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)*

Adatszerkezetek

Absztrakt adattípus (ADT)

Absztrakt adatszerkezet (ADS)

Reprezentáció (ábrázolás)

Implementáció (fejlesztés, programnyelvi megvalósítás)

Fizikai (memória) szint

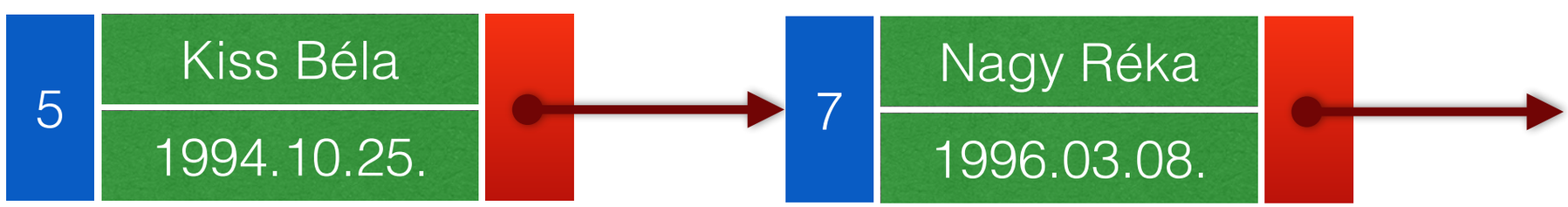
Adat tárolása

kulcsok

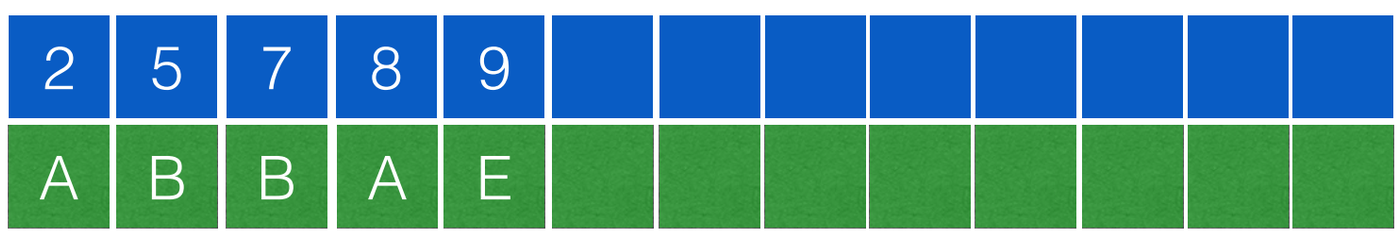
adattagok

szerkezeti elemek

Dinamikus



Statikus



műveletek

létrehozás

adat felvétele

adat módosítása

adat keresése

minden adat törlése (üresít)

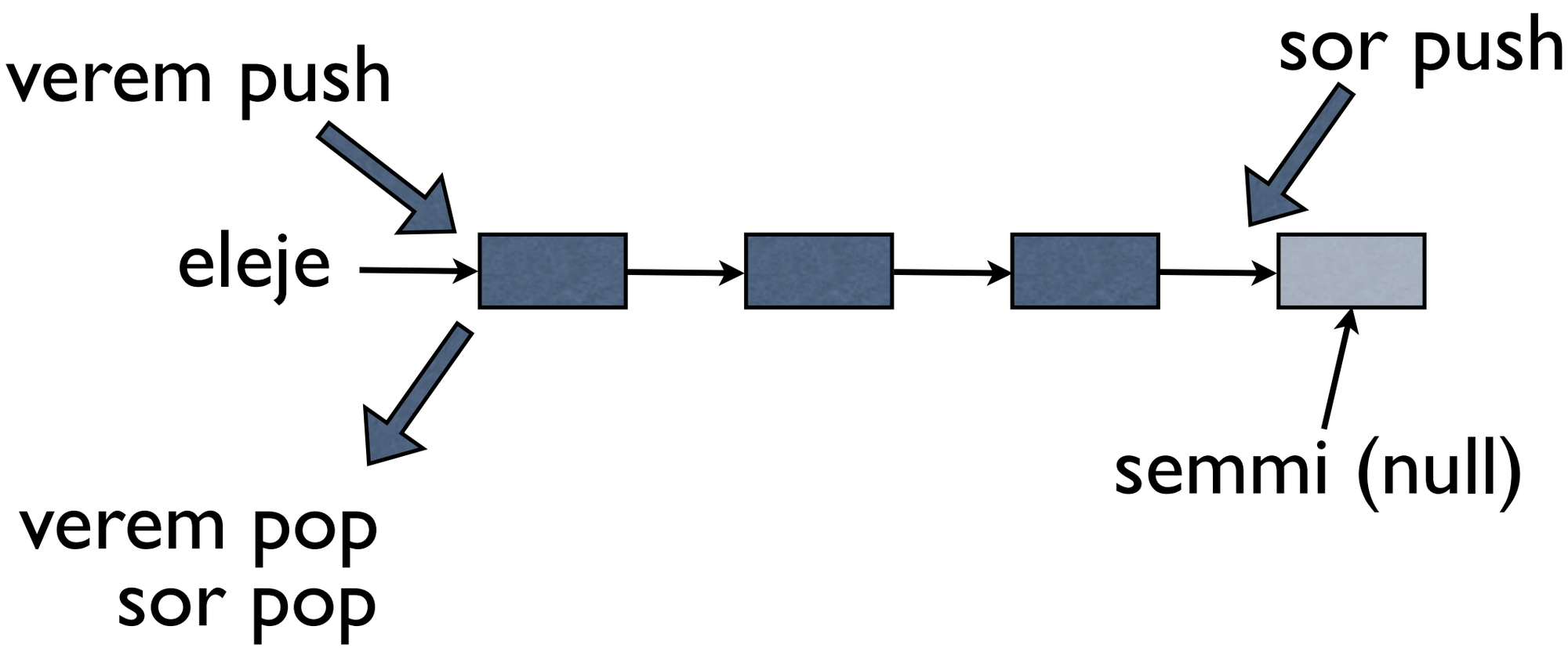
elemszám visszaadása

adat törlése

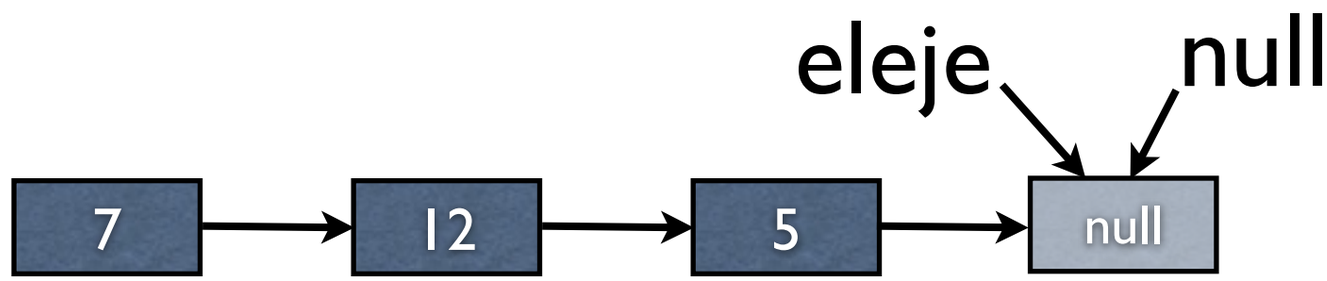
adatszerkezet felszámolása (megszüntet)

Verem, sor megvalósítása

alapműveletek: push, pop



Verem (Stack) példa



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

pop() pop() pop()

```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());
}
```

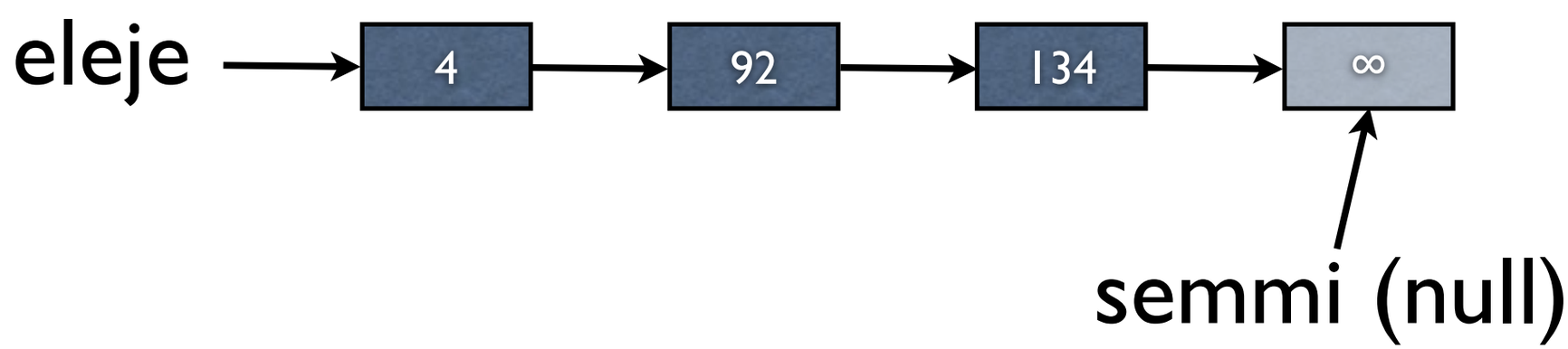
```java
public class sor {
    int elemszam ;
    class elem {
        int ertek;
        elem kovetkezo ;
    }
    elem semmi, eleje ;
    sor() {
        elem q=new elem() ;
        semmi=q ;
        eleje=q ;
        elemszam=0 ;
    }
    void betesz(int x) {
        elem q=new elem() ;
       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

Láncolt lista

Halmaz műveletek

Rendezett lista

Keresés, beszúrás, törlés: O(n)



Ugrólisták

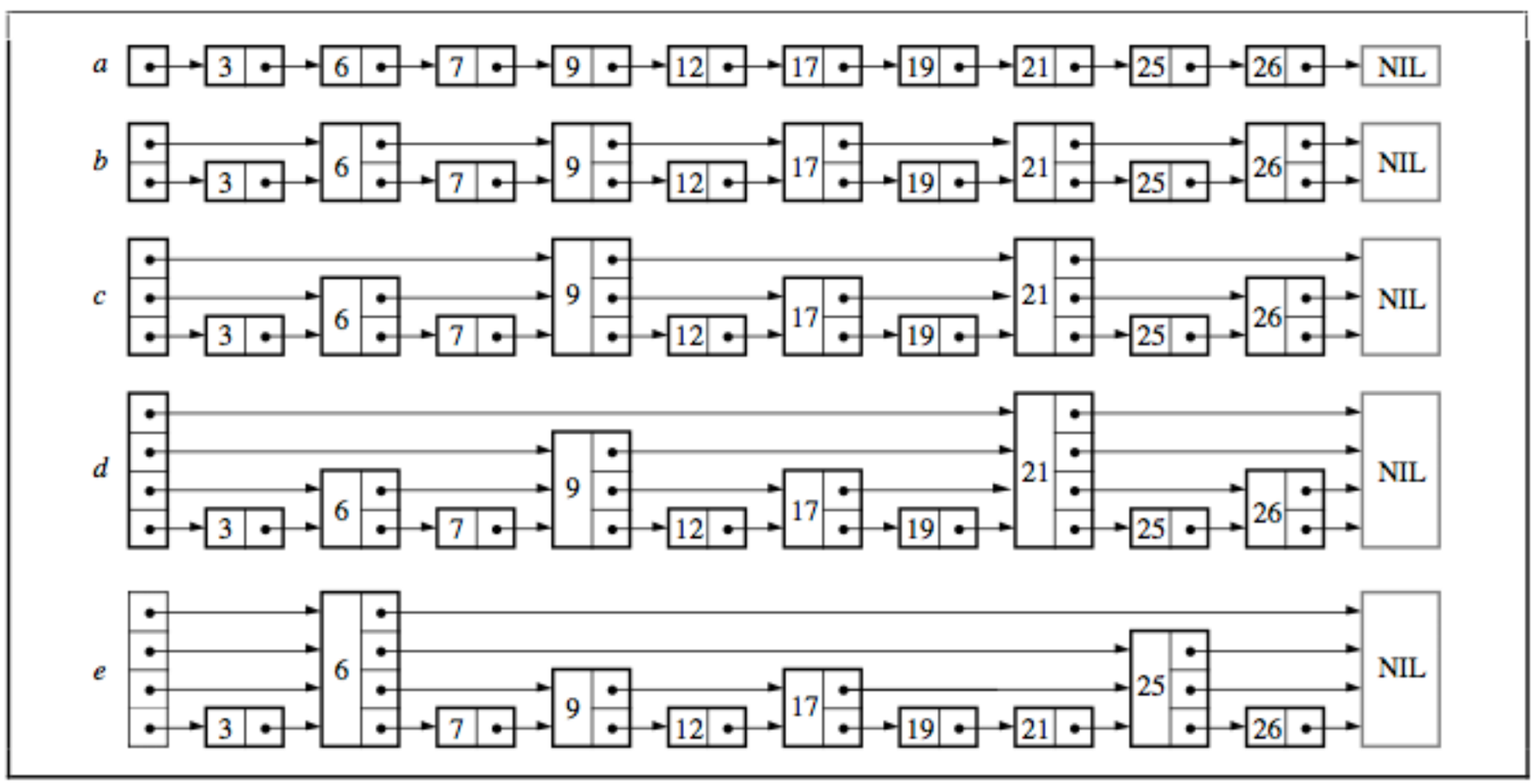


FIGURE 1 - Linked lists with additional pointers

Ugrólisták

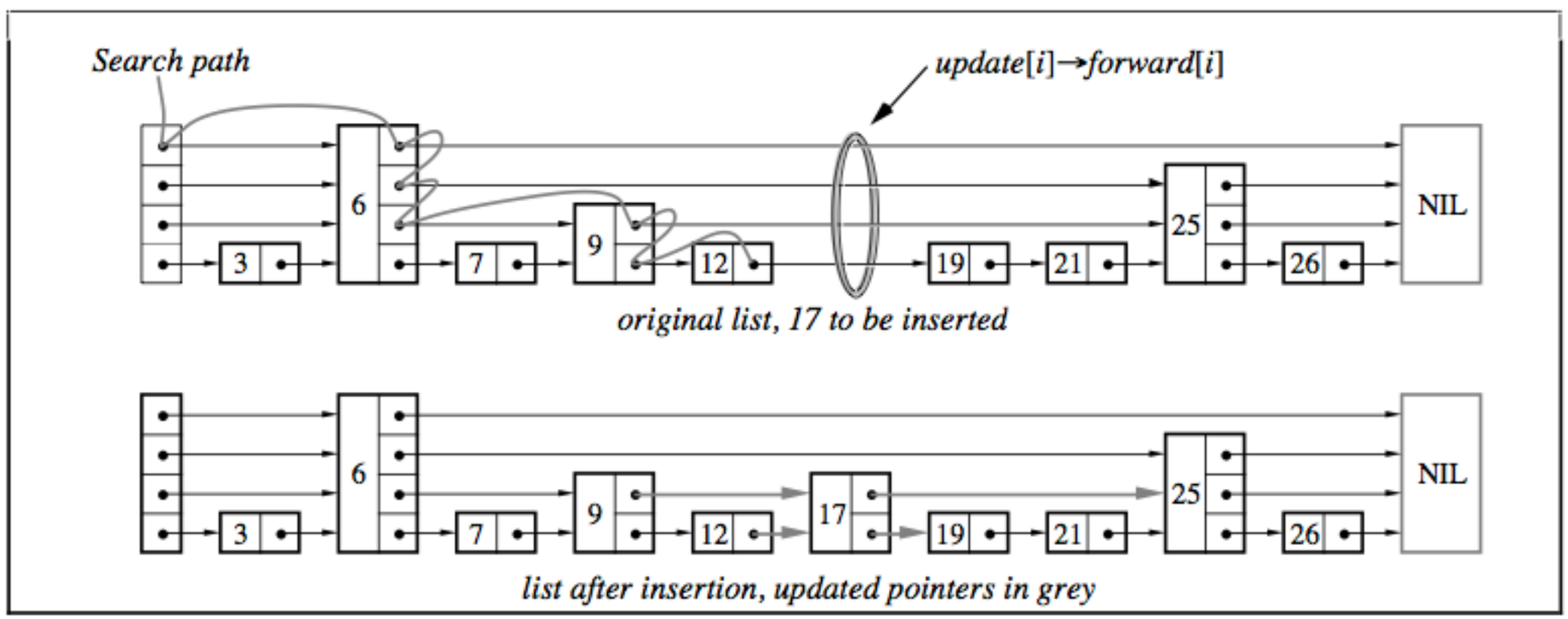


FIGURE 3 - Pictorial description of steps involved in performing an insertion

Ugrólisták

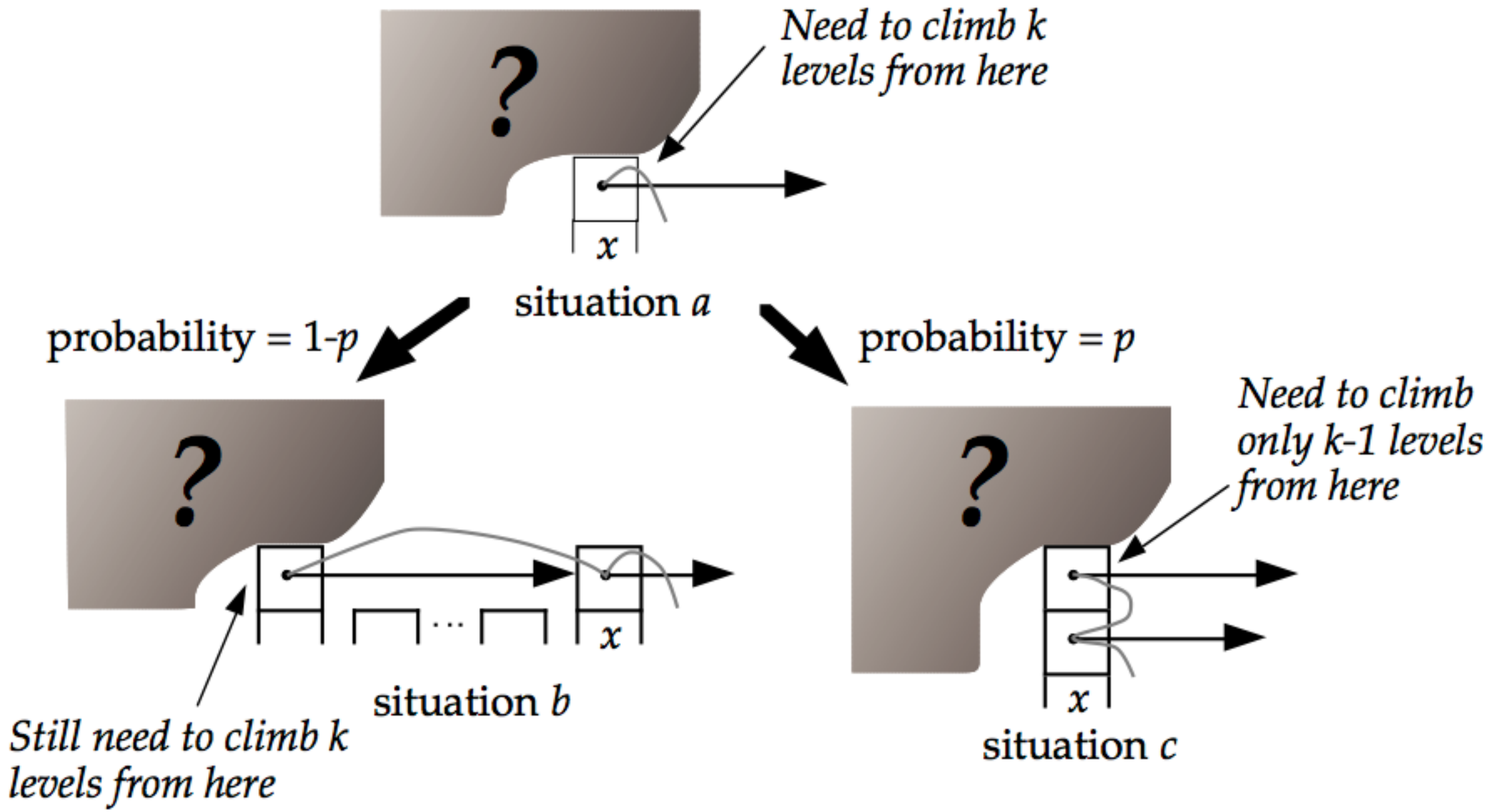
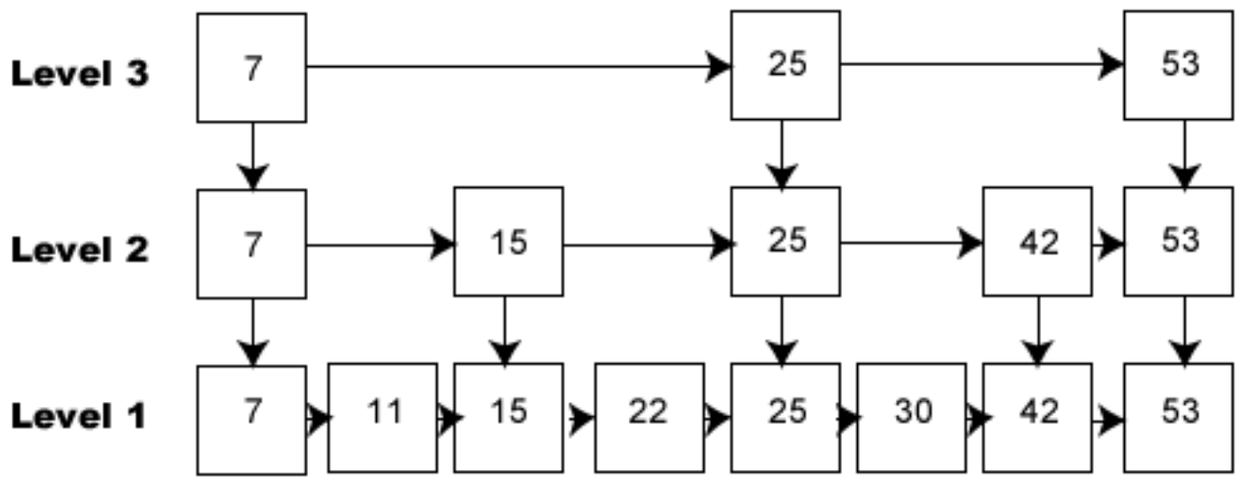
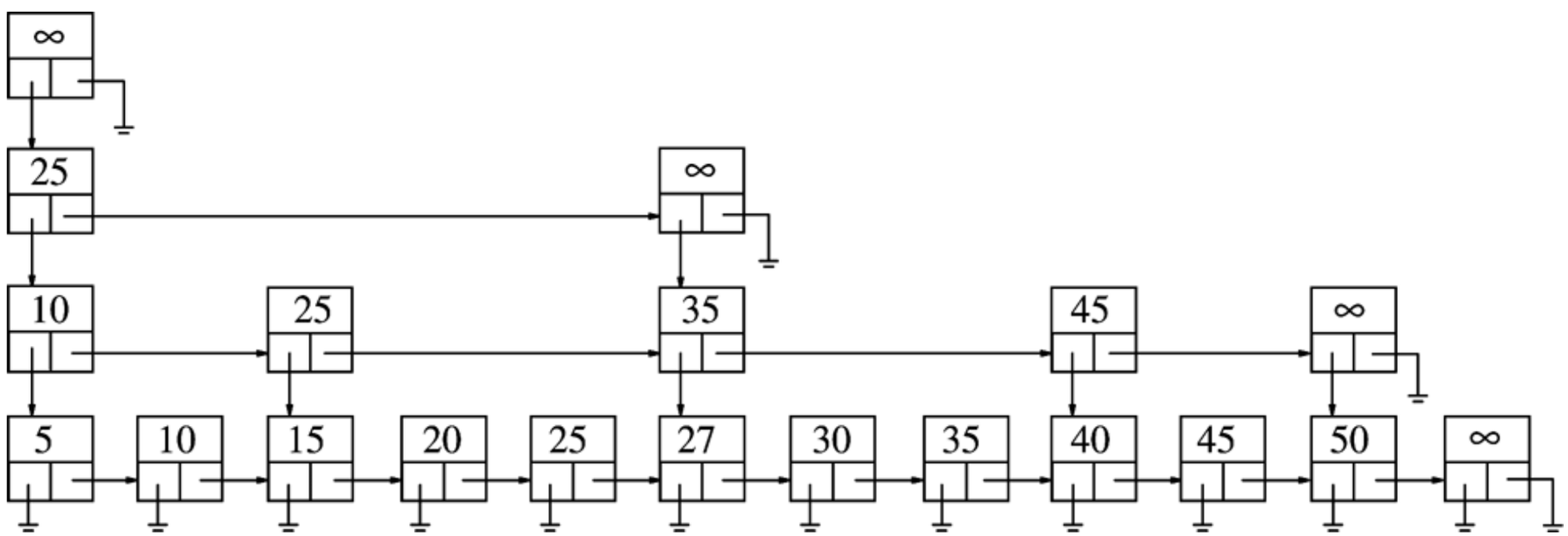


FIGURE 6 - Possible situations in backwards traversal of the search path

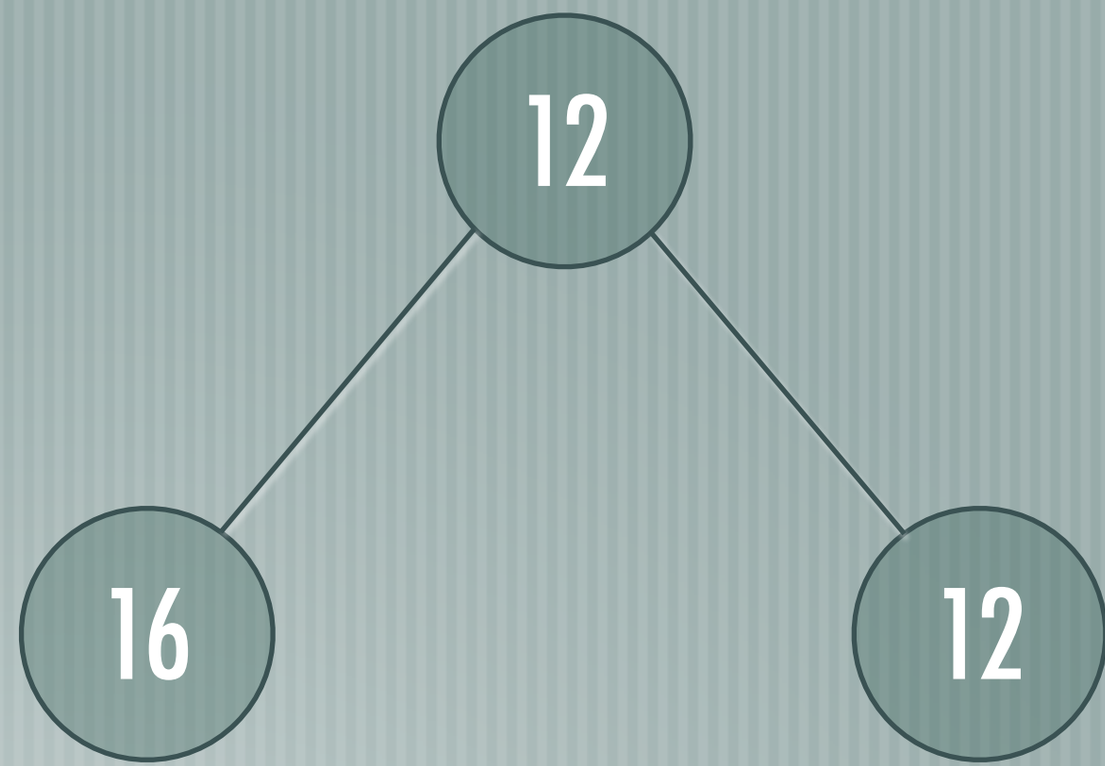

Level 3
7
25
53
Level 2
7
15
25
42
53
Level 1
7
11
15
22
25
30
42
53


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

Kupac adatszerkezet

Kupac adatszerkezet tulajdonsága:

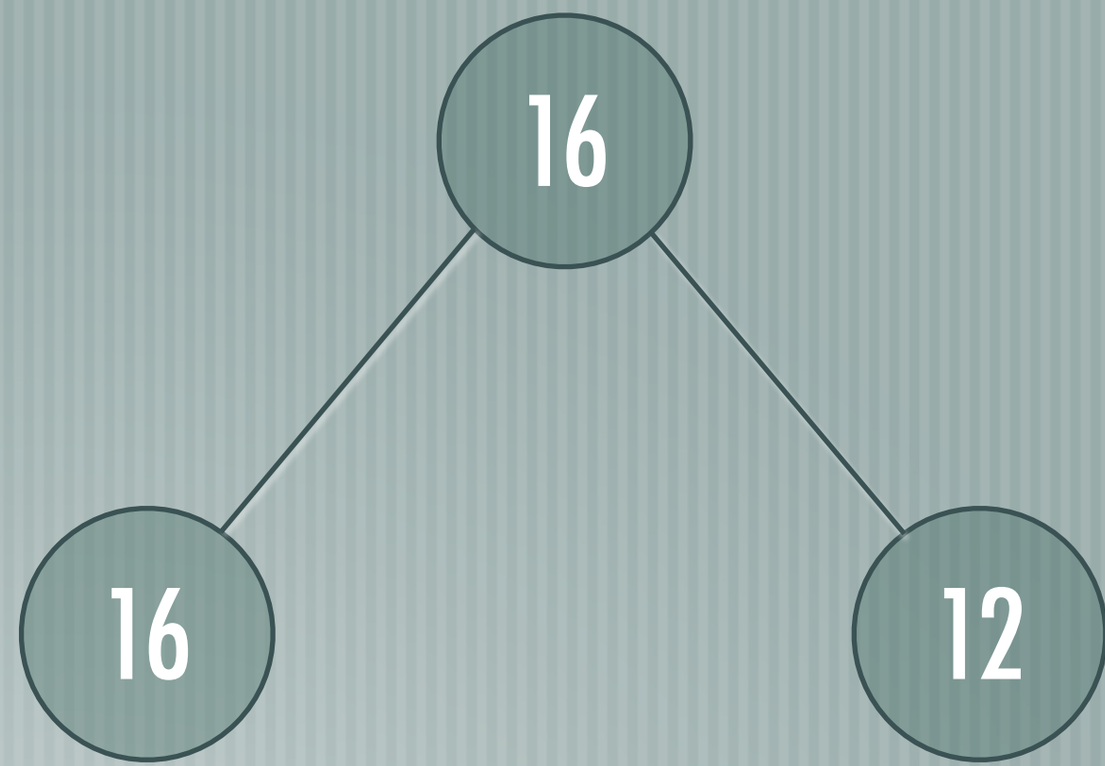
minimum kupac:



$$\forall p \in Q_{pac} : p_{qlcs} \leq \min(\langle p_{bal} \rangle_{qlcs}, \langle p_{jobb} \rangle_{qlcs})$$

Kupac adatszerkezet tulajdonsága:

maximum kupac:



$$\forall p \in Q_{pac} : p_{qlcs} \geq \max(\langle p_{bal} \rangle_{qlcs}, \langle p_{jobb} \rangle_{qlcs})$$

5, 8, 2, 7, 1

maximum kupac

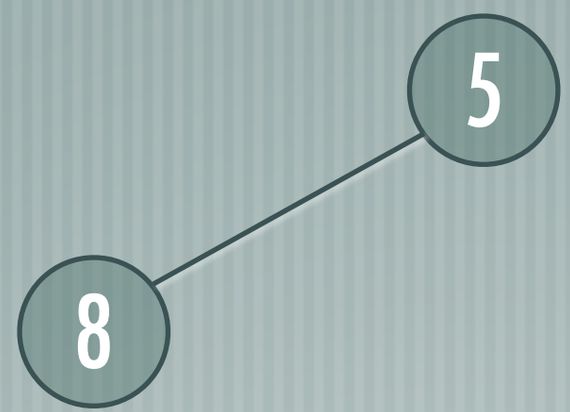
void s.sorba(T x)

5

5, 8, 2, 7, 1

maximum kupac

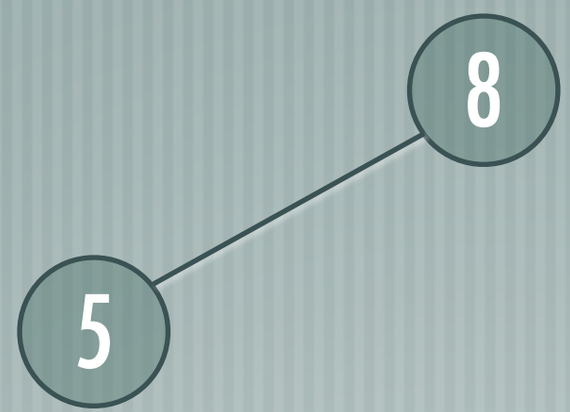
void s.sorba(T x)



5, 8, 2, 7, 1

maximum kupac

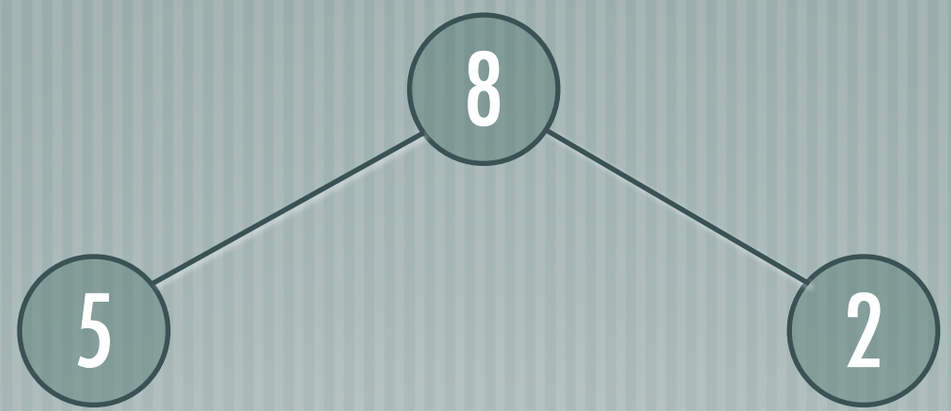
void s.sorba(T x)



5, 8, 2, 7, 1

maximum kupac

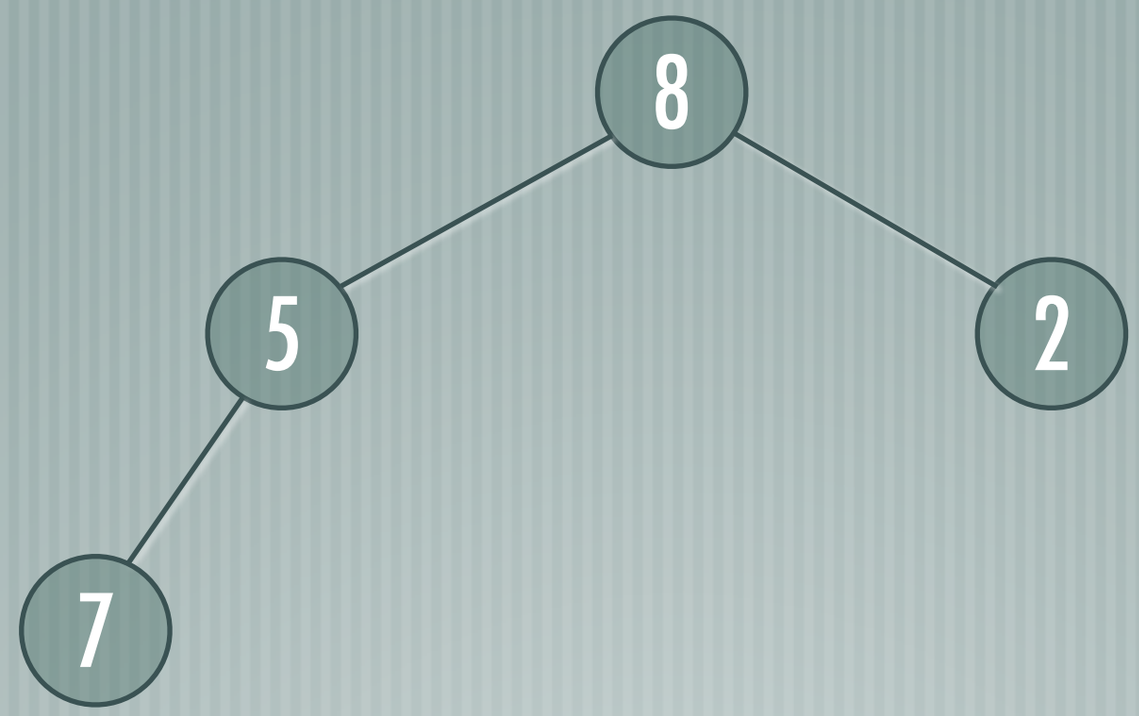
void s.sorba(T x)



5, 8, 2, 7, 1

maximum kupac

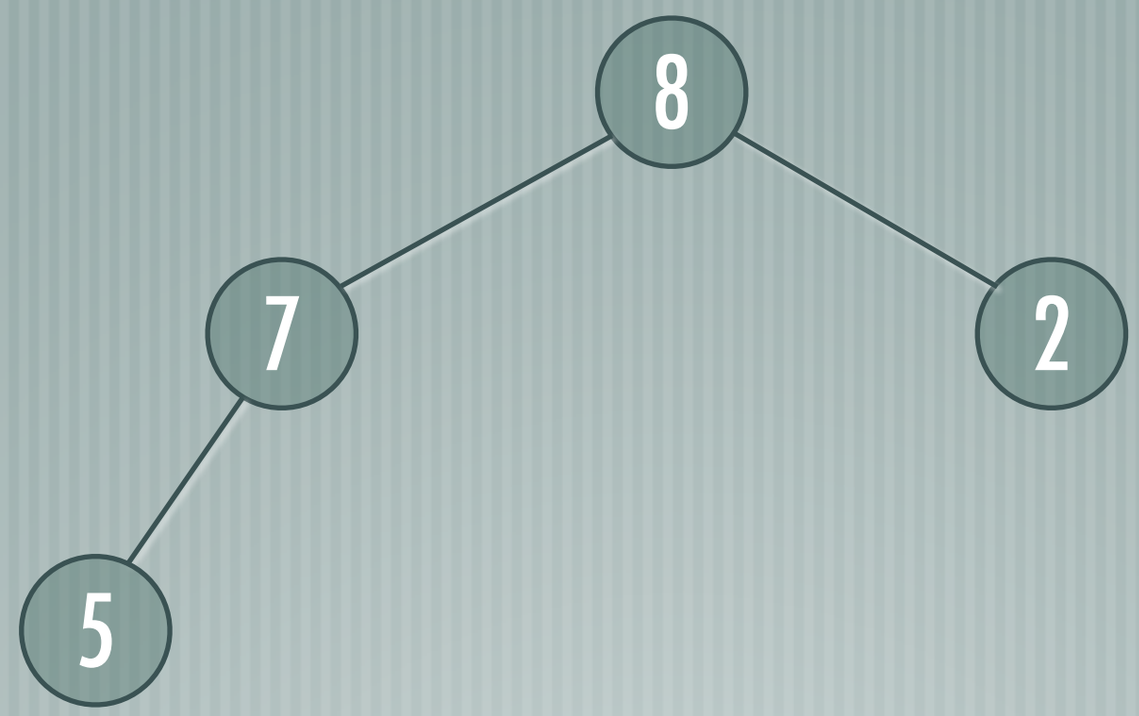
void s.sorba(T x)



5, 8, 2, 7, 1

maximum kupac

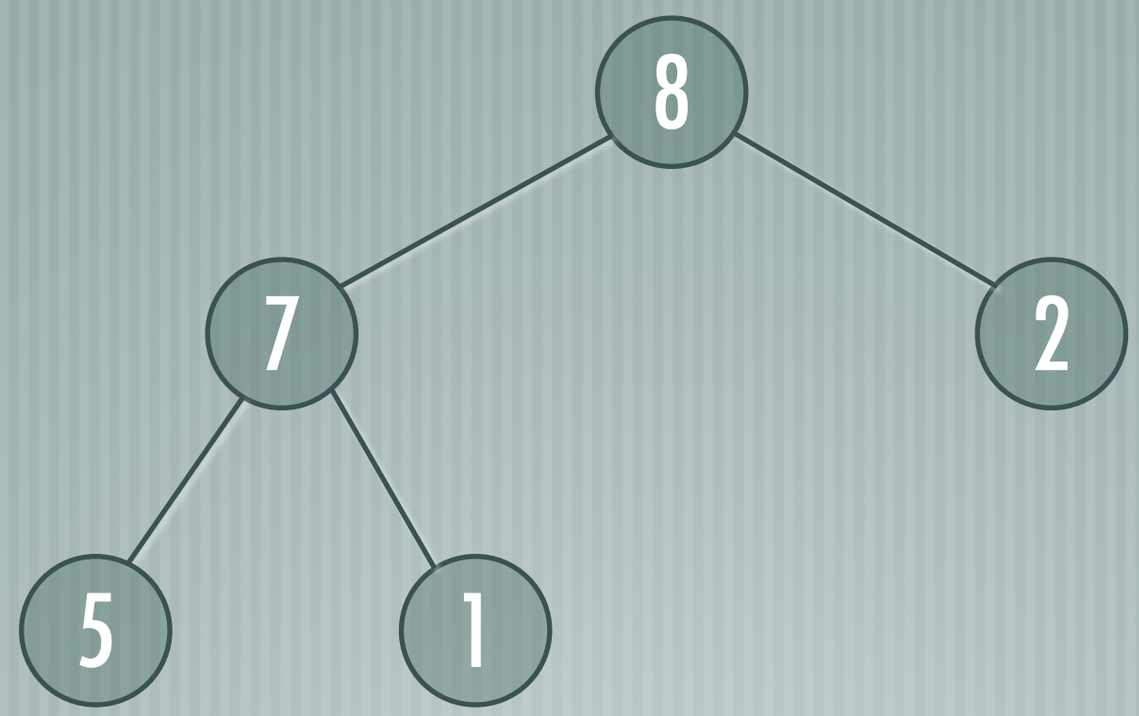
void s.sorba(T x)



5, 8, 2, 7, 1

maximum kupac

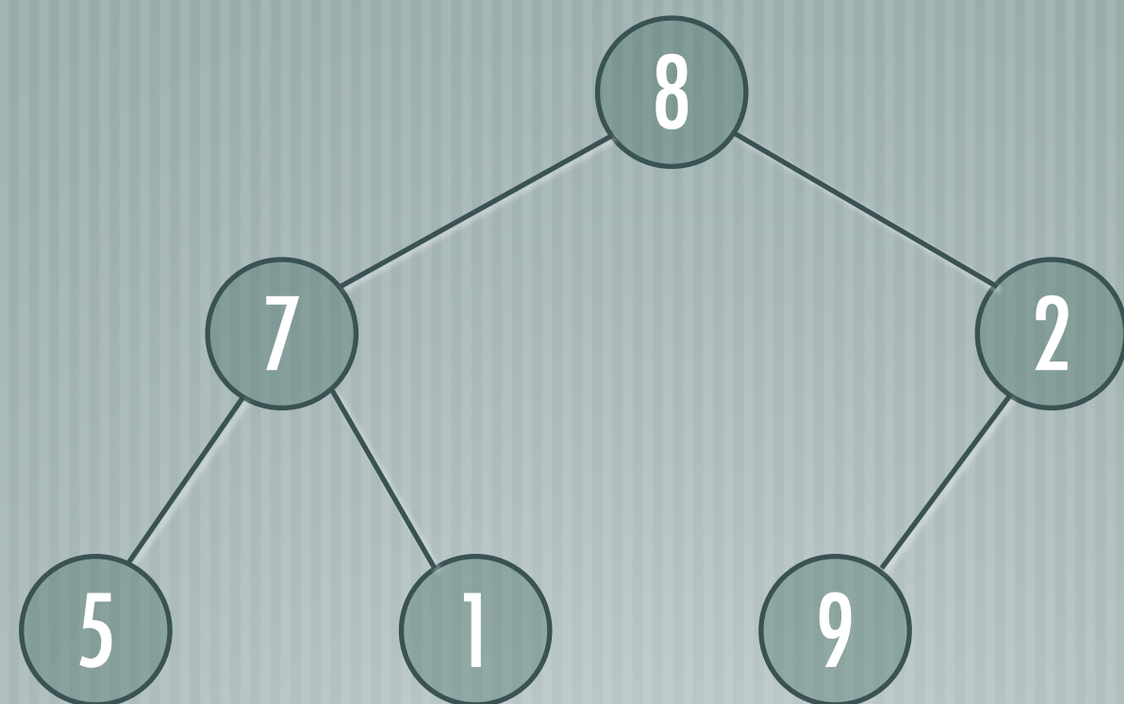
void s.sorba(T x)



5, 8, 2, 7, 1, 9

maximum kupac

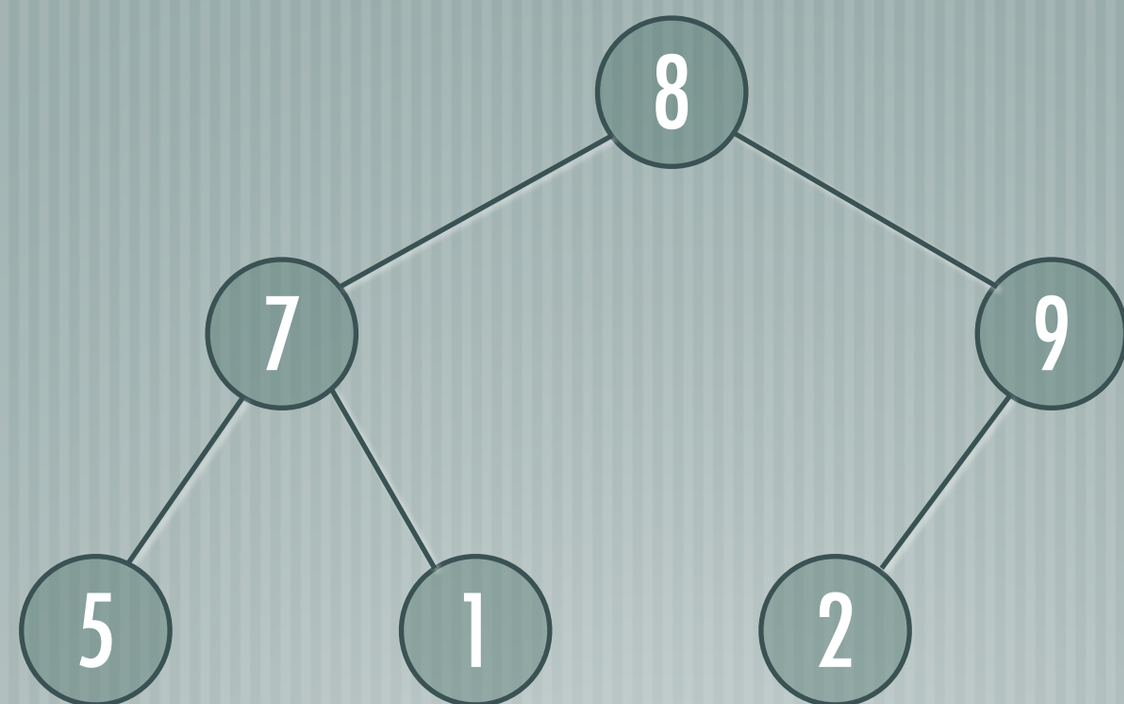
void s.sorba(T x)



5, 8, 2, 7, 1, 9

maximum kupac

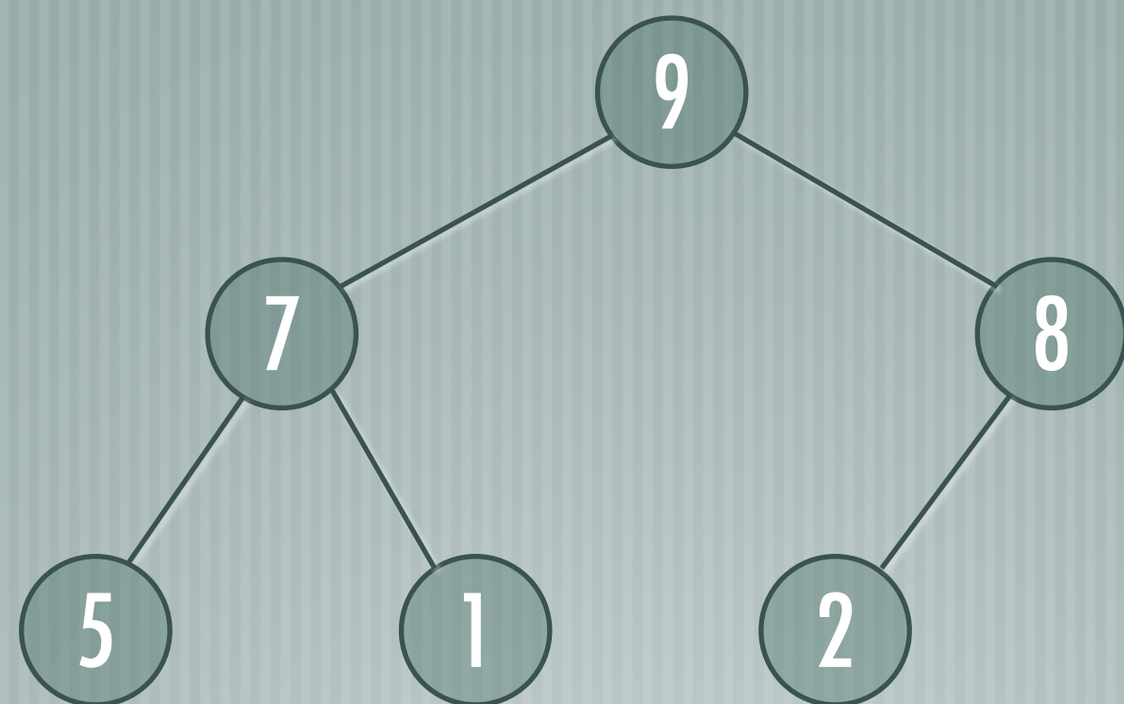
void s.sorba(T x)



5, 8, 2, 7, 1, 9

maximum kupac

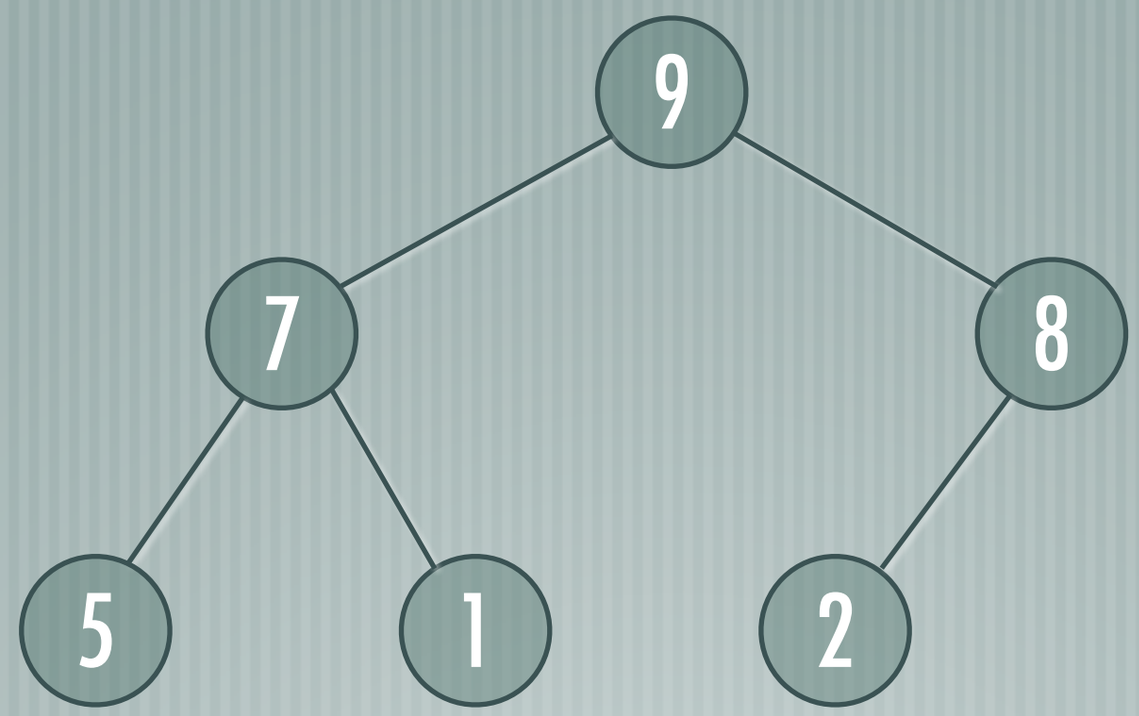
void s.sorba(T x)



sorba művelet időigénye O(log(n))

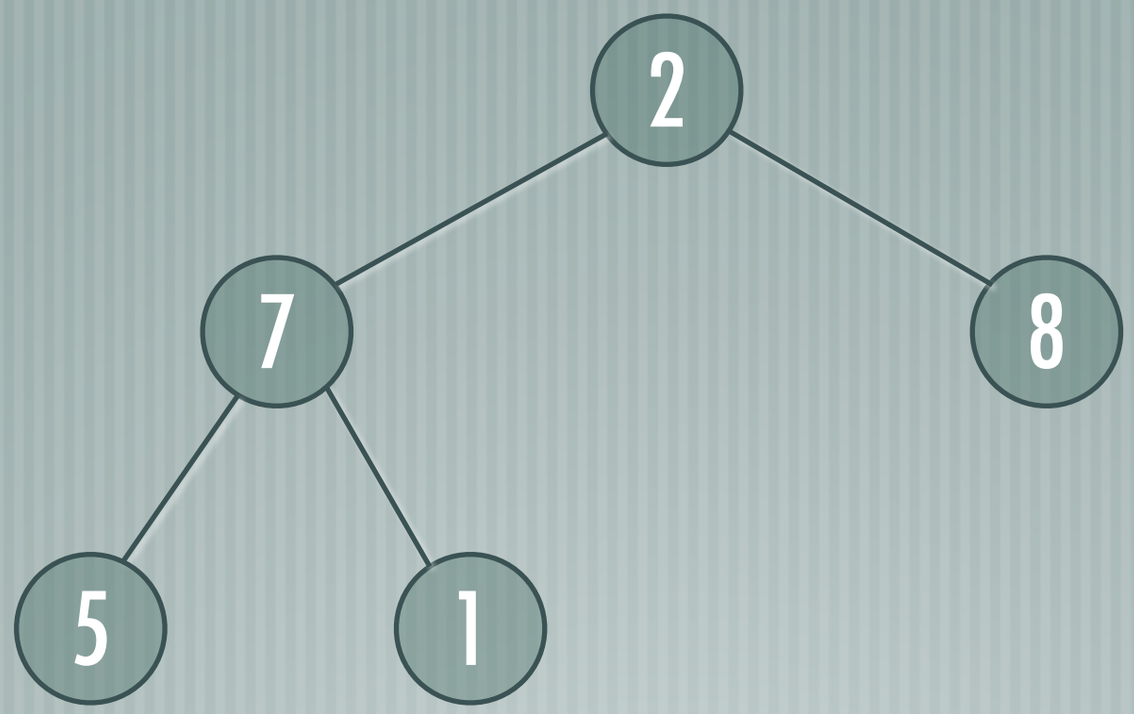
maximum kupac

T s.sorbol()



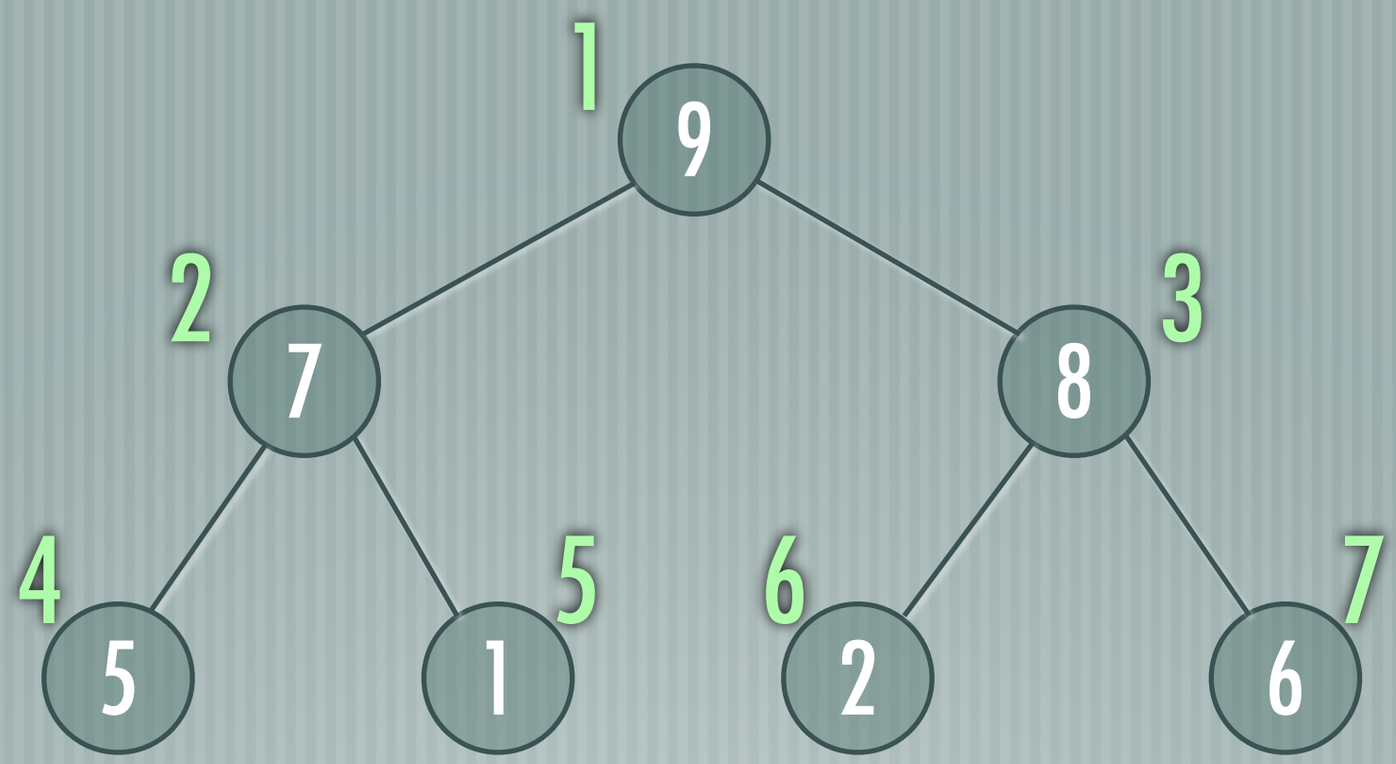
maximum kupac

T s.sorbol()



sorbol művelet időigénye O(log(n))

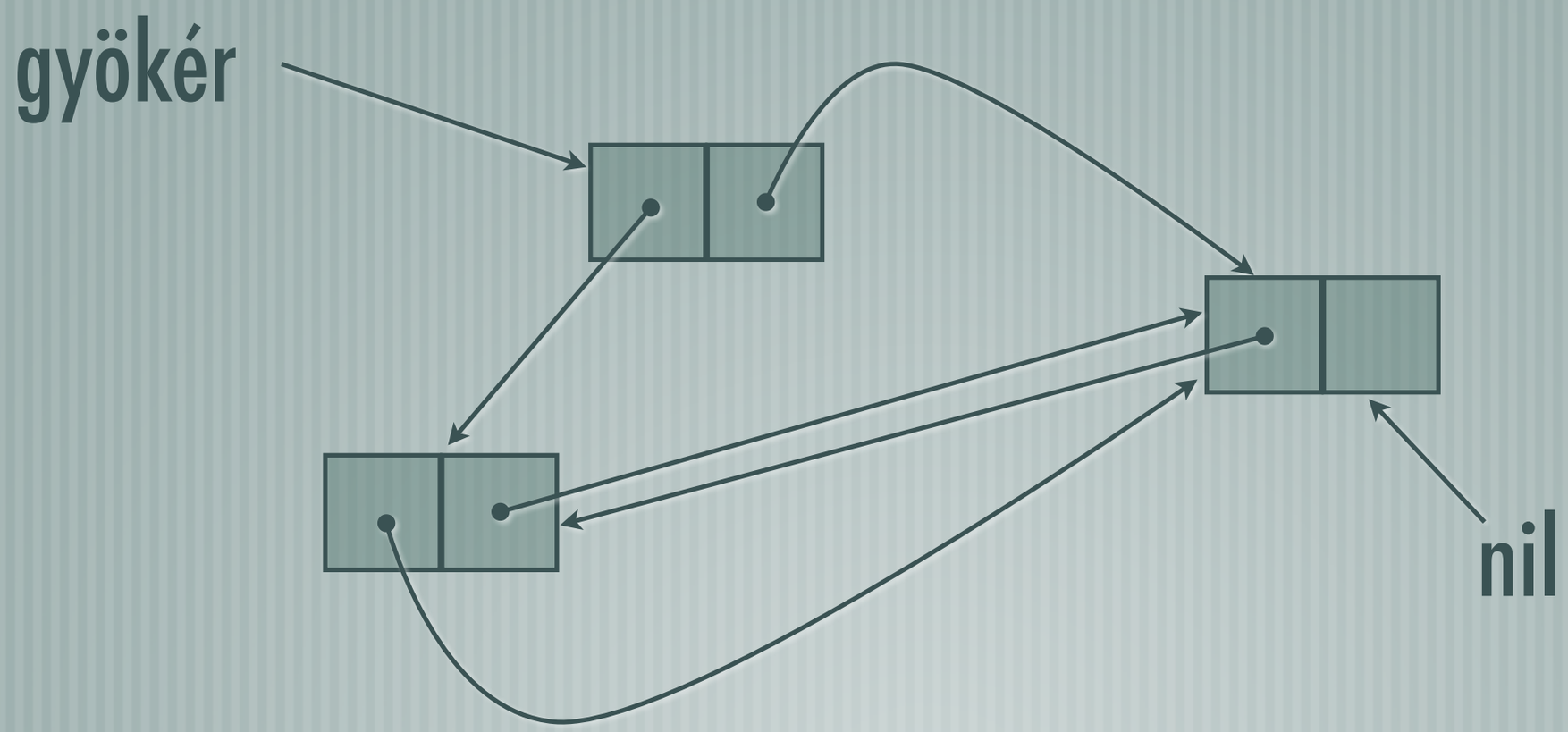
A kupac adatszerkezet már tanult megvalósítása



1	2	3	4	5	6	7
9	7	8	5	1	2	6

Nem dinamikus adatszerkezet

Bináris fa 2 pointeres megvalósítása



Halmazok egyesítése

Kruskal algoritmus

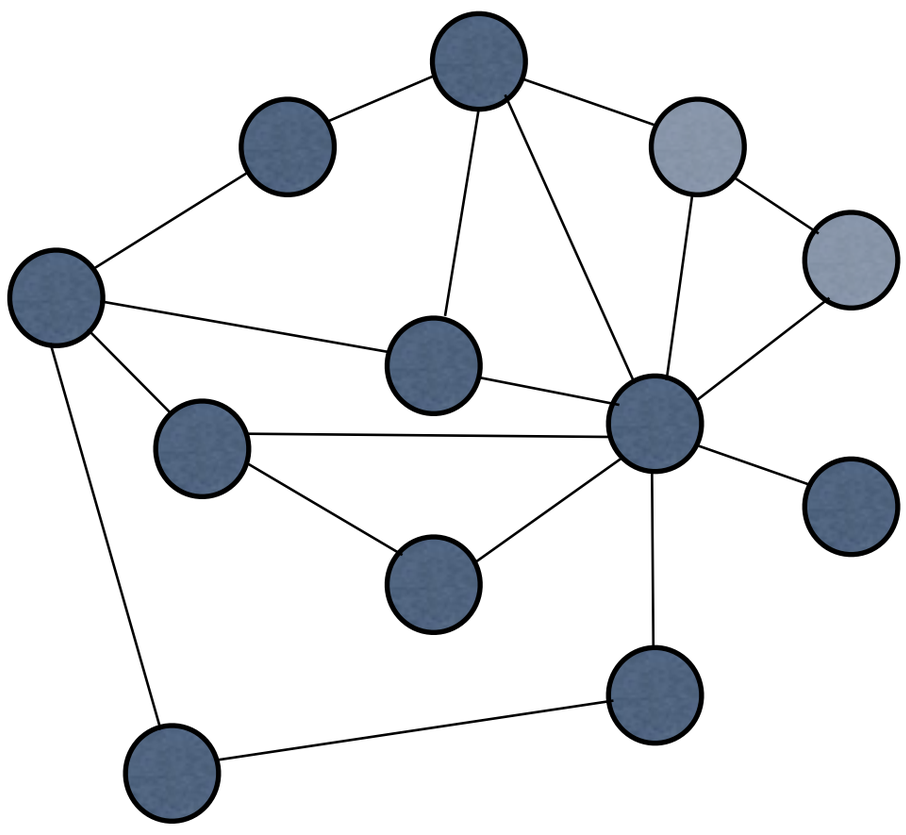
felelevenítés

```
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

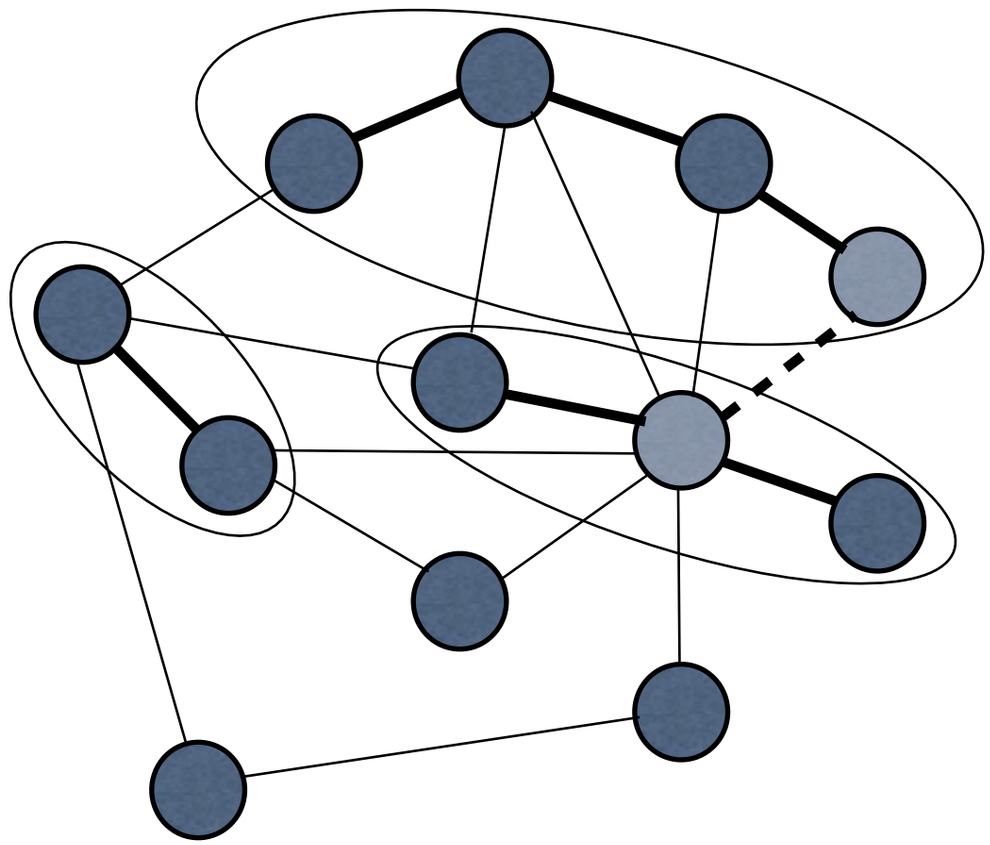
Kruskal algoritmus

felelevenítés



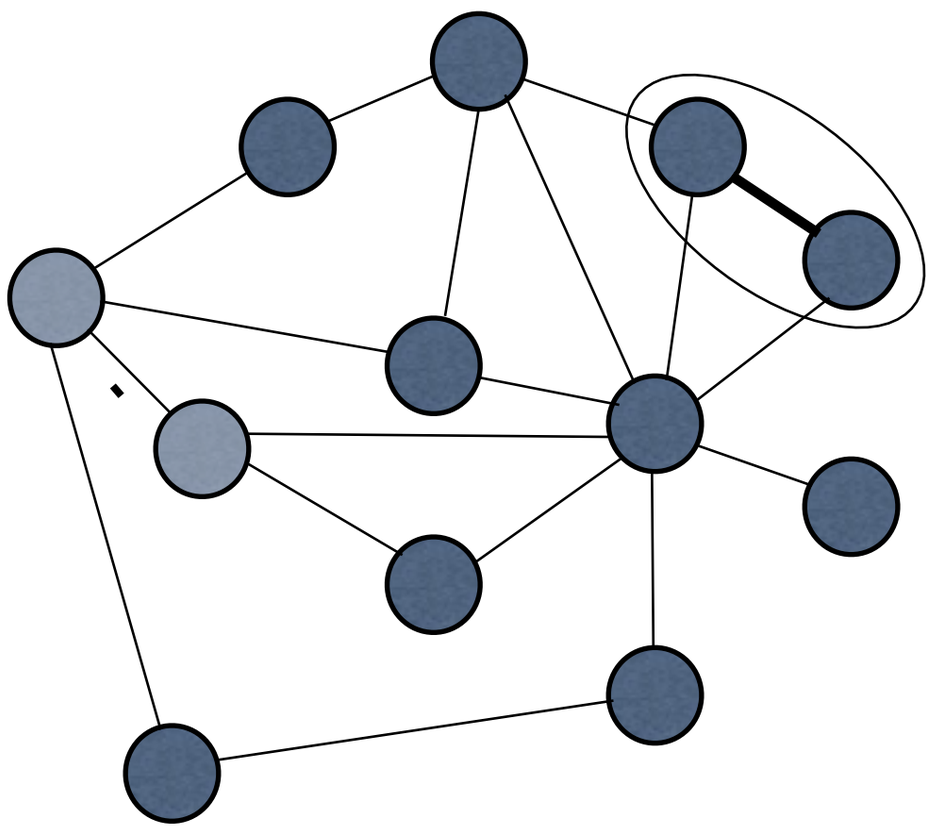
Honnan lehet tudni, hogy a következő él beválasztásával a feszítőerdő erdő (azaz körmentes) marad?

