Algoritmusok és Adatszerkezetek II.

előadás

Felelős tanszék: Számítógépes algoritmusok és mesterséges intelligencia tanszék

Nappali tagozaton:

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

Gyakorlat: heti 1 óra / 1kredit. Teljesítés módja: Aláírás.

A kurzus felvételének előfeltételei : Algoritmusok és adatszerkezetek I. vagy Algoritmizálás

- objektumok struktúrája a memóriában, adatszerkezetek tervezésének alapjai
- a rendezett halmaz adattípus és megvalósítása ugrólistával
- diszjunkt halmazok kezelése, az UnioHolvan adattípus és alkalmazása
- keresőfák és kiegyensúlyozott keresőfák: AVL -, piros-fekete -, 2-3 - és B-fák
- önszervező keresőfák, amortizációs költségelemzés, hasítótáblák
- a prioritási sor és kupac, egyesíthető prioritási sor binomiális és Fibonacci kupaccal
- geometriai algoritmusok
- mintaillesztés, a Knuth-Morris-Pratt algoritmus
- probléma megoldási módszerek II, BT, B&B
- véletlenített algoritmusok, közelítő algoritmusok, on-line algoritmusok
- számelméleti algoritmusok, nyilvános kulcsú titkosítás
- hálózatelemzési modellek és módszerek

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 számítógépprogramozás 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)*

A gyakorlat (gyakorlati jegy) teljesítésének feltételei

A félév során 4 db kis ZH és 2db nagy ZH lesz az alábbi menetrend szerint:
- 3. hét: kis
- 5. hét: kis
- 7. hét: nagy
- 9. hét: kis
- 11. hét: kis
-13. hét: nagy
-14. hét: nagy javító)

A nagy ZH-k egyenként 40-40, a kis ZH-k 5-5 pontosak. A nagy ZH-kból külön-külön el kell érni minimum 40%-ot, ennek teljesülése esetén a gyakorlati jegy a félév során elért összpontszám alapján a következő ponthatárok szerint alakul:

85-100	5
75-84	4
63-74	3
50-62	2
0-49	1

Lehet továbbá pluszpontot szerezni (a félév során hallgatónként maximum 10-et), ami nem számítható bele a 2-es eléréséhez szükséges minimumpontszámba. Pluszponthoz órai munkával, vagy a gyakorlatvezetővel egyeztetett szorgalmi feladatok megoldásával lehet jutni.

A kurzus teljesítésének feltételei:

gyakorlati_jegy != 1

Kollokvium

- Elérhető maximális összpontszám: 100
- Teljesítendő minimális összpontszám: 50

A kollokvium 3 részből áll:

- A. 5 kérdés a kurzus anyagát lefedő témakörökből
  - Elérhető maximális pontszám: 50
  - Teljesítendő minimális pontszám: 20
- B. Teljesen kidolgozandó tétel
  - Elérhető maximális pontszám: 20
  - Teljesítendő minimális pontszám: 8
- C. Feladatok megoldása
  - Elérhető maximális pontszám: 30
  - Teljesítendő minimális pontszám: 15

A kurzus teljesítésének értékelése:

- 85 - jeles (5)
- 70 - 84 jó (4)
- 60 - 69 közepes (3)
- 50 - 59 elégséges (2)
- 0 - 49 elégtelen (1)

Adatszerkezetek

Absztrakt adattípus (ADT)

Absztrakt adatszerkezet (ADS)

Reprezentáció (ábrázolás)

Implementáció (fejlesztés, programnyelvi megvalósítás)

Fizikai (memória) szint

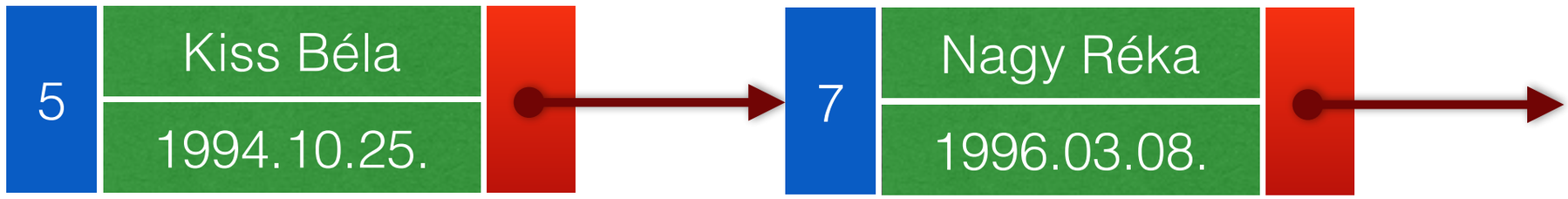
Adat tárolása

kulcsok

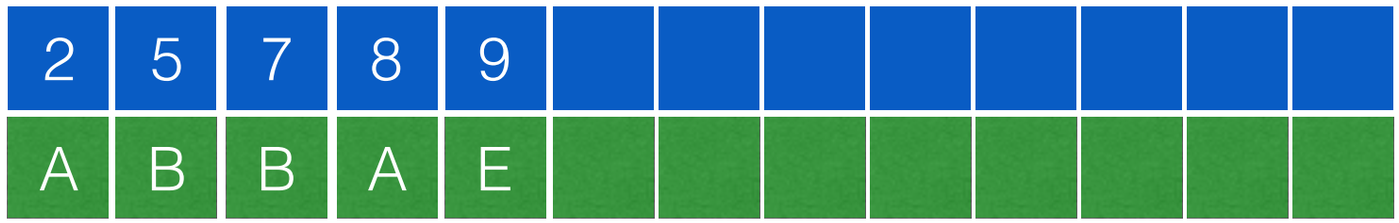
adattagok

szerkezeti elemek

Dinamikus



Statikus



műveletek

létrehozás

adat felvétele

adat keresése

adat módosítása

minden adat törlése (üresít)

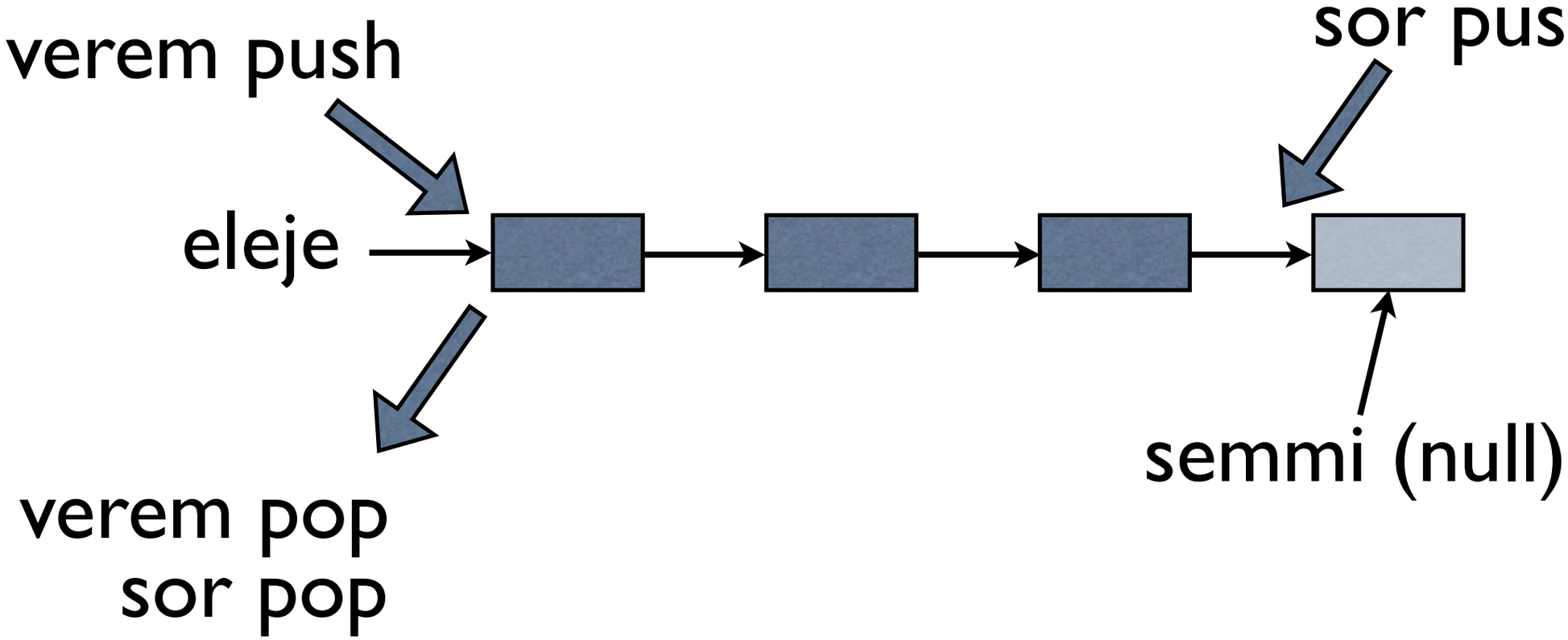
elemszám visszaadása

adat törlése

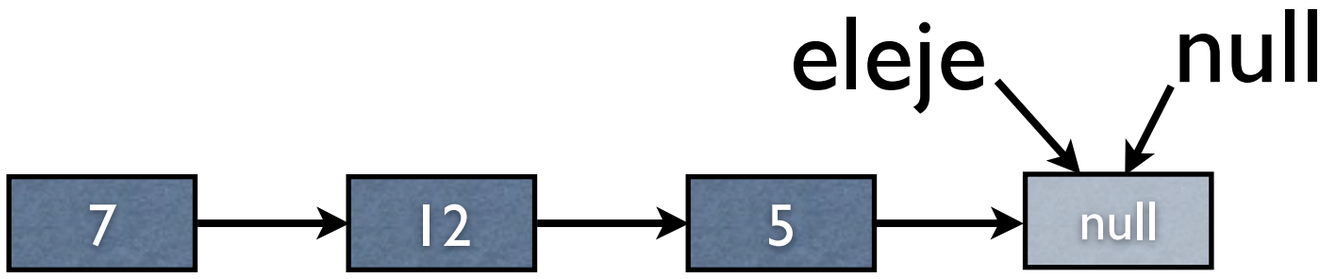
adatszerkezet felszámolása (megszüntet)

Verem, sor megvalósítása

alapműveletek: push, pop



Verem (Stack) példa



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

pop() pop() pop()

```
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());
}
```

```
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()

```
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());
}
```

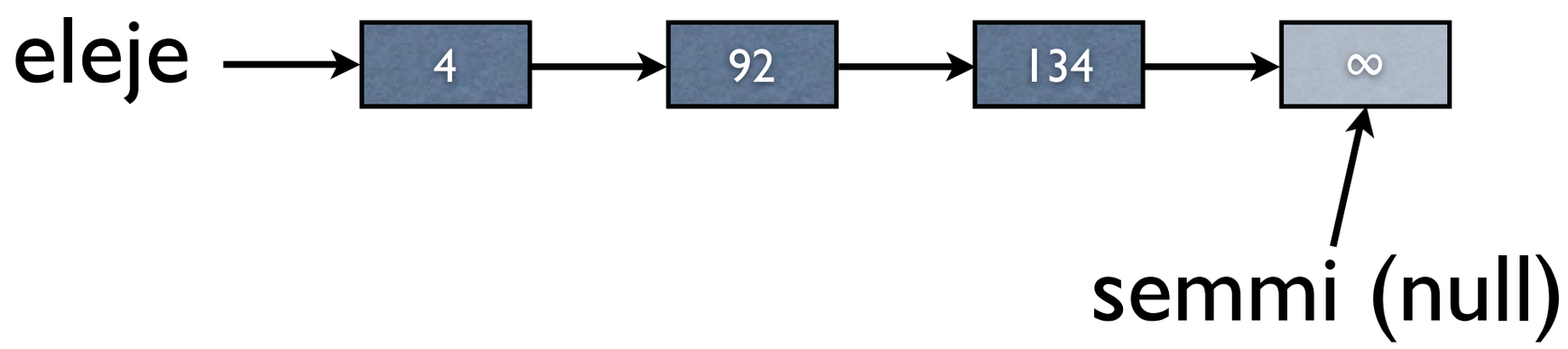
```
public class sor {
    int elemszam ;
    class elem {
        int ertek;
        elem kovetkezo ;
    }
    elem semmi, eleje ;
    sor() {
        elem q=new elem() ;
        semmi=q ;
        eleje=q ;
        elemszam=0 ;
    }
    void betesz(int x) {
        elem q=new elem() ;
        semmi.kovetkezo=q ;
        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 ;
    }
}
```

Láncolt lista

Halmaz műveletek

Rendezett lista

Keresés, beszúrás, törlés: O(n)



Ugrólisták

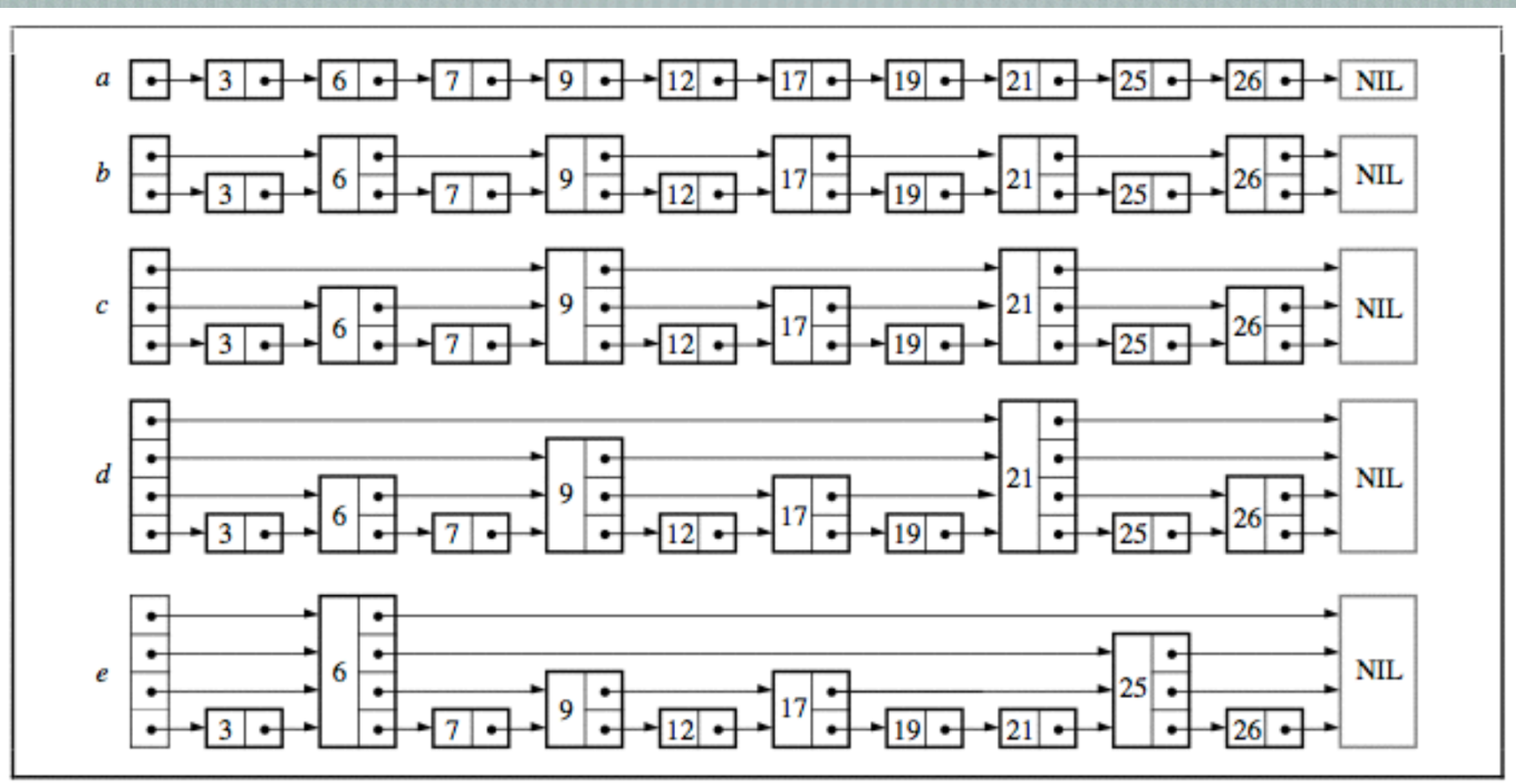


FIGURE 1 - Linked lists with additional pointers

Ugrólisták

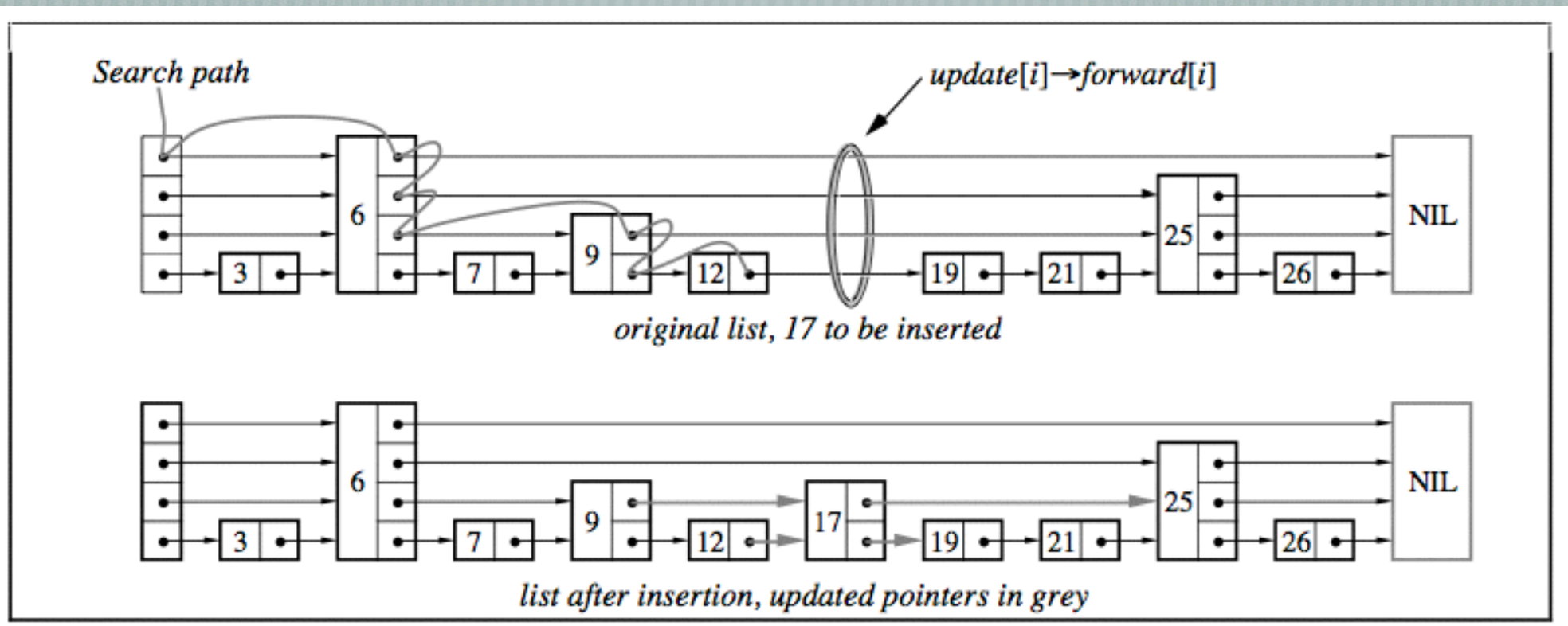


FIGURE 3 - Pictorial description of steps involved in performing an insertion

Ugrólisták

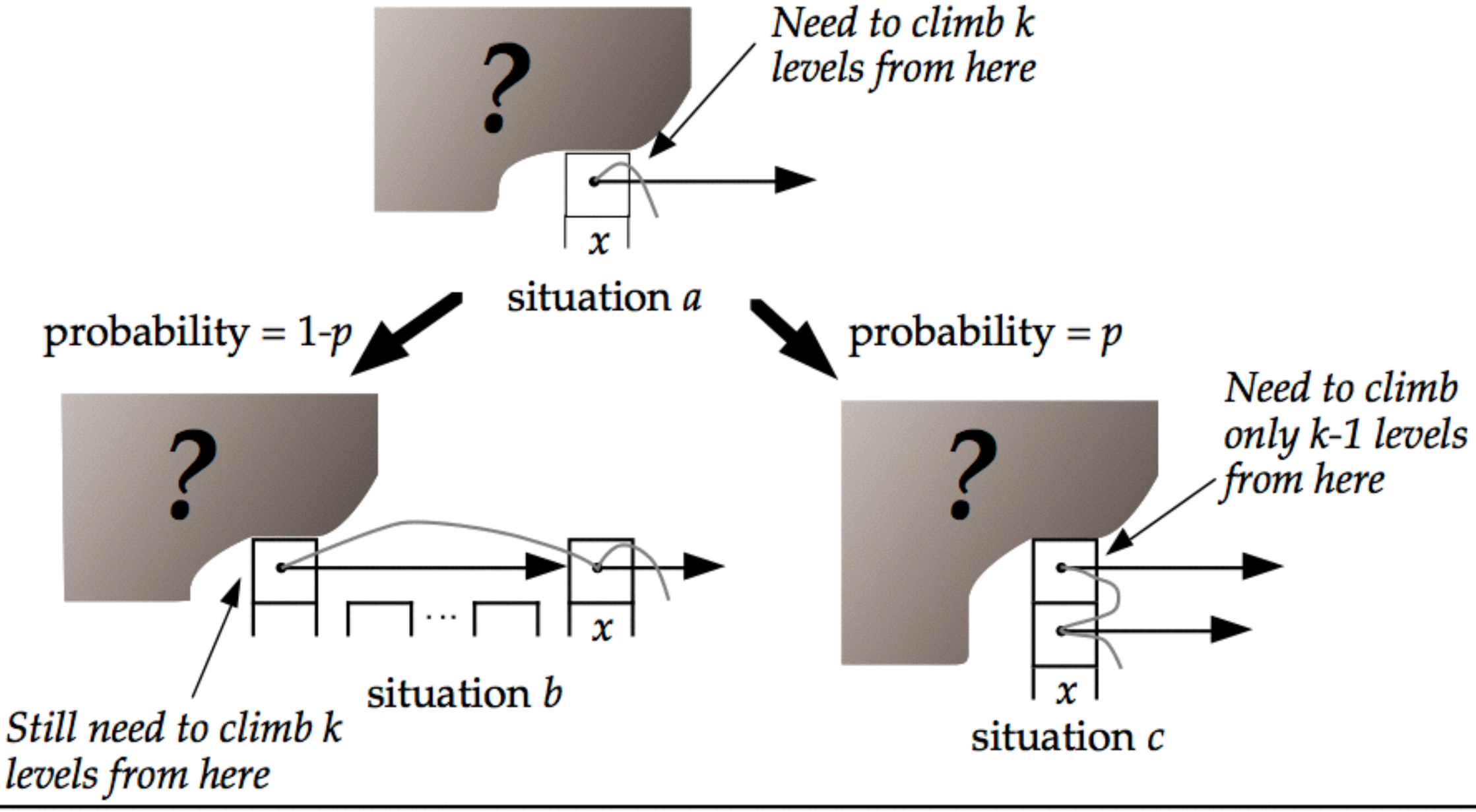
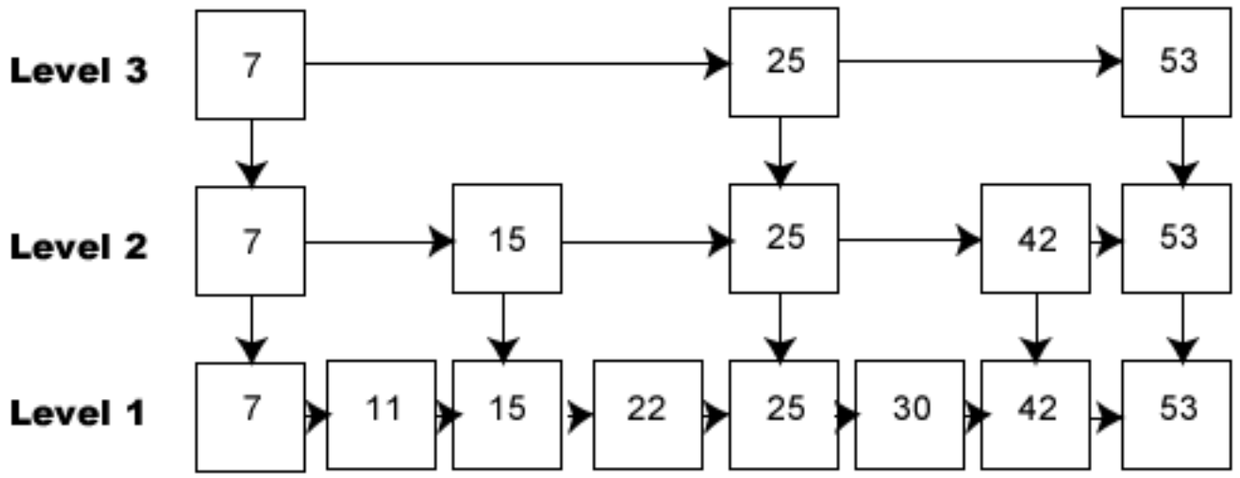
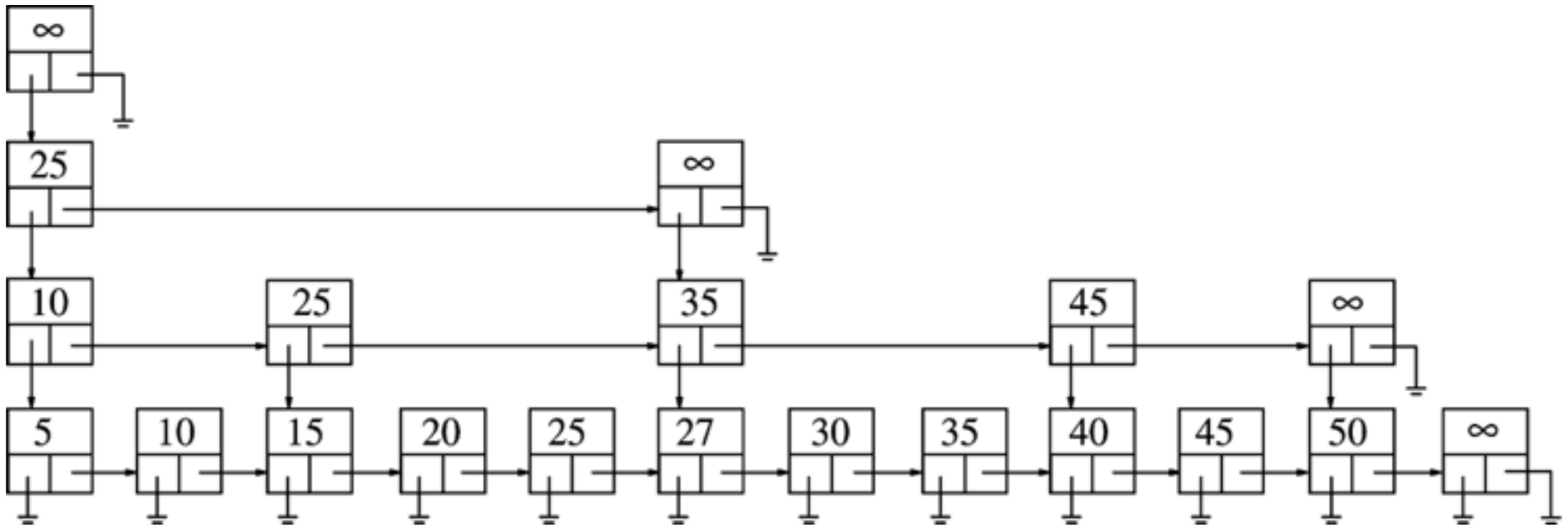


FIGURE 6 - Possible situations in backwards traversal of the search path


Level 3
7
25
53
Level 2
7
15
25
42
53
Level 1
7
11
15
22
25
30
42
53


∞
25
∞
10
25
35
45
∞
5
10
15
20
25
27
30
35
40
45
50
∞

Halmazok egyesítése

Kruskal algoritmus

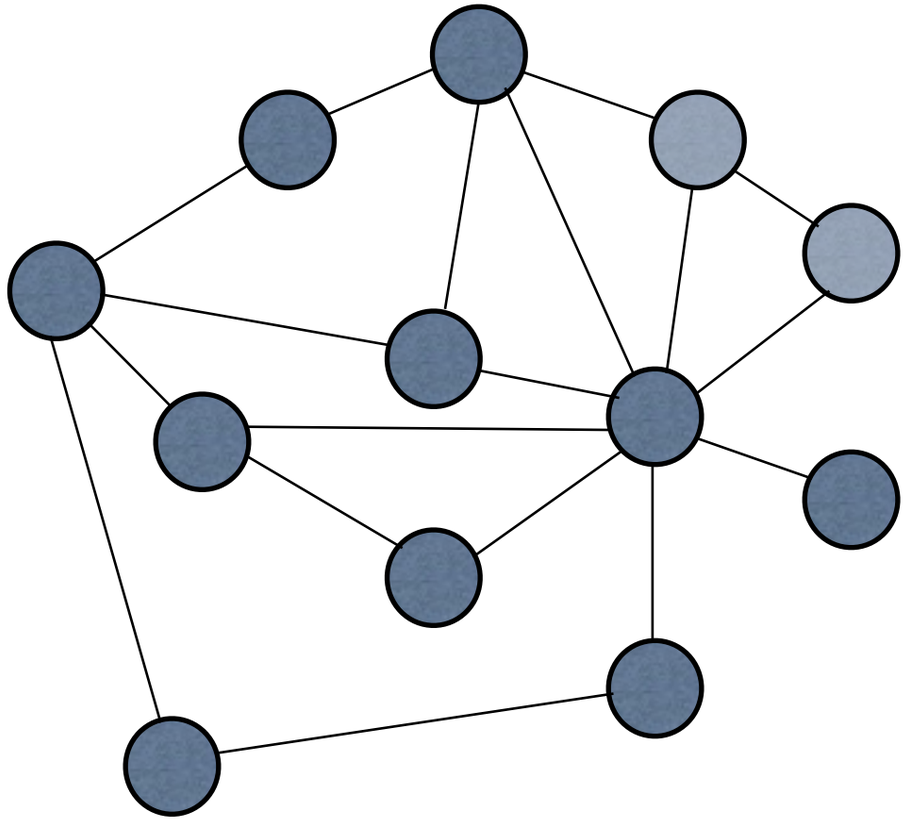
felelevenítés

