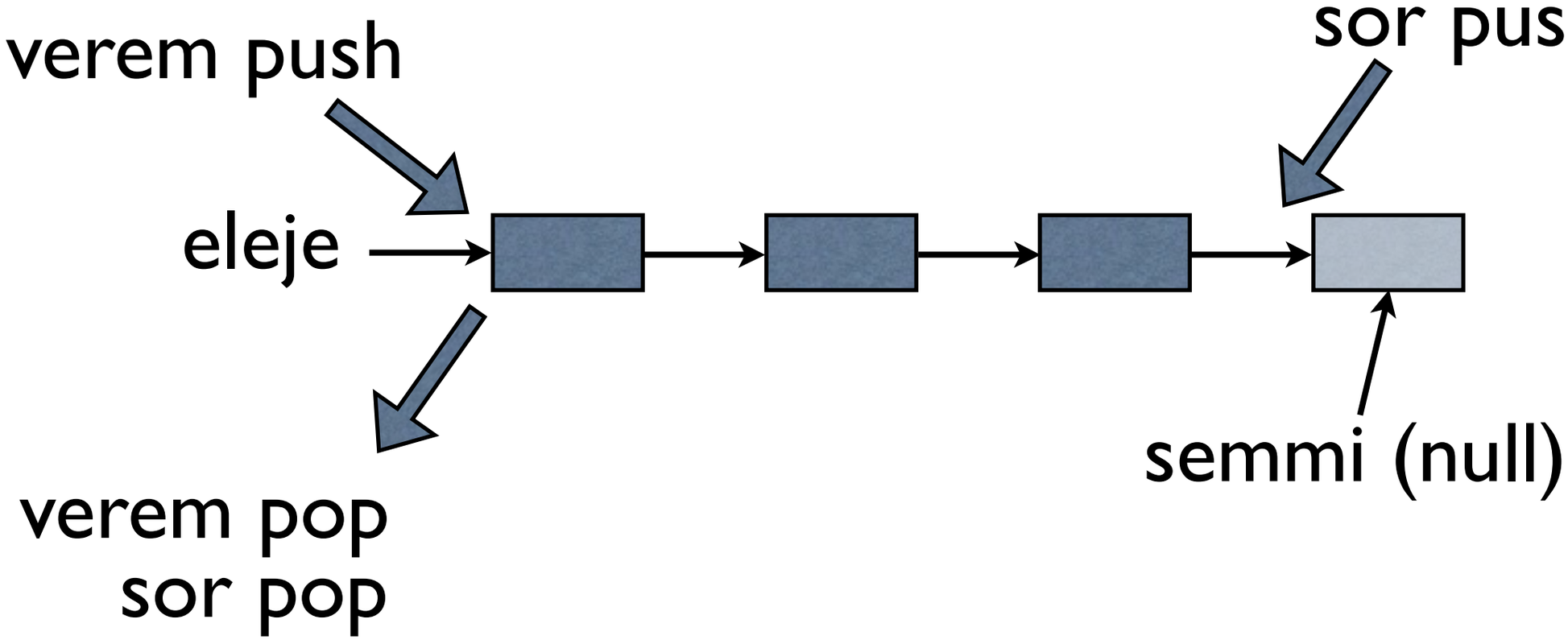
Statikus és dinamikus adatszerkezetek

- a program futtatása előtt le kell foglalni a szükséges memóriát
- tipikus megvalósítás: tömb

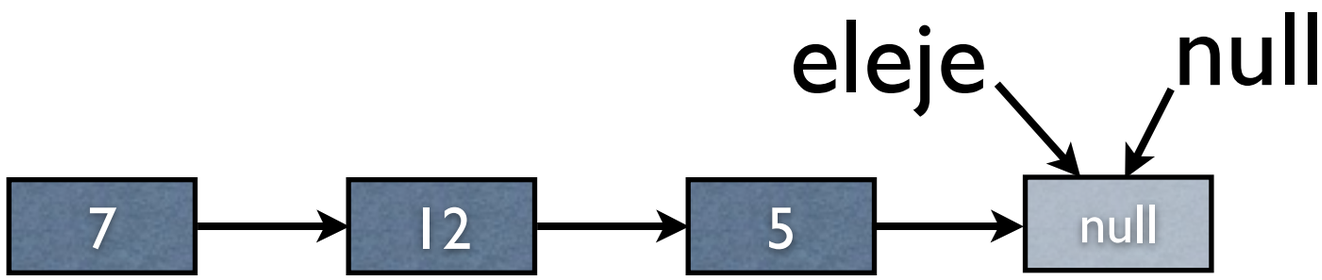
- a program futása közben folyamatosan tudunk memóriát foglalni és felszabadítani
- tipikus megvalósítás: pointerek alkalmazásával

Verem, sor megvalósítása

Alapműveletek: pop, push



Verem (Stack) példa



push(5) push(12) push(7)

pop() pop() pop()

```java
public static void main (String args[]) {
      verem v=new verem() ;
      v.betesz(5) ;
      v.betesz(8) ;
      v.betesz(12) ;
      System.out.println(v.kivesz());
      System.out.println(v.kivesz());
      System.out.println(v.kivesz());
}
```

```java
public class verem {
    int elemszam ;
    struct elem {
        int ertek;
        elem kovetkezo ;
    }
    elem semmi, eleje ;
    verem() {
        elem q=new elem() ;
        semmi=q ;
        eleje=q ;
        elemszam=0 ;
    }
    void betesz(int x) {
        elem q=new elem() ;
        q.kovetkezo=eleje ;
        q.ertek=x ;
        eleje=q ;
        elemszam++ ;
    }
    int kivesz() {
        if (eleje!=semmi) {
            int visszateresi_ertek ;
            visszateresi_ertek=eleje.ertek ;
            eleje=eleje.kovetkezo ;
            elemszam-- ;
            return visszateresi_ertek;
        } else return 0 ;
    }
}
```