Példa



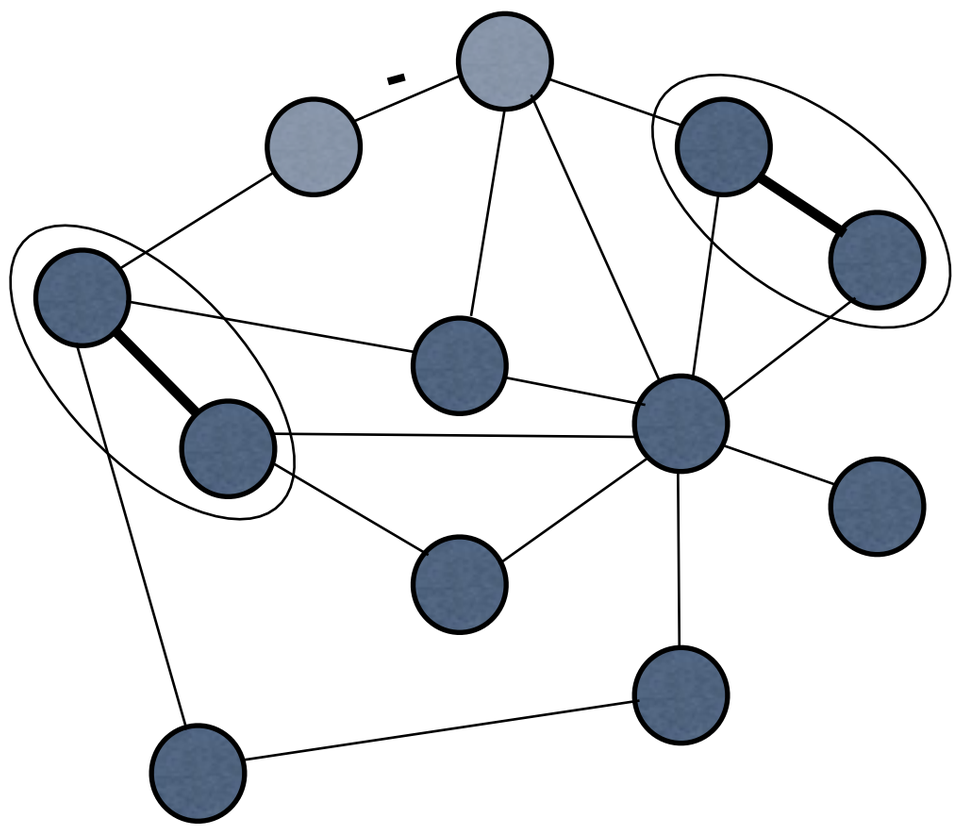
Kruskal algoritmus

felelevenítés



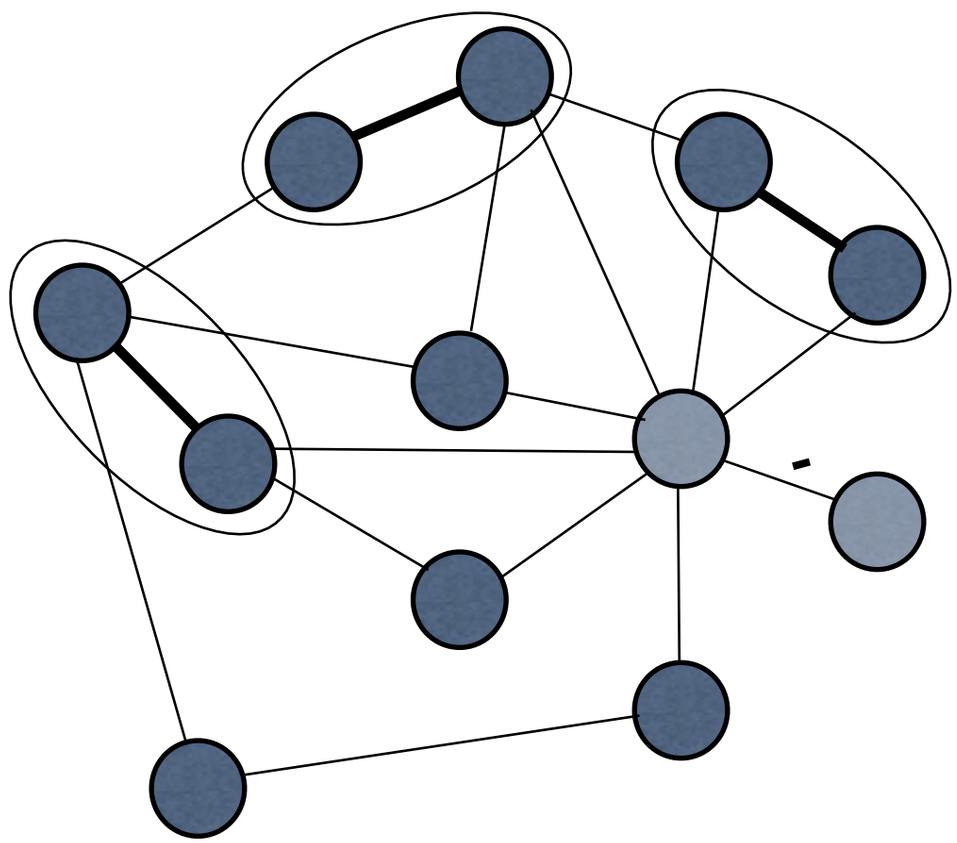
Kruskal algoritmus

felelevenítés



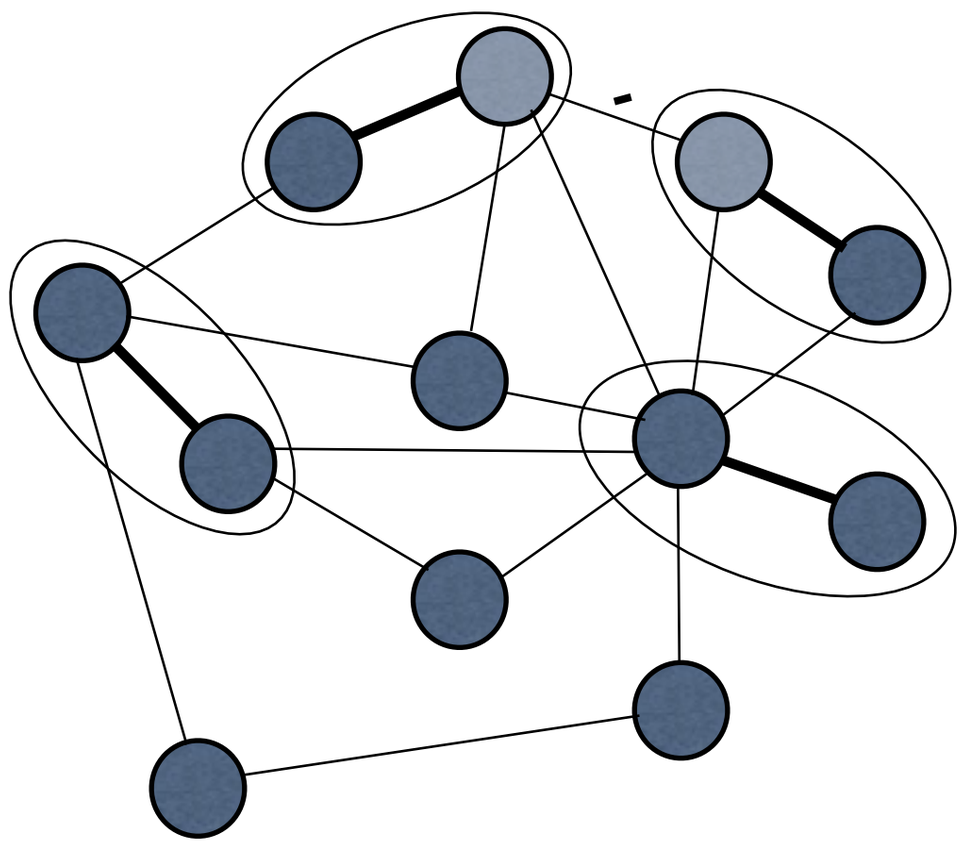
Kruskal algoritmus

felelevenítés



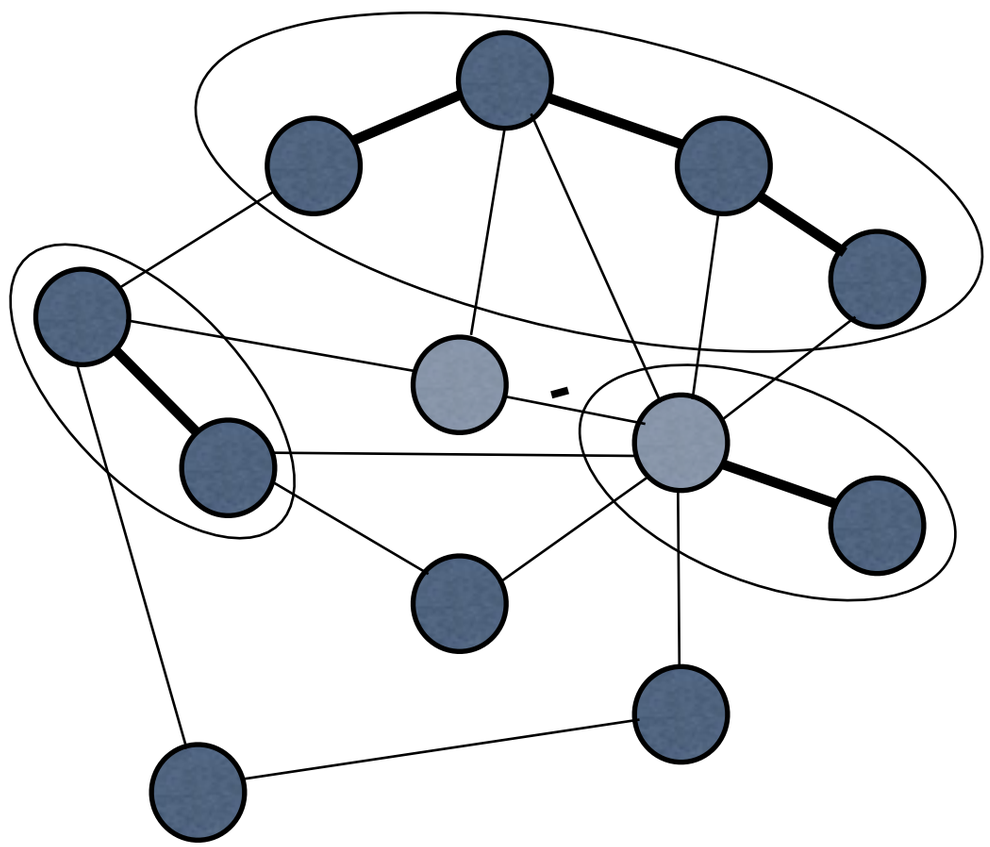
Kruskal algoritmus

felelevenítés



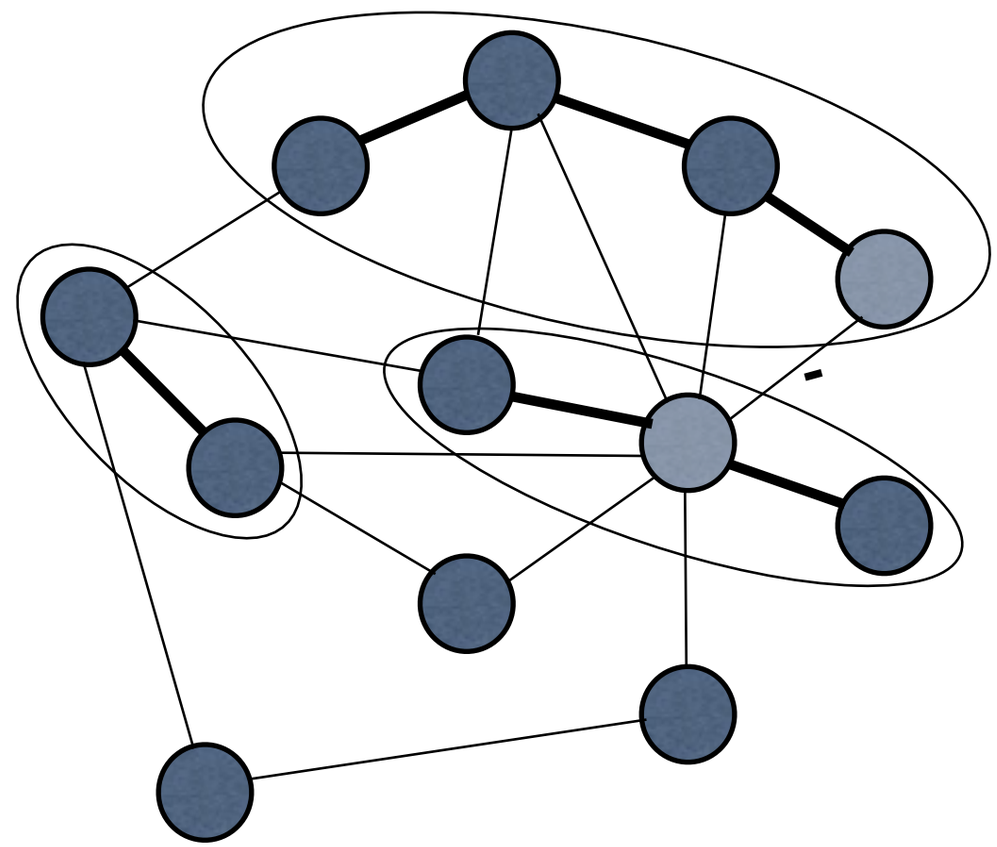
Kruskal algoritmus

felelevenítés



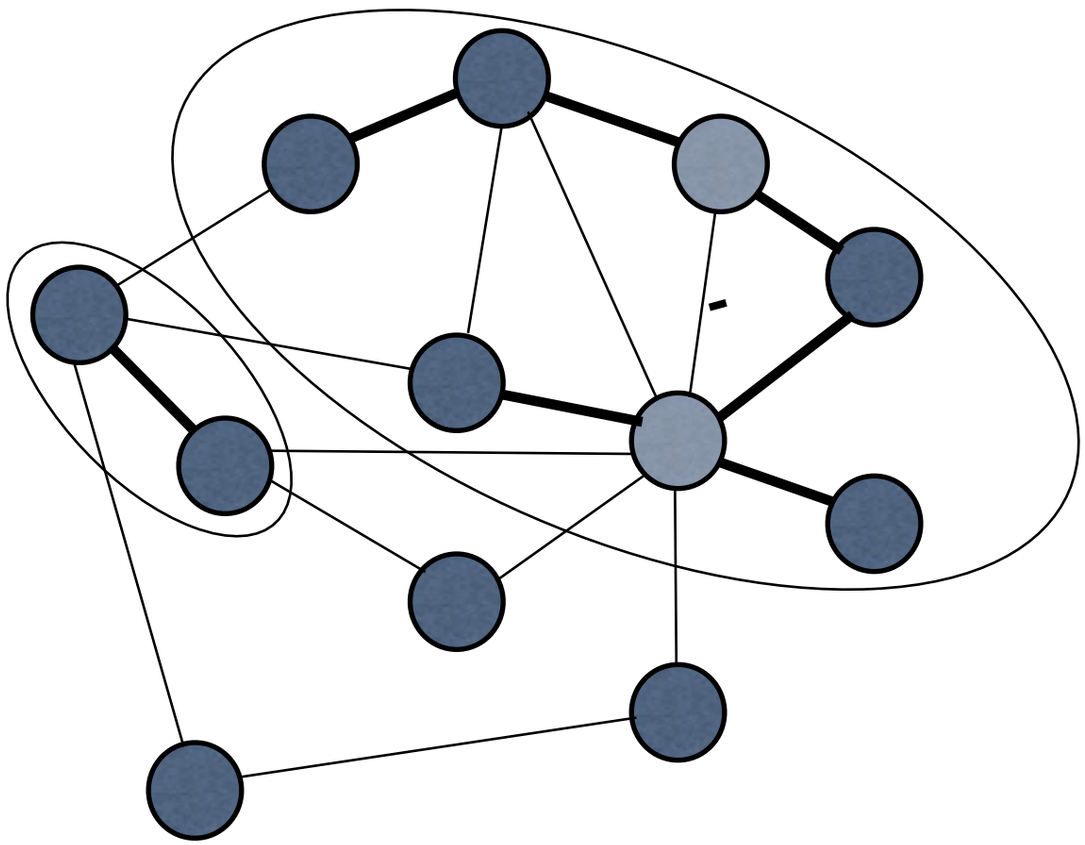
Kruskal algoritmus

felelevenítés



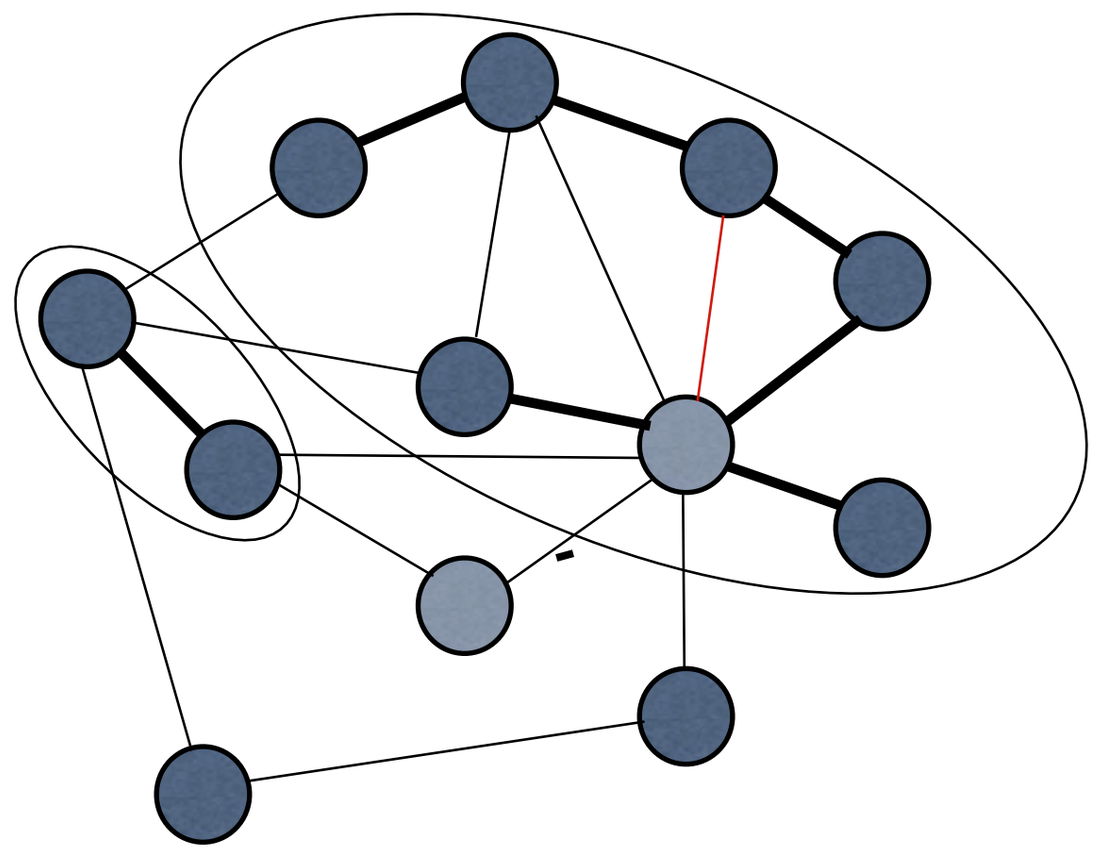
Kruskal algoritmus

felelevenítés



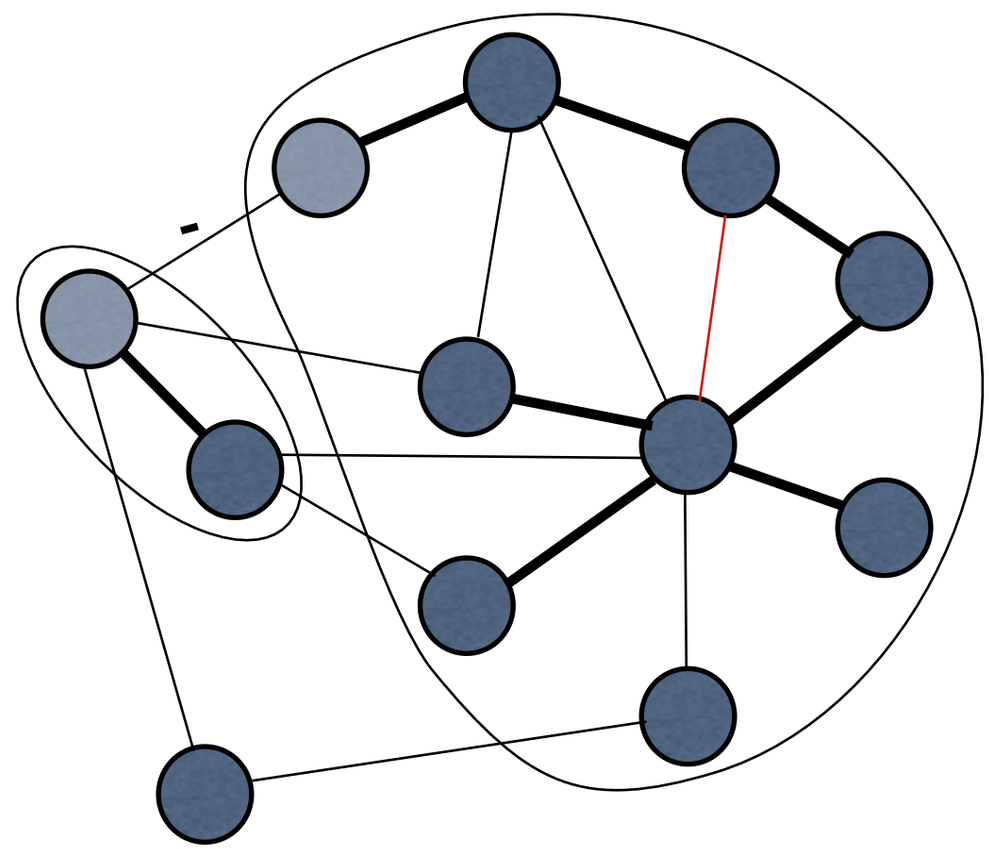
Kruskal algoritmus

felelevenítés



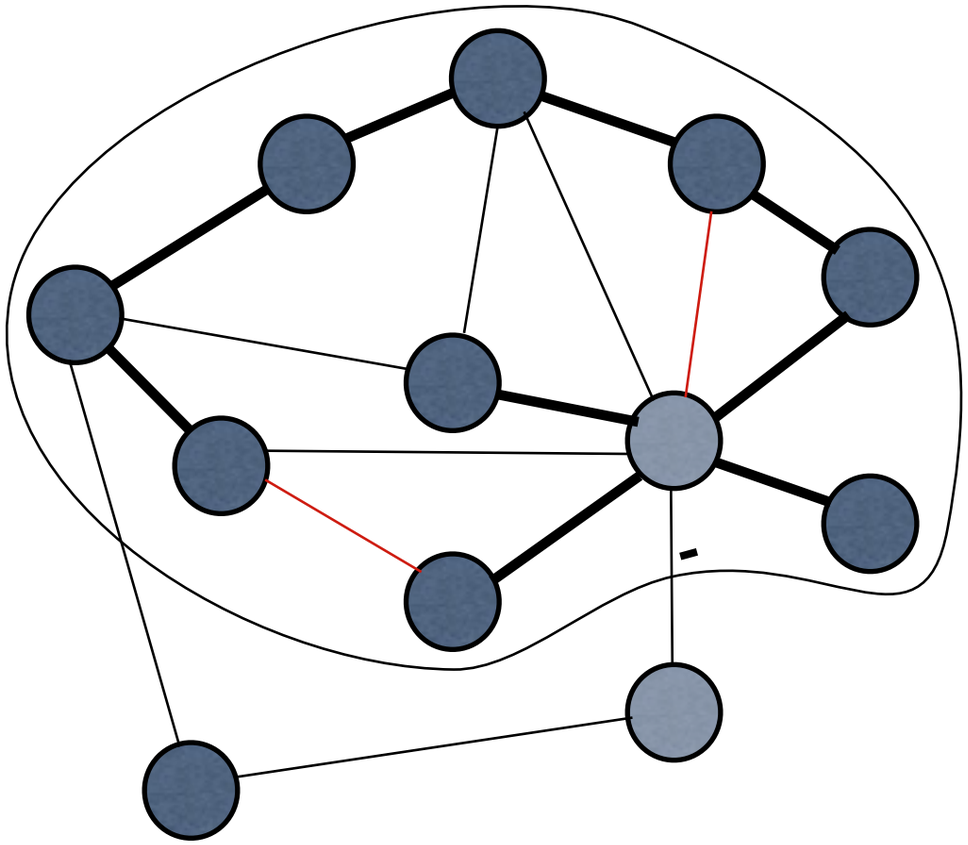
Kruskal algoritmus

felelevenítés



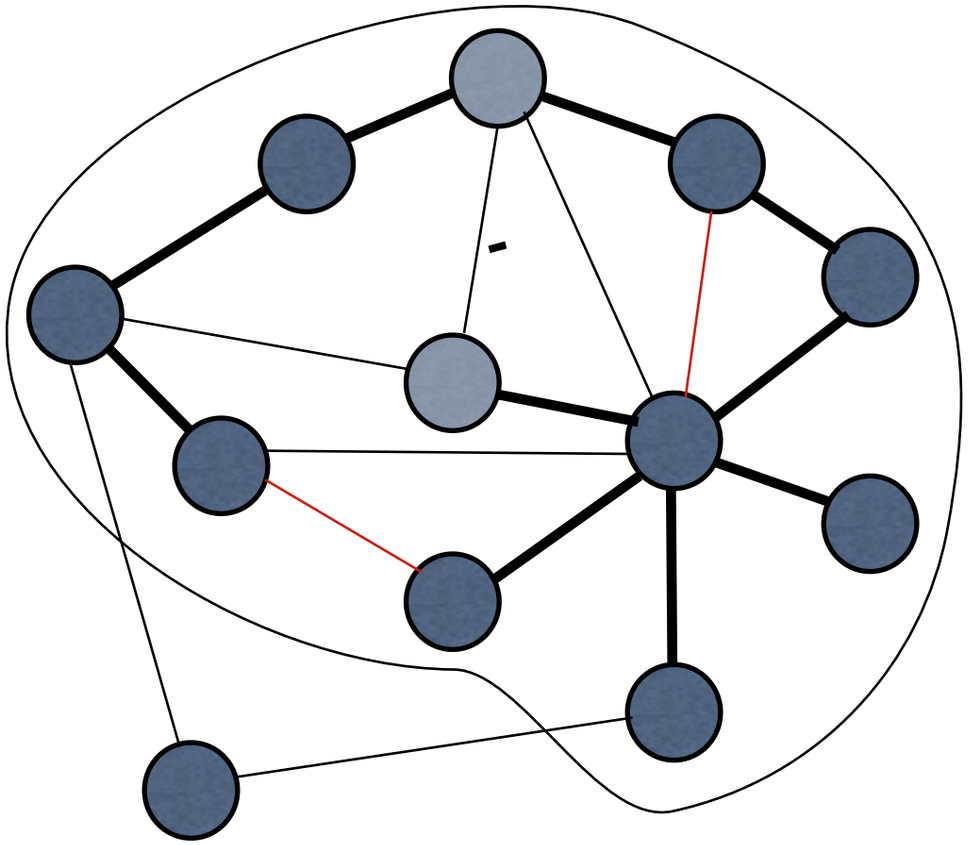
Kruskal algoritmus

felelevenítés



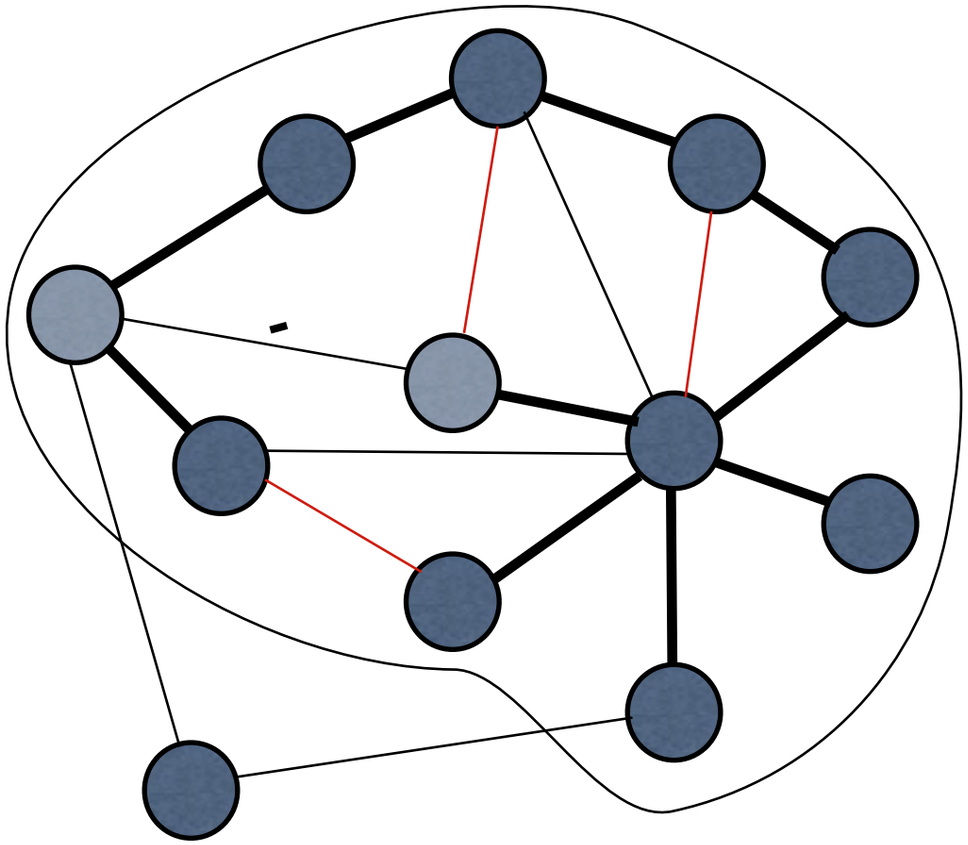
Kruskal algoritmus

felelevenítés



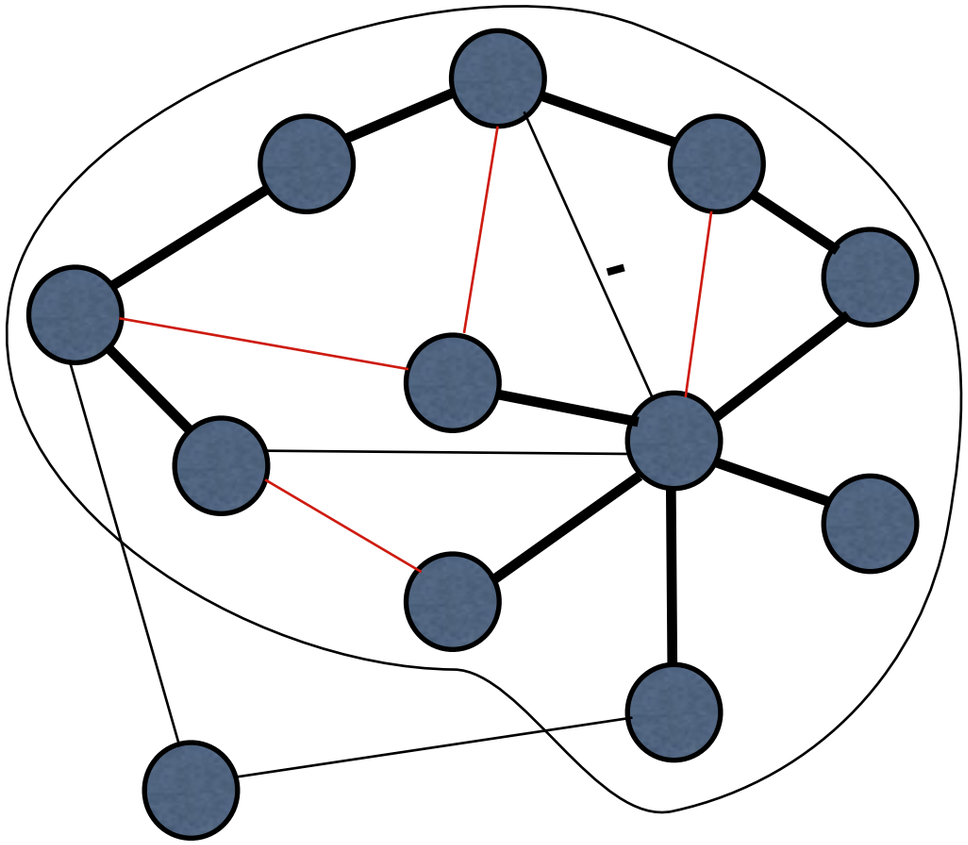
Kruskal algoritmus

felelevenítés



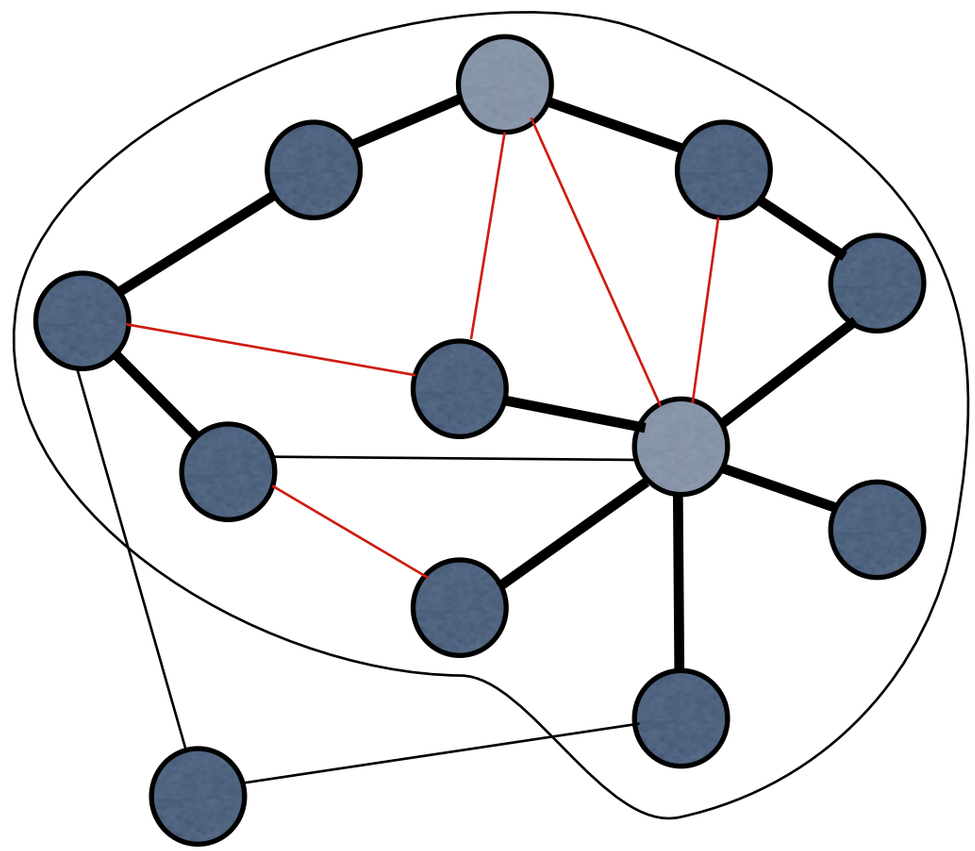
Kruskal algoritmus

felelevenítés



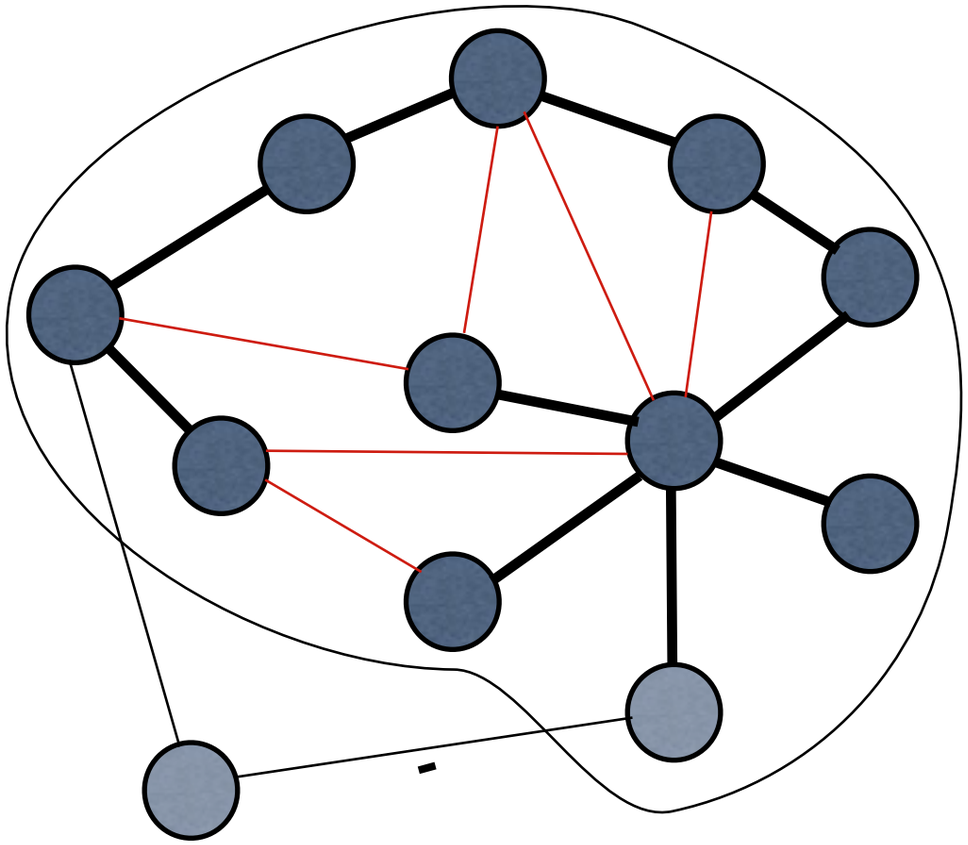
Kruskal algoritmus

felelevenítés



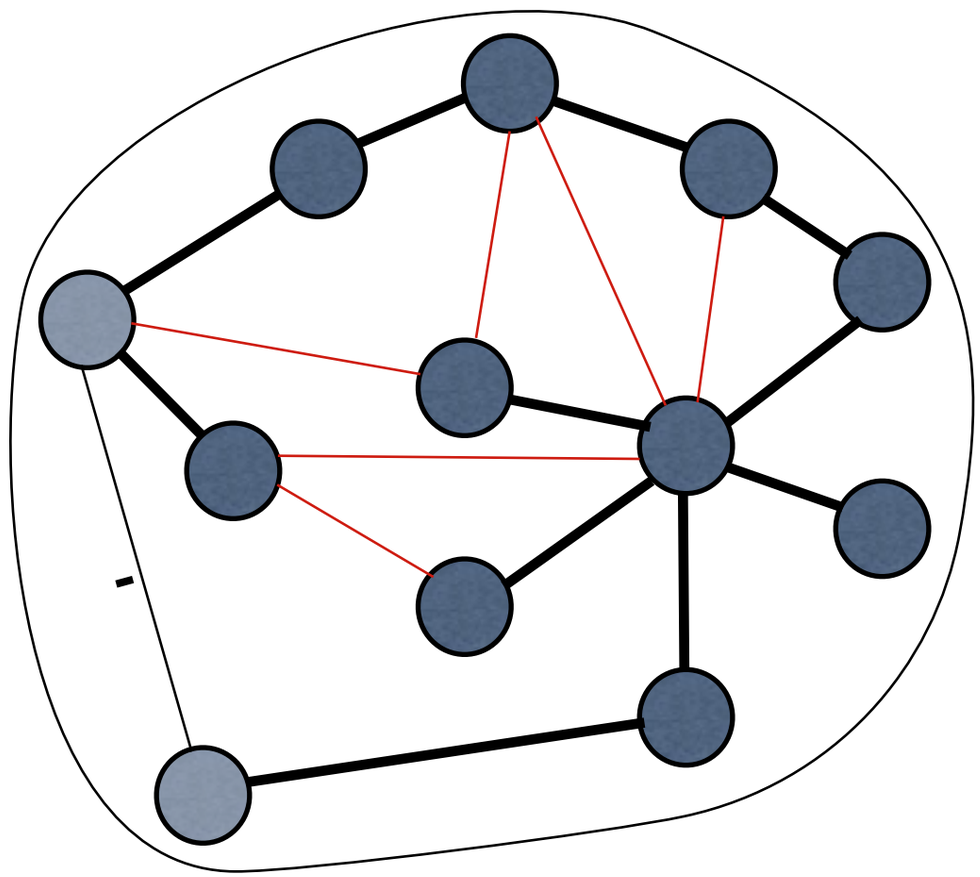
Kruskal algoritmus

felelevenítés



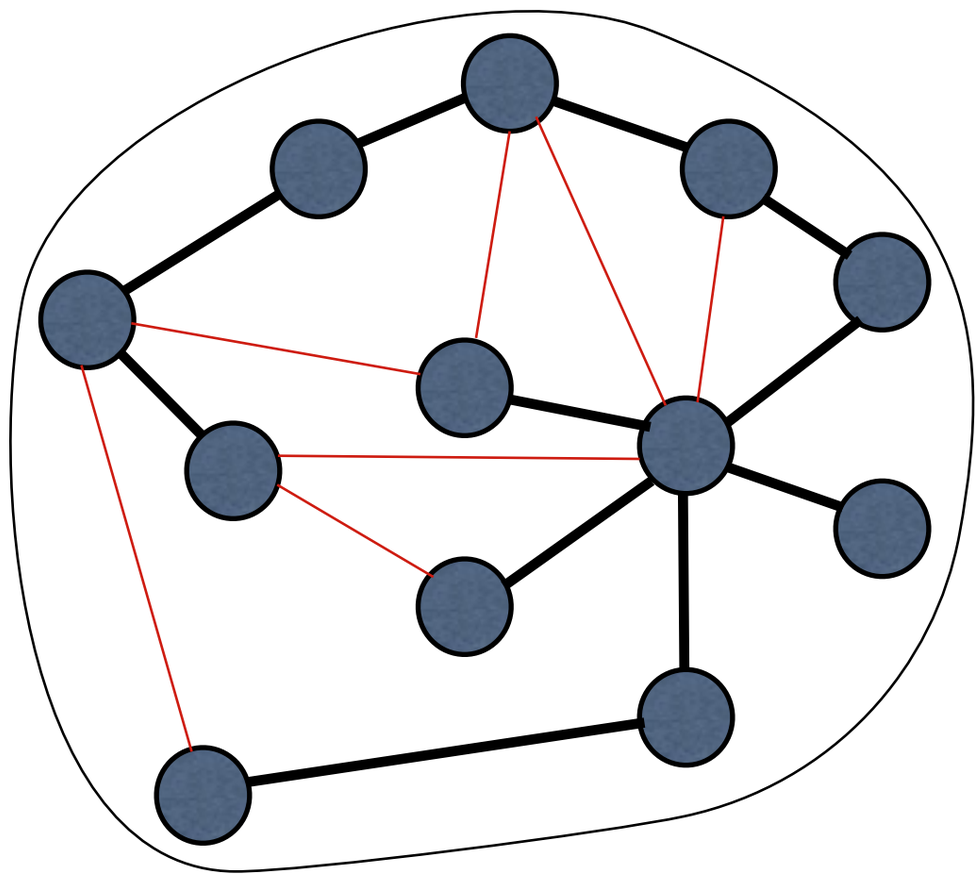
Kruskal algoritmus

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) ;
   }
}
```

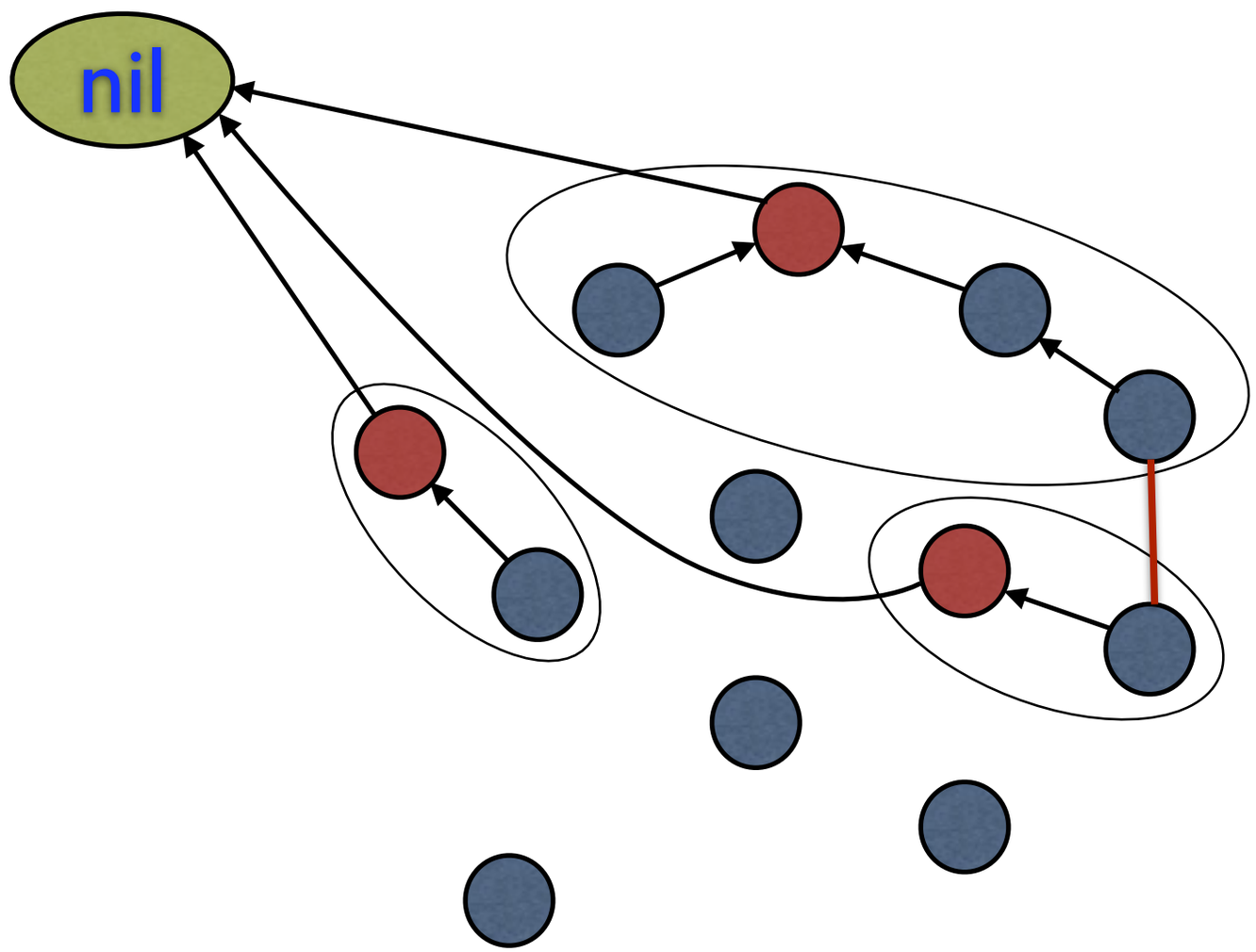
Honnan lehet tudni, hogy az adott él 2 végén lévő pont egy halmazban van-e?

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) ;
   }
 }
```

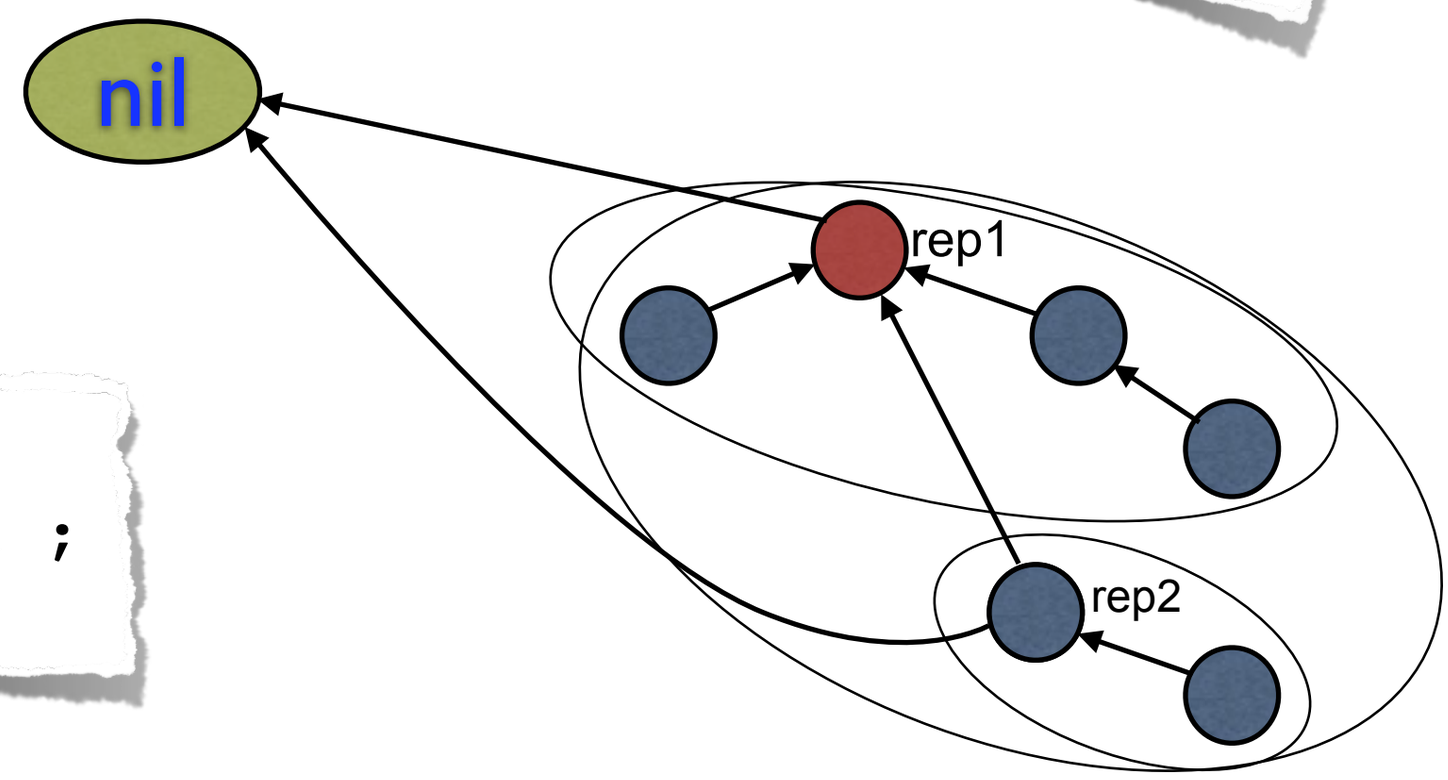
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
   }
   return q ;
}
```

Kruskal algoritmus halmazerdővel

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

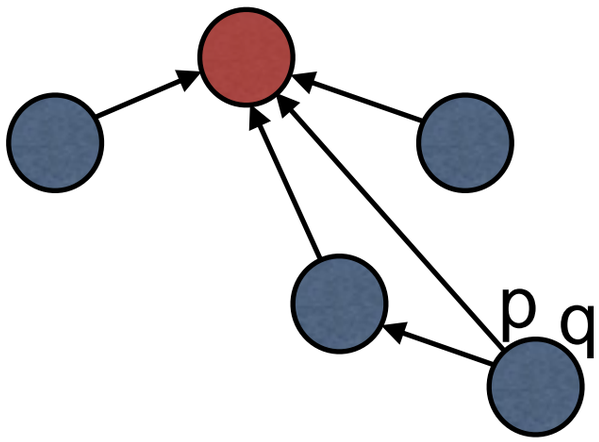
```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

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



Halmazerdő tömörítése

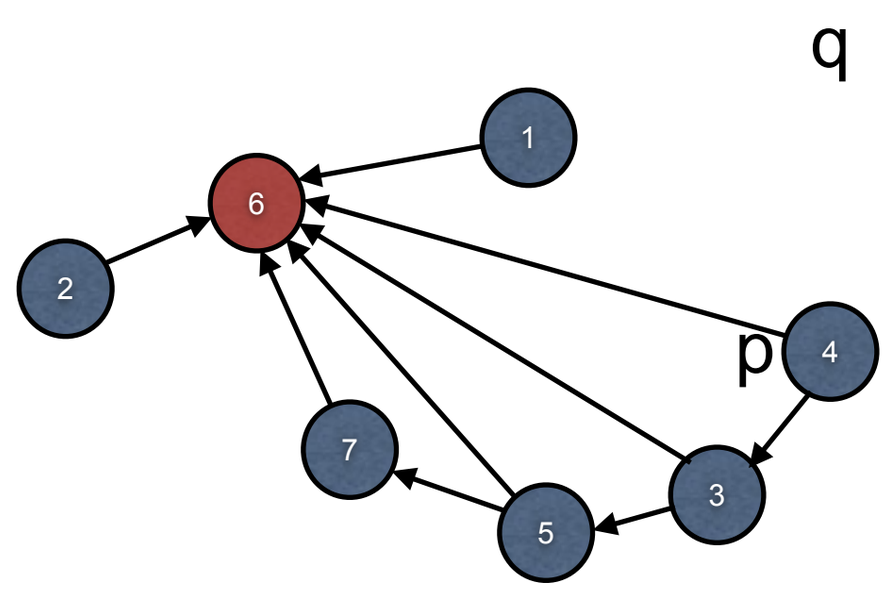
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
      q.kovetkezo=p.kovetkezo ;
   }
   return p ;
}
```



Halmazerdő hatékonyabb tömörítése

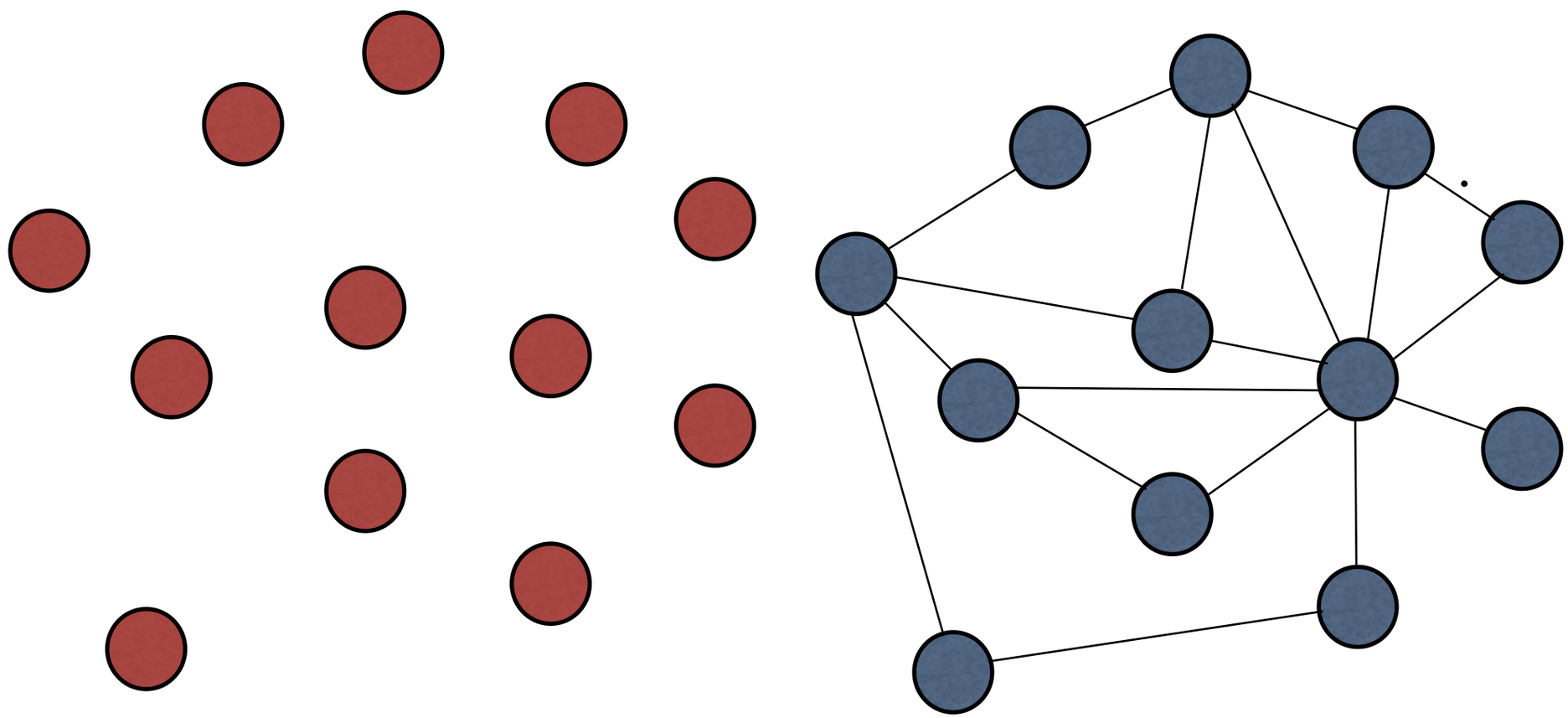
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      if (p.kovetkezo.kovetkezo!=nil) sorba(p) ;
      p=p.kovetkezo ;
   }
   while (!sor_ures()) {
      q=sorbol() ;
      q.kovetkezo=p ;
   }
   return p ;
}
```

5 3 4



Halmazok egyesítése

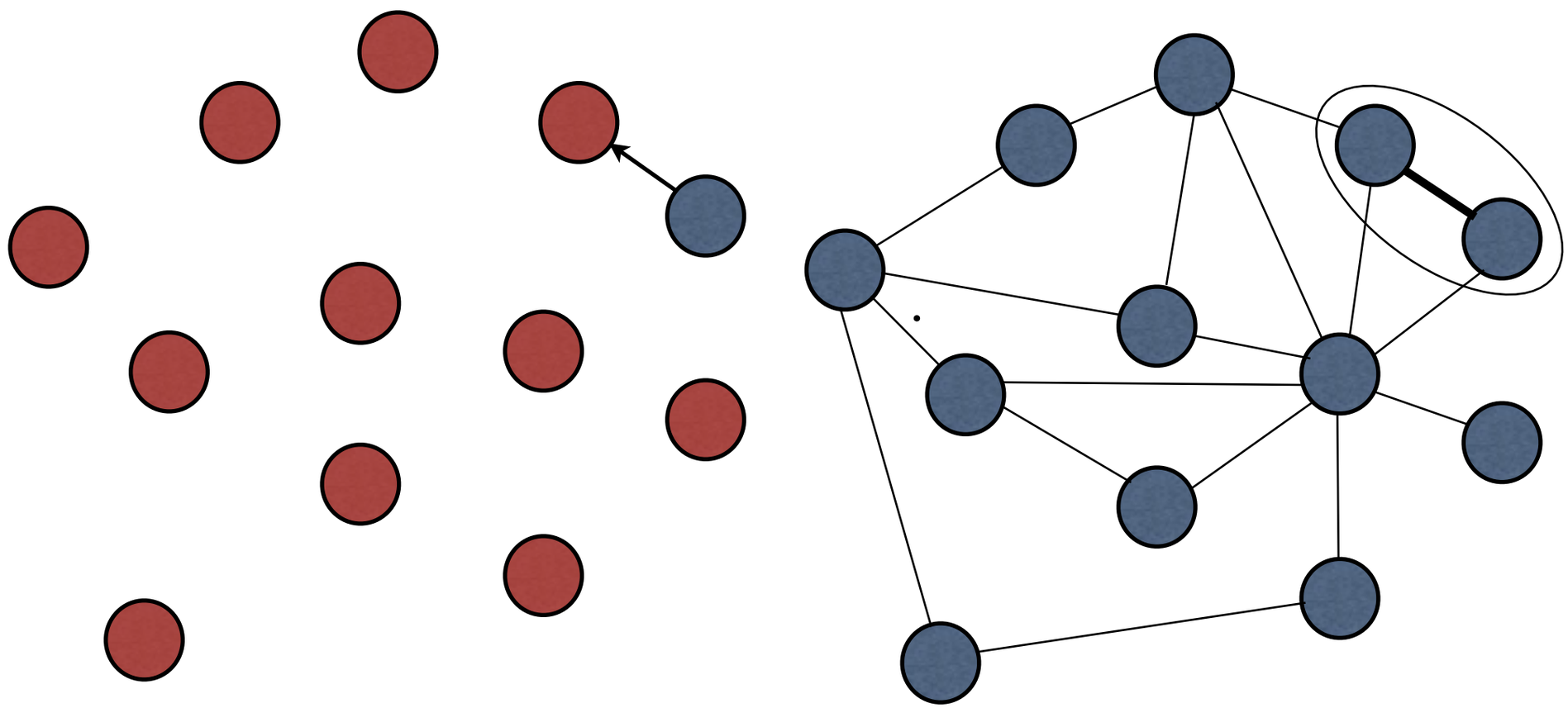
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

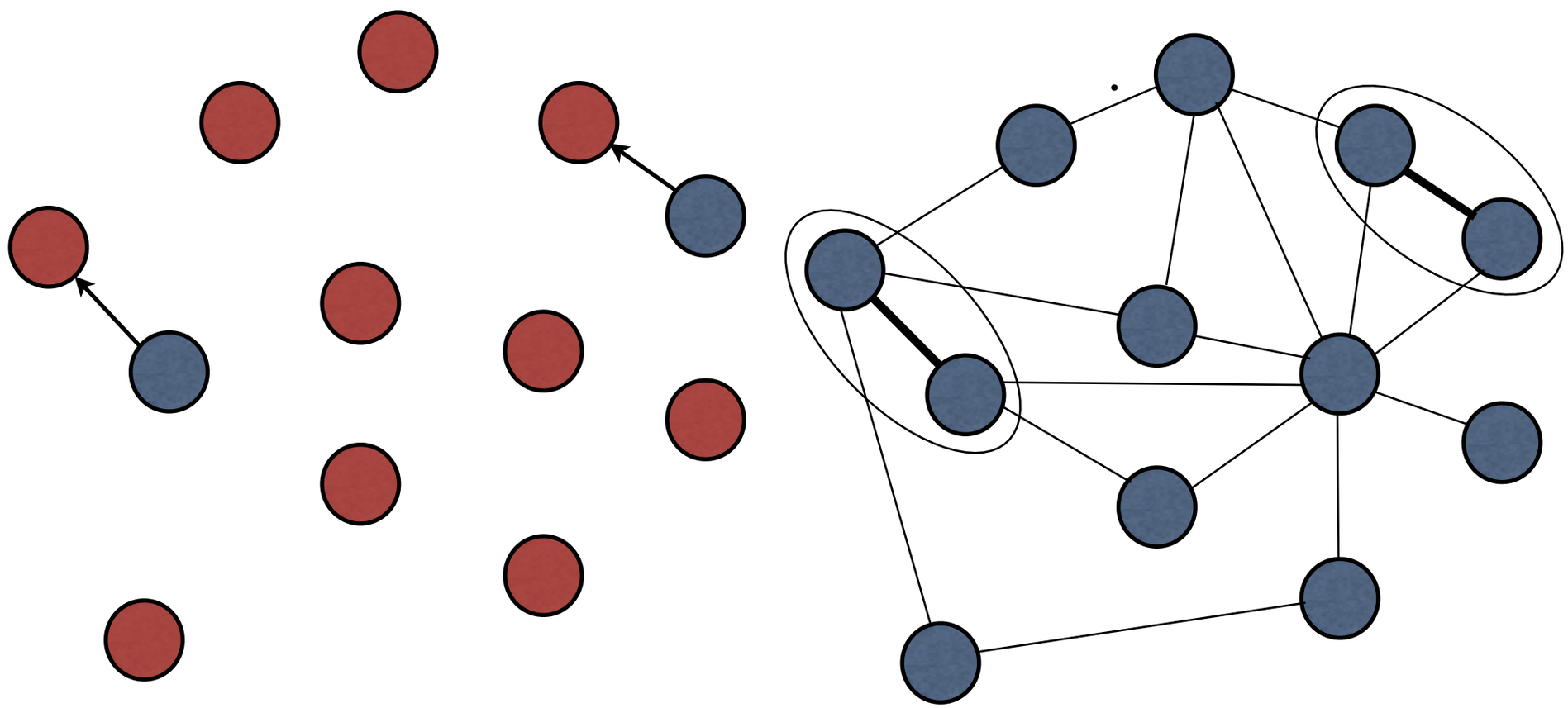
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

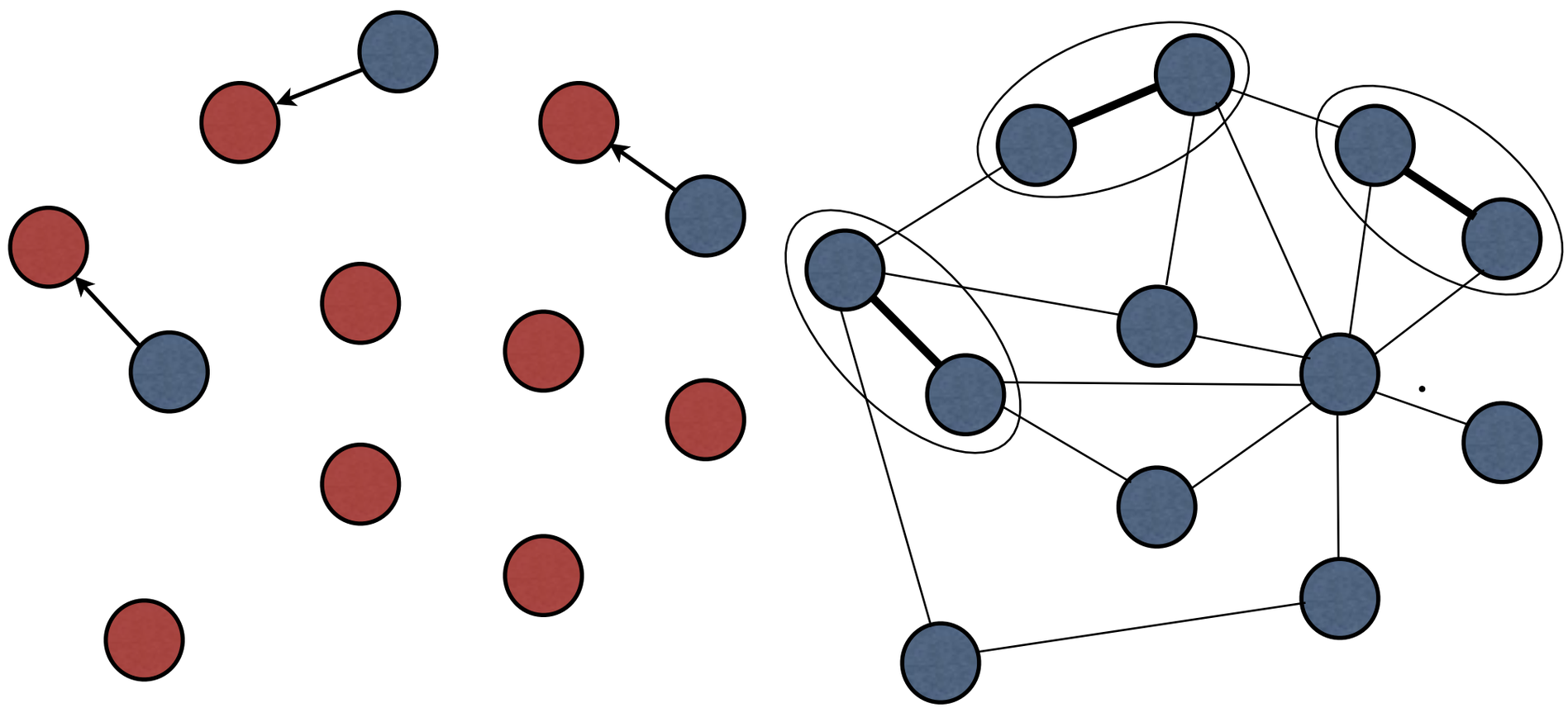
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

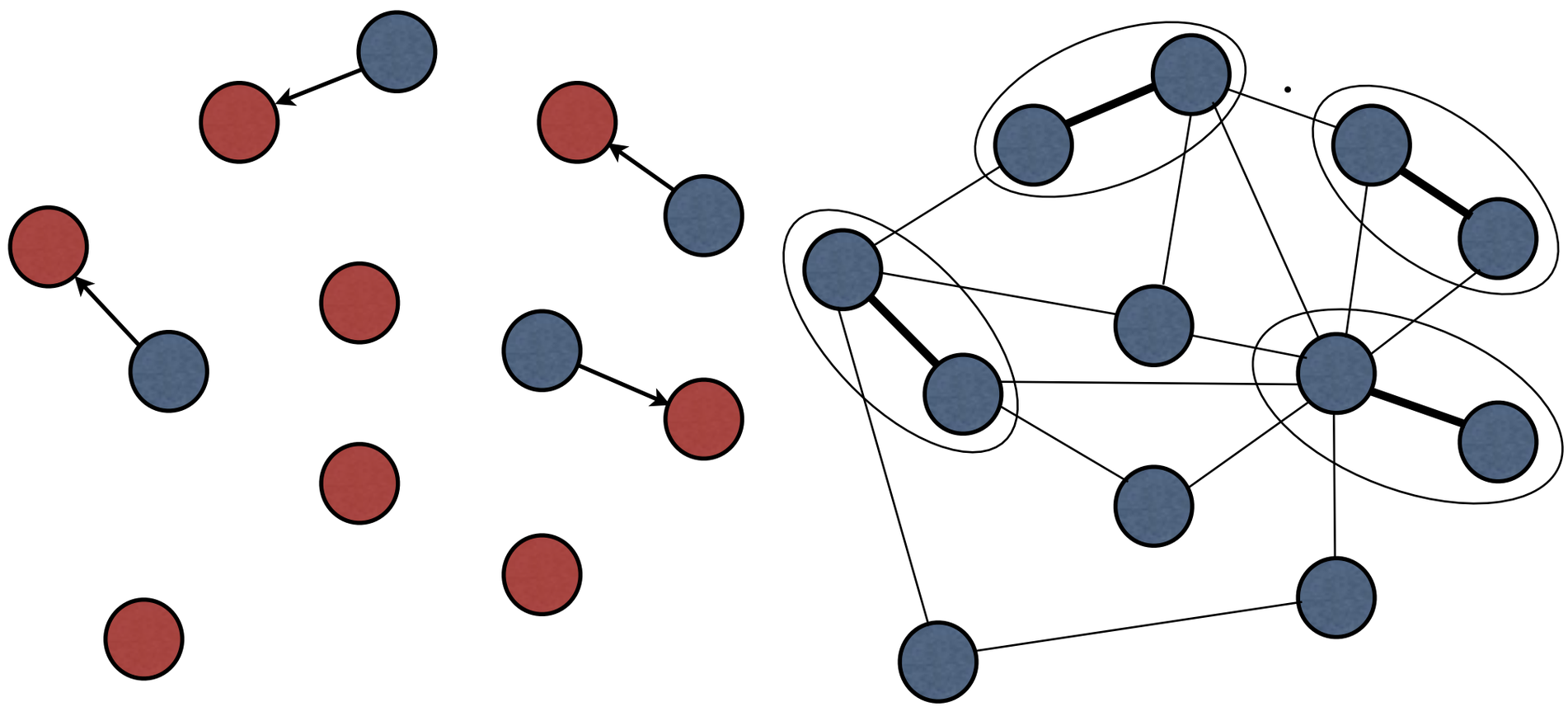
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