```
kezdetben a gráf minden pontja külön halmazban van ;
vegyük a gráfból az éleket növekvő sorrendben {
   ha az él 2 végpontja külön halmazban van {
      az él legyen része a minimális feszítőfának ;
      a 2 végpontot tartalmazó halmazt egyesítsük ;
   }
}
```

Halmazok egyesítése

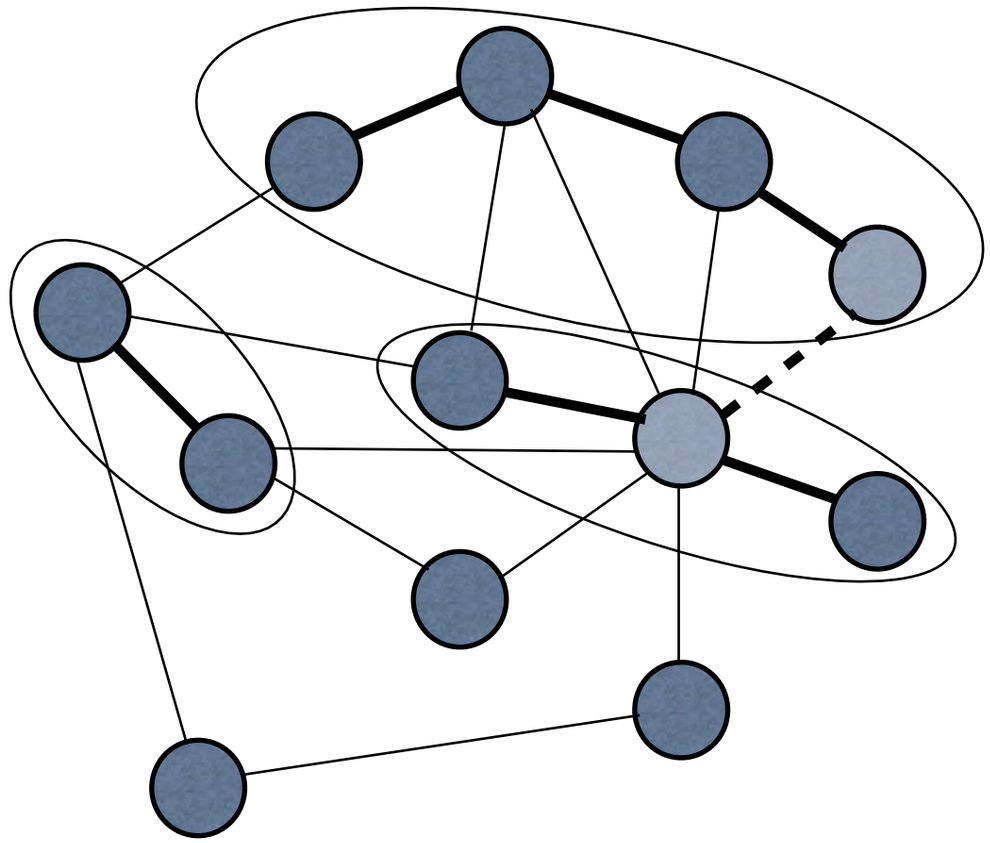
Kruskal algoritmus

felelevenítés



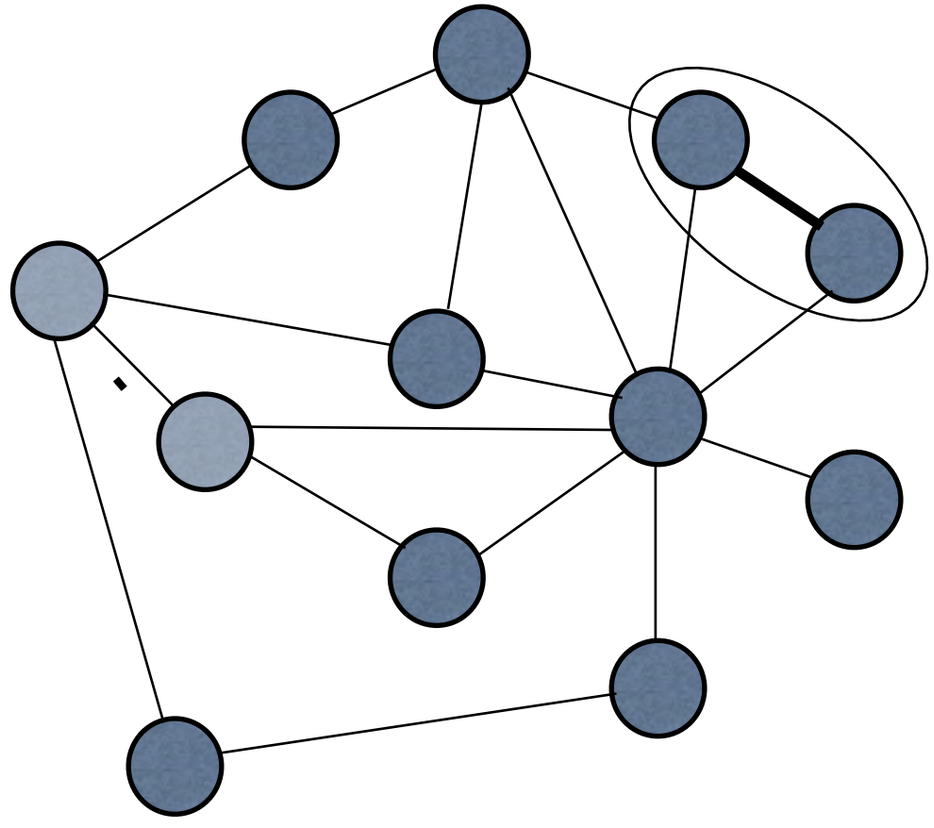
Honnan lehet tudni, hogy a következő él beválasztásával a feszítőerdő erdő (azaz körmentes) marad?

Példa



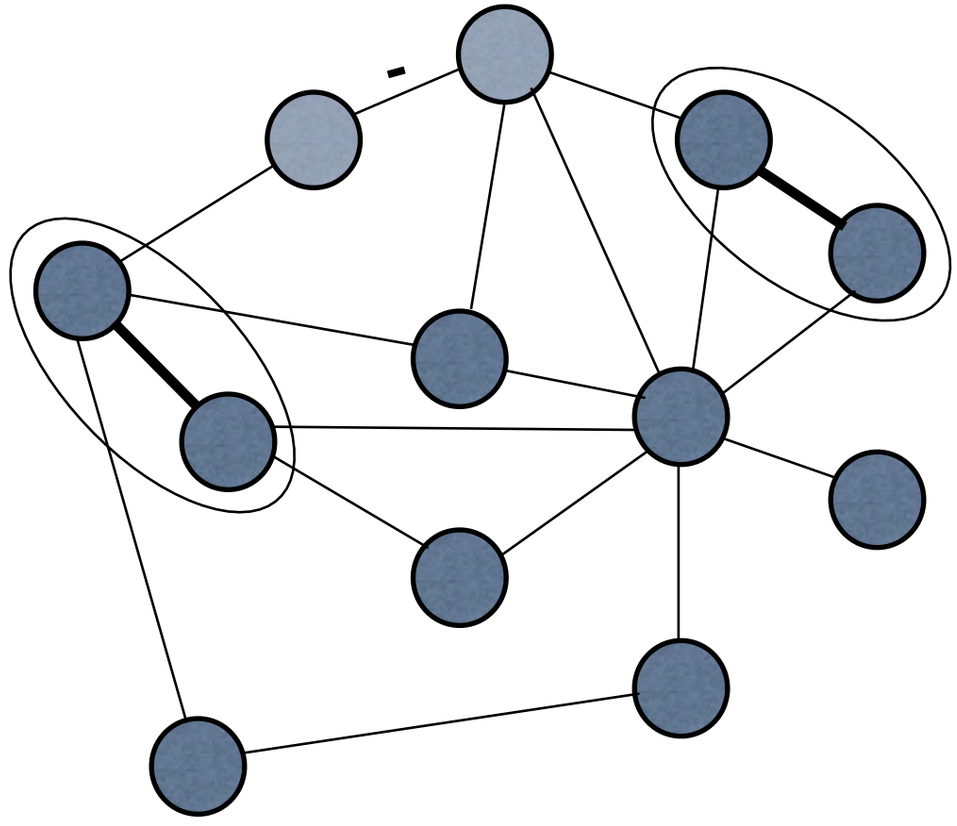
Kruskal algoritmus

felelevenítés



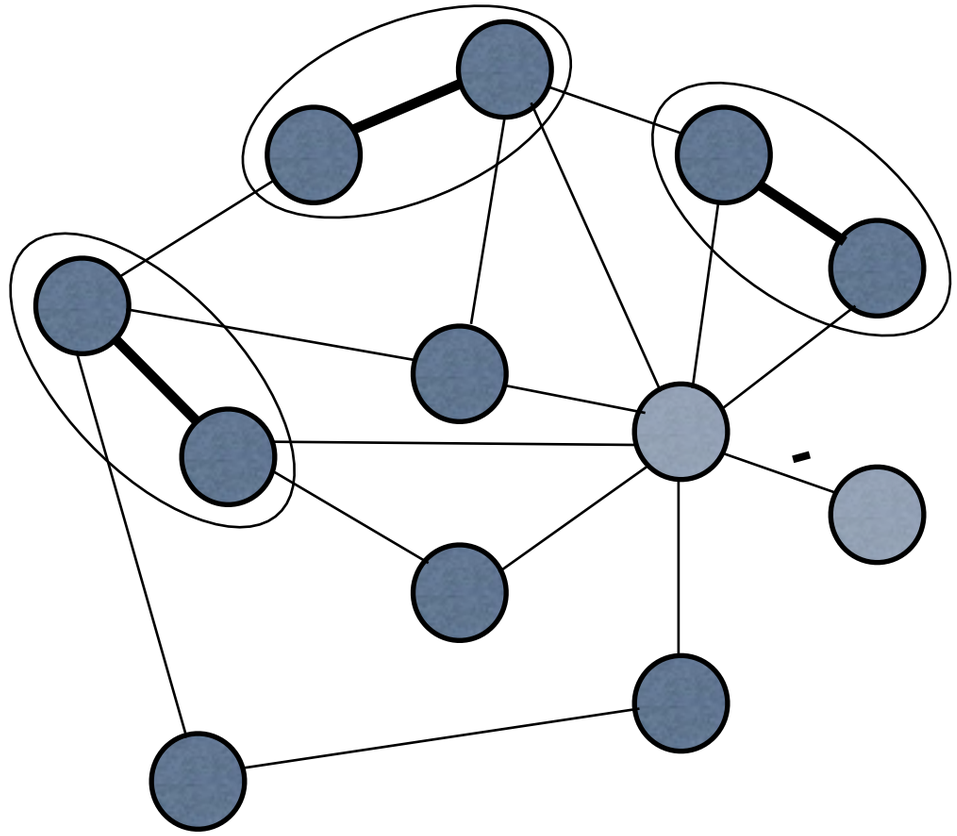
Kruskal algoritmus

felelevenítés



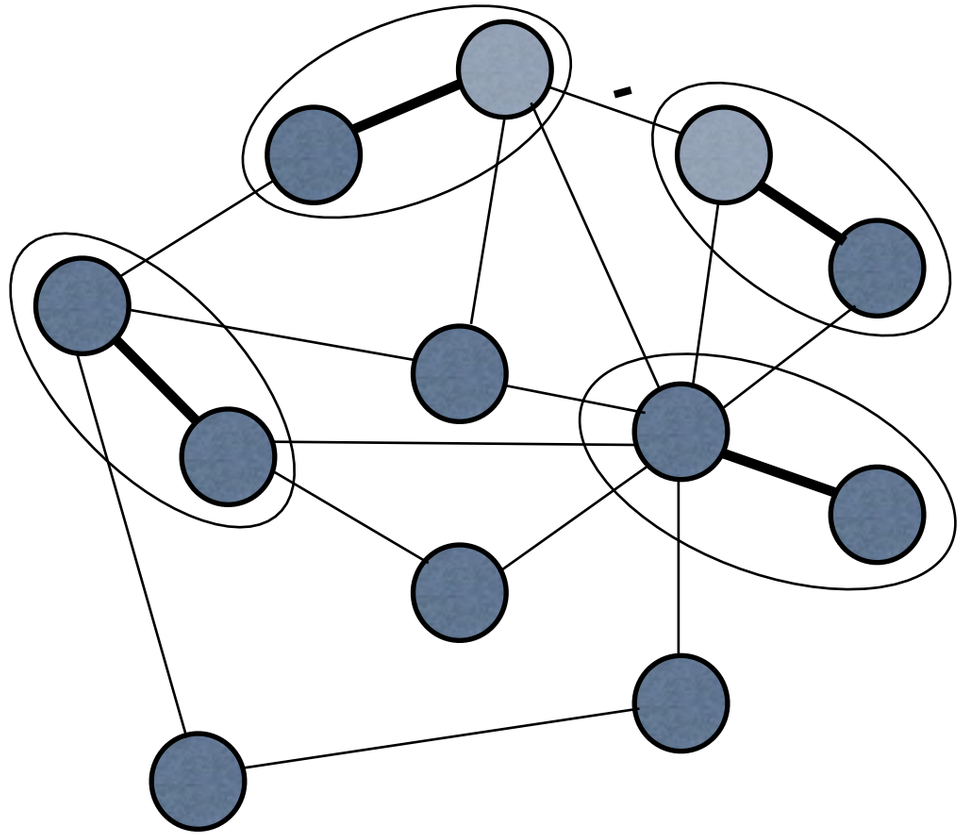
Kruskal algoritmus

felelevenítés



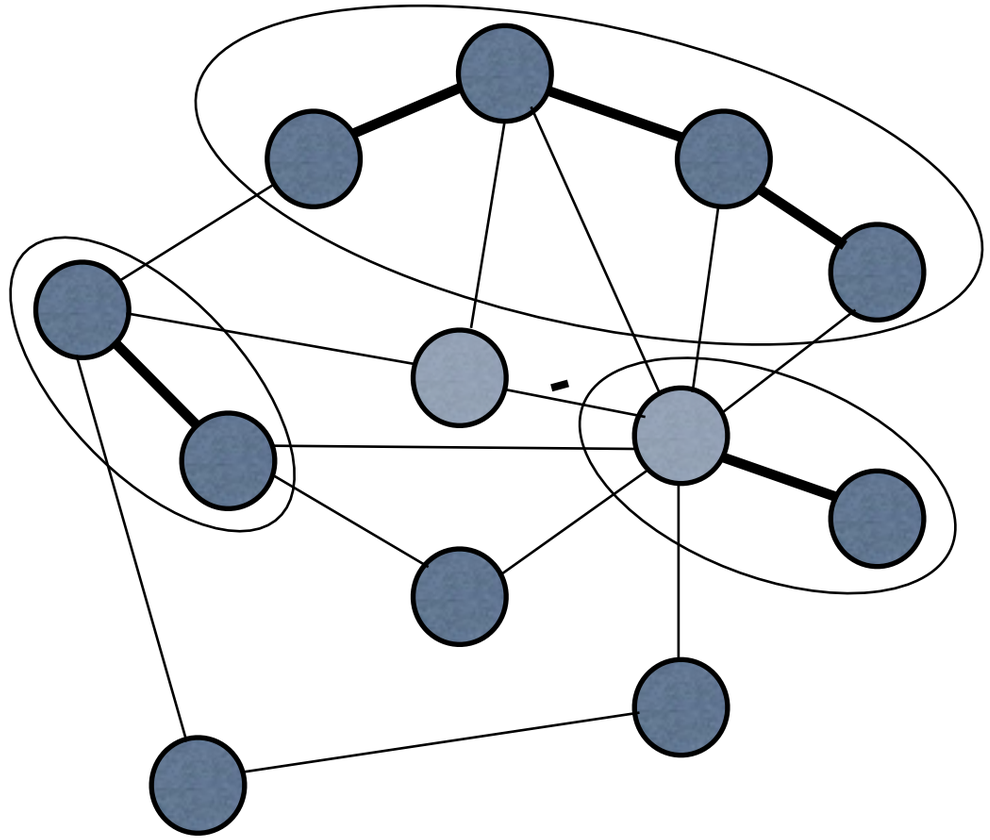
Kruskal algoritmus

felelevenítés



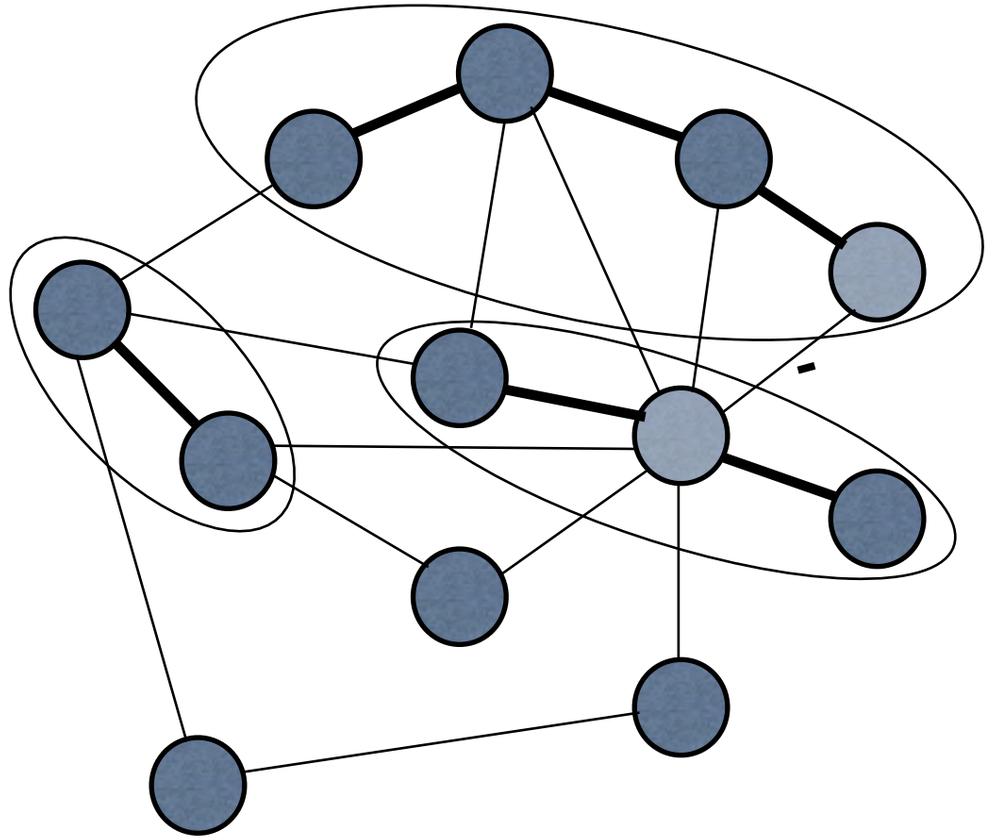
Kruskal algoritmus

felelevenítés



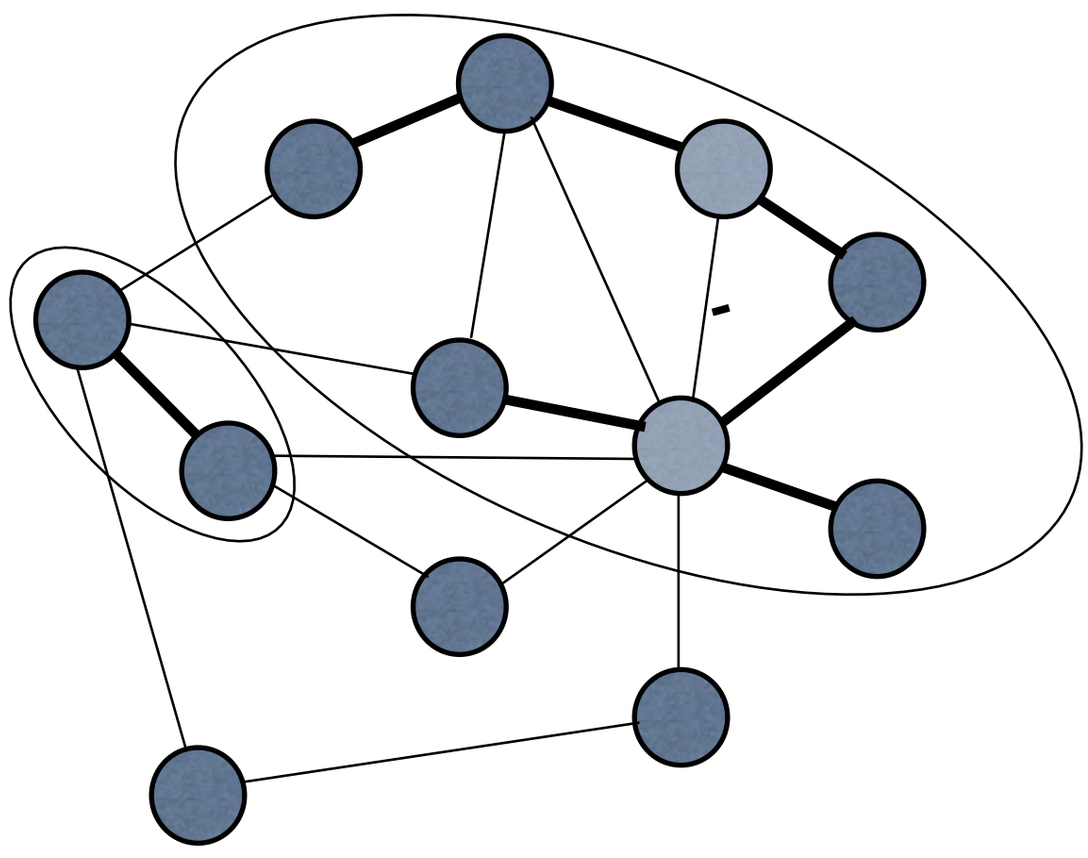
Kruskal algoritmus

felelevenítés



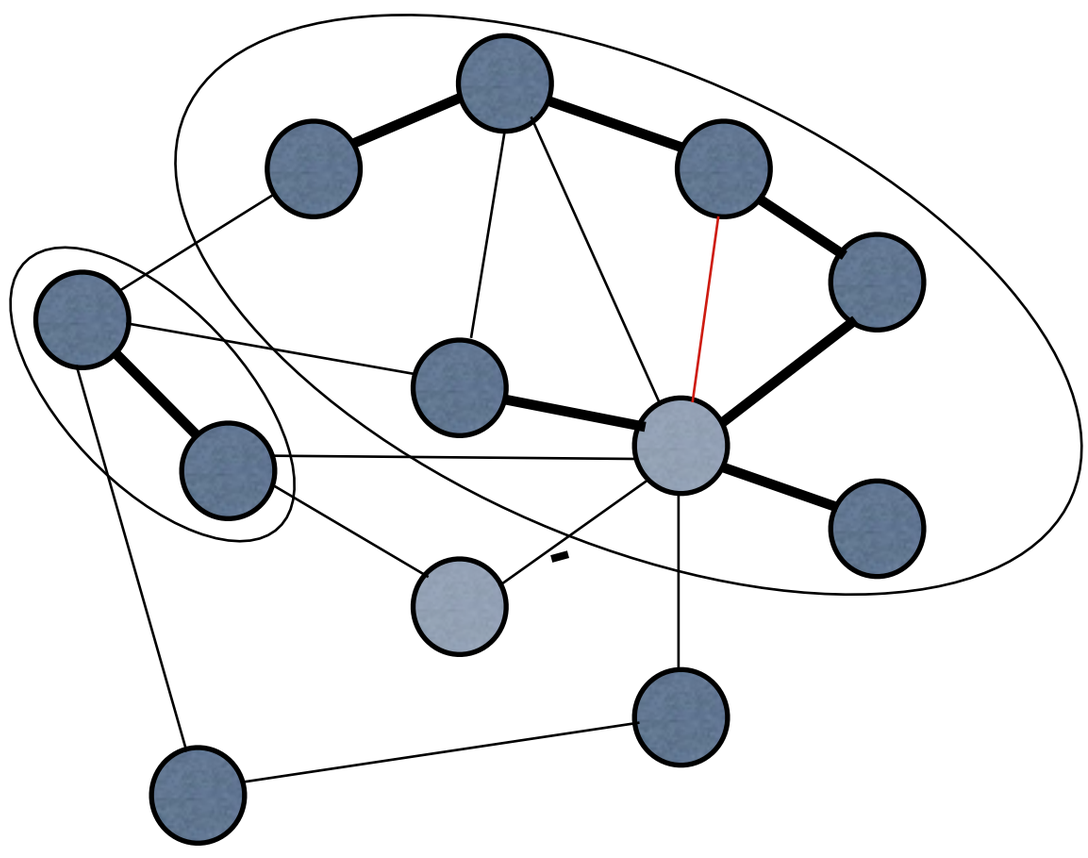
Kruskal algoritmus

felelevenítés



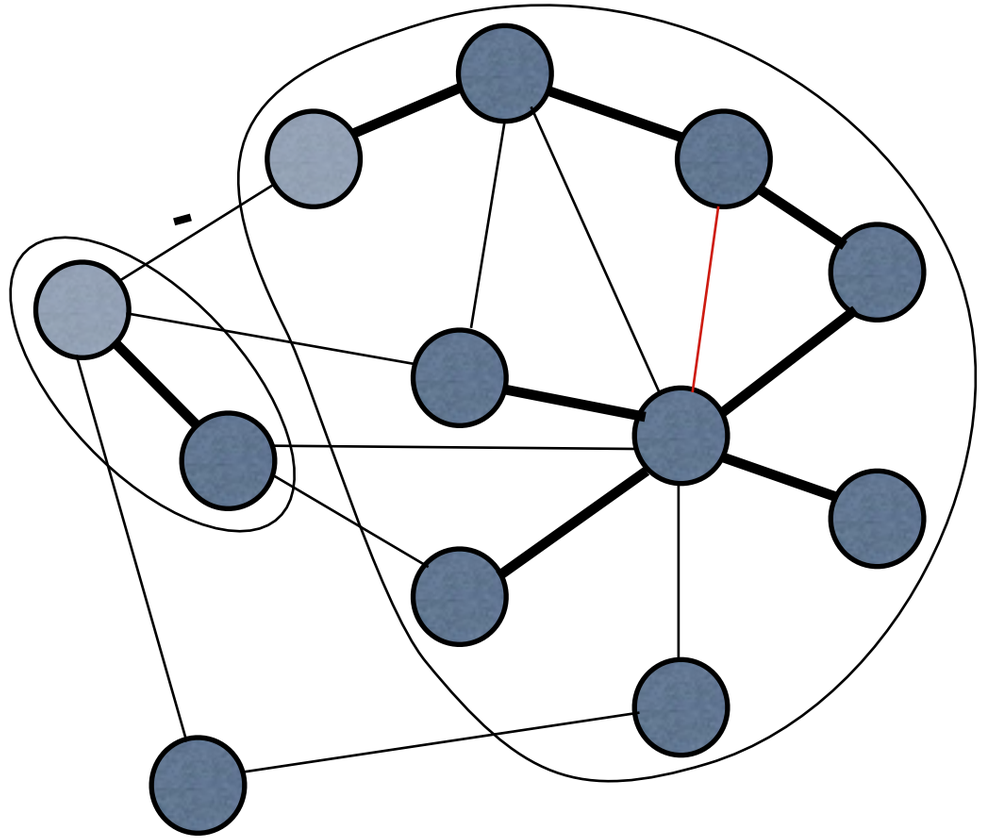
Kruskal algoritmus

felelevenítés



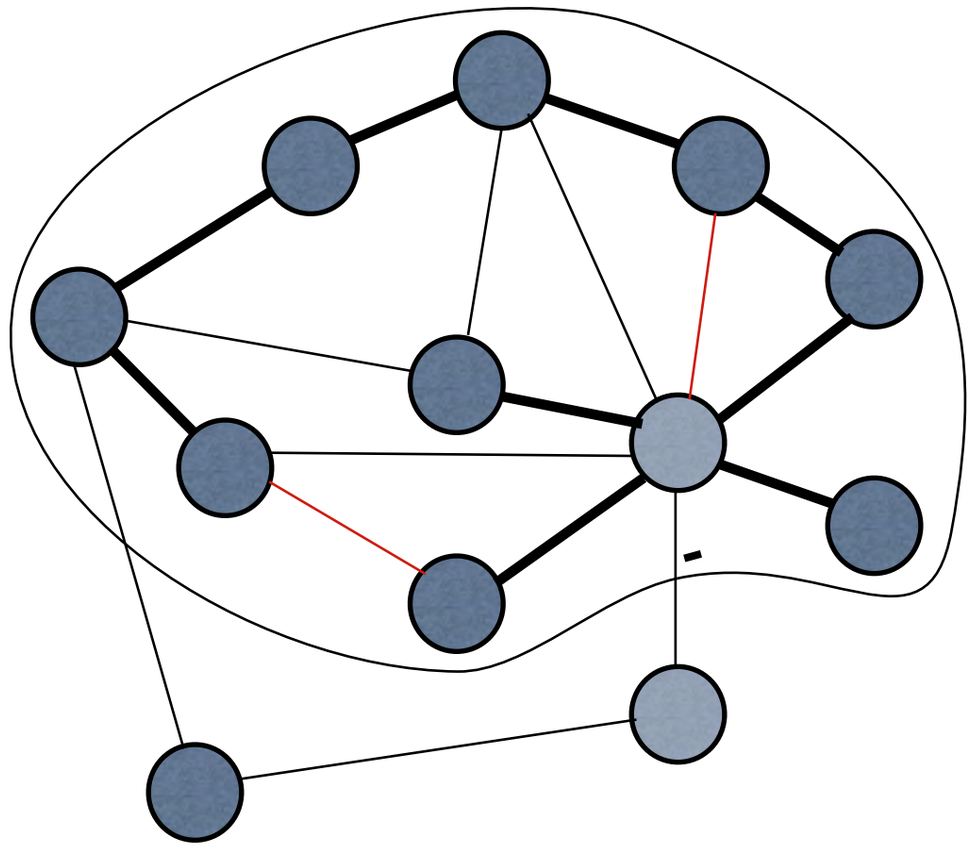
Kruskal algoritmus

felelevenítés



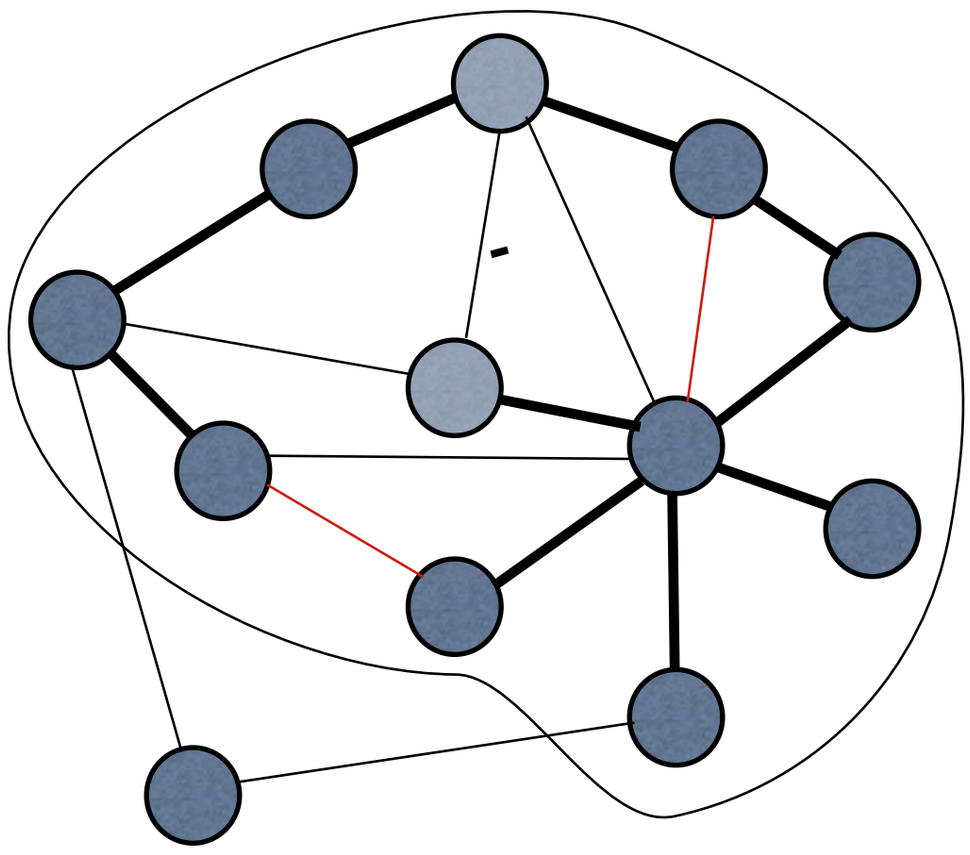
Kruskal algoritmus

felelevenítés



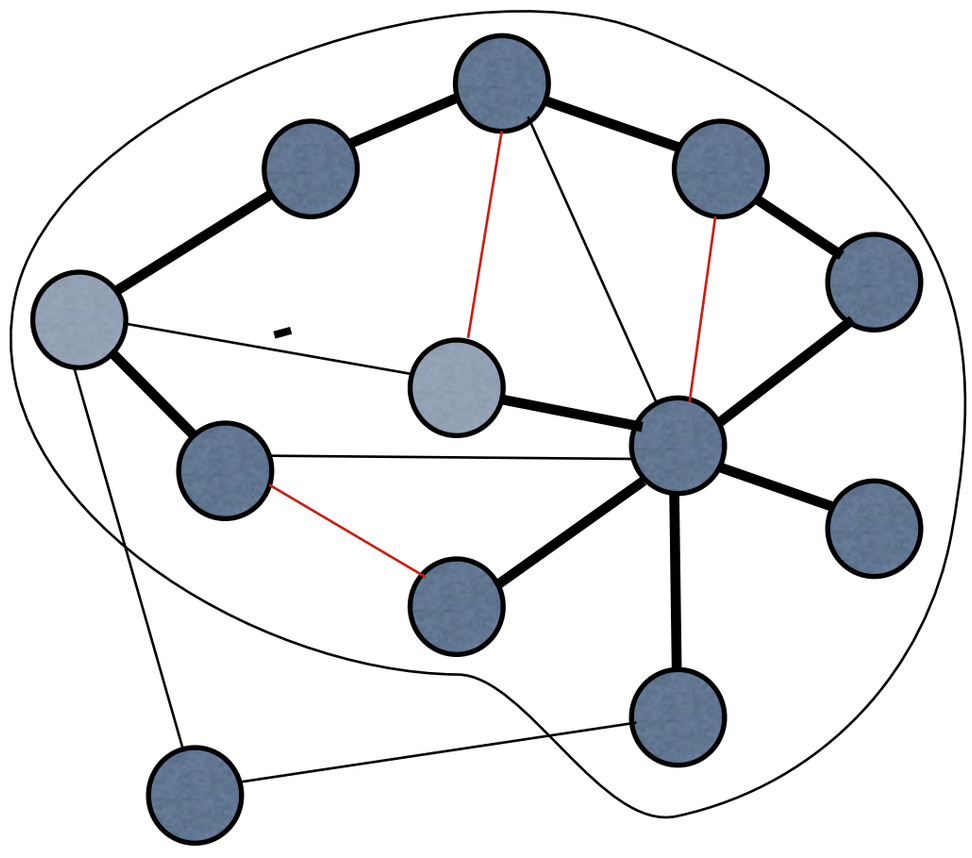
Kruskal algoritmus

felelevenítés



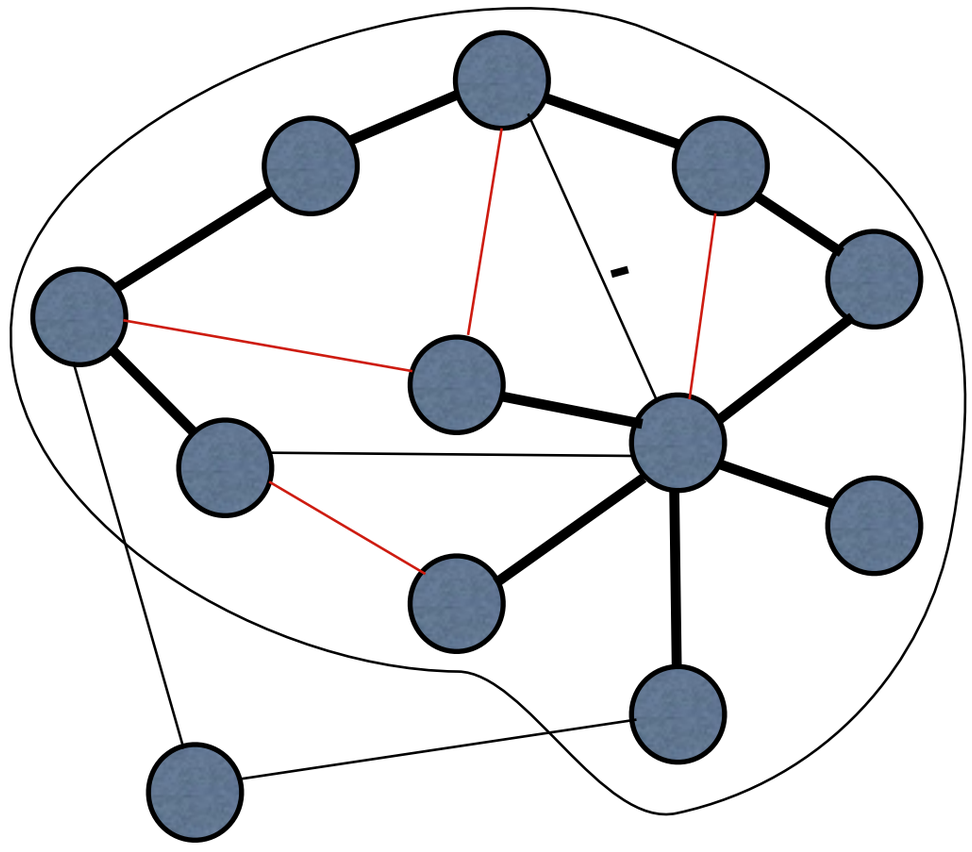
Kruskal algoritmus

felelevenítés



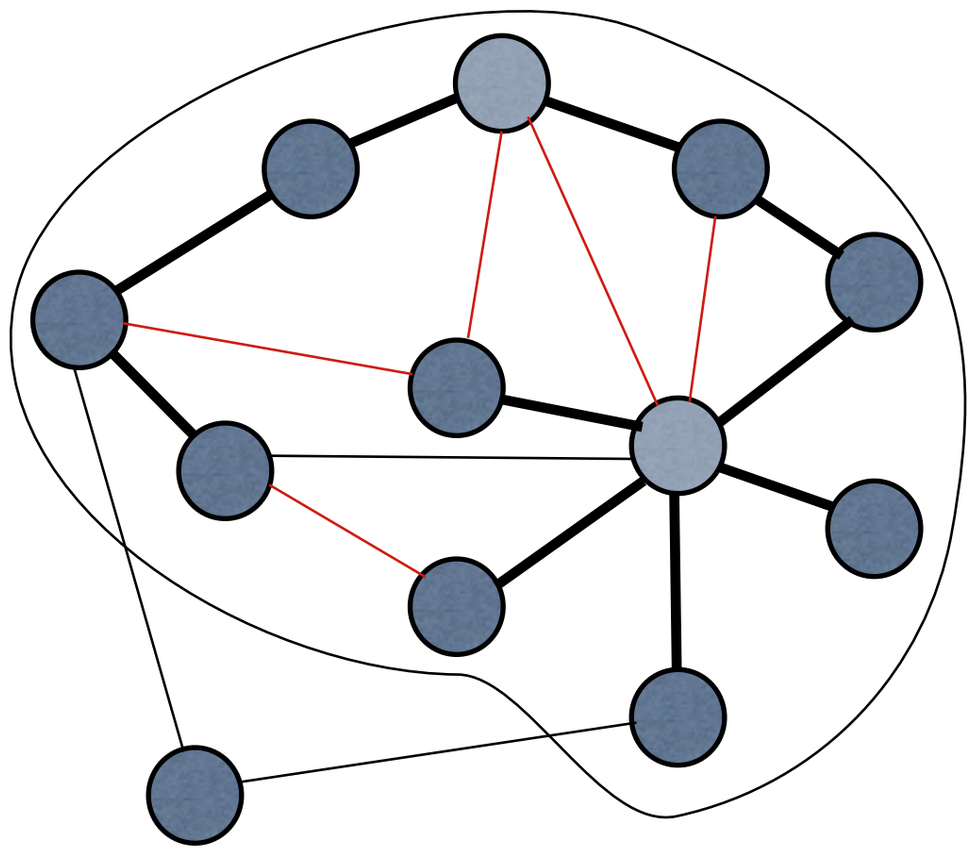
Kruskal algoritmus

felelevenítés



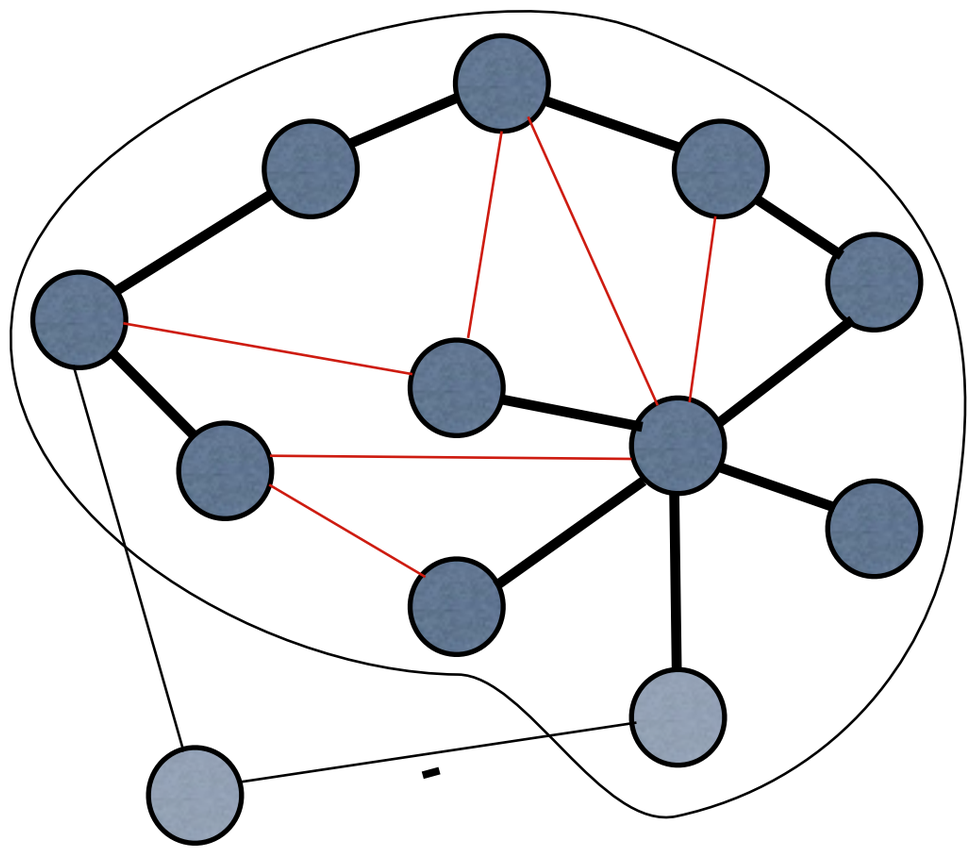
Kruskal algoritmus

felelevenítés



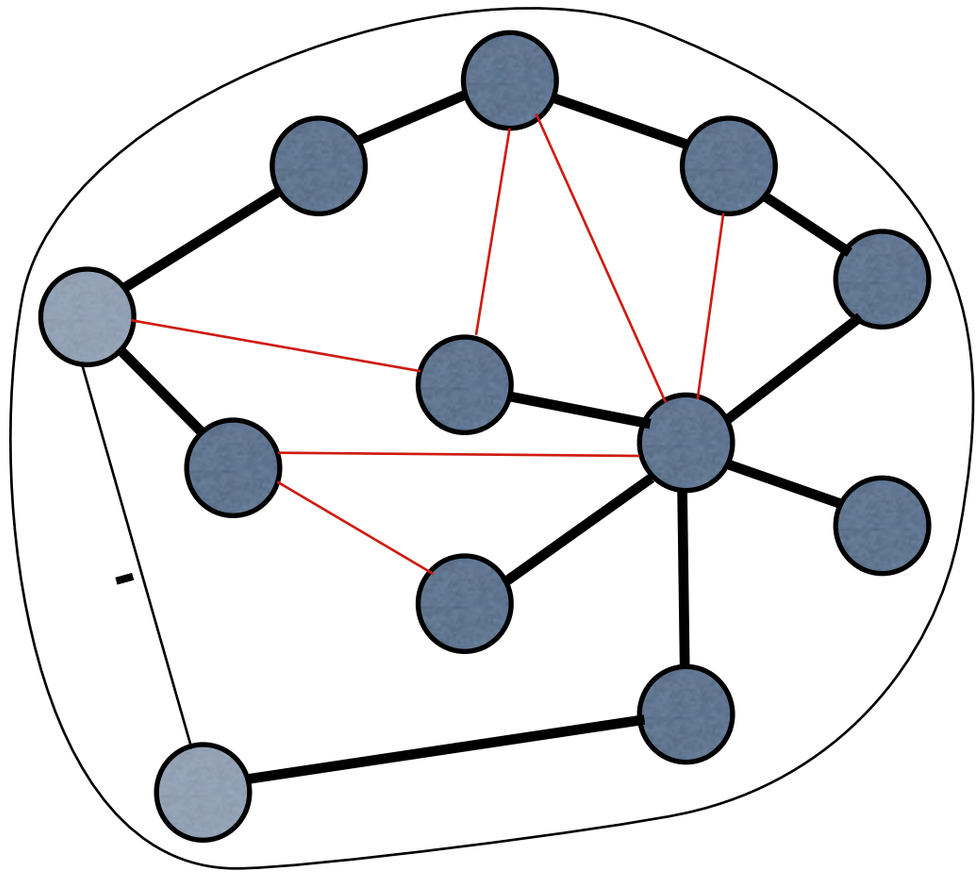
Kruskal algoritmus

felelevenítés



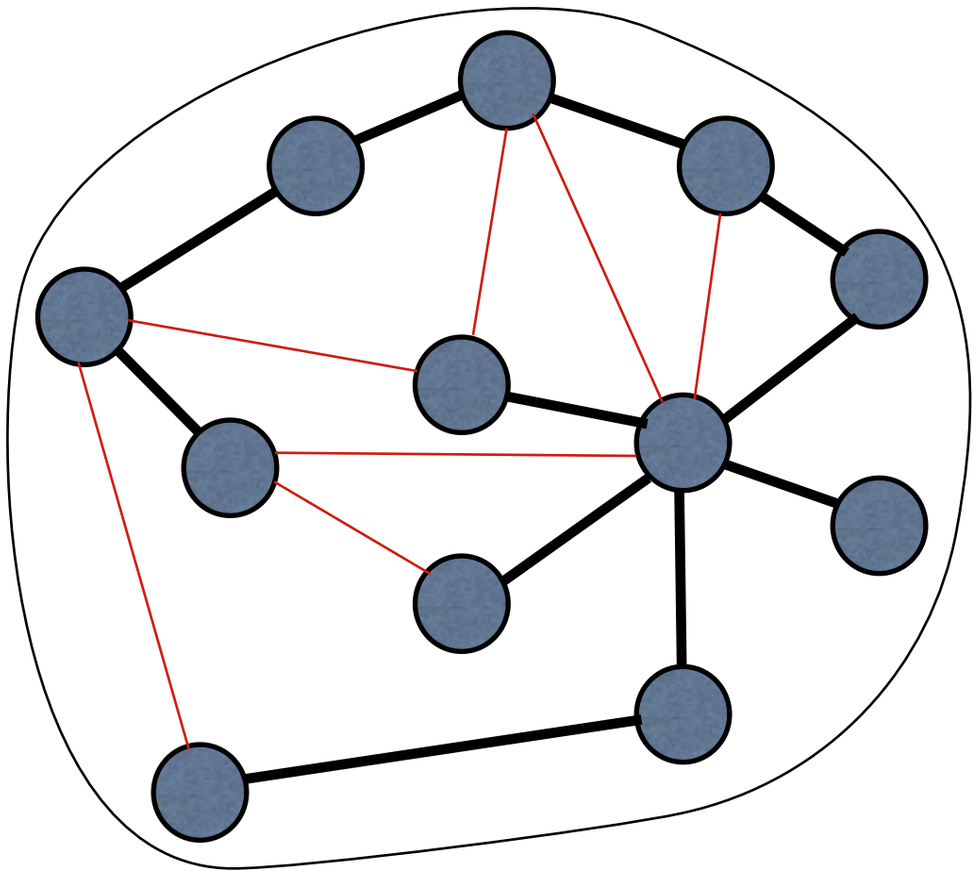
Kruskal algoritmus

felelevenítés



Kruskal algoritmus

felelevenítés



Kruskal algoritmus

felelevenítés

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

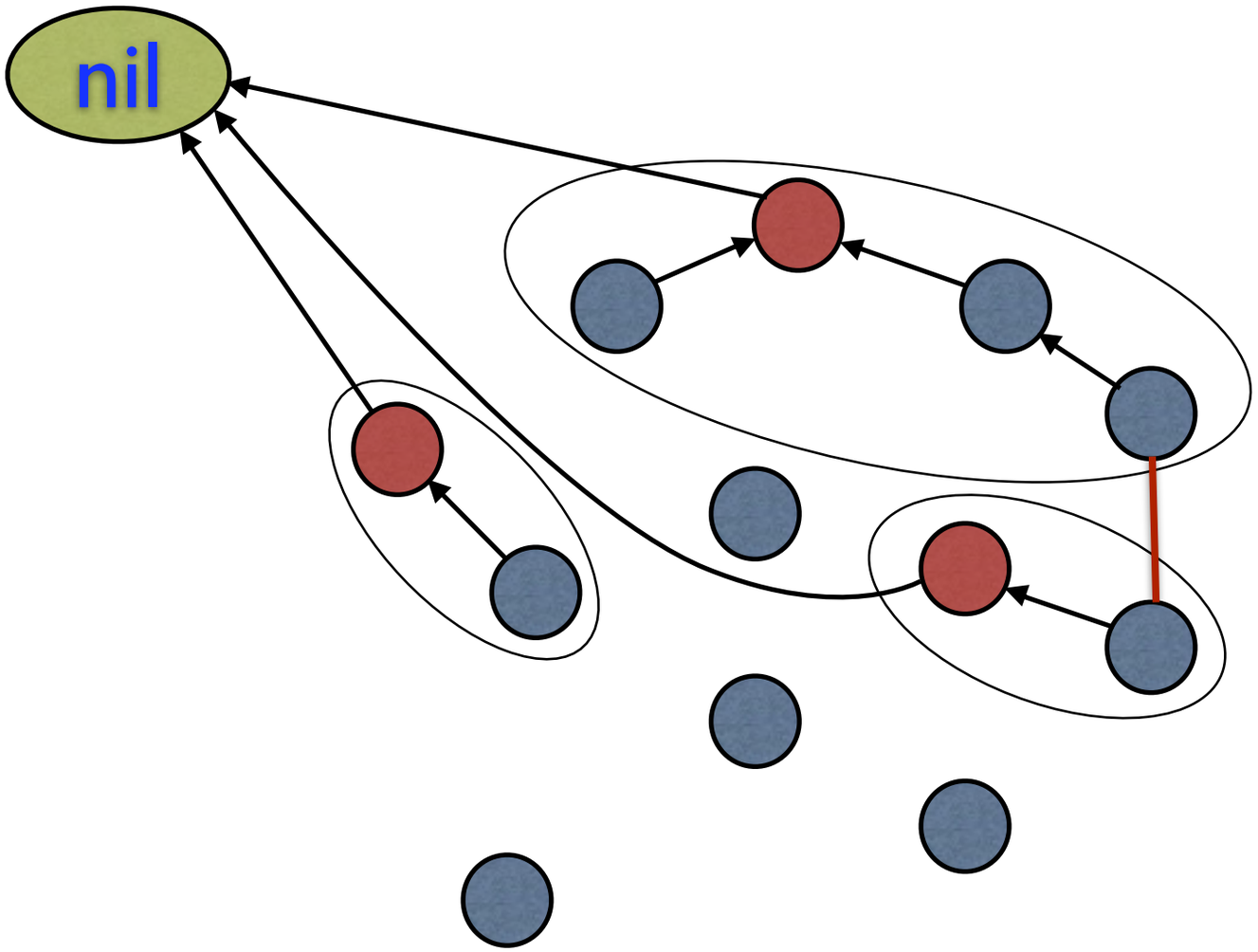
Honnan lehet tudni, hogy az adott él 2 végén lévő pont egy halmazban van-e?

Hogyan tudom a kezdetben n halmazt hatékonyan egyesítgetni?

Halmazok listájában az összes halmazt végigkeresni O(n) lenne :(

Halmaz (piros-fekete fa) adatszerkezettel 2 halmaz egyesítése O(n) lenne :(

Halmazerdő adattípus



Kruskal algoritmus halmazerdővel

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

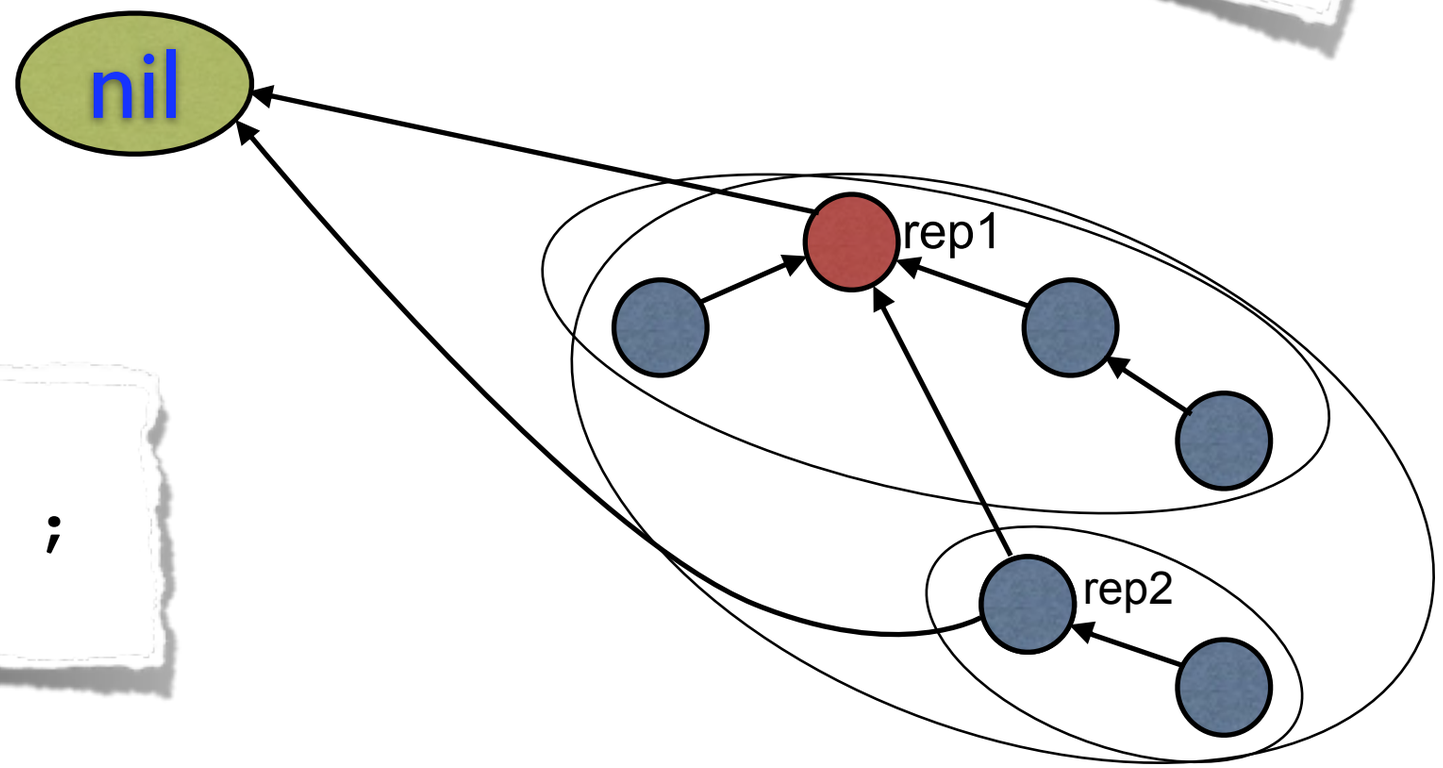
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
   }
   return q ;
}
```