Sor (Queue) példa

push(5) push(12) push(7)

pop() pop() pop()

```java
public static void main (String args[]) {
      sor s=new sor() ;
      s.betesz(5) ;
      s.betesz(8) ;
      s.betesz(12) ;
      System.out.println(s.kivesz());
      System.out.println(s.kivesz());
      System.out.println(s.kivesz());
}
```

```java
public class sor {
    int elemszam ;
    class elem {
        int ertek;
        elem kovetkezo ;
    }
    elem semmi, eleje ;
    sor() {
        elem q=new elem() ;
        semmi=q ;
        eleje=q ;
        elemszam=0 ;
    }
    void betesz(int x) {
        elem q=new elem() ;
        semmi.kovetkezo=q ;
        semmi.ertek=x ;
        semmi=q ;
        elemszam++ ;
    }
    int kivesz() {
        if (eleje!=semmi) {
            int visszateresi_ertek ;
            visszateresi_ertek=eleje.ertek ;
            eleje=eleje.kovetkezo ;
            elemszam-- ;
            return visszateresi_ertek;
        } else return 0 ;
    }
}
```

Halmaz, Asszociatív tömb

C++ Map

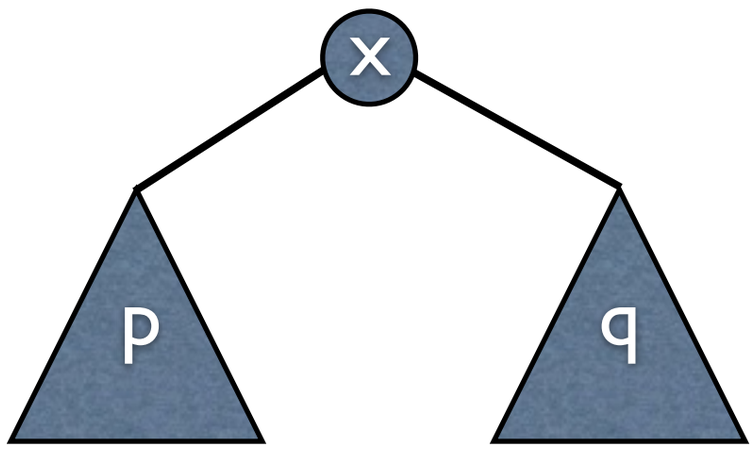
Java Treemap, Hashmap

PHP Array

Bináris keresőfa

Az F = (M,R,Adat) absztrakt adatszerkezetet bináris keresőfának nevezzük, ha

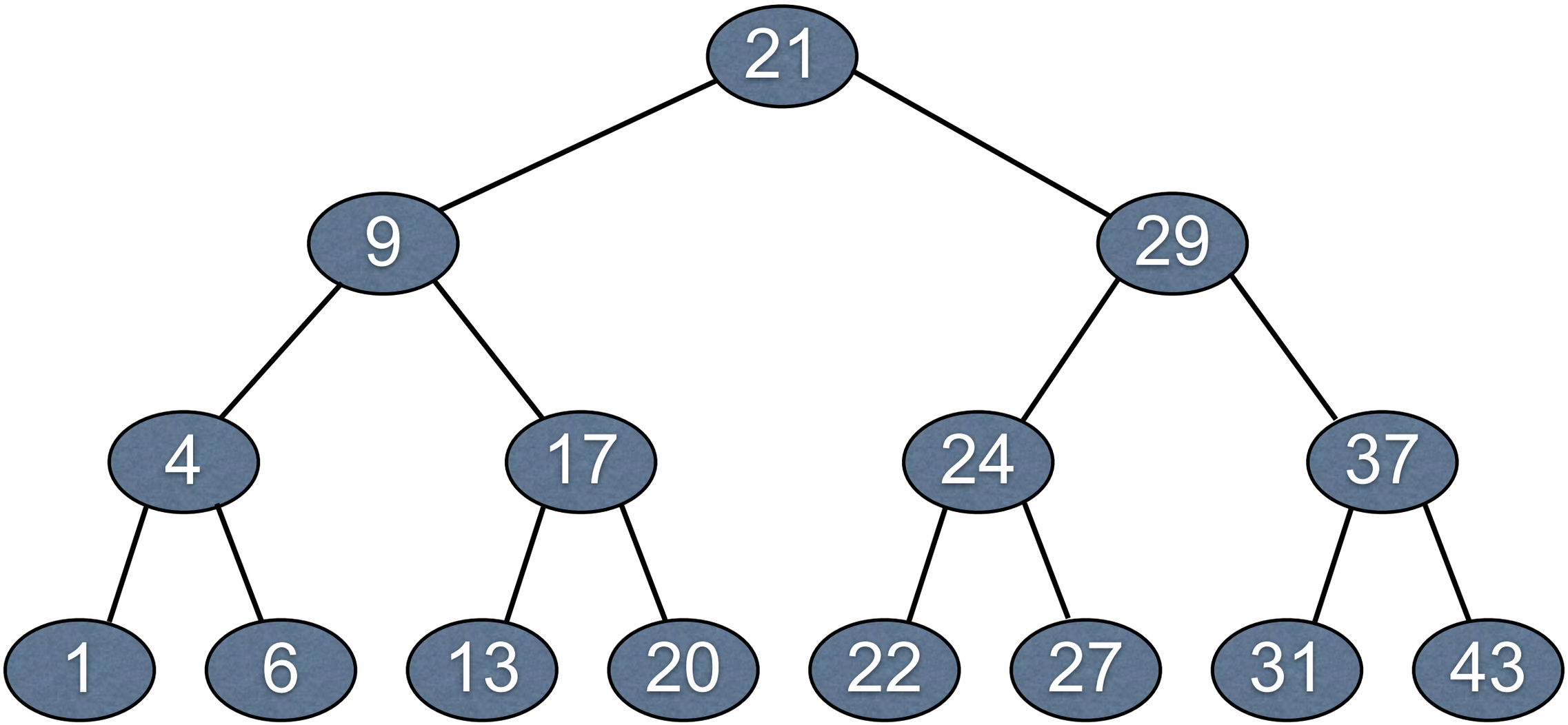
- F bináris fa, $R = \{bal, jobb, apa\}$, $bal, jobb, apa : M \to M$,
- $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
- $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(kulcs(p) \leq kulcs(x) \leq kulcs(q))$