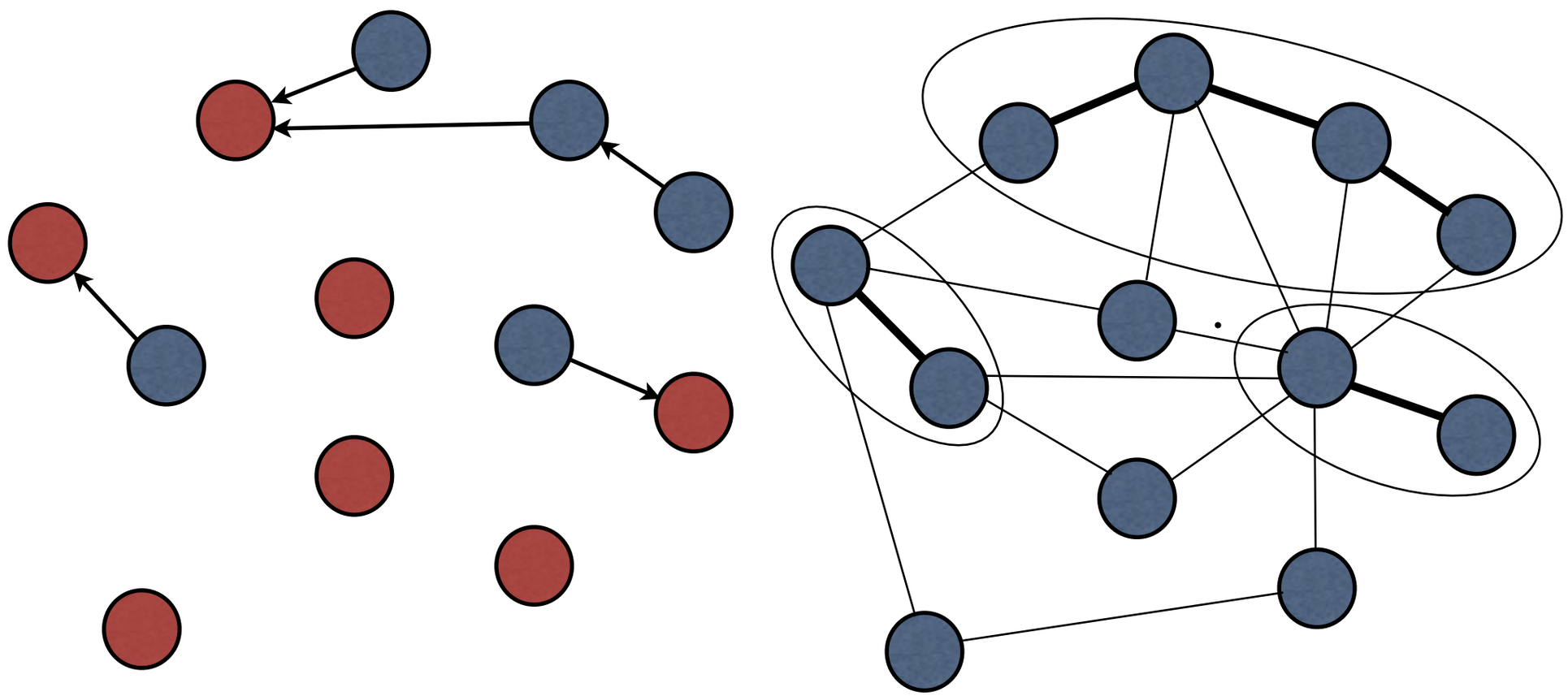
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

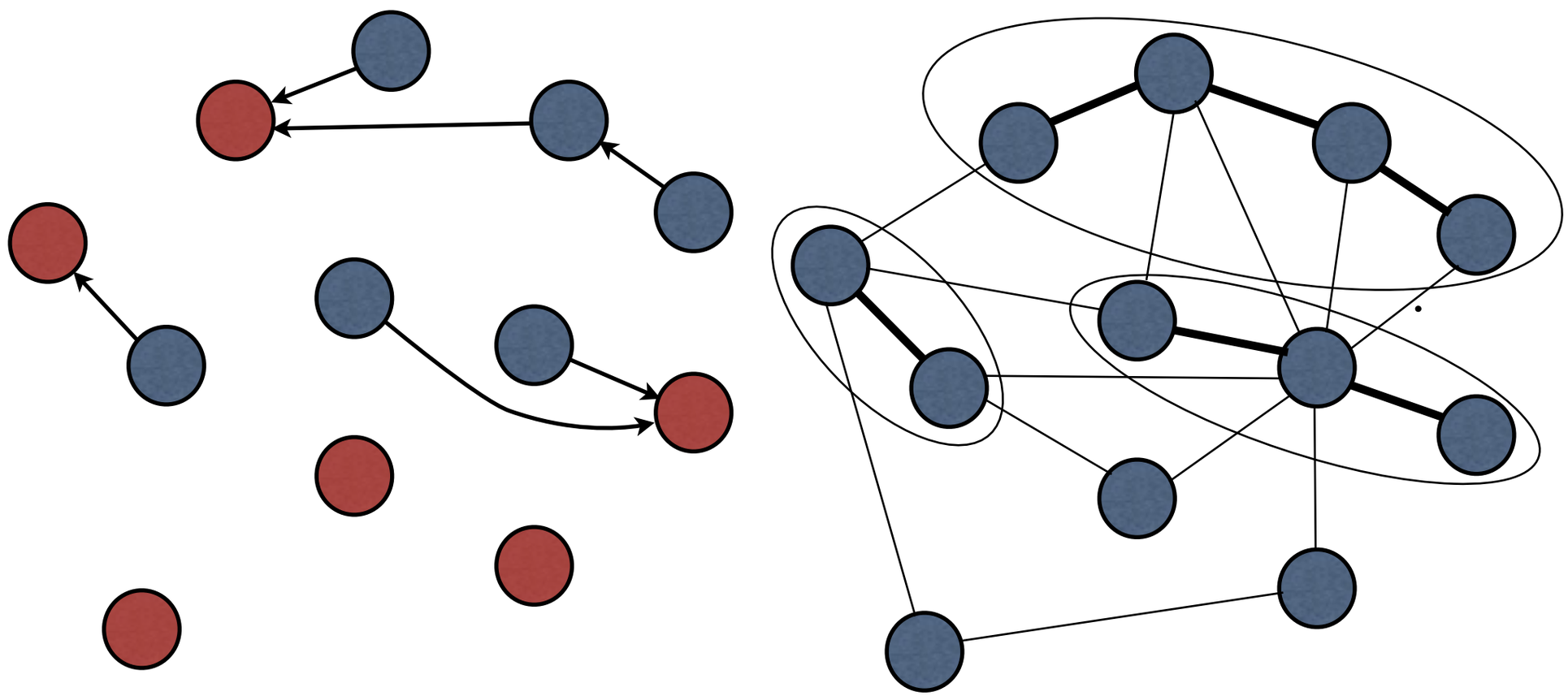
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

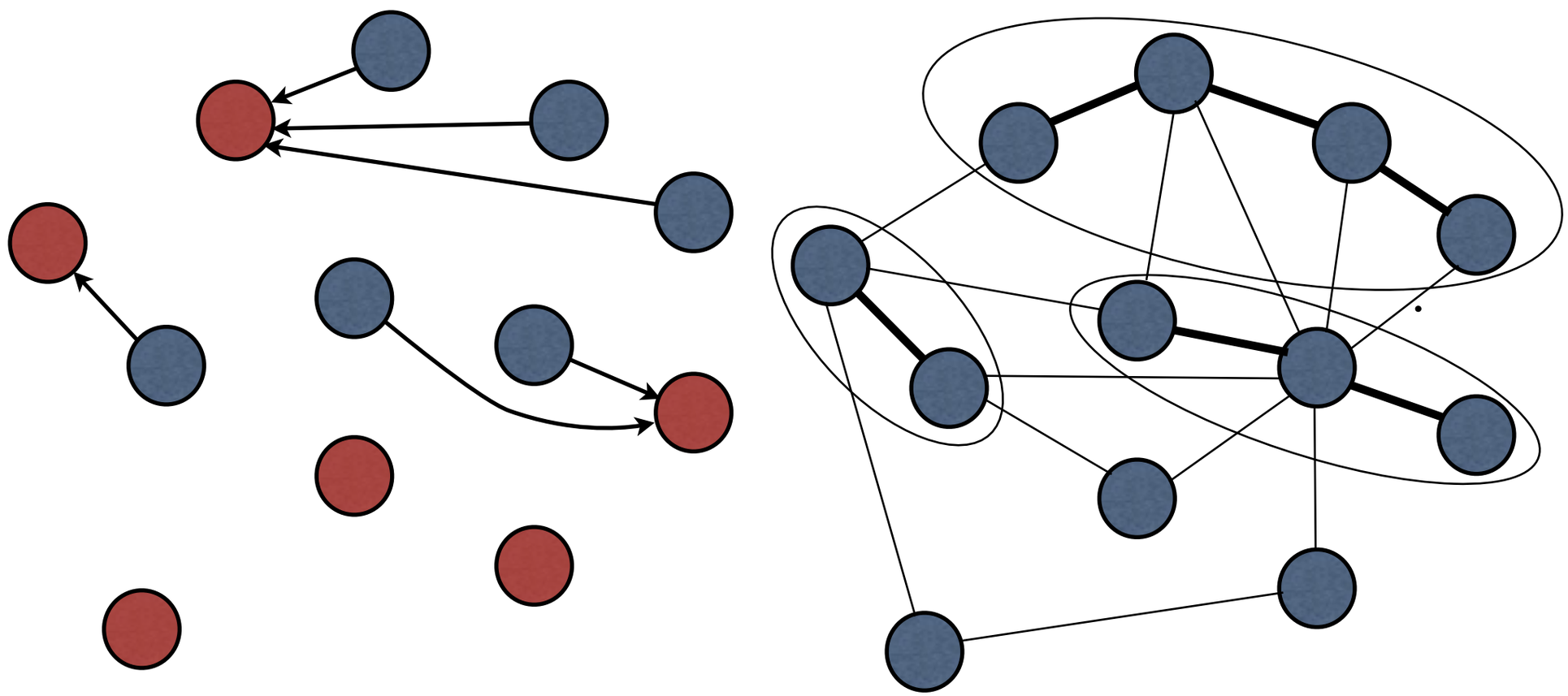
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

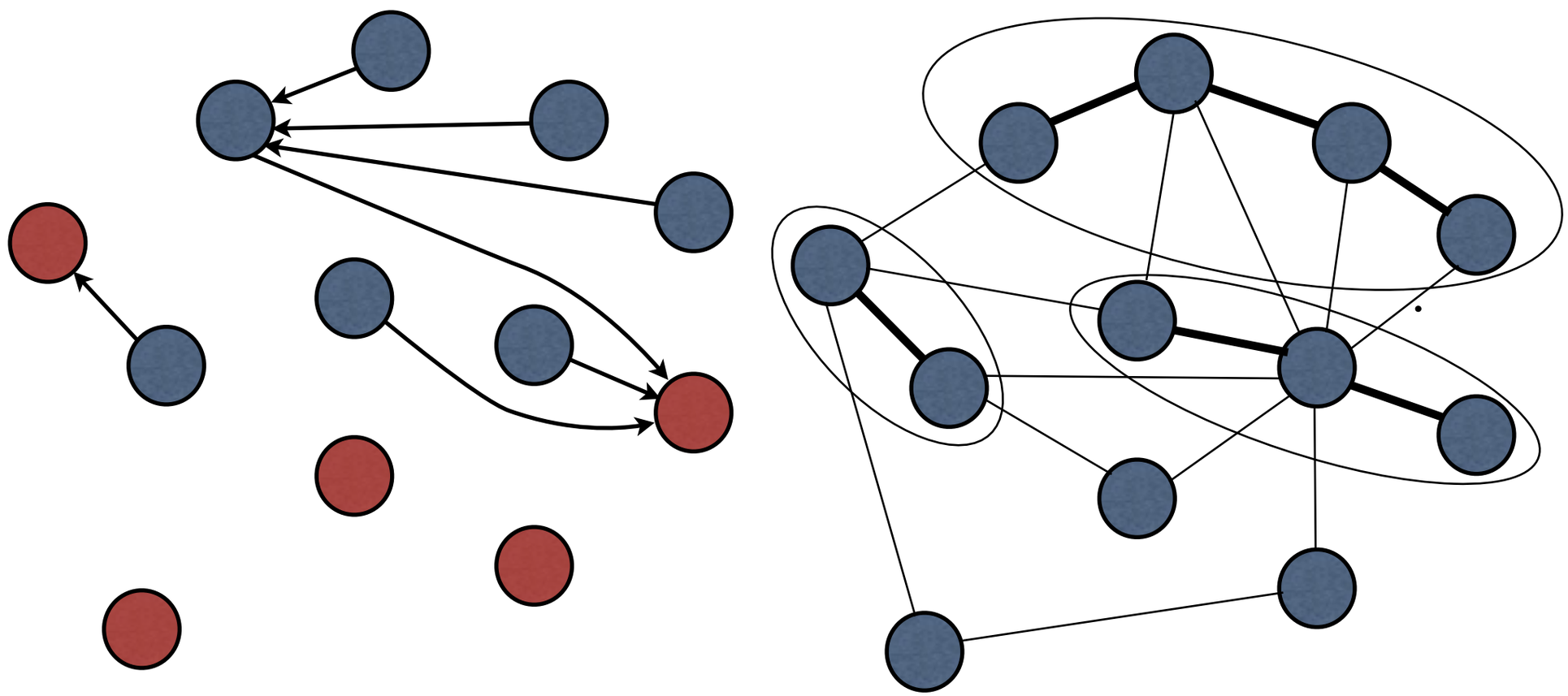
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

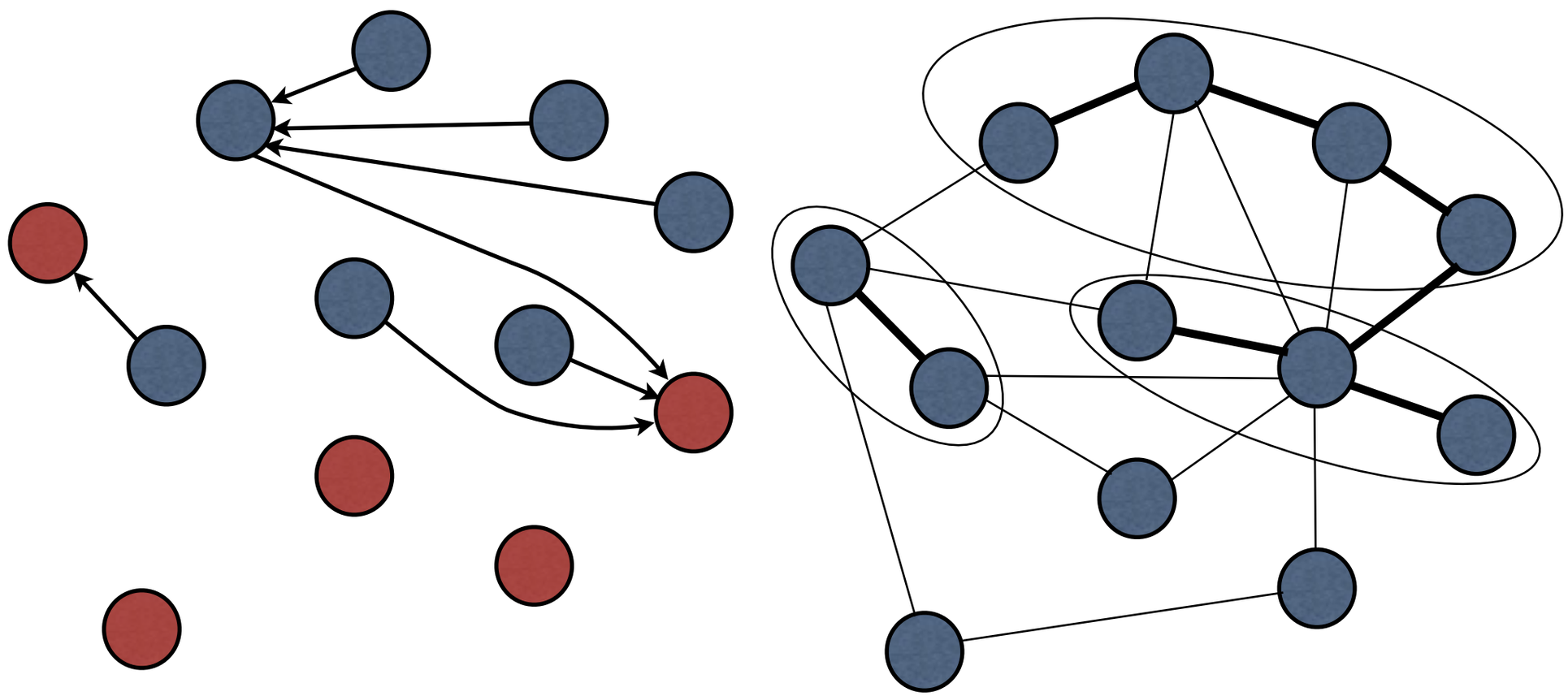
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

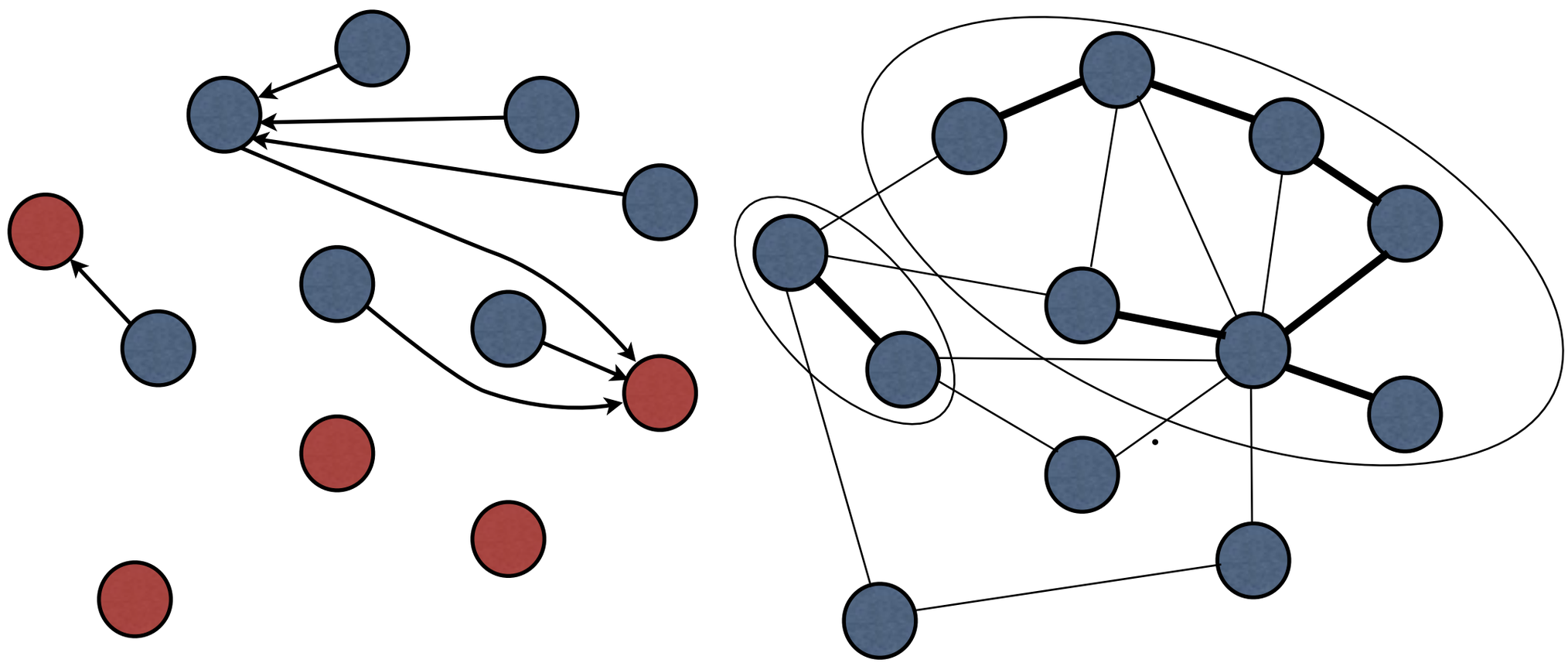
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

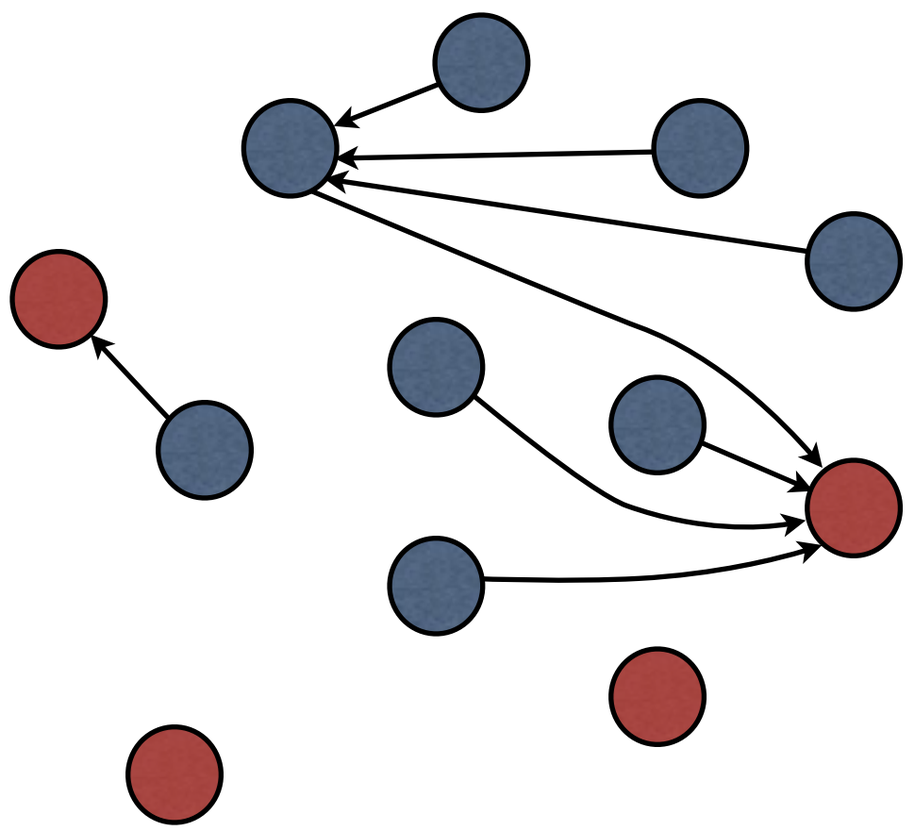
Kruskal algoritmus

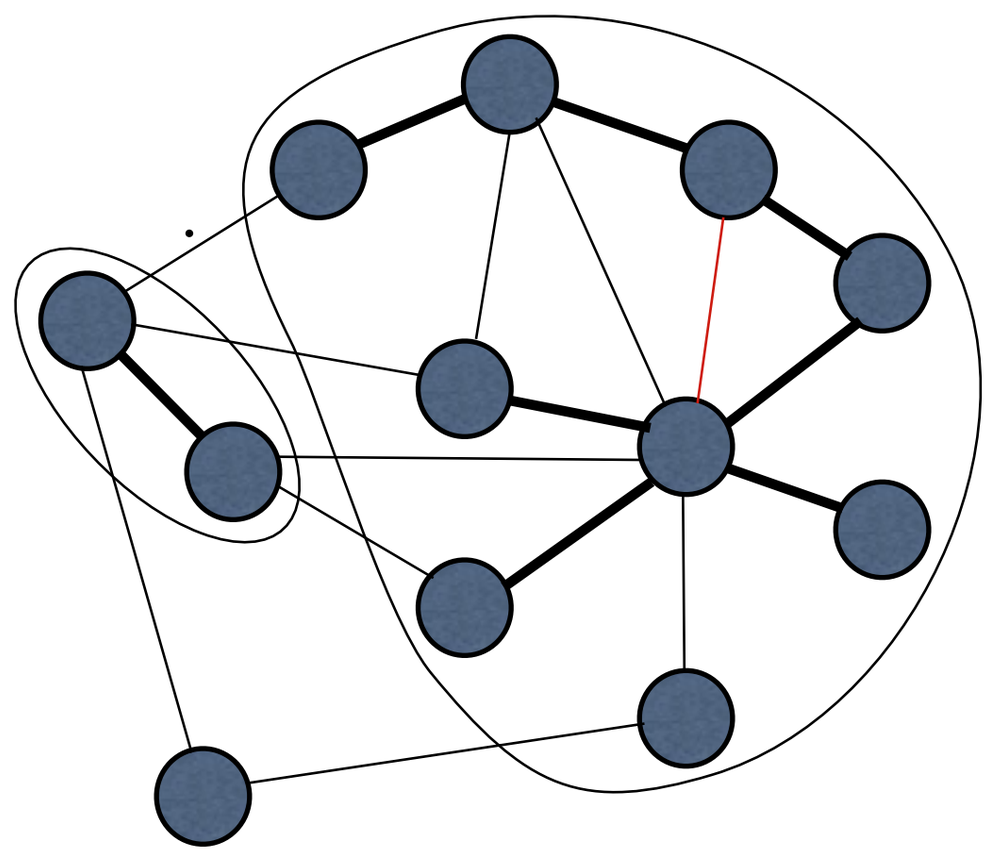


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

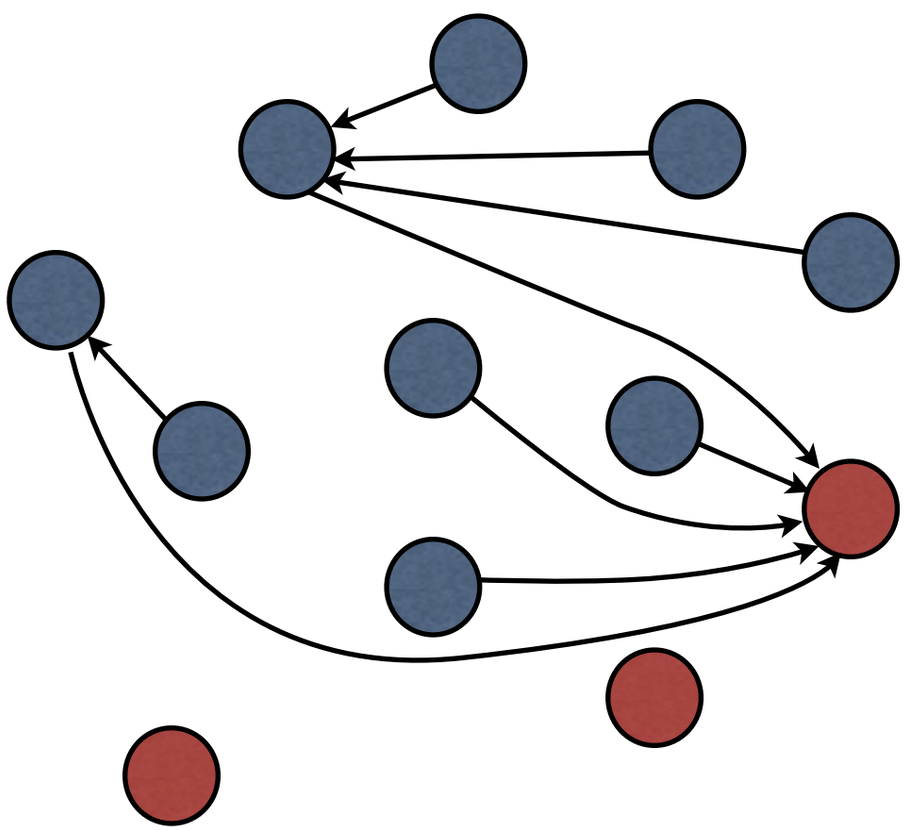


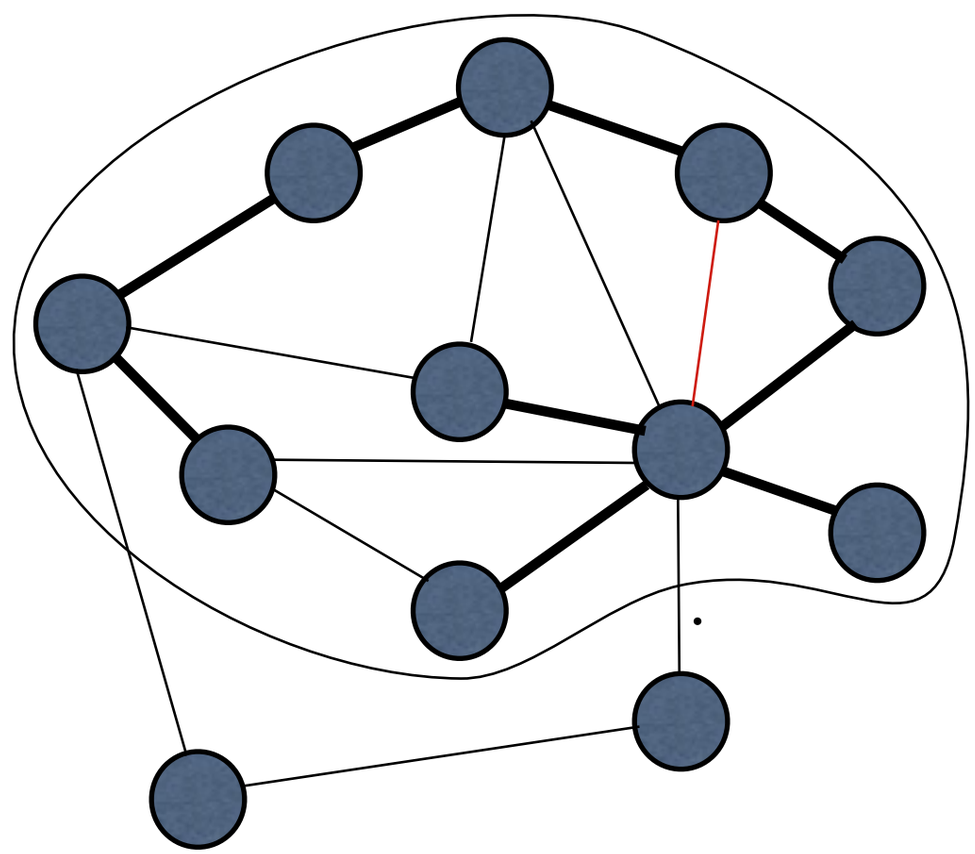


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

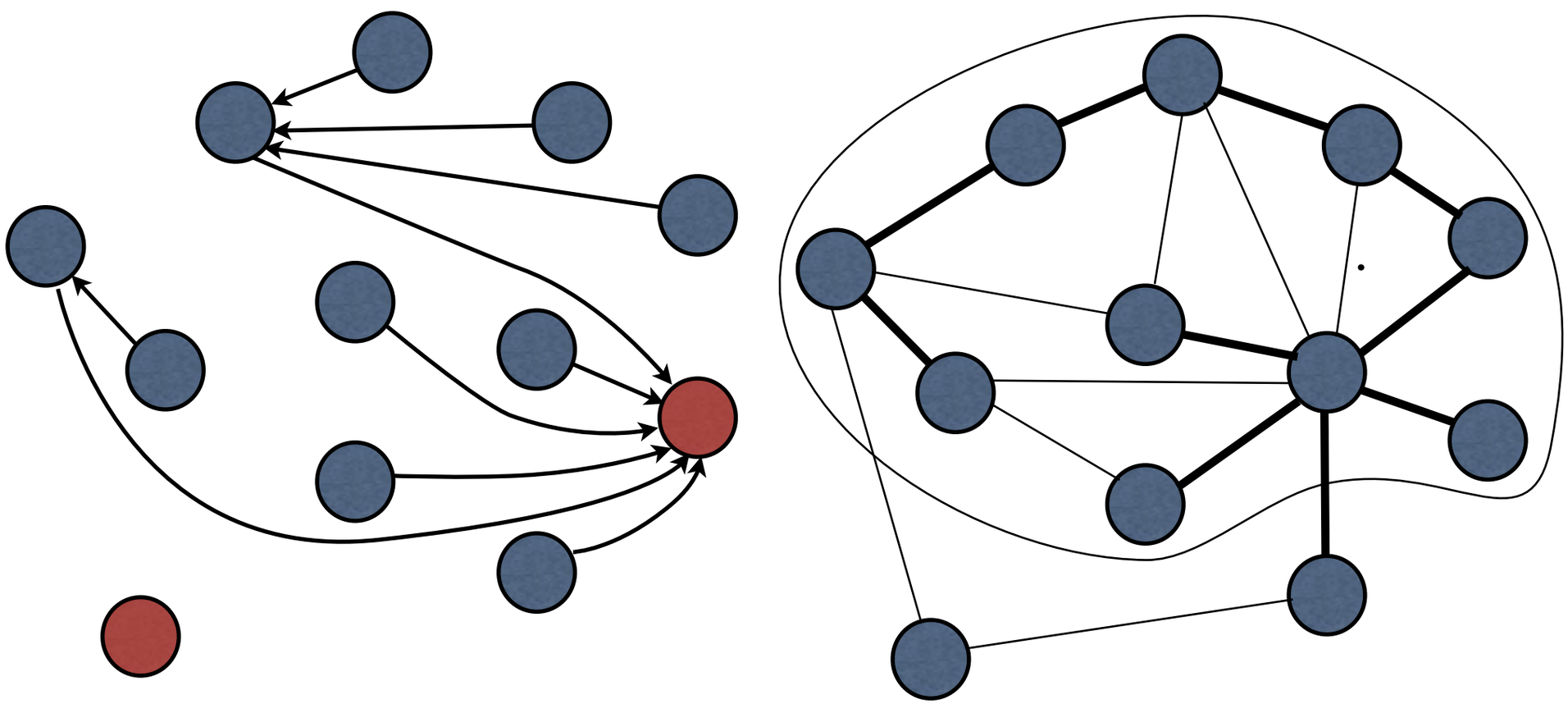




Halmazok egyesítése

halmazerdő felépítése

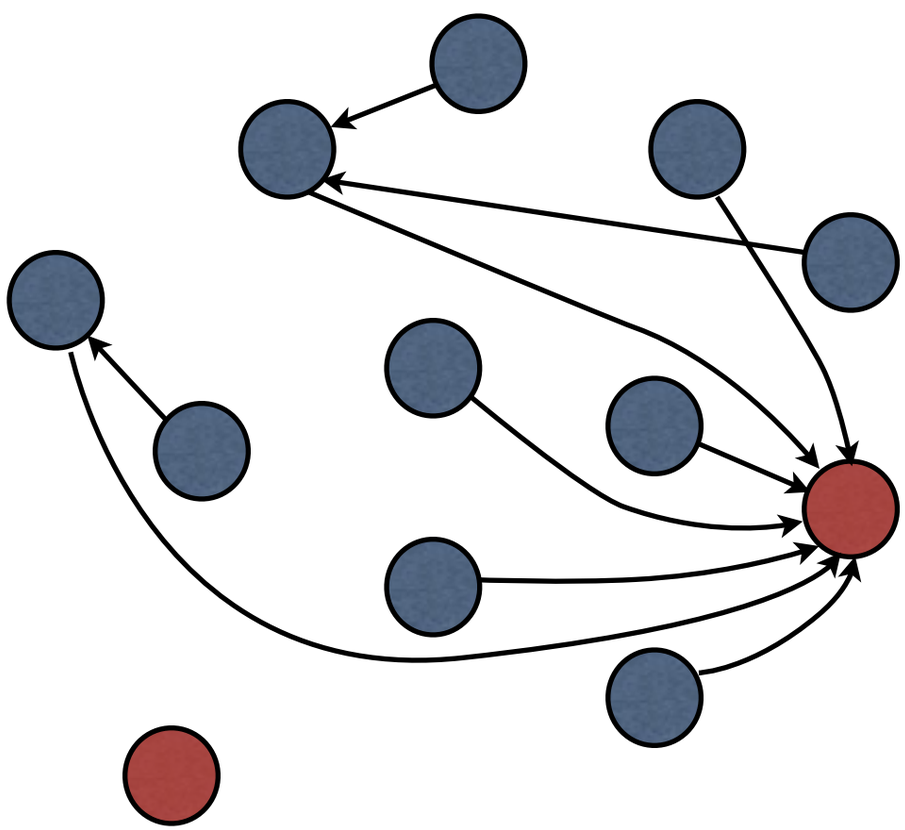
Kruskal algoritmus

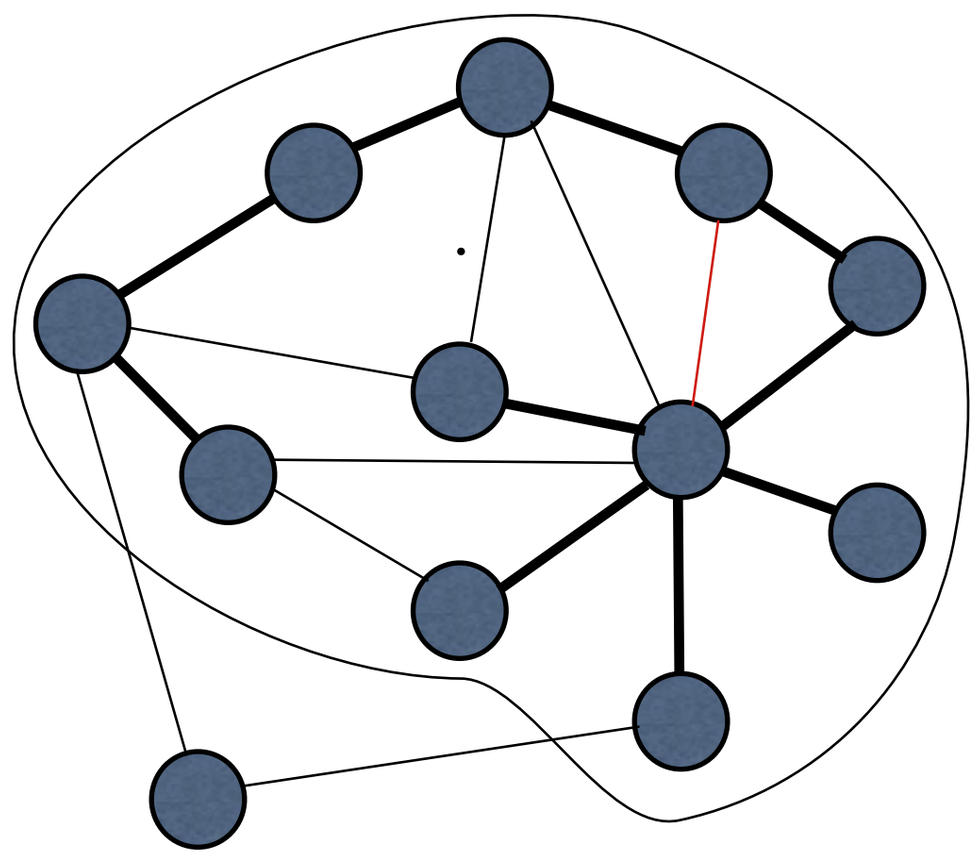


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

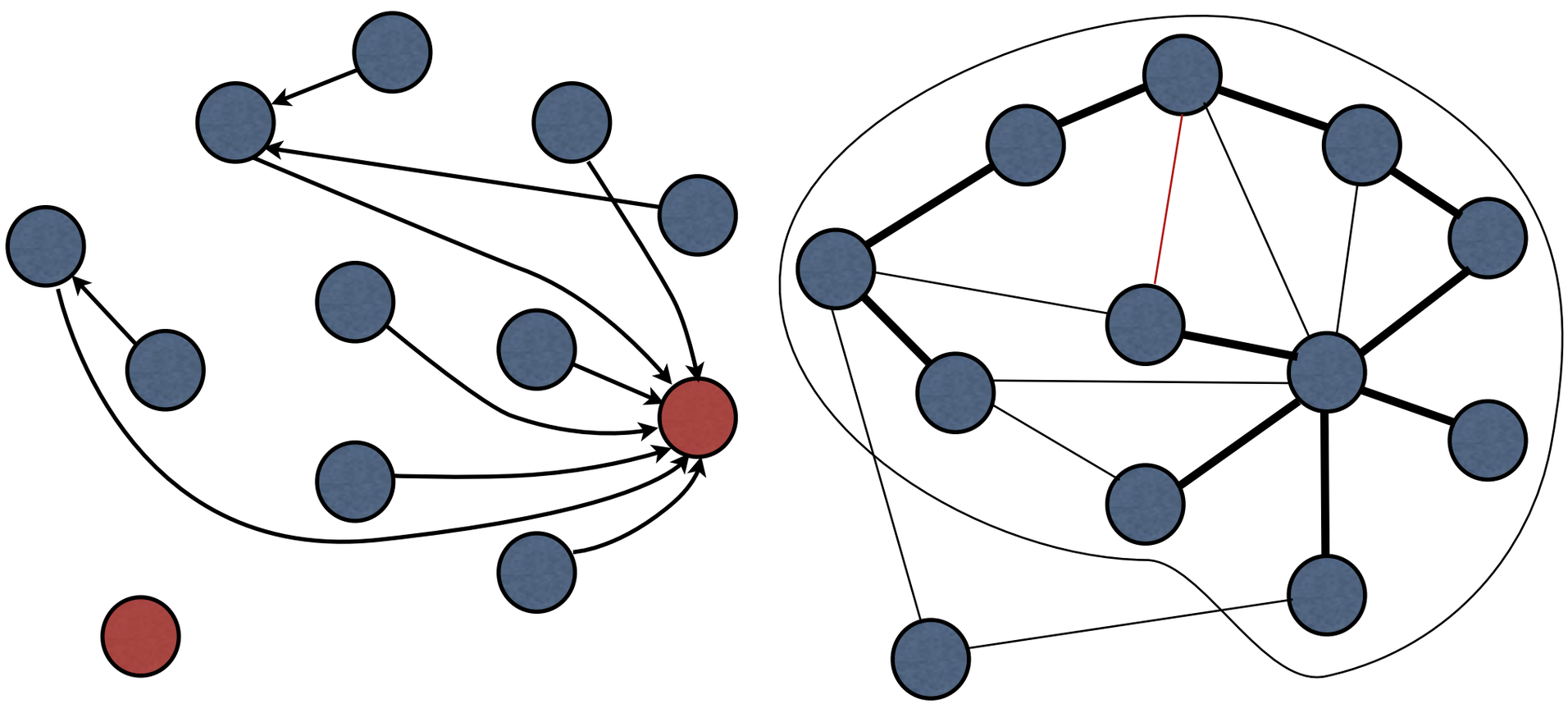




Halmazok egyesítése

halmazerdő felépítése

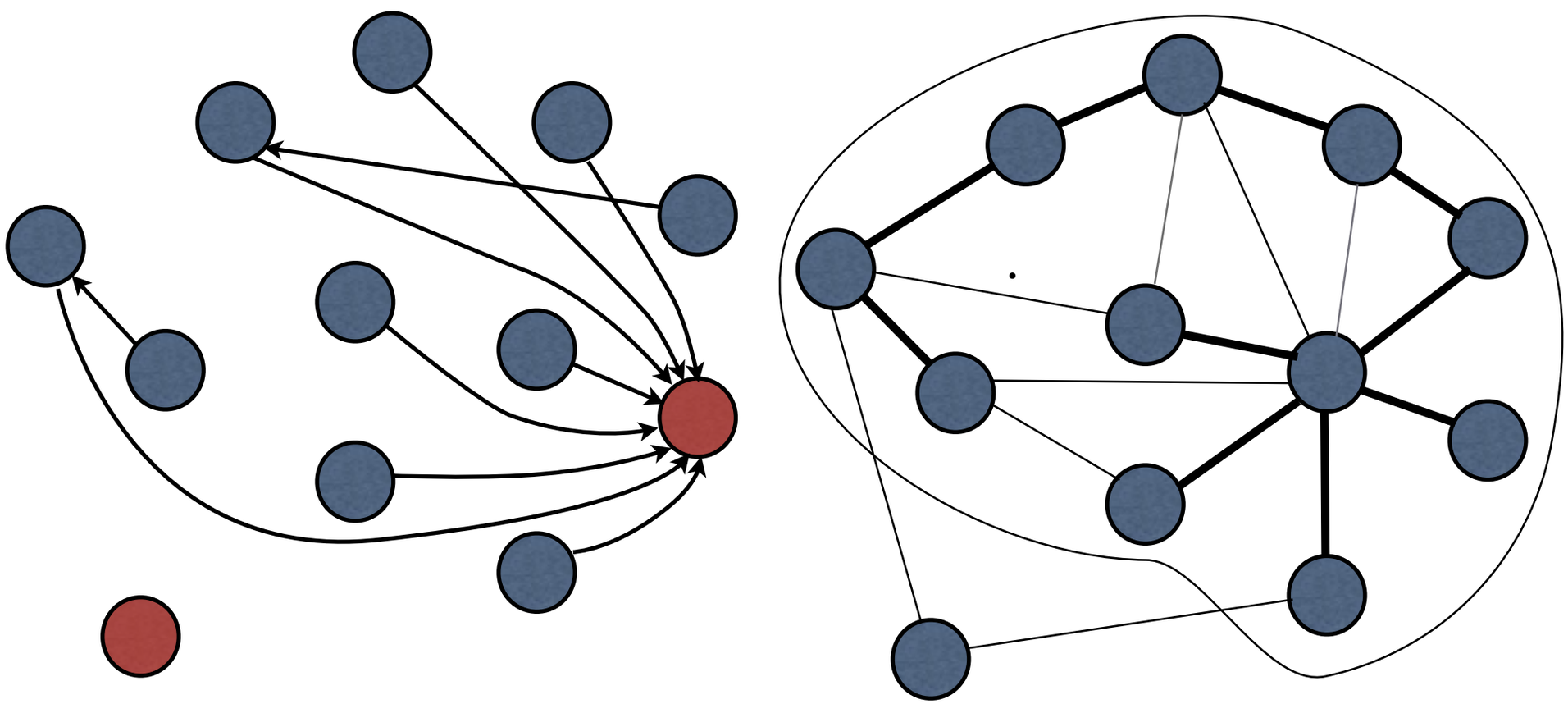
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

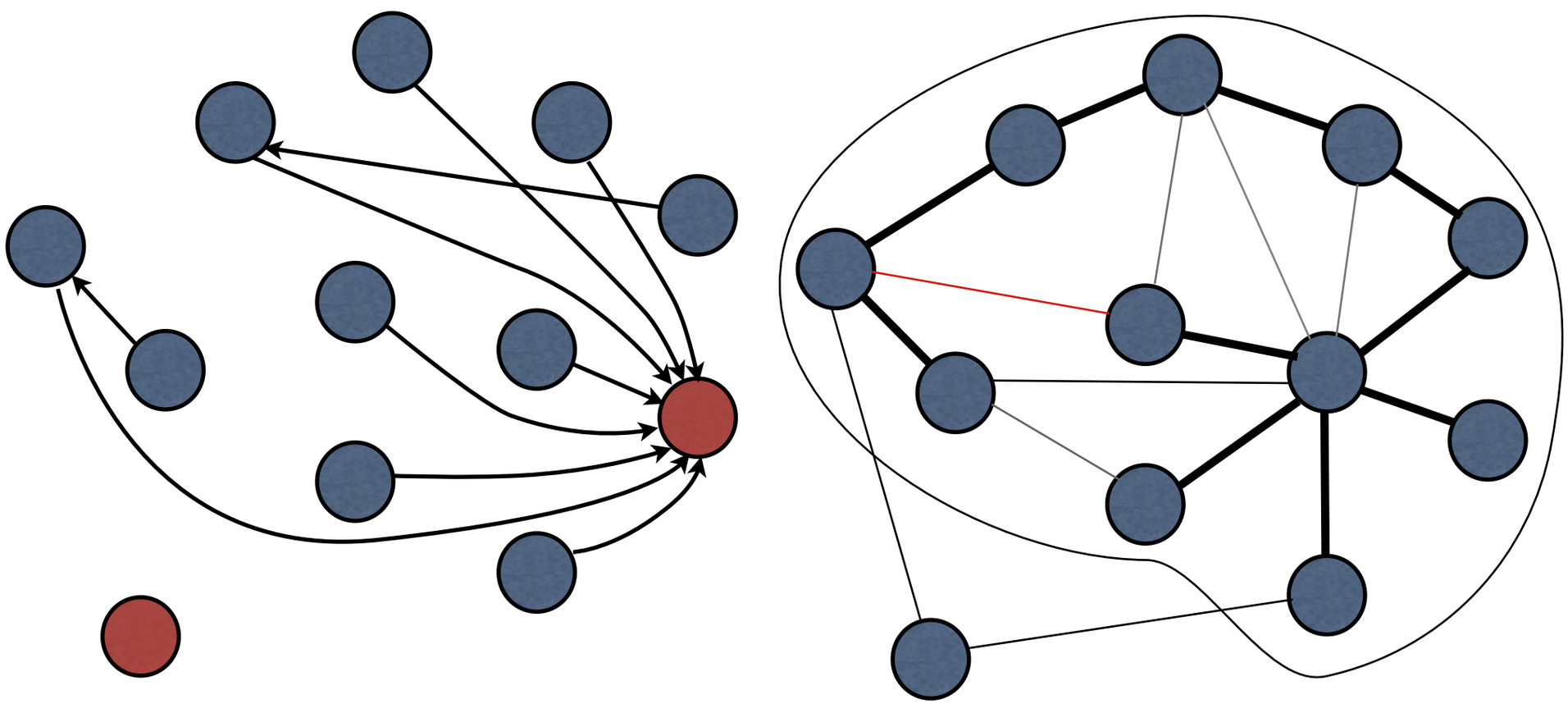
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

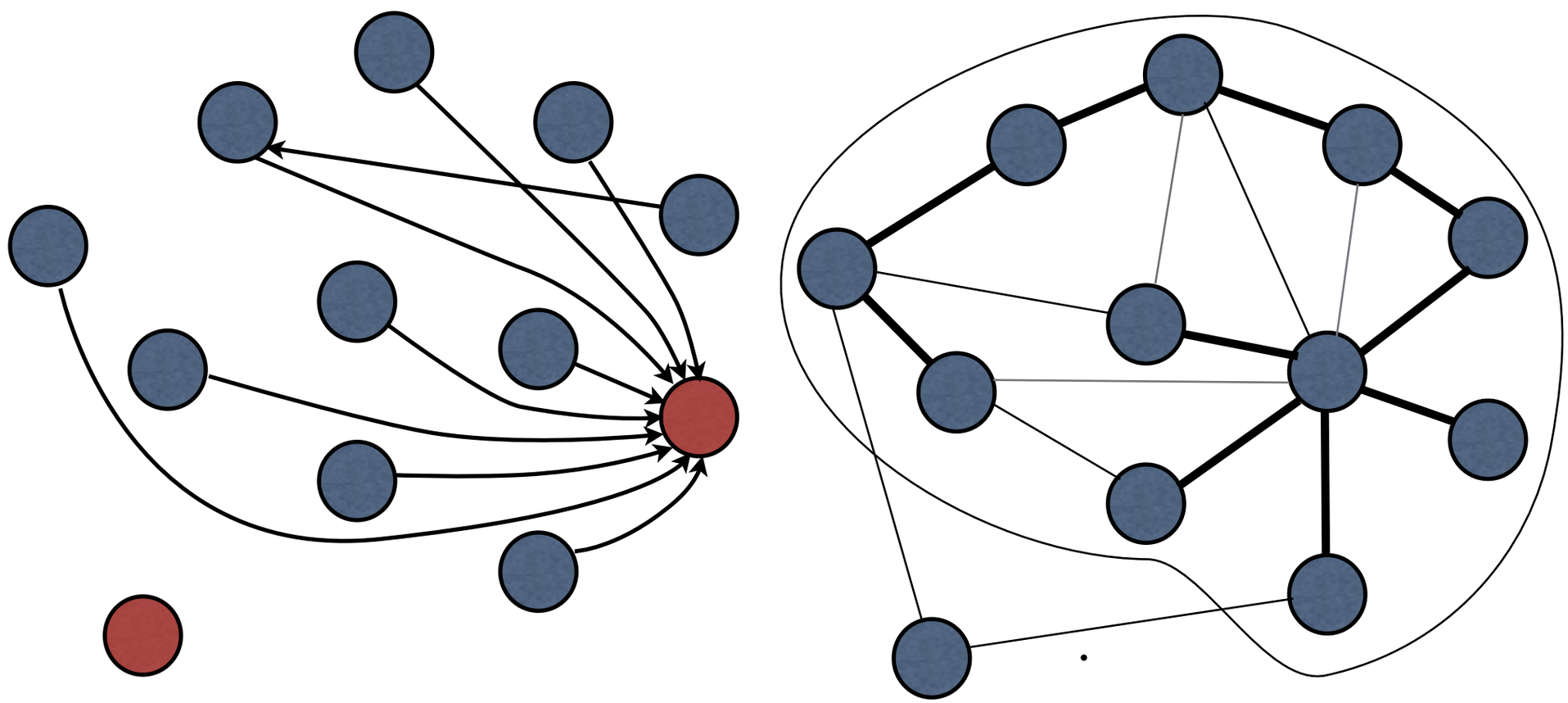
Kruskal algoritmus



Halmazok egyesítése

halmazerdő felépítése

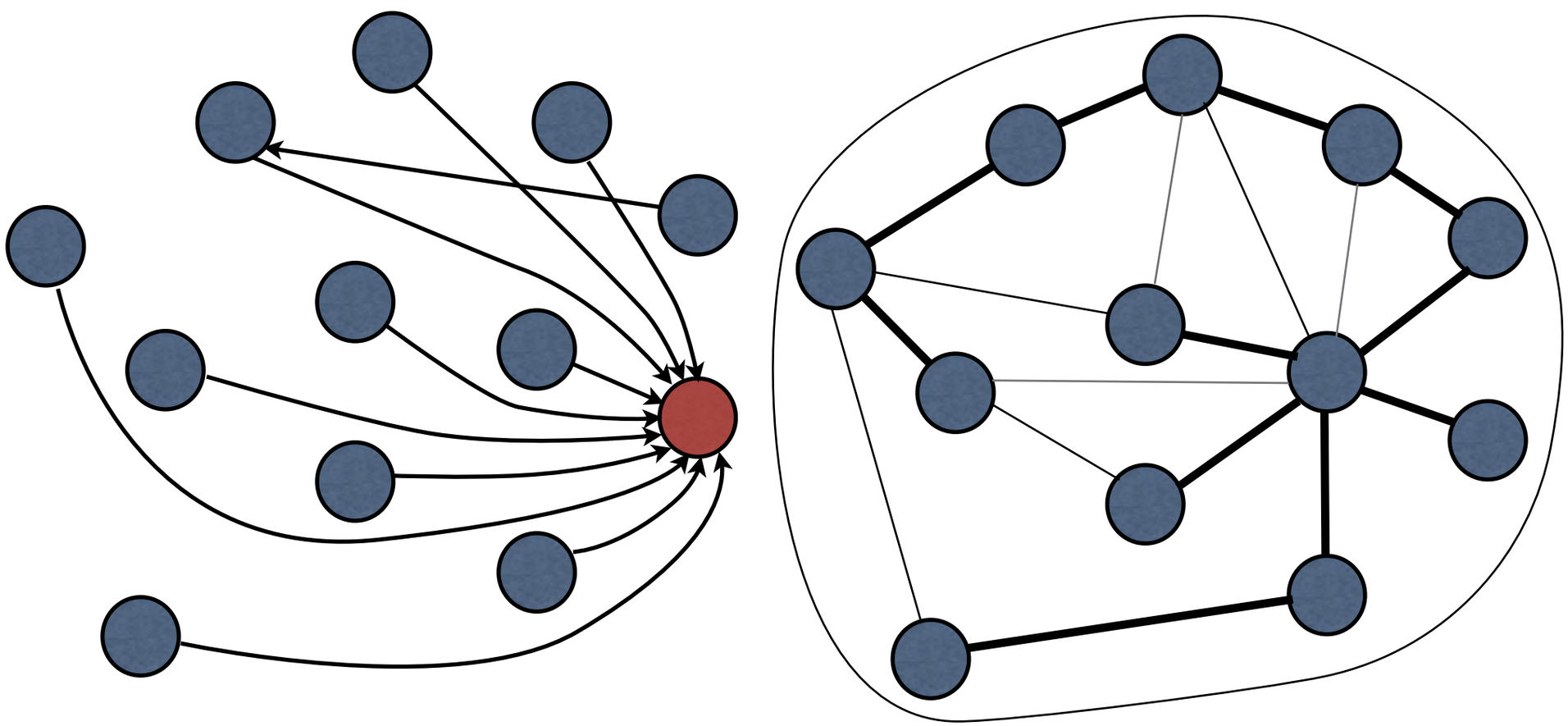
Kruskal algoritmus



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) ;
   }
}
```

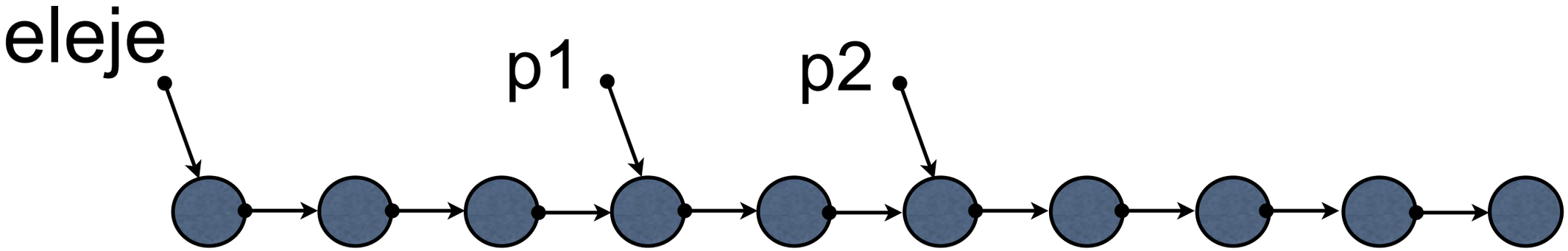
unio 1-el hosszabítja az utakat

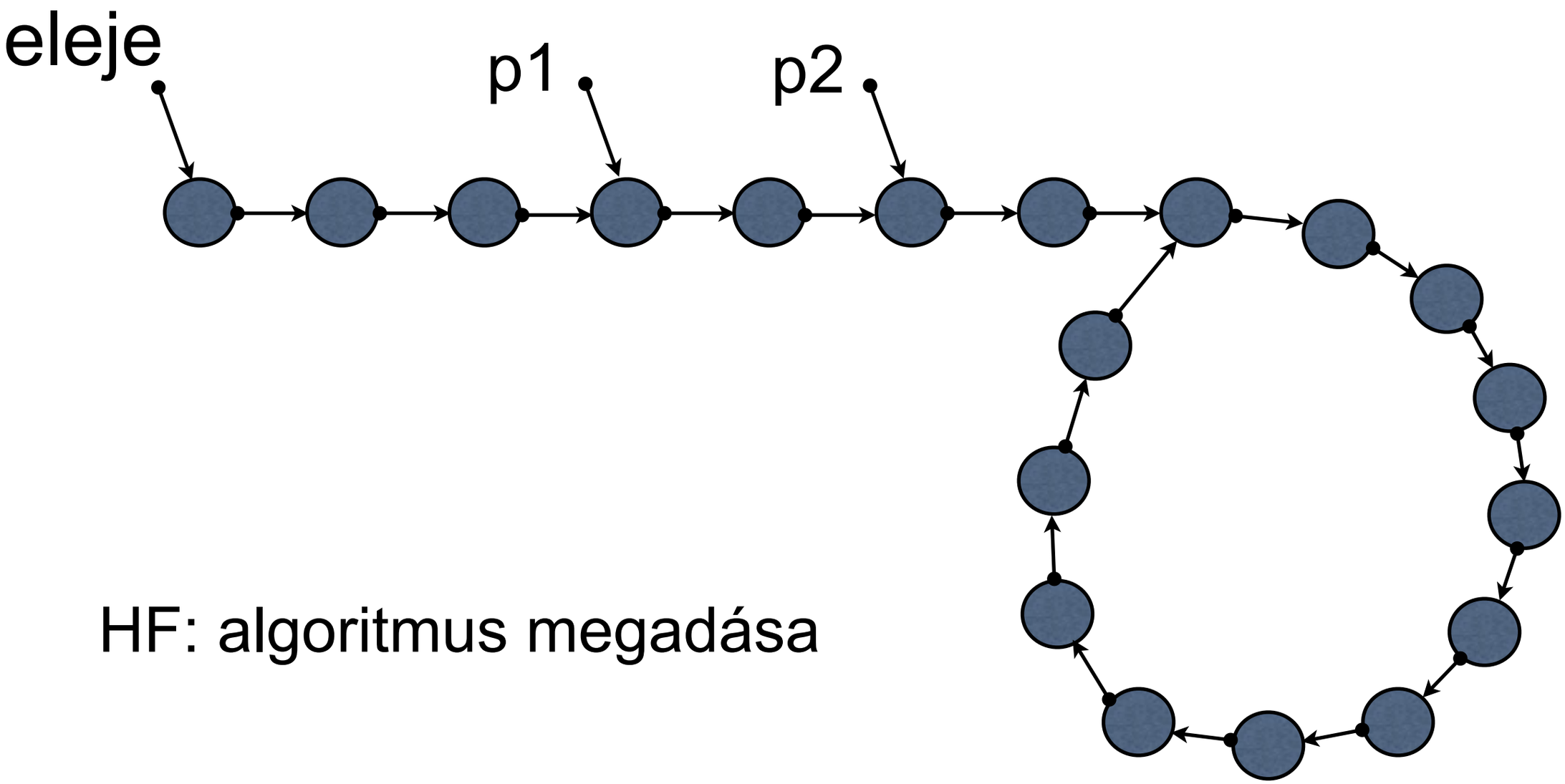
rep 1-hosszúra rövidíti az érintett pontokon az utakat

rep legalább 2-szer többször fut le mint unio

HF: halmazerdő átlagos magassága

Futási idő elemzése: **[2] 444-455**





HF: algoritmus megadása

problémamegoldási stratégiák

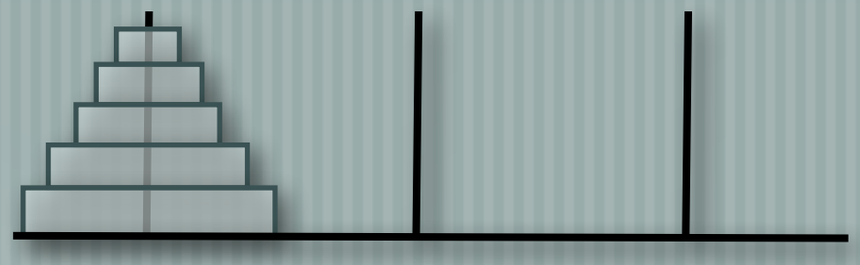
: egyszerűsítés

Rekurzió

Rekurzió módszere

- Vizsgáljuk meg a problémát! Van-e ekvivalens részprobléma?
- Oldjuk meg a problémát, mintha ismernénk a részprobléma megoldását!
- Ha elemi problémához jutunk, oldjuk meg!

Hanoi tornyai legendája



- Kolostor, 100 pap
- Adott: 3 rúd, 100 korong.
- Cél: 1 → 2 úgy, hogy csak nagyobbra rakhatunk kisebbet, és egyszerre csak 1 korongot mozgatunk.
- Főpap megoldása:
  - a tanítványaim átpakolják a felső 99-et 1 → 3
  - én átrakom a legalsót 1 → 2
  - a tanítványaim átpakolják a 99-et 3 → 2

Rekurzív definíció

- legyen f(n, a, b) jelentése: pakolunk n magasat a ⇒ b
- a, b a rúd sorszáma 1, 2 vagy 3.
- definíció:

  $f(n, a, b) = a \Rightarrow b$, ha $n = 1$

  $f(n, a, b) = f(n-1,a,6-a-b) + a \Rightarrow b + f(n - 1, 6 - a - b, b)$, ha $n > 1$
- a 6 - a - b mindig azt rudat jeleni amelyiknek a sorszáma nem a és nem b, mivel a rudak sorszámának összege 6:

  $a + b + c = 6 \Rightarrow c = 6 - a - b$

Hanoi tornyai megoldás

$f(n, a, b) = a \Rightarrow b$, ha $n = 1$

$f(n, a, b) = f(n-1,a,6-a-b) + a \Rightarrow b + f(n-1, 6-a-b, b)$

legyen a torony 3 magas, cél: összes $1 \Rightarrow 2$

$f(3,1,2) = f(2,1,3) + 1 \Rightarrow 2 + f(2,3,2)$

$f(2,1,3) = f(1,1,2) + 1 \Rightarrow 3 + f(1,2,3)$

$f(2,3,2) = f(1,3,1) + 3 \Rightarrow 2 + f(1,1,2)$

$1 \Rightarrow 2,\ 1 \Rightarrow 3,\ 2 \Rightarrow 3,\ 1 \Rightarrow 2,\ 3 \Rightarrow 1,\ 3 \Rightarrow 2,\ 1 \Rightarrow 2$

TÜKÖRSZÓ PROBLÉMA

Adott S string.
Minimum hány karaktert kell törölni, hogy tükörszót kapjunk?