Kruskal algoritmus halmazerdővel

```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
   }
   return q ;
}
```

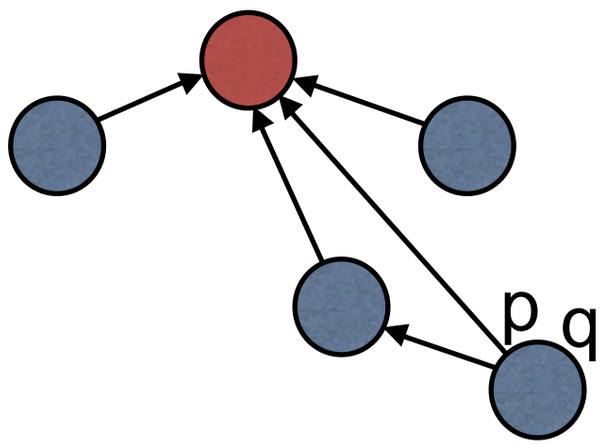
```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

```
unio(rep1,rep2) {
   rep2.kovetkezo=rep1 ;
}
```



Halmazerdő tömörítése

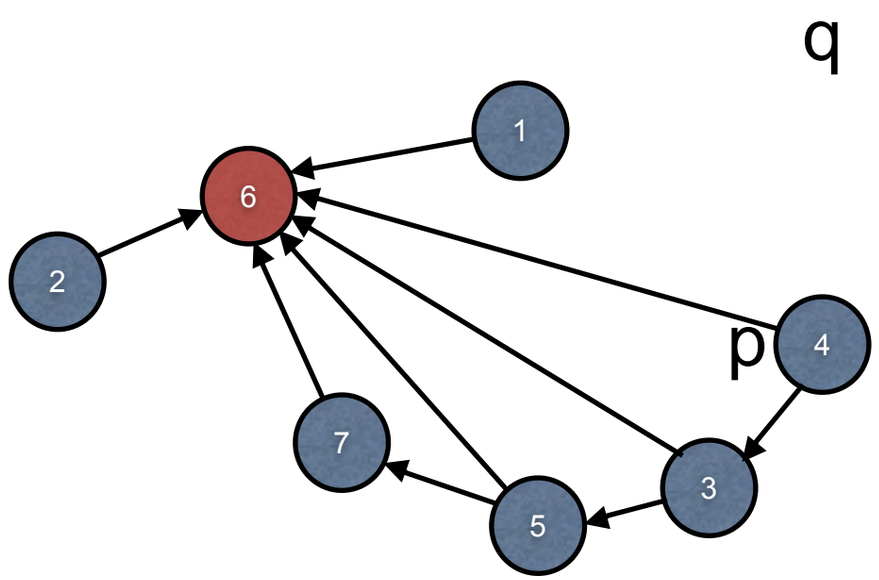
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
      q.kovetkezo=p.kovetkezo ;
   }
   return p ;
}
```