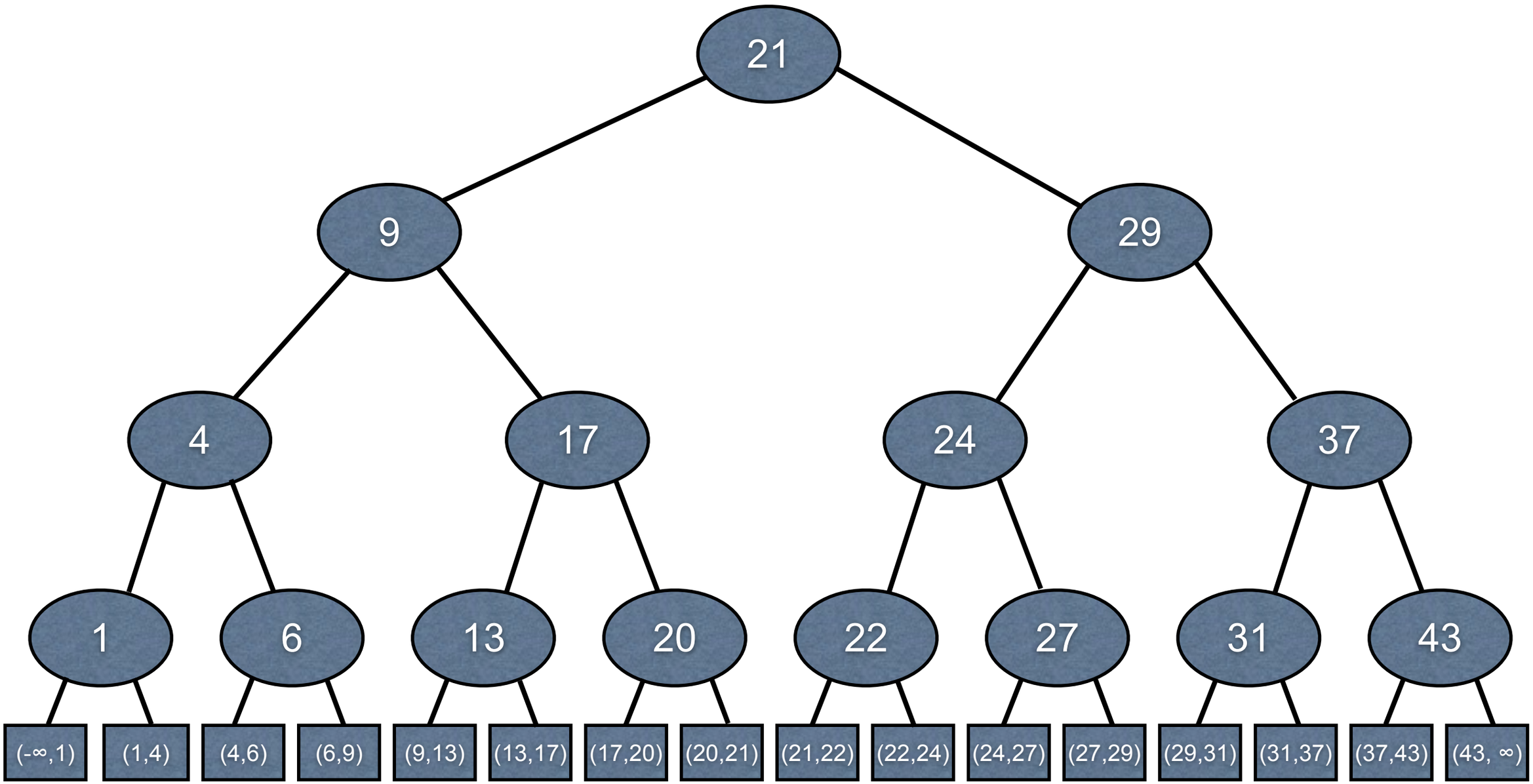
[2] 232-241

fázunk...

Bináris keresőfa



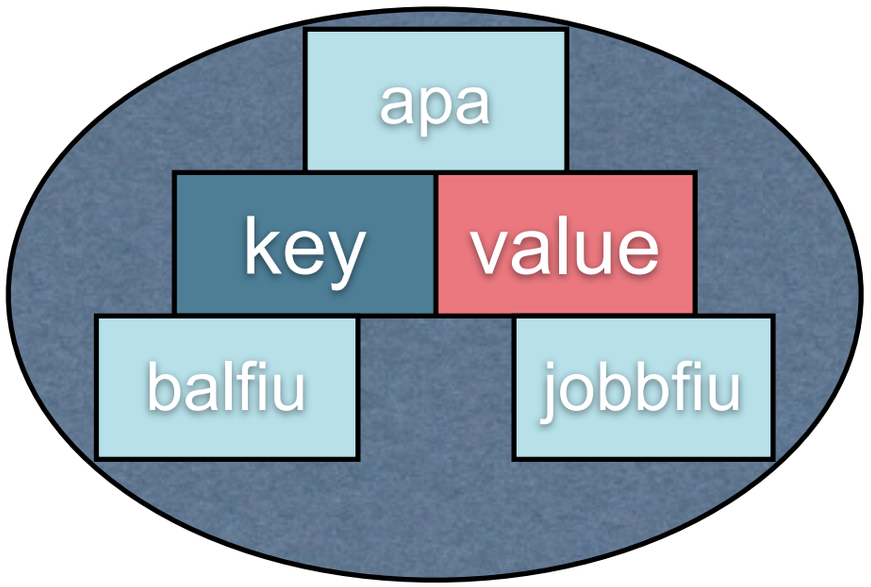
Bináris keresőfa



Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

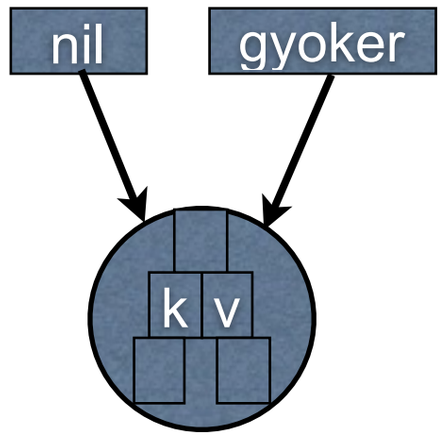
```
struct fapont{
    int key, value ;
    fapont bal, jobb, apa ;
}
```