```
from functools import cache

@cache # rekurzió-memorizálás
```

$f(i, j) =$

- ha $i >= j$, akkor: 0
- egyébként
  - ha $s[i] = s[j]$, akkor $f(i+1, j-1)$
  - egyébként: $min(f(i+1, j), f(i, j-1)) + 1$

```
def tsz(s):
    def f(i=0, j=len(s)-1):
        nonlocal s
        if j <= i:
            return 0
        elif s[i] == s[j]:
            return f(i + 1, j - 1)
        else:
            return 1 + min(f(i + 1, j), f(i, j - 1))
    return f()
print(tsz("géza kék az ég"))
# http://www.inf.u-szeged.hu/~tnemeth/examples/algoexamples/Tukorszo_forras.html
```

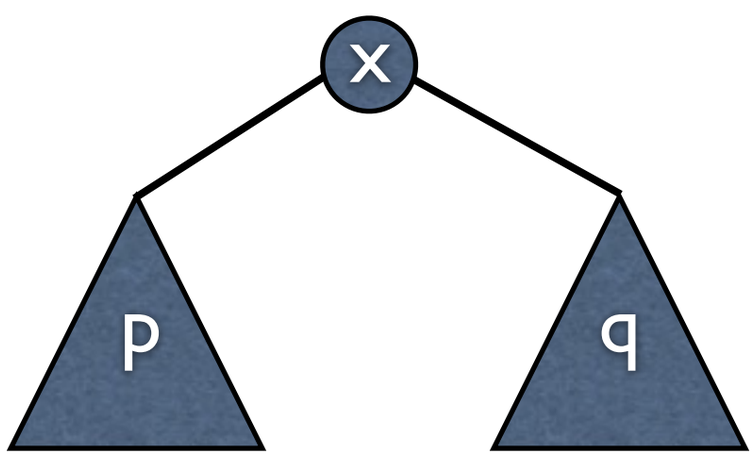
Dinamikus programozás módszere

```
let o = {
    mi(i=0,j=this.s.length-1) {
      !this.t   ? this.t = Array(this.s.length)
                           .fill()
                           .map(v=>Array(this.s.length)
                           .fill(0))
                : null
     return this.t[i][j] ? this.t[i][j] : i>=j ? this.t[i][j] = 1 :
            this.s[i]===this.s[j] ? this.t[i][j] = this.mi(i+1,j-1) :
            this.t[i][j] = Math.min(this.mi(i+1,j),this.mi(i,j-1))+1
   },
    mo(i=0,j=this.s.length-1) {
      let s=this.s.split('')
      do this.t[i][j-1]+1===this.t[i][j] ? s[j--]='.' :
         this.t[i+1][j]+1===this.t[i][j] ? s[i++]='.' : ( i++, j-- )
      while (this.t[i][j]-1)
      return s.join('')
    }
}
o.s = 'pelda'
console.log(`
    ${ o.x = o.mi()-1 } karakter törlésével a tükörszó:
    ${ o.x?o.mo():o.s }
`)
```

Bináris keresőfa

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

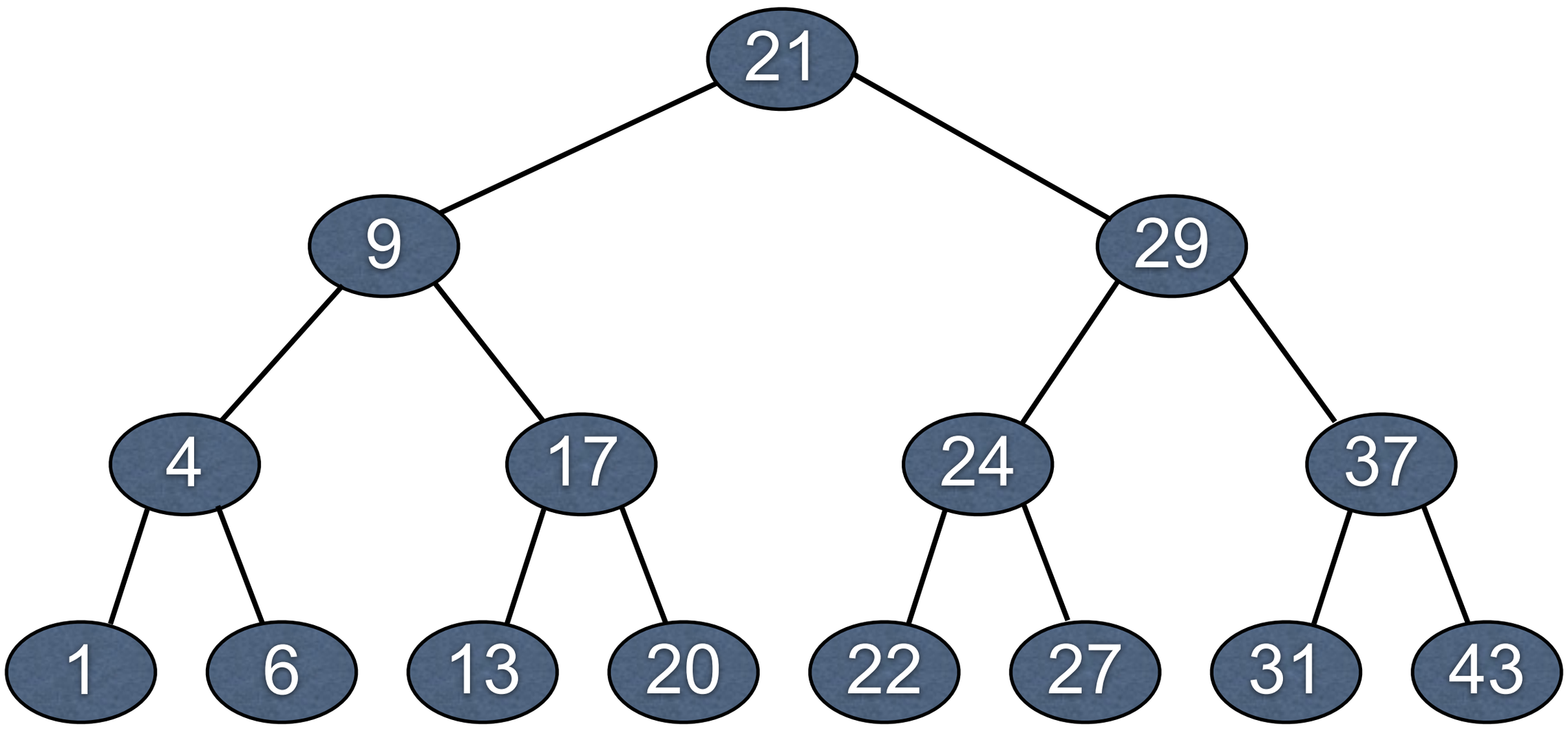
- F bináris fa, $R = \{bal, jobb, apa\}$, $bal, jobb, apa : M \to M$,
- $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
- $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(kulcs(p) \leq kulcs(x) \leq kulcs(q))$



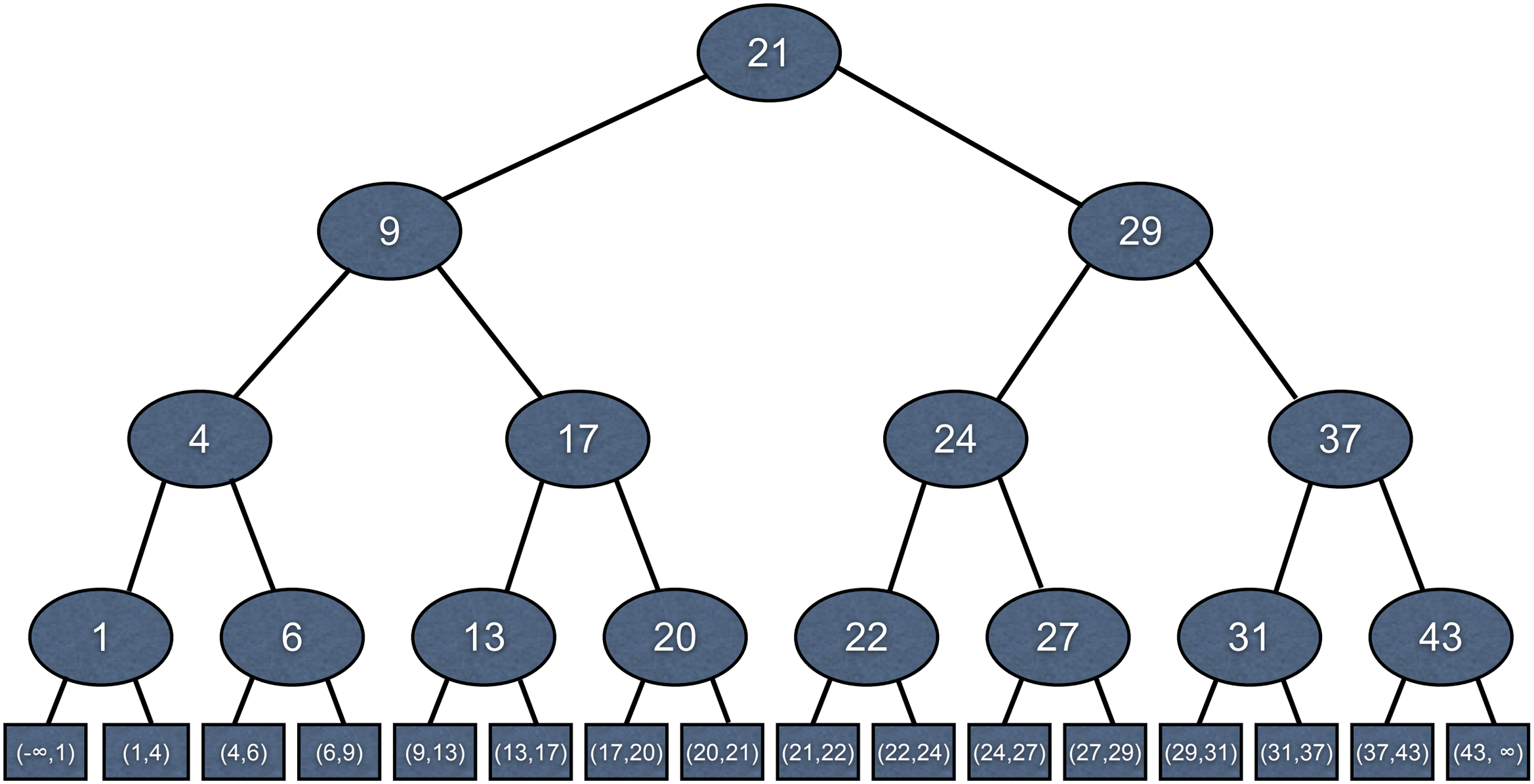
[2] 232-241

fázunk...

Bináris keresőfa



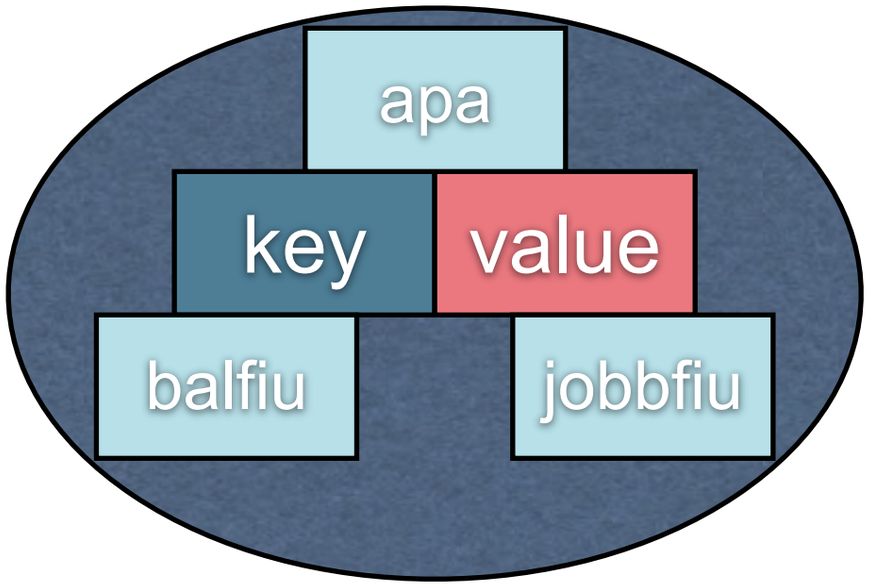
Bináris keresőfa



Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

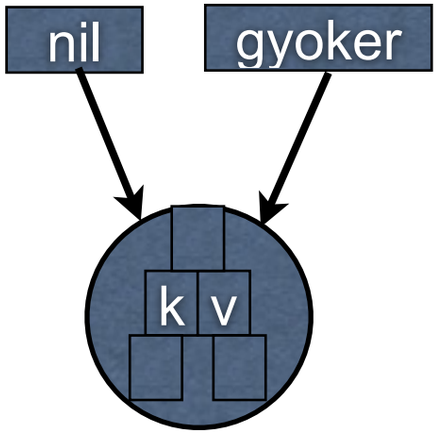
```
struct fapont{
    int key, value ;
    fapont bal, jobb, apa ;
}
```



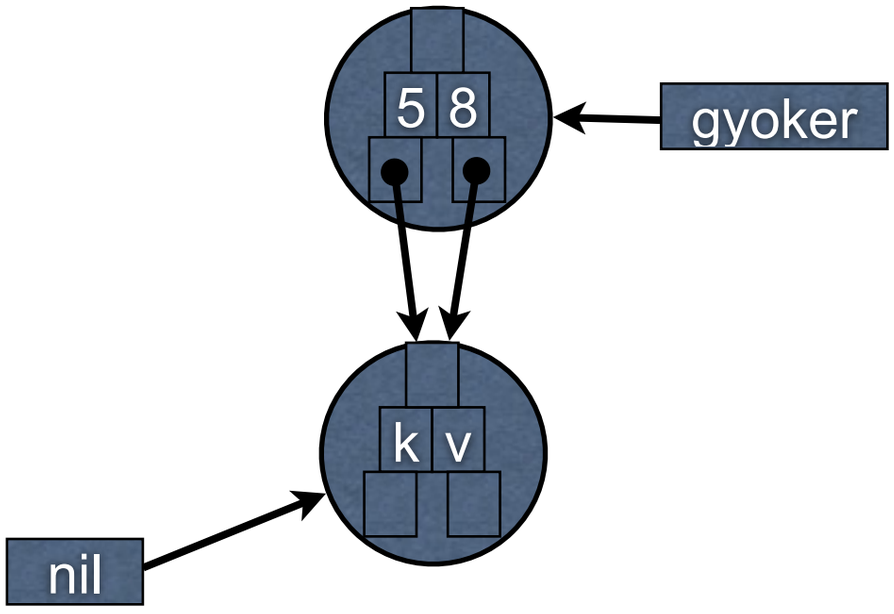
Bináris fa tárolása

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) {
   ...
  }
  ...
}
```

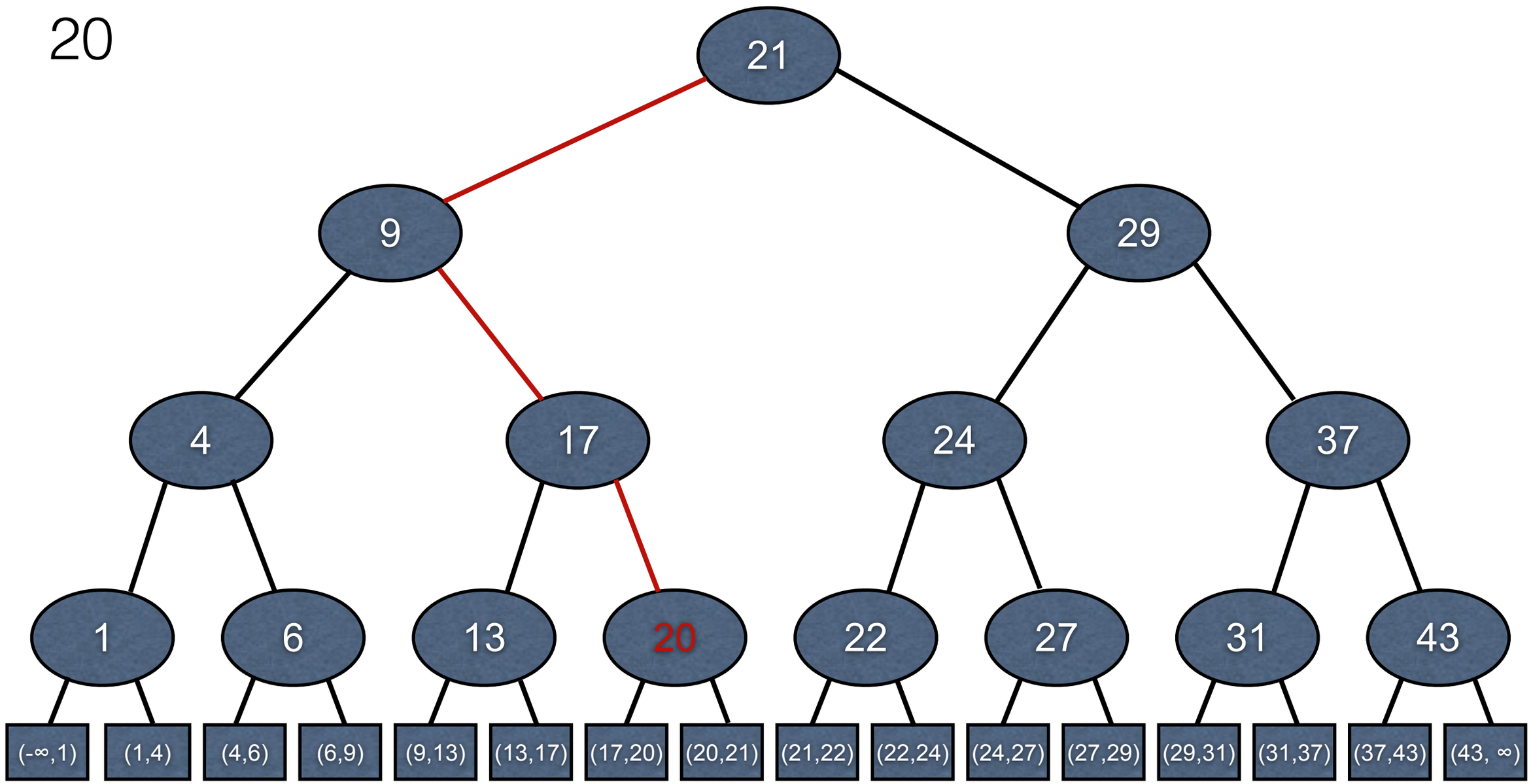


beszur(5, 8)

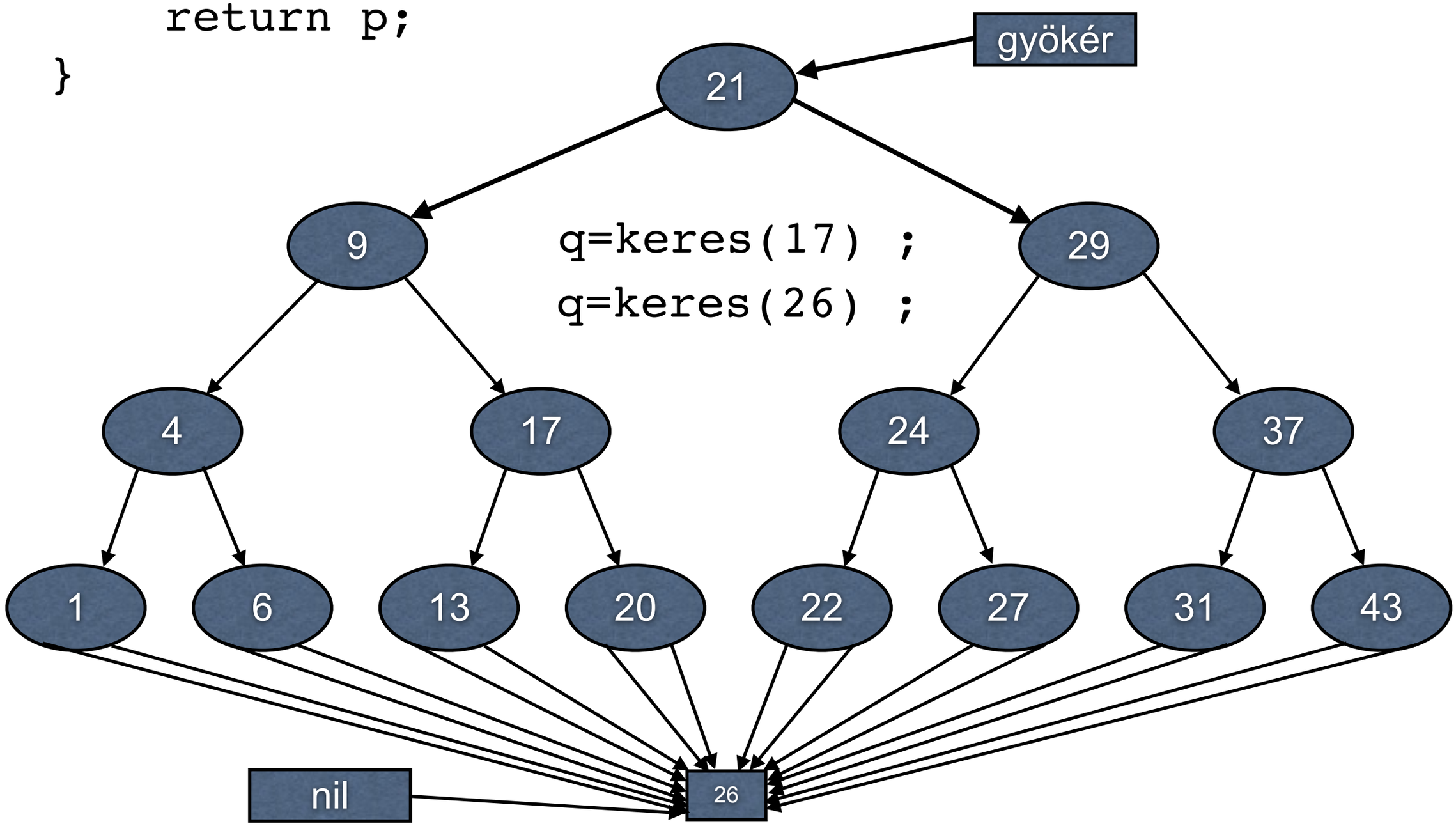


Keresés keresőfában

20

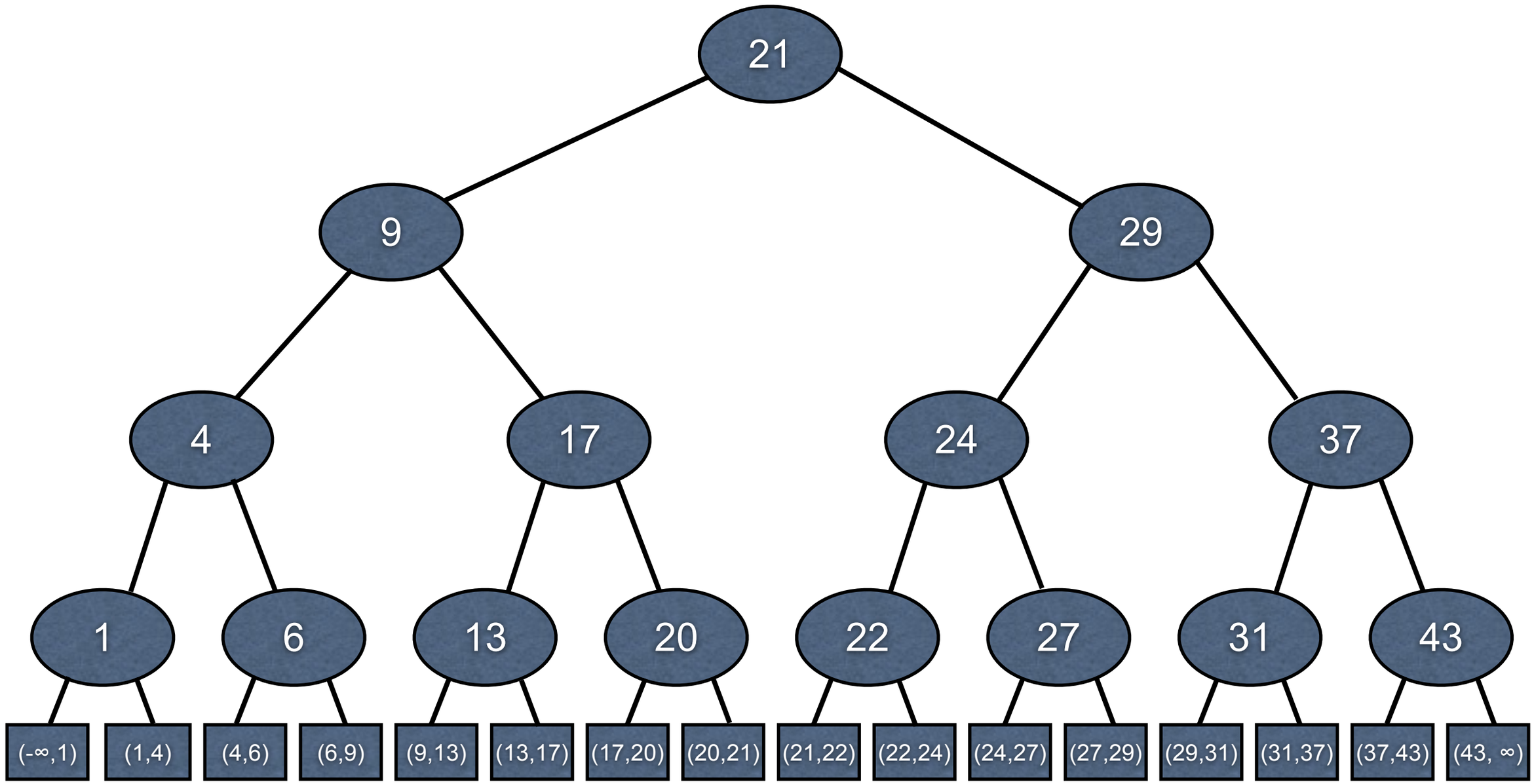


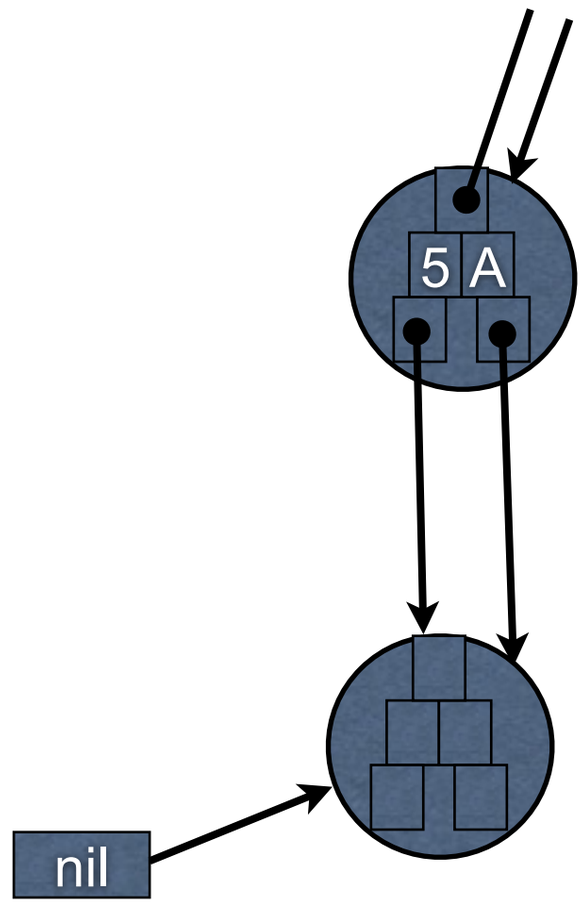
```
fapont keres(int x) {
    fapont p=gyoker ;
    nil.key=x ;
    while (p.key!=x) p=x<p.key?p.bal:p.jobb ;
    return p;
}
```

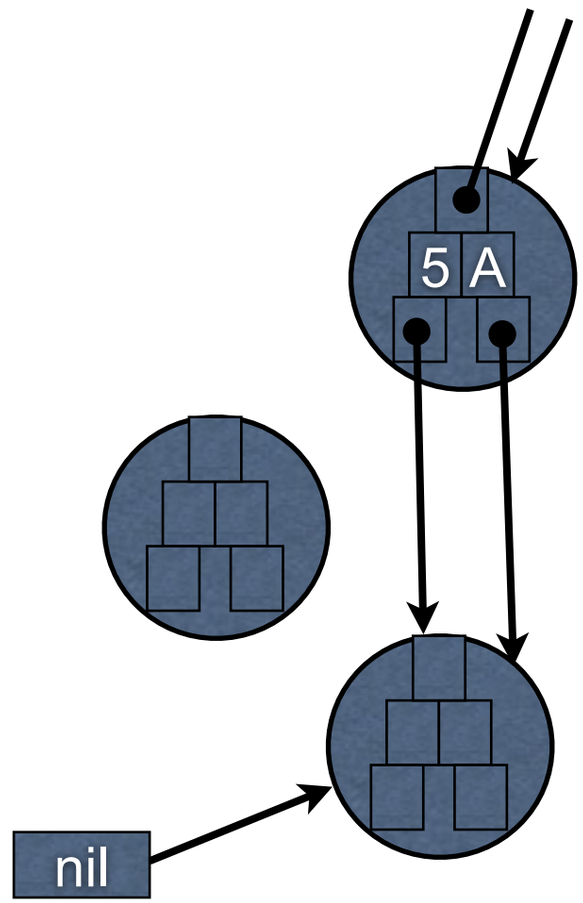


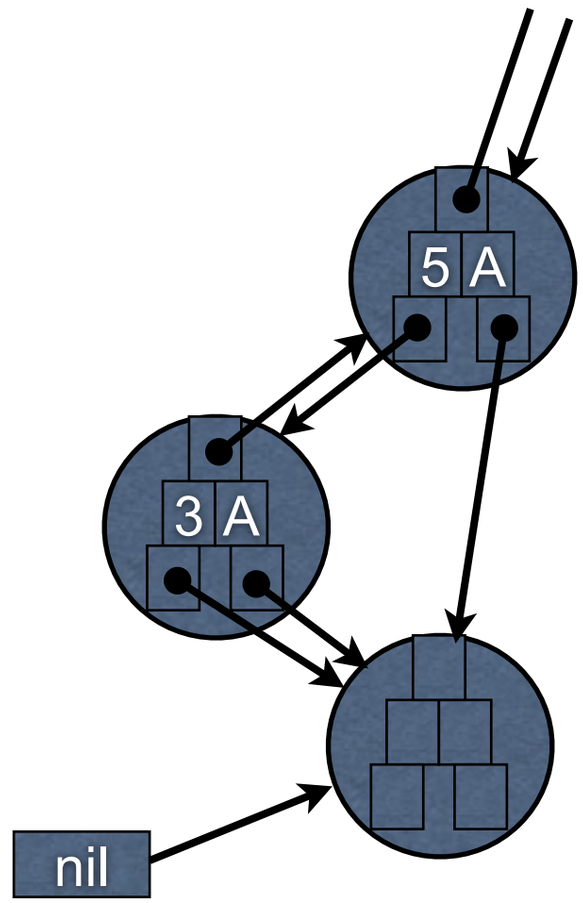

```
q=keres(17) ;
q=keres(26) ;
```

Beszúrás keresőfába

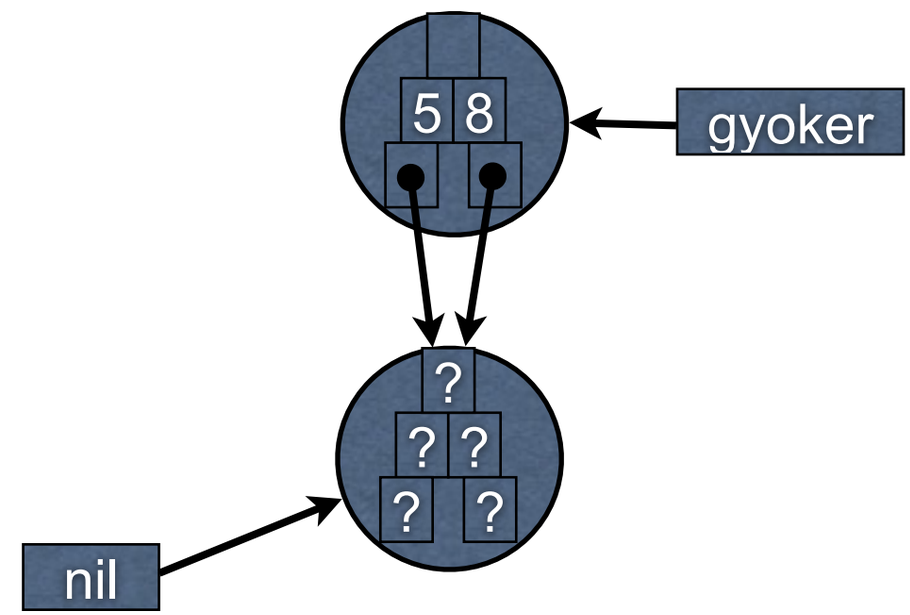



5
A
nil

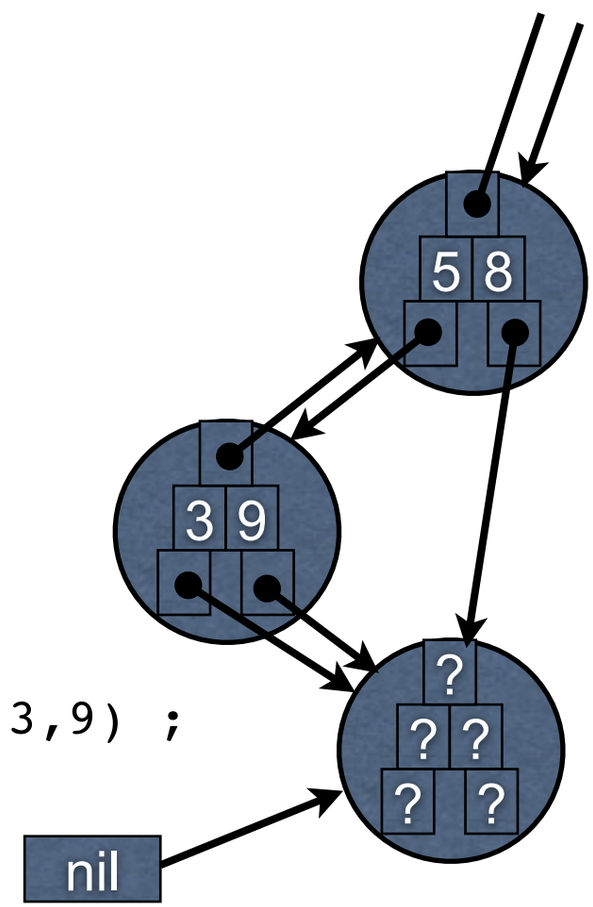

5
A
nil


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 ;
      }
   }
```

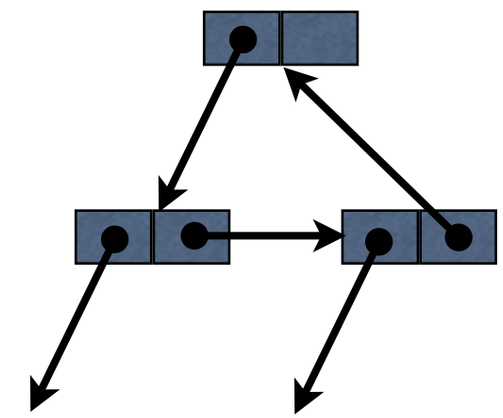


beszur(3,9) ;



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)

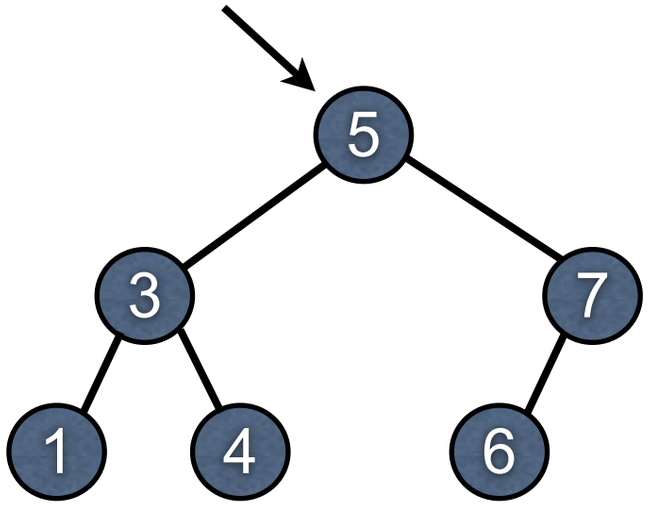
Bináris fa inorder bejárása

```
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)

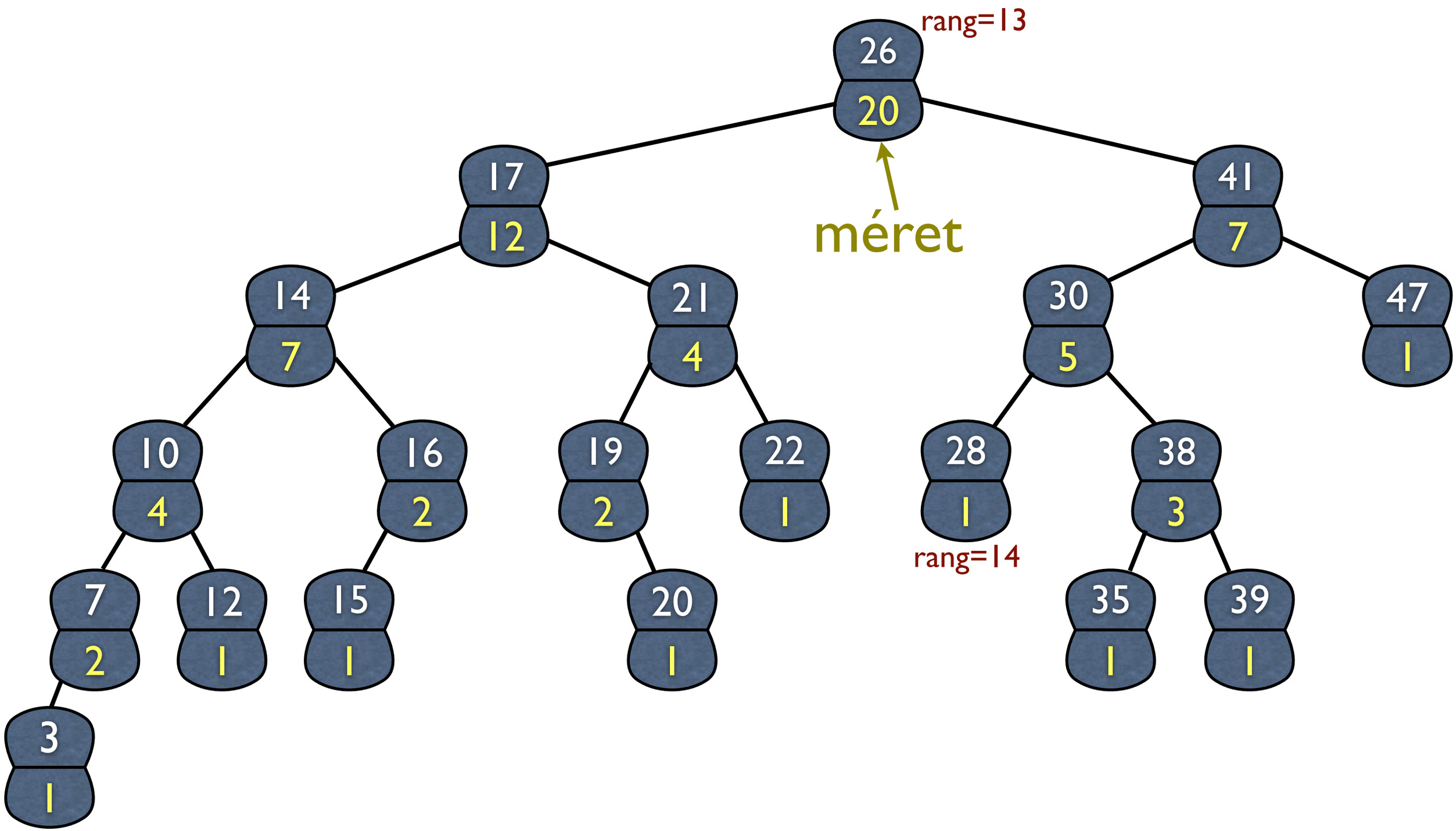
Bejárás - csökkenő sorrendben

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

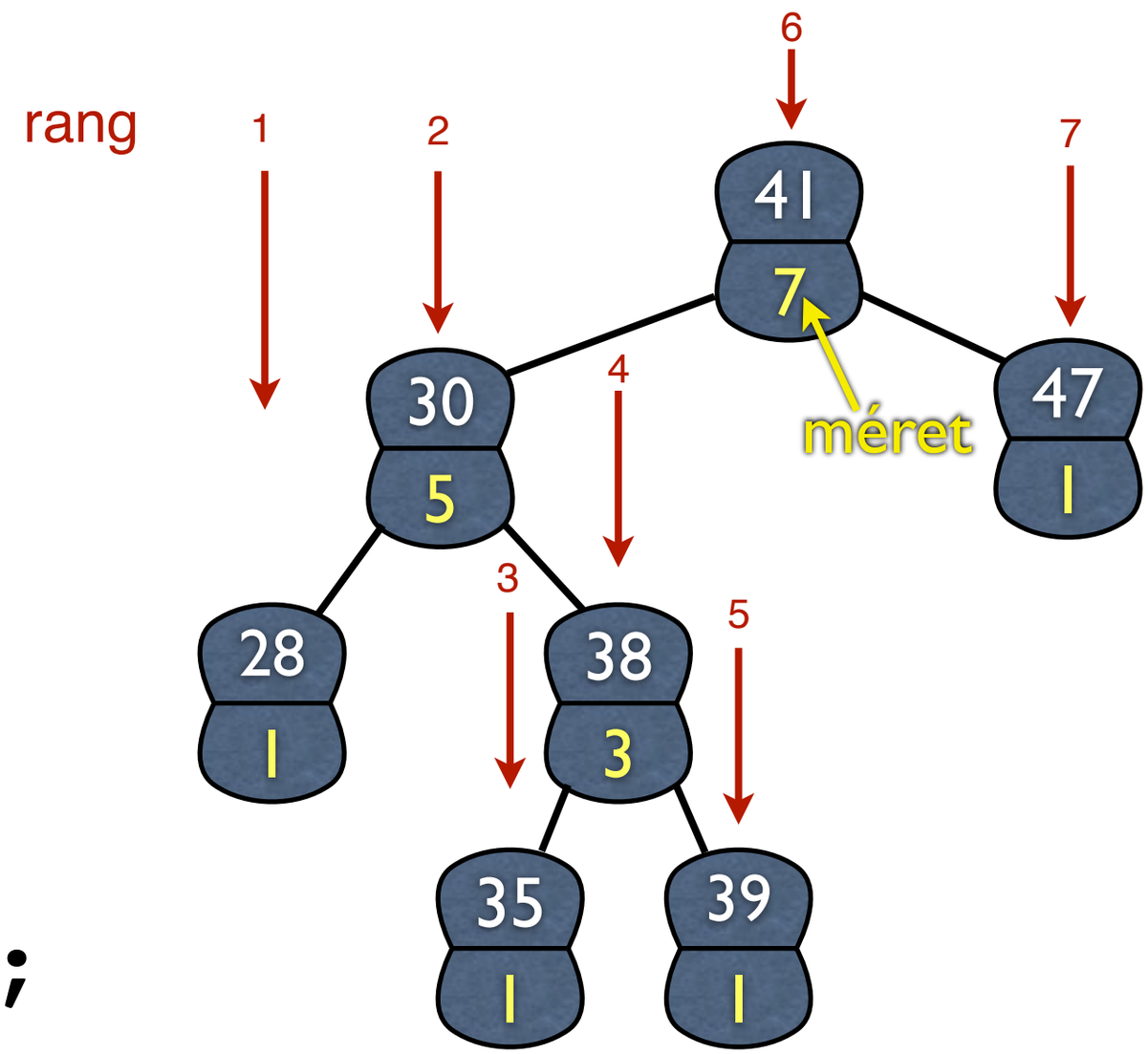
Rendezettminta-fa

- 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
rang=14
38
3
7
2
12
1
15
1
20
1
35
1
39
1
3
1

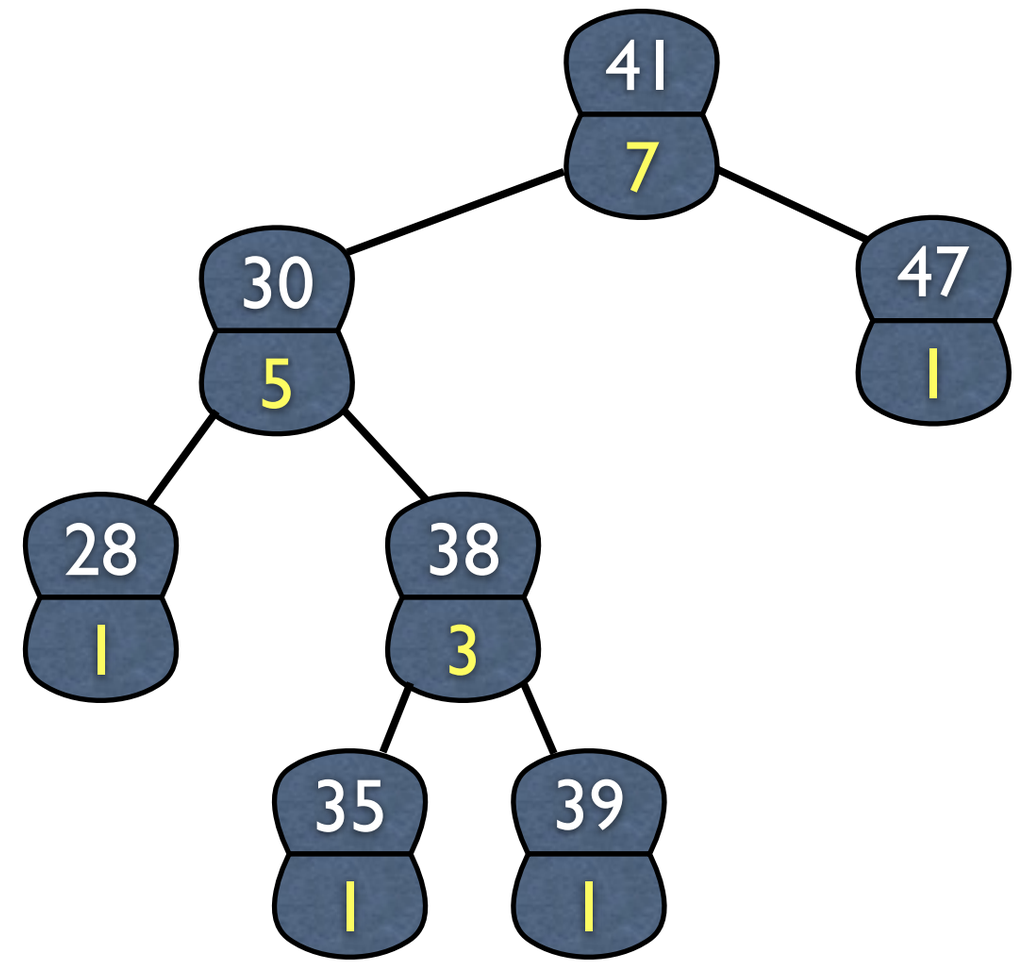
Adott rangú elem keresése




```
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



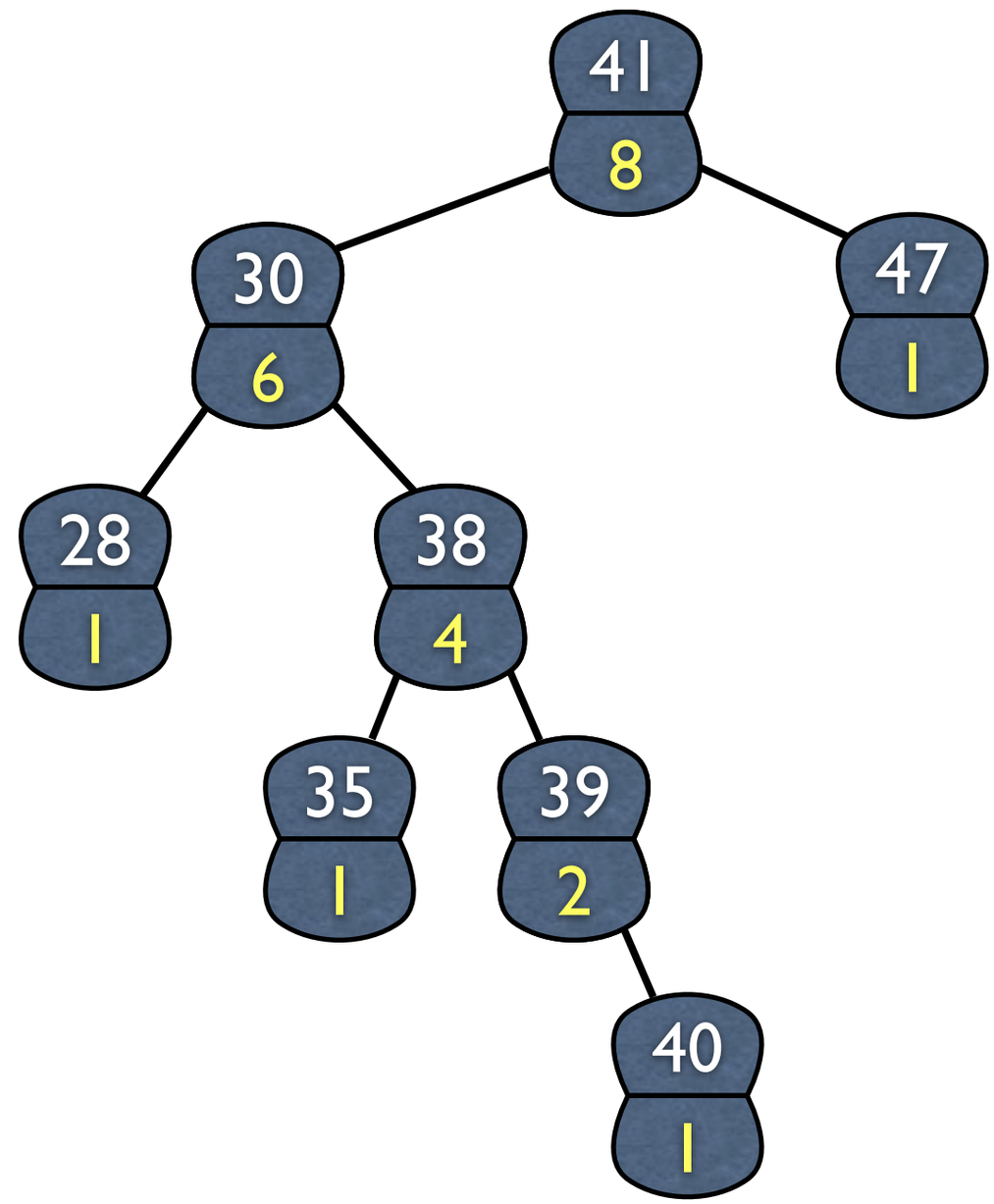

```
rmrang(p) {
  r=p.bal.meret+1 ;
  q=p ;
  while (q!=gyoker) {
    if (q==q.apa.jobb) r+=q.apa.bal.meret+1;
    q=q.apa ;
  }
  return r ;
}
```

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-- ;
  }
}
```

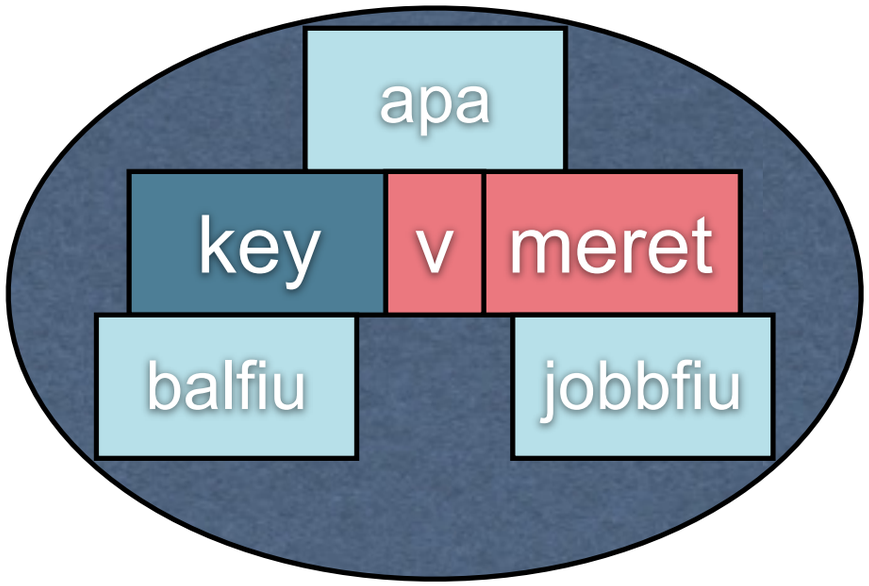


O(h)

Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

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

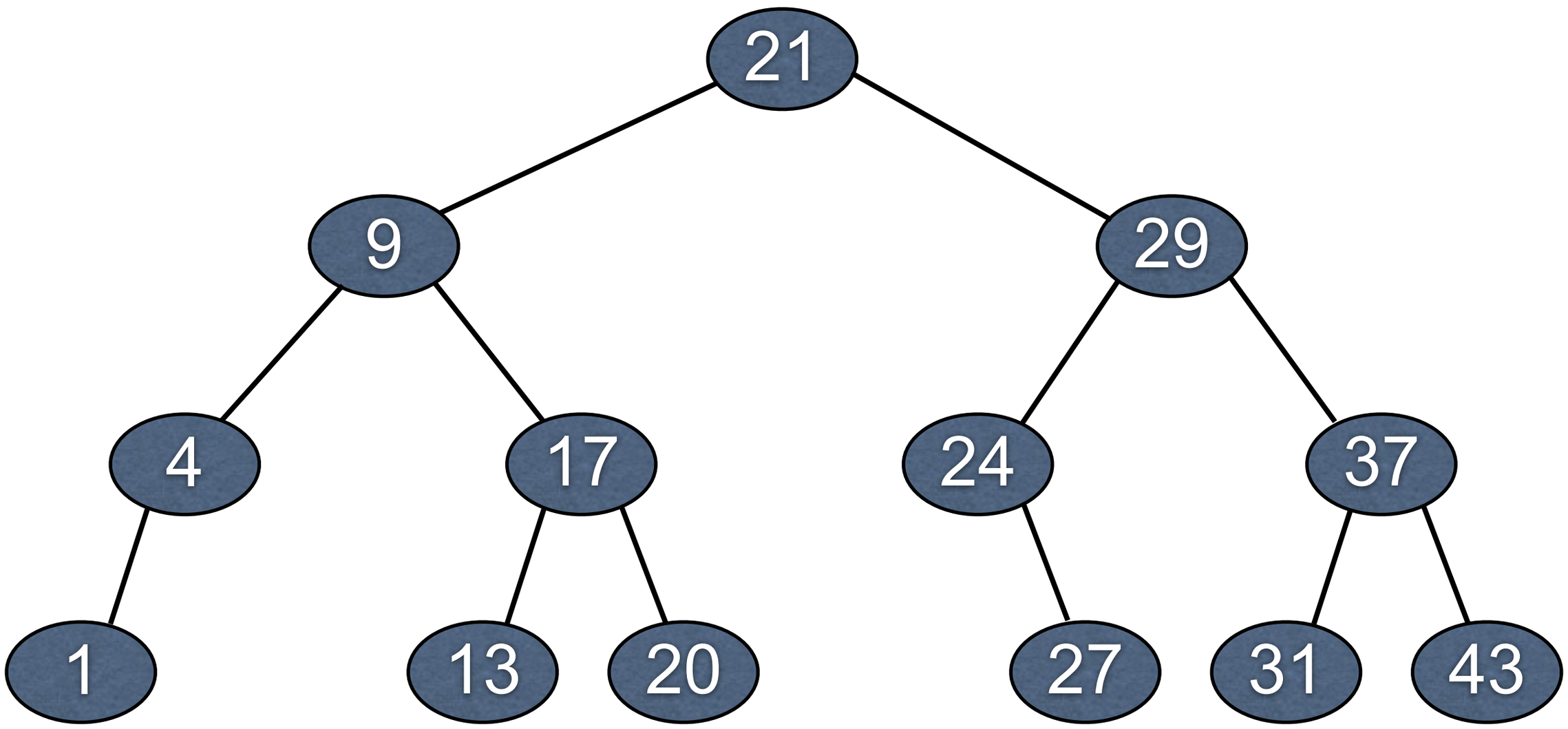


Rendezettmintafa

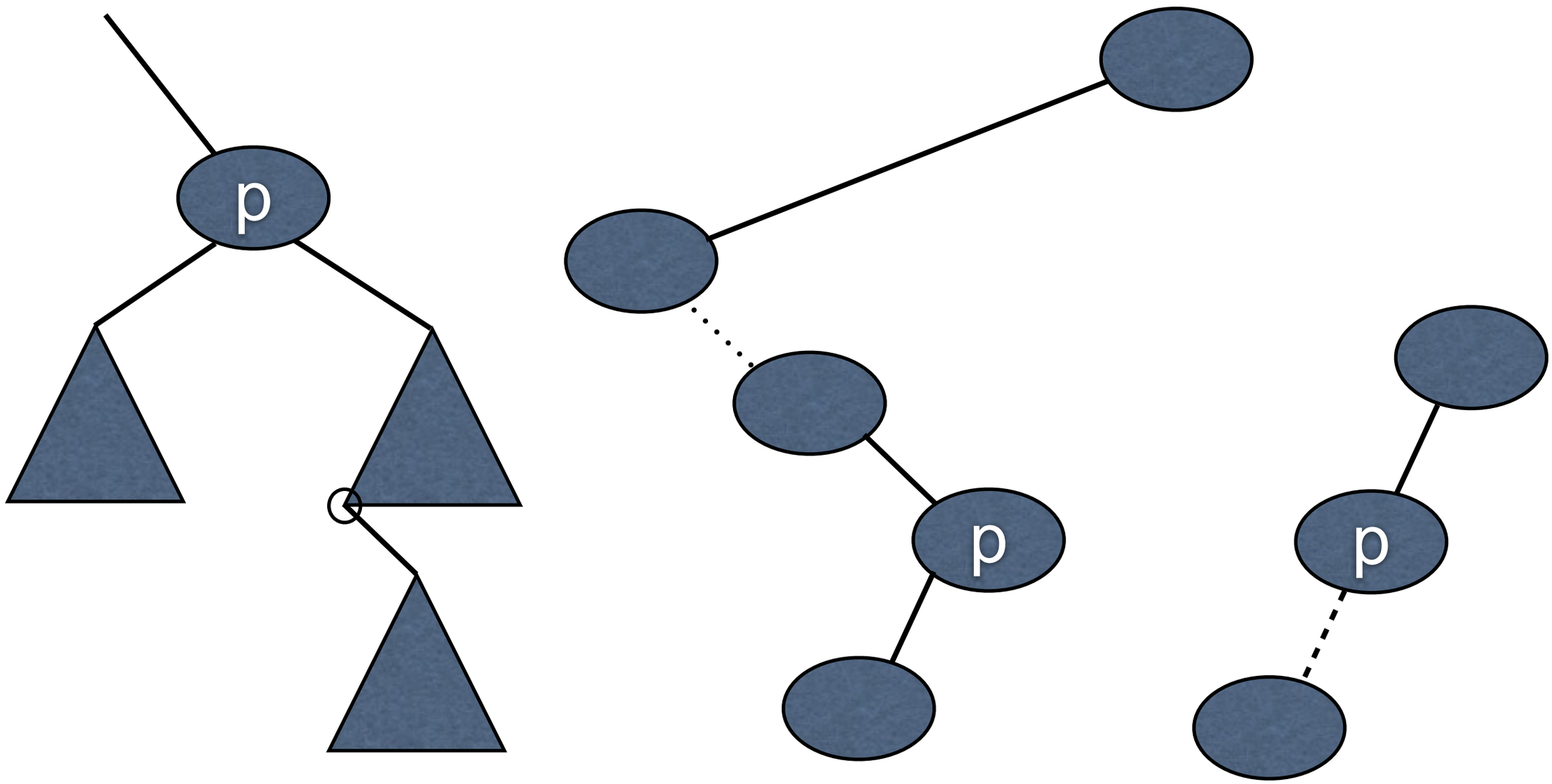
Műveletek:

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*)

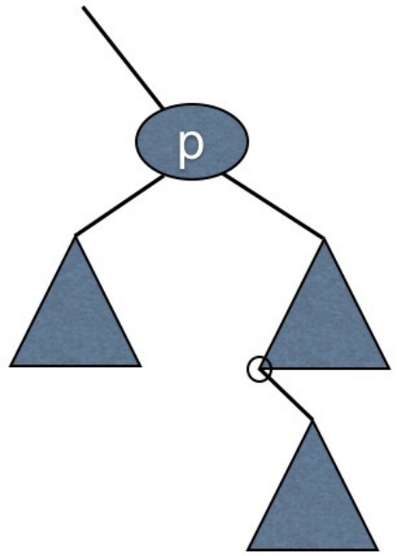
Elem rákövetkezője



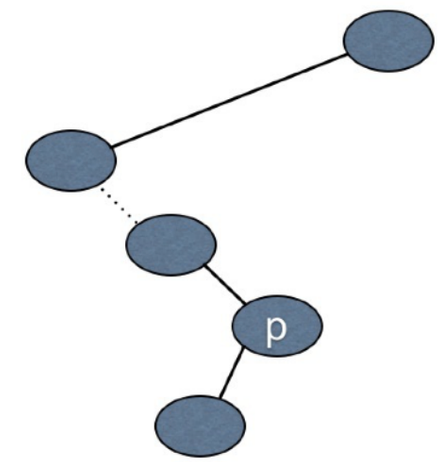
p rákövetkezője



O(h)



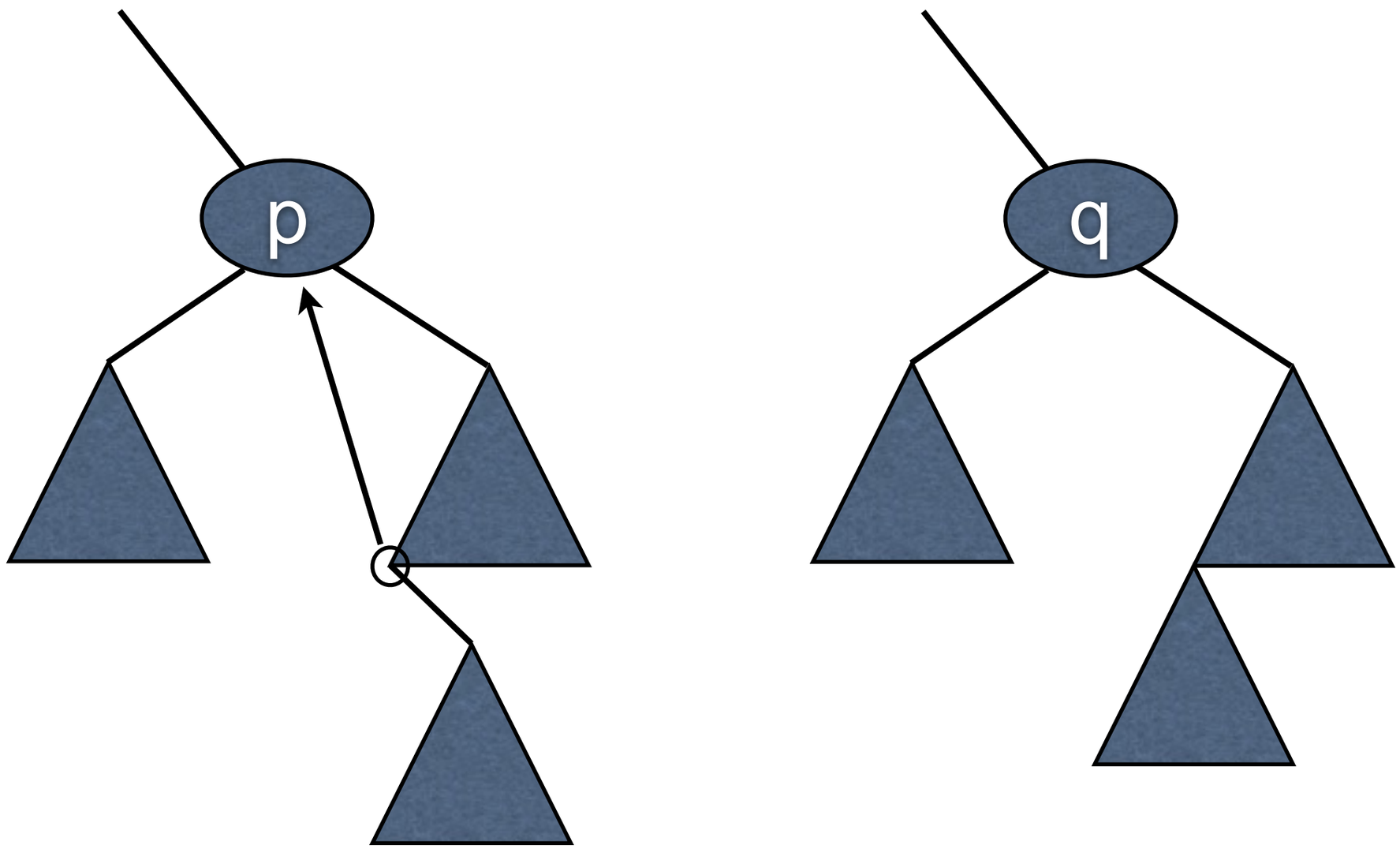
p rákövetkezője




```
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
    }
}
```

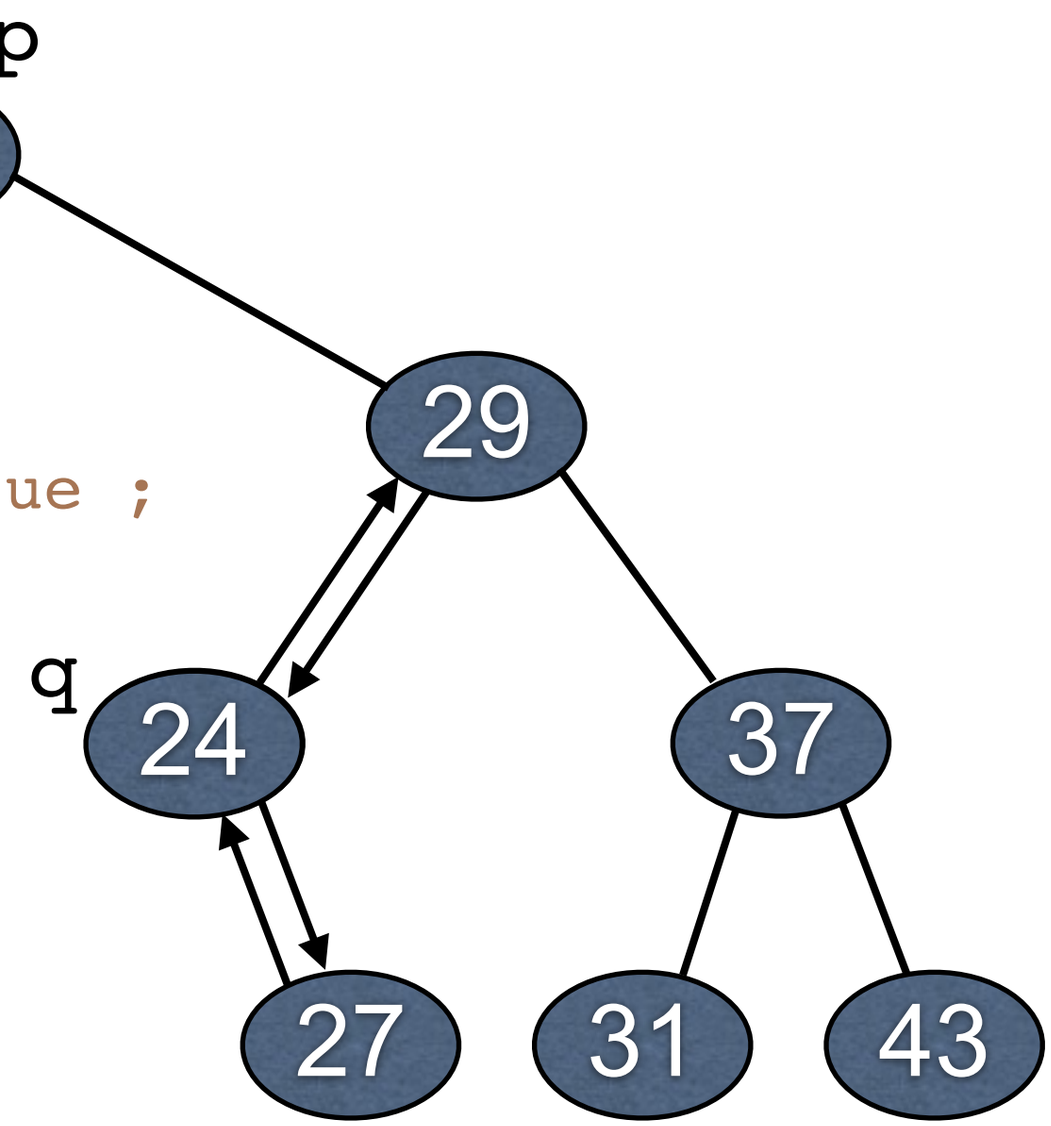
O(h)

Törlés bináris keresőfából



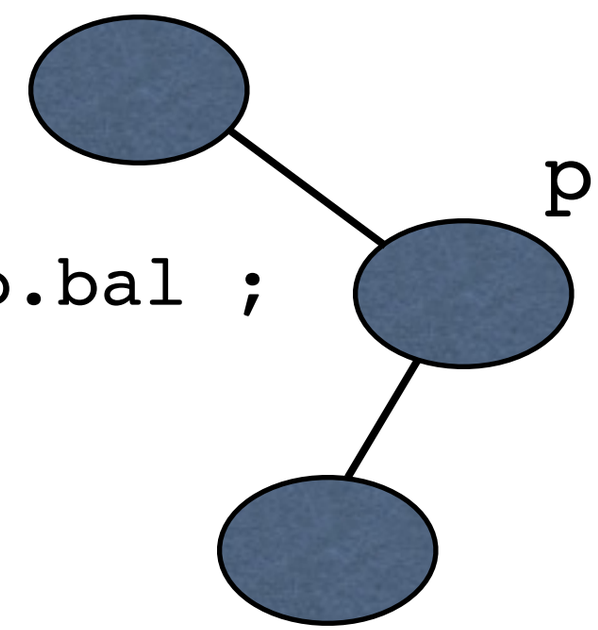
O(h)

```
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 {
     ...
  }
}
```