Halmazerdő hatékonyabb tömörítése

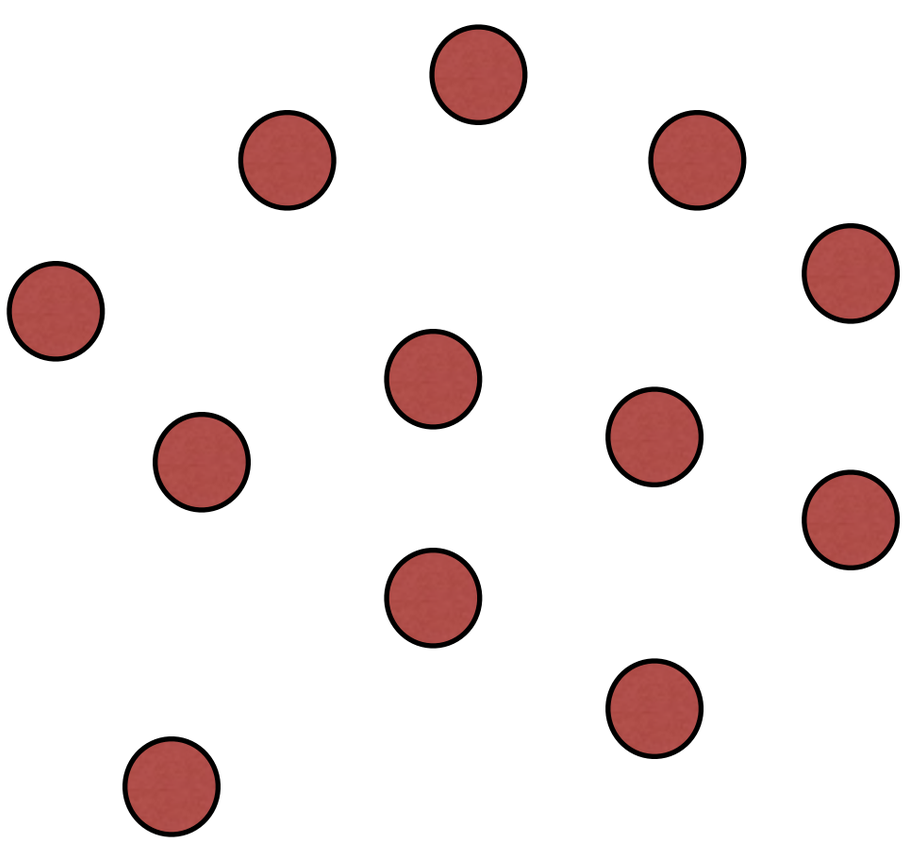
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      if (p.kovetkezo.kovetkezo!=nil) sorba(p) ;
      p=p.kovetkezo ;
   }
   while (!sor_ures()) {
      q=sorbol() ;
      q.kovetkezo=p ;
   }
   return p ;
}
```

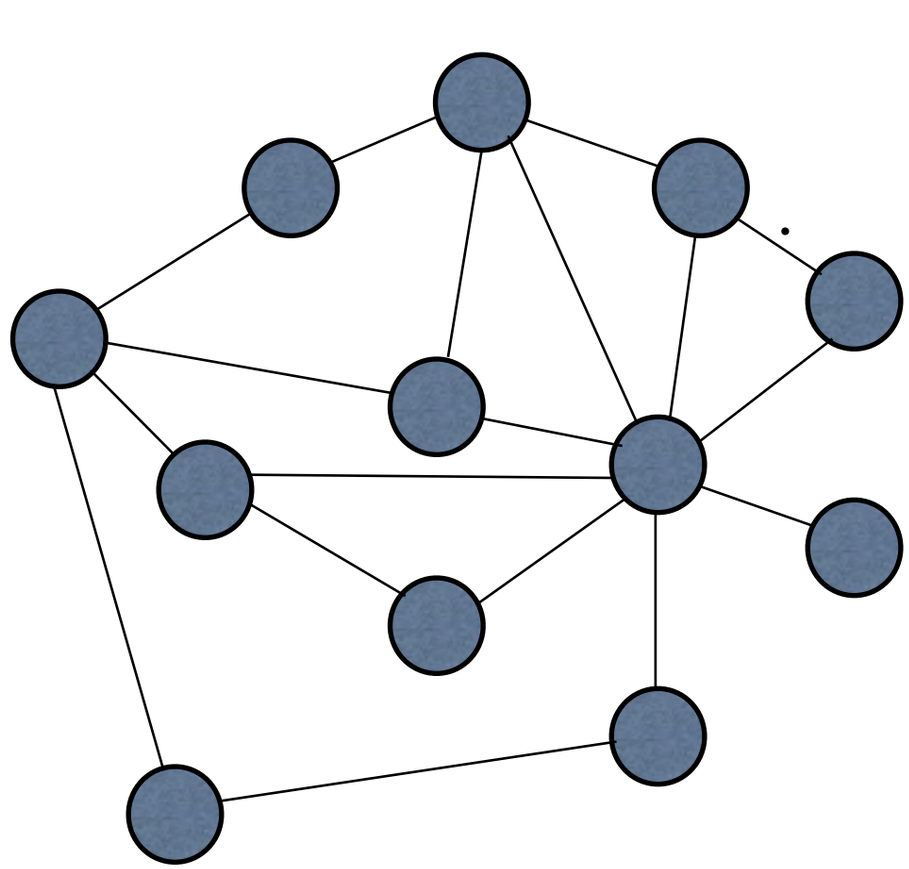
5 3 4



Halmazok egyesítése

Kruskal algoritmus

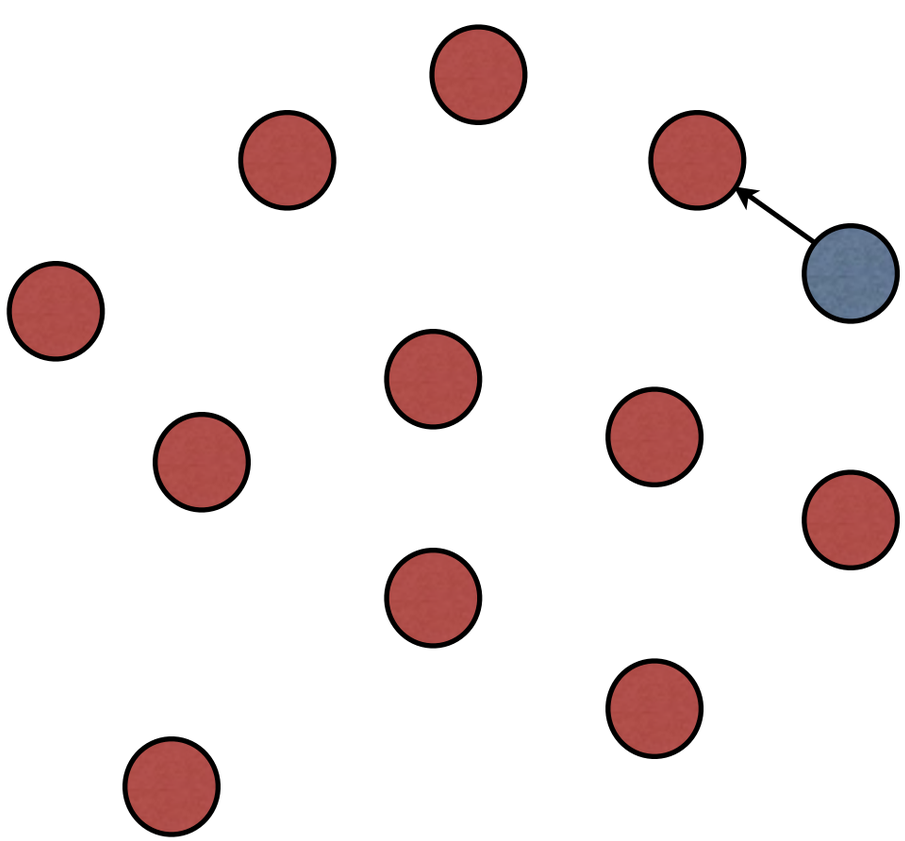


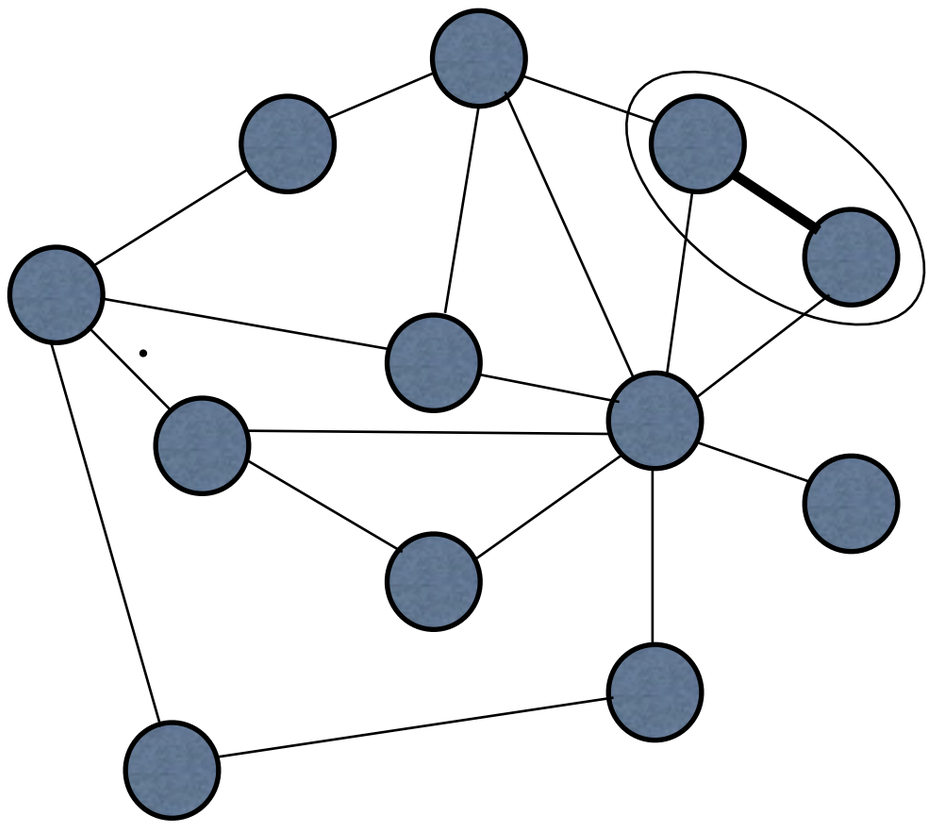


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

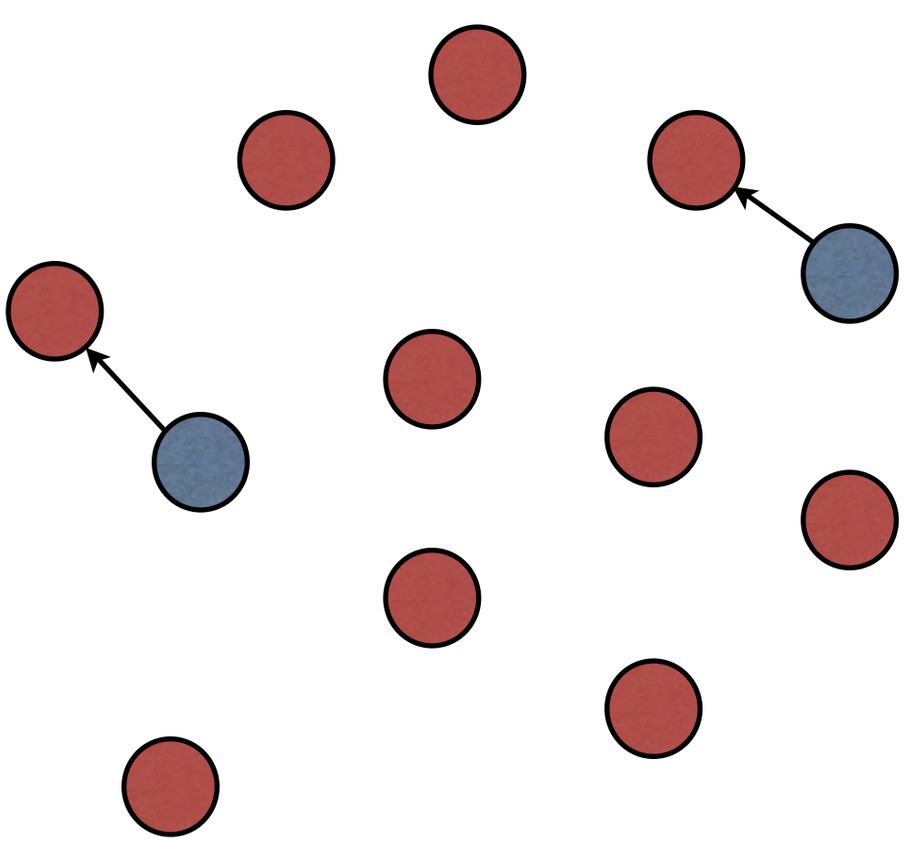


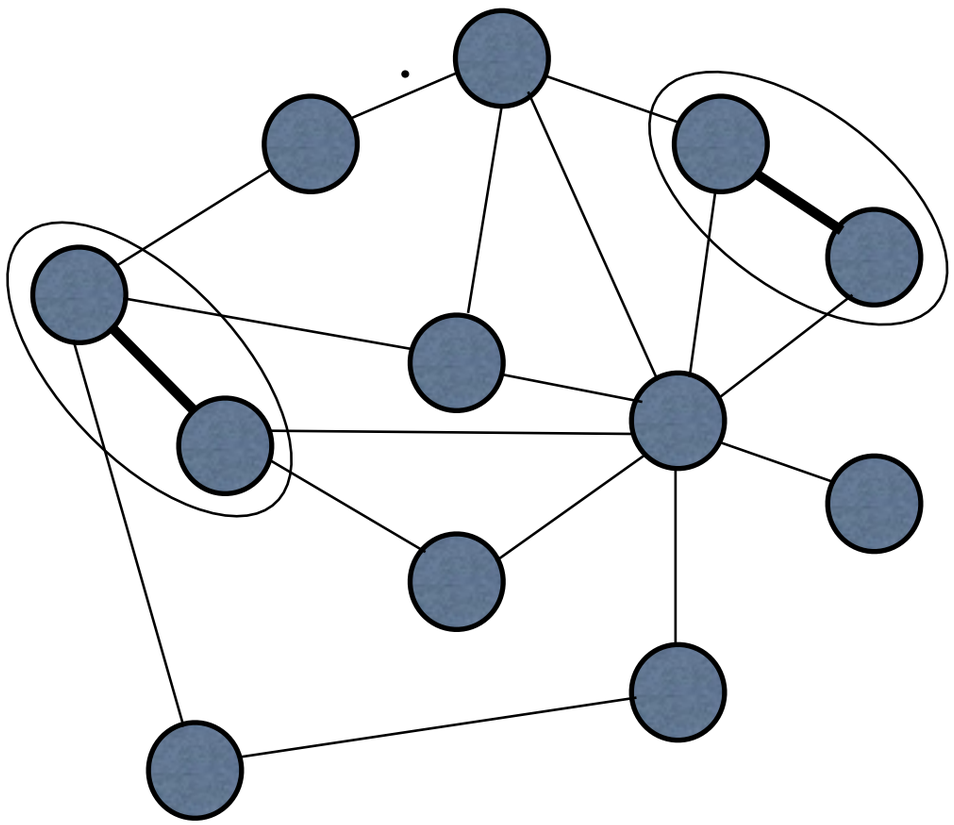


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

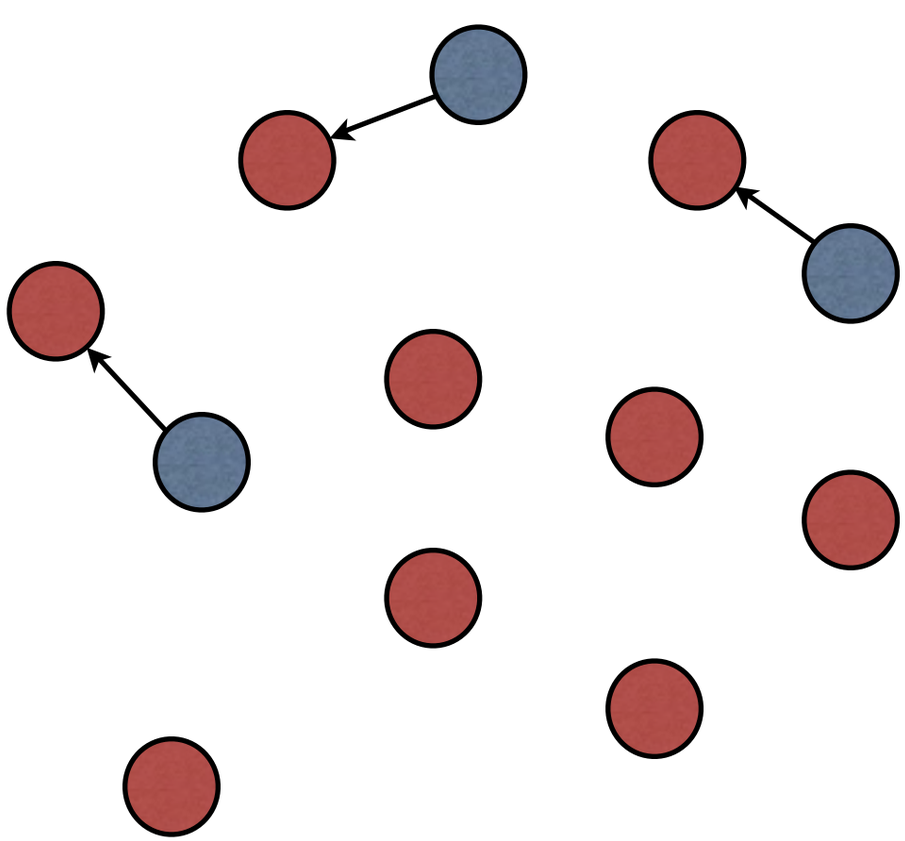


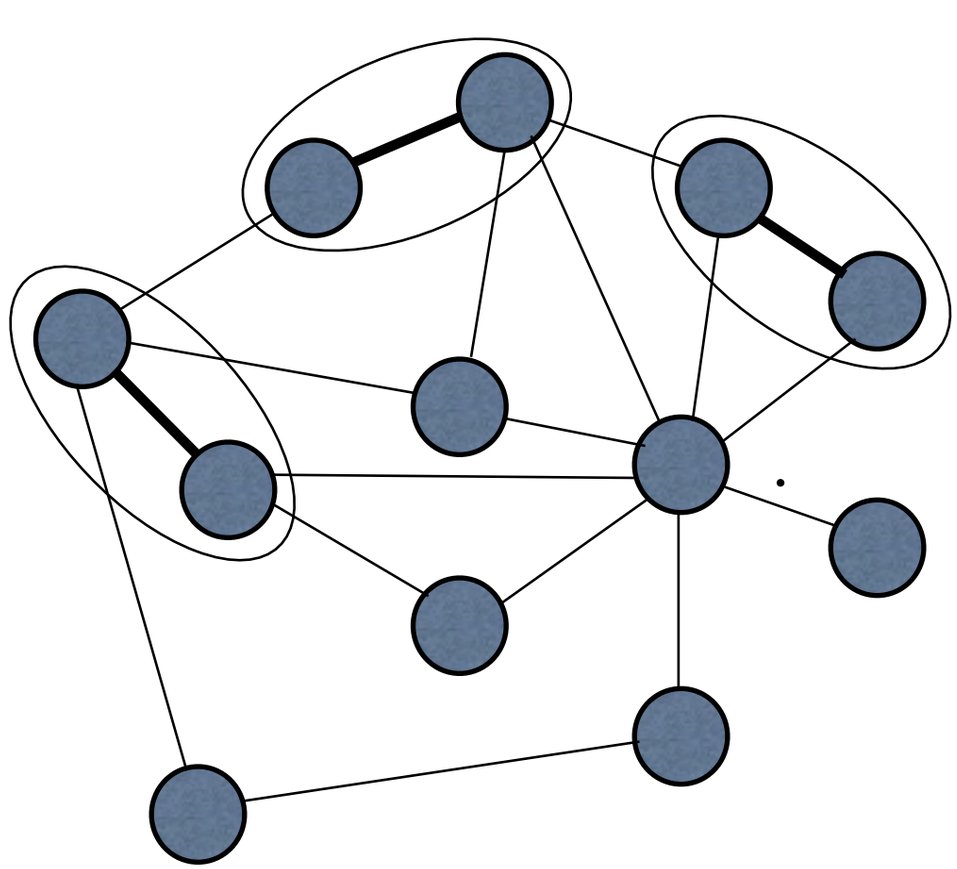


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

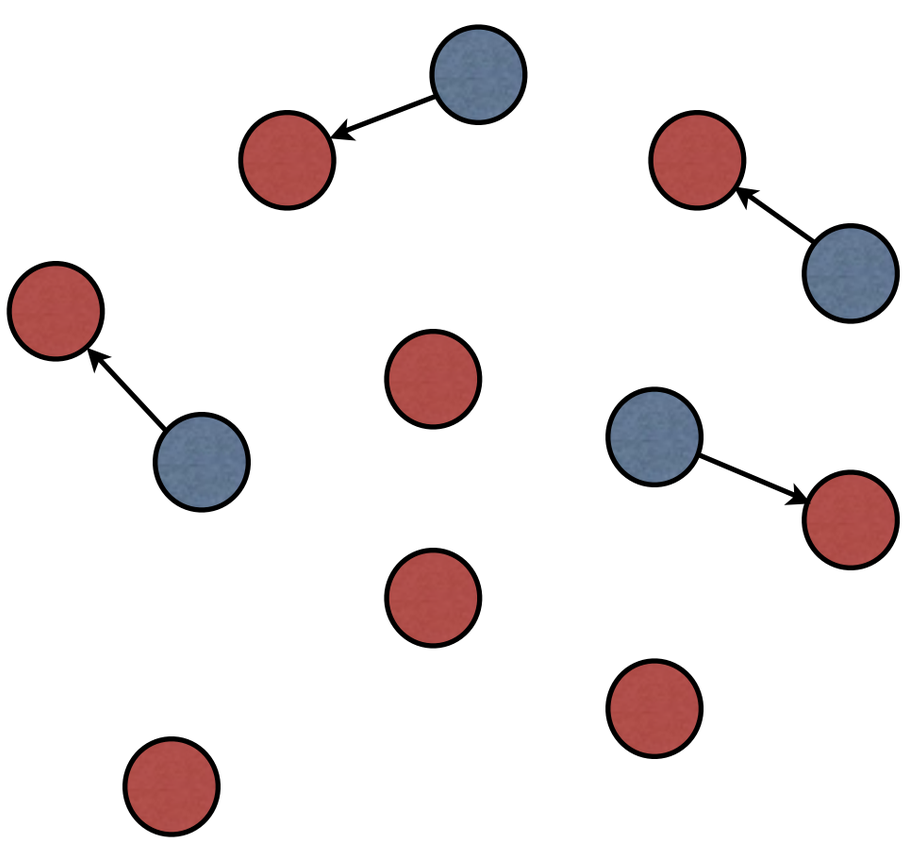


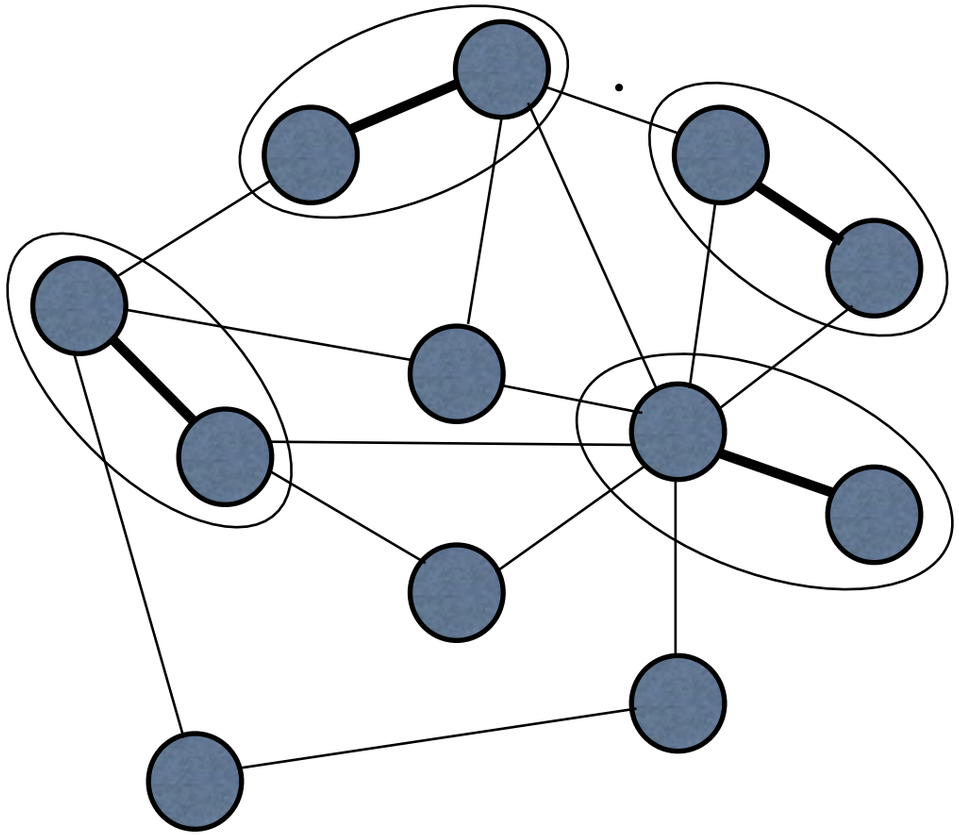


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

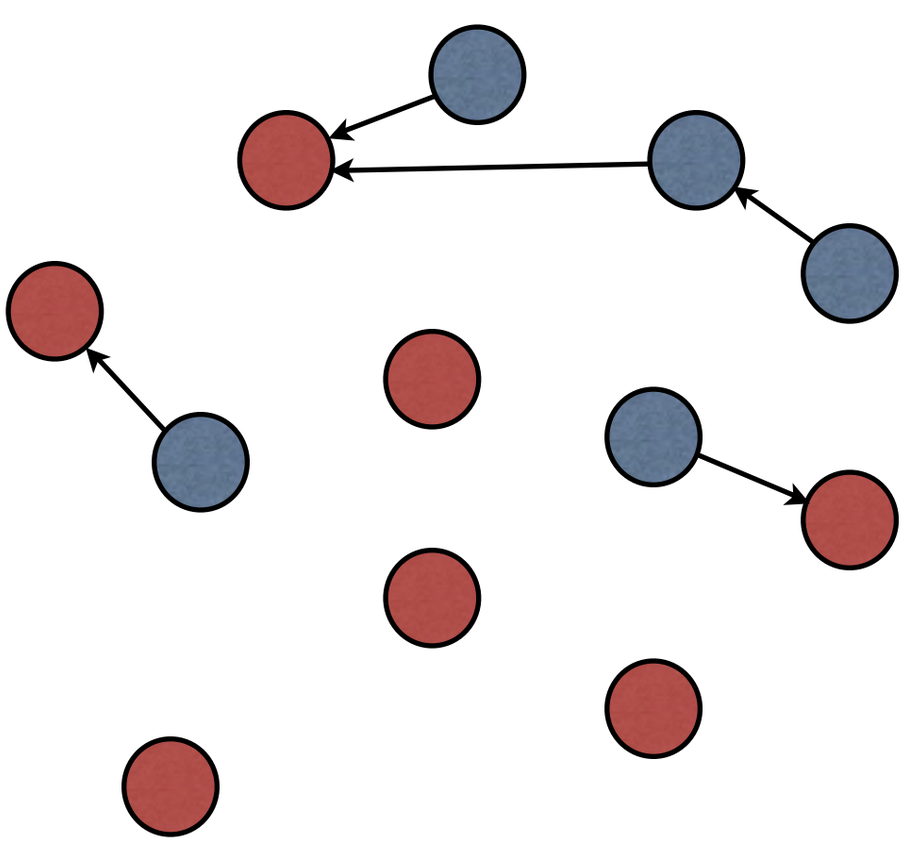


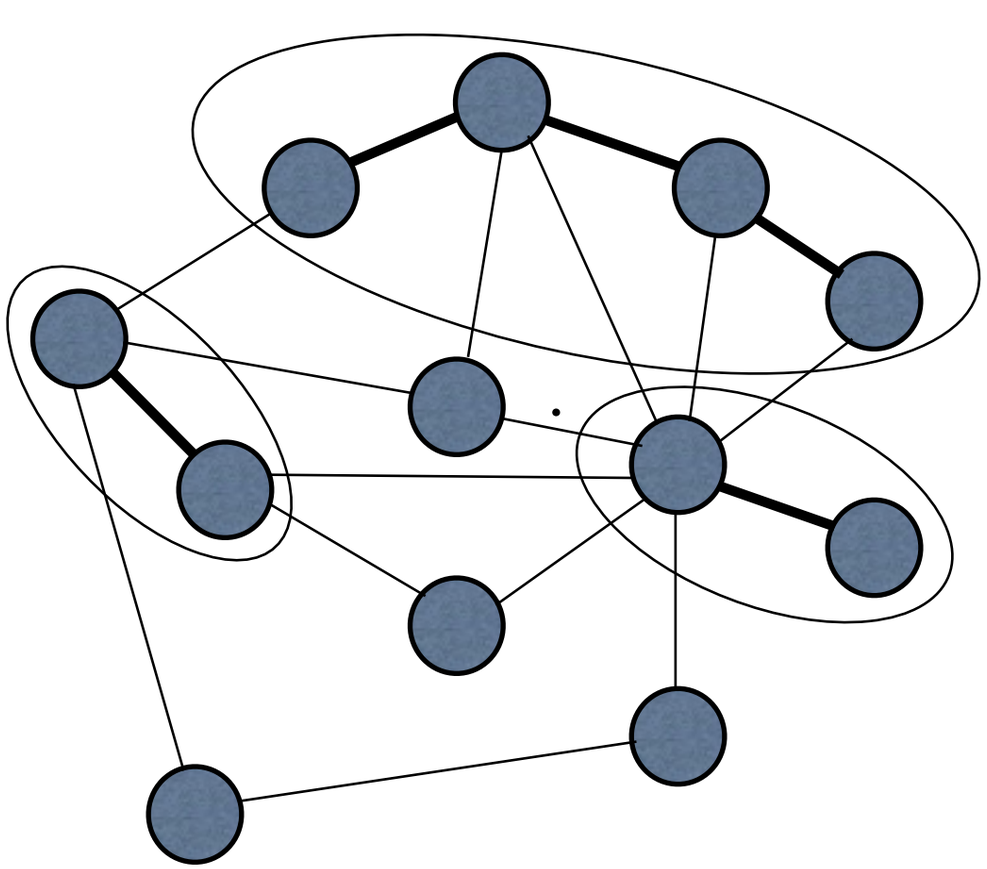


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

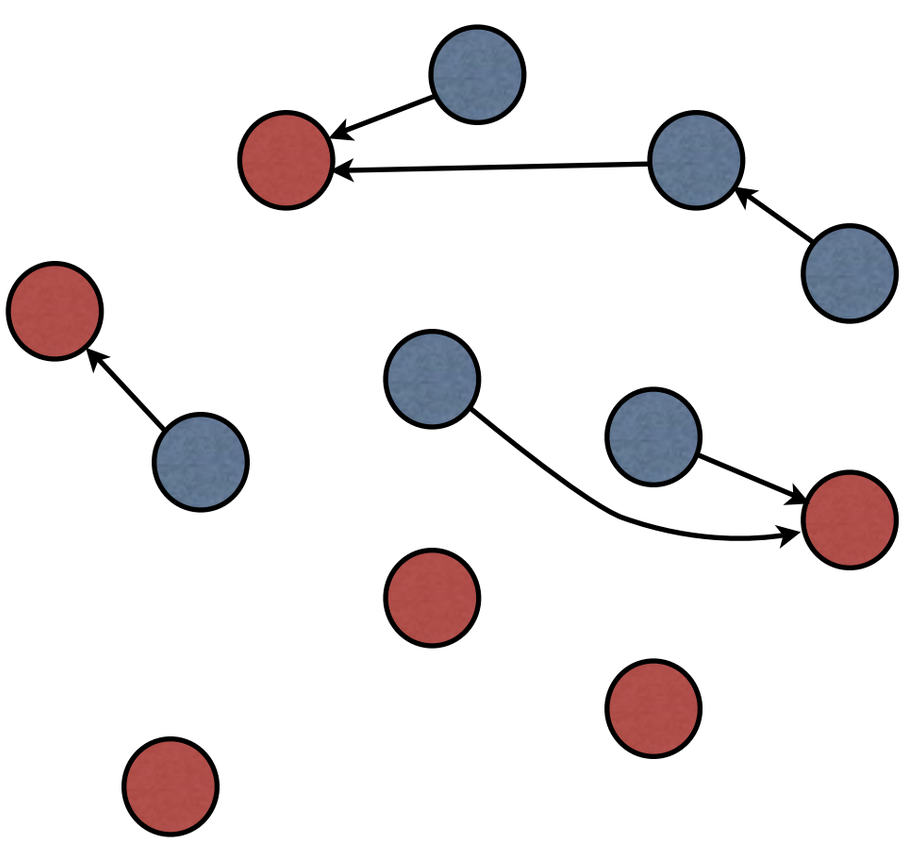


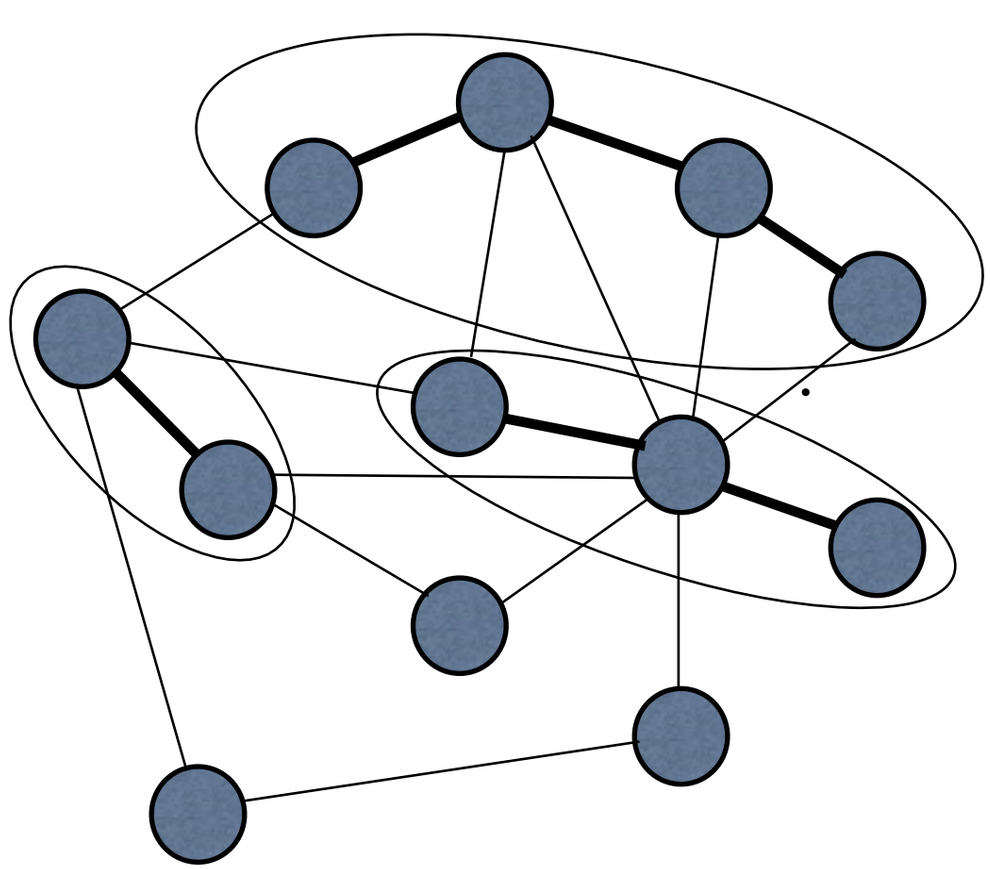


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

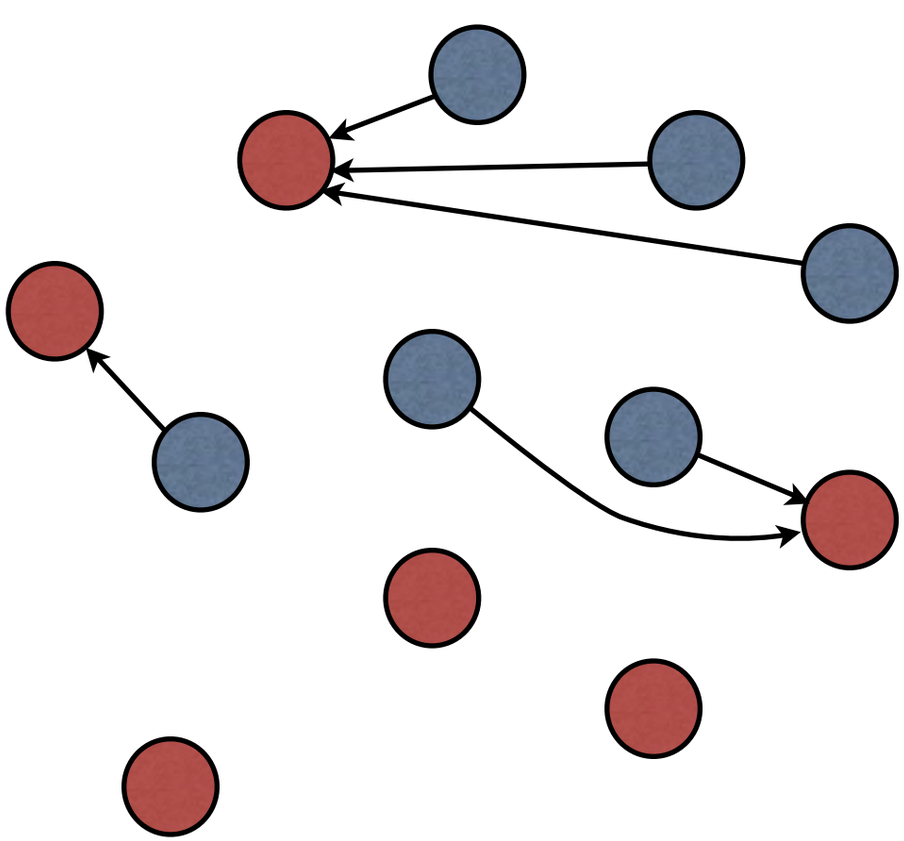


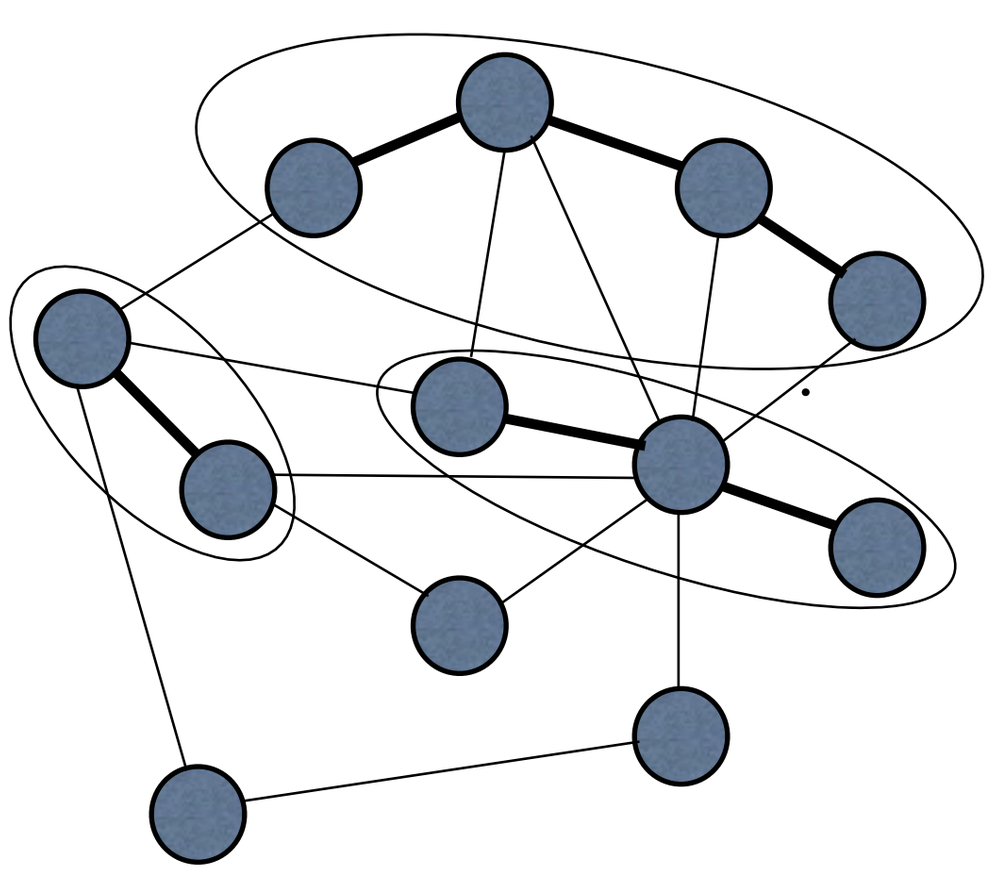


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

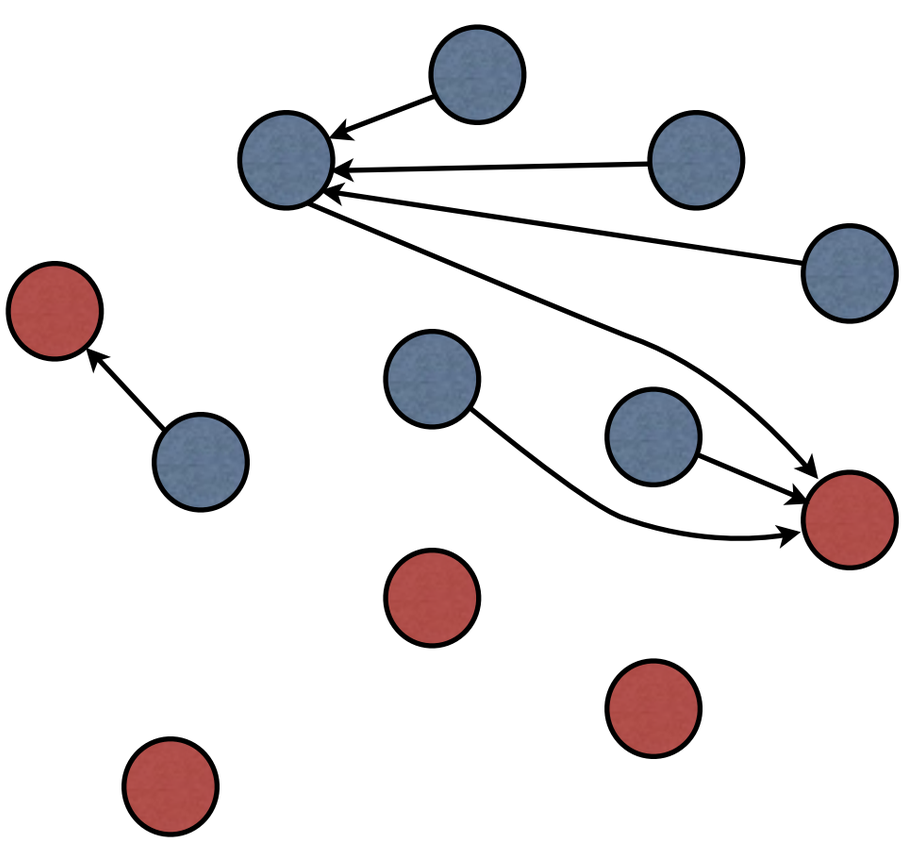


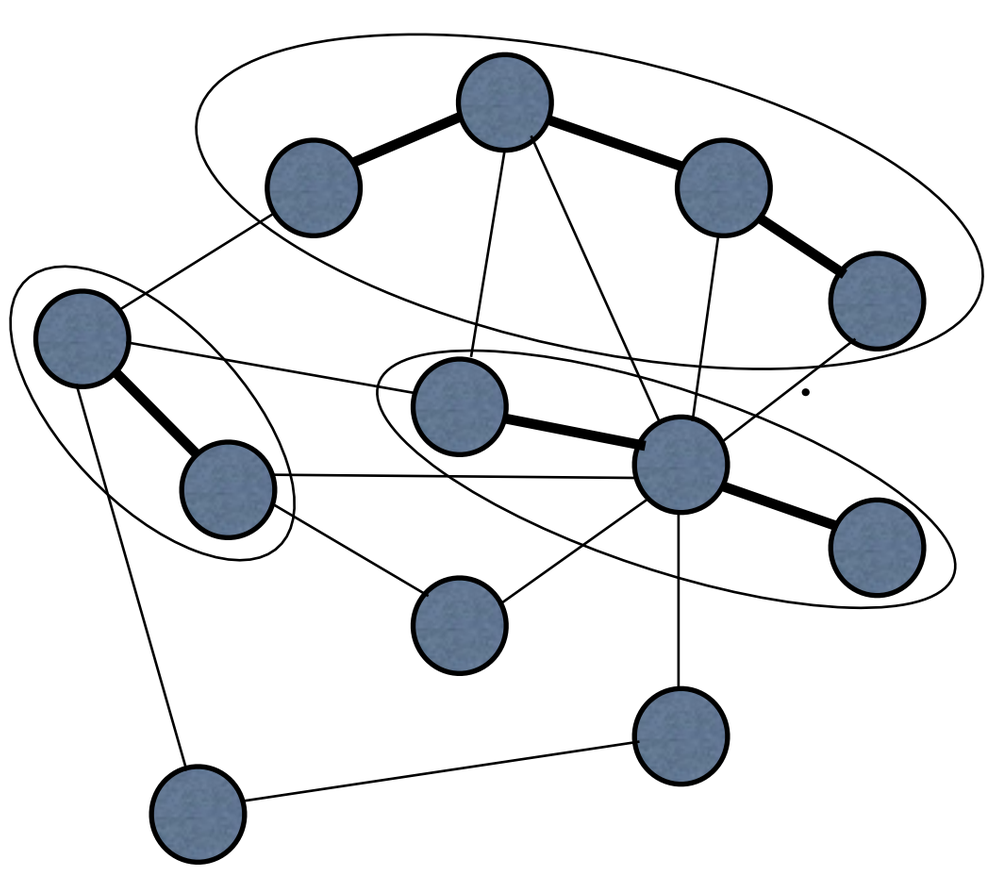


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

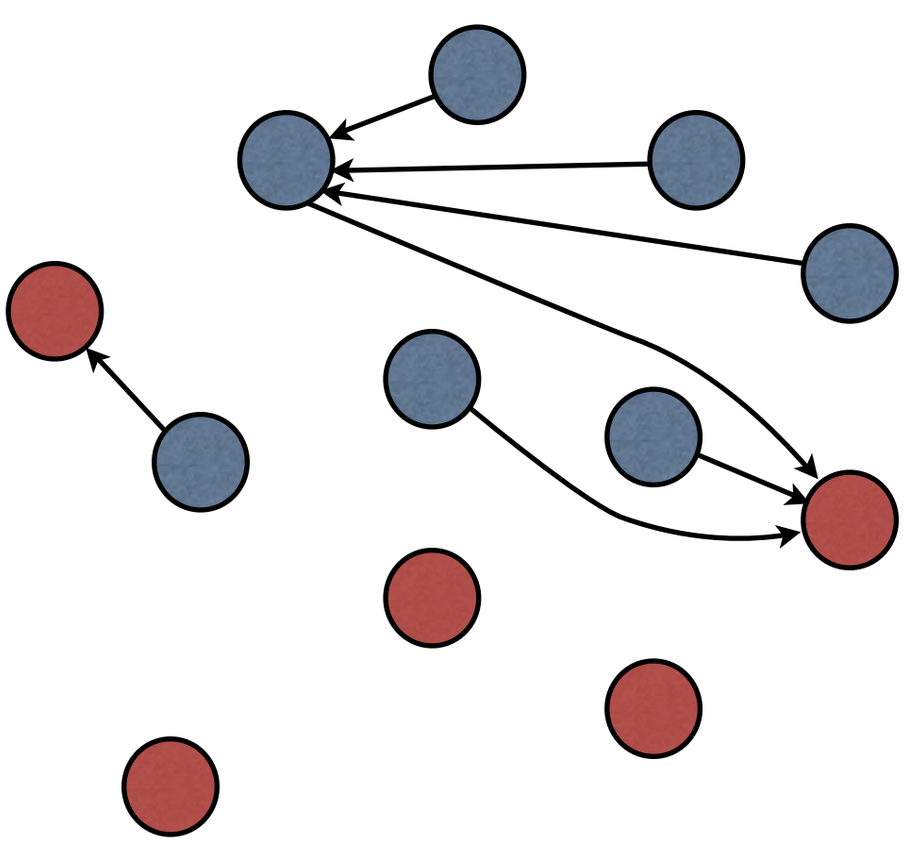


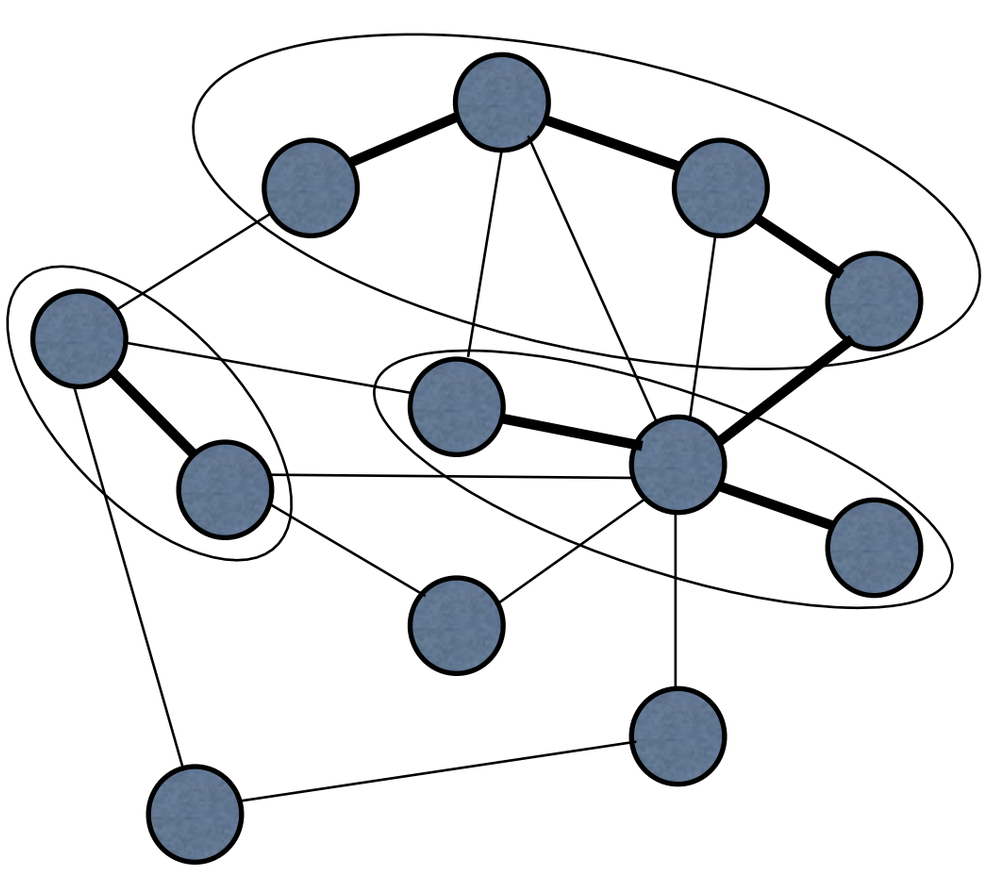


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

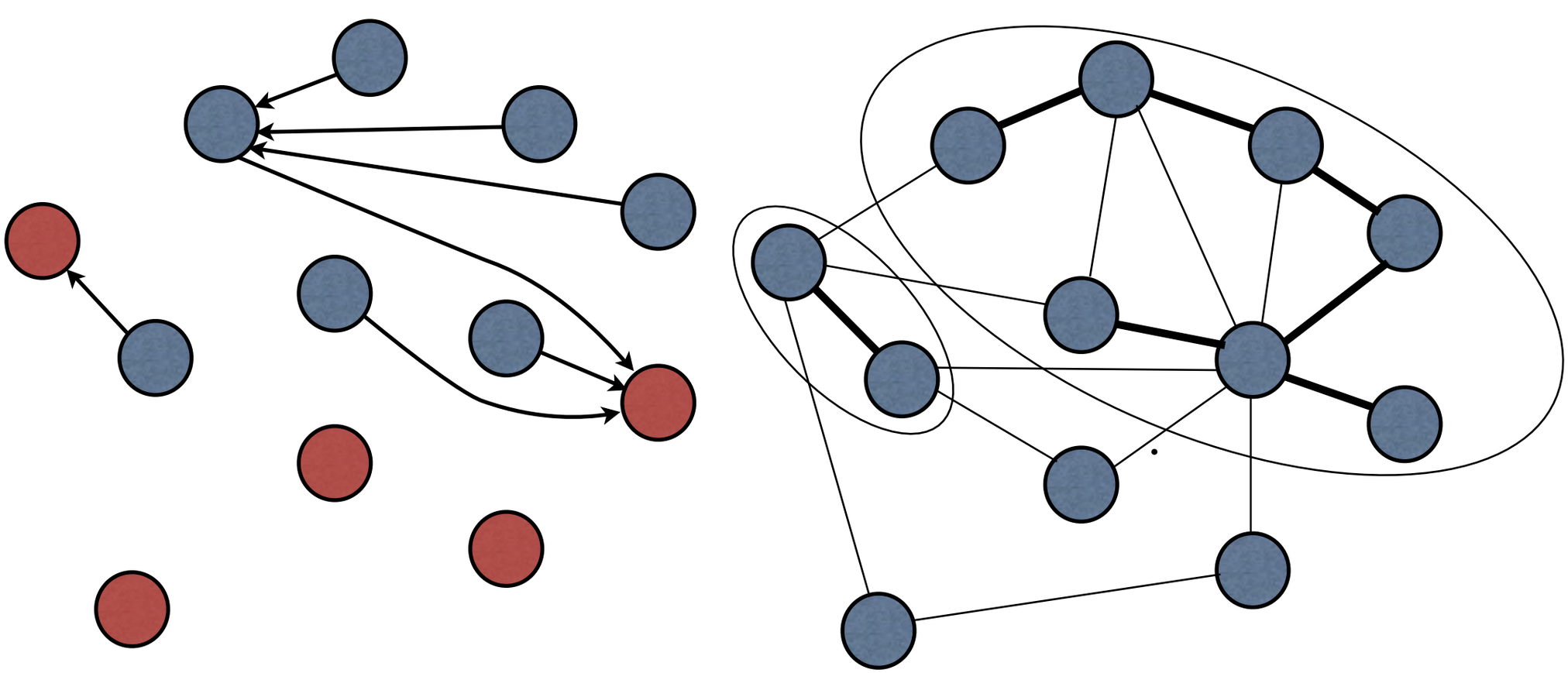




Halmazok egyesítése