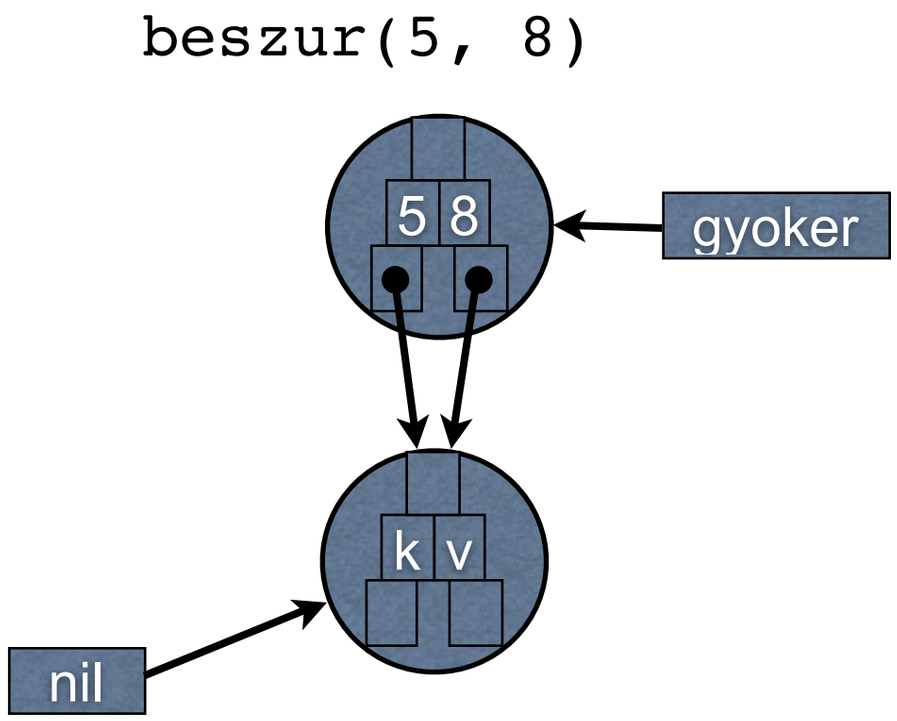


Bináris fa tárolása

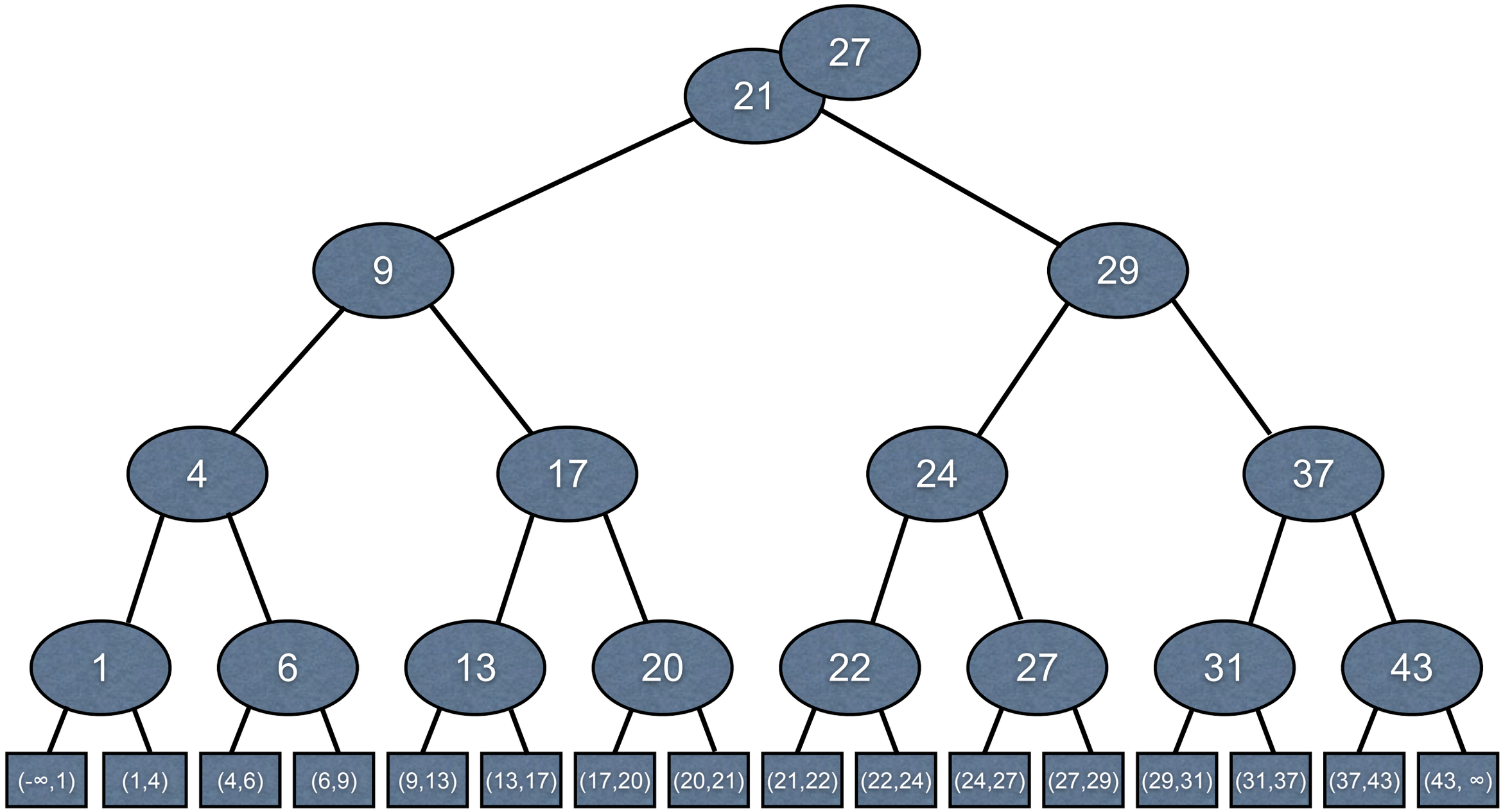
első megközelítés, a gyakorlatban nem így van!

```
public class fa {
   int elemszam ;
   struct fapont{
      int key, value ;
      fapont bal, jobb, apa ;
   }
   fapont nil, gyoker ;
   void fa() {
      fapont q = new fapont() ;
      nil=q ;
      gyoker=q ;
      elemszam=0 ;
   }
   boolean beszur(int x, int v) {
   ...
  }
  ...
}
```