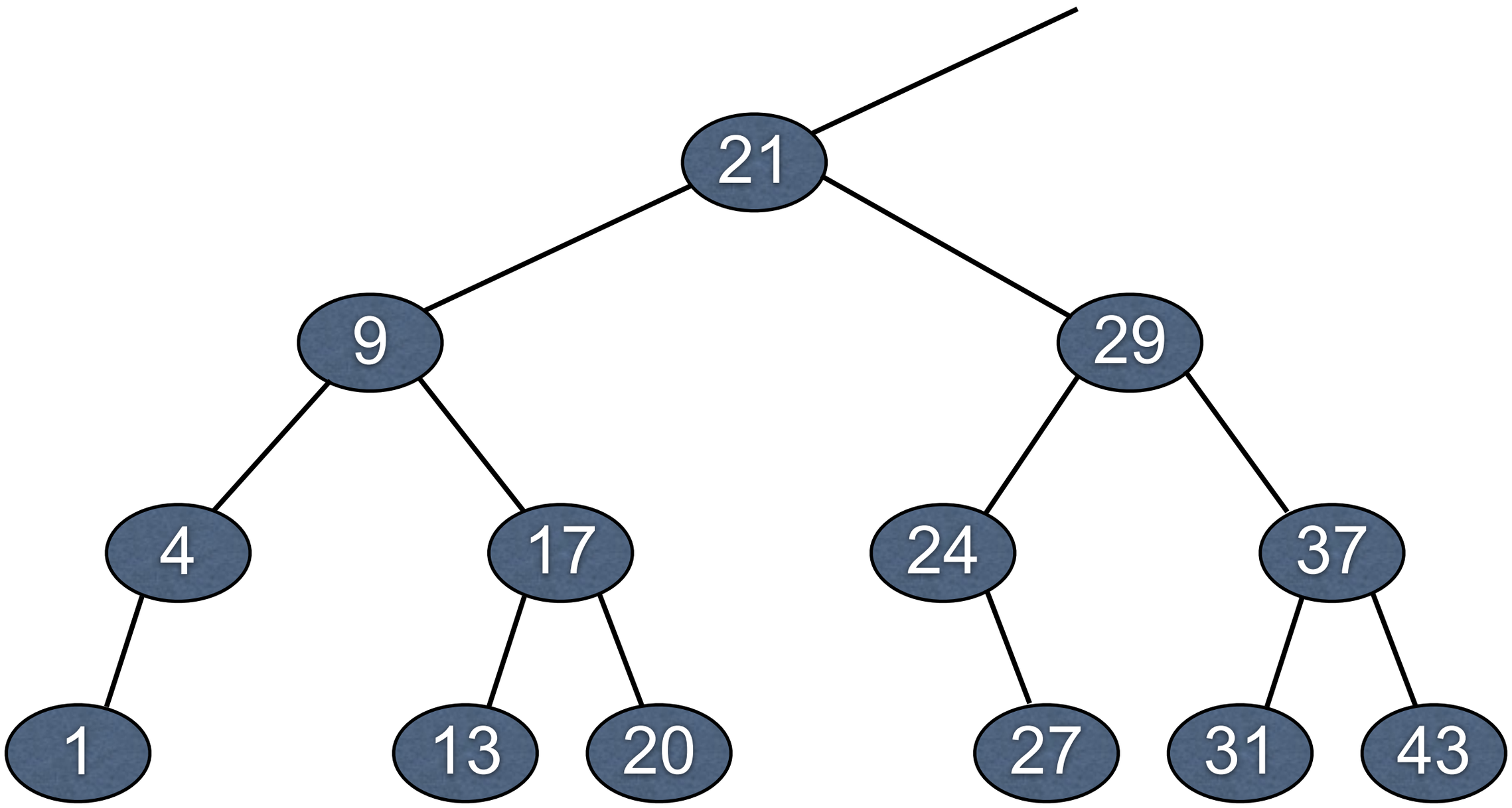
O(h)

```
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

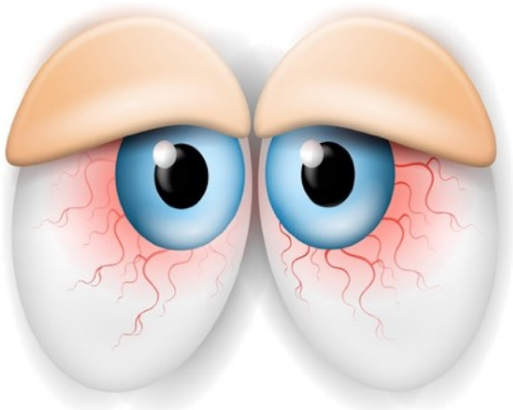


Futási idők elemzése

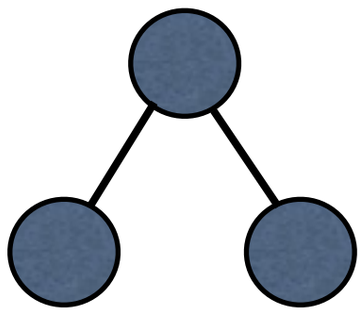
- Keresés
- Beszúrás
- Törlés

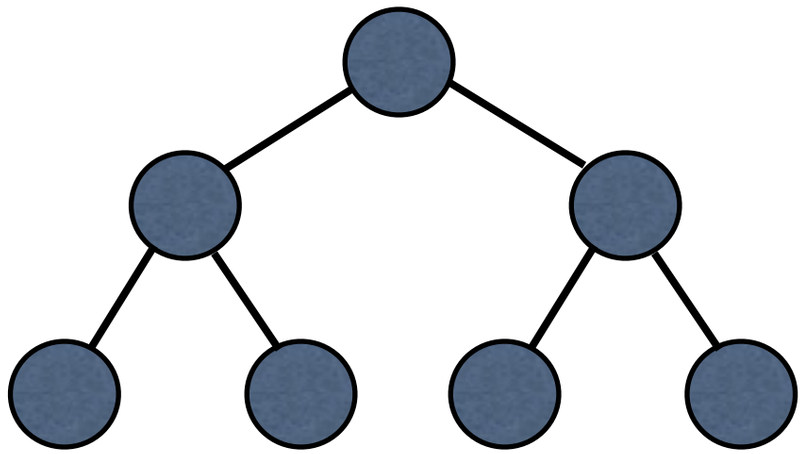
Mennyi a futási idő?

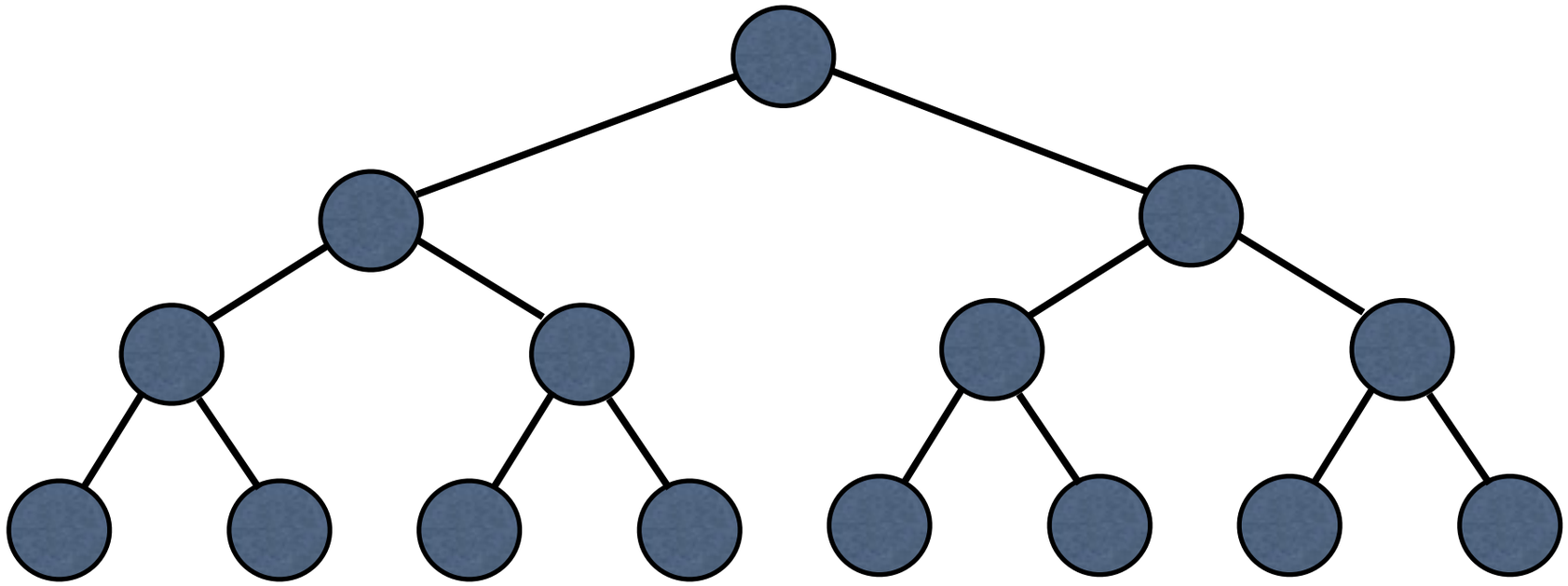
$O(h)$



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

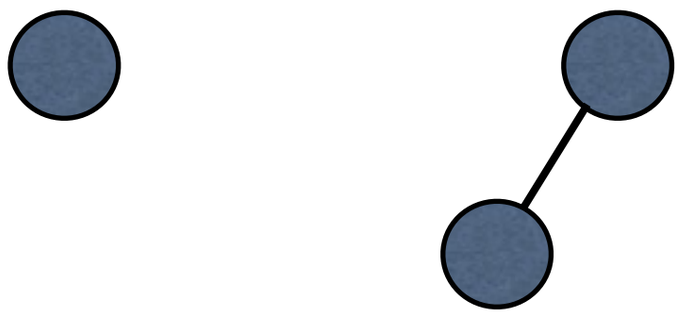
$$n = 2^h - 1$$

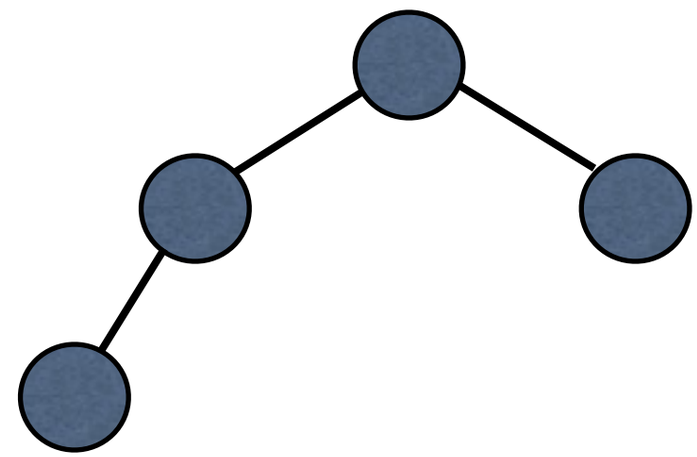
$$n + 1 = 2^h$$

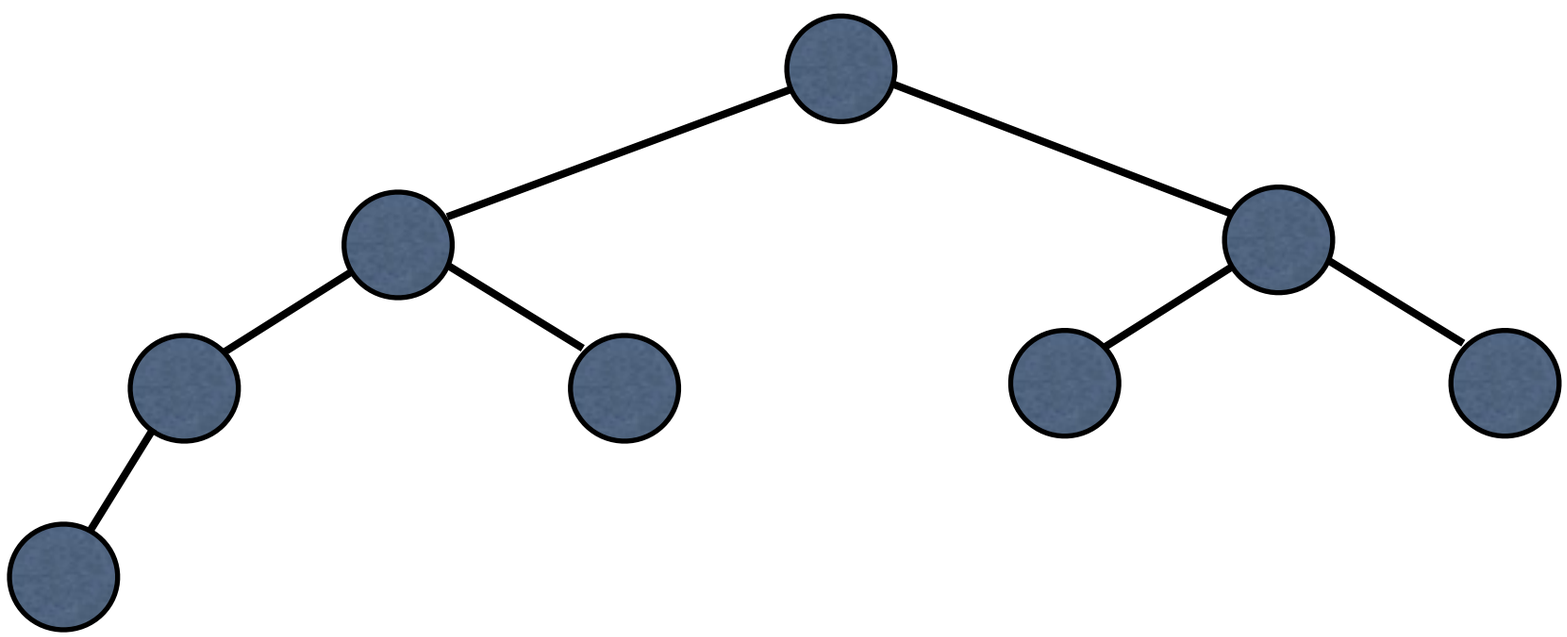
$$\log_2(n+1) = h$$

$$h = O(logn)$$

Majdnem teljes bináris keresőfa

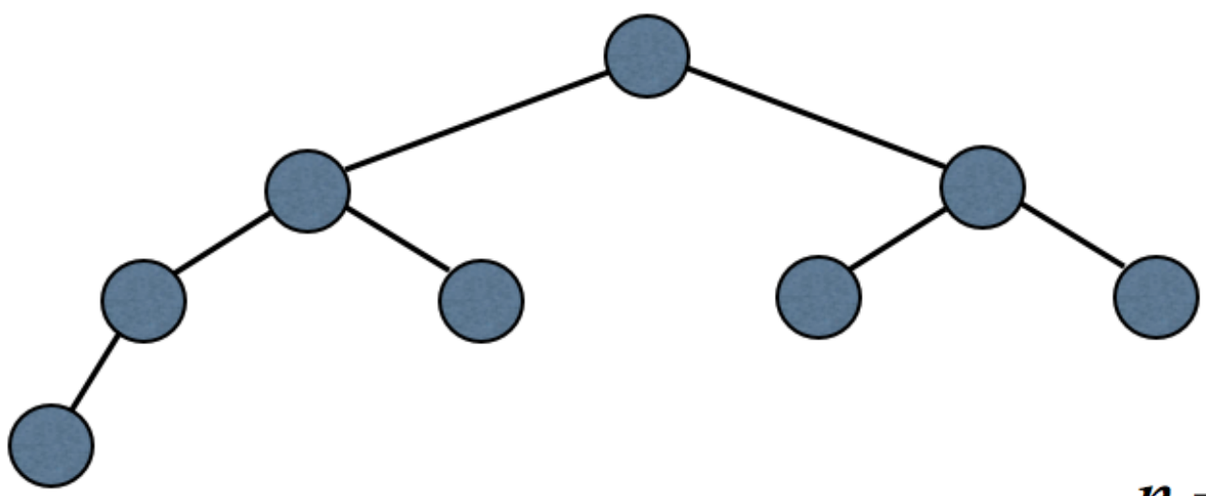






h	n(h)	2^{h-1}
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
k	2^{k-1}	2^{k-1}

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}

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

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

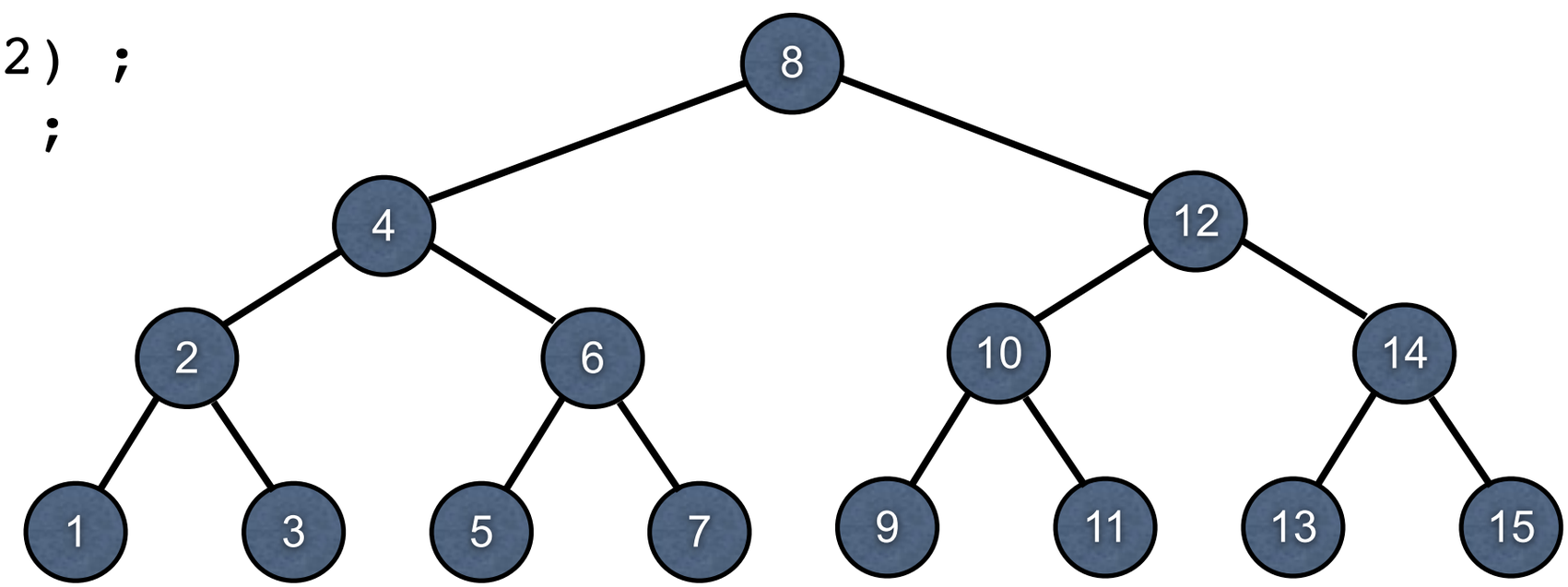
$$\log_2 n + 1 = h$$

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

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

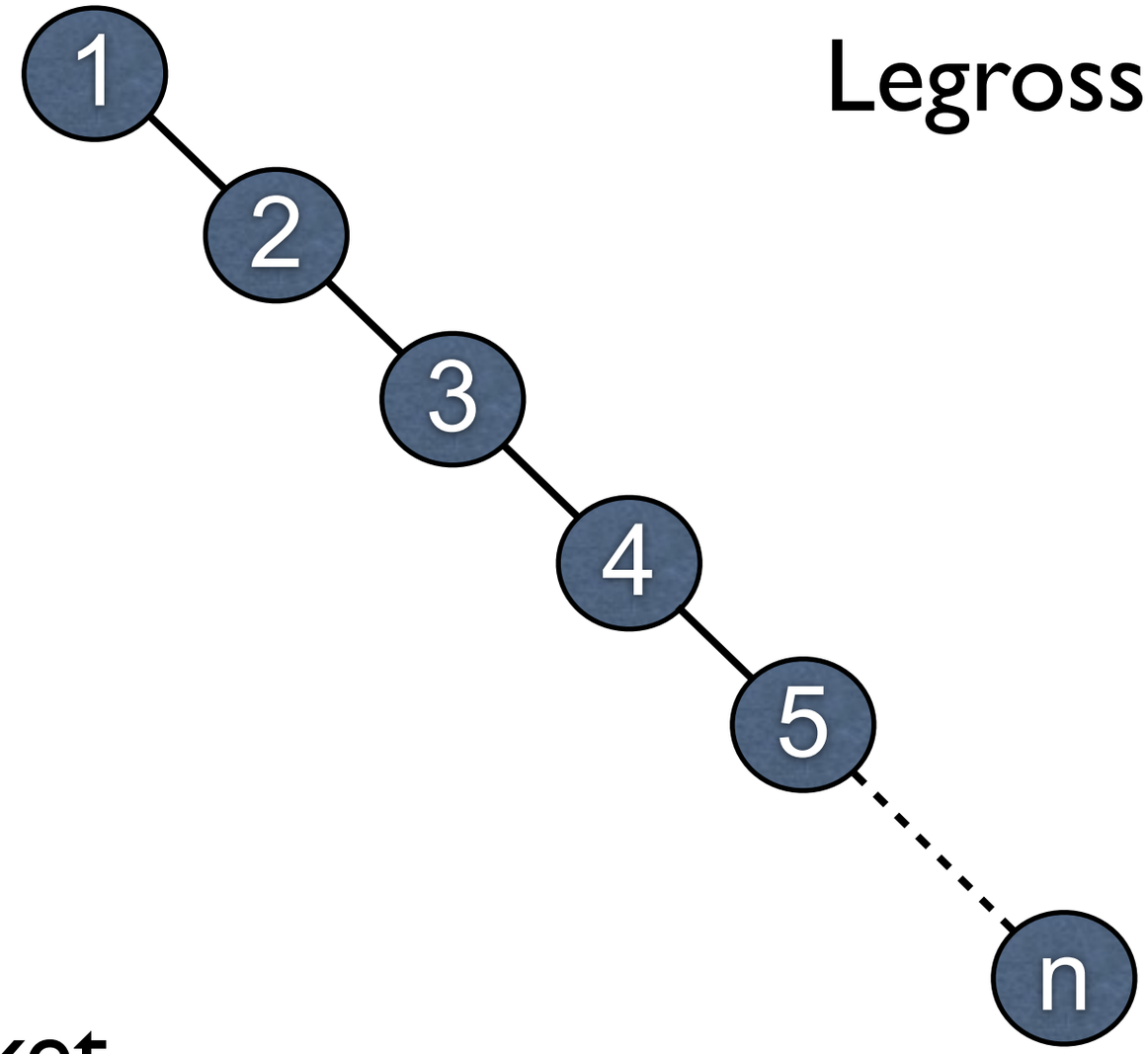
Véletlen építésű bináris keresőfa

- 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



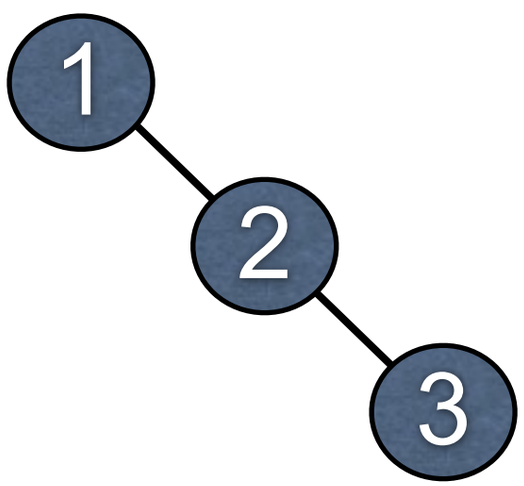
Legrosszabb eset

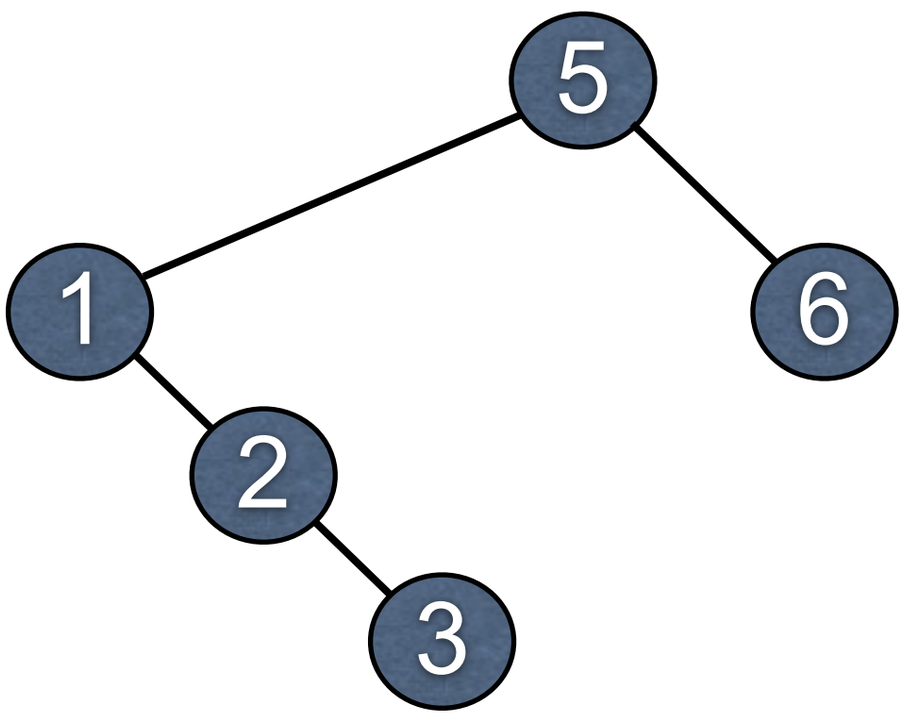
h=n

Hogy lehetne javítani?

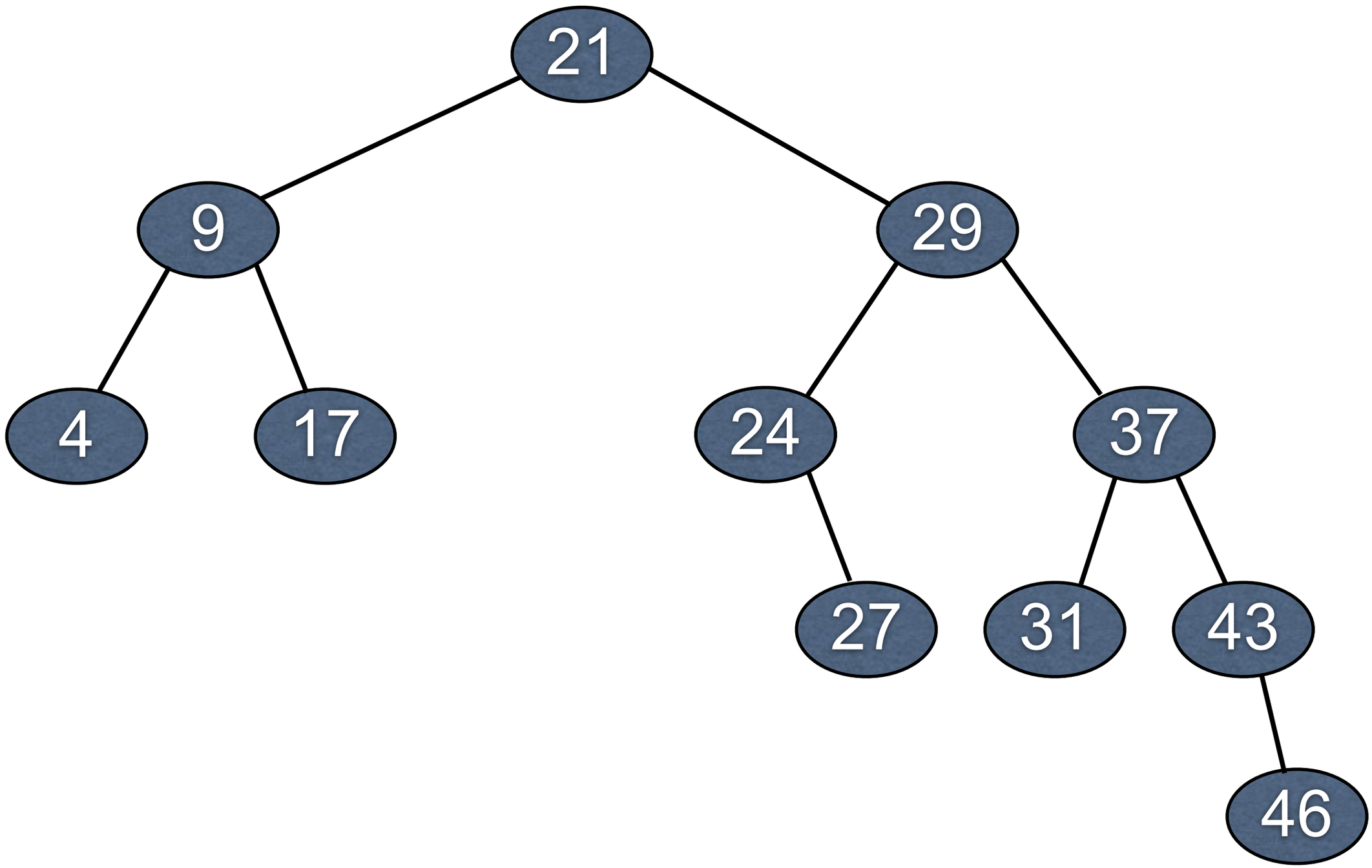
Mivel az elemek egyesével jönnek

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





Példa



AVL fa

Az **AVL-fa** alatt egy ön-kiegyensúlyozó bináris keresőfát értünk.

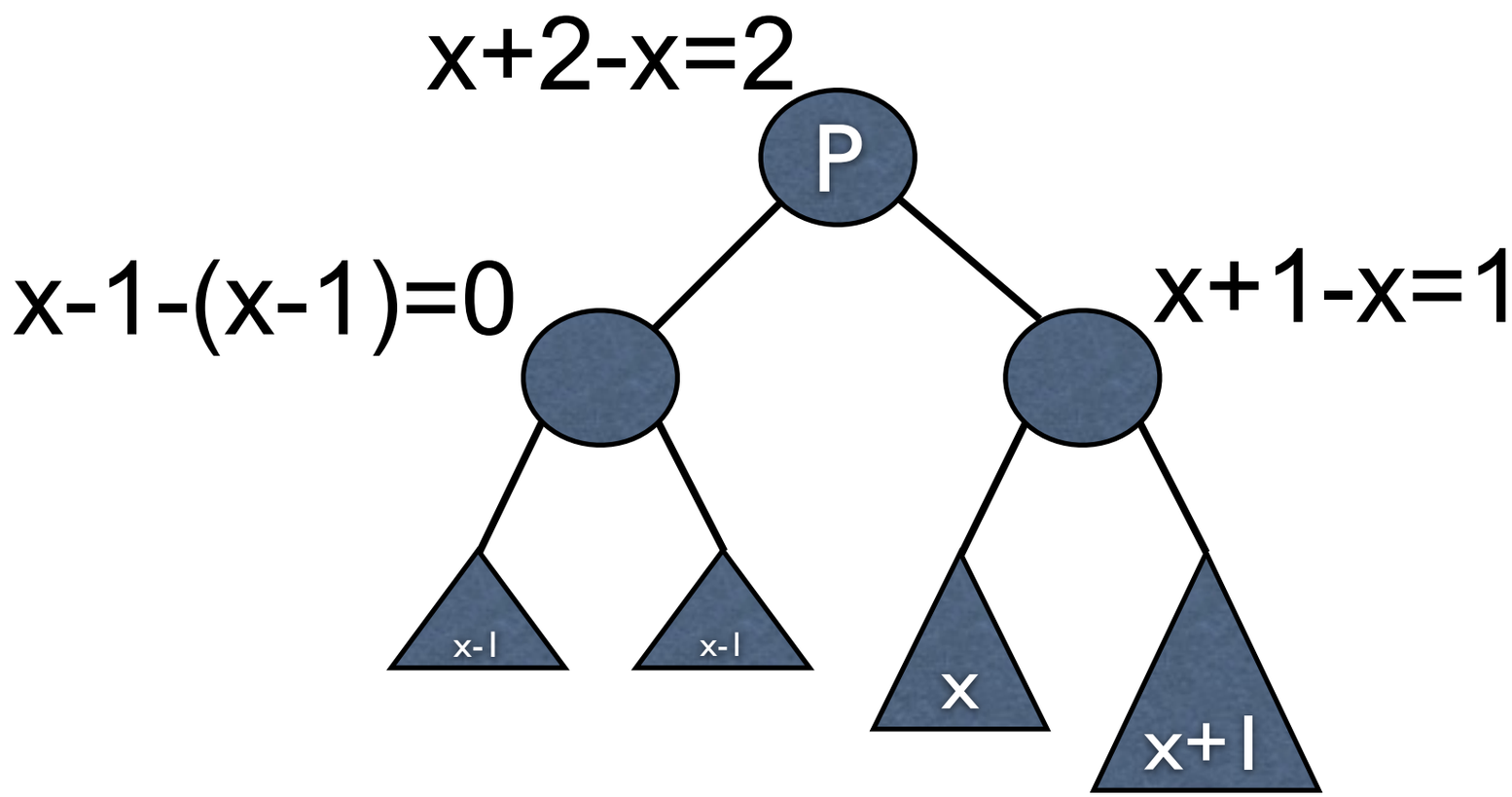
Egy AVL-fában bármely csúcspont két részfájának magassága közti különbség legfeljebb egy.

Az AVL-fa nevét két feltalálójáról G. M. Adelson-Velsky-ről és E. M. Landis -ról kapta, akik 1963 -ban publikálták [4] *An algorithm for the organization of information (Egy algoritmus az információ szervezettségéhez)* című cikkükben.

[3]

p pont egyensúlyfaktora

egy(p)=h(p.jobb)-h(p.bal)

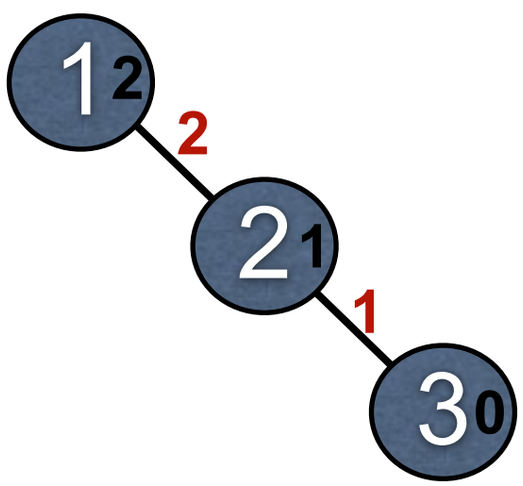


[1] 278

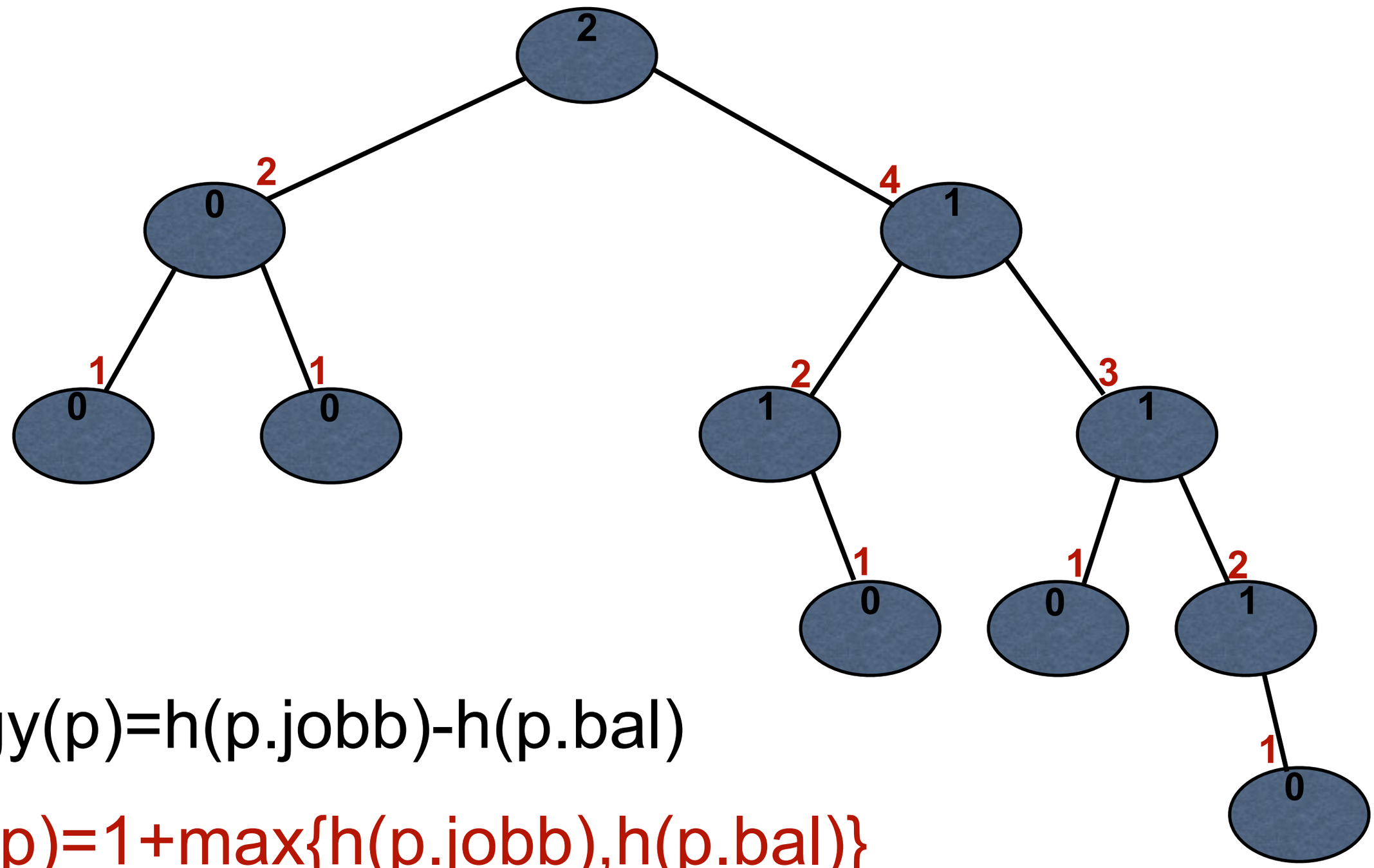
Példa

h(p)=1+max{h(p.jobb),h(p.bal)}

egy(p)=h(p.jobb)-h(p.bal)



Példa



egy(p)=h(p.jobb)-h(p.bal)

h(p)=1+max{h(p.jobb),h(p.bal)}

AVL fa

Def A $P \in F$ pont (magasság-)egyensúlya:

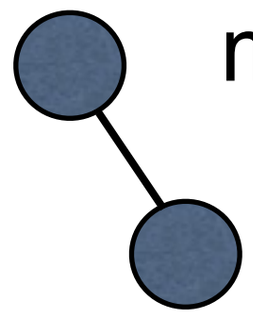
$$Egy(P) = h(Job(P)) - h(Bal(P))$$

Def Az F binfa AVL-fa, ha $(\forall P \in F)(-1 \leq Egy(P) \leq 1)$

tétel Ha F AVL-fa, akkor $h(F) \leq 1.44 \cdot lg(n+1)$, ahol n az F fa pontjainak számát jelöli.

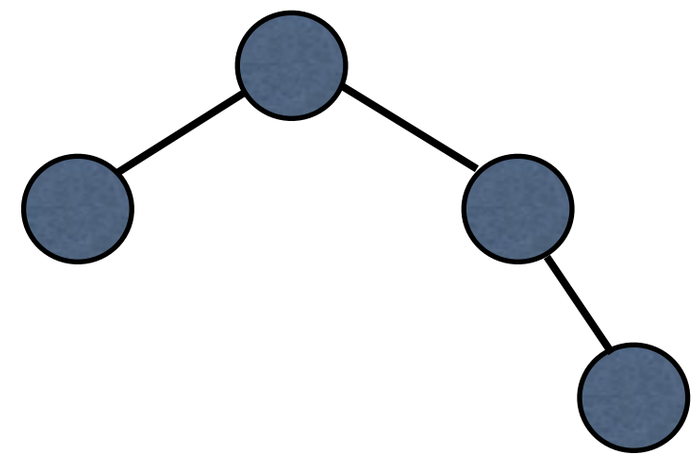
Legkevesebb pontot tartalmazó h magasságú AVL fa

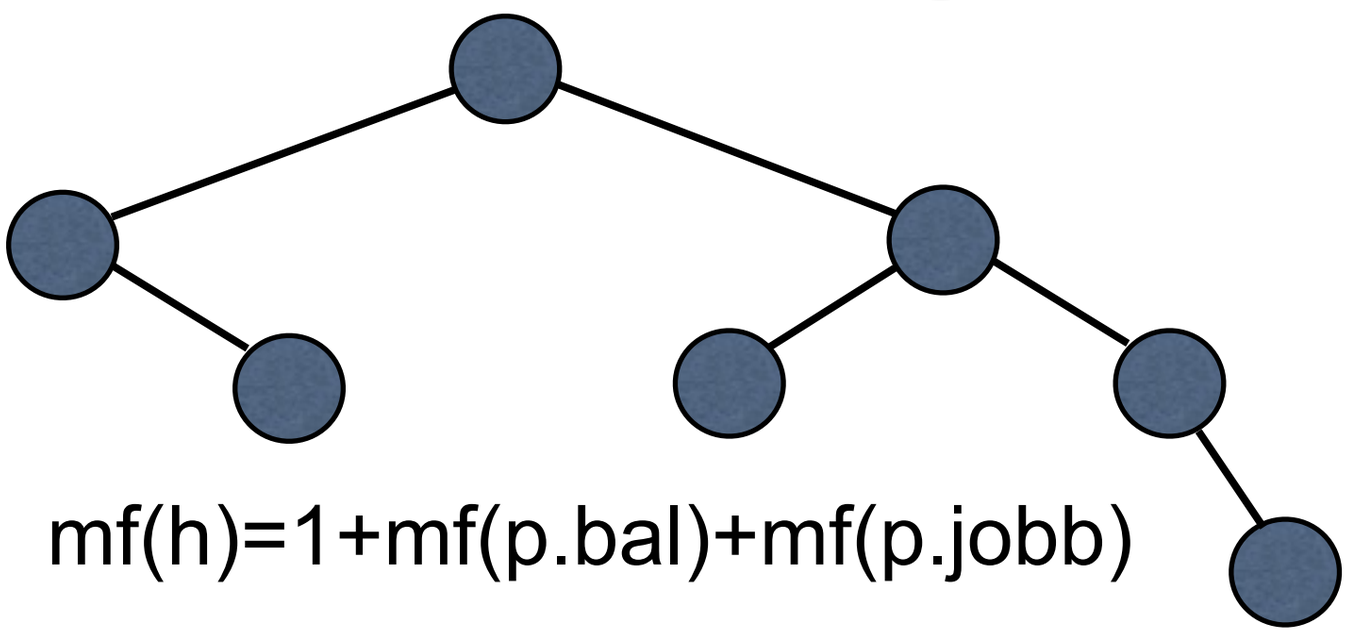
mf(1)=1



mf(2)=2

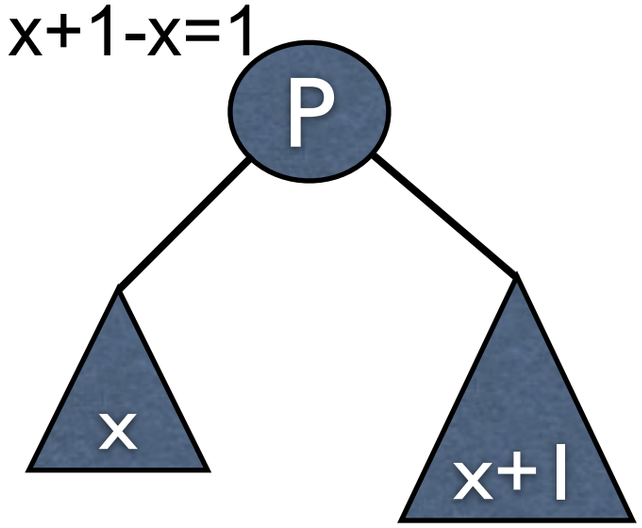
mf(3)=1+mf(1)+mf(2)=4





mf(h)=1+mf(p.bal)+mf(p.jobb)

x+1-x=1



mf(h)=fib(h)+1

fib(h)=fib(h-2)+fib(h-1)

[1] 276

Def A $P \in F$ pont (magasság-)egyensúlya:

$$Egy(P) = h(Job(P)) - h(Bal(P))$$

Def Az F binfa AVL-fa, ha $(\forall P \in F)(-1 \leq Egy(P) \leq 1)$
tétel Ha F AVL-fa, akkor $h(F) \leq 1.44 \cdot lg(n+1)$, ahol n az F fa pontjainak számát jelöli.

Biz Legyen N_m az m magasságú, legkevesebb pontot tartalmazó AVL-fa pontjainak száma.
Ekkor $N_m \leq |F|$, ha F AVL-fa és $h(F) = m$.
$N_0 = 0$, $N_1 = 1$, $N_2 = 2$ és $N_m = 1 + N_{m-2} + N_{m-1}$, ha m > 1.
Legyen $B_i = N_i + 1$. Ekkor $B_0 = 1$,$B_1 = 2$ és $B_m = B_{m-2} + B_{m-1}$ ha (m > 1).
lemma $\Phi^m \leq B_m$, ahol $\Phi = (1+\sqrt{5})/2$.
$1 = \Phi^0 \leq B_0$, $\Phi \leq B_1$. Teljes indukció alapján, ha $2, \ldots, m-1$-re áll az egyenlőtlenség
$B_m = B_{m-2} + B_{m-1} \leq \Phi^{m-2} + \Phi^{m-1} = \Phi^{m-2}(1+\Phi)$.
Viszont $(1+\Phi) = \Phi^2$, így a lemmát igazoltuk.

Tehát $\Phi^m \leq B_m = N_m + 1 \leq n+1$, azaz
$m \cdot \log \Phi \leq \log(n+1)$.
Következésképpen $h(F) = m \leq \frac{1}{\log \Phi} \log(n+1) = 1.44 \cdot \log(n+1)$. [2]

□

Műveletek

Mind a beszúrás mind pedig a törlés esetén az algoritmus elsőként a bináris keresőfáknál tanultaknak megfelelően beszúrja illetve törli az elemet, majd a beszúráshoz illetve törléshez tartozó bővítőúton illetve törlőúton felfele haladva lokális forgatásokkal helyreállítja a tulajdonságot.

Def $U = [P_0, \ldots, P_m]$ X-bővítőút, ha

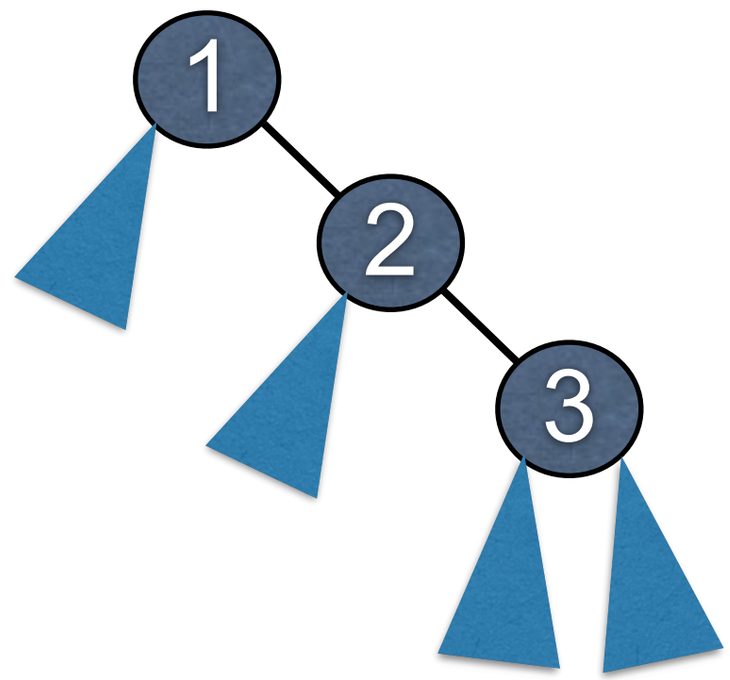
- ha $X < Adat(P_i)$ akkor $P_{i+1} = Bal(P_i)$ és ha $X > Adat(P_i)$ akkor $P_{i+1} = Jobb(P_i)$ $i < m$ esetén
- ha $X < Adat(P_m)$ akkor $Bal(P_m) = Nil$ és ha $X > Adat(P_m)$ akkor $Jobb(P_m) = Nil$

Def $U = [P_0, \ldots, P_m]$ X-törlőút, ha

- $\exists 0 \leq t \leq m$ amelyre $Adat(P_t) = X$ és ha $X < Adat(P_i)$ akkor $P_{i+1} = Bal(P_i)$ és ha $X > Adat(P_i)$ akkor $P_{i+1} = Jobb(P_i)$ $i < t$ esetén
- t = m és $Bal(P_t) = Nil$ vagy $Jobb(P_t) = Nil$ vagy $t < m$ és $Bal(P_t) \neq Nil \wedge Jobb(P_t) \neq Nil$ és $P_{t+1} = Jobb(P_t)$,$_{Pi+1} = Bal(P_i)$ ha $(t < i < m)$

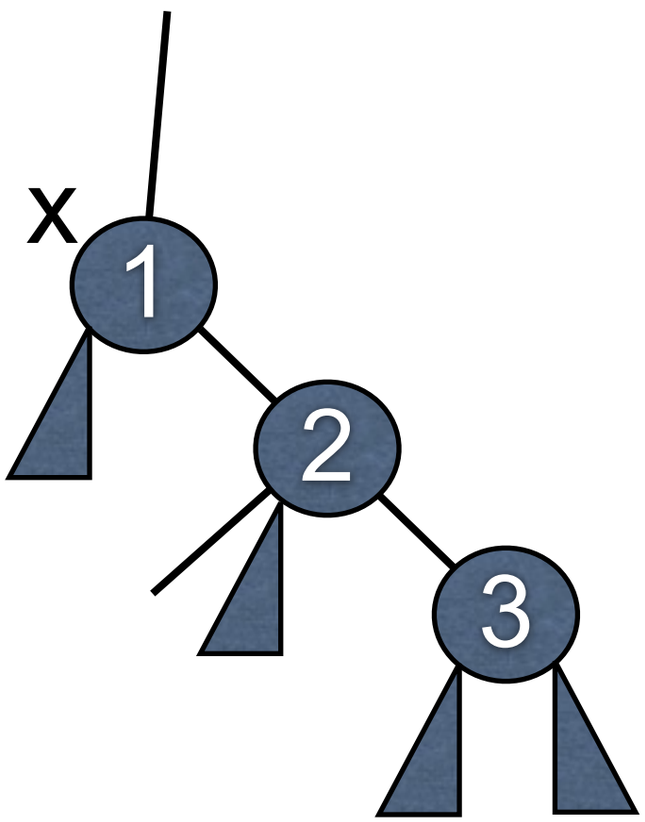
[2]

Forgatás



[1] 279-280

Forgatás

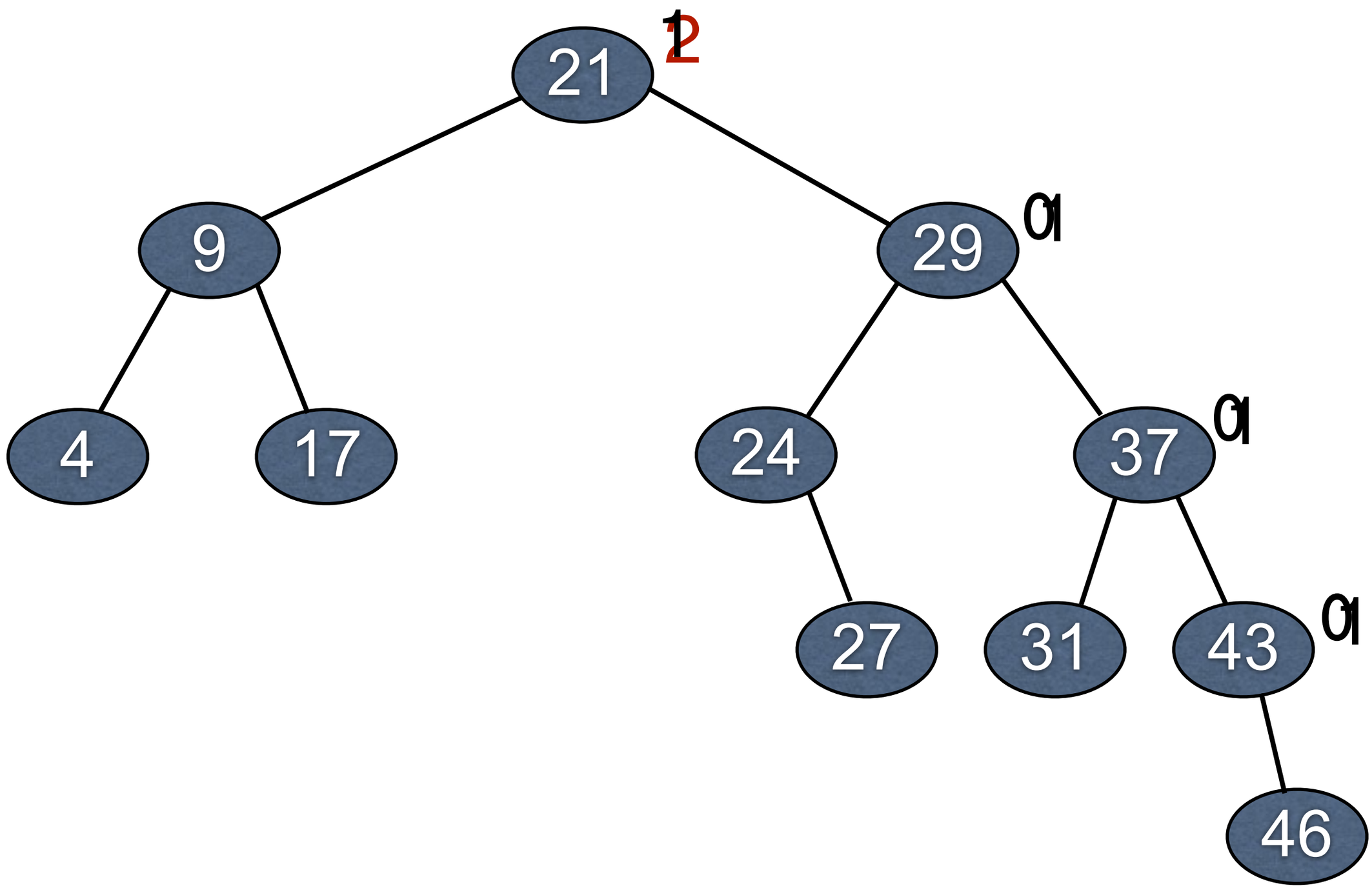



```
void balraforgat(fapont x) {
y    fapont y ;
     y=x.jobb ;
     x.jobb=y.bal ;
     if (x.jobb!=nil) x.jobb.apa=x;
     y.apa=x.apa;
     if (y.apa==nil) gyoker=y ;
     else {
        if (x==y.apa.bal) y.apa.bal=y ;
        else y.apa.jobb=y ;
     }
     y.bal=x ;
     x.apa=y ;
}
```

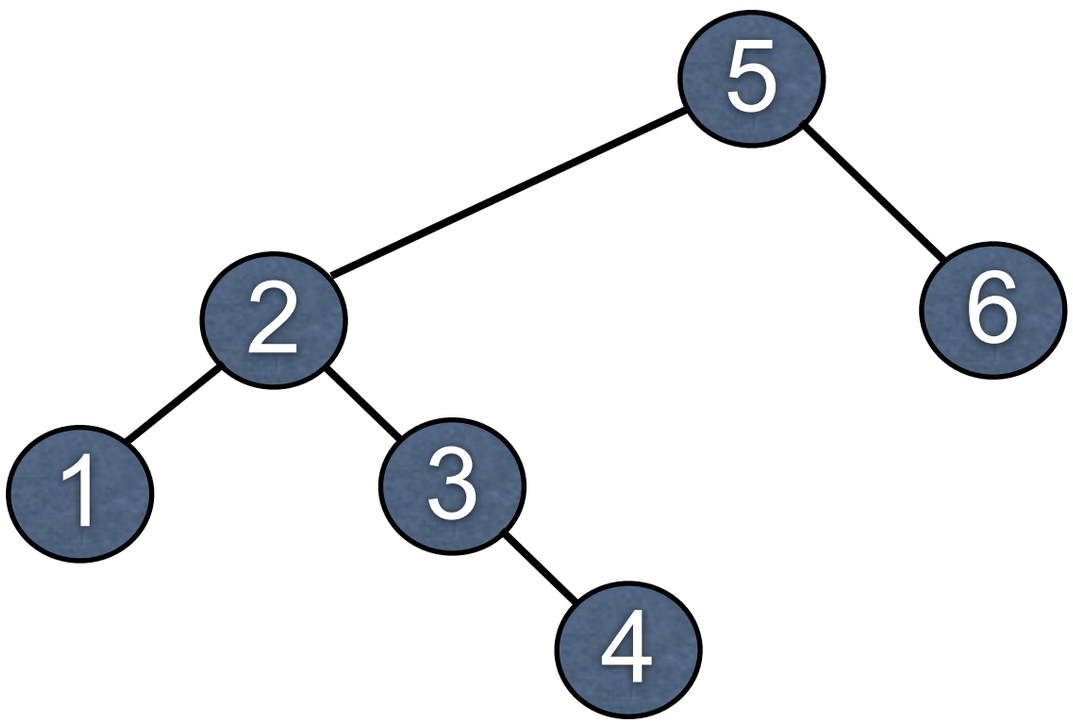
$O(1)$

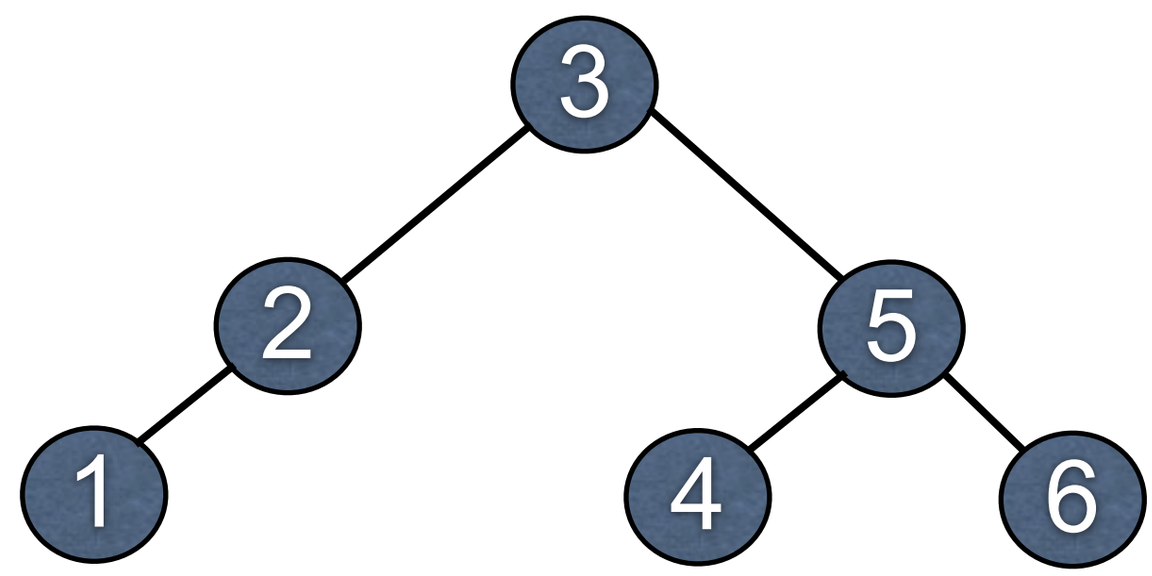
[1] 279-282

Példa

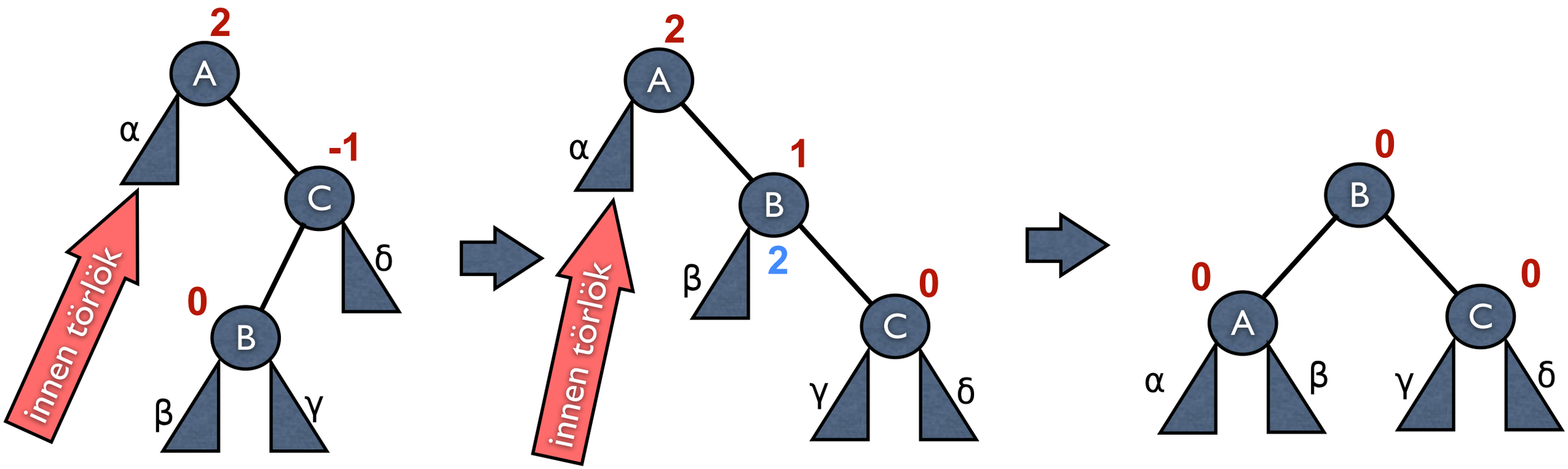


Példa



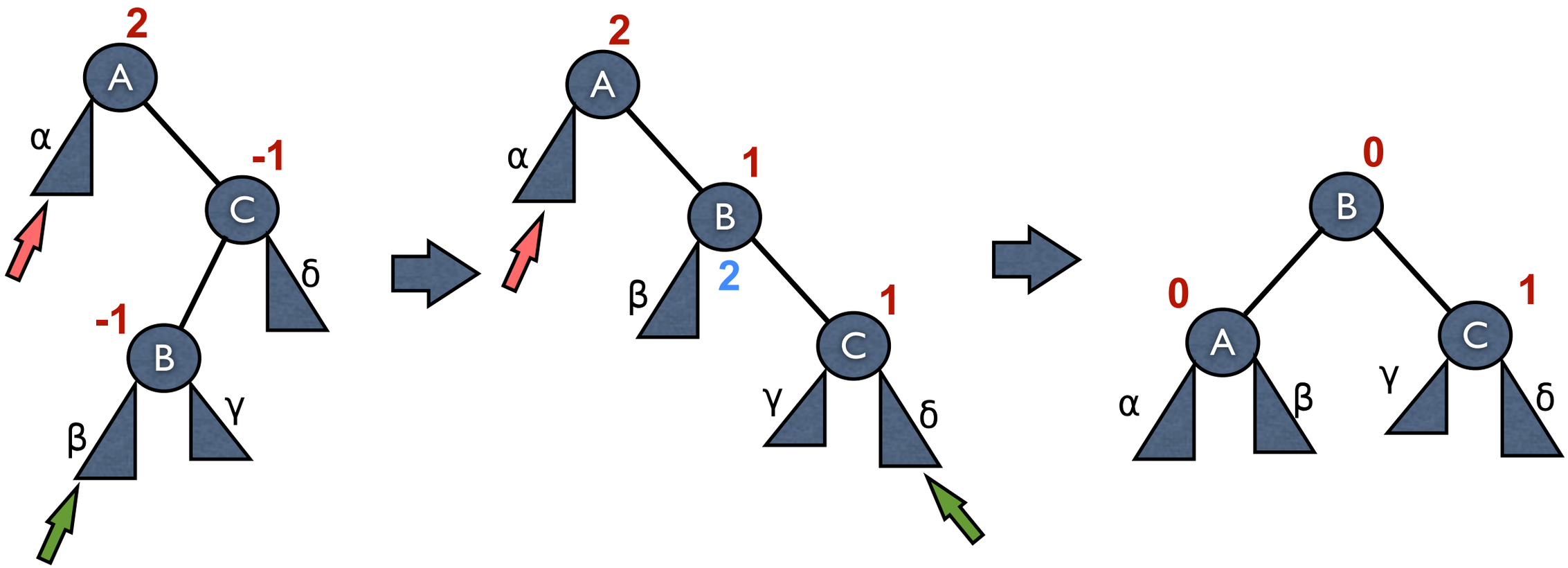


1. eset



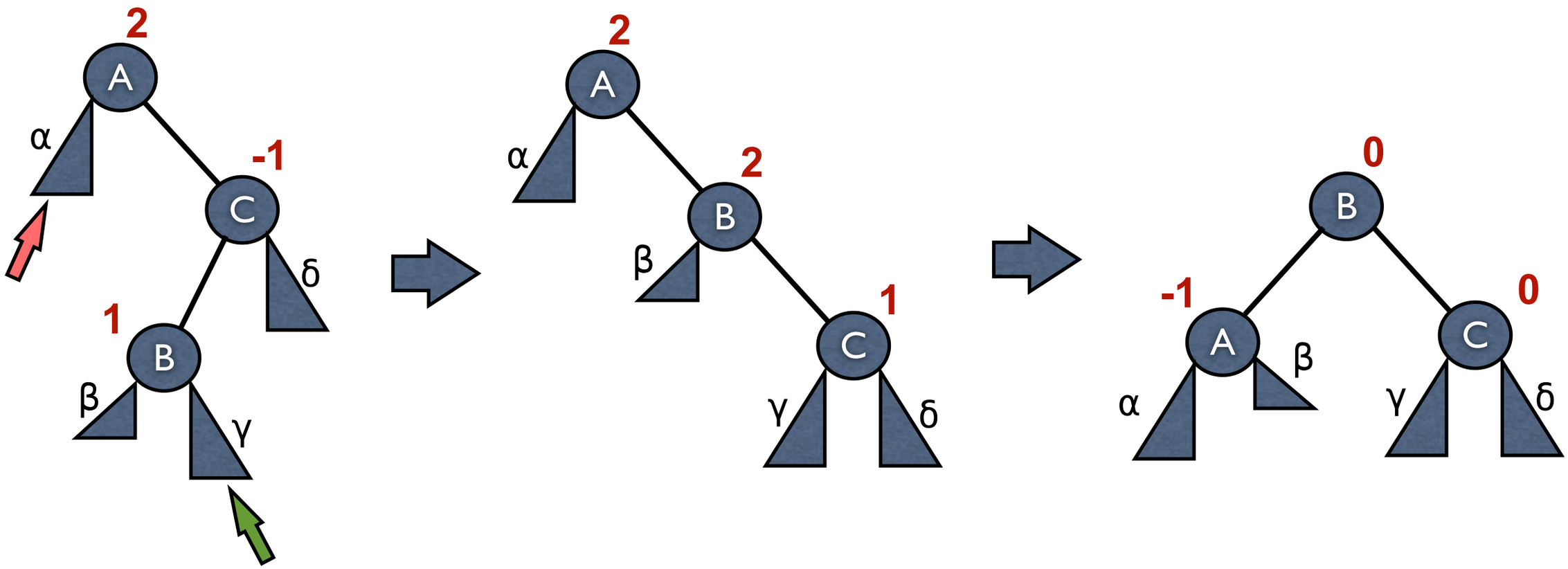
$\alpha=\beta=\gamma=\delta$

2. eset



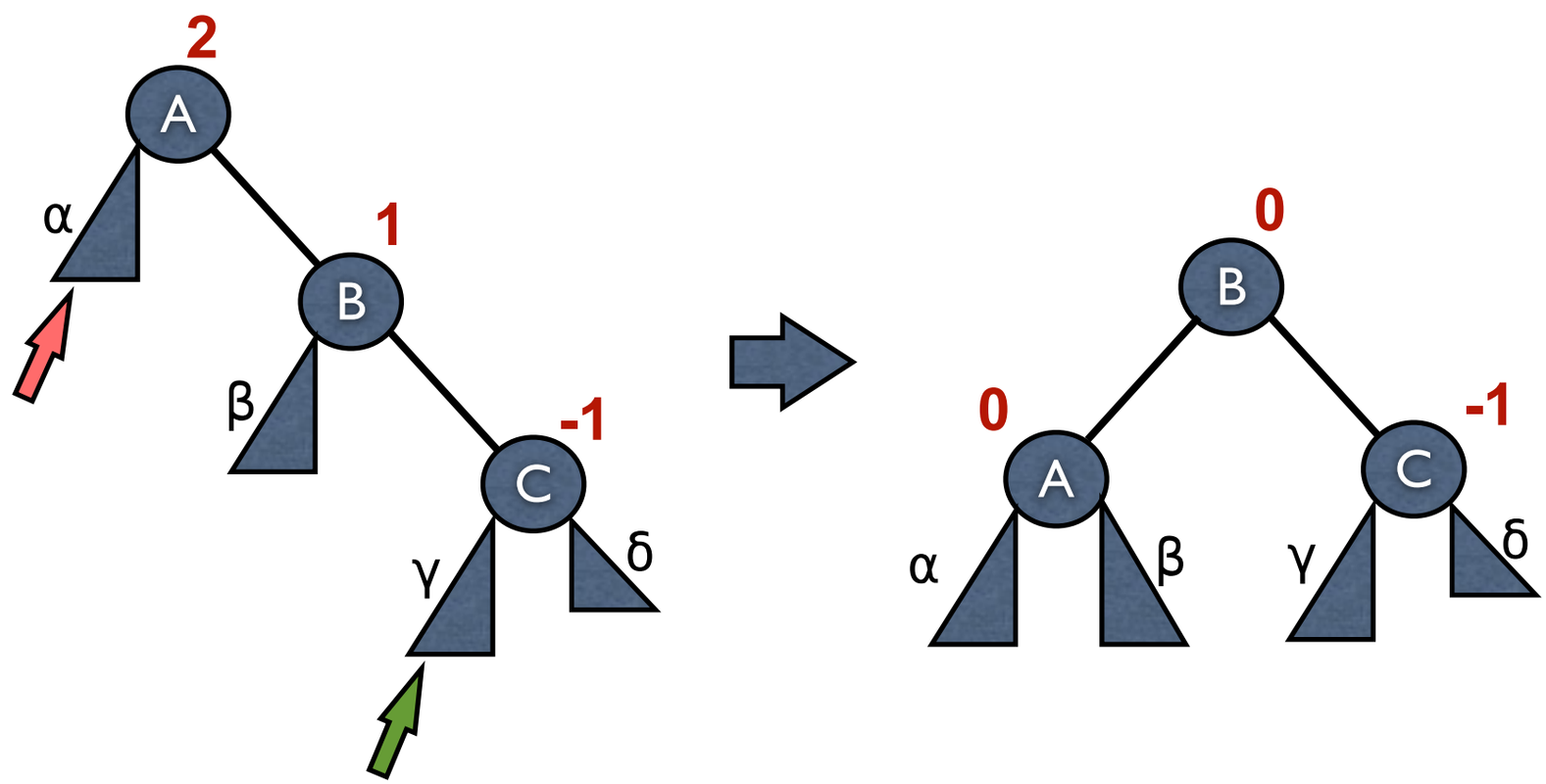
$$\alpha=\beta=(\gamma+1)=\delta$$

3. eset



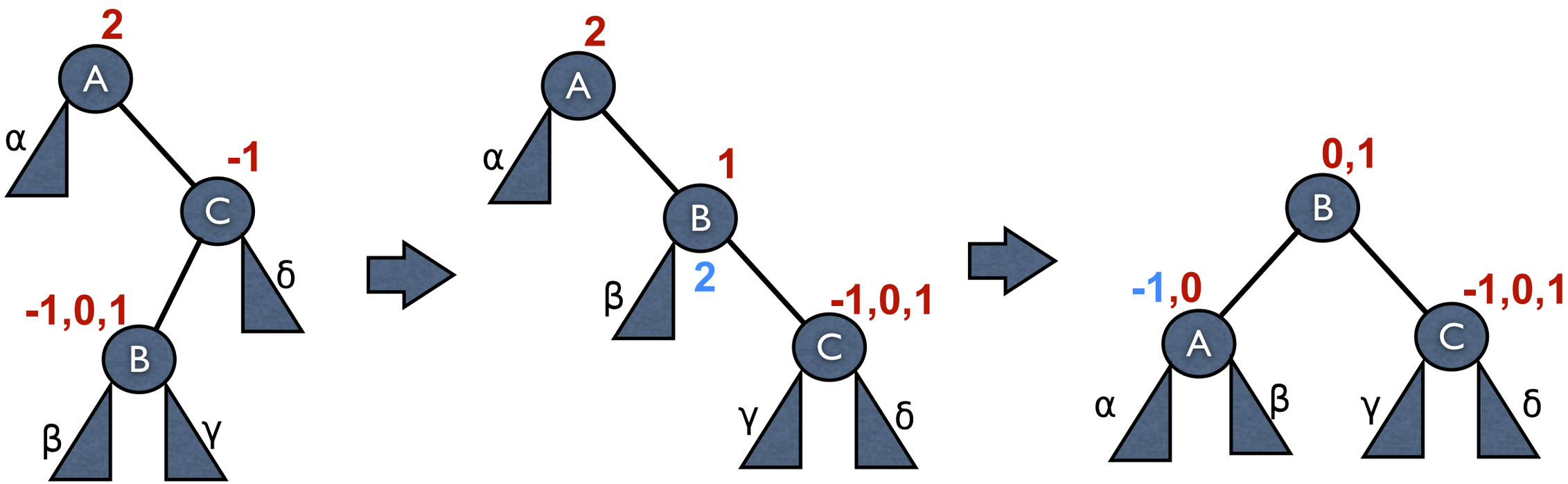
$\alpha = (\beta + 1) = \gamma = \delta$

4. eset



$$\alpha=\beta=\gamma=(\delta+1)$$

Összes eset



$\alpha=\beta=\gamma=\delta \longrightarrow \alpha=\beta=\gamma=\delta$

$\alpha=\beta=(\gamma+1)=\delta \longrightarrow \alpha=\beta=(\gamma+1)=\delta$

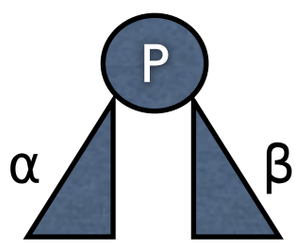
$\alpha=\beta=\gamma=(\delta+1)$

$\alpha=(\beta+1)=\gamma=\delta \longrightarrow \alpha=(\beta-1)=\gamma=\delta$

fj(p,q)

És ezek tükörképei

Egyensúlyfaktorok javítása beszúrás

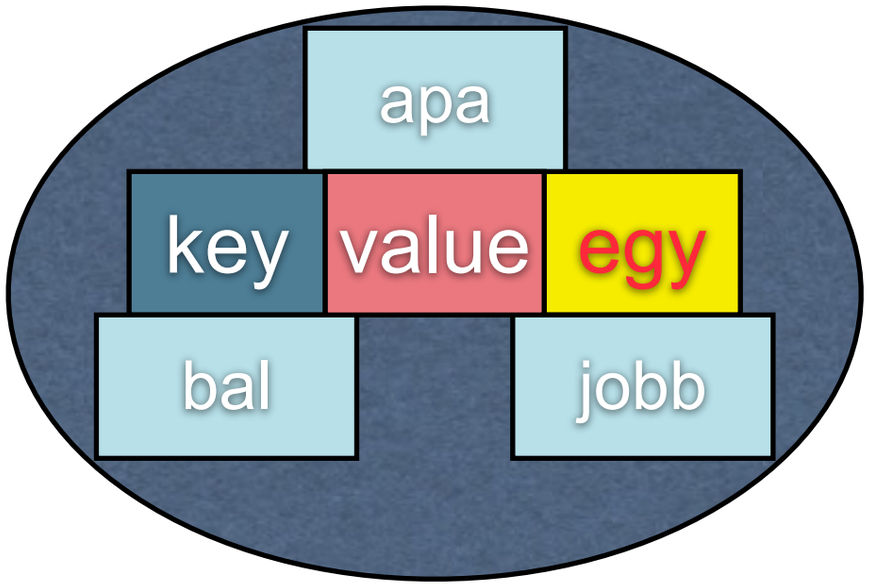


	α=β	α<β	α>β	
x	**0**	**1**	**-1**	beszúrás előtt
bal: αb(x)	**-1**	**0**	**-2**	beszúrás után
jobb: βj(x)	**1**	**2**	**0**	beszúrás után

Fapont tárolása

második megközelítés, a gyakorlatban nem így tároljuk!

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



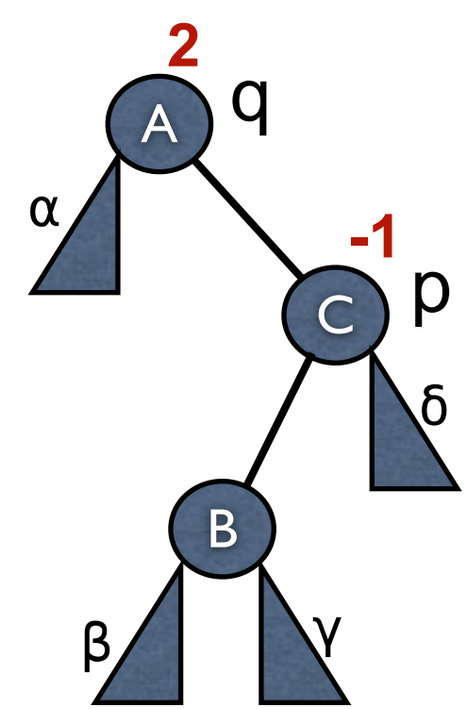
Egyensúlyfaktorok javítása

kezdetben p a beszúrt vagy törölt elem