halmazerdő felépítése

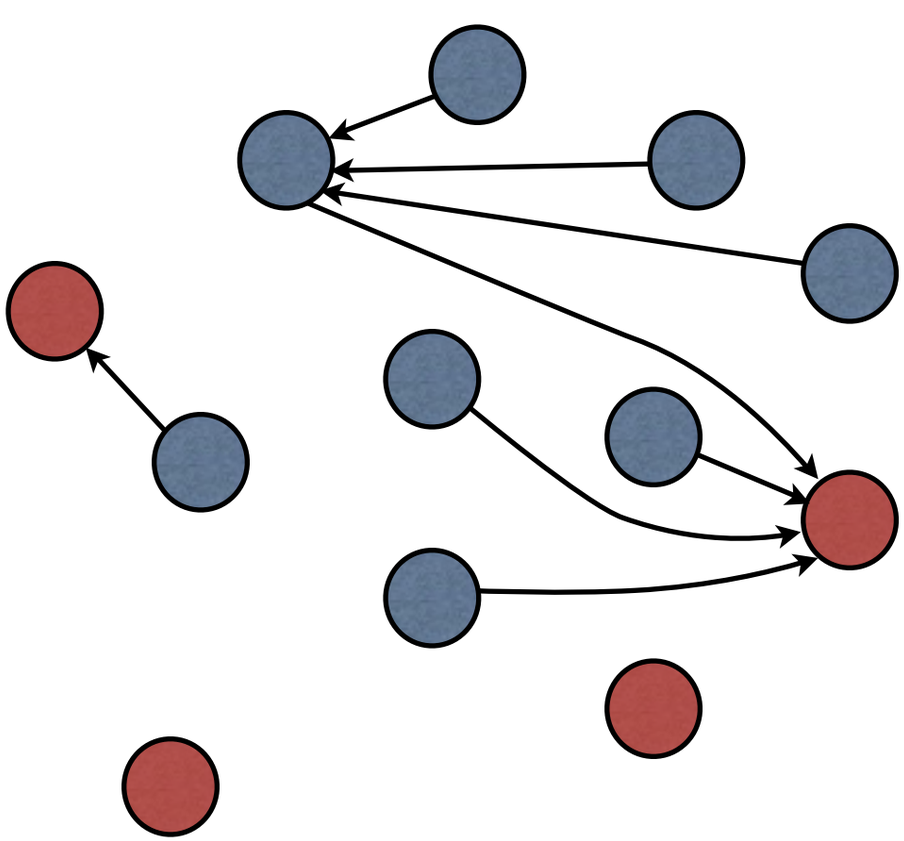
Kruskal algoritmus

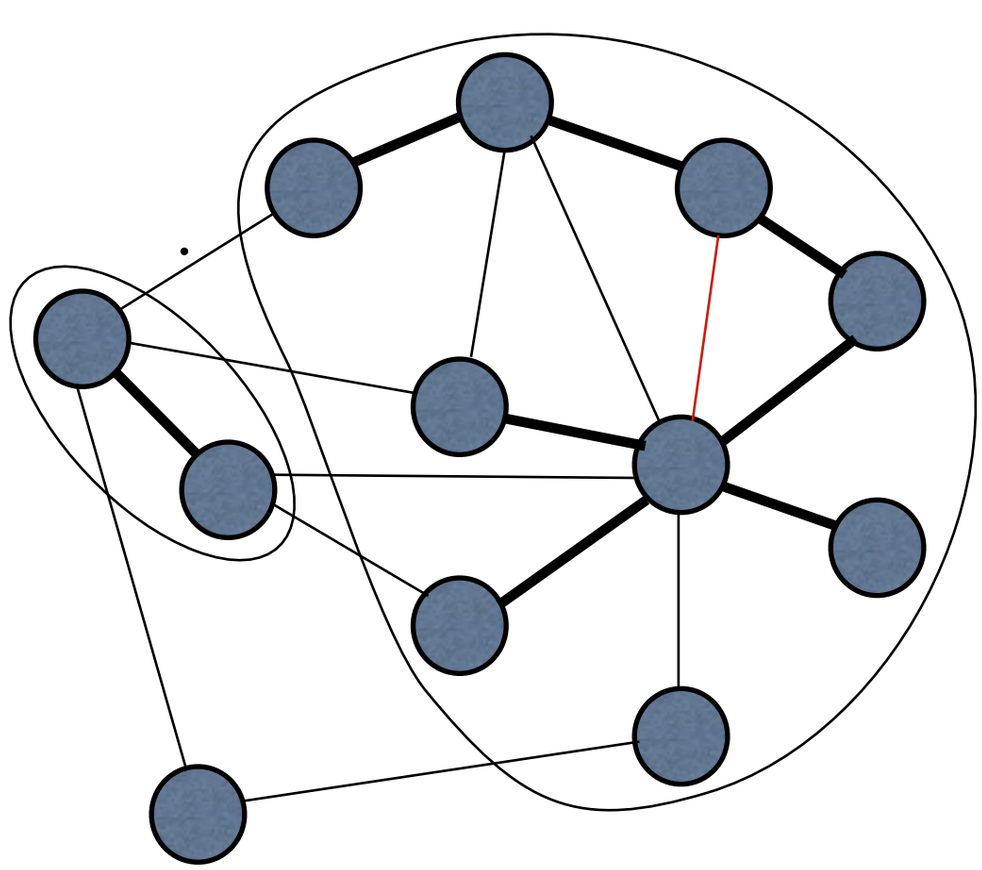


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

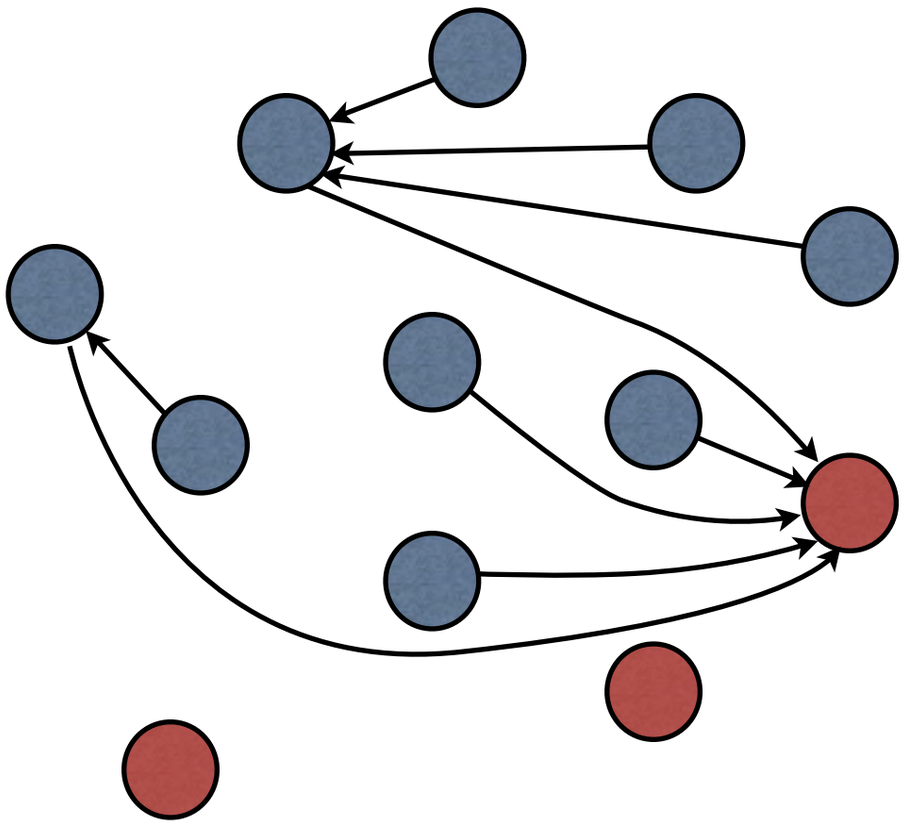


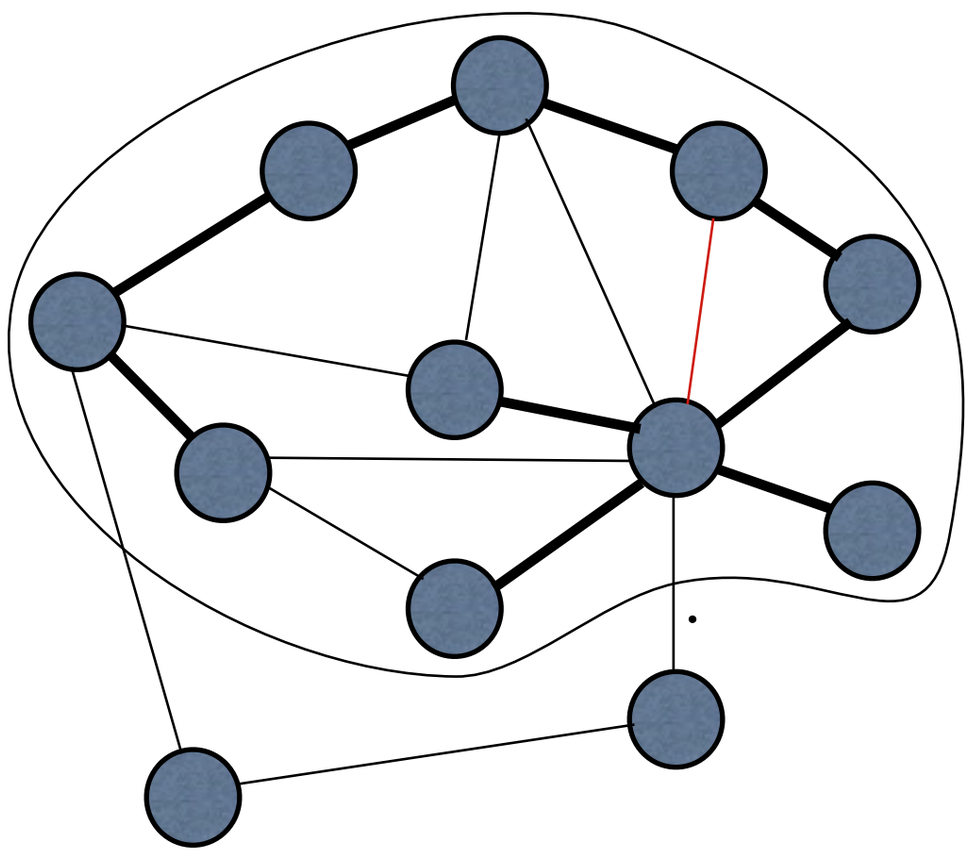


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

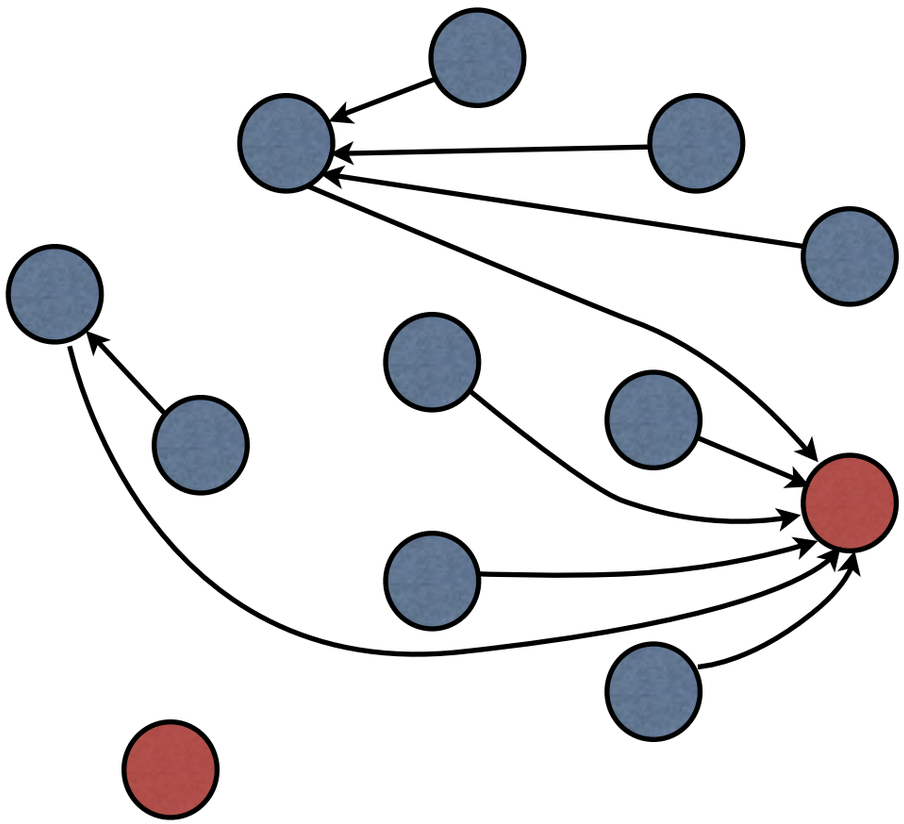


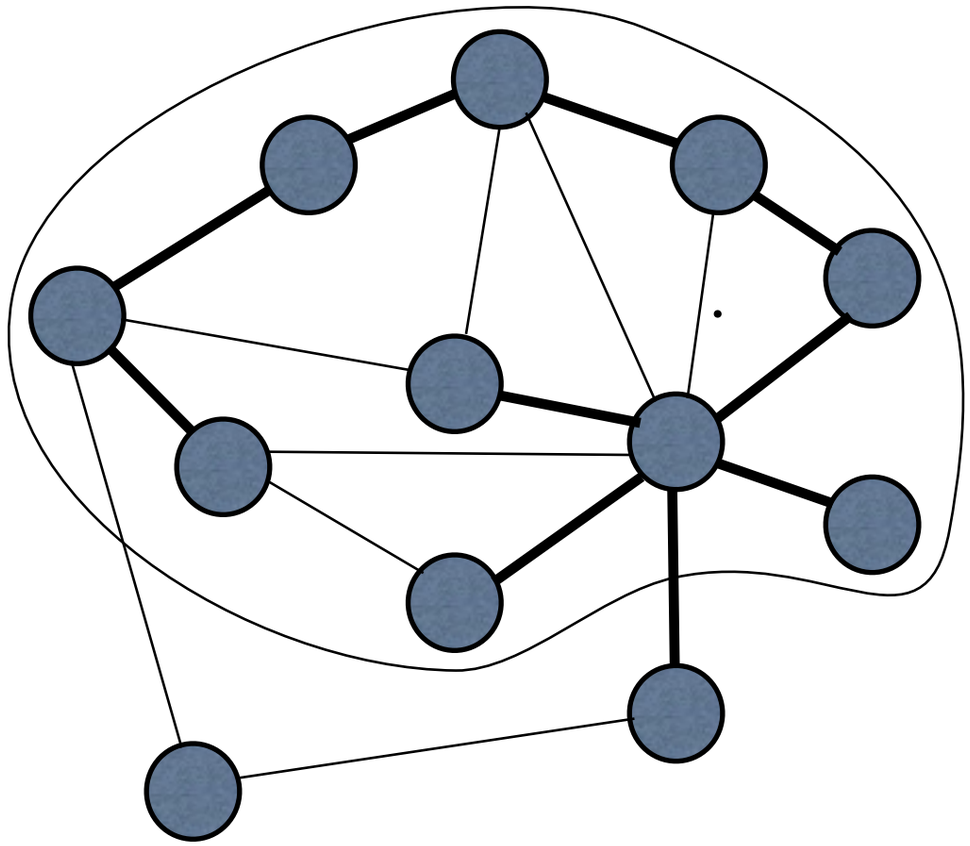


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

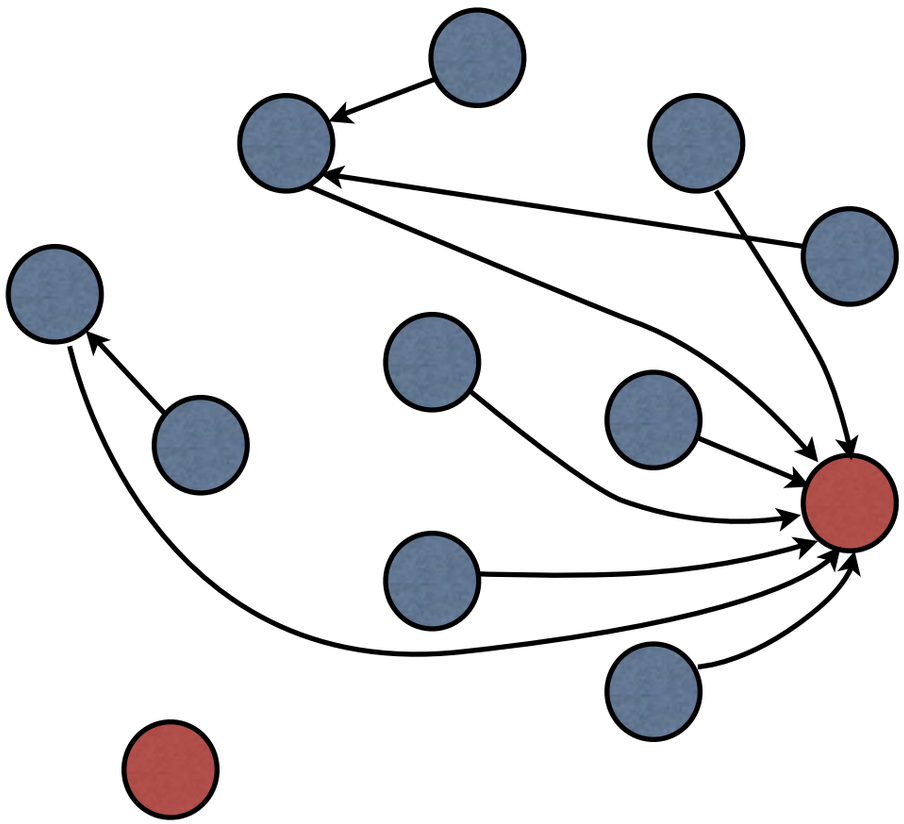


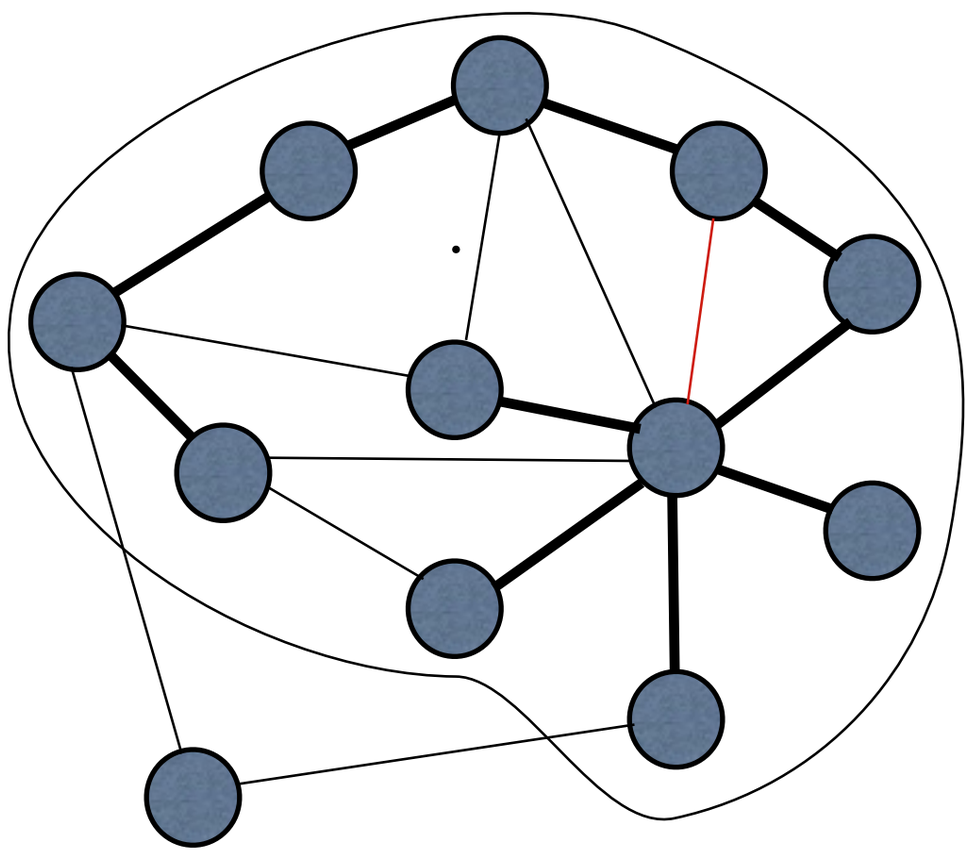


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

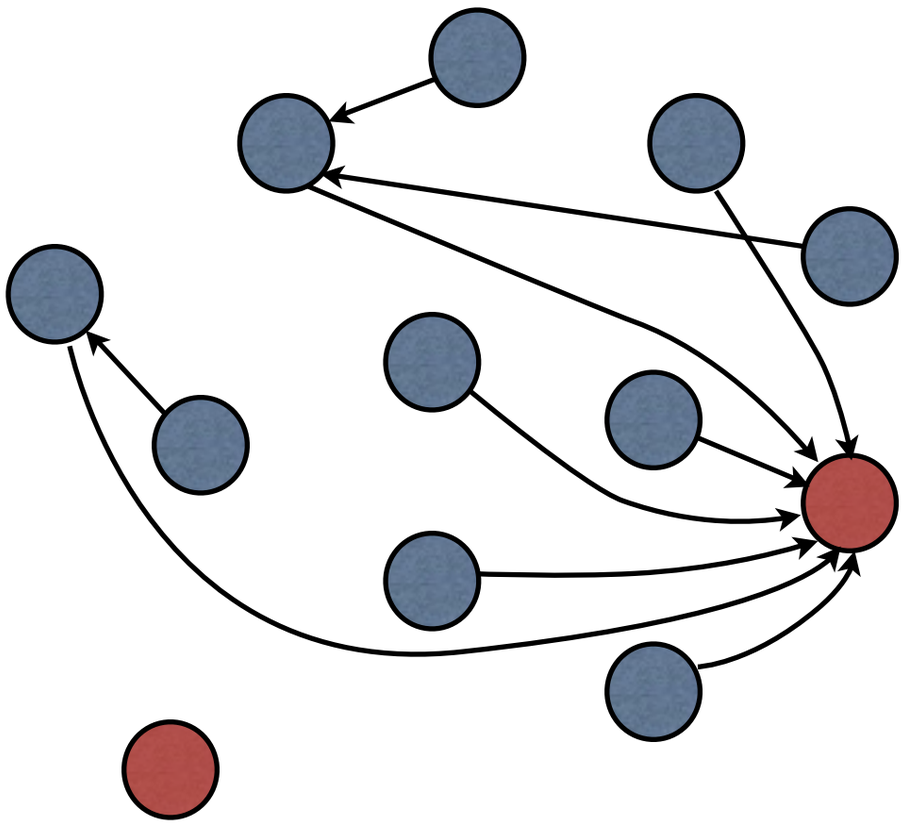


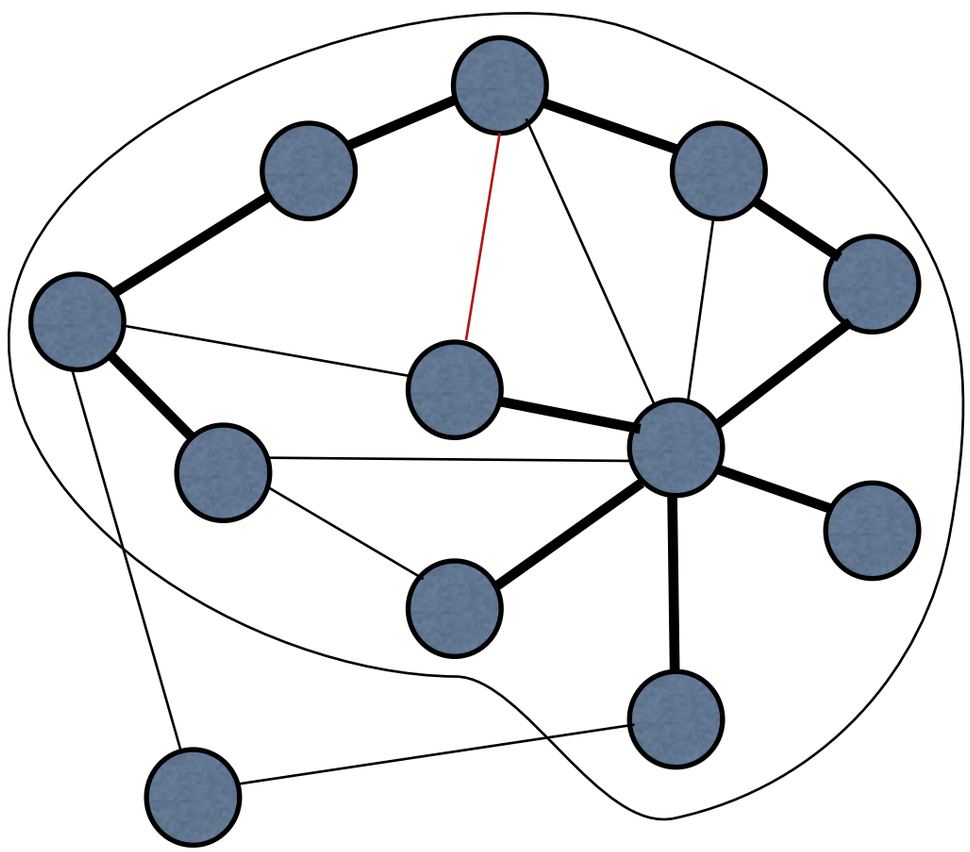


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

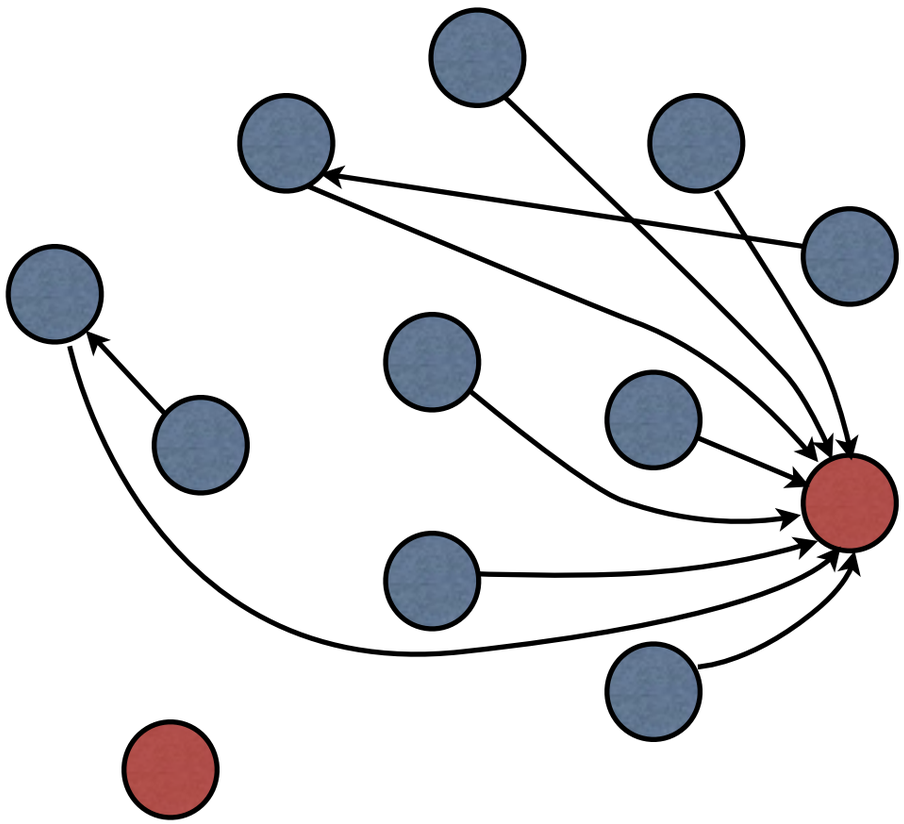


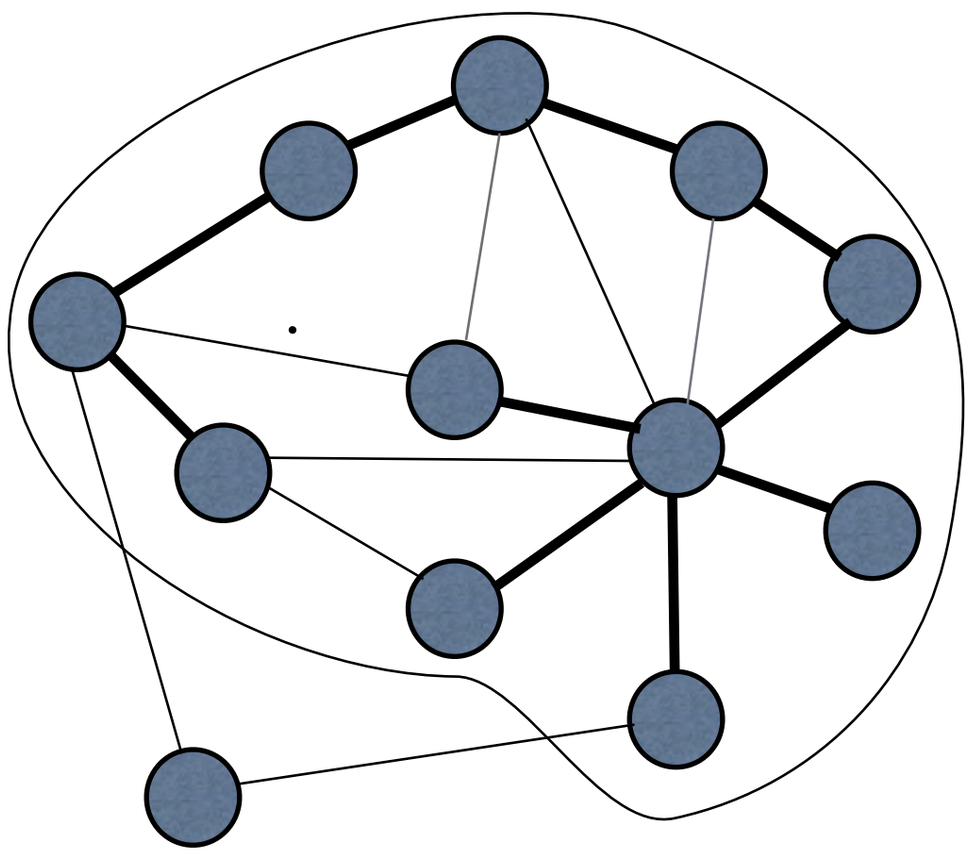


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

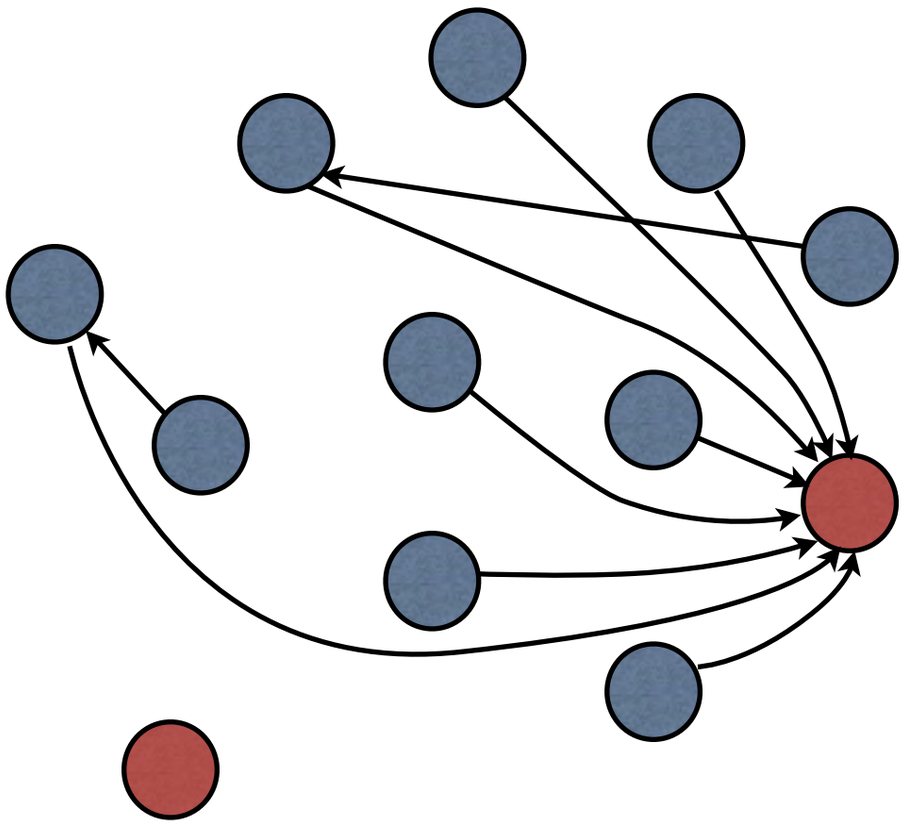


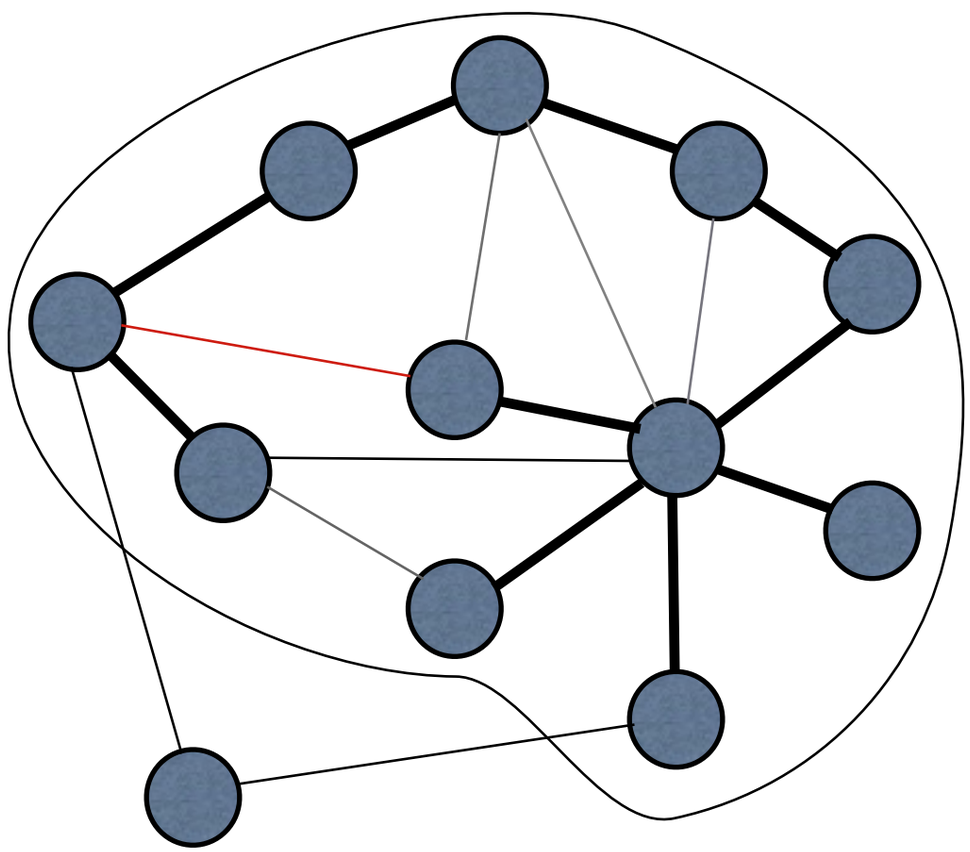


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

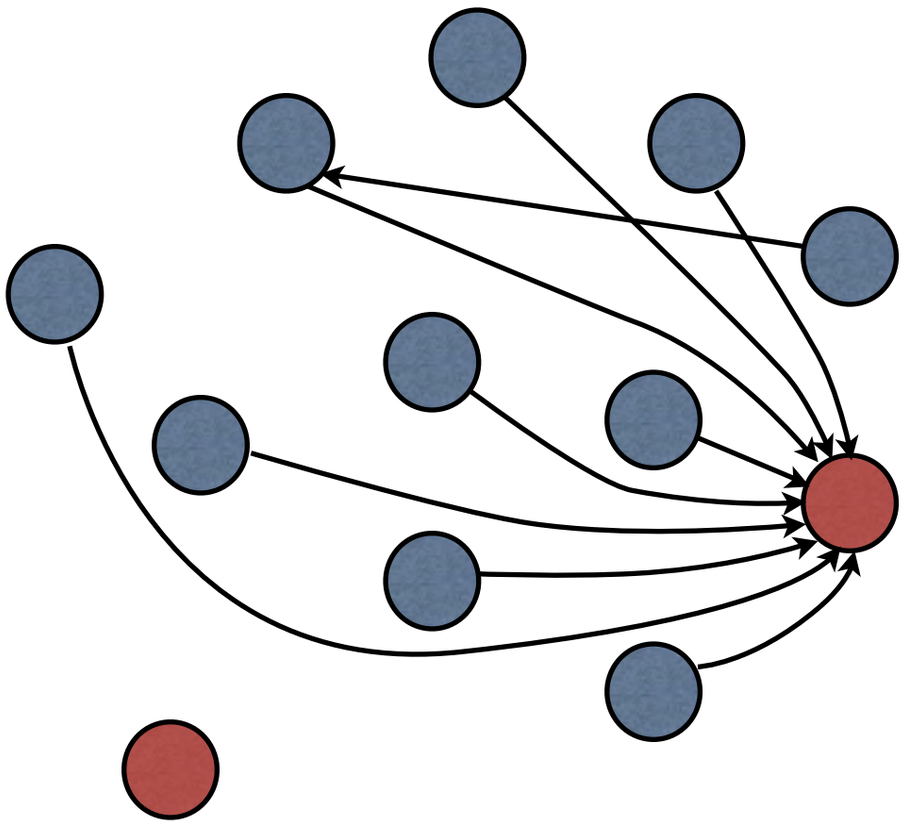


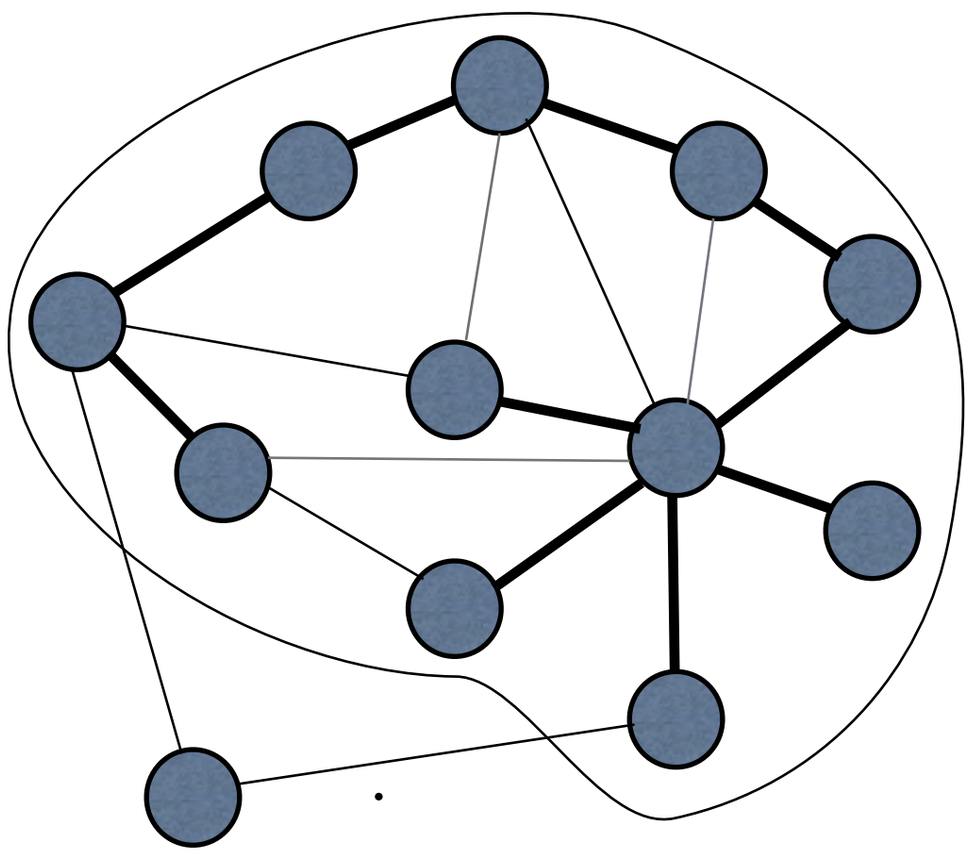


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

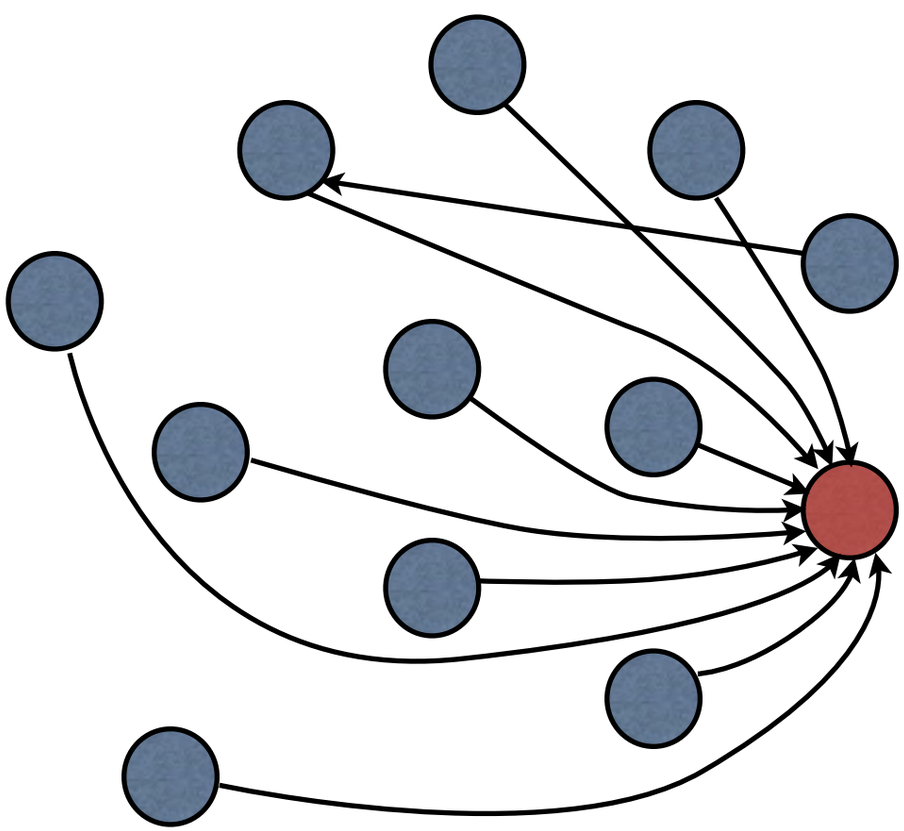


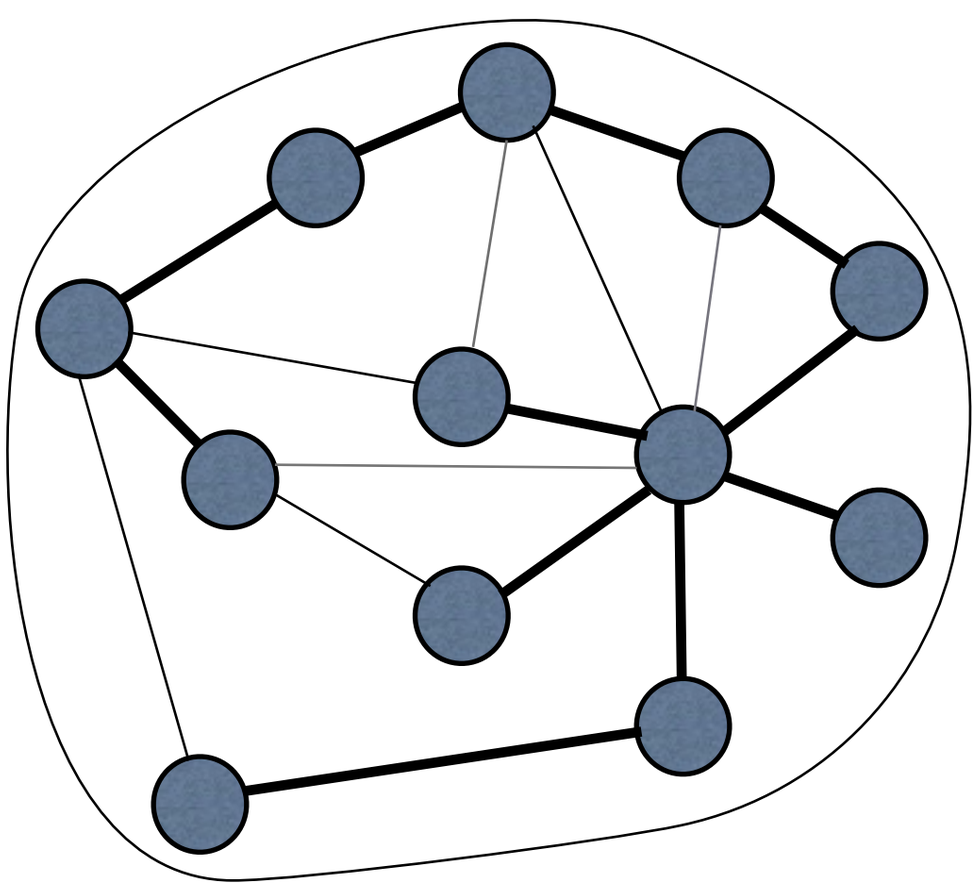


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus





Halmazerdő átlagos magassága

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

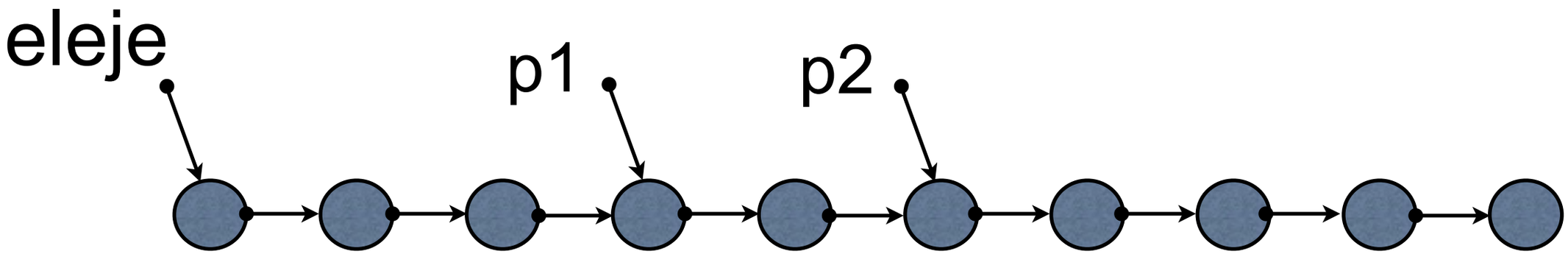
unio 1-el hosszabítja az utakat

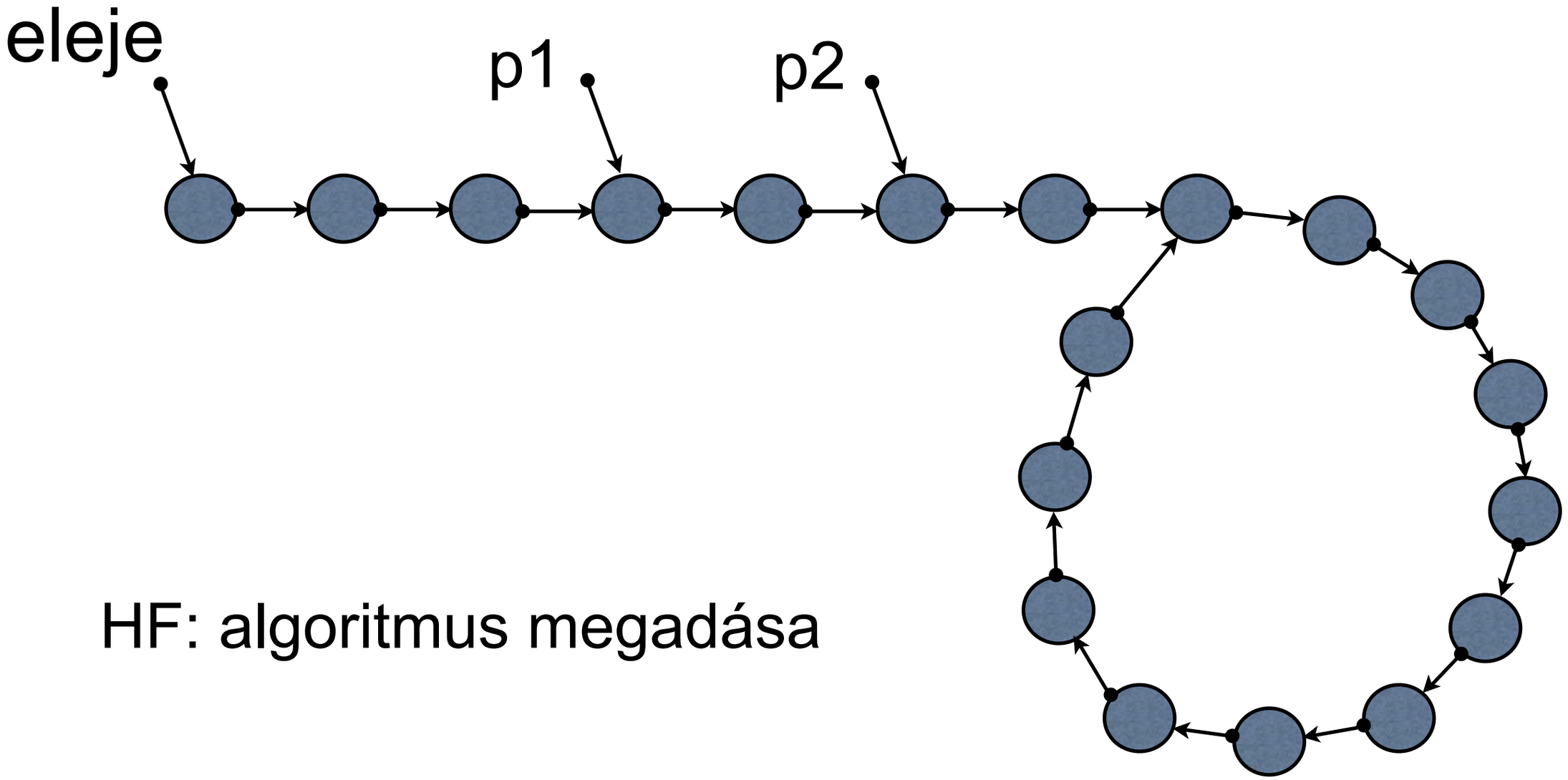
rep 1-hosszúra rövidíti az érintett pontokon az utakat

rep legalább 2-szer többször fut le mint unio

HF: halmazerdő átlagos magassága

Futási idő elemzése: **[2] 444-455**



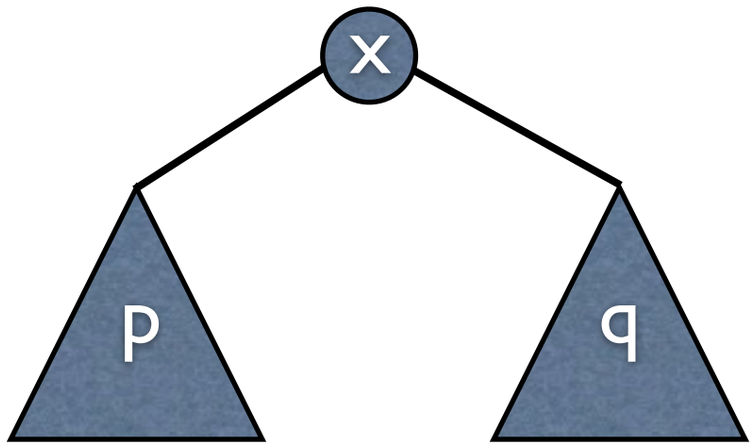