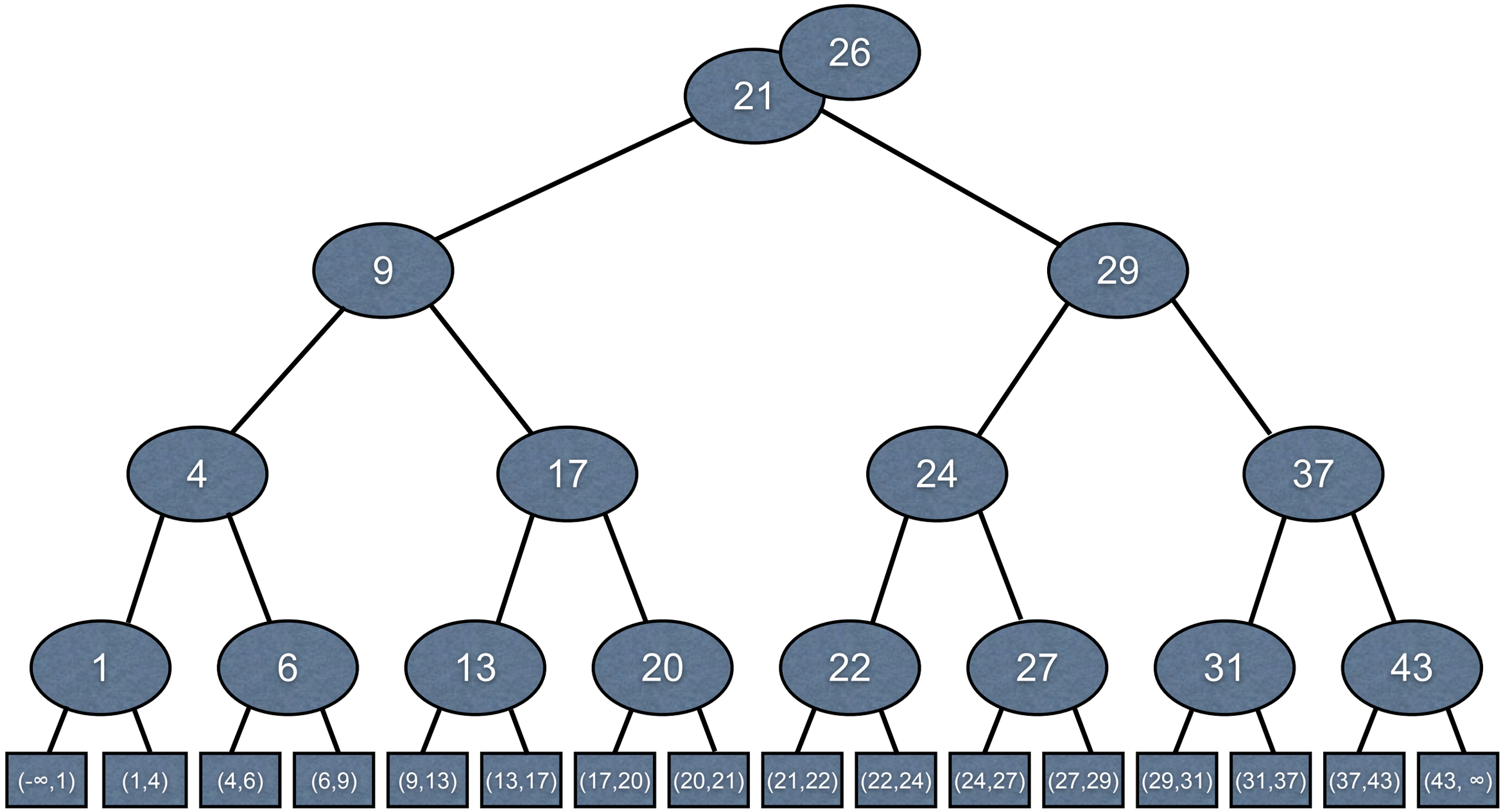




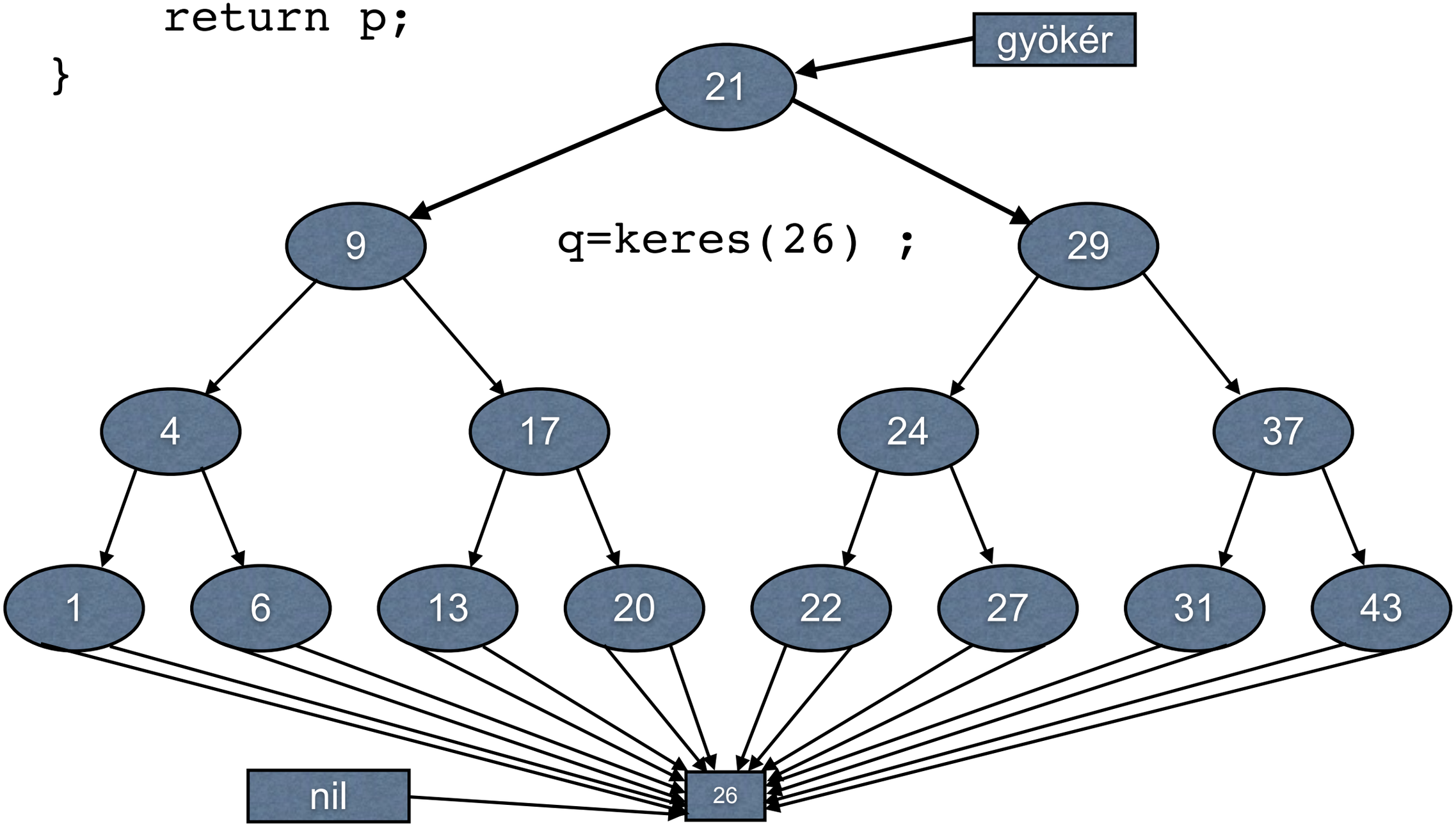
Keresés keresőfában

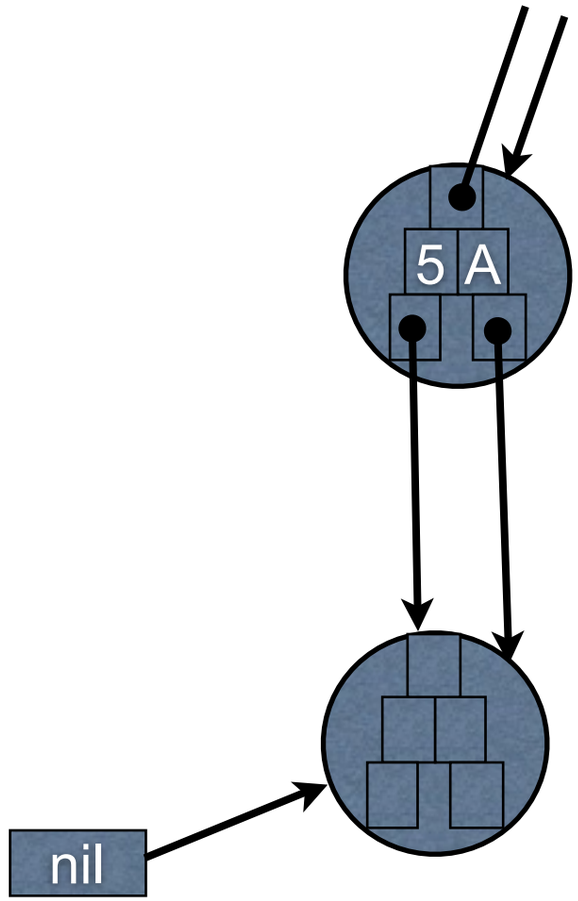


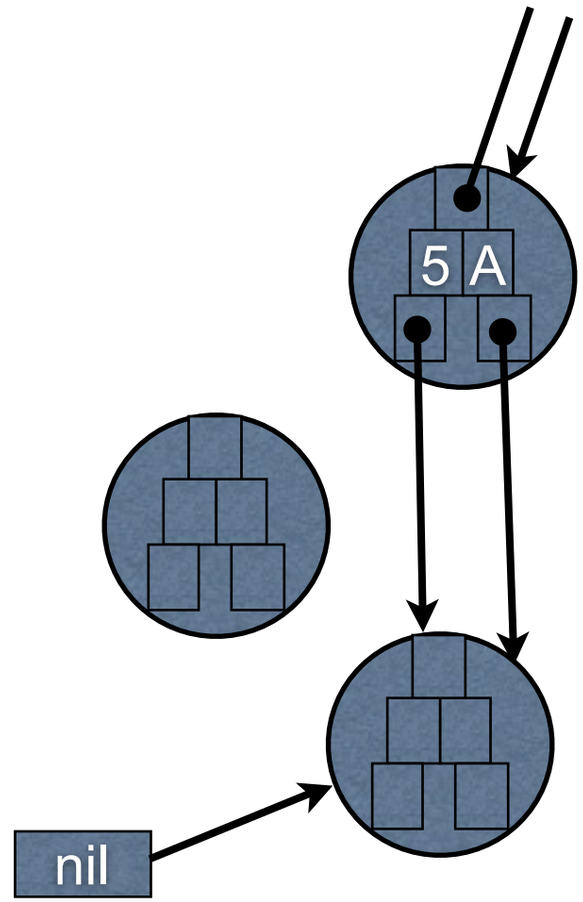
Keresés keresőfában

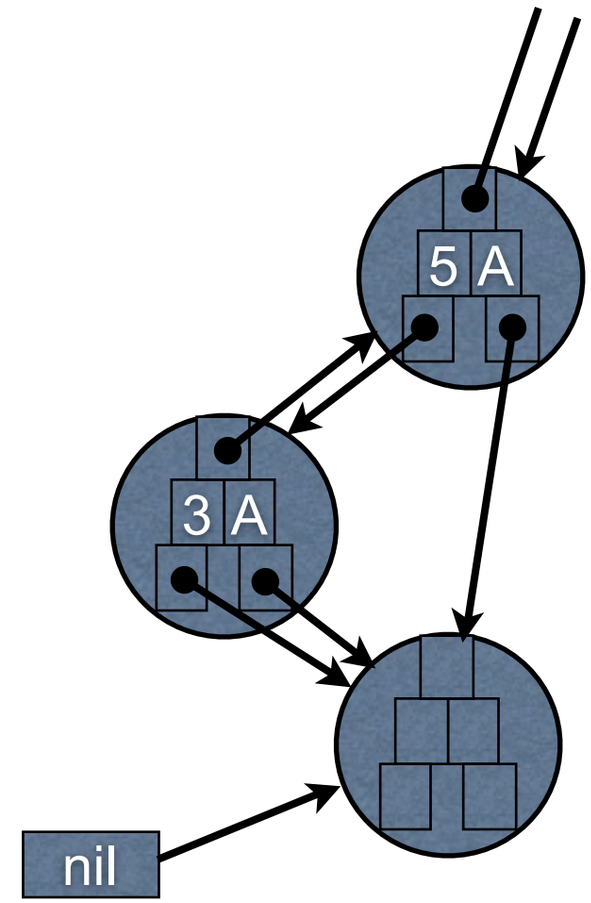


```
fapont keres(int x) {
     fapont p=gyoker ;
     nil.key=x ;
     while (p.key!=x) p=x<p.key?p.bal:p.jobb ;
     return p;
}
```