```
while (p!=gyoker && cv) {
   q=p.apa ;
   if (p==q.jobb) {
      egyq=++q.egy ;
      if (egyq==2) {
         if (p.egy==-1) {
            jobbraforgat(p) ;
         }
         balraforgat(q) ;
         fj(p,q) ;        //egyensúlyfaktorokat rendbetesz
         cv=false ;       //kilép
      }
   } else ...            // bal eset
   p=p.apa ;
}
```

x	0	1	-1
eb(x)	-1	0	-2
ej(x)	1	2	0

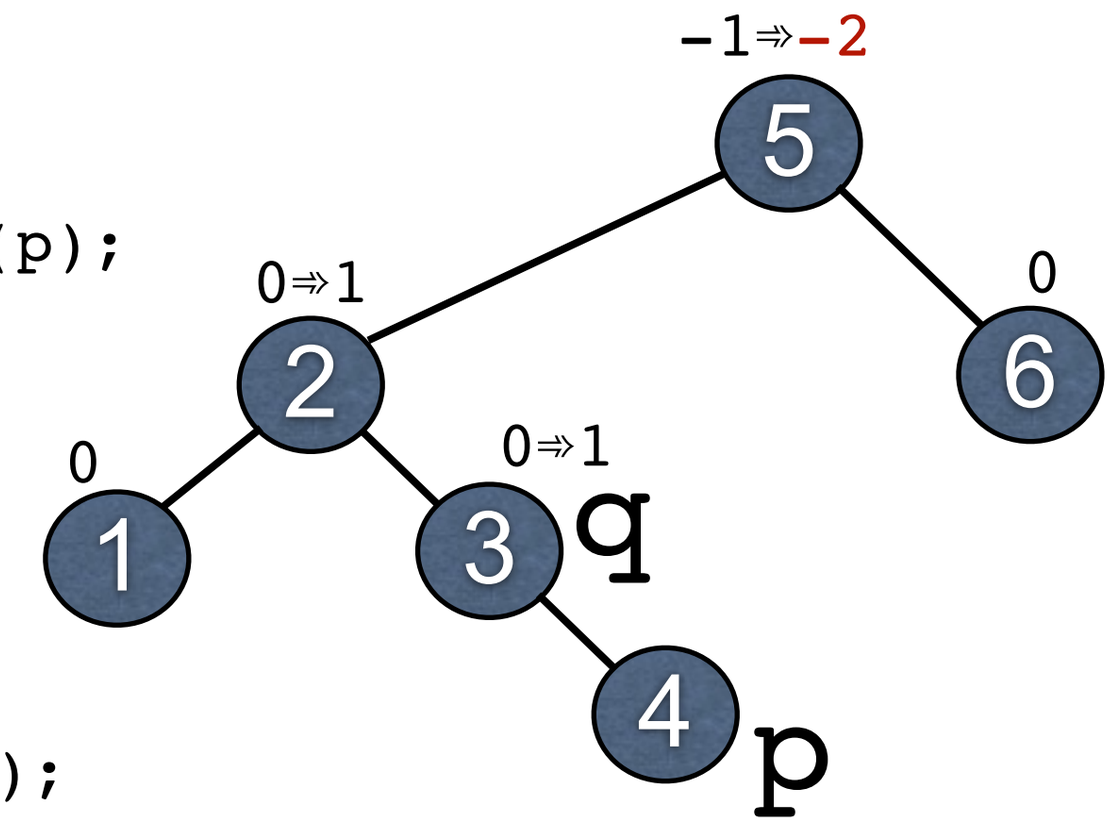
törlésnél (testver(p)==-1)

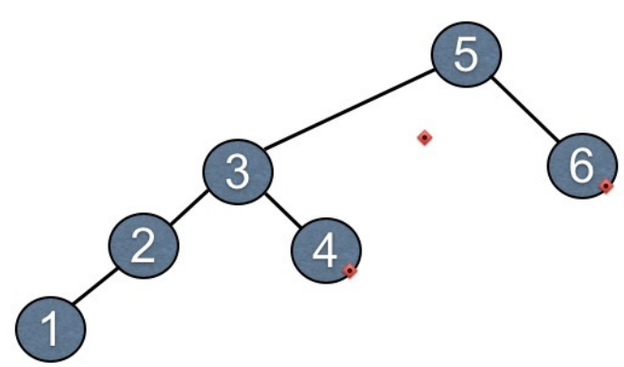


$$O(log\mathbf{n})$$

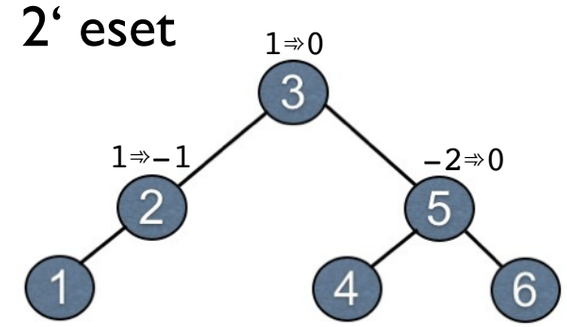
Példa

```
while (p!=gyoker && cv) {
⇒  q=p.apa ;
   if (p==q.jobb) {
      egyq=++q.egy ;
      if (egyq==2) {
         if (p.egy==-1) jobbraforgat(p);
         balraforgat(q);
         cv=false;  fj(p,q) ;
      }
   } else {
      egyq=--q.egy;
      if (egyq==-2) {
         if (p.egy==1) balraforgat(p);
         jobbraforgat(q);
         cv=false;
         fj(p,q) ;
      }
   }
   p=p.apa;
}
```

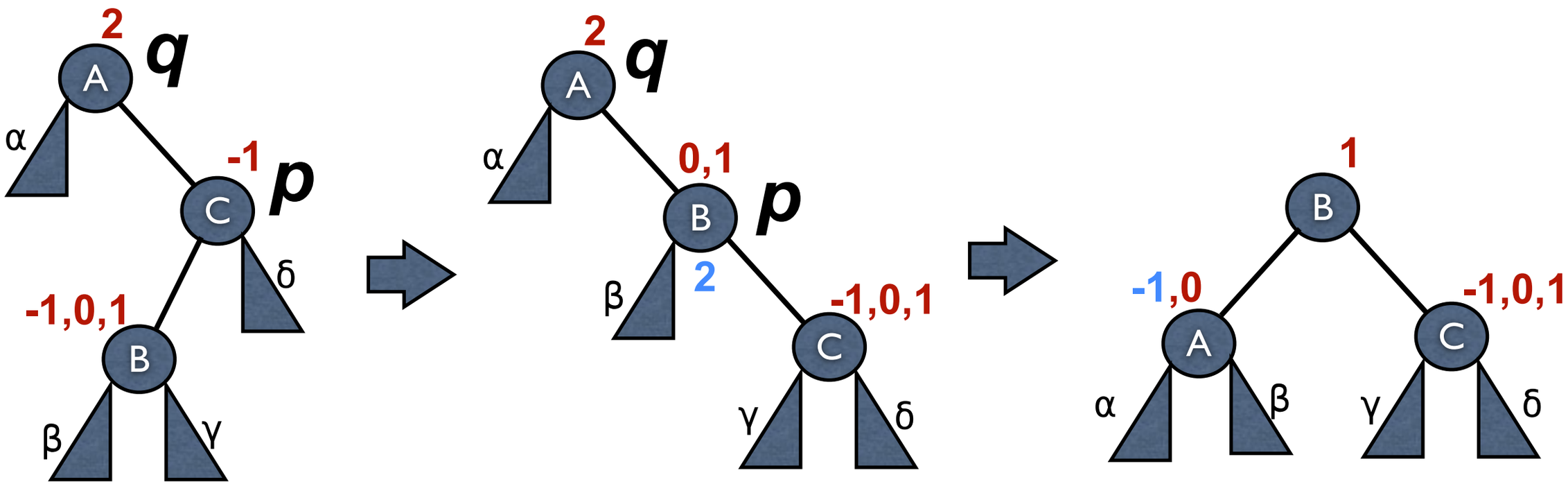




2‘ eset



fj(p,q)



$\alpha=\beta=\gamma=\delta \longrightarrow \alpha=\beta=\gamma=\delta$

$\alpha=\beta=(\gamma+1)=\delta \longrightarrow \alpha=\beta=(\gamma+1)=\delta$

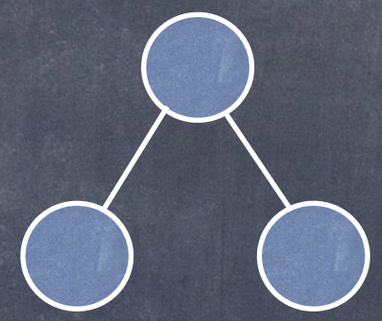
$\alpha=\beta=\gamma=(\delta+1)$

$\alpha=(\beta+1)=\gamma=\delta \longrightarrow \alpha=(\beta-1)=\gamma=\delta$

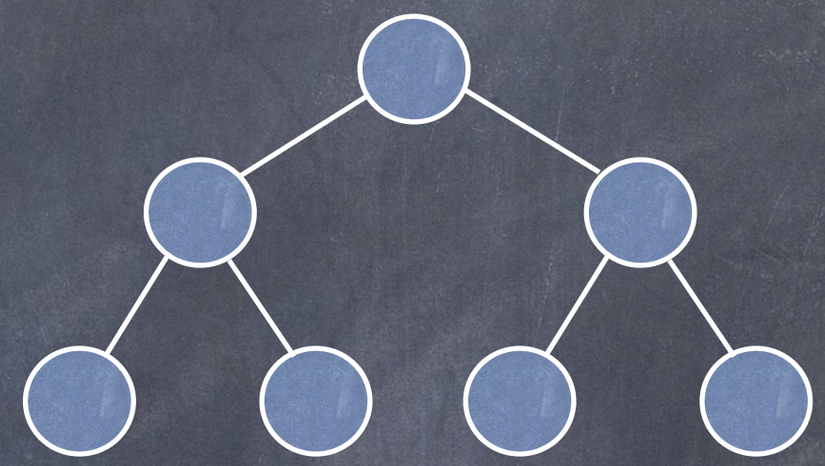
És ezek tükörképei

Teljes bináris keresőfák

n=1



n=3



n=7

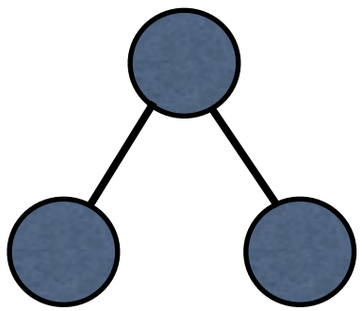
n=2

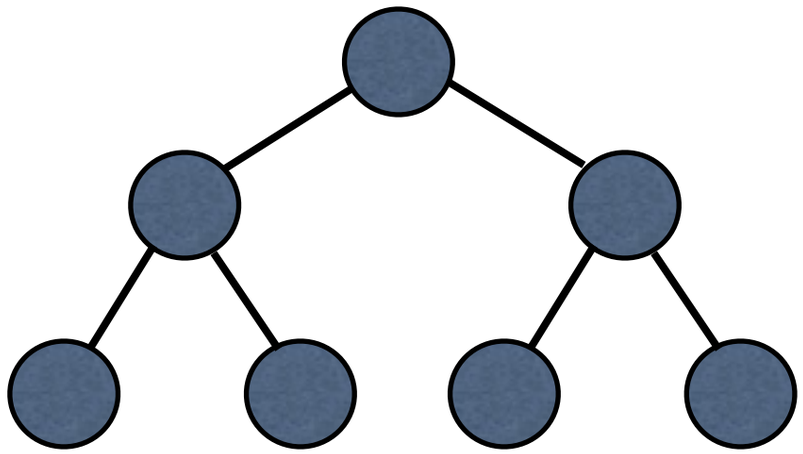
n=4

n=5

n=6

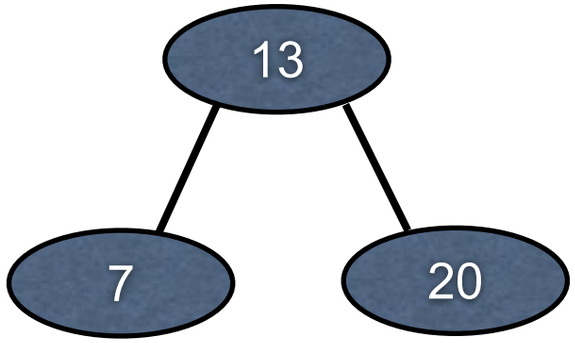
n=8 .. 14

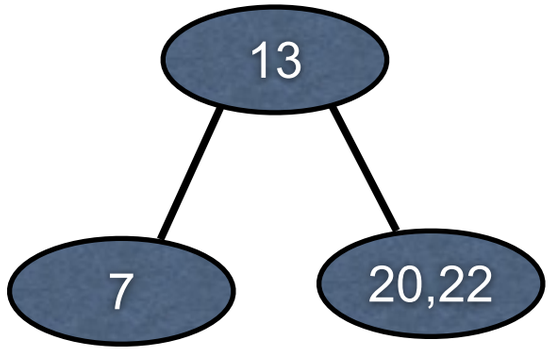


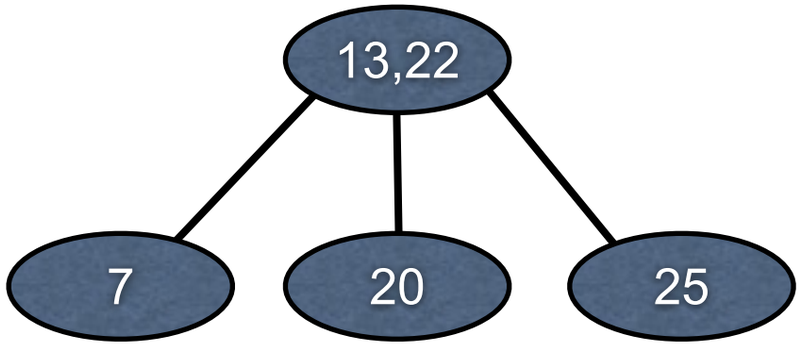


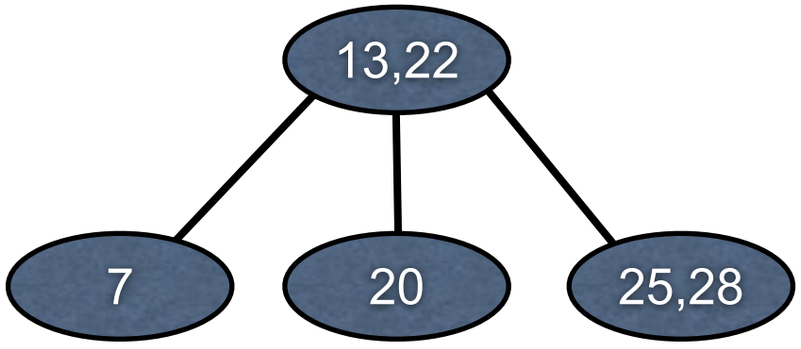
7

7,13


13
7
20


13
7
20,22

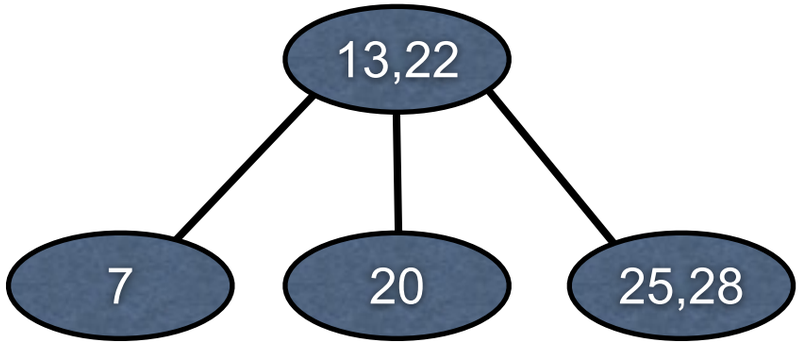

13,22
7
20
25


13,22
7
20
25,28

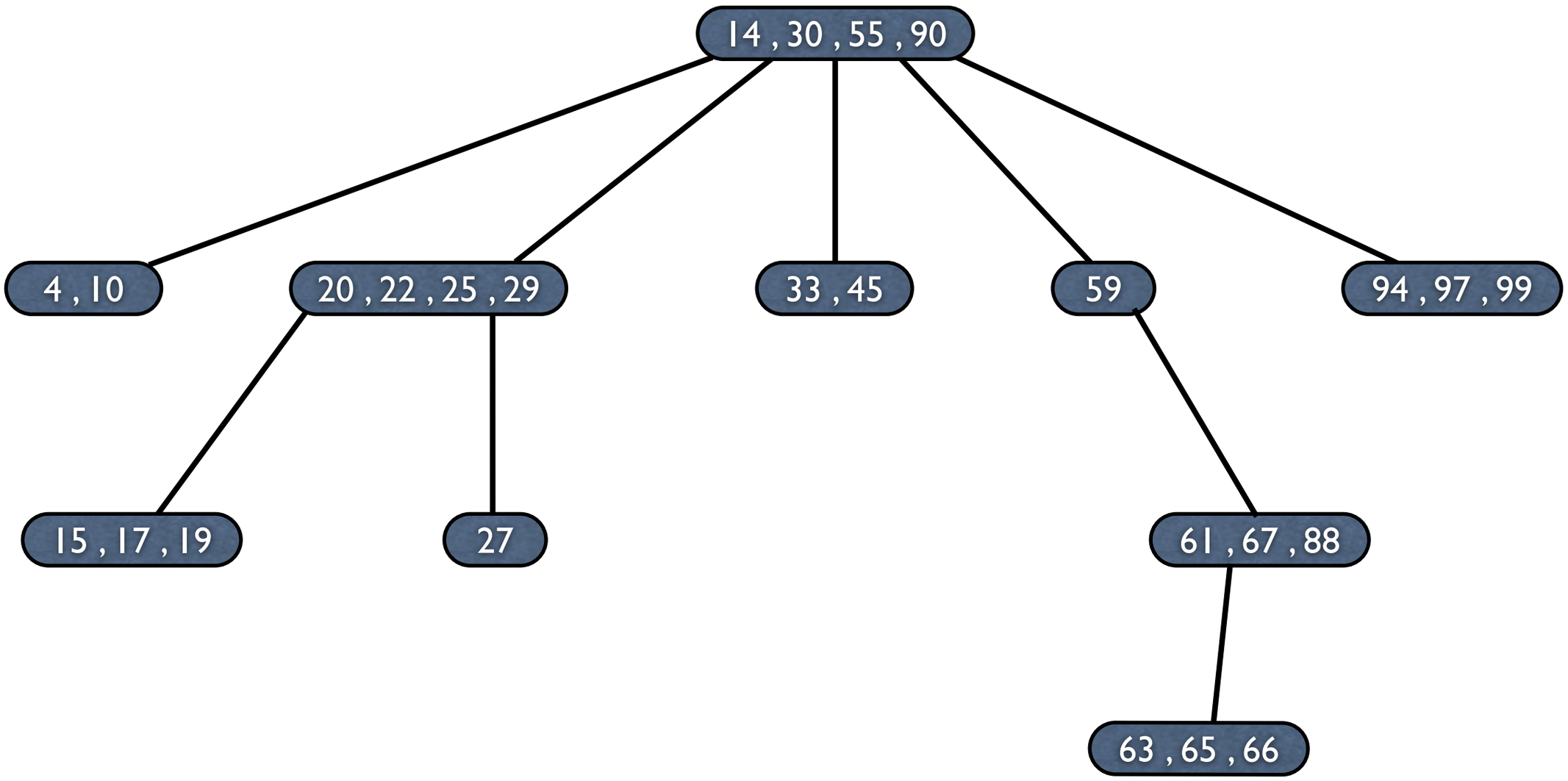
Általános keresőfa

Az általános keresőfa olyan absztrakt adatszerkezet, amely fa és minden cellájában nem csak egy kulcs (adat), hanem kulcsok egy rendezett sorozata van. Tehát minden $p \in F$ esetén $Adat(p) = [a_1,...,a_k]$.
Ekkor azt mondjuk, hogy p rangja k, $Rang(p) = k$.
Def. Az F fa (általános) keresőfa, ha $\forall p \in F$ esetén:

- p-nek $Rang(p)+1 = Fok(p)$ számú fia van.
- $Adat(p) = [a_1,...,a_k]$ egy rendezett sorozat.
- $\max(F\ fiu(p,i)) \leq a_i \leq \min(F\ fiu(p,i+1))$, minden $i = 1,...,Rang(p)$, ahol $F\ fiu(p,i)$ az F fa $fiu(p,i)$ gyökerű részfája.

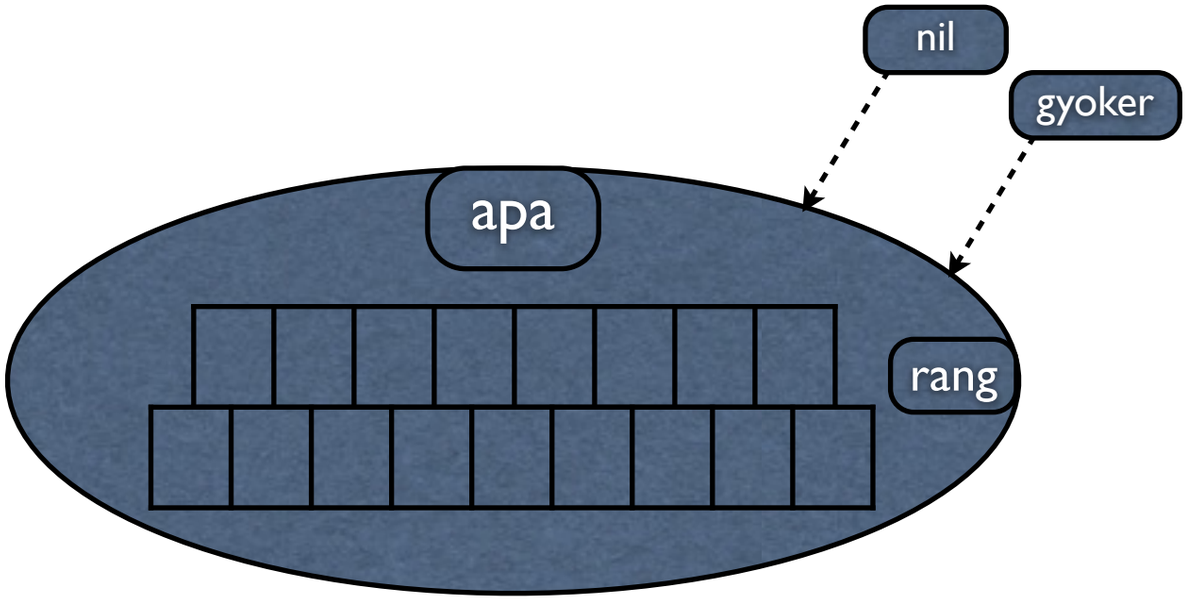


Példa

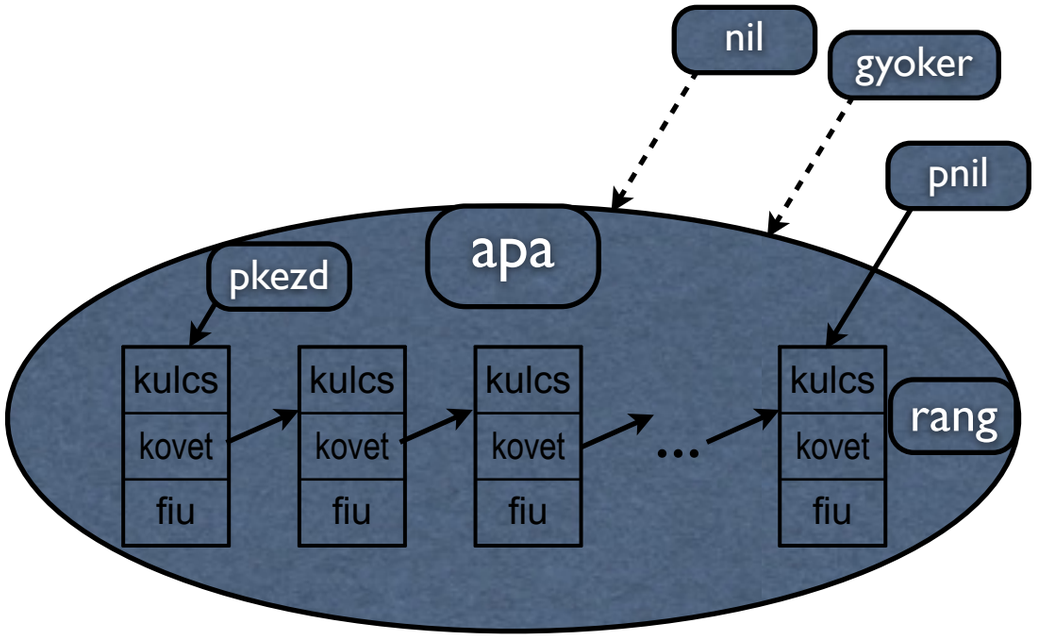

14 , 30 , 55 , 90
4 , 10
20 , 22 , 25 , 29
33 , 45
59
94 , 97 , 99
15 , 17 , 19
27
61 , 67 , 88
63 , 65 , 66

Fapont tárolása

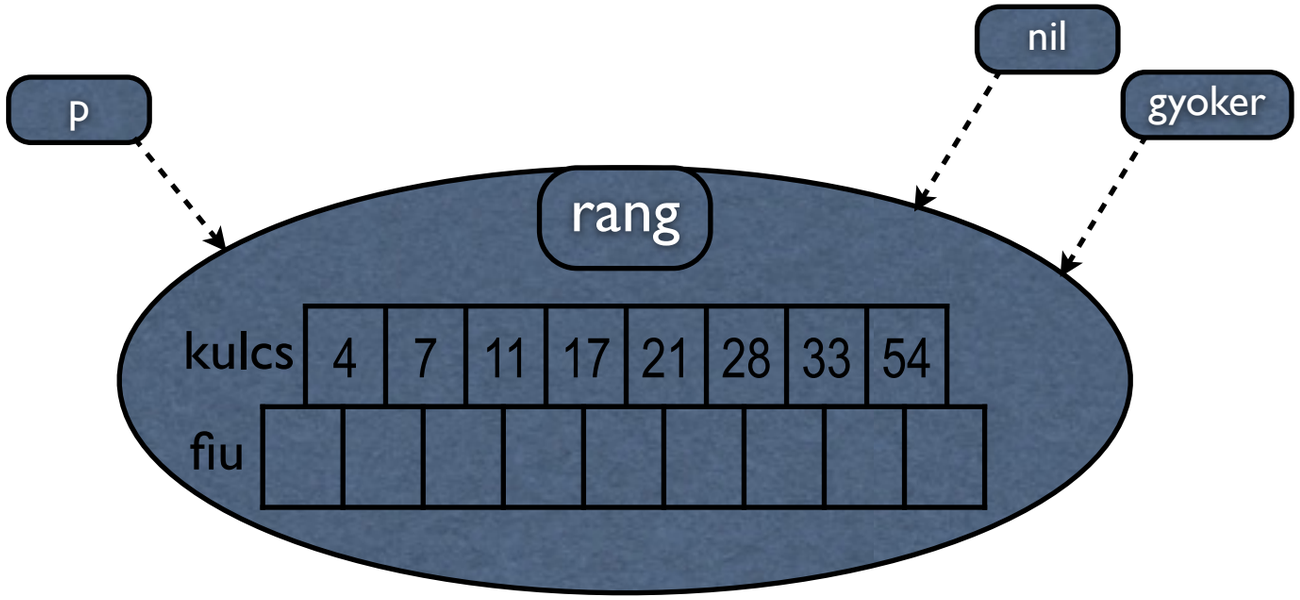
```
fapont class<T> {
   int rang ;
   fapont apa ;
   <T>[] kulcs;
   fapont[] fiu ;
}
```




```
fapont class<T> {
   int rang ;
   fapont apa ;
   pontelem class<T>{
      <T> kulcs ;
      fapont fiu ;
      pontelem kovet ;
   } ;
}
```



Keresés általános keresőfában

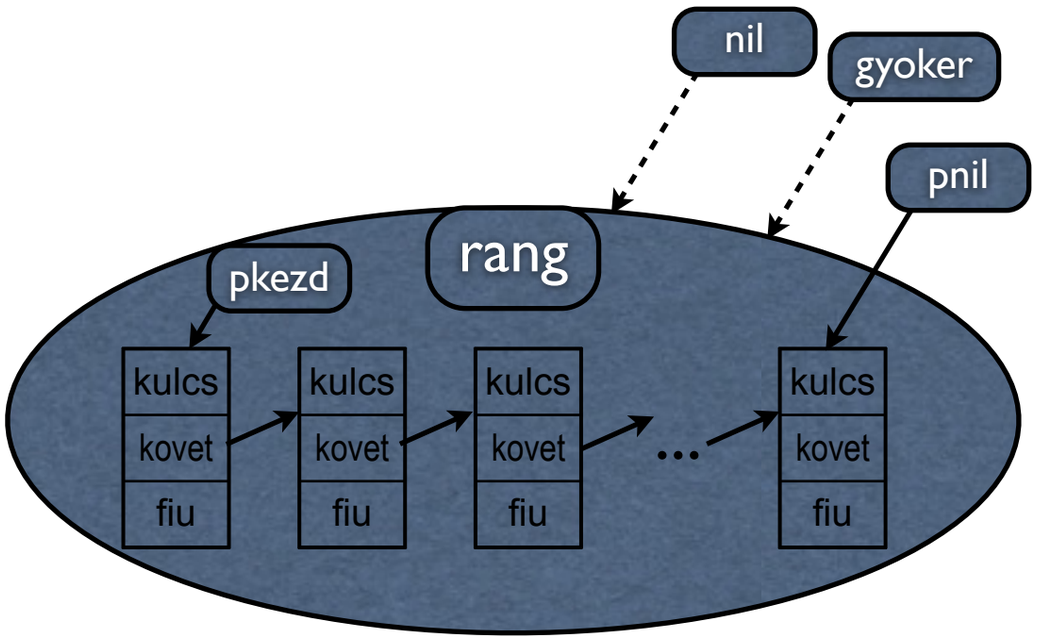



```
keres<T>(<T> x,fapont p) {
   if (p==nil) return nil ;
   int i=1 ;
   while (i<=p.rang && x>p.kulcs[i]) i++ ;
   if (x==p.kulcs[i]) return (p,i) ;
   return(keres(x, p.fiu[i])) ;
}
```

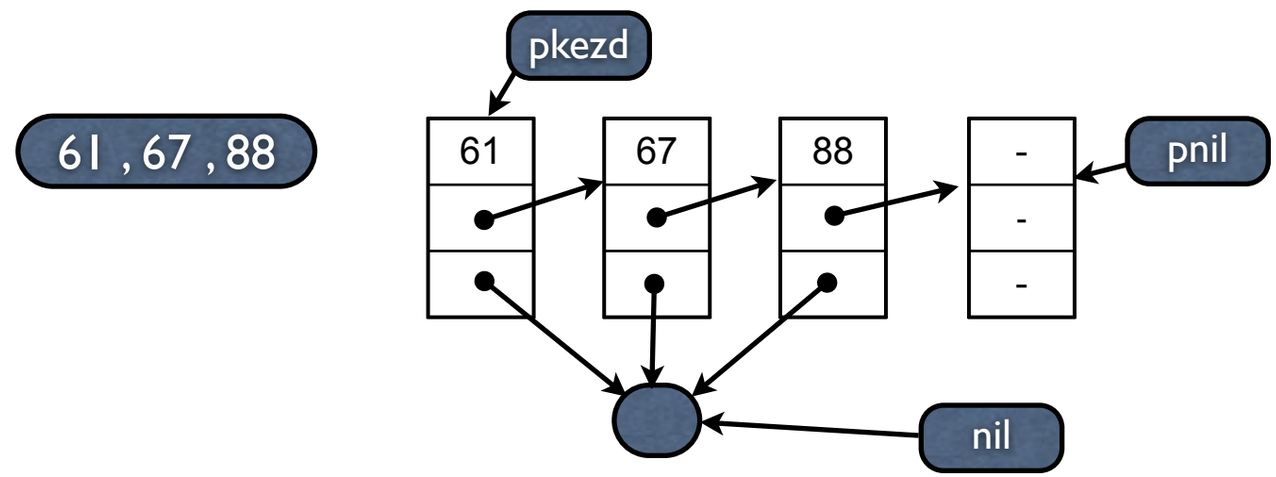
Keresés általános keresőfában

```
class par {
   fapont p;
   pontelem e;
}
par(q,i) {
   p=q ;
   e=i ;
}

int keres<T>(<T> x) {
   fapont q,p=gyoker;
   pontelem i;
   do {
      i=p.pkezd;
      while (i!=pnil && x>i.kulcs) i=i.kovet;
      if (i!=pnil && x==i.kulcs) return par(p,i);
      q=p; p=i.fiu ;
   } while (p!=nil);
   return nil ;
}
```



Beszúrás általános keresőfába




```
int beszur<T>(<T> x) {
   int poz;
   fapont q,p=gyoker;
   pontelem i;
   do {
      i=p.pkezd; poz=0;
      while (i!=pnil && x>i.kulcs) { i=i.kovet; poz++; }
      if (i!=pnil && x==i.kulcs) return -1;
      q=p; p=i.fiu ;
   } while (i.fiu!=nil);
   listabaszur(q,i,poz) ;  ????
   return 0 ;
}
```

B-fa

Def. Az F fa t-rendű ($t \geq 2$) B-fa, ha teljesül rá az alábbi 4 feltétel:

- Általános keresőfa.
- A gyökér kivételével minden $p \in F$ pontra $t \leq Rang(p) \leq 2t$,

  a gyökér rangja legfeljebb 2t.
- Minden $p \in F$ nem levél pontra $Fiu(p,i) \neq Nil$, $i = 1,...,Rang(p) +1$.
- Minden $p \in F$ levélpontra $d(p) = h(F)$,

  azaz minden levél pont mélysége azonos.

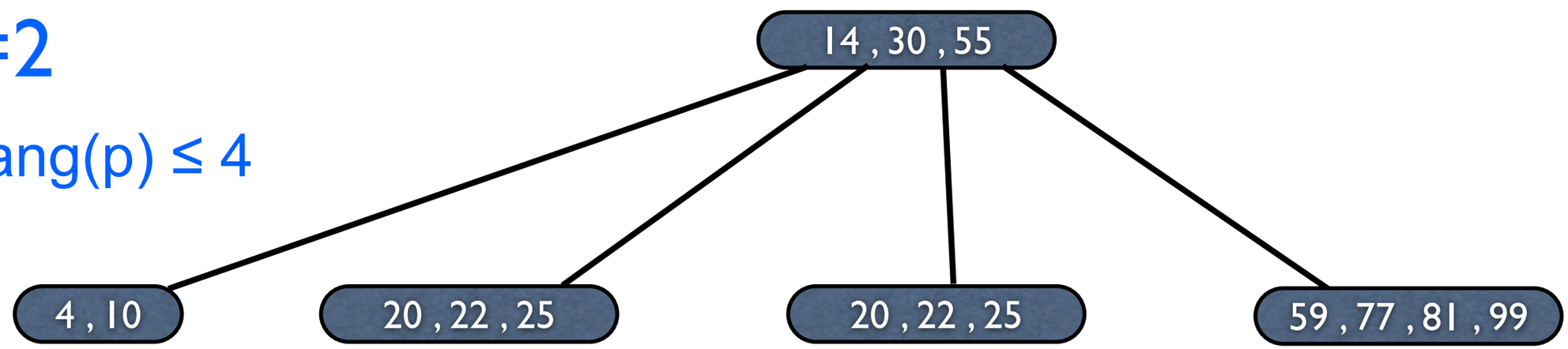
A gyökér kivételével minden $p \in F$ pontra $t \leq Rang(p) \leq 2t$,

a gyökér rangja legfeljebb 2t.

$t \leq Rang(p) \leq 2t$

pl. t=2

$2 \leq Rang(p) \leq 4$



p pontot telített, ha $Rang(p) = 2t$,

p minimális, ha $Rang(p) = t$.

Minden ponthoz tartozik egy *Levél* logikai változó.

B-fák magassága

Tétel: Minden n adatot tartalmazó t-rendű F B-fára:

$$h(f) < \log_t n$$

Biz. Vizsgáljuk meg, hogy egy h magasságú t rendű B-fában mennyi az adatok minimálisan lehetséges száma.
A gyökér legalább egy adatot tartalmaz, és van legalább két fia.
A első szinten mindkét fiú legalább t adatot tartalmaz, és mindkettőnek van legalább t+1 fia.
Azaz a második szinten legalább 2 (t + 1) pont van.
A **második szinten** a 2 (t + 1) pont mindegyike tartalmaz legalább t adatot, így a szinten **az adatok száma** minimum 2 t (t + 1) továbbá minden pontnak van legalább t + 1 fia, így a következő (3.) szinten a pontok száma 2 (t+1) (t+1).
Teljes indukcióval igazolható, hogy az i-edik szinten a pontok száma legalább $2(t+1)^{i-1}$
, így az adatok száma legalább $2t(t+1)^{i-1}$.
Következésképpen egy h magas fában az adatok száma:

$$n \geq 2t(t+1)^{h-1} \geq 2tt^{h-1} = 2t^h > t^h$$

$$\log_t n > \log_t t^h = h \log_t t = h$$

	pontok száma	adatok száma
gy	1	1
1	2	2t
2	2(t+1)	2t(t+1)
3	2(t+1)(t+1)	2t(t+1)(t+1)
i	$2(t+1)^{i-1}$	$2t(t+1)^{i-1}$

Comer (1979, p. 127) and Cormen et al. (year, pp. 383–384) give a slightly different expression for the worst case height (perhaps because the root node is considered to have height 0).

$$h \leq \lfloor \log_d(\frac{n+1}{2}) \rfloor.$$

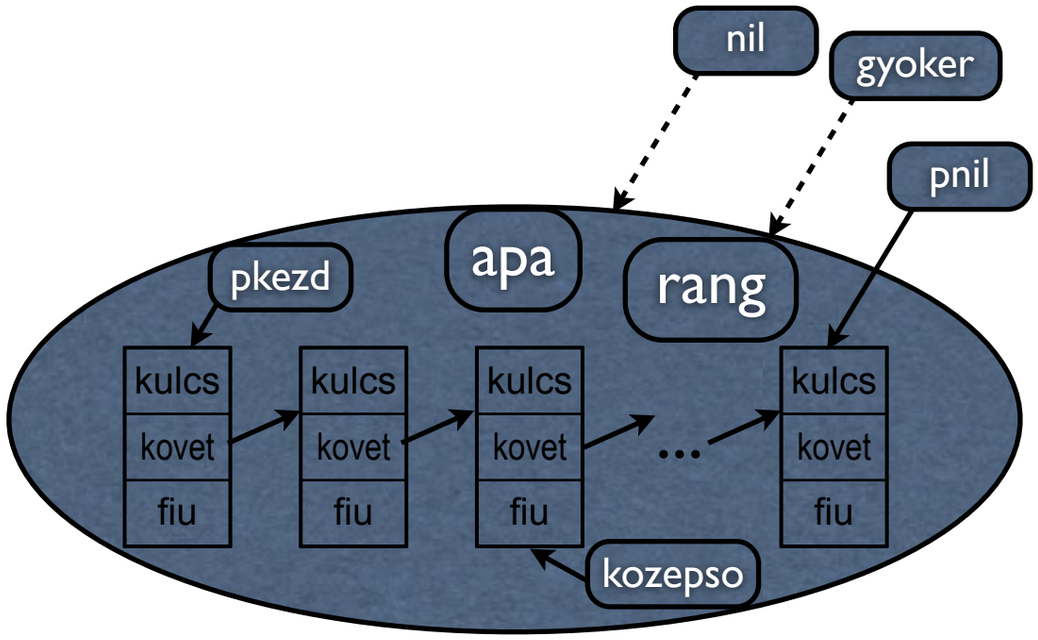
Pont szétvágása

A beszúrás során a pontot abba a levélbe szúrjuk be, ahol a sikertelen keresése véget érne. Ekkor ha ez a levél telített, akkor a beszúrás sértené a pontok rangjára vonatkozó korlátot. Ennek megelőzésére a beszúrás során megszüntetjük a telített pontokat. Erre használhatjuk a telített pontok szétvágását megvalósító alábbi eljárást:

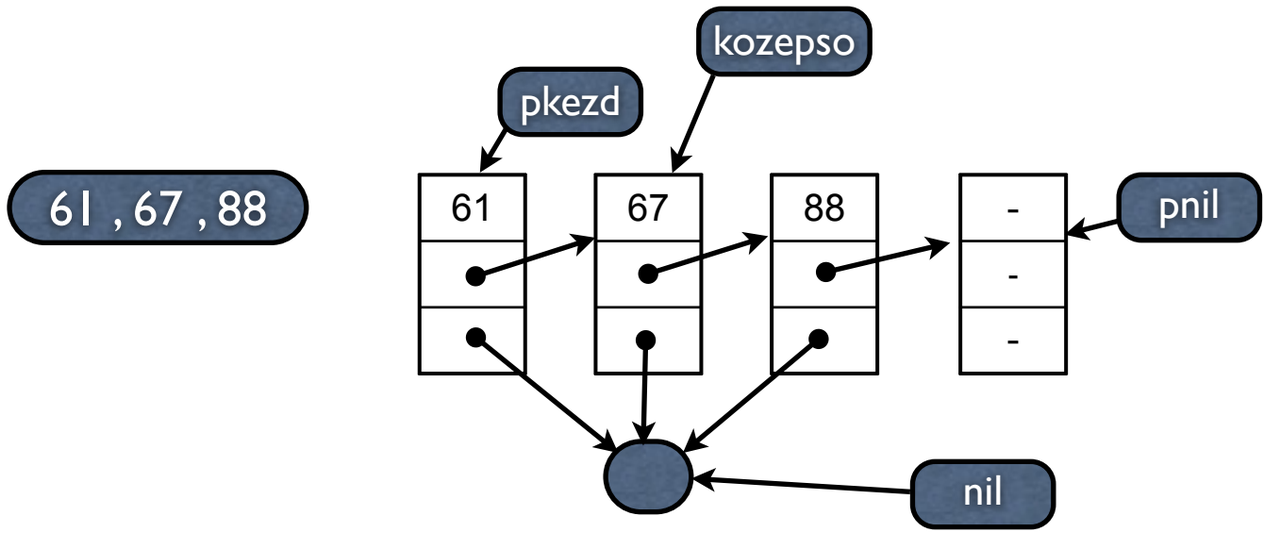
Az algoritmus az y pontot, *amelyre rang(y) = 2(t−1) és amely az p pont i-edik fia* vágja ketté a közepén. A középső adat felmegy az p-be.

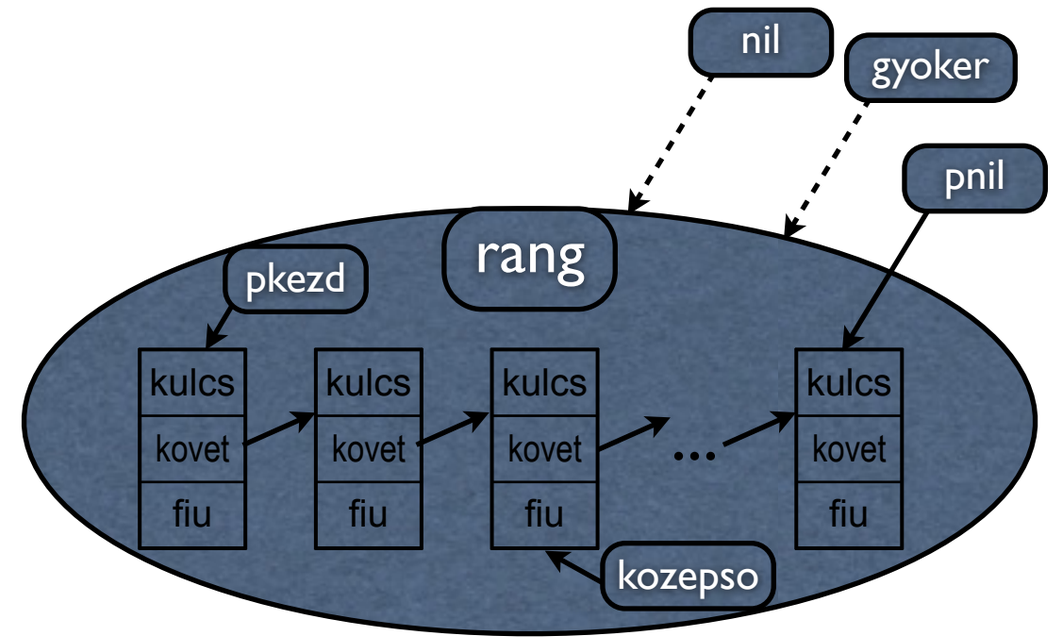
B-fa pont tárolása

```
bfa class<T> {
   fapont class<T> {
      fapont apa ;
      int rang ;
      pontelem class<T>{
         <T> kulcs ;
         fapont fiu ;
         pontelem kovet ;
      } ;
      pontelem pkezd,kozepso;
   bfa() {
      fapont q=new fapont() ;
      nil=q ;
      gyoker=q ;
      pontelem e=new pontelem ;
      e.fiu=nil ;
      e.kovet=pnil ;
      pkezd=e ;
      kozepso=e ;
   }
   ...
}
```

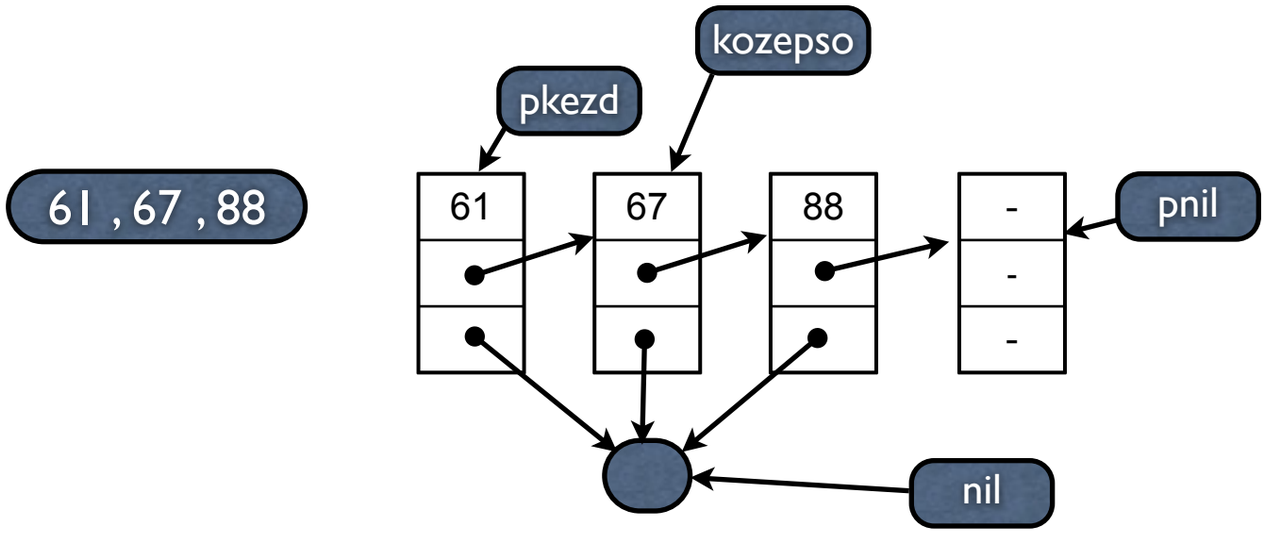


B-fa beszúr



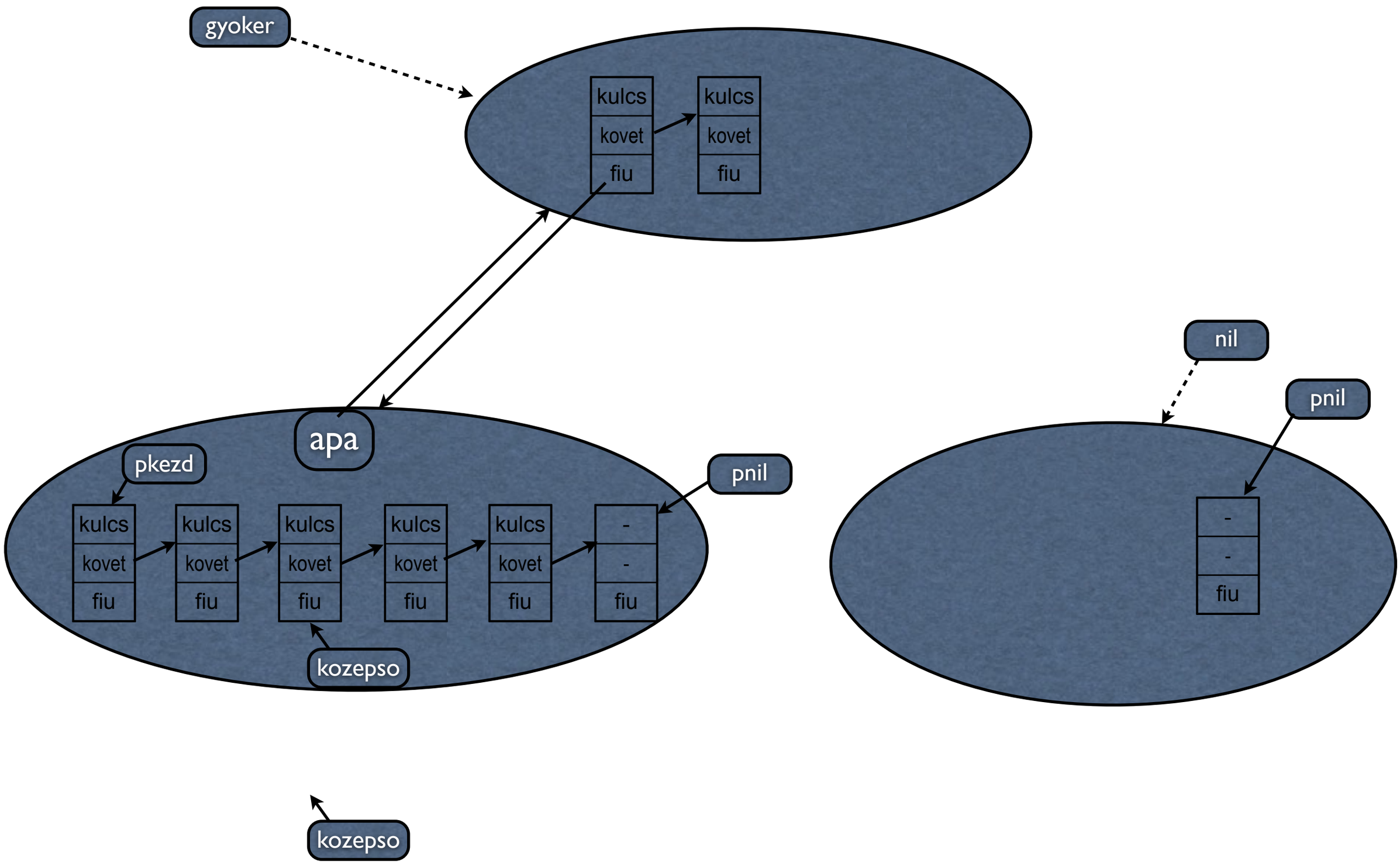



```
int beszur<T>(<T> x) {
   int poz;
   fapont q,p=gyoker;
   pontelem i;
   do {
      i=p.pkezd; poz=0;
      while (i!=pnil && x>i.kulcs) { i=i.kovet; poz++; }
      if (i!=pnil && x==i.kulcs) return -1;
      p=i.fiu ;
   } while (i.fiu!=nil);
   listabaszur(p.apa,i,poz,x) ;
   return 0 ;
}
```

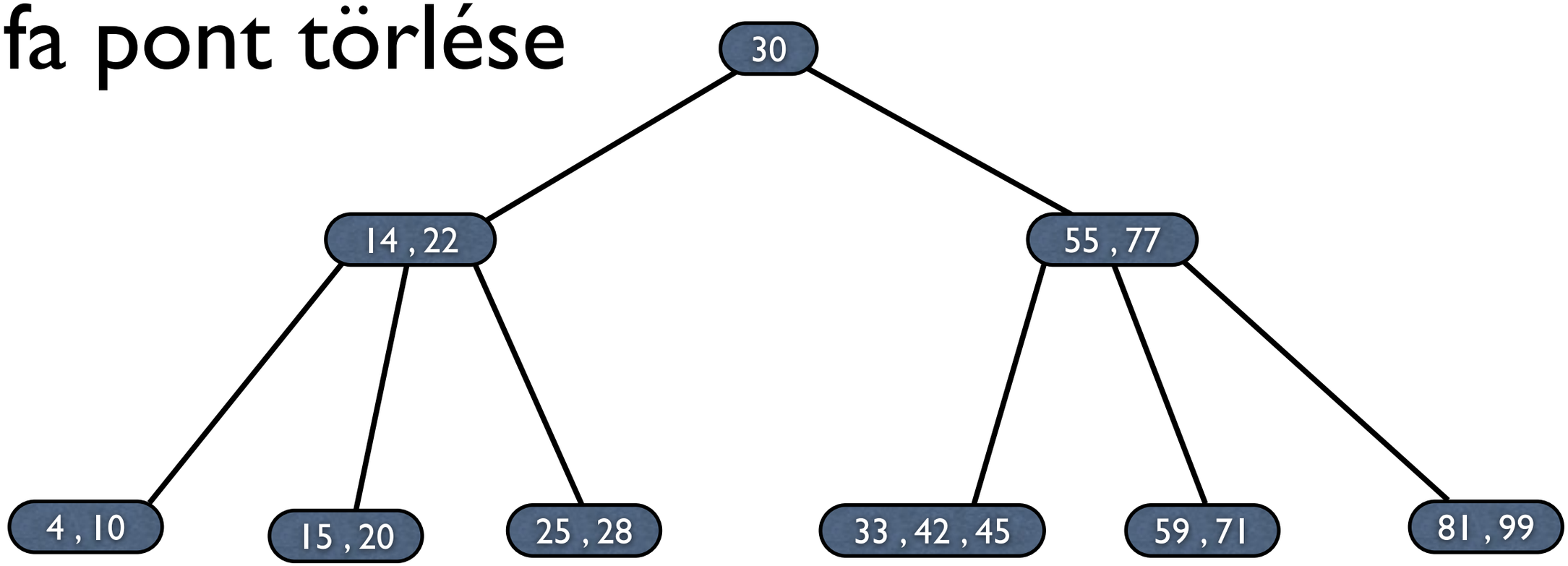