HF: algoritmus megadása

Bináris keresőfa

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

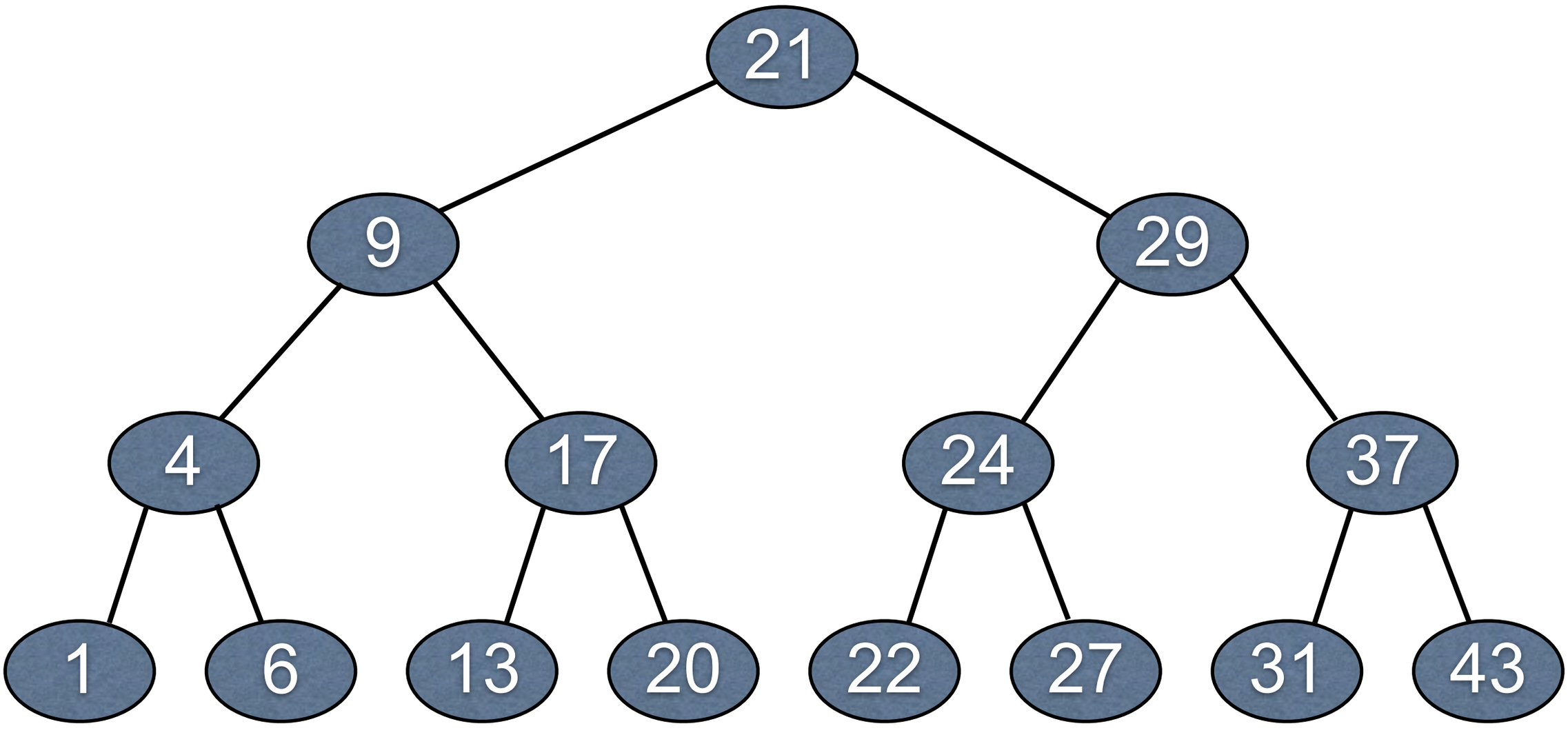
- F bináris fa, $R = \{bal, jobb, apa\}$, $bal, jobb, apa : M \to M$,
- $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
- $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(kulcs(p) \leq kulcs(x) \leq kulcs(q))$



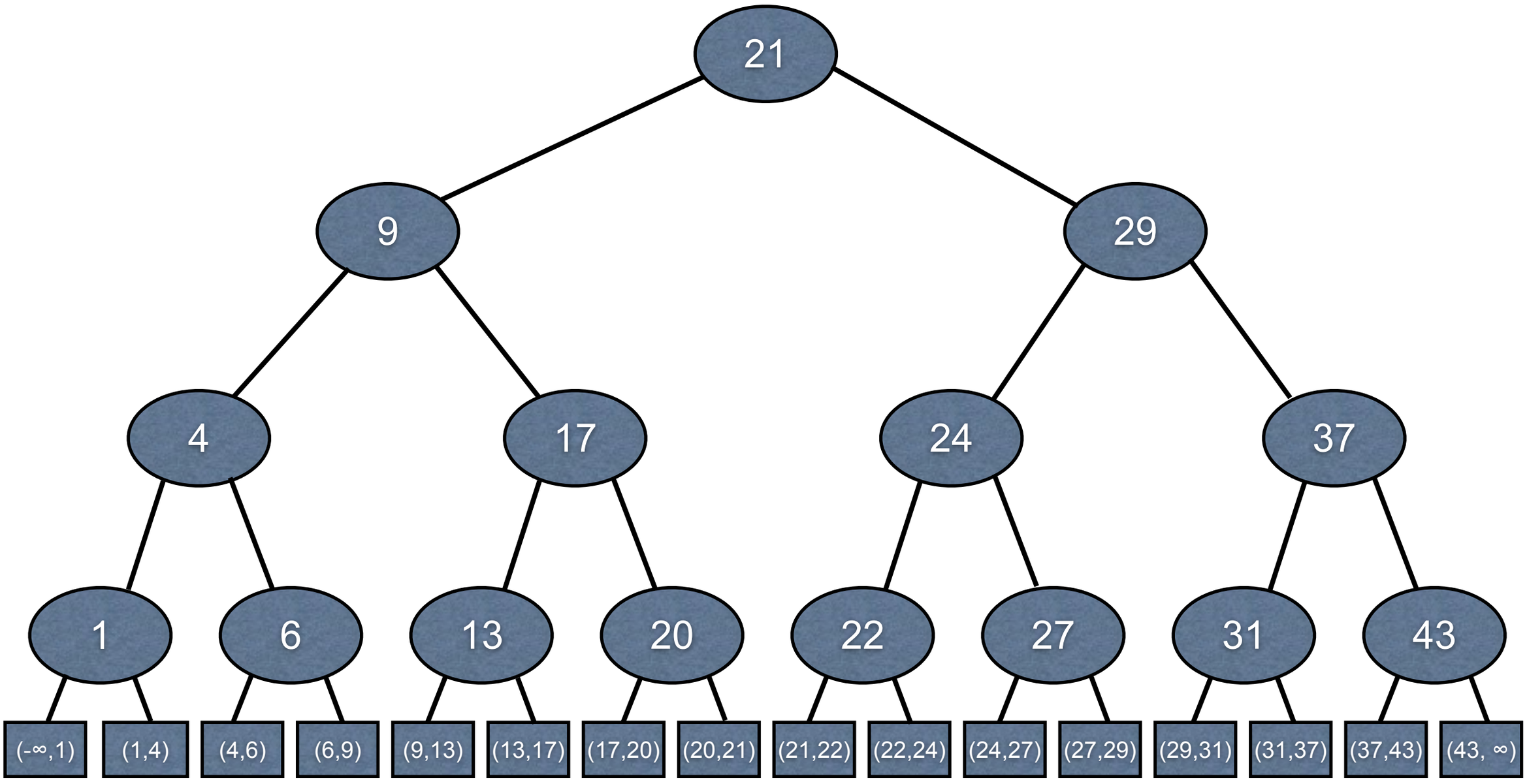
[2] 232-241

fázunk...

Bináris keresőfa



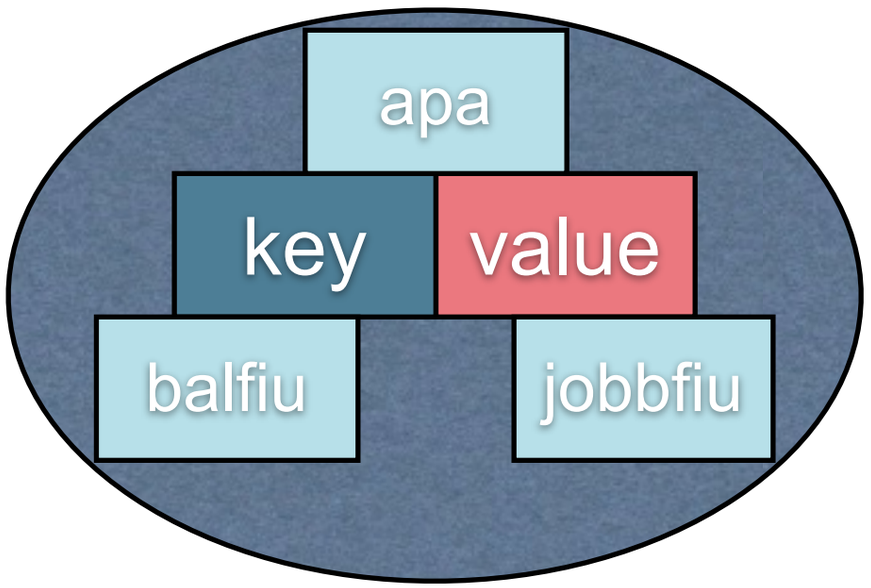
Bináris keresőfa



Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

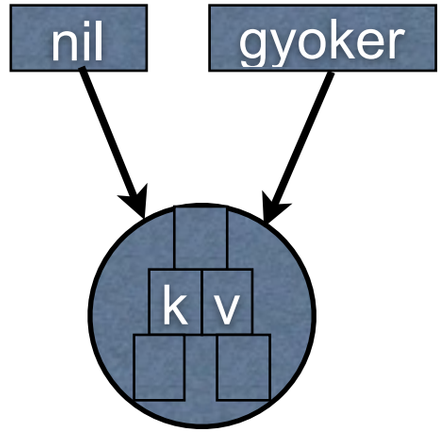
```
struct fapont{
    int key, value ;
    fapont bal, jobb, apa ;
}
```

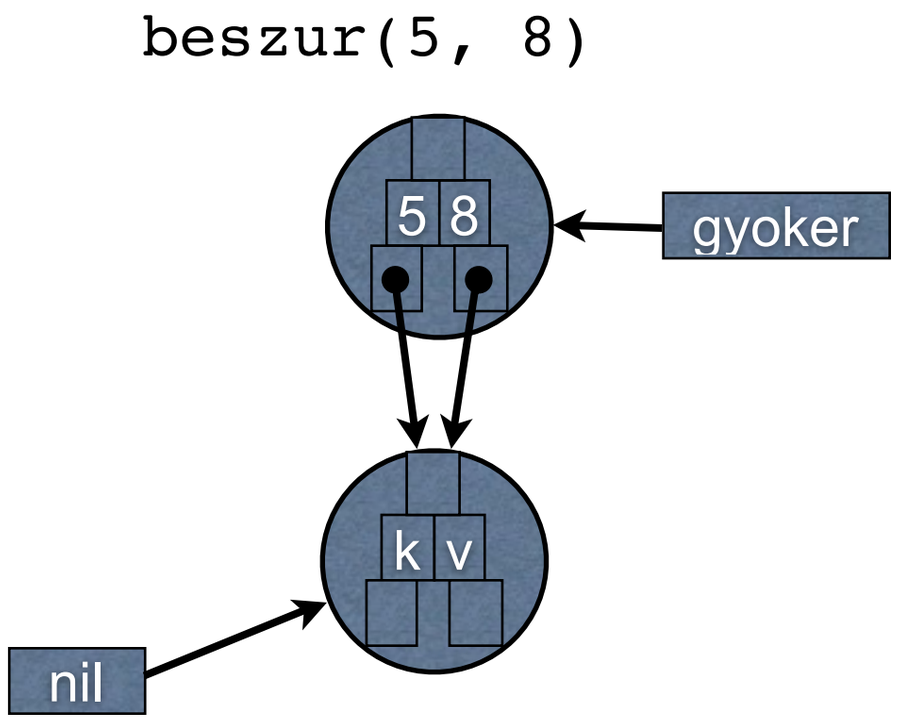


Bináris fa tárolása

első megközelítés, a gyakorlatban nem így van!

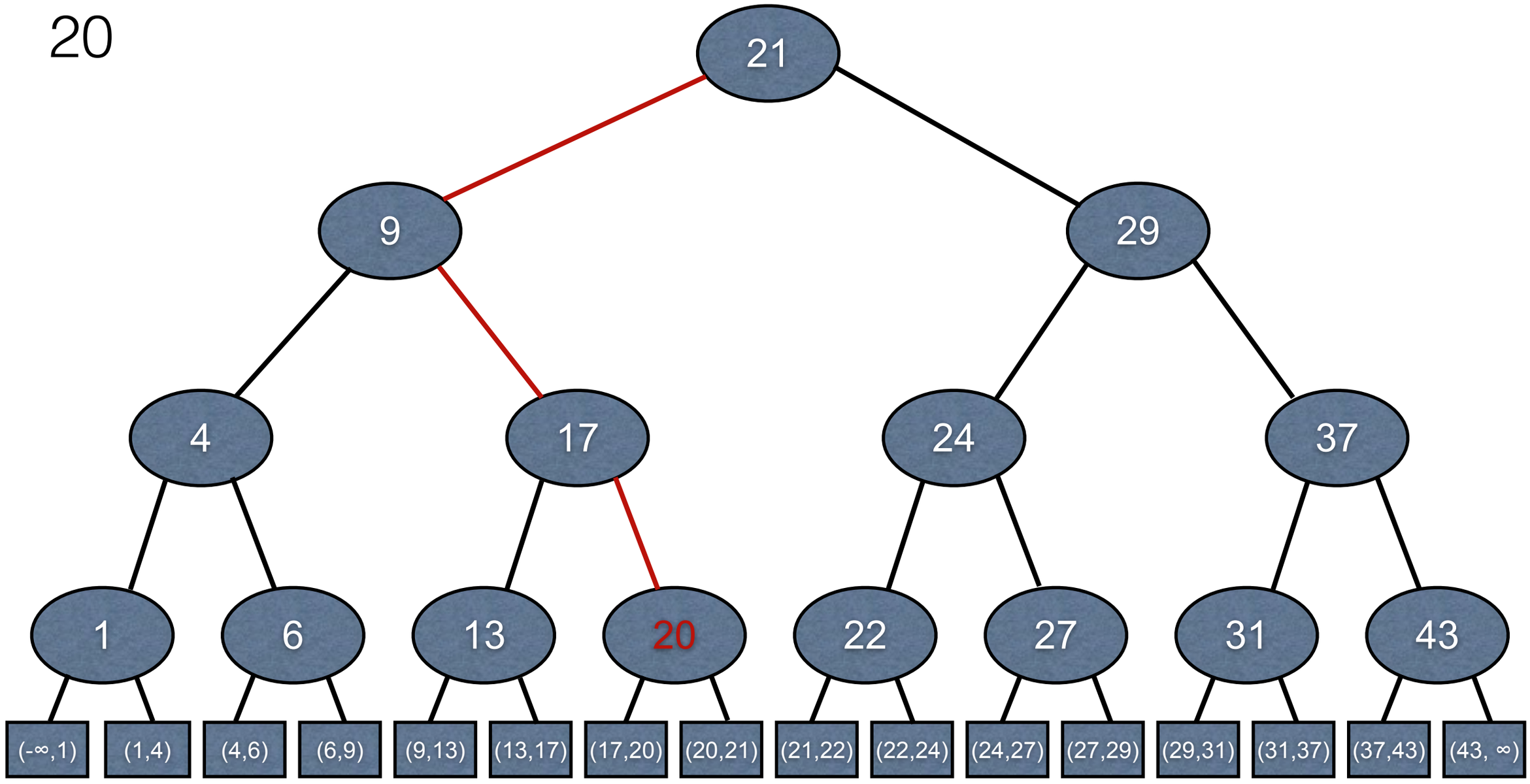
```
public class fa {
   int elemszam ;
   struct fapont{
      int key, value ;
      fapont bal, jobb, apa ;
   }
   fapont nil, gyoker ;
   void fa() {
      fapont q = new fapont() ;
      nil=q ;
      gyoker=q ;
      elemszam=0 ;
   }
   boolean beszur(int x, int v) {
   ...
  }
  ...
}
```





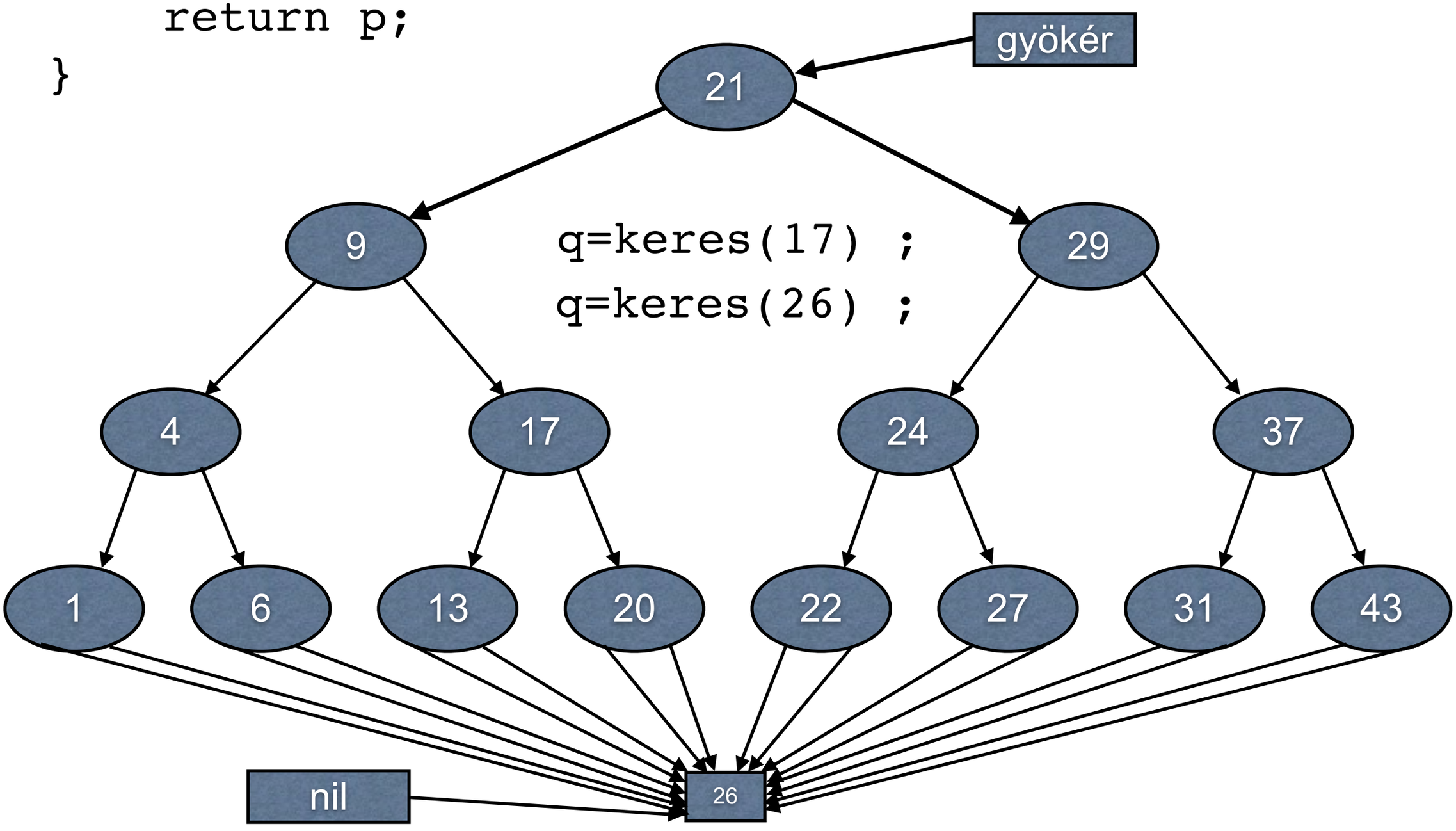
Keresés keresőfában

20

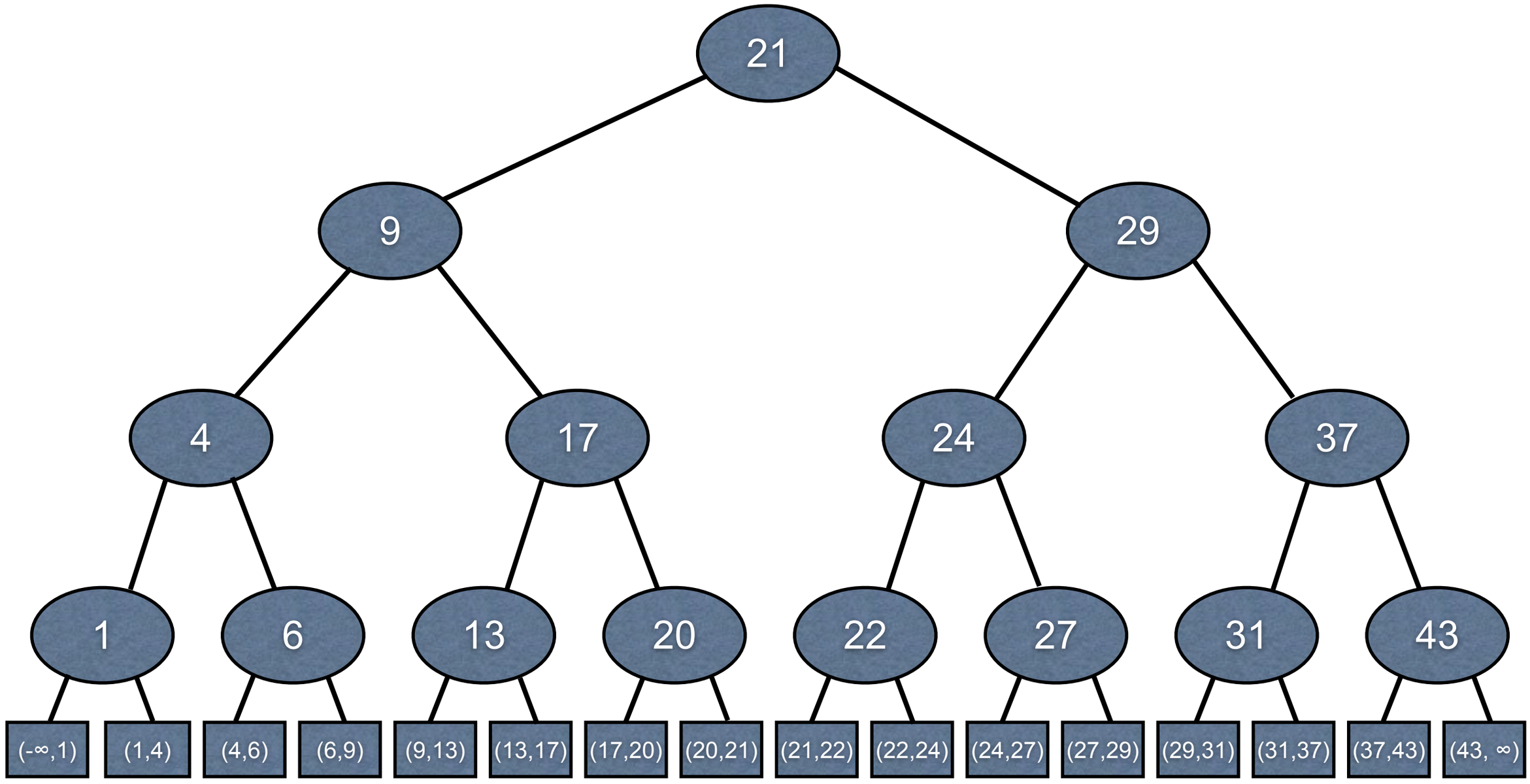


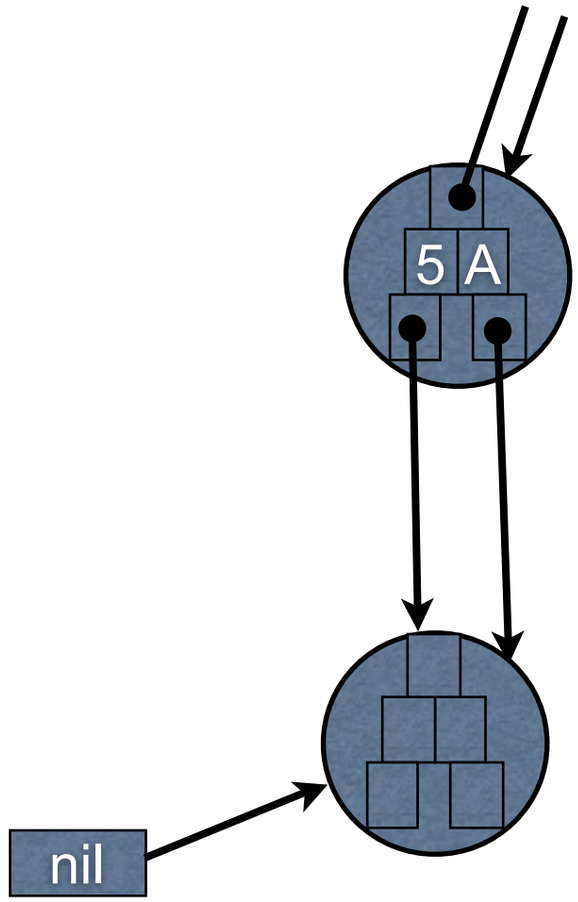
```
fapont keres(int x) {
     fapont p=gyoker ;
     nil.key=x ;
     while (p.key!=x) p=x<p.key?p.bal:p.jobb ;
     return p;
}
```

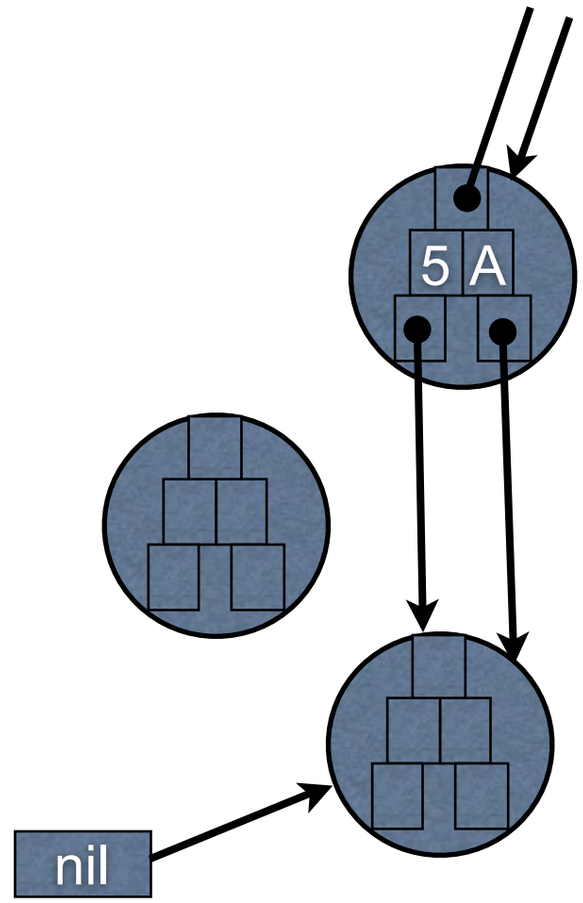
```
q=keres(17) ;
q=keres(26) ;
```

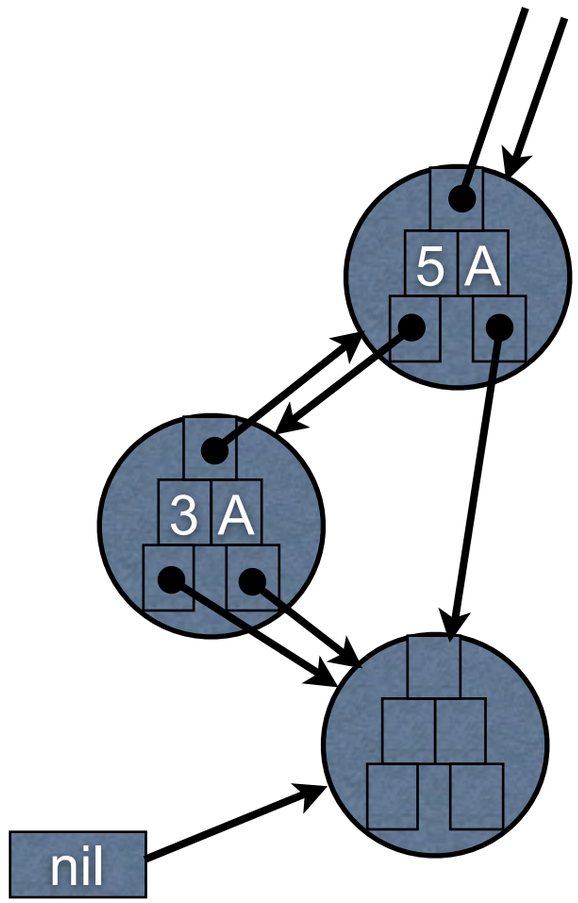


Beszúrás keresőfába

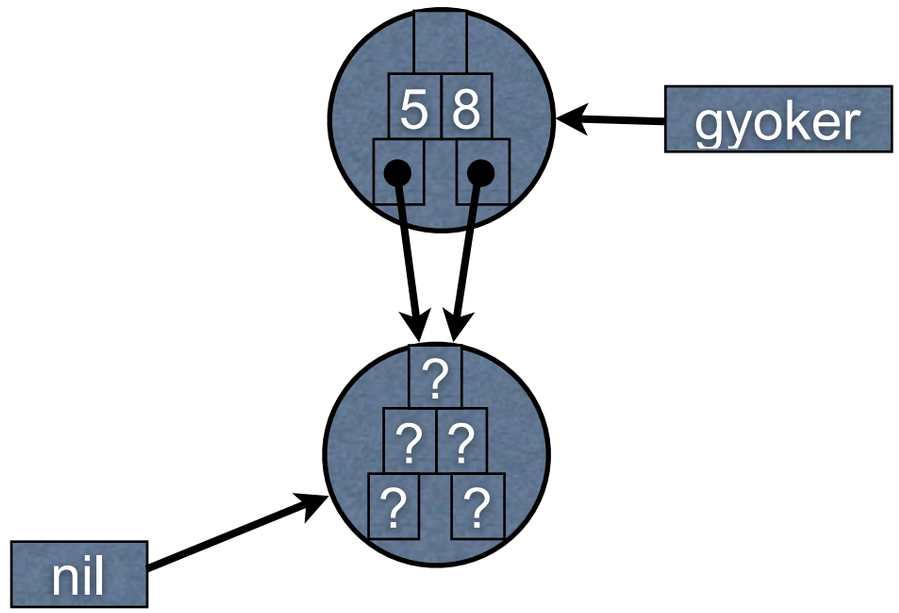



5 A
nil

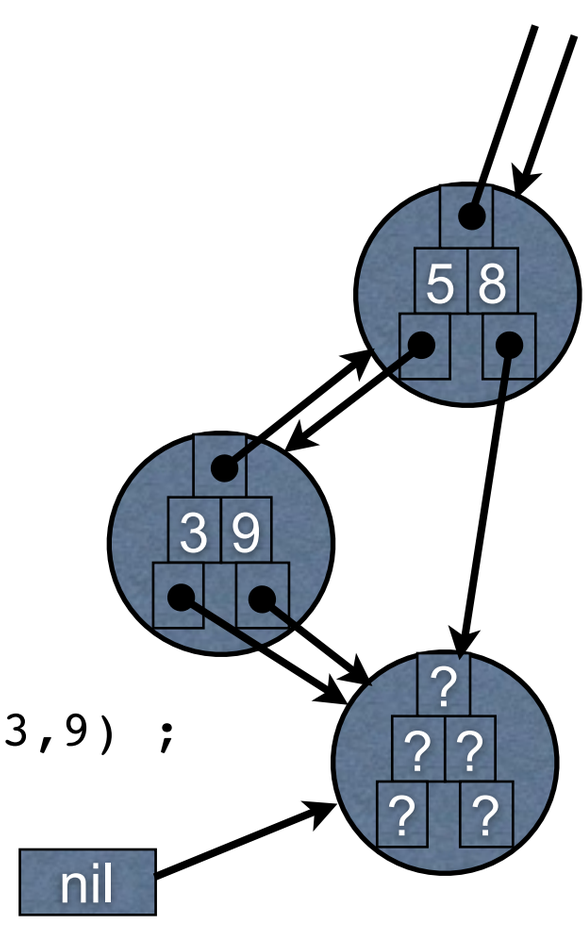

5 A
nil


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 ;
      }
}
```