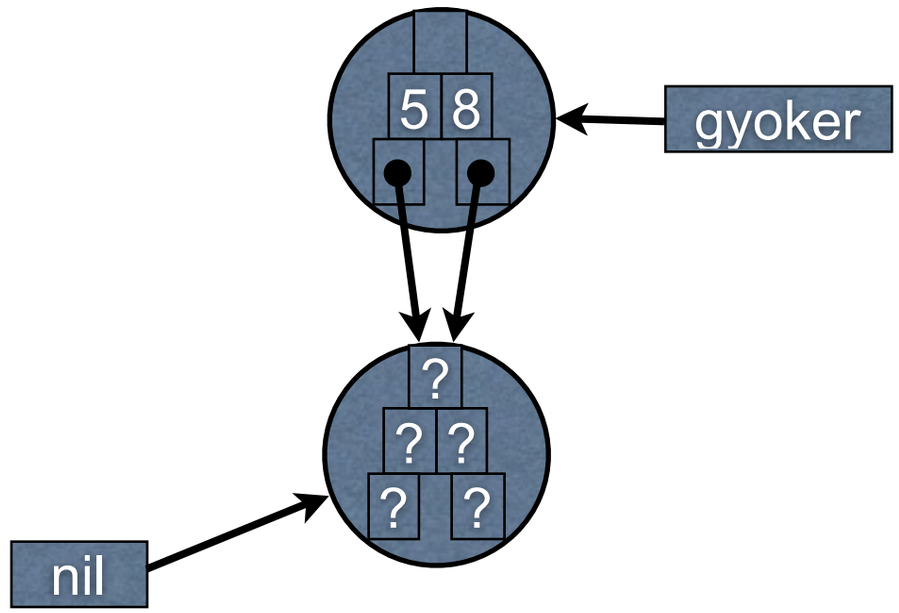



5 A
nil

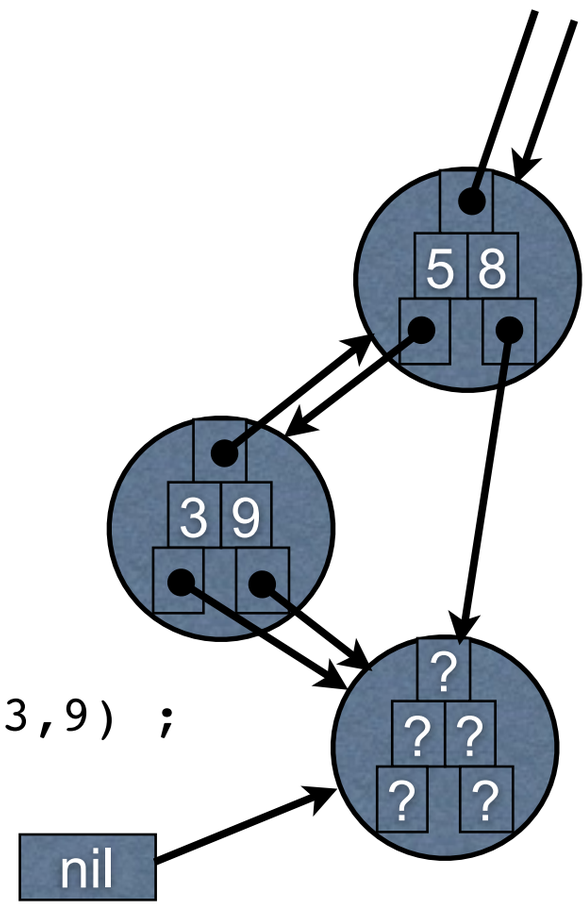

5 A
nil


5 A
3 A
nil

```
boolean beszur(int x, int v) {
      fapont p,papa ;
      p=gyoker ;
      papa=nil ;
      while (p!=nil && p.key!=x) {
         papa=p ;
         if (x<p.key) p=p.bal;
         else p=p.jobb;
      }
      if (p==nil) {
         fapont ujpont = new fapont();
         if (gyoker==p) gyoker=ujpont ;
         ujpont.key=x ;
         ujpont.value=v ;
         ujpont.bal=nil;
         ujpont.jobb=nil;
         ujpont.apa=papa;
         if (papa!=nil) {
            if (x>papa.key) papa.jobb=ujpont;
            else  papa.bal=ujpont;
         }
         elemszam++ ;
         return true ;
      } else {
         p.value=v ;
         return false ;
      }
}
```