```
listabaszur<T>(fapont p, pontelem i, int poz, <T>x) {
   pontelem e=new pontelem();
   e.kulcs=x ;
   e.kovet=i.kovet ;
   i.kovet=e ;
   p.rang++ ;
   if (p.rang % 2) {
      if (poz>maxrang/2) kozepso=kozepso.kovet ;
      else //előrréb állítani?
   }
   if (p.rang>maxrang) szetszed(p) ;
}
```

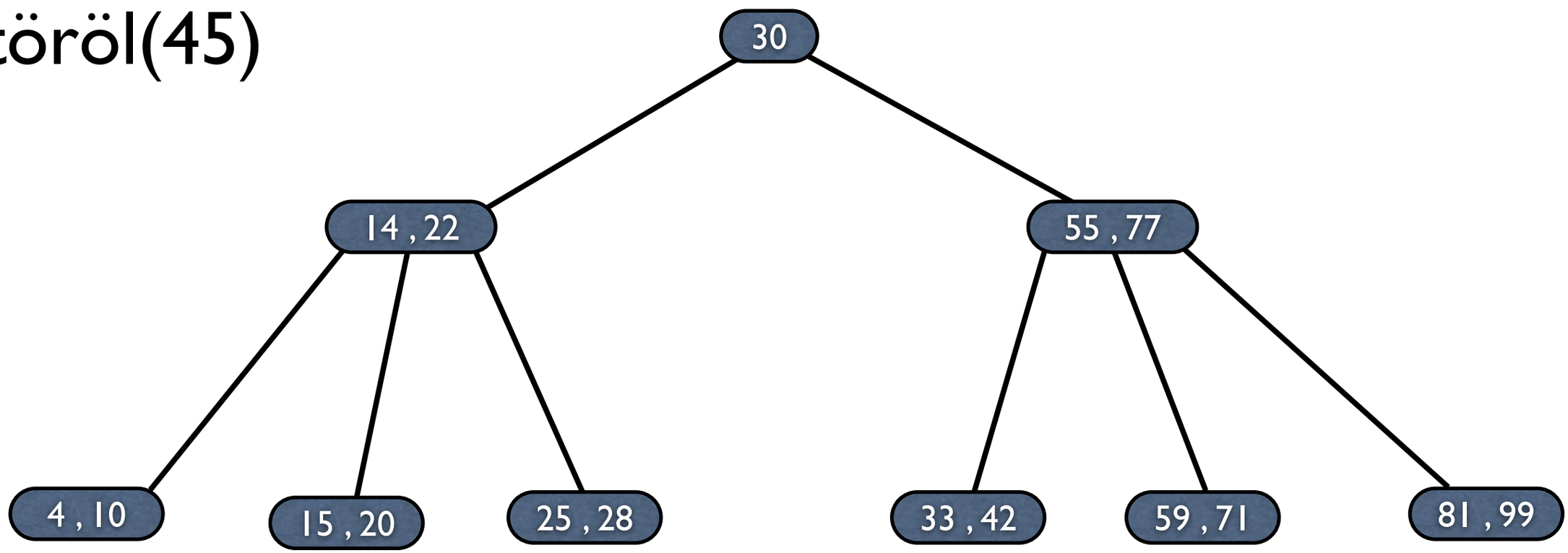
Szétszed


gyoker
kulcs
kovet
fiu
kulcs
kovet
fiu
nil
pnil
apa
pkezd
pnil
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
-
-
fiu
-
-
fiu
kozepso
kozepso

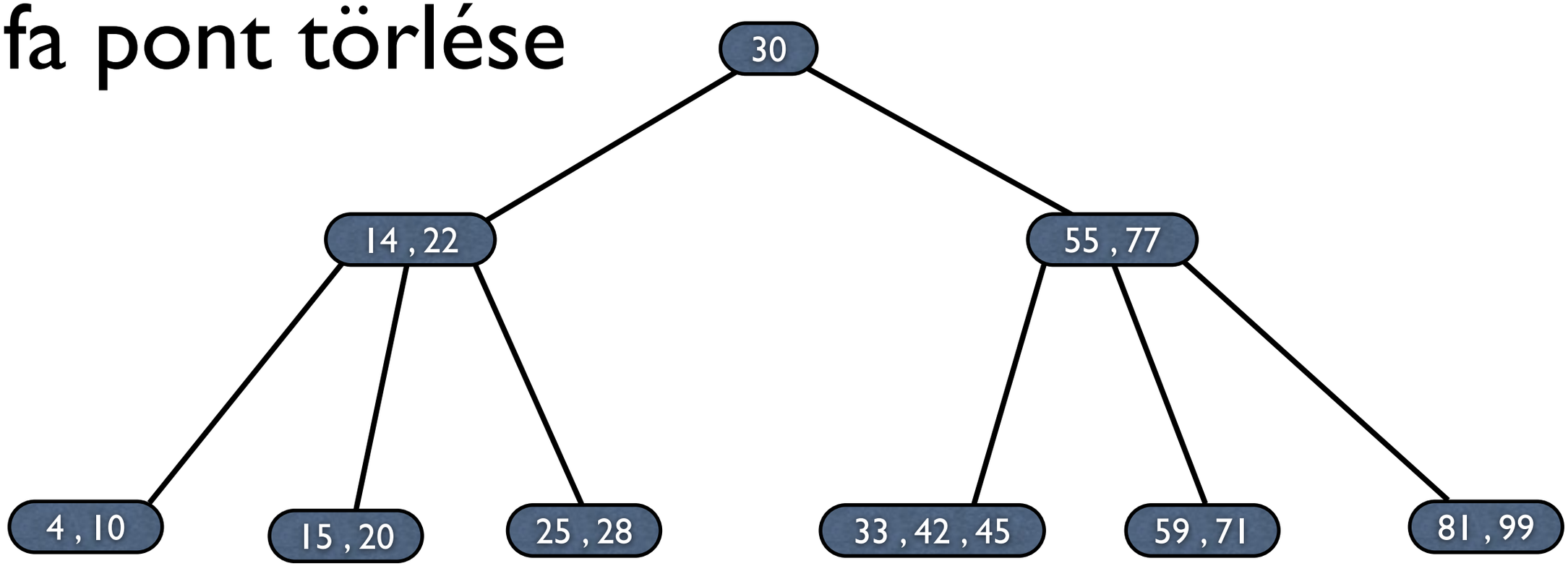
B-fa pont törlése



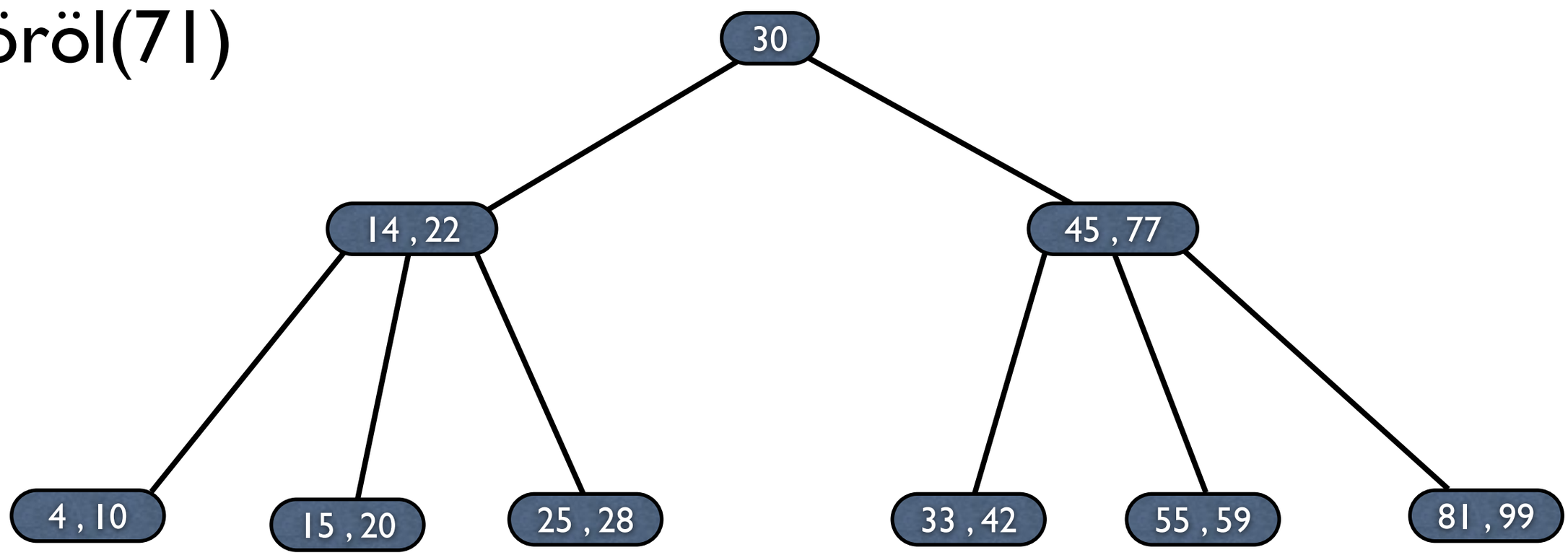
töröl(45)



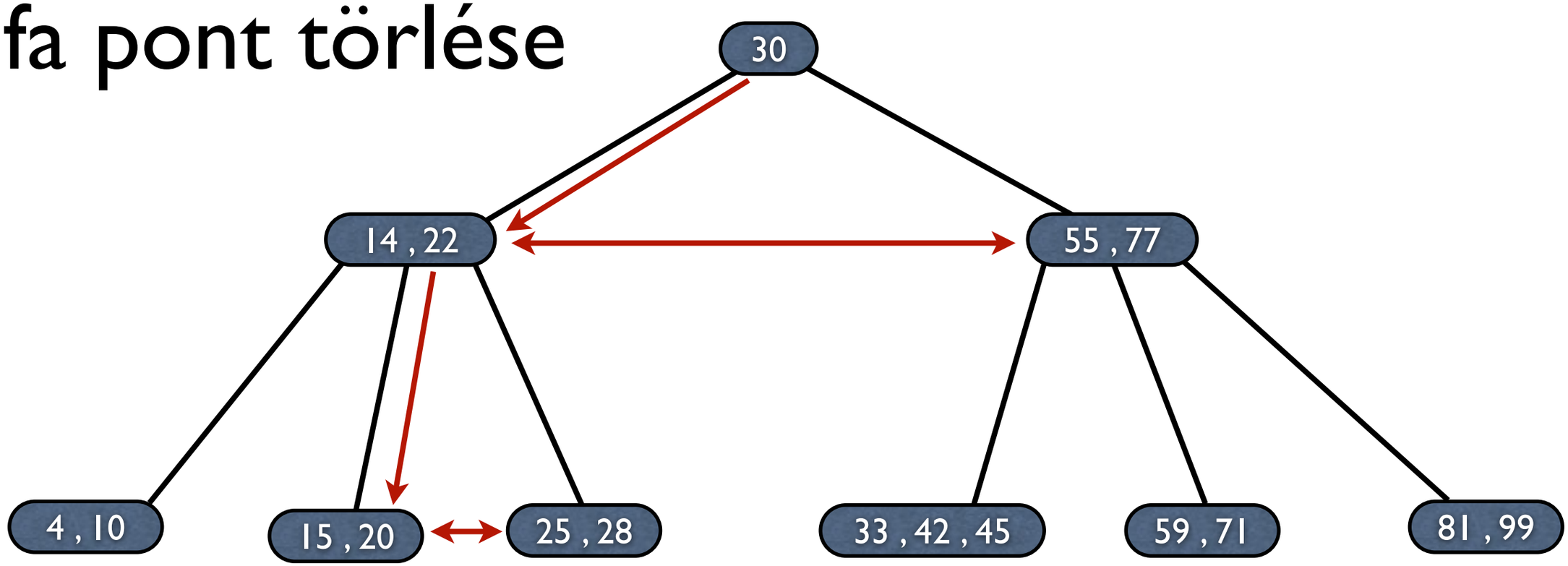
B-fa pont törlése



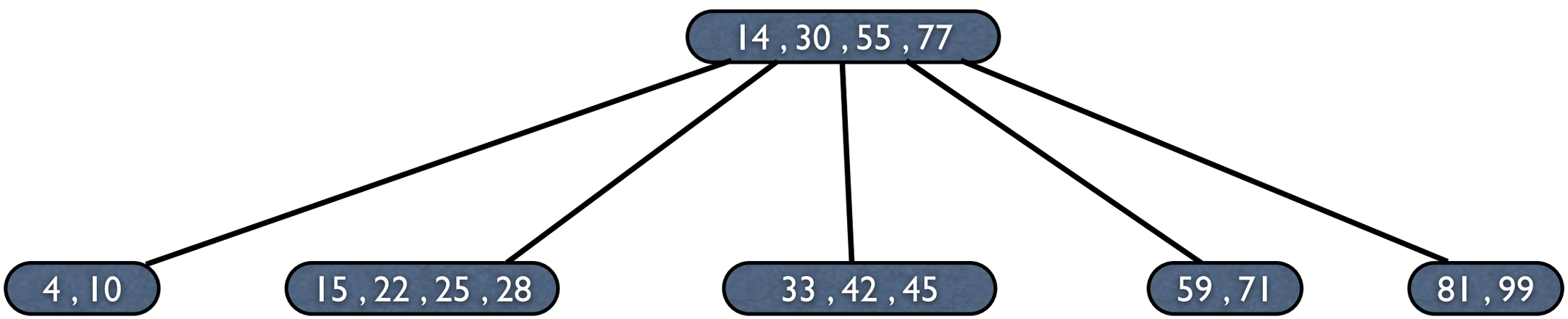
töröl(71)



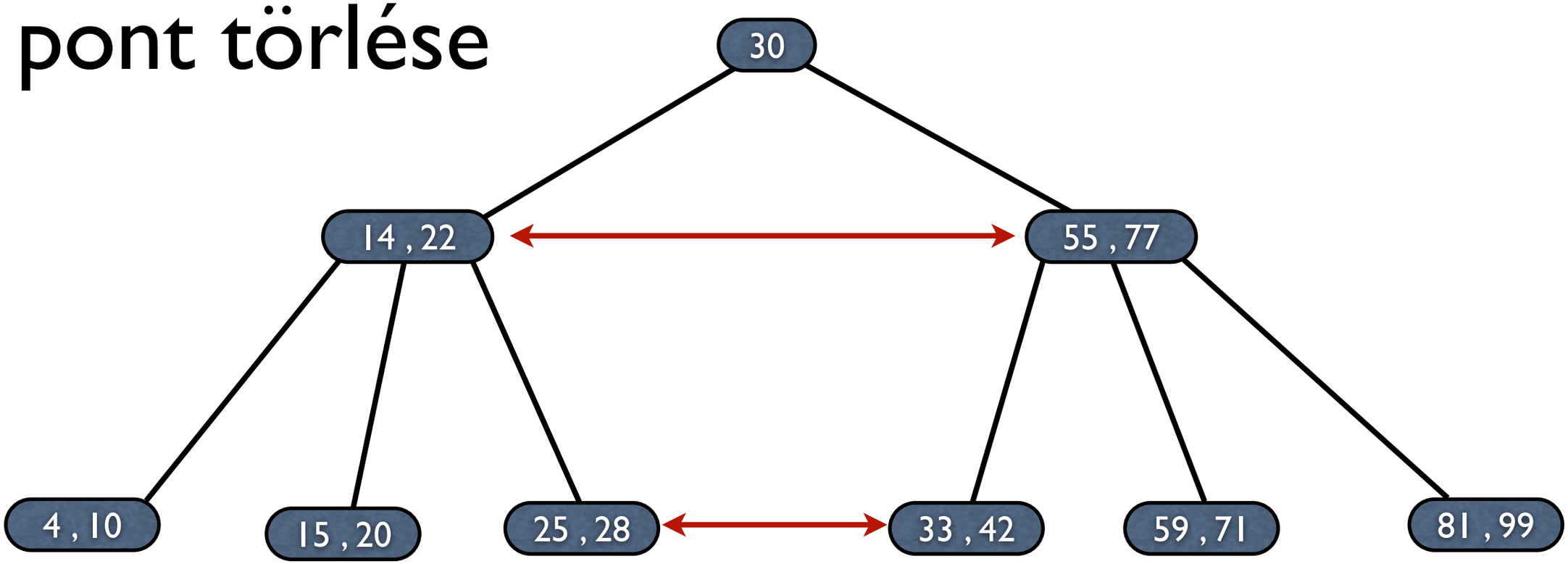
B-fa pont törlése



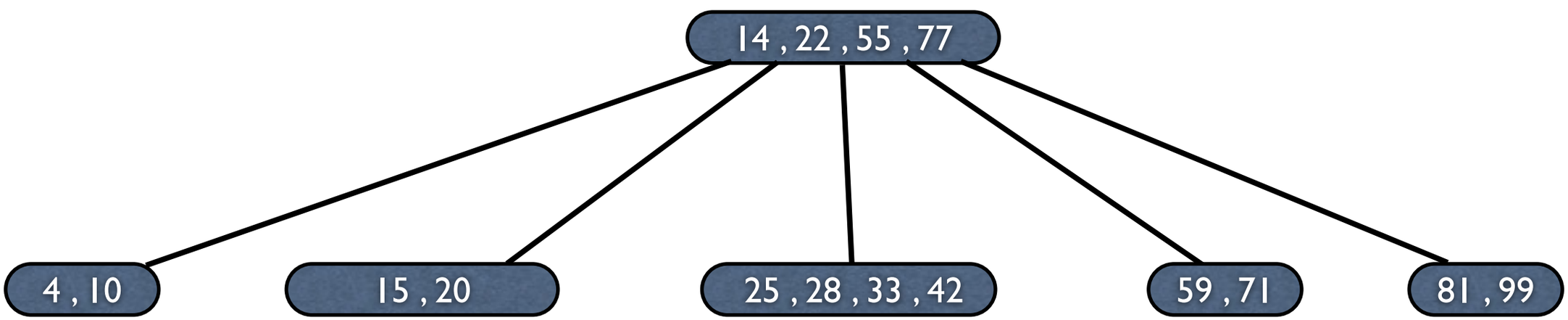
töröl(20)



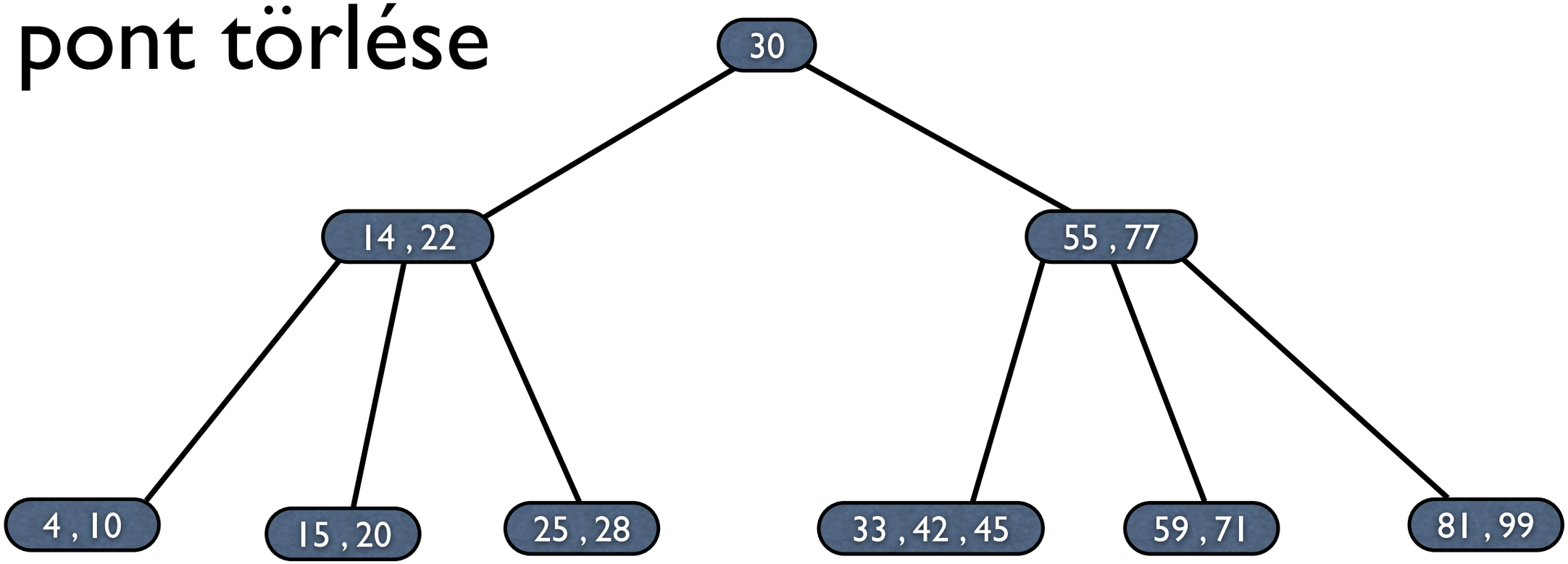
B-fa pont törlése



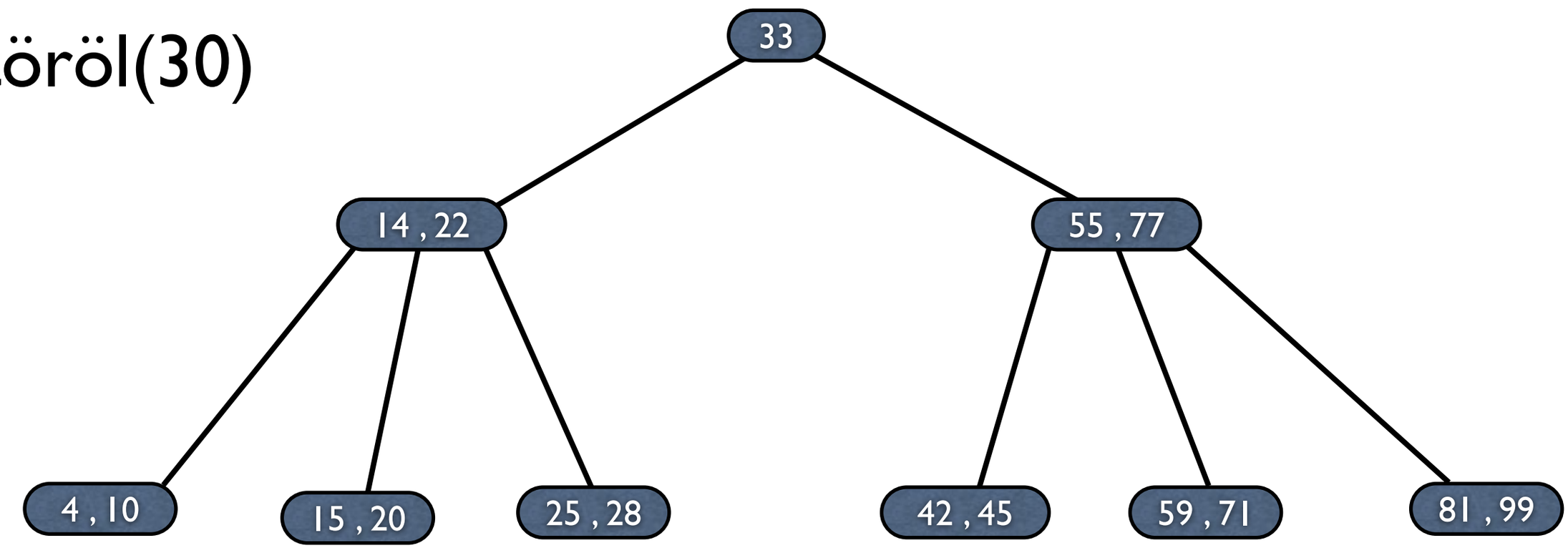
töröl(30)



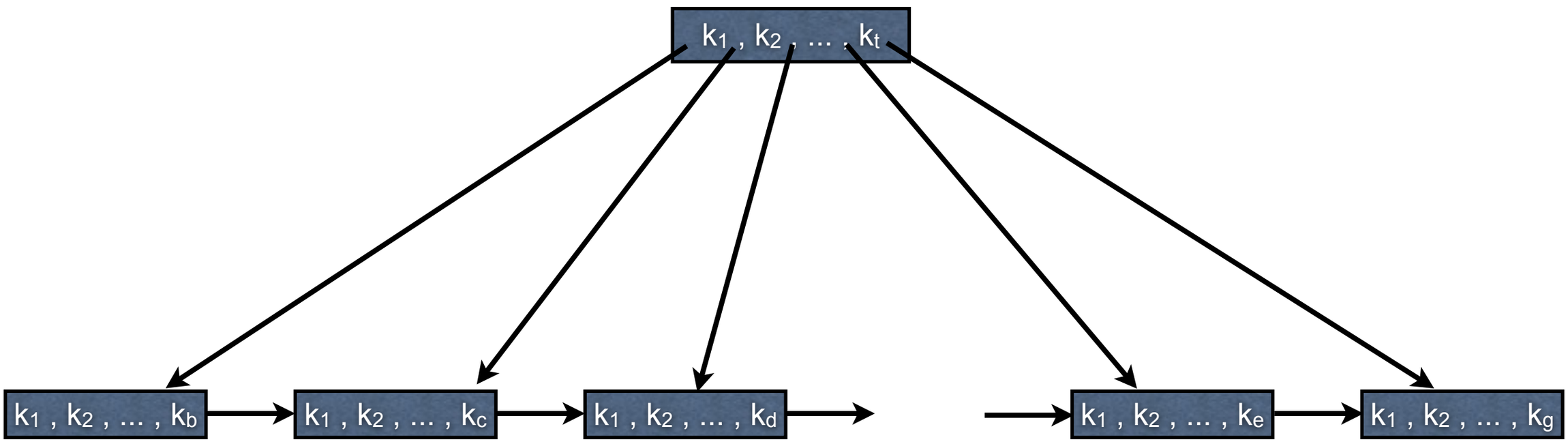
B-fa pont törlése



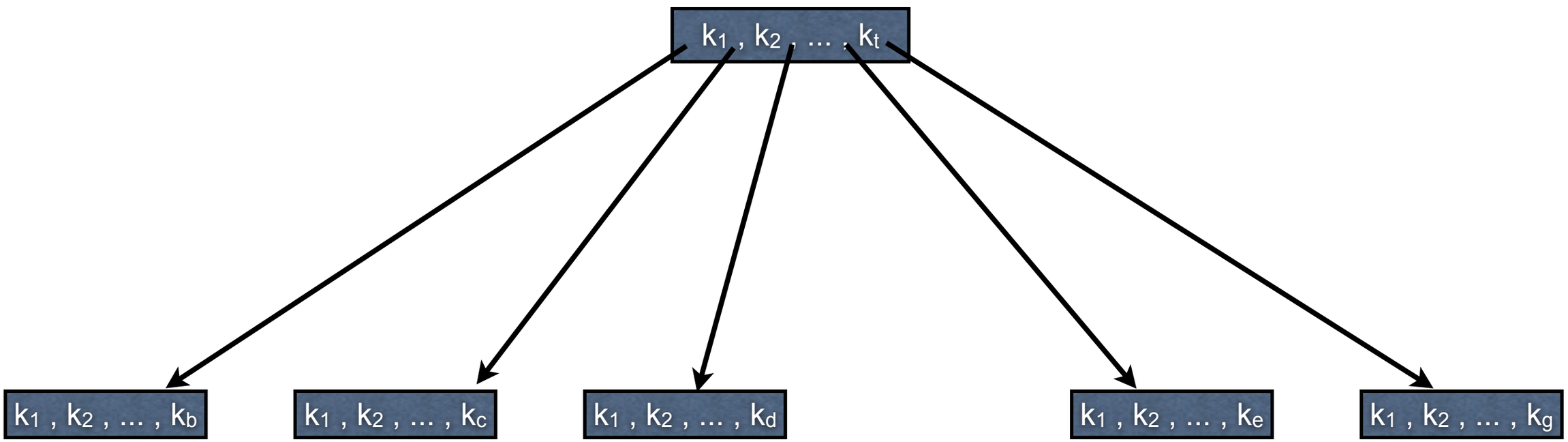
töröl(30)



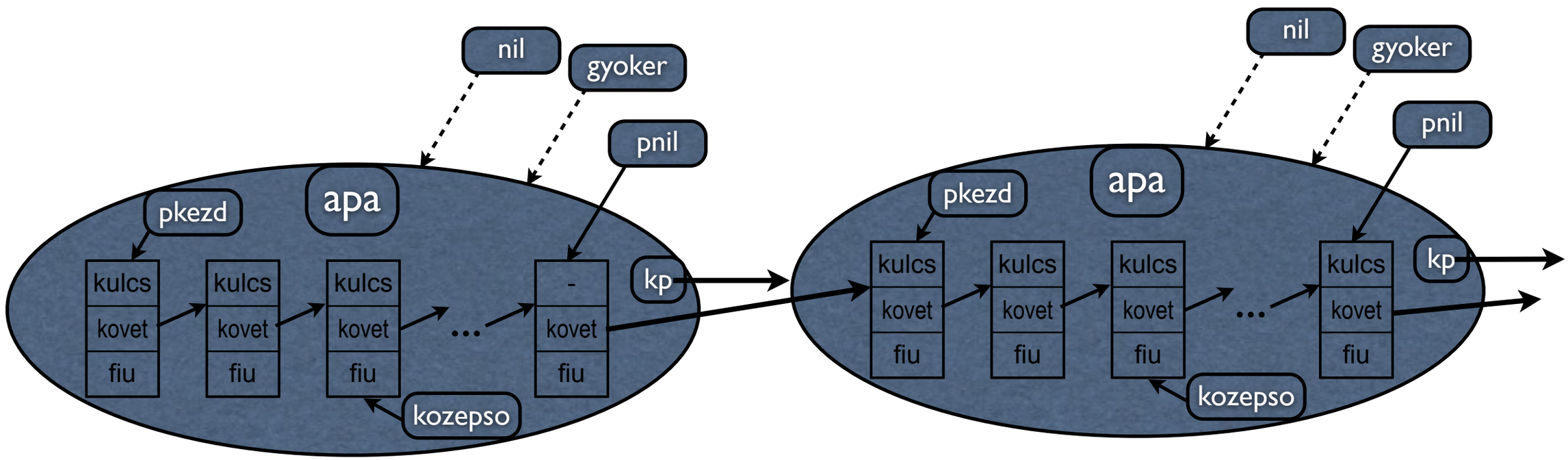
B+ fa



B+ fa



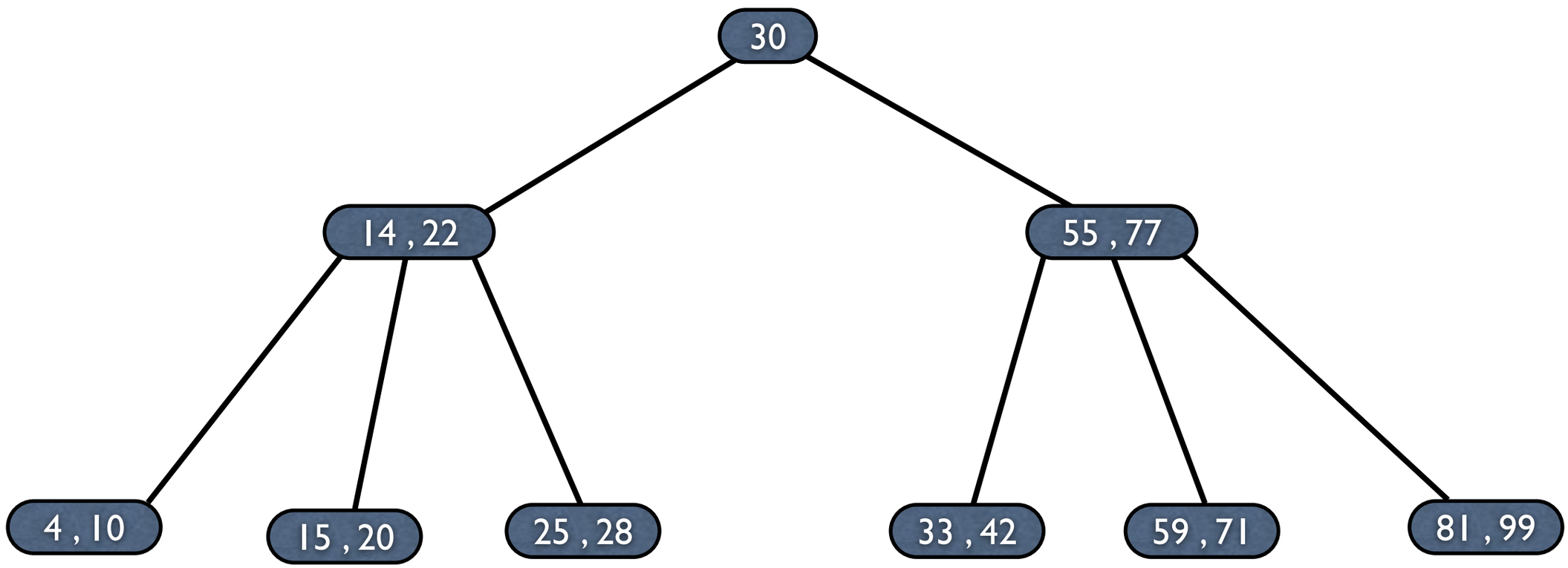
B+ fa


nil
gyoker
pnil
apa
pkezd
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
...
-
kovet
fiu
kp
kozepso
nil
gyoker
pnil
apa
pkezd
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
...
kulcs
kovet
fiu
kp
kozepso

2-3 fa

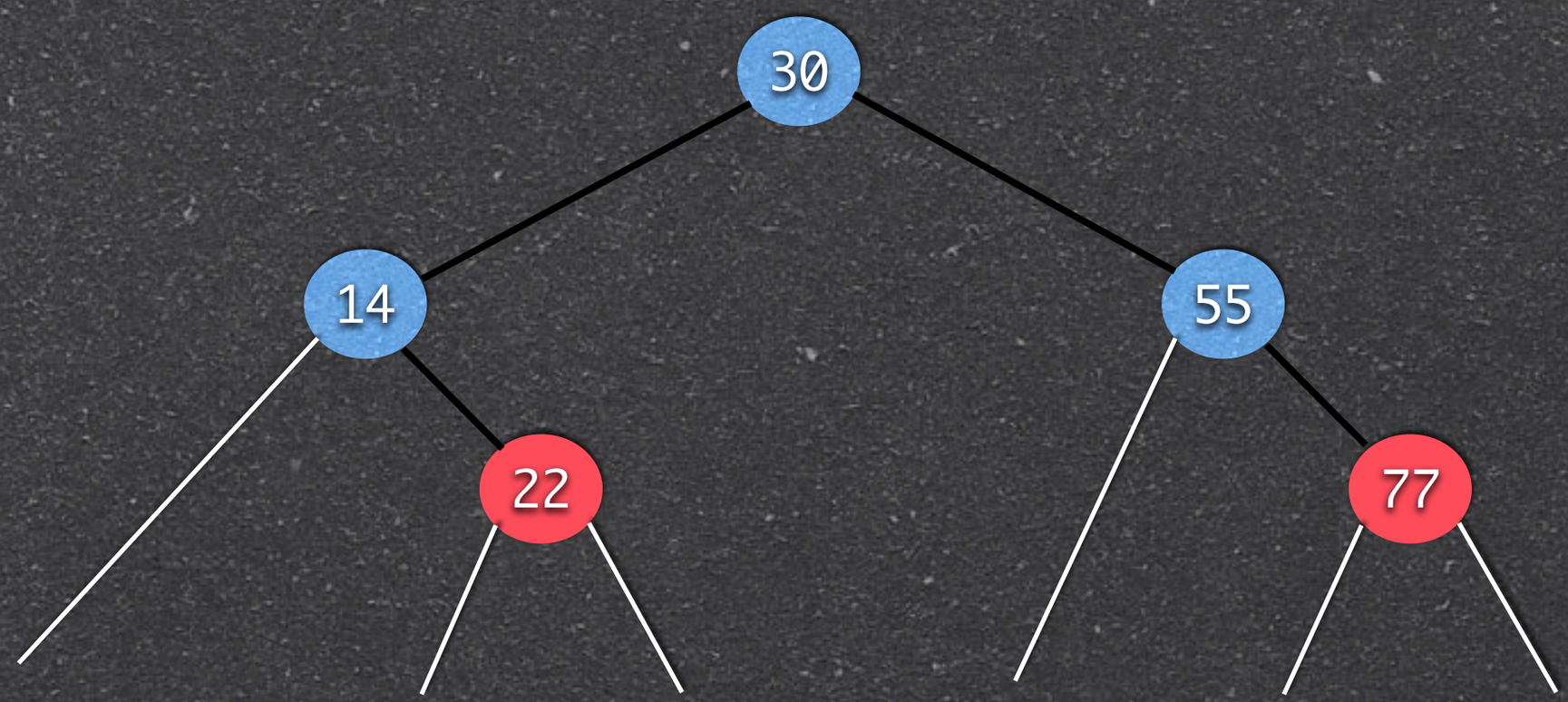
2-3 fa az 1 rangú B-fa

Példa



Piros-Fekete fák

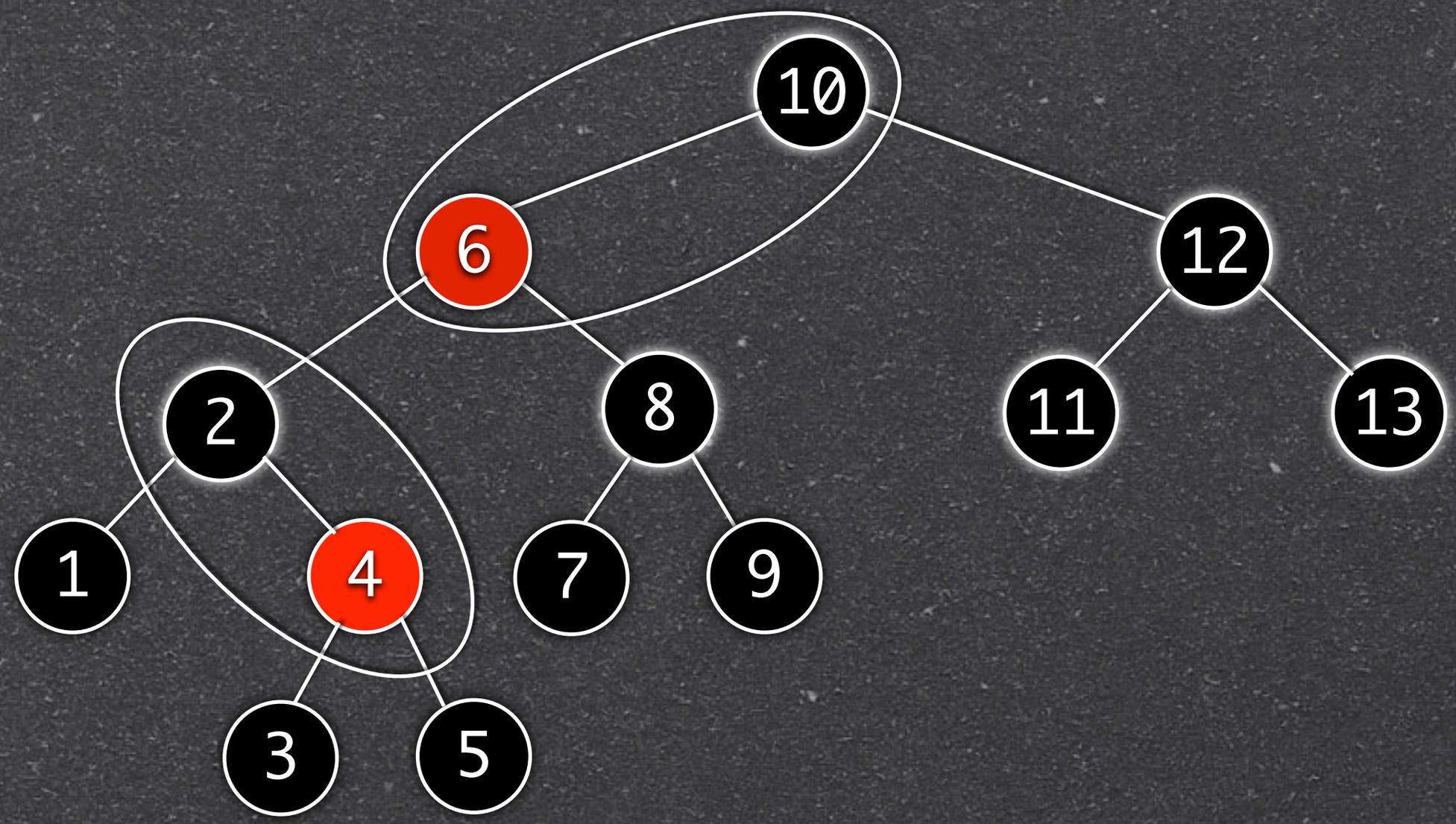
Példa



Piros-Fekete fák

A piros-fekete fa olyan bináris keresőfa, amelynek minden pontja egy extra bit információt tartalmaz, ez a pont színe, amelynek értékei: **PIROS** vagy **FEKETE**.

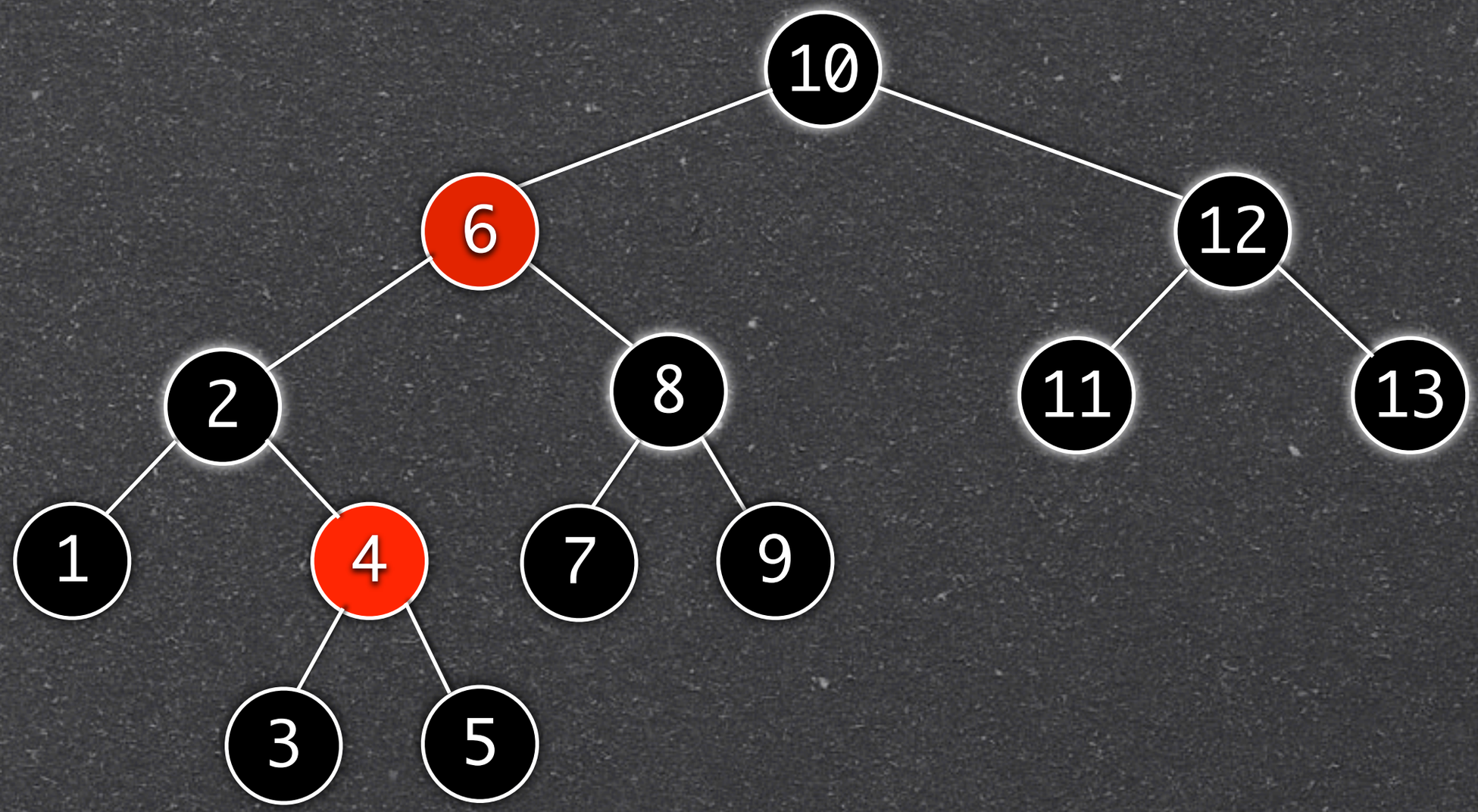
Tehát a fa minden pontja tartalmazza a szín, kulcs, bal, jobb és apa mezőket. Ha egy ponthoz tartozó fiú vagy apa nem létezik, akkor a megfelelo mező a NIL értéket tartalmazza. Úgy tekintjük, hogy az ilyen NIL mutató értékek a bináris keresőfa külső (levél) pontjaira mutatnak, míg a fa kulcsot tartalmazó pontjai a belső pontok. Megvalósítás során ezeket a külső pontokat egyetlen őrszem ponttal ábrázoljuk, amelyet NIL jelöl.


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

Piros-Fekete fák

A piros-fekete fák azok, amelyekre teljesülnek a következő tulajdonságok:

1. Minden pont színe vagy **PIROS** vagy **FEKETE**.
2. A gyökérpont színe **FEKETE**.
3. Minden levél (a NIL pontokat tekintjük levélnek) színe **FEKETE**.
4. Minden **PIROS** pontnak mindkét fia **FEKETE**.
5. Bármely pontból bármely levélig vezető úton ugyanannyi **FEKETE** pont van.


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

Egy x pont fekete-magasságának nevezzük az x pontból induló, levélig vezető úton található, x-en kívüli fekete pontok számát, és fm(x)-szel jelöljük.
Az 5. tulajdonság miatt a fekete-magasság jól definiált, mivel minden ilyen út azonos számú fekete pontot tartalmaz.
Egy piros-fekete fa fekete-magasságát a fa gyökérpontjának fekete-magasságaként definiáljuk.
A piros-fekete fákra teljesül a következő állítás:

<u>Tétel:</u> Bármely n belső pontot tartalmazó piros-fekete fa magassága legfeljebb 2log(n+1)

Biz(1). Teljes indukcióval igazolható, hogy a fa minden x gyökerű részfája legalább $2^{fm(x)}-1$ belső pontot tartalmaz.

Ha x magassága 0, akkor x levél (nil), tehát az x gyökerű részfának valóban 0 belső pontja van.

Tegyük fel, hogy x magassága pozitív, és két fia van. Mindkét fiú fekete-magassága vagy fm(x), vagy fm(x)-1 attól függően, hogy a színe piros vagy fekete.

Mivel x fiainak magassága kisebb, mint x magassága, így az indukciós feltevés alapján mindkét részfa legalább $2^{fm(x)-1} - 1$ belső pontot tartalmaz. Tehát az x gyökerű részfa belső pontjainak száma legalább $(2^{fm(x)-1}-1) + (2^{fm(x)-1}-1) + 1 = 2^{fm(x)}-1$.

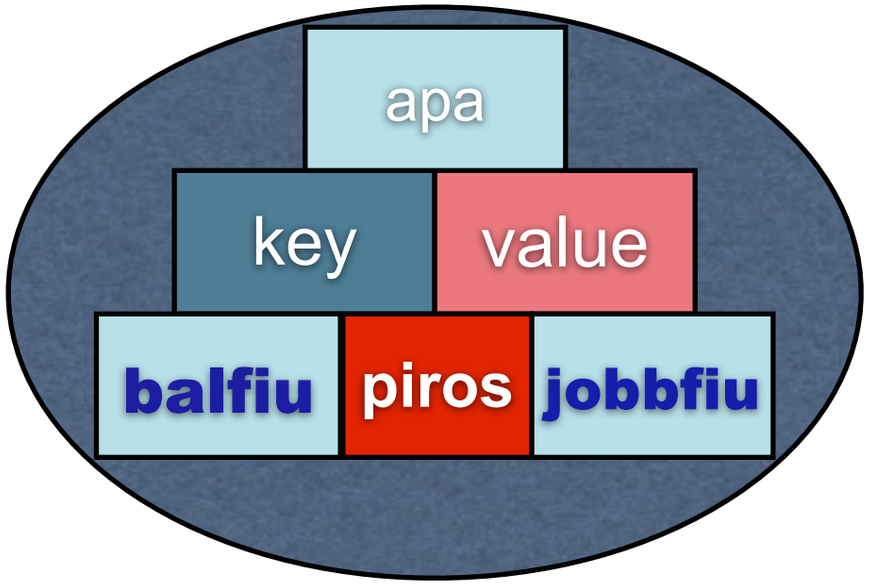
Legyen x magassága h.
A 4. tulajdonság szerint minden a gyökértől levélig haladó út, legalább feleannyi fekete pontot tartalmaz, mint ezen út pontjainak száma, nem számítva a gyökeret.
Tehát a gyökér fekete-magassága legalább h/2, így $n \geq 2^{h/2} - 1$.
Tehát azt kapjuk, hogy $\log(n+1) \geq h/2$, azaz $h \leq 2\log(n+1)$. □

Következésképpen a keresőfa műveletek időigénye $O(\log n)$, ahol n a pontok száma.
Azt kell megvizsgálni a piros-fekete fa tulajdonságok karbantartásának mekkora az időigénye.

Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

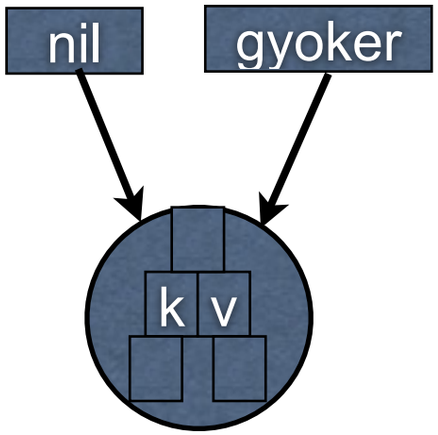
```
struct fapont{
    int key, value ;
    boolean piros ;
    fapont bal, jobb, apa ;
}
```



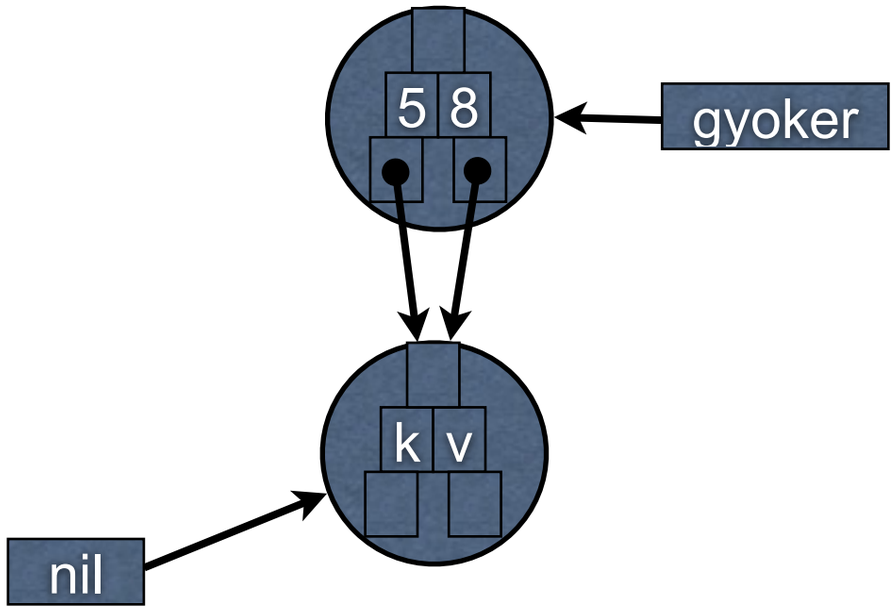
Piros-fekete fa tárolása

első megközelítés, a gyakorlatban nem így van!

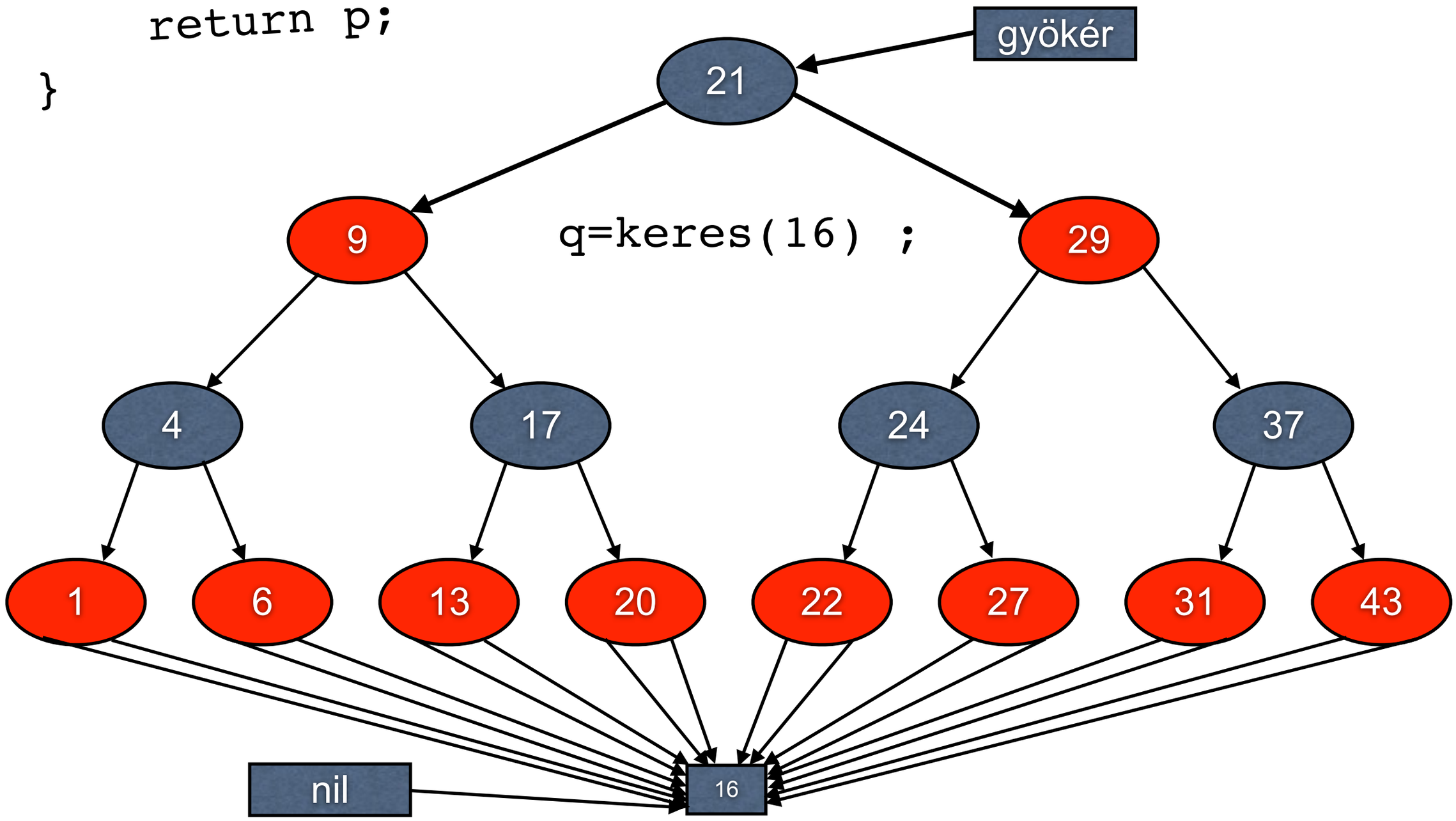
```
public class fa {
   int elemszam ;
   struct fapont{
      int key, value ;
      boolean piros ;
      fapont bal, jobb, apa ;
   }
   fapont nil, gyoker ;
   void fa() {
      fapont q = new fapont() ;
      nil=q ;
      gyoker=q ;
      elemszam=0 ;
      q.piros=false ;
   }
   boolean beszur(int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



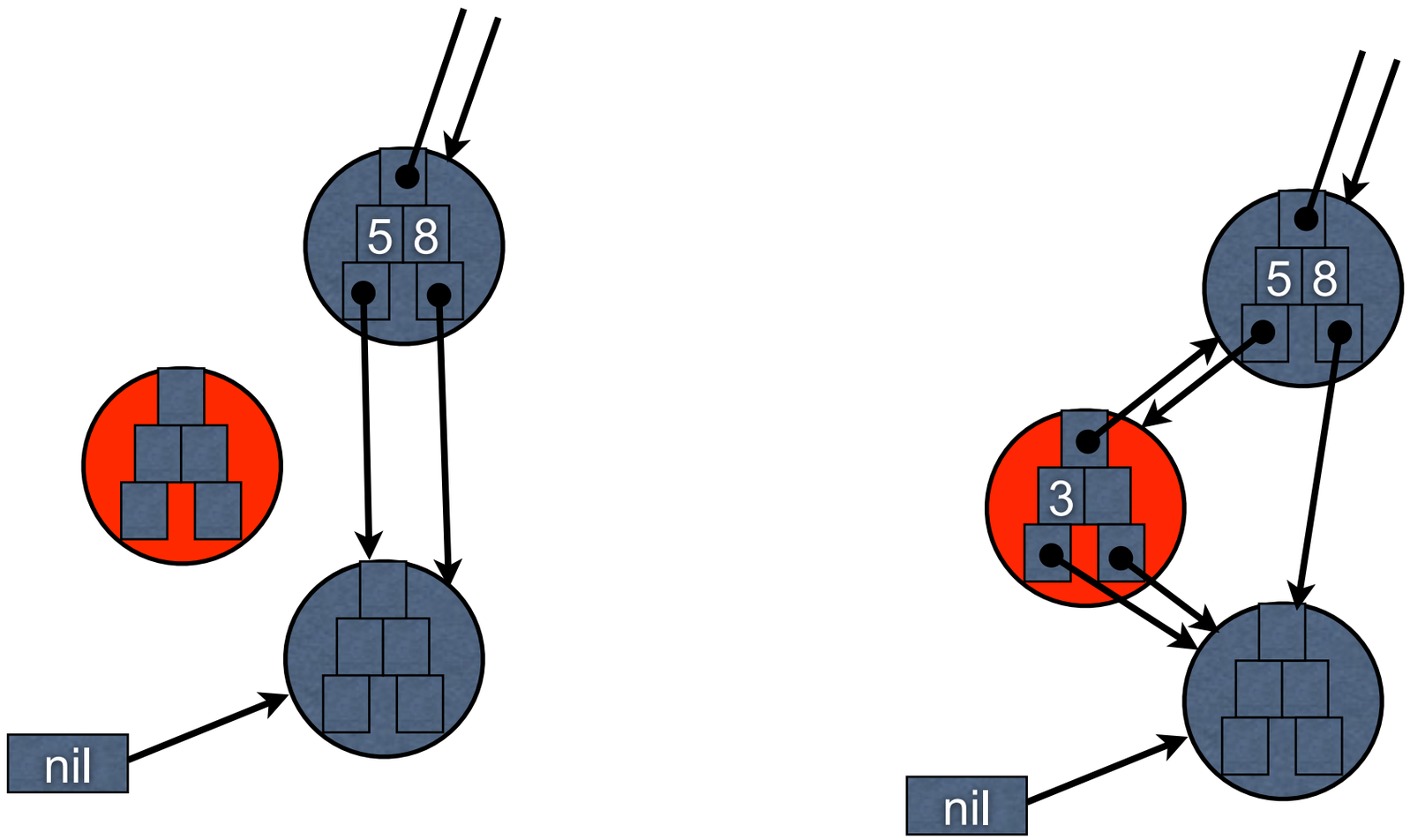
beszur(5, 8)



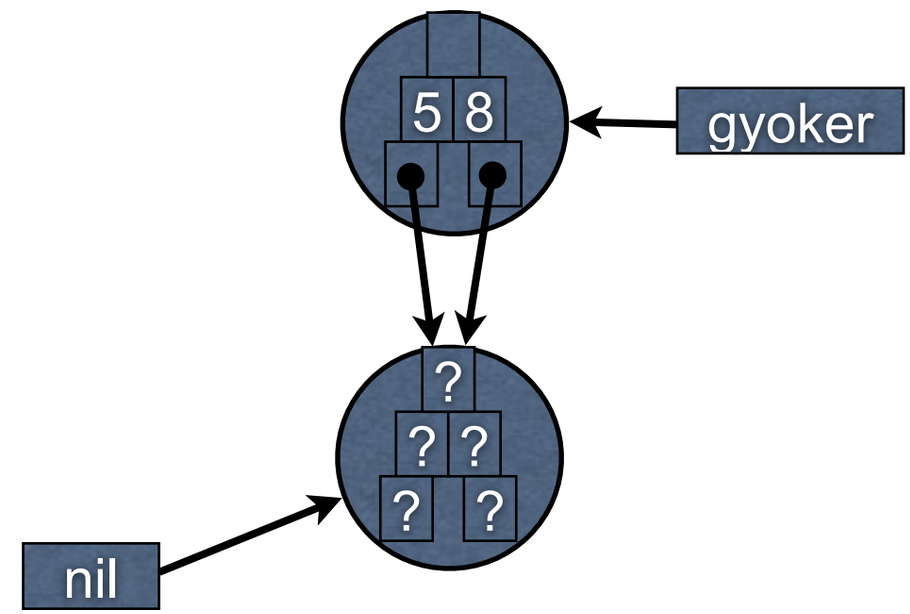
```
fapont keres(int x) {
    fapont p=gyoker ;
    nil.key=x ;
    while (p.key!=x) p=x<p.key?p.bal:p.jobb ;
    return p;
}
```



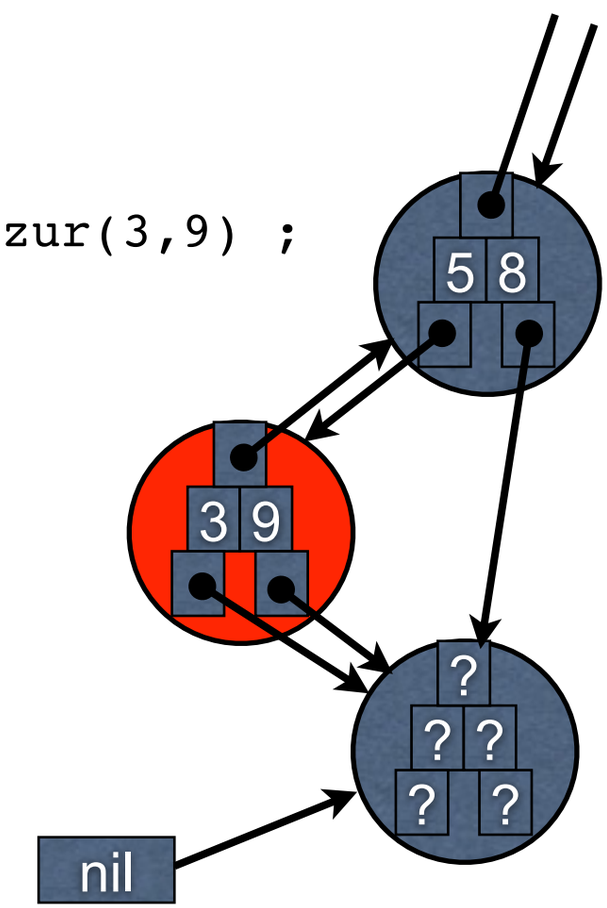
q=keres(16) ;


5 8
nil
5 8
3
nil

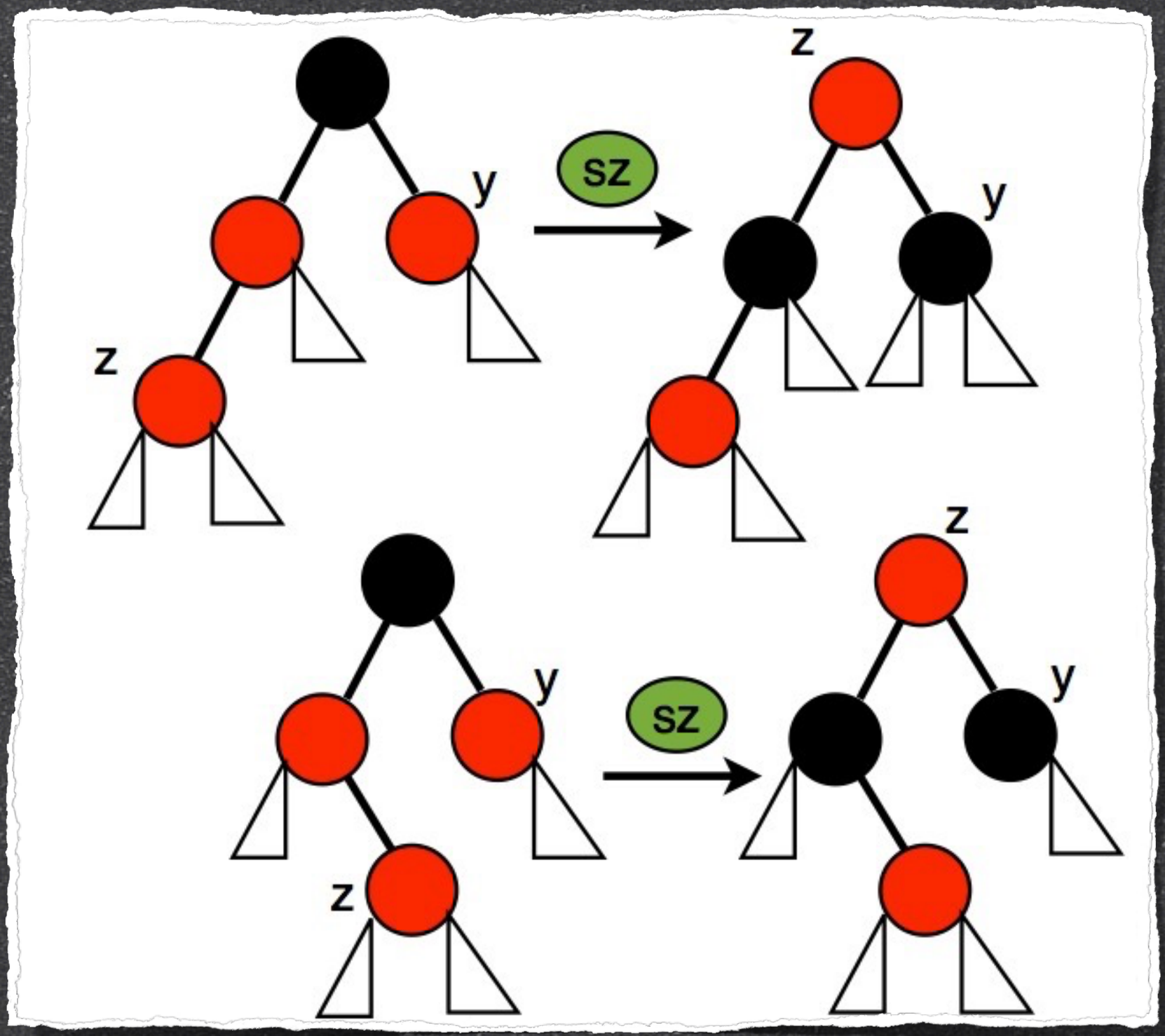
```
boolean beszur(int x, int v) {
      fapont p,papa ;
      p=gyoker ;
      papa=nil ;
      while (p!=nil && p.key!=x) {
         papa=p ;
         if (x<p.key) p=p.bal;
         else p=p.jobb;
      }
      if (p==nil) {
         fapont ujpont = new fapont();
         if (gyoker==p) gyoker=ujpont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



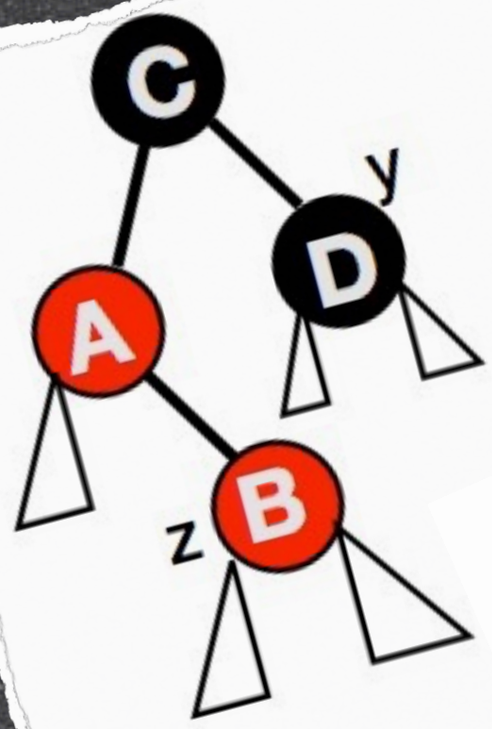
beszur(3,9) ;

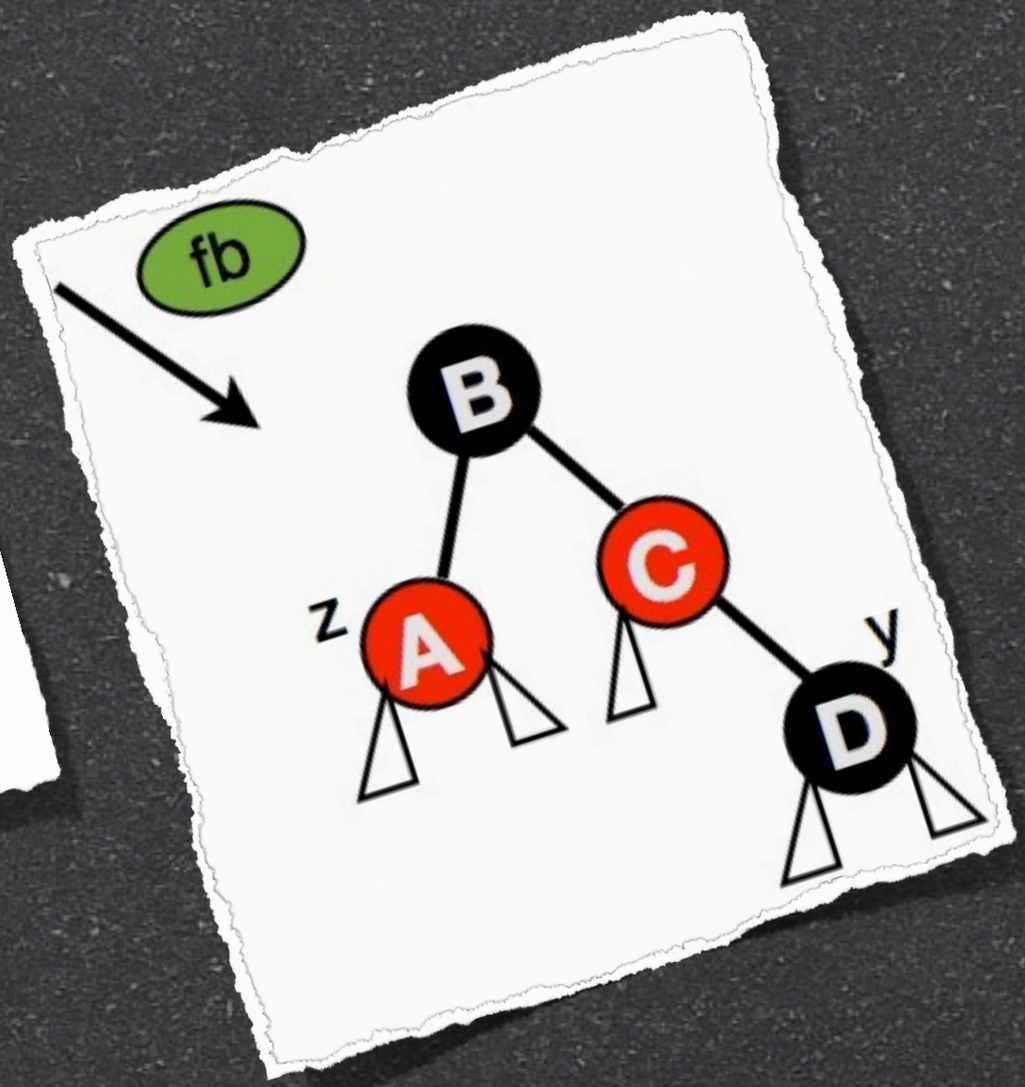


Beszúrás utáni javítás I.



Beszúrás utáni javítás II.

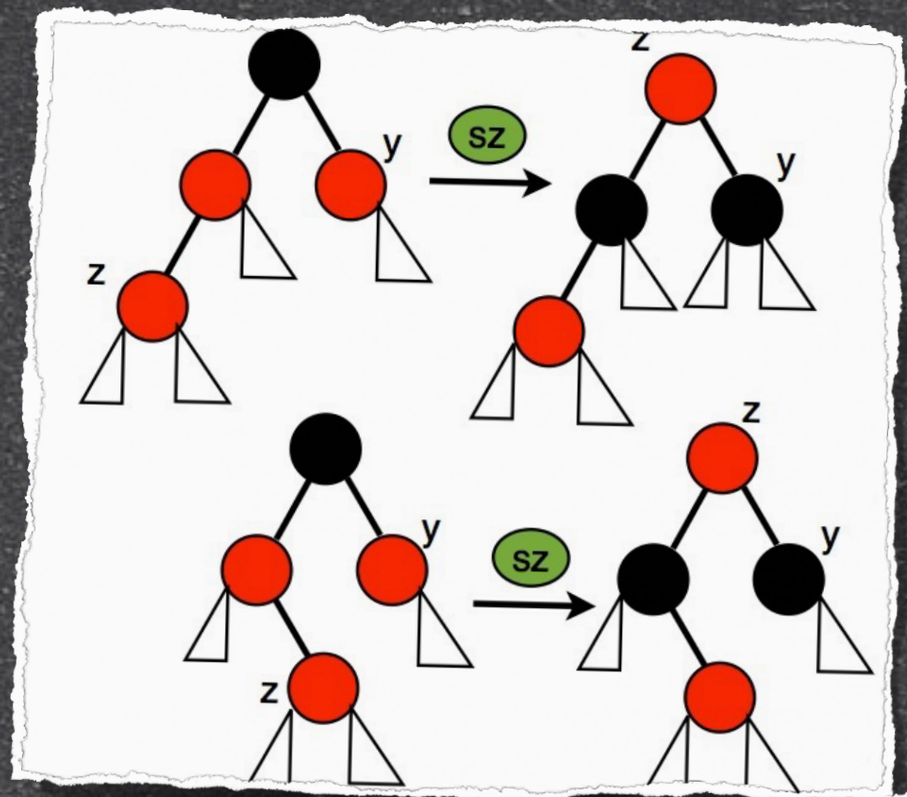


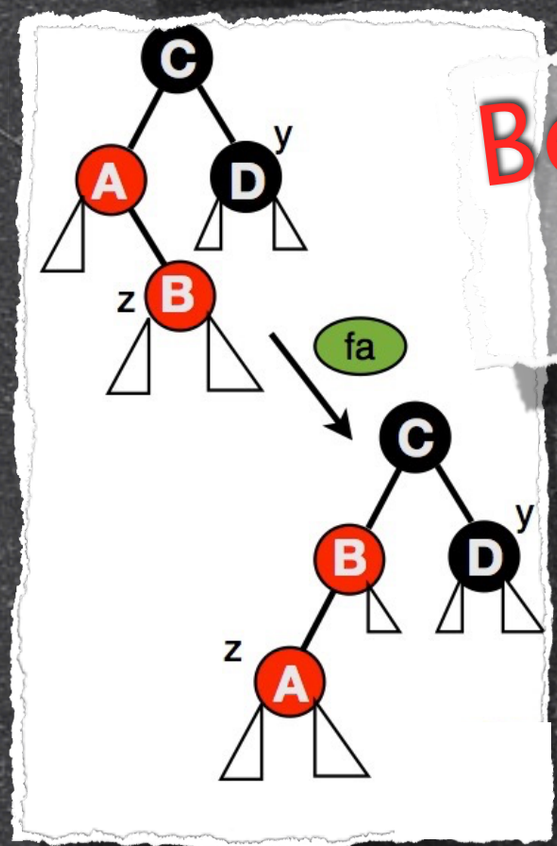


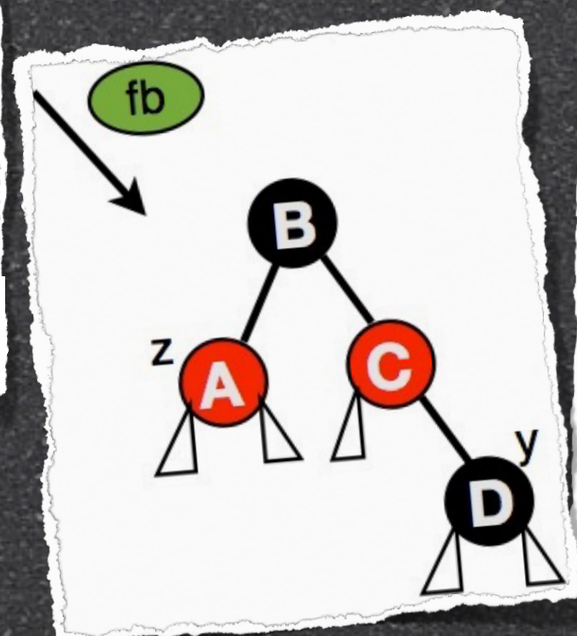
```
void balraforgat(fapont x) {
    fapont y ;
    y=x.jobb ;
    x.jobb=y.bal ;
    if (x.jobb!=nil) x.jobb.apa=x;
    y.apa=x.apa;
    if (y.apa==nil) gyoker=y ;
    else {
      if (x==y.apa.bal) y.apa.bal=y ;
      else y.apa.jobb=y ;
    }
    y.bal=x ;
    x.apa=y ;
}
```

0(1)

Beszúrás utáni javítás