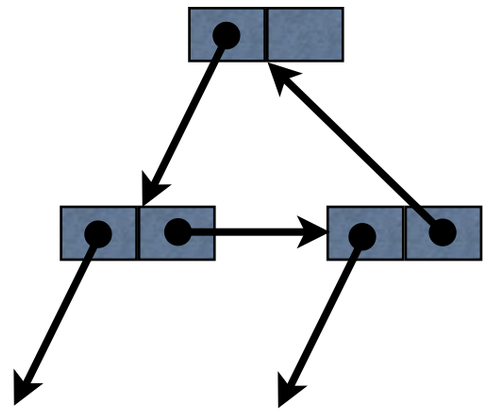


beszur(3,9) ;



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)

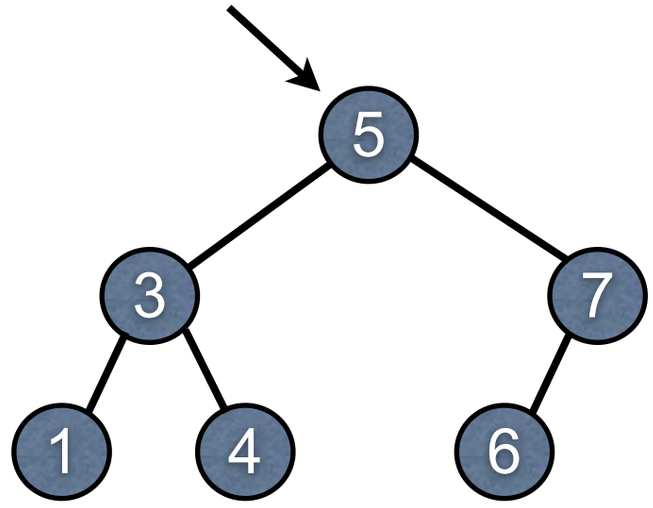
Bináris fa inorder bejárása

```
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)

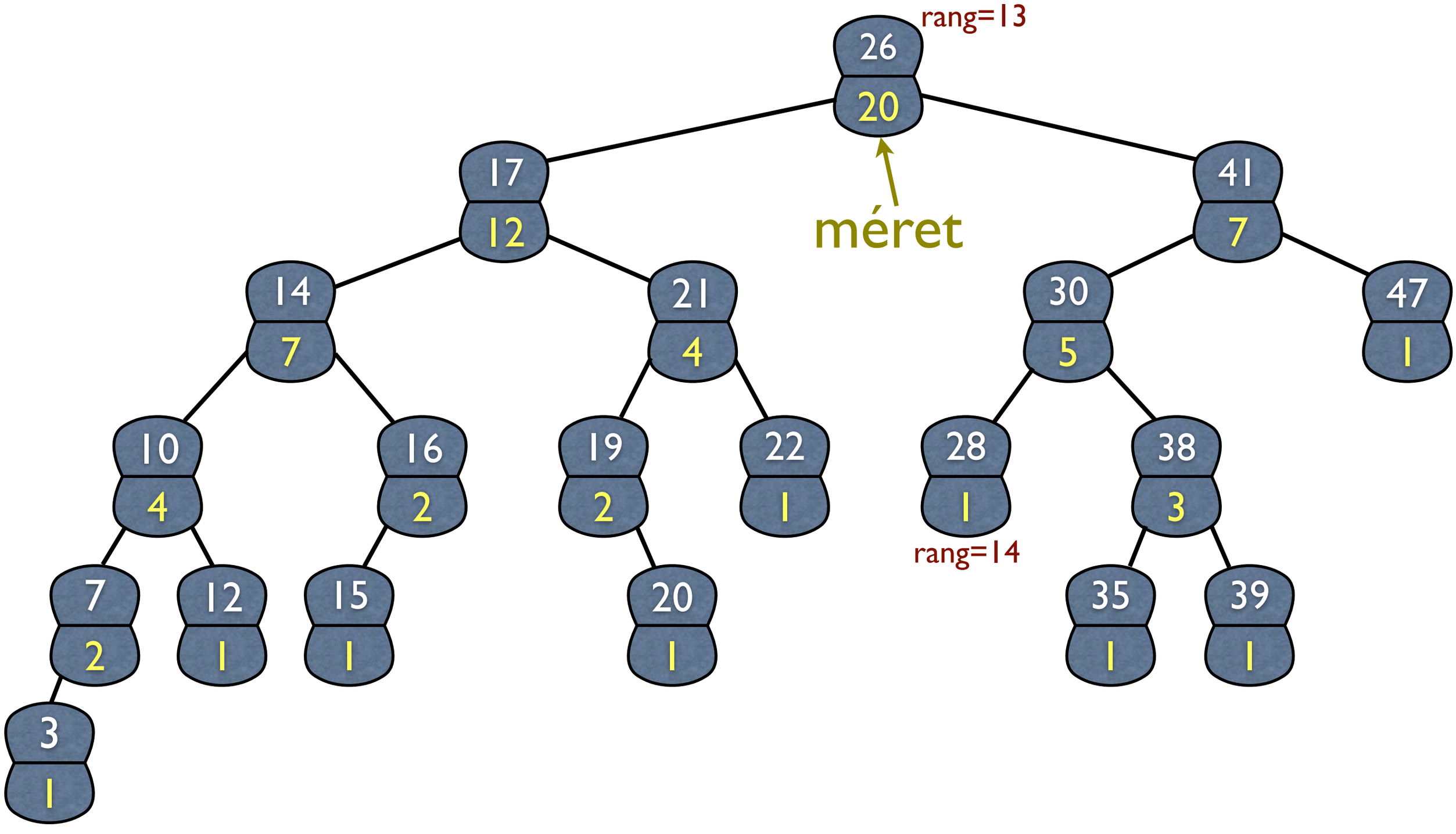
Bejárás - csökkenő sorrendben

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

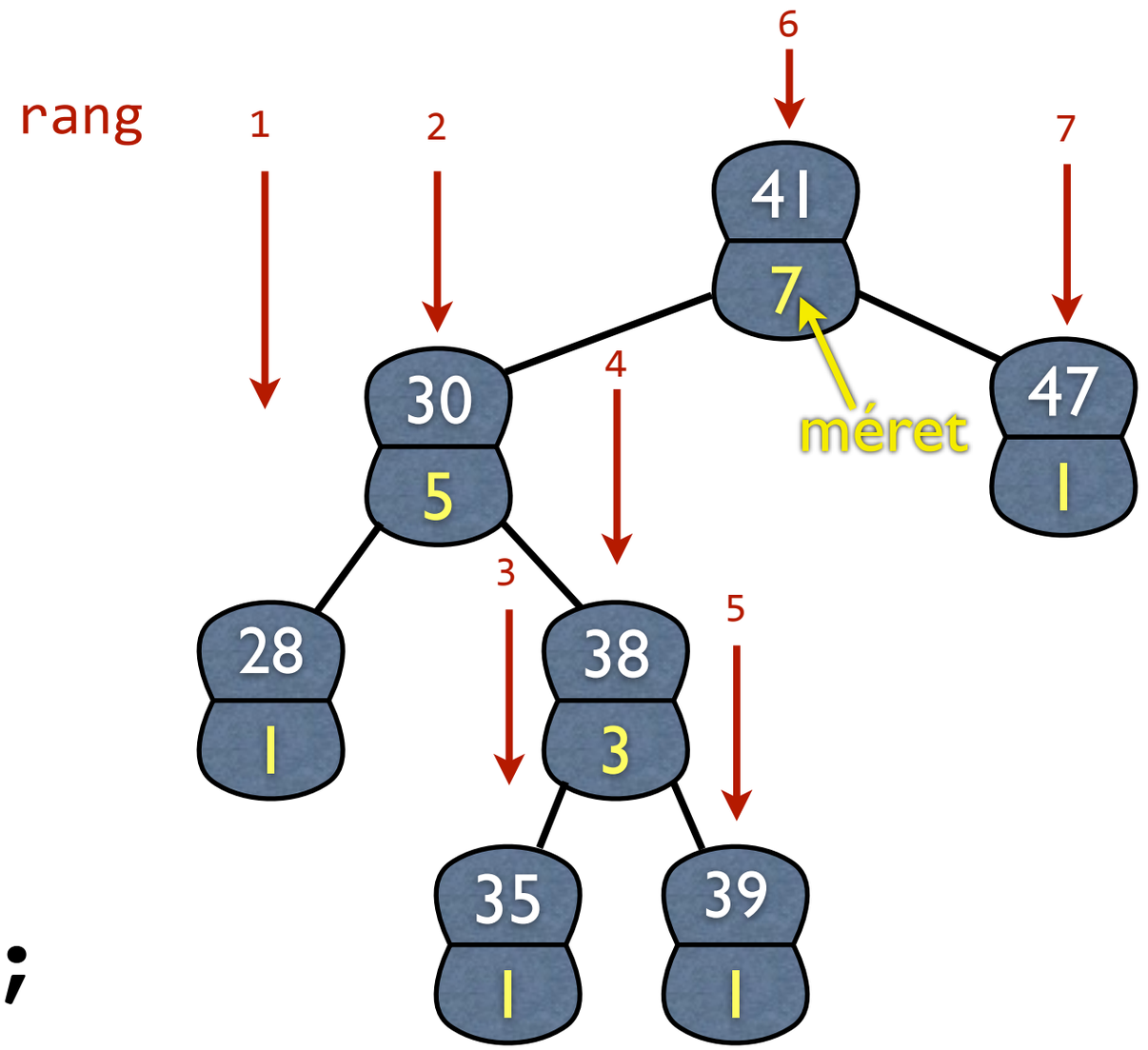
Rendezettminta-fa

- Minden p ponthoz tároljuk a p gyökerű fa belső pontjainak számát (méretét)
- Adott elem rangja: az elem sorszáma (sorrendben hányadik az adatszekezetben)
- Adott rangú elem keresése - T[r]
- Adott elem rangjának meghatározása **[2] 272-277**
- Egyéb bővítési lehetőségek (pl. intervallumfák) **[2] 279-285**

Példa


rang=13
26
20
méret
17
12
41
7
14
7
21
4
30
5
47
1
10
4
16
2
19
2
22
1
28
1
38
3
rang=14
7
2
12
1
15
1
20
1
35
1
39
1
3
1

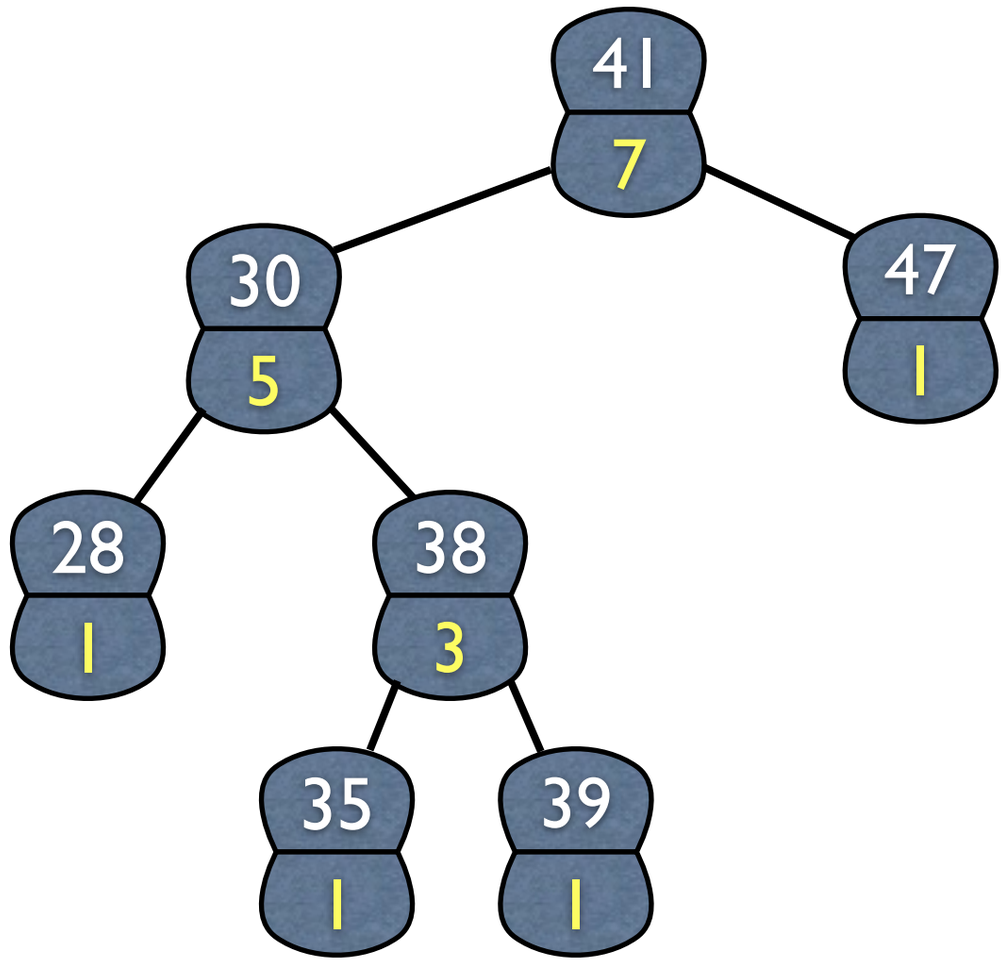
Adott rangú elem keresése




```
rmkeres(p,x) {
  r=p.bal.meret+1 ;
  if (x==r) return p ;
  if (x<r) return rmkeres(p.bal,x) ;
  else return rmkeres(p.jobb,x-r) ;
}
```

O(h)

Elem rangjának meghatározása



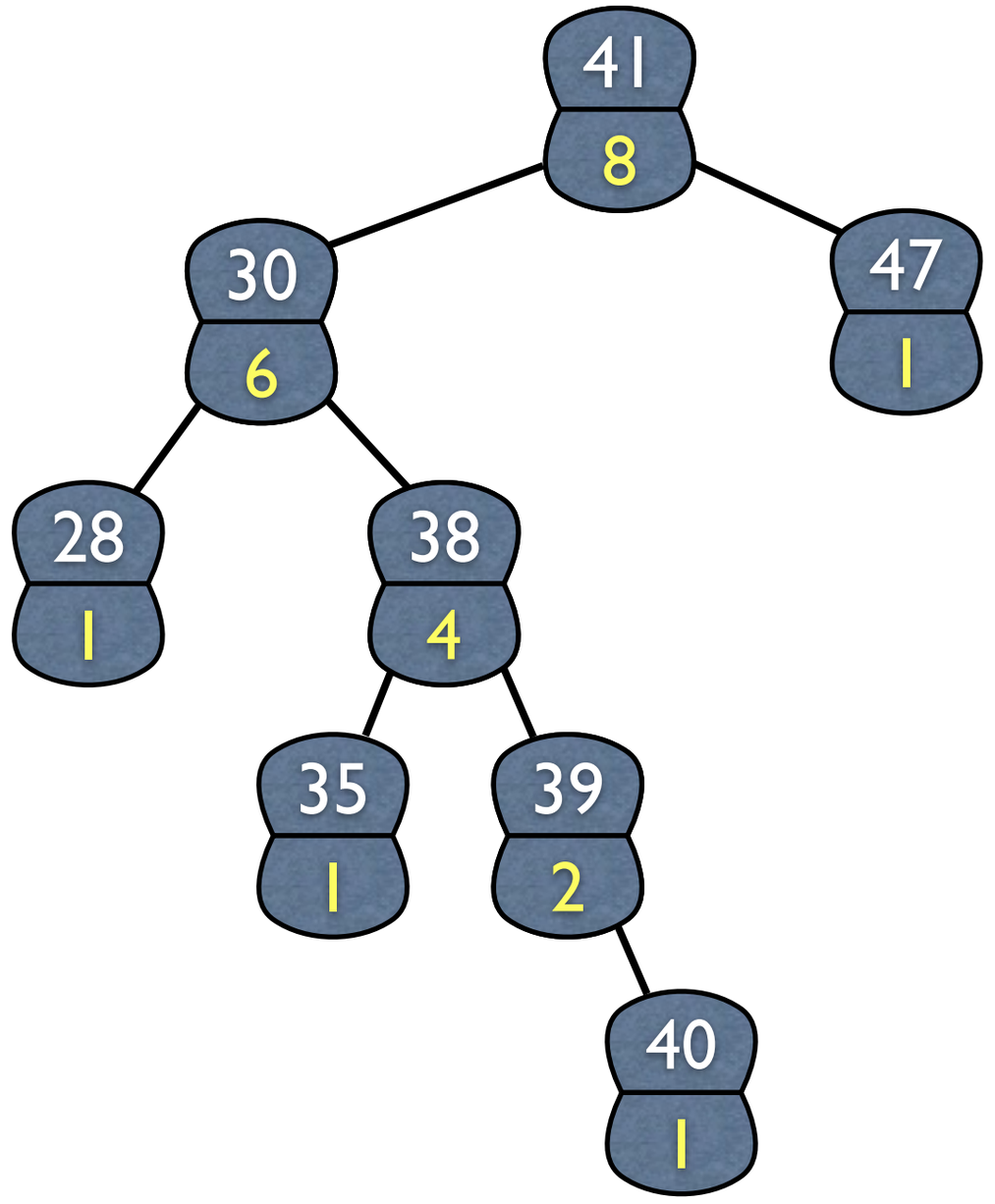

```
rmrang(p) {
  r=p.bal.meret+1 ;
  q=p ;
  while (q!=gyoker) {
    if (q==q.apa.jobb) r+=q.apa.bal.meret+1;
    q=q.apa ;
  }
  return r ;
}
```

O(h)

Méret információ fenntartása

```
beszjav(p) {
  while (p!=gyoker) {
    p=p.apa;
    p.meret++ ;
  }
}

torljav(p) {
  while (p!=gyoker) {
    p=p.apa;
    p.meret-- ;
  }
}
```

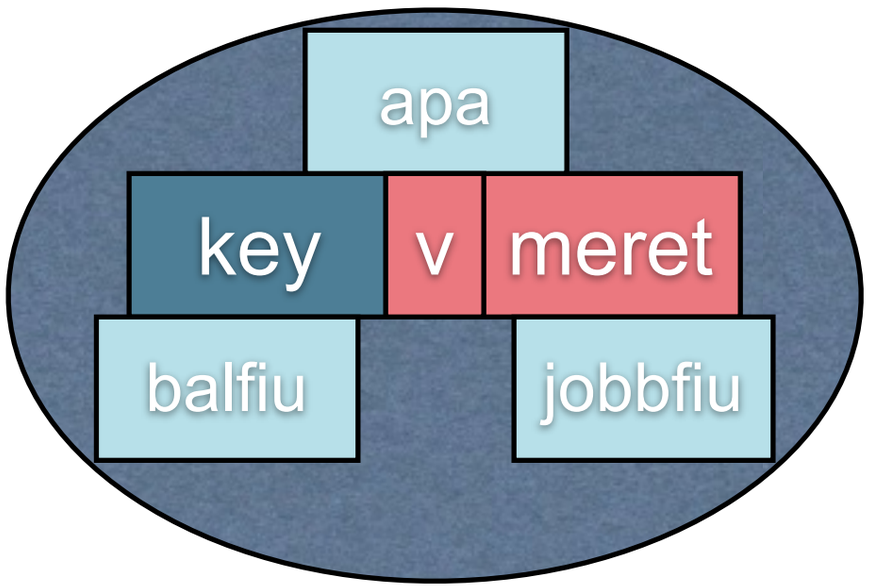


O(h)

Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

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

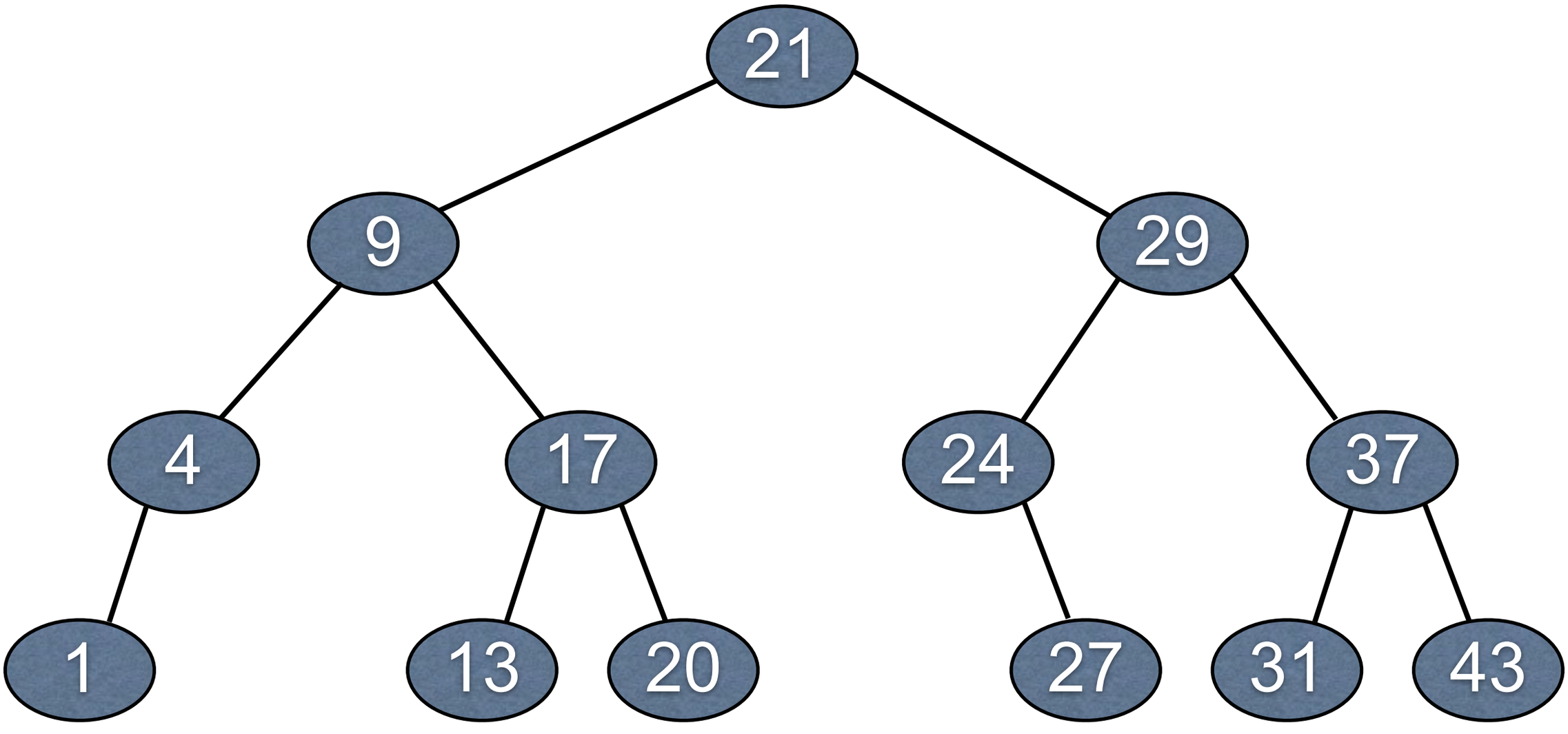


Rendezettmintafa

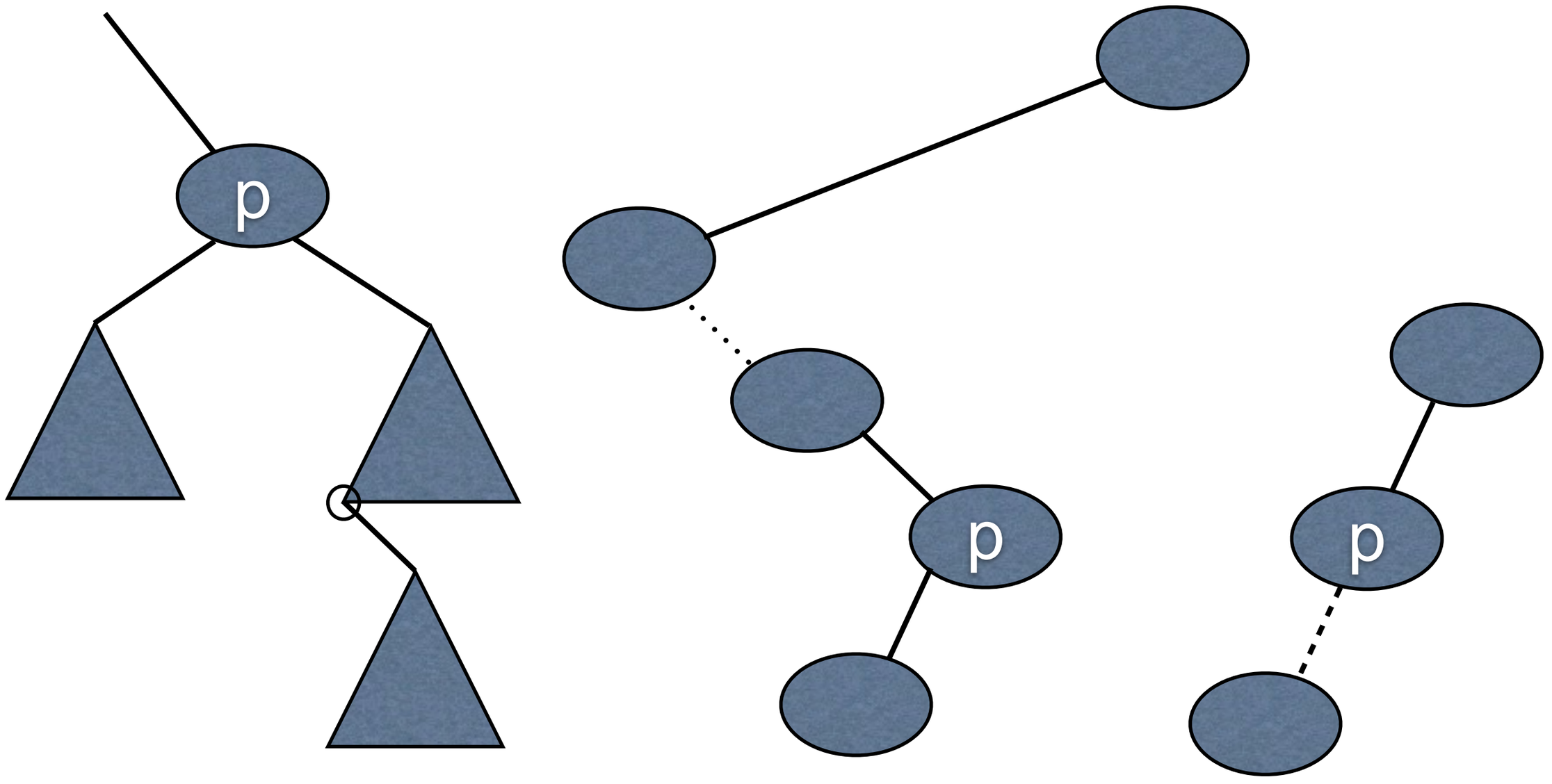
Műveletek:

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*)**
int *elemszám*(fapont p)	**O(*1*)**
int *rang*(fapont p)	**O(*h*)**
fapont *rangkeres*(int x)	**O(*h*)**

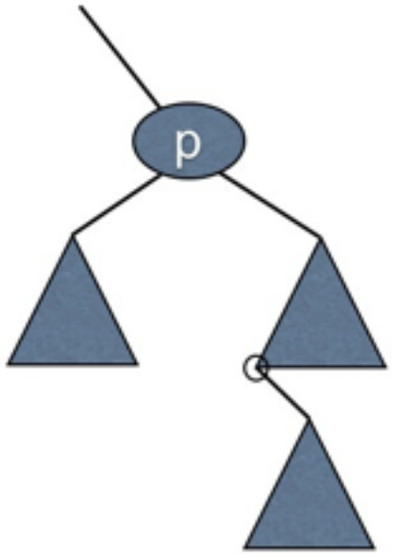
Elem rákövetkezője



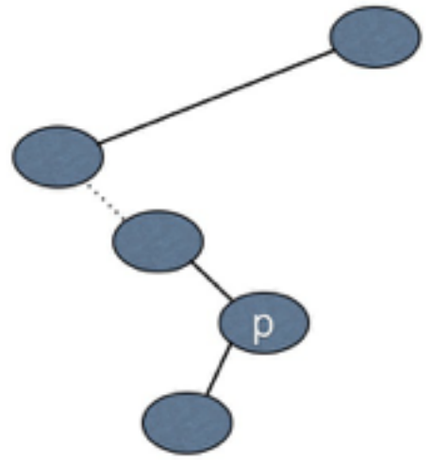
p rákövetkezője



O(h)



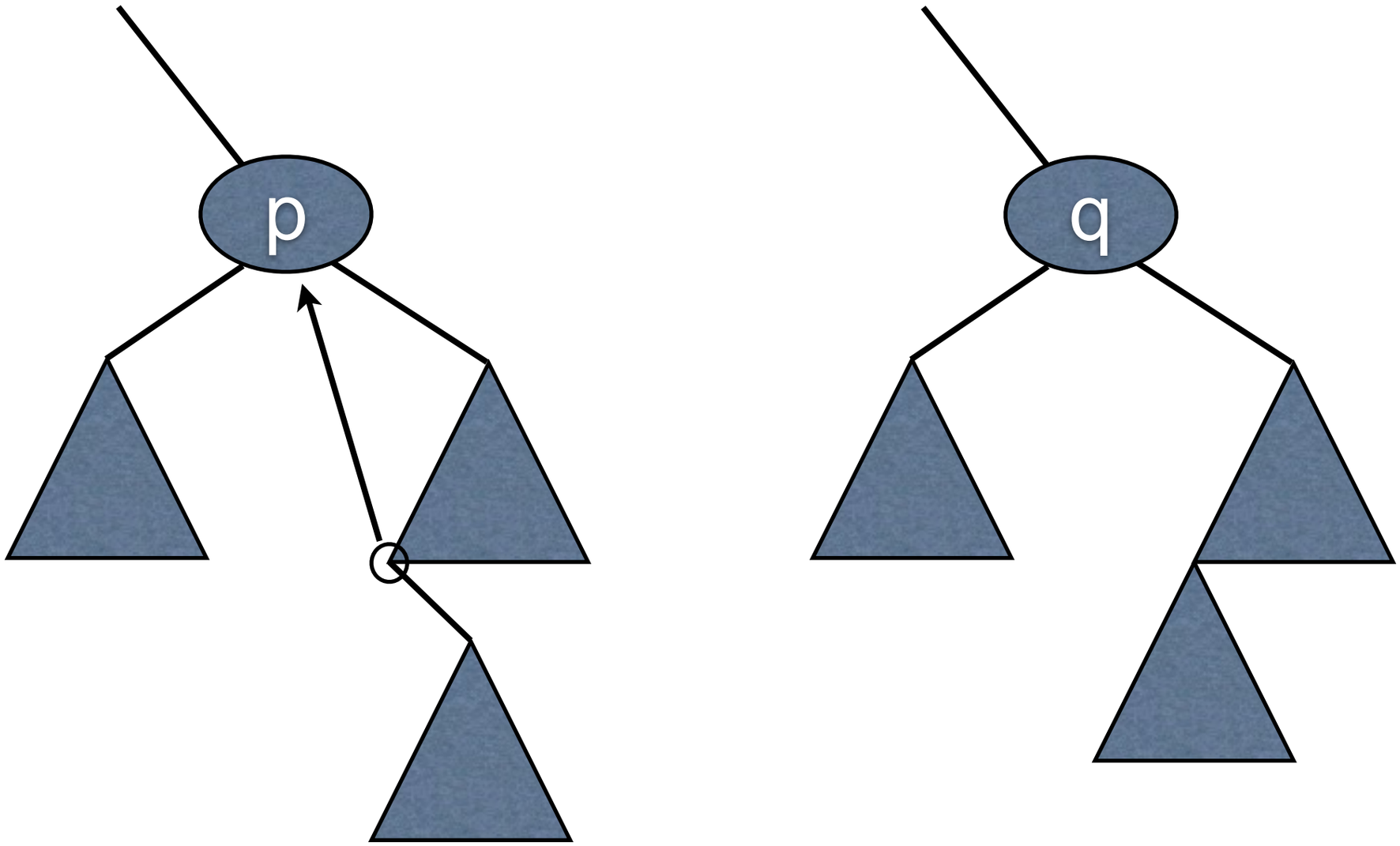
p rákövetkezője