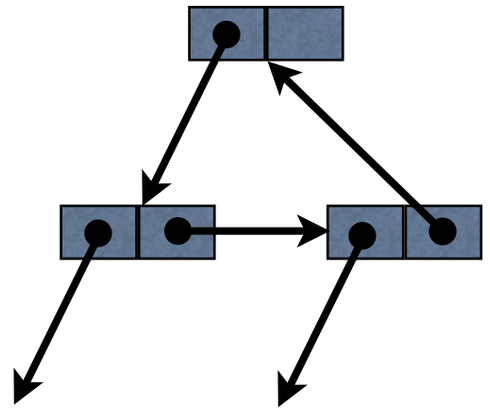


beszur(3,9) ;



Bináris fák tárolása

csúcsonkét 2 pointer és egy bit, mely azt jelöli, hogy bal, vagy jobb gyerek



1. pointer
   - bal gyerek, ha létezik
   - jobb gyerek, ha csak az létezik.
   - nil: ha egyáltalán nincs gyereke.

2. pointer
   - ha ő maga bal gyerek, akkor a testvérre mutat, ha az létezik, különben a szülőre.
   - ha jobb gyerek: a szülőre mutat.
   - gyökérben nil

Fapont tárolása

```
struct fapont<K,V>{
    <K> key ;
    <V> value ;
    fapont fiu, tespa ;
    boolean bal ;
}
```

Műveletek és futási idő igények:

bool *beszúr*(<K> key, <V> value)	**O(*h*)**
fapont *keres*(<K> x)	**O(*h*)**
bool *töröl*(fapont p)	**O(*h*)**
fapont *rákövetkező*(fapont p)	**O(*h*)**
***mindetkiir(*fapont p)**	**O(n)**
mindenttöröl(fapont p)	**O(n)**
int elemszám(fapont p)	**O(*h*)**
int elemszám()	**O(1)**

Bináris keresőfák jellemzői

- p gyökerű részfa belső pontjainak száma

```
int s(fapont p) {
   if (p==nil) return 0 ;
   return 1+s(p.bal)+s(p.jobb) ;
}
```

O(n)

- p gyökerű részfa magassága

```
int h(fapont p) {
   if (p==nil) return 0 ;
   return 1+Math.max(h(p.bal),h(p.jobb)) ;
}
```

O(n)

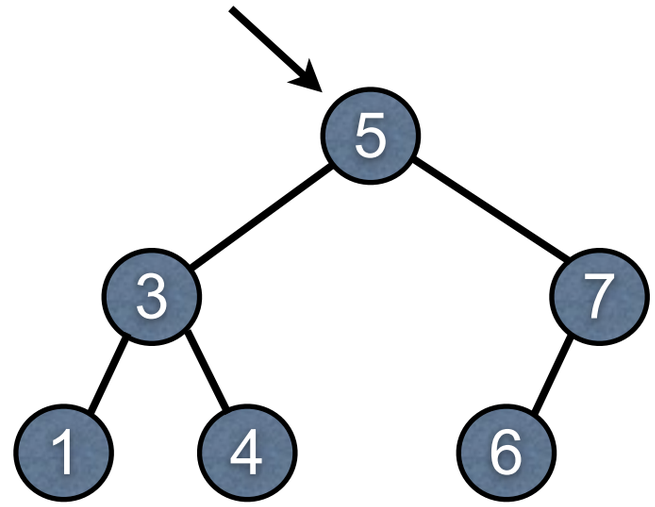
Bináris fa inorder bejárása

```
function bejar(p){
    if (p.bal!=nil) bejar(p.bal) ;
    muvelet(p) ;
    if (p.jobb!=nil) bejar(p.jobb) ;
}
```

O(n)

5

```
function bejar(p){
      if (p.bal!=nil) bejar(p.bal) ;
      muvelet(p) ;
      if (p.jobb!=nil) bejar(p.jobb) ;
}
```



3 7

```
function bejar(p){
      if (p.bal!=nil) bejar(p.bal) ;
      muvelet(p) ;
      if (p.jobb!=nil) bejar(p.jobb) ;
}
```

146

```
function bejar(p){
      if (p.bal!=nil) bejar(p.bal) ;
      muvelet(p) ;
      if (p.jobb!=nil) bejar(p.jobb) ;
}
```

1 3 4 5 6 7

Preorder - Inorder - Postorder bejárás

```
function bejar(p){ //preorder
     muvelet(p) ;
     if (p.bal!=nil) bejar(p.bal) ;
     if (p.jobb!=nil) bejar(p.jobb) ;
}
```

```
function bejar(p){ //inorder
     if (p.bal!=nil) bejar(p.bal) ;
     muvelet(p) ;
     if (p.jobb!=nil) bejar(p.jobb) ;
}
```

```
function bejar(p){ //postorder
     if (p.bal!=nil) bejar(p.bal) ;
     if (p.jobb!=nil) bejar(p.jobb) ;
     muvelet(p) ;
}
```

O(n)

Bejárás - csökkenő sorrendben

```
function bejar(p){ //inorder rev.
        if (p.jobb!=nil) bejar(p.jobb) ;
        muvelet(p) ;
        if (p.bal!=nil) bejar(p.bal) ;
}
```