```
fapont kovetkezo(fapont p) {
     if (p.jobb!=nil) {                  //ha van jobb fiú
       p=p.jobb ;                        //akkor a jobb részfa
       while (p.bal!=nil) p=p.bal;       //bal szélső eleme
       return p;
     } else {                            // ha nincs jobb fiú
       while (p.apa!=nil) {              // amíg van apa
          if (p.apa.bal==p)              // ha p bal gyerek
             return p.apa;               // p apja a rákövetkező
          p=p.apa ;                      // megyünk felfelé p-vel
       };
       return nil ;                      // nincs rákövetkező
     }
}
```

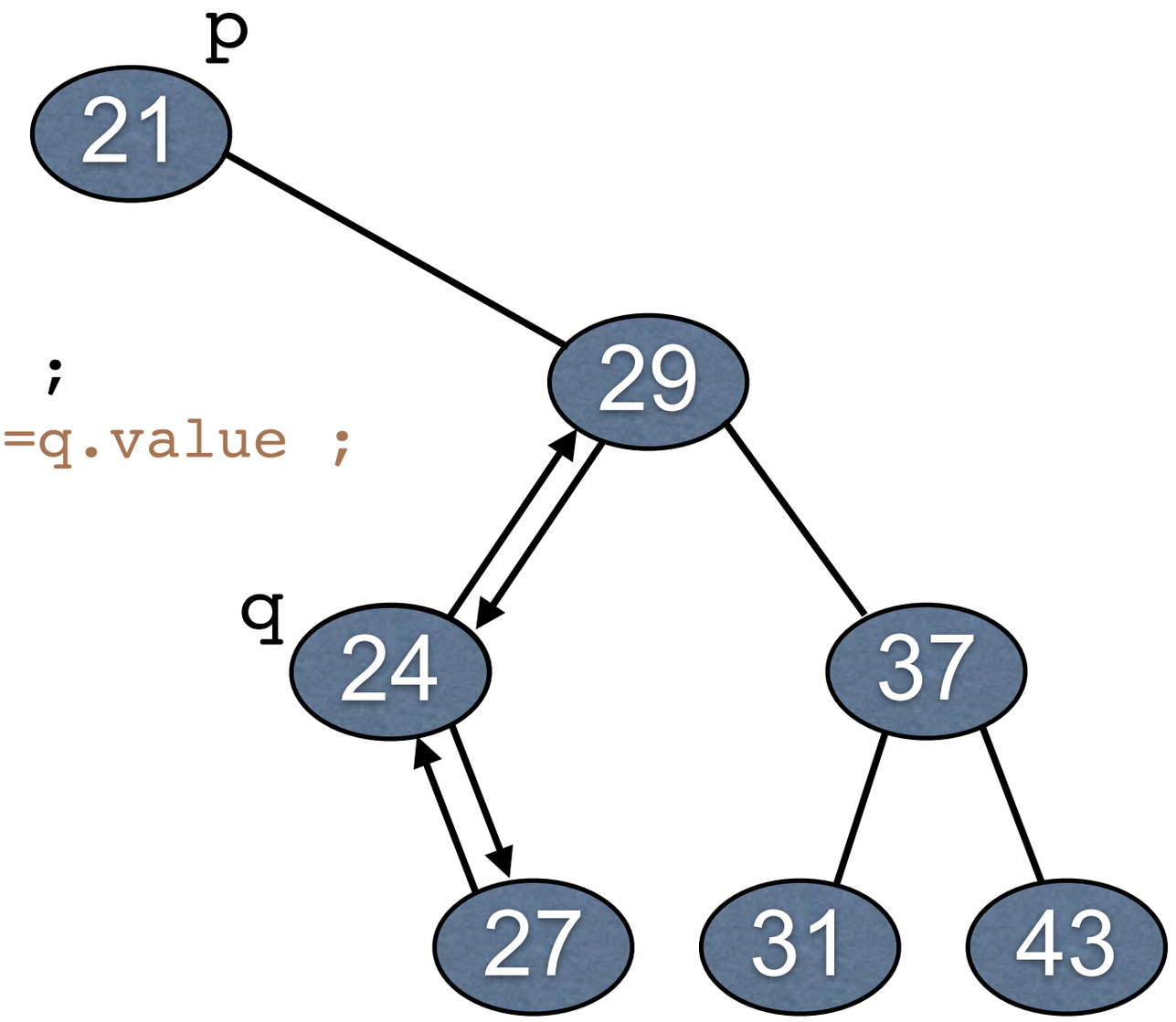
O(h)

Törlés bináris keresőfából



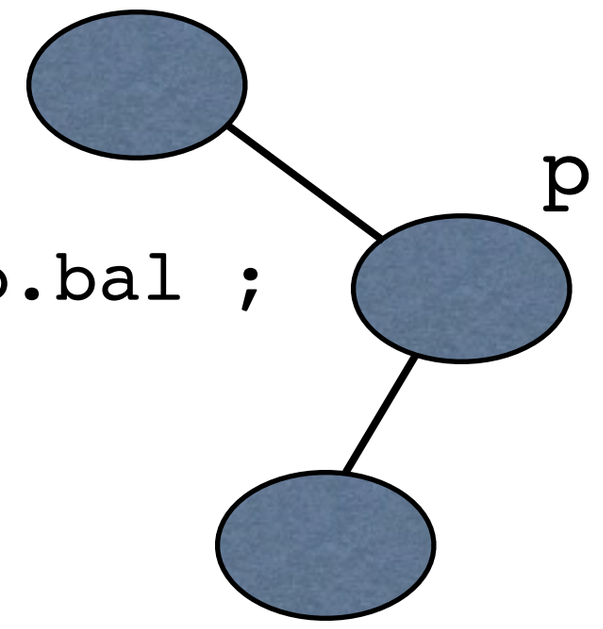
O(h)

```
void torol(fapont p) {
   if (p.jobb!=nil) {
      fapont q=kovetkezo(p) ;
      p.key=q.key ; p.value=q.value ;
      q.apa.bal=q.jobb ;
      q.jobb.apa=q.apa ;
      delete q ;
   } else {
      ...
   }
}
```



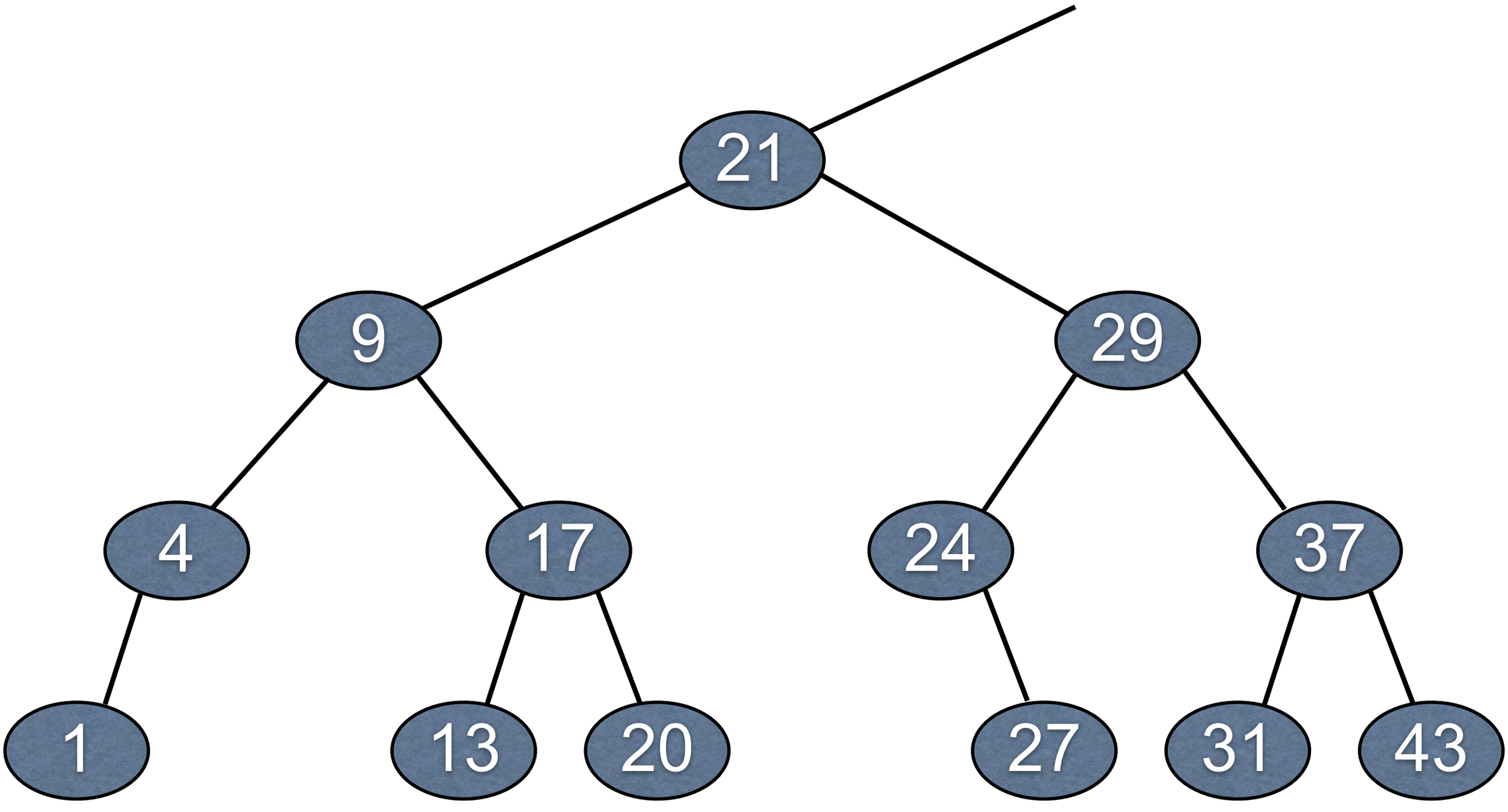
O(h)

```
void torol(fapont p) {
    if (p.jobb!=nil) {
      fapont q=kovetkezo(p) ;
      p.key=q.key ; p.value=q.value ;
      q.apa.bal=q.jobb ;
      q.jobb.apa=q.apa ;
      delete q ;
    } else {
      if (p.apa.jobb==p) p.apa.jobb=p.bal ;
      else p.apa.bal=p.bal ;
      p.bal.apa=p.apa ;
      delete p ;
    }
}
```



O(h)

Elem törlése példa

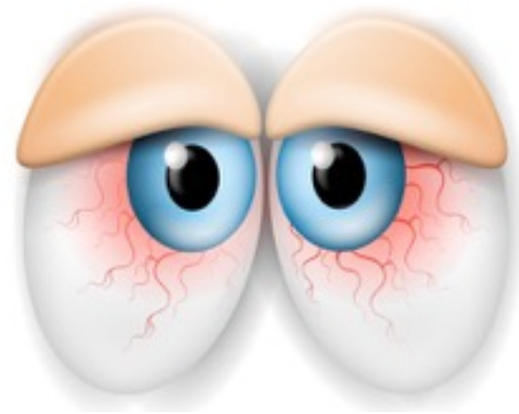


Futási idők elemzése

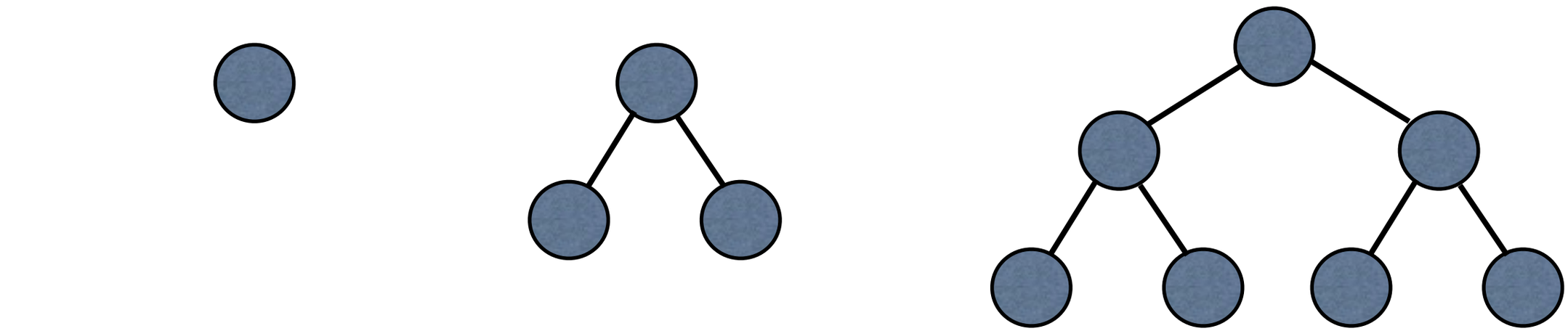
- Keresés
- Beszúrás
- Törlés

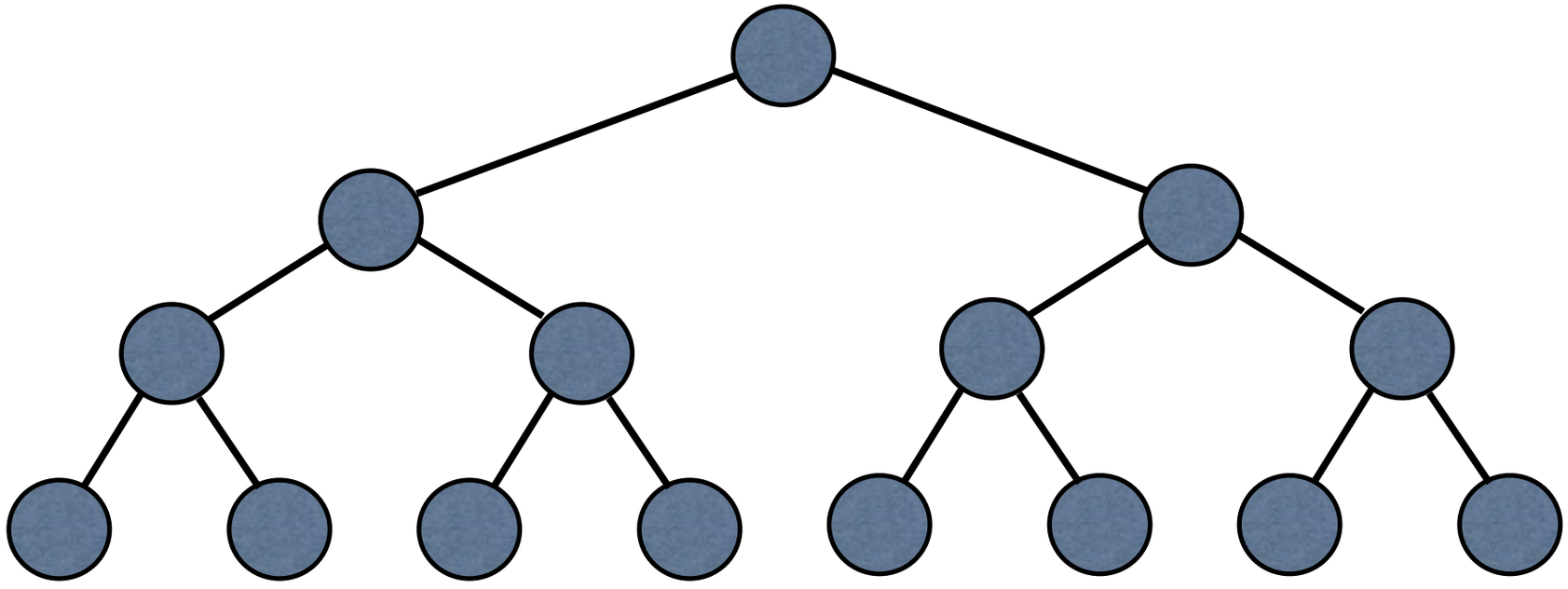
Mennyi a futási idő?

$O(h)$



Teljes bináris keresőfa





h	n(h)	2^h
1	**1**	2
2	**3**	4
3	**7**	8
4	**15**	16
5	**31**	32
k	**$2^k - 1$**	2^k

n pontot tartalmazó teljes bináris keresőfa magassága

h	n(h)	2^h
1	1	2
2	3	4
3	7	8
4	15	16
5	31	32
h	2^h-1	2^k

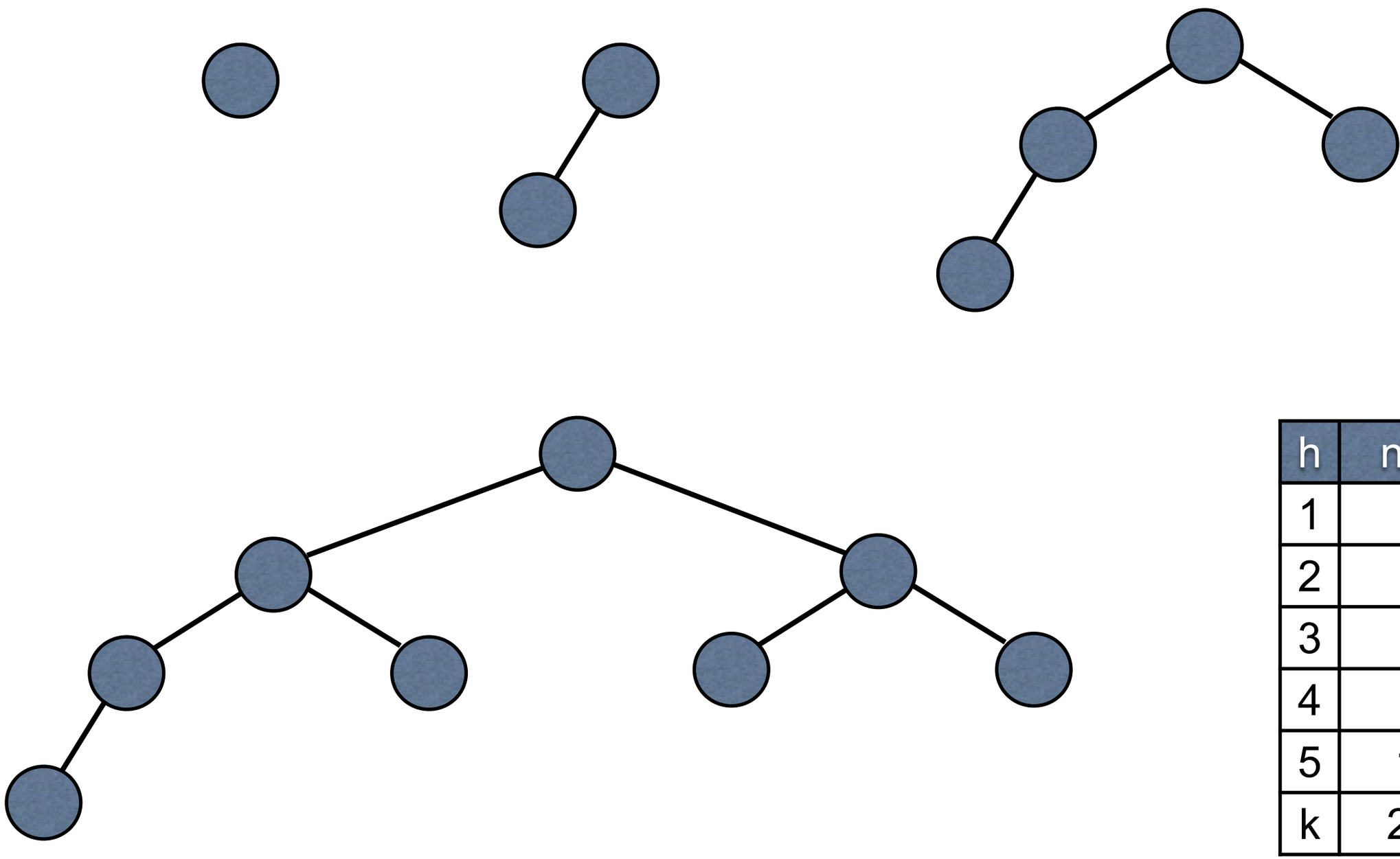
$$n = 2^h - 1$$

$$n + 1 = 2^h$$

$$\log_2(n+1) = h$$

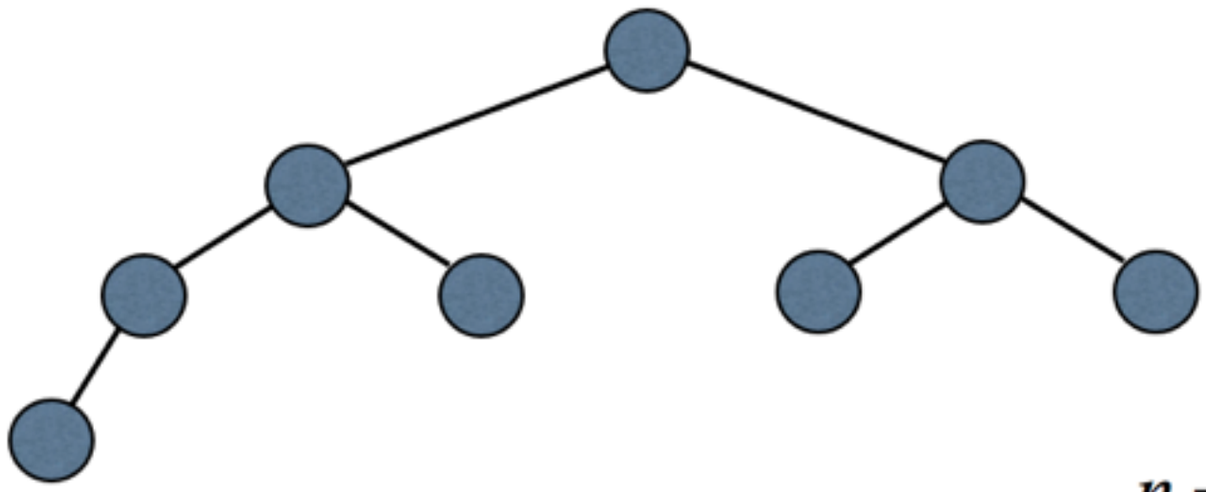
$$h = O(\log n)$$

Majdnem teljes bináris keresőfa



h	n(h)	2^{h-1}
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
k	2^{k-1}	2^{k-1}

n pontot tartalmazó majdnem teljes bináris keresőfa magassága



h	n(h)	2^{h-1}
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
k	2^{k-1}	2^{k-1}

$$n = 2^{h-1}$$

$$\log_2 n = \log_2 2^{h-1} = (h-1)\log_2 2 = h-1$$

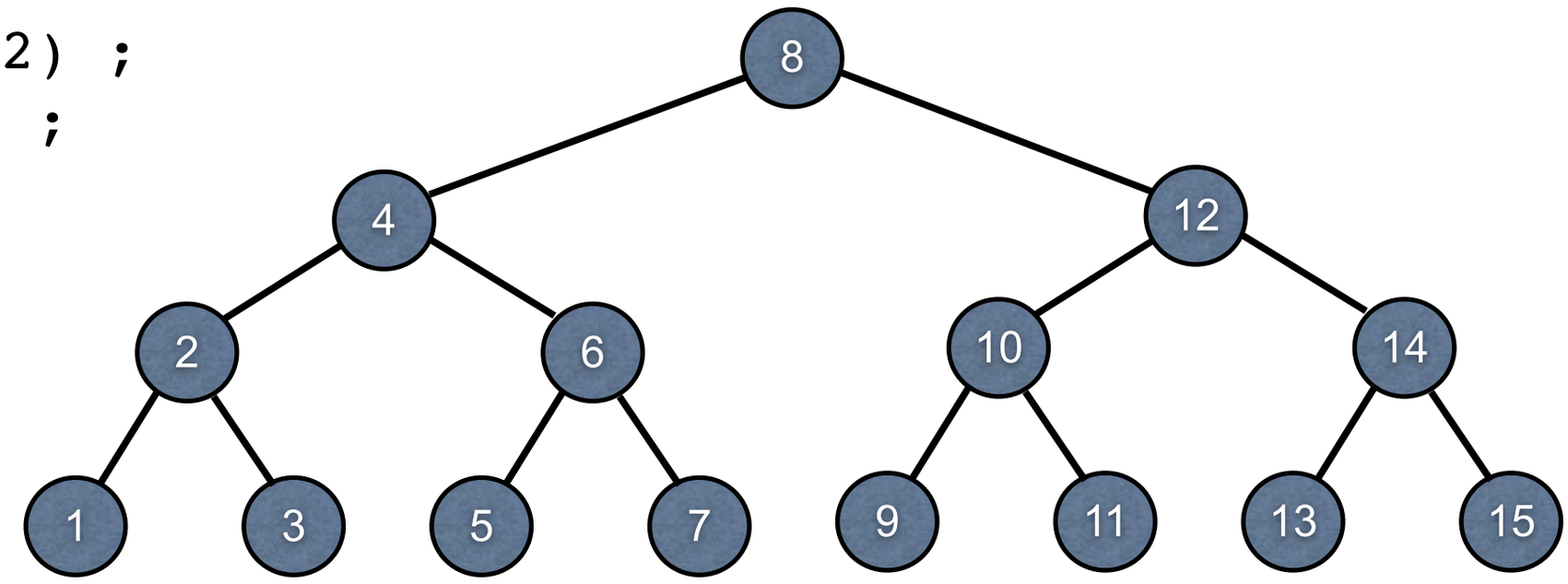
$$\log_2 n + 1 = h$$

$$\mathrm{h}=\mathrm{O}(\mathit{logn})$$

Majdnem teljes bináris keresőfa építése

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
f(i,j) {
   if (i<=j) {
      f=int((i+j)/2) ;
      beszur(T[f]) ;
      f(i,f-1) ;
      f(f+1,j) ;
   }
}
```



Építés: O (n · *log* n)

Optimális bináris keresőfa

Adott kulcsoknak egy $K=(k_1,...,k_n)$ sorozata

Minden k_i kulcshoz ismert annak p_i előfordulási valószínűsége és

Minden $d_i=(k_i,k_{i+1})$ intervallumhoz ismert annak q_i előfordulási valószínűsége, hogy arra az intervallumra keresünk, $d_0= (-\infty,k_i)$, $d_n= (k_n,\infty)$

Optimális bináris keresőfa építhető
(pl. dinamikus programozás módszerével)

Építés: $O(n^2)$

[2] 314-321

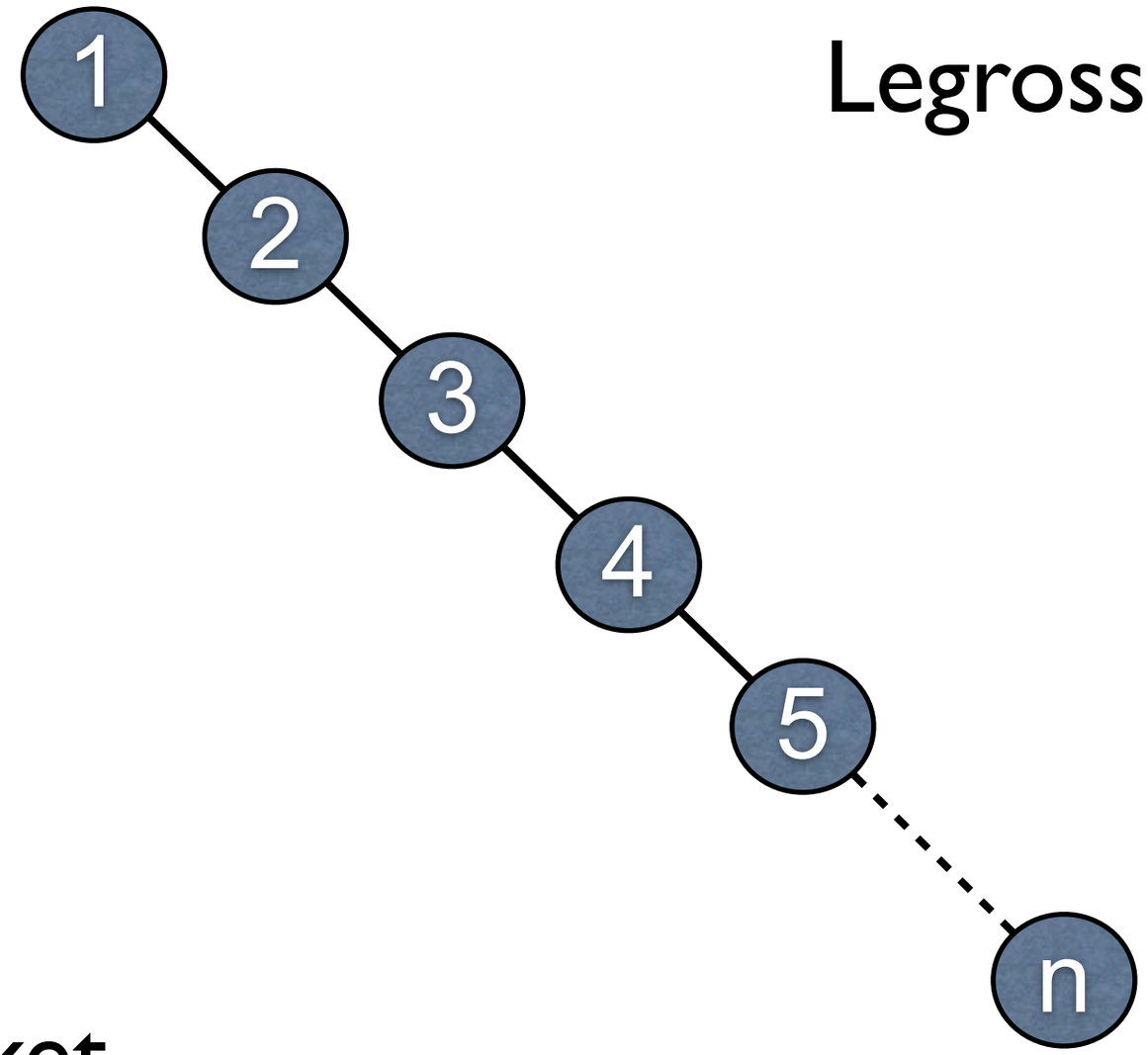
Véletlen építésű bináris keresőfa

- Adott n egymástól különböző kulcs, melyekből bináris keresőfát építünk úgy, hogy a kulcsokat valamilyen sorrendben egymás után beszúrjuk a kezdetben üres fába.
- Ha itt minden sorrend, vagyis az n kulcsnak mind az n! permutációja egyformán valószínű, akkor a kapott fát véletlen építésű bináris keresőfának nevezzük.
- **Tétel:** Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfa várható magassága: $O(\log_2 n)$.

[2] 241-244

Szúrjuk be rendre az 1, 2, 3, 4, 5, …, n elemeket!

- 1
- 2
- 3
- 4
- 5
- ...
- n elemeket



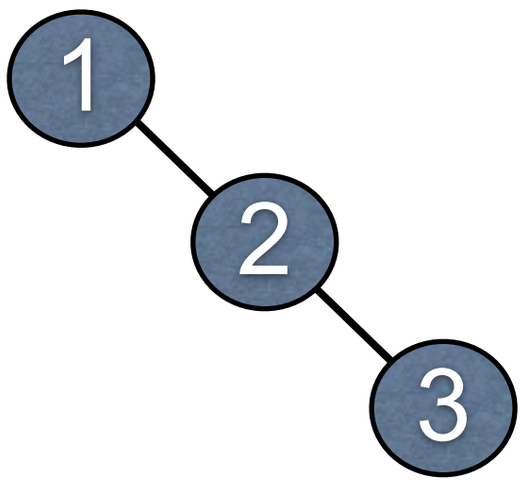
Legrosszabb eset

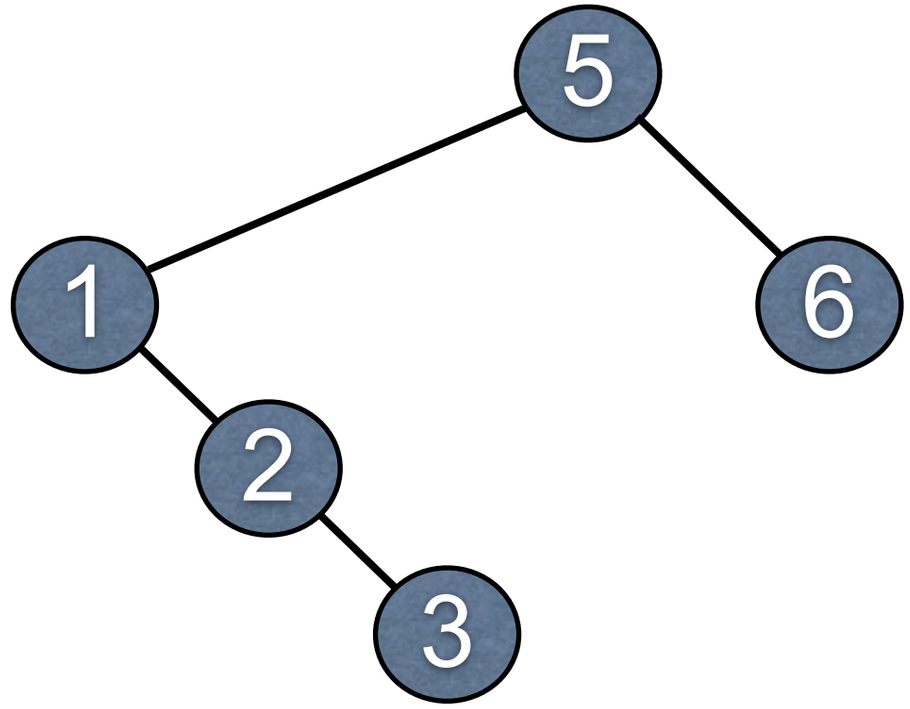
h=n

Hogy lehetne javítani?

Mivel az elemek egyesével jönnek

Minden beszúrás (és törlés) után rögtön “javítani” kellene





Példa