```
void pf-beszurjavit(fapont z) {
  while (z.apa.piros) {
    if (z.apa==z.apa.apa.bal) {
      y=z.apa.apa.jobb ;
      if (y.piros) {
        z.apa.piros=false ;
        z.apa.apa.piros=true ;
        y.piros=false ;
        z=z.apa.apa ;
      } else {
        if (z==z.apa.jobb) {
          z=z.apa ;
          balraforgat(z) ;
        }
        z.apa.piros=false ;
        z.apa.apa.piros=true ;
        jobbraforgat(z.apa.apa) ;
      }
    } else {
      //szimmetrikus eset
    }
  }
  gyoker.piros=false ;
}
```

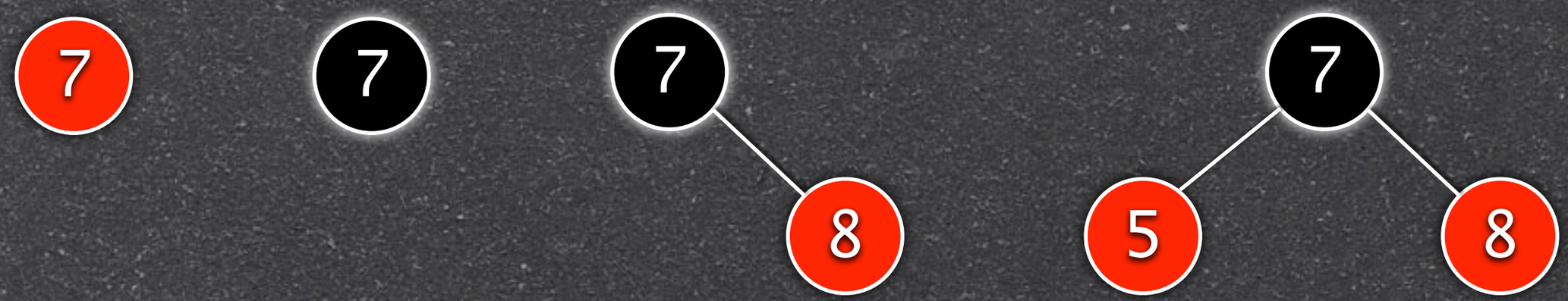
sz

fa

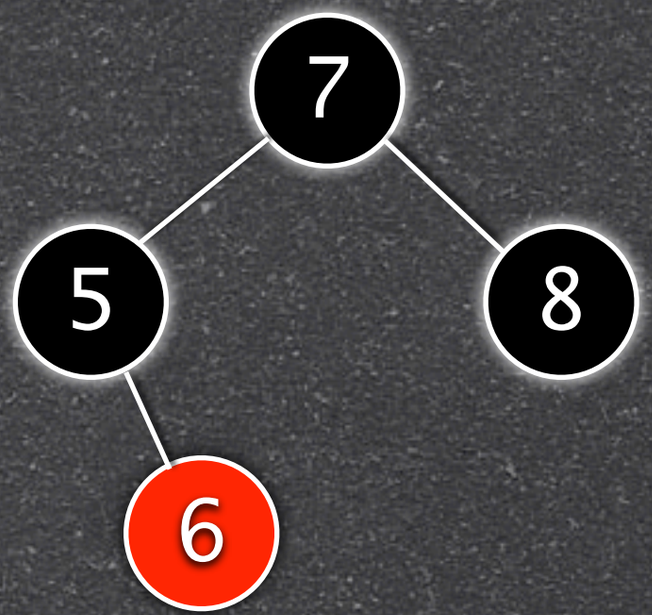
fb

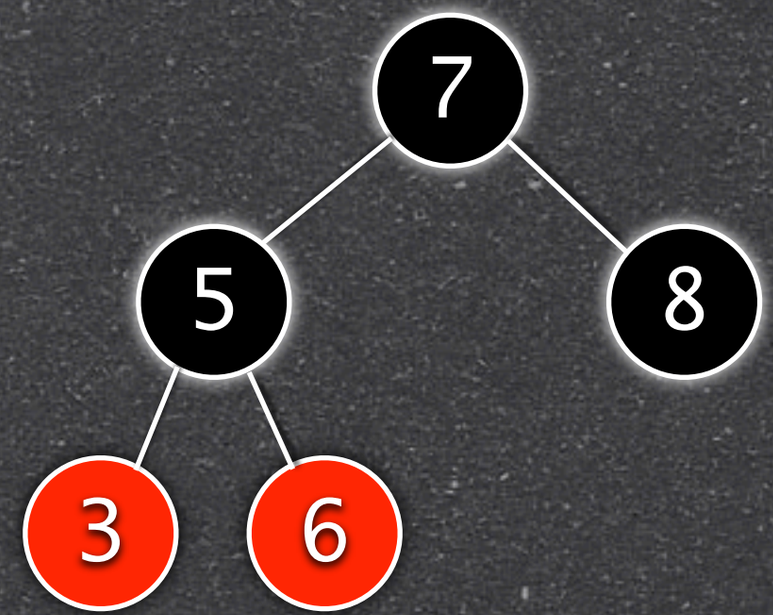
O(logn)

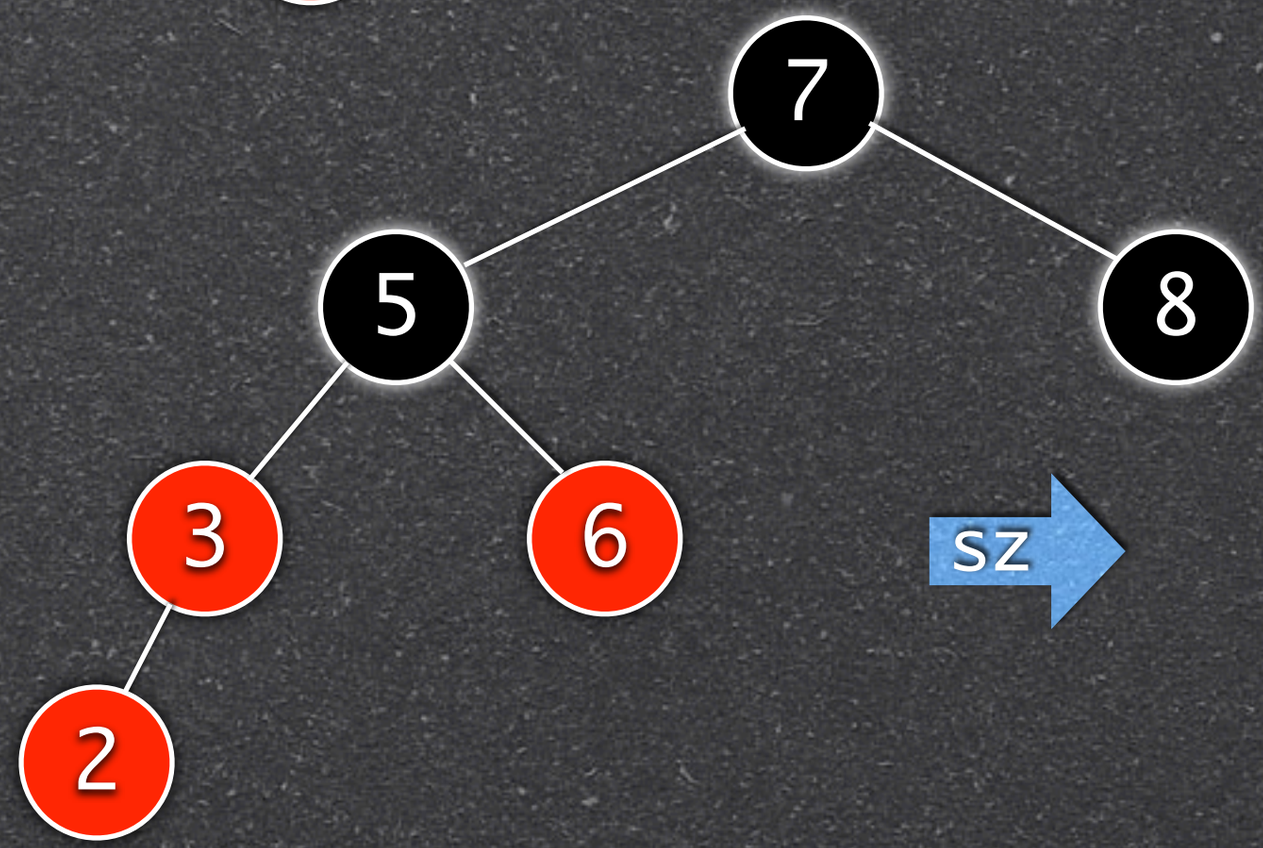
példa: 7, 8, 5, 6

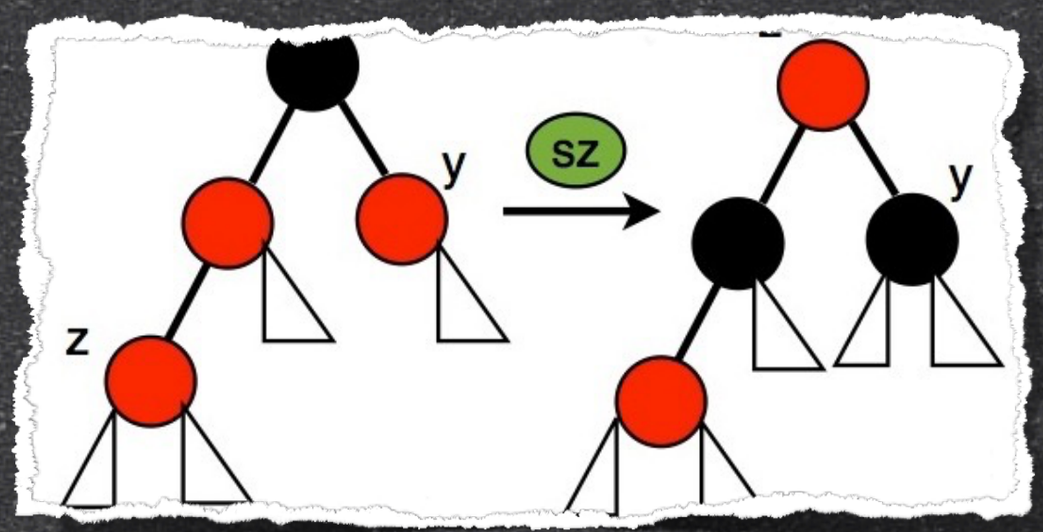


példa: 7, 8, 5, 6, 3, 2

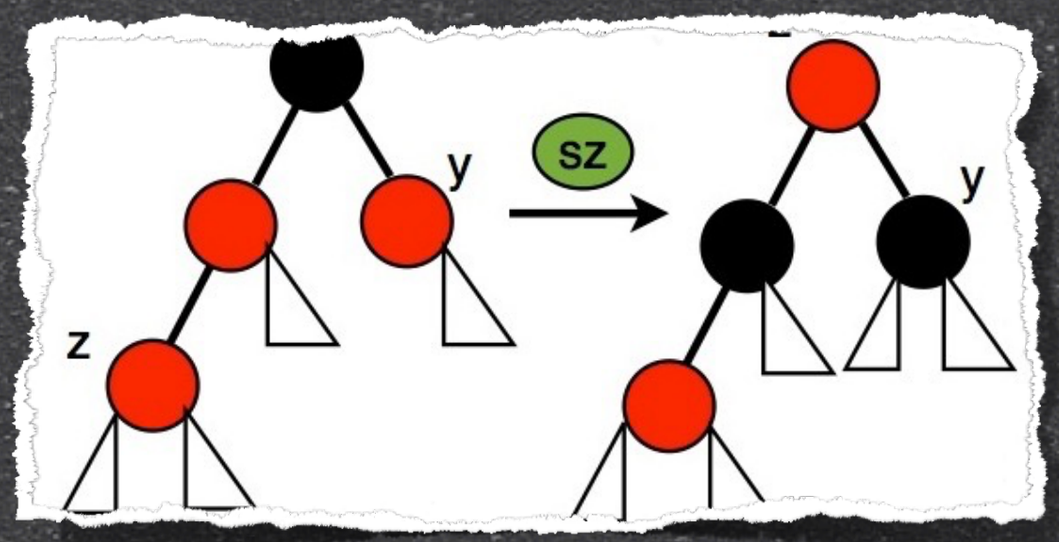


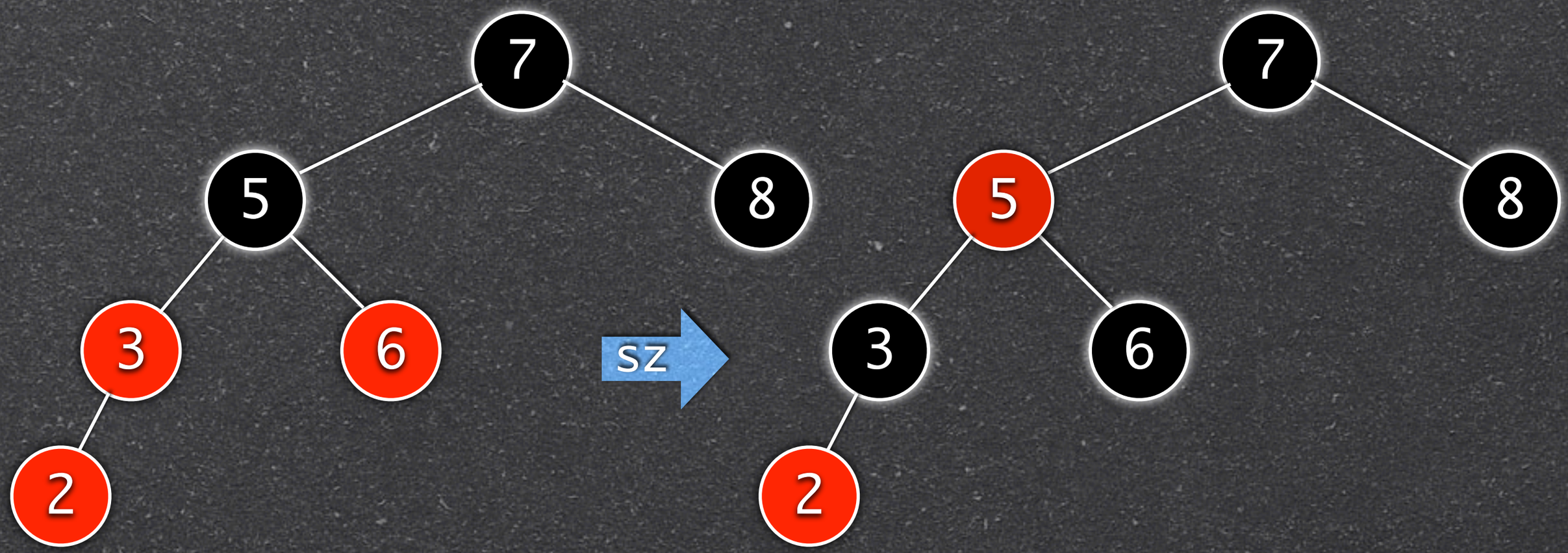




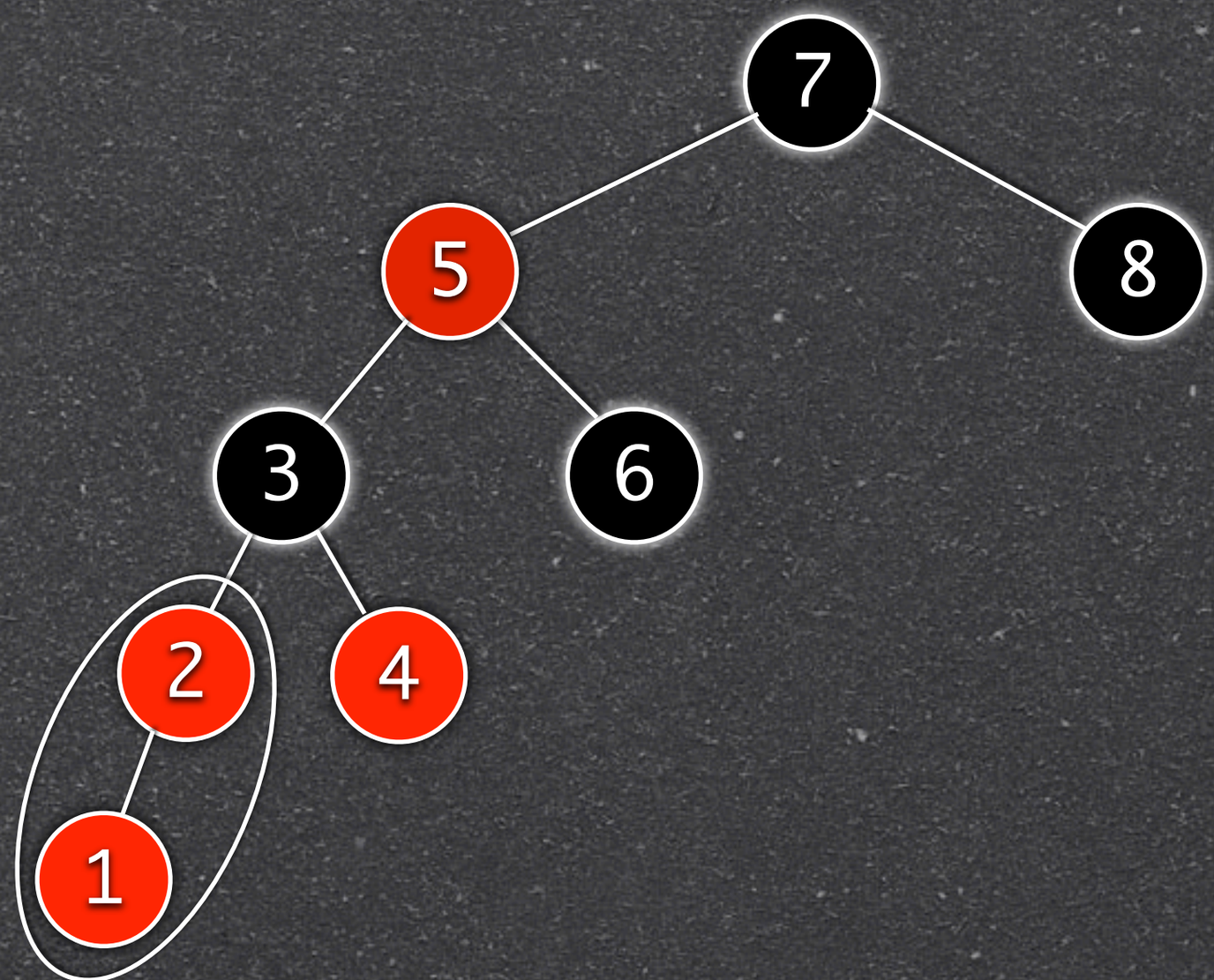


példa: 7, 8, 5, 6, 3, 2

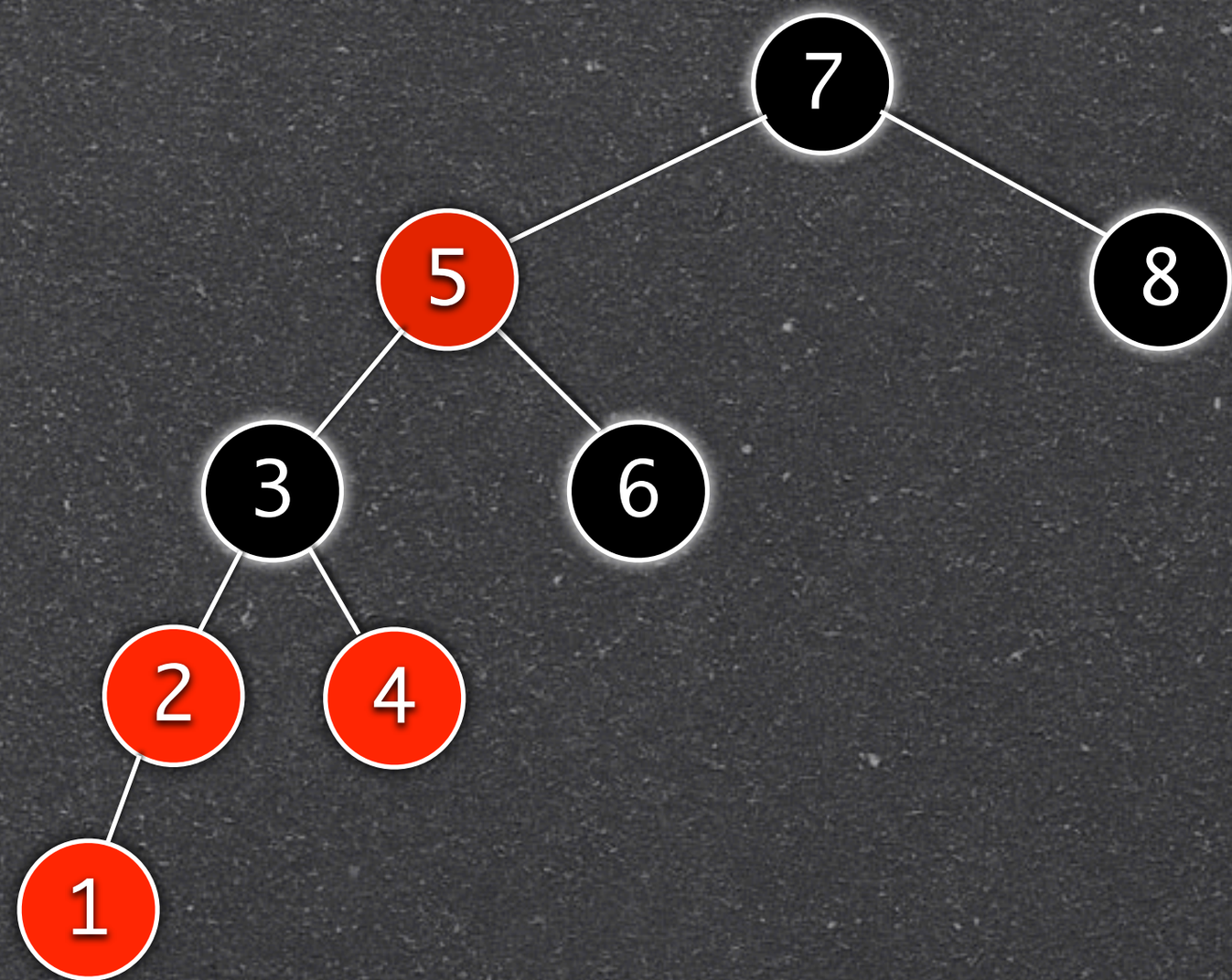


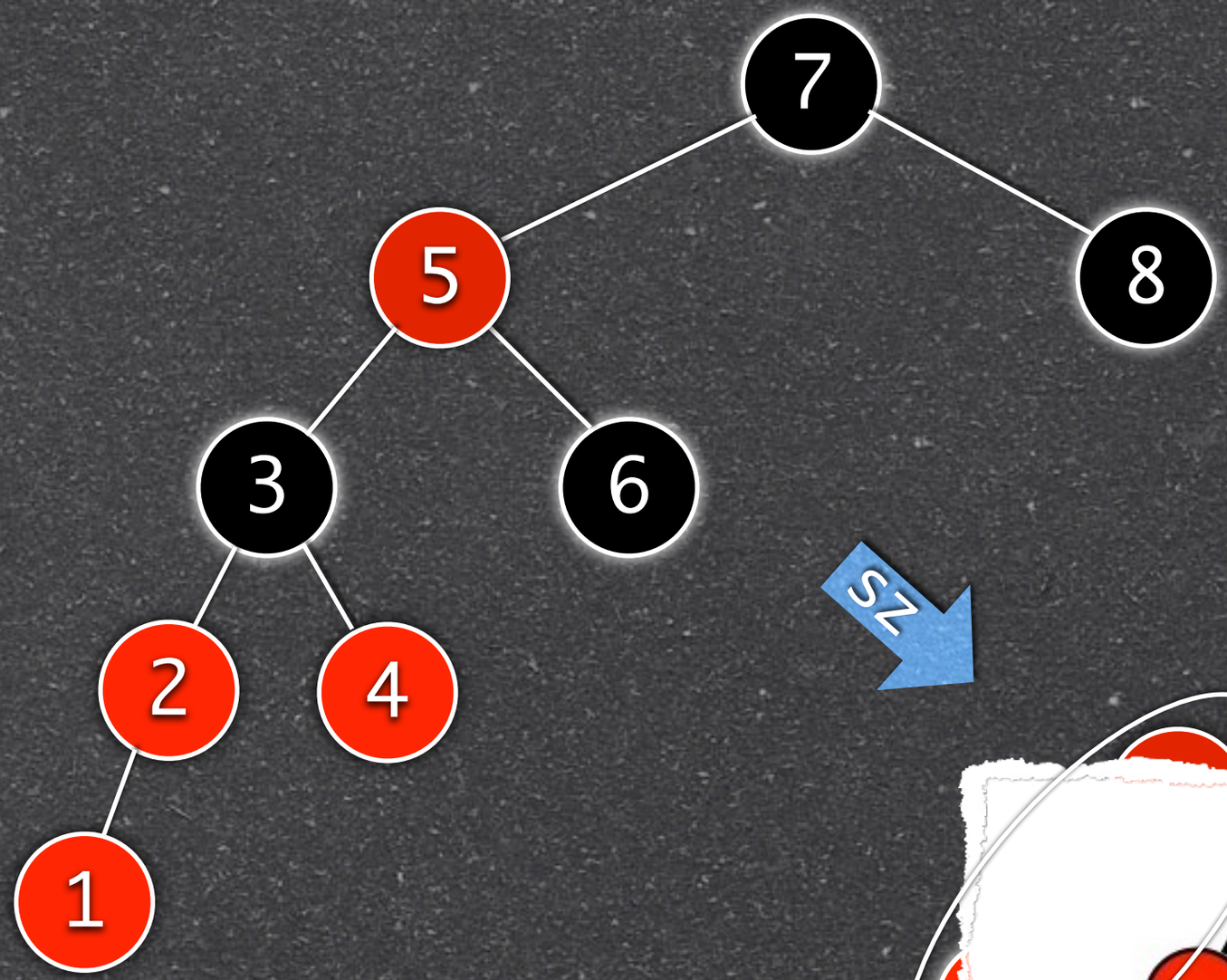


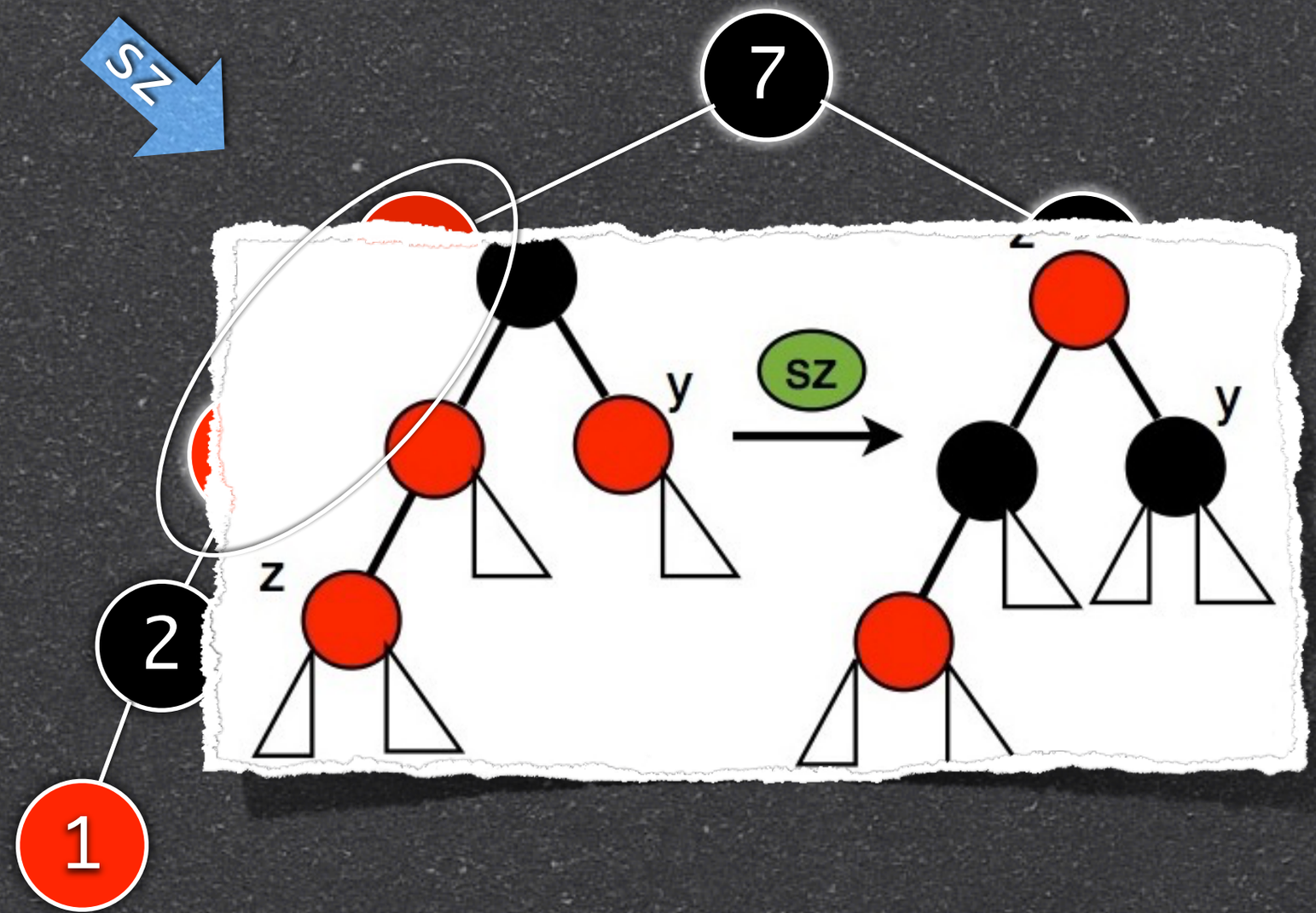
példa: 7, 8, 5, 6, 3, 2, 4, 1

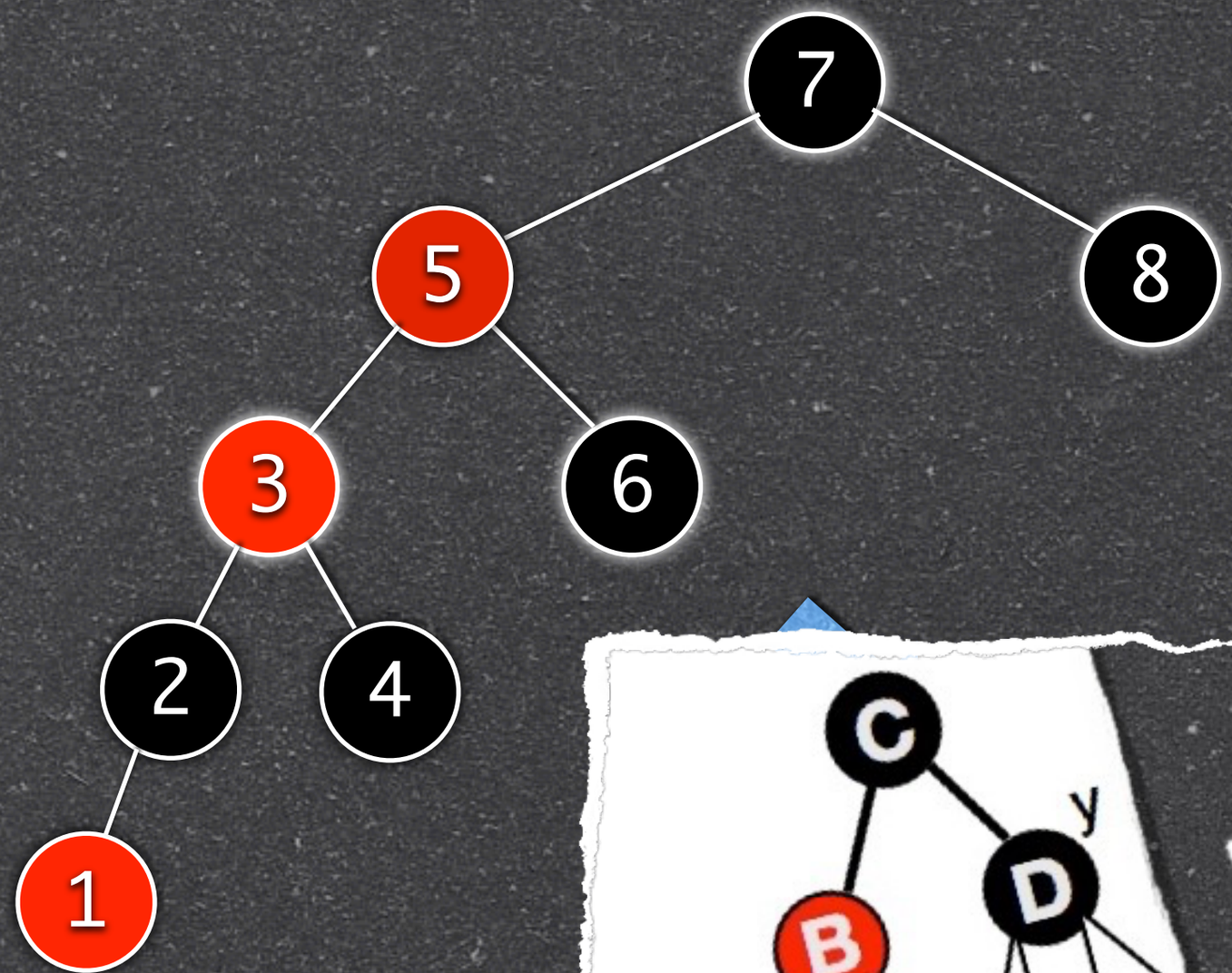


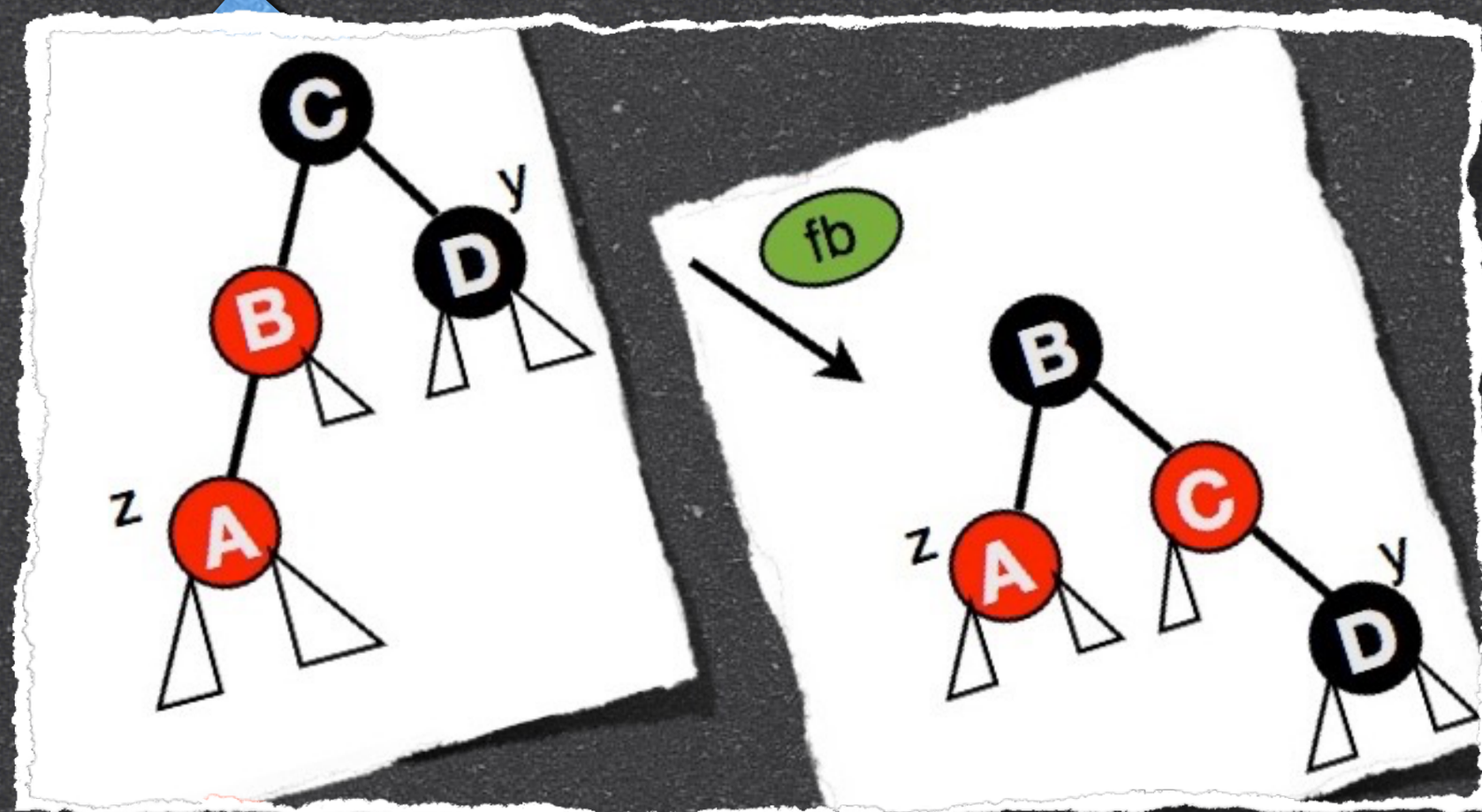
példa: 7, 8, 5, 6, 3, 2, 4, 1



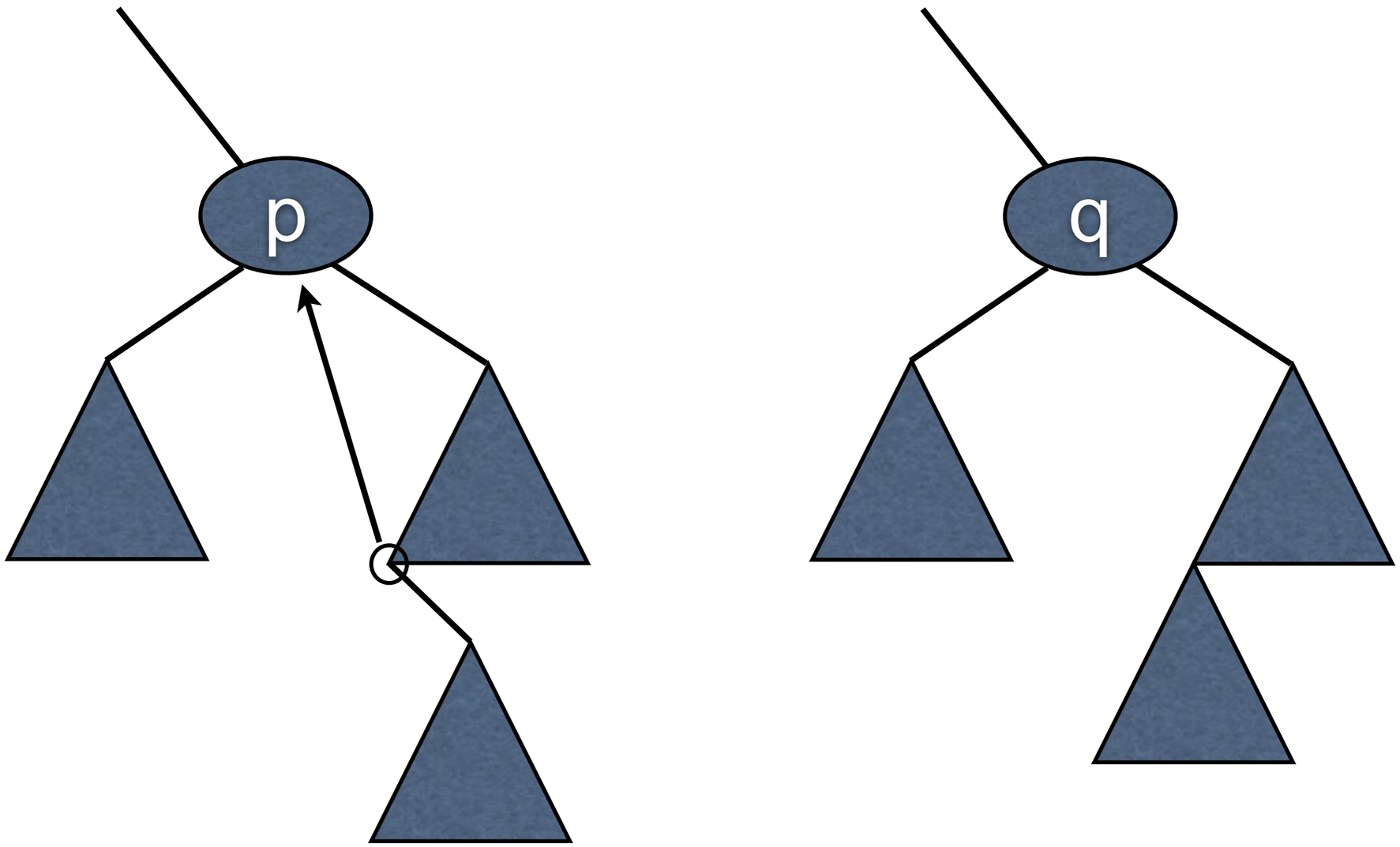

7
5
8
3
6
2
4
1


SZ
7
2
1
z
y
SZ
z
y


7
5
8
3
6
2
4
1


C
y
D
B
z
A
fb
B
C
z
A
y
D

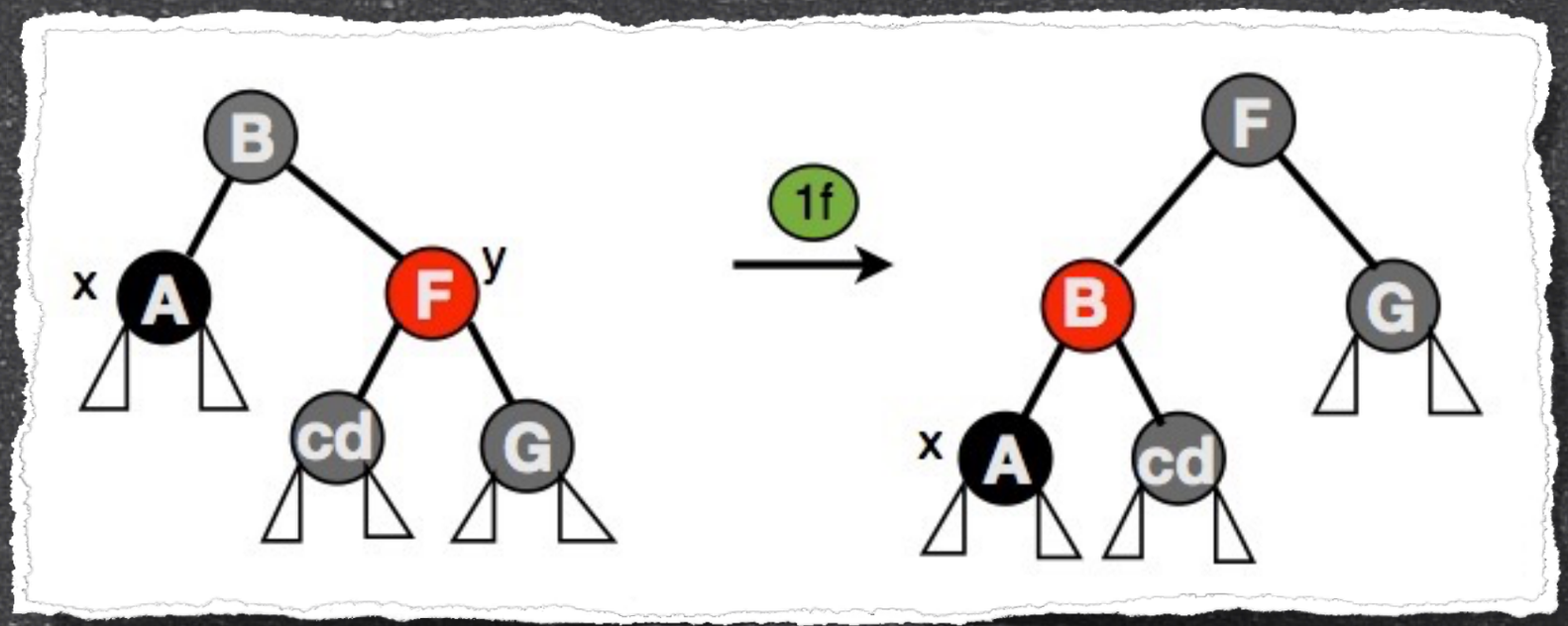
Törlés piros-fekete fából

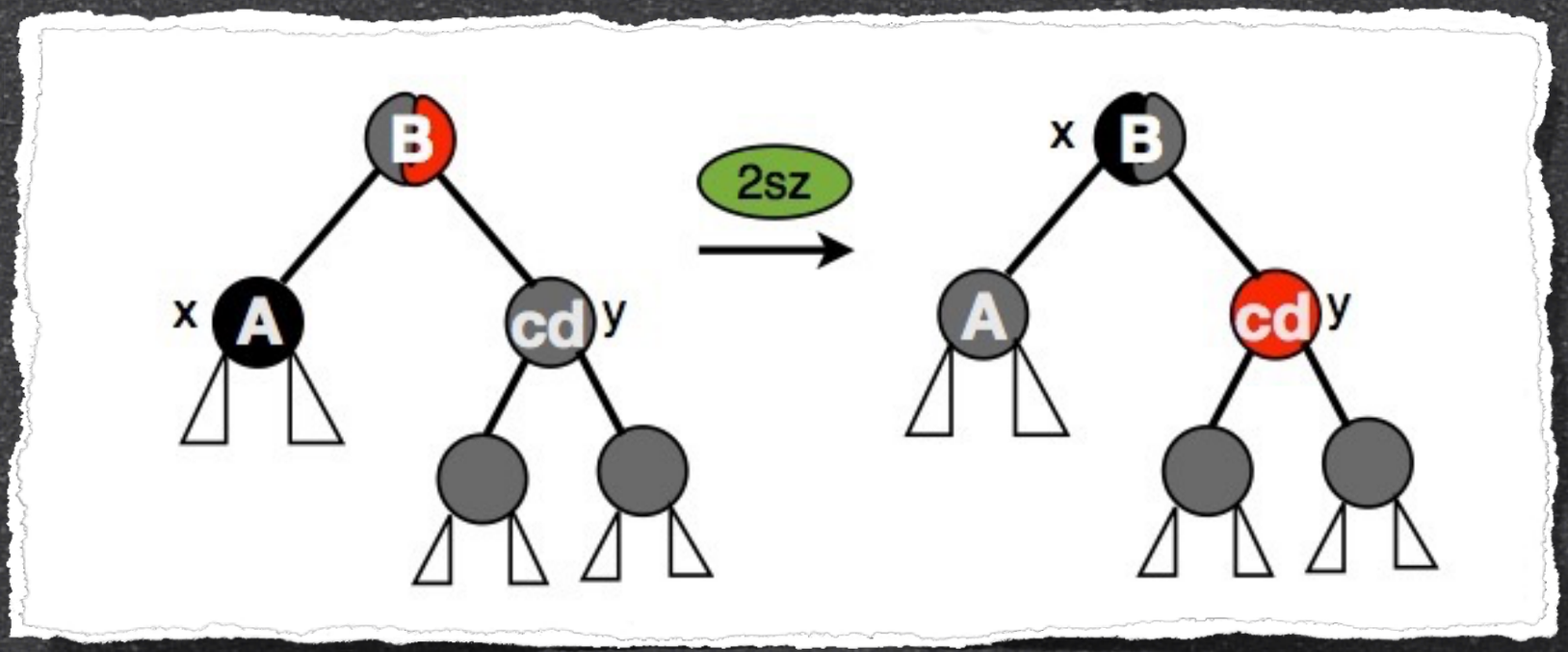


Törlés Piros-Fekete fából

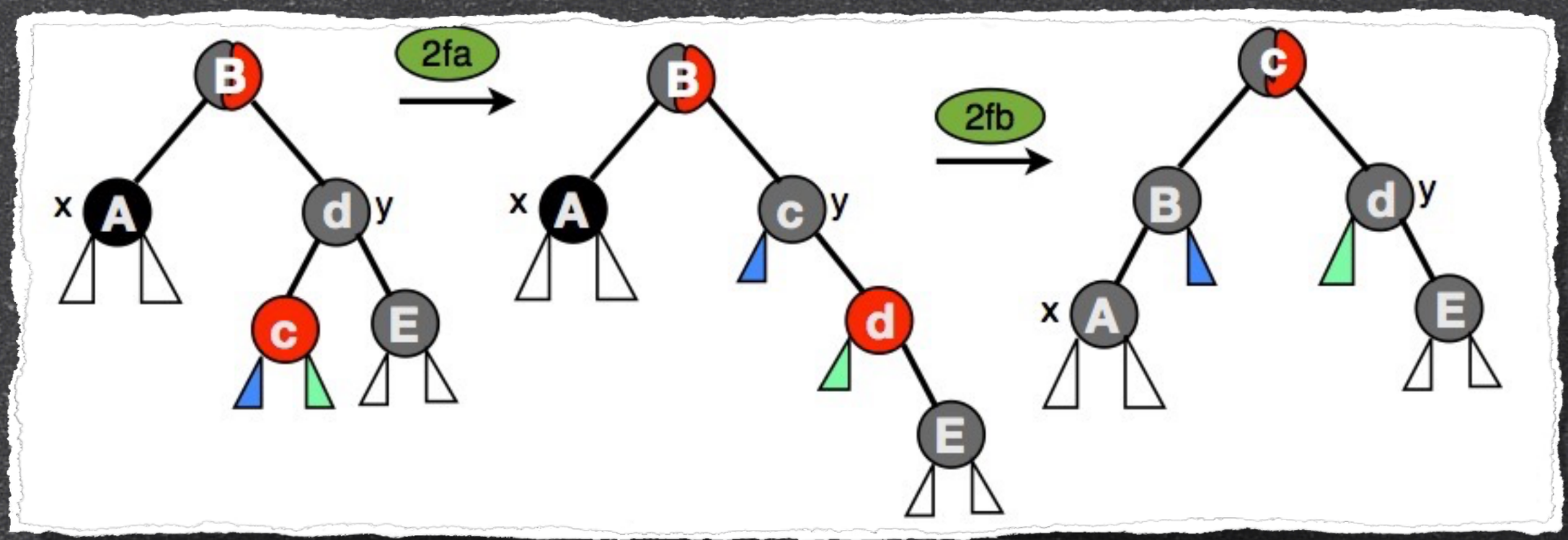
A fából a keresofa Töröl műveletének megfelelően töröljük az adott pontot. Ha a pont vagy a fia piros akkor nincs másteendő, egyébként sérül az 5. tulajdonság és ezt helyre kell állítani. Ezt úgy oldjuk meg, hogy a ténylegesen törölt pont fia kap egy extra fekete értéket, és duplán fekete lesz, majd forgatásokkal és átszínezéssel ezt megszüntetjük.

Törlés utáni javítás I.



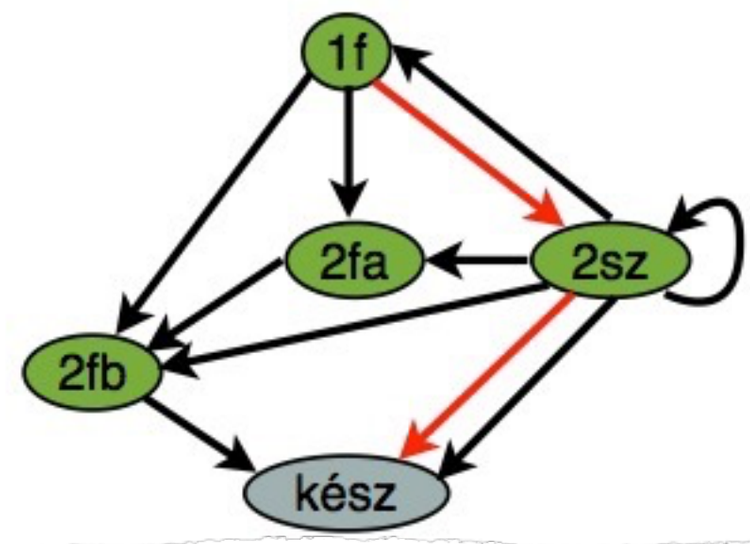


Törlés utáni javítás II.

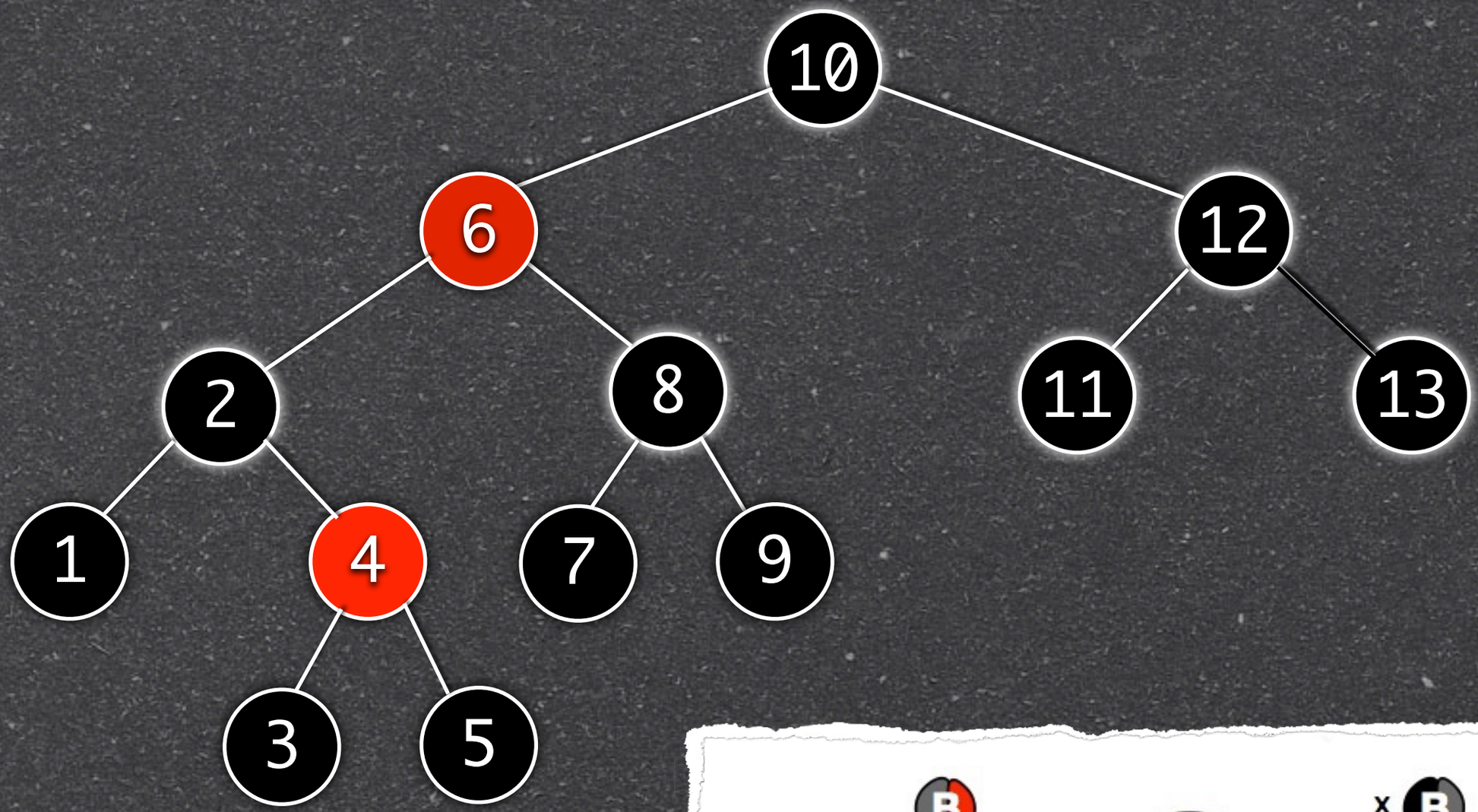


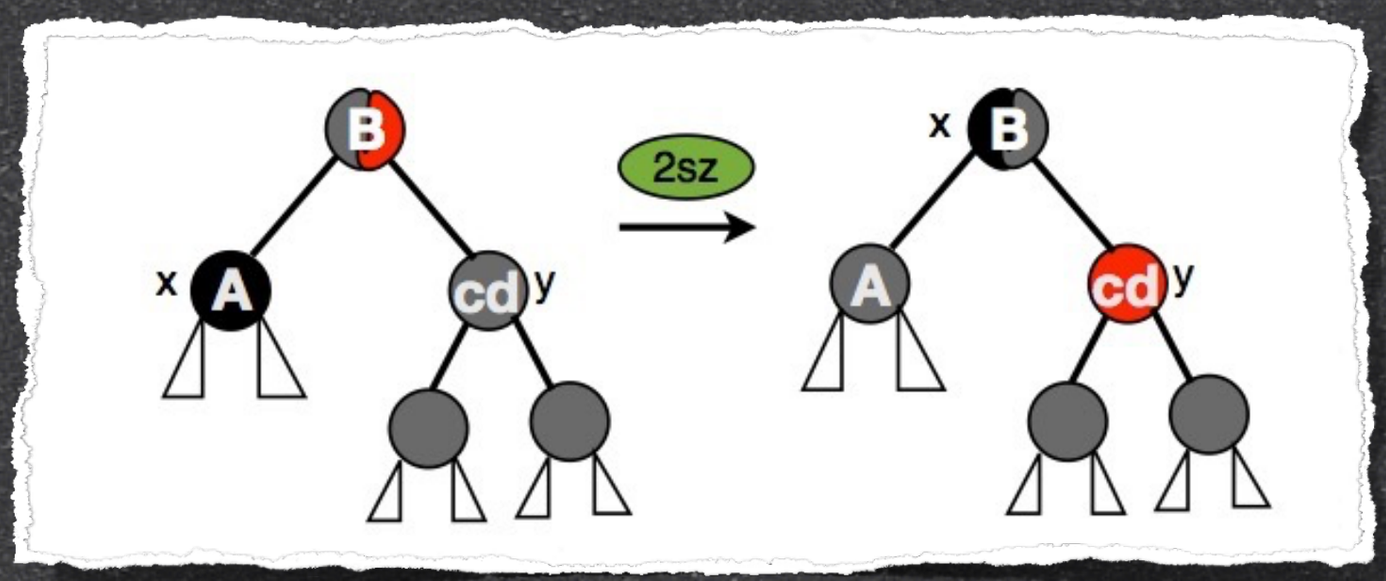
Törlés utáni javítás

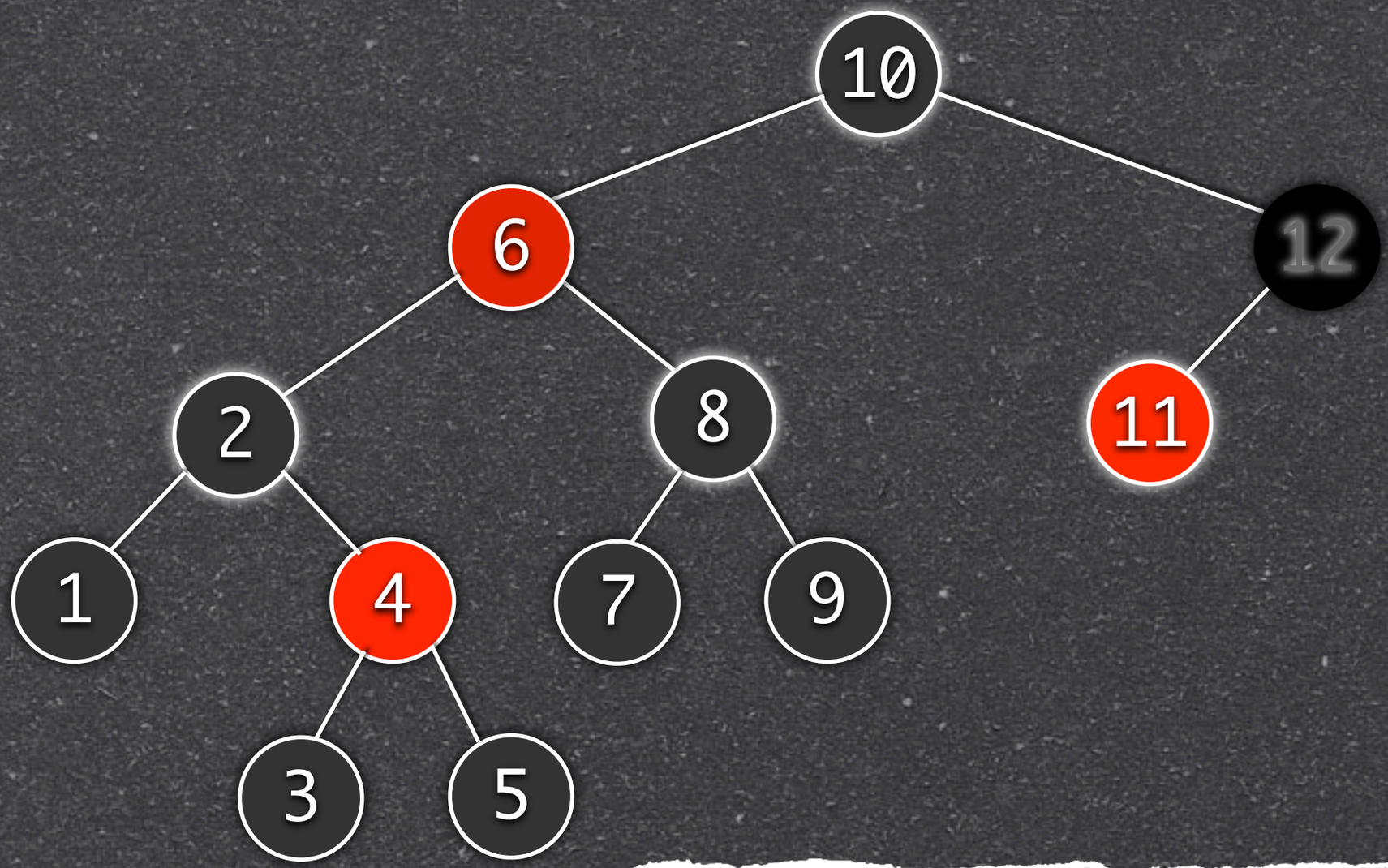
1. eset: testvér piros -> nagybácsi és testvér fekete, apa piros (forgat is)
2. eset: testvér fekete
   (sz) mindkét unokaöccs fekete -> kész vagy probléma 1 szinntel feljebb (csak szinez)
   (fa) csak a jobb unokaöccs fekete -> jobb unokaöccs piros (forgat is)
   (fb) jobb unokaöccs piros -> kész (forgat is)

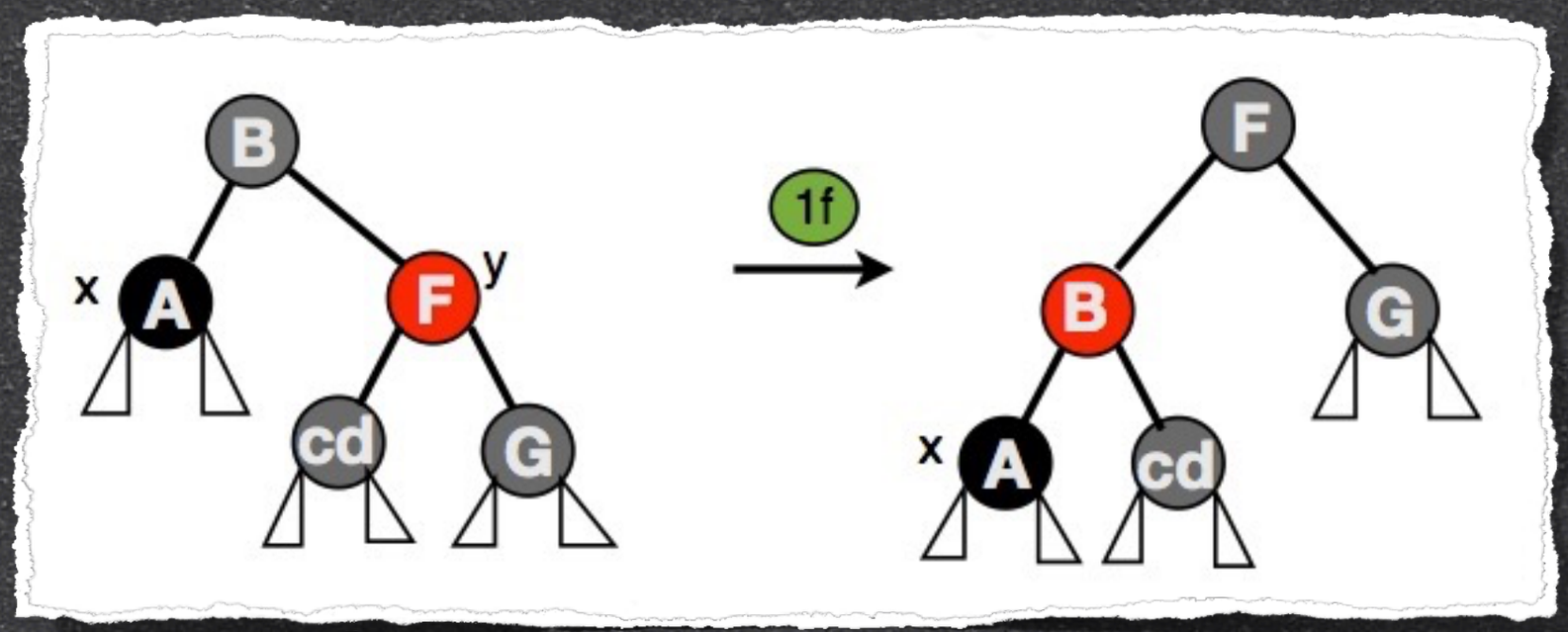


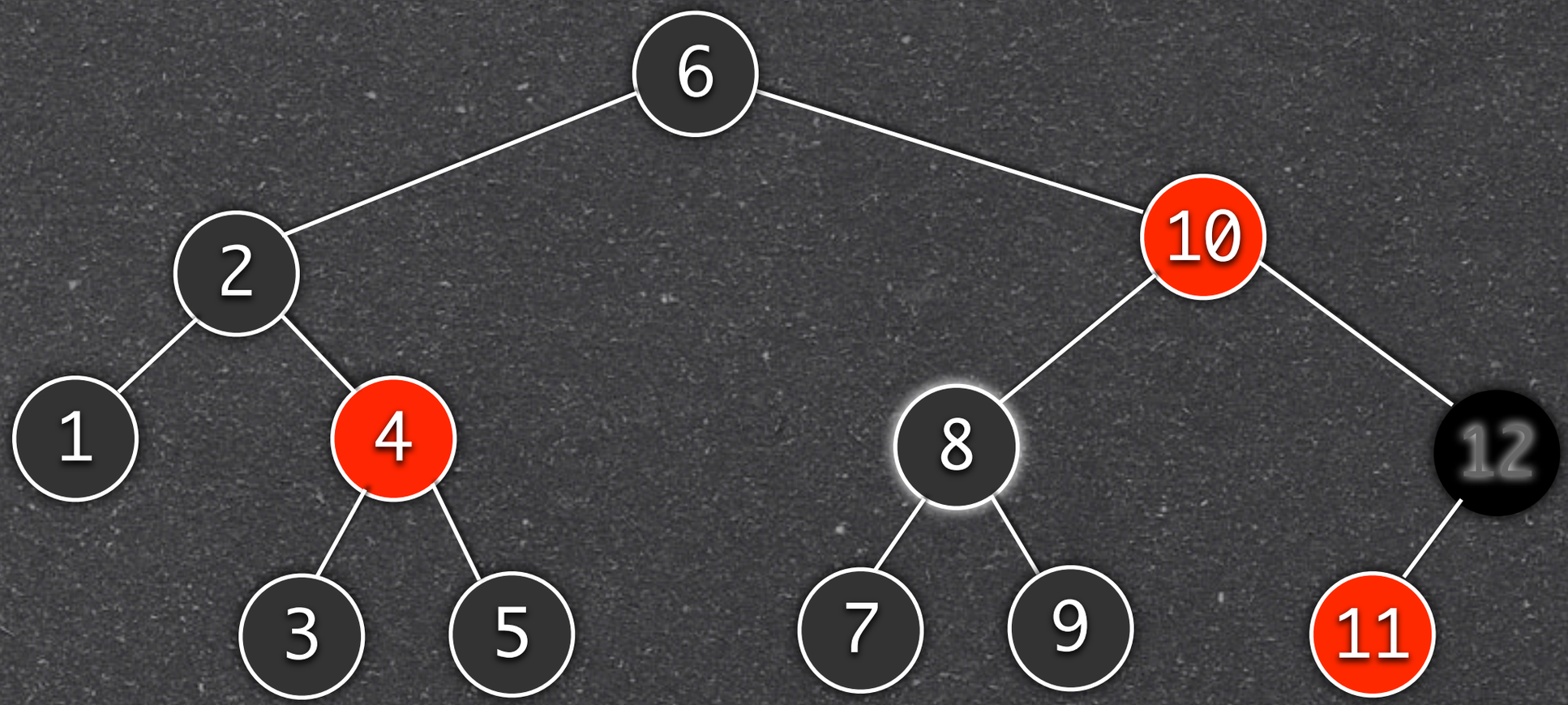
O(logn)

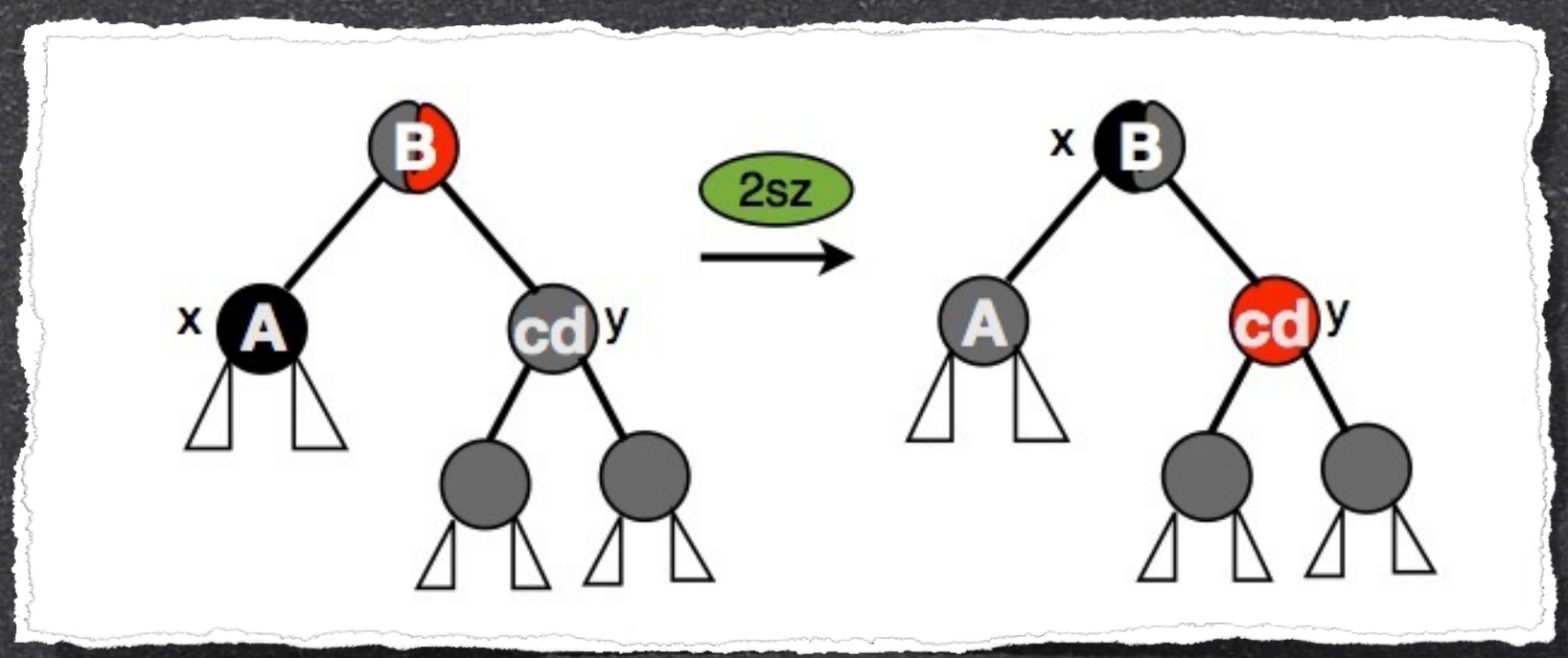

10
6
12
2
8
11
13
1
4
7
9
3
5

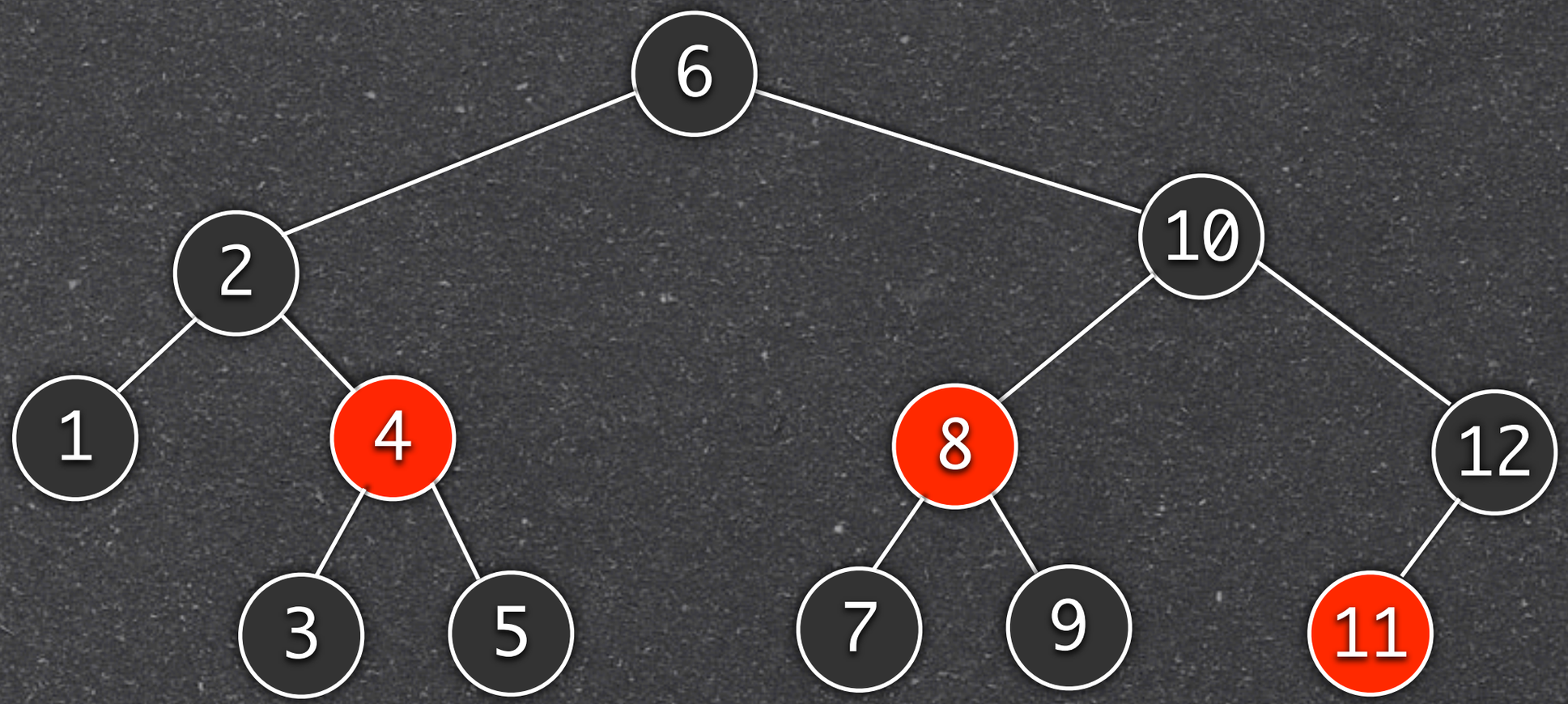

B
x
A
cd
y
2sz
x
B
A
cd
y

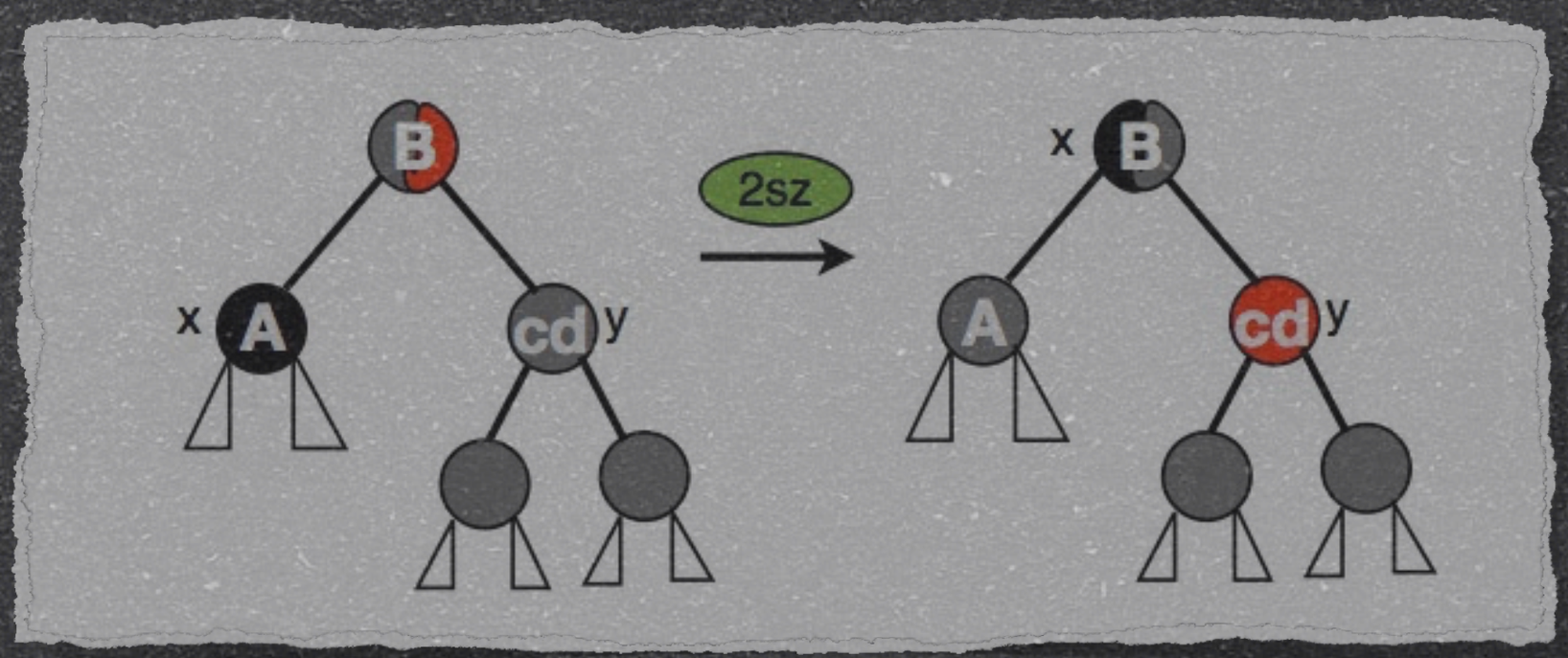

10
6
12
2
8
11
1
4
7
9
3
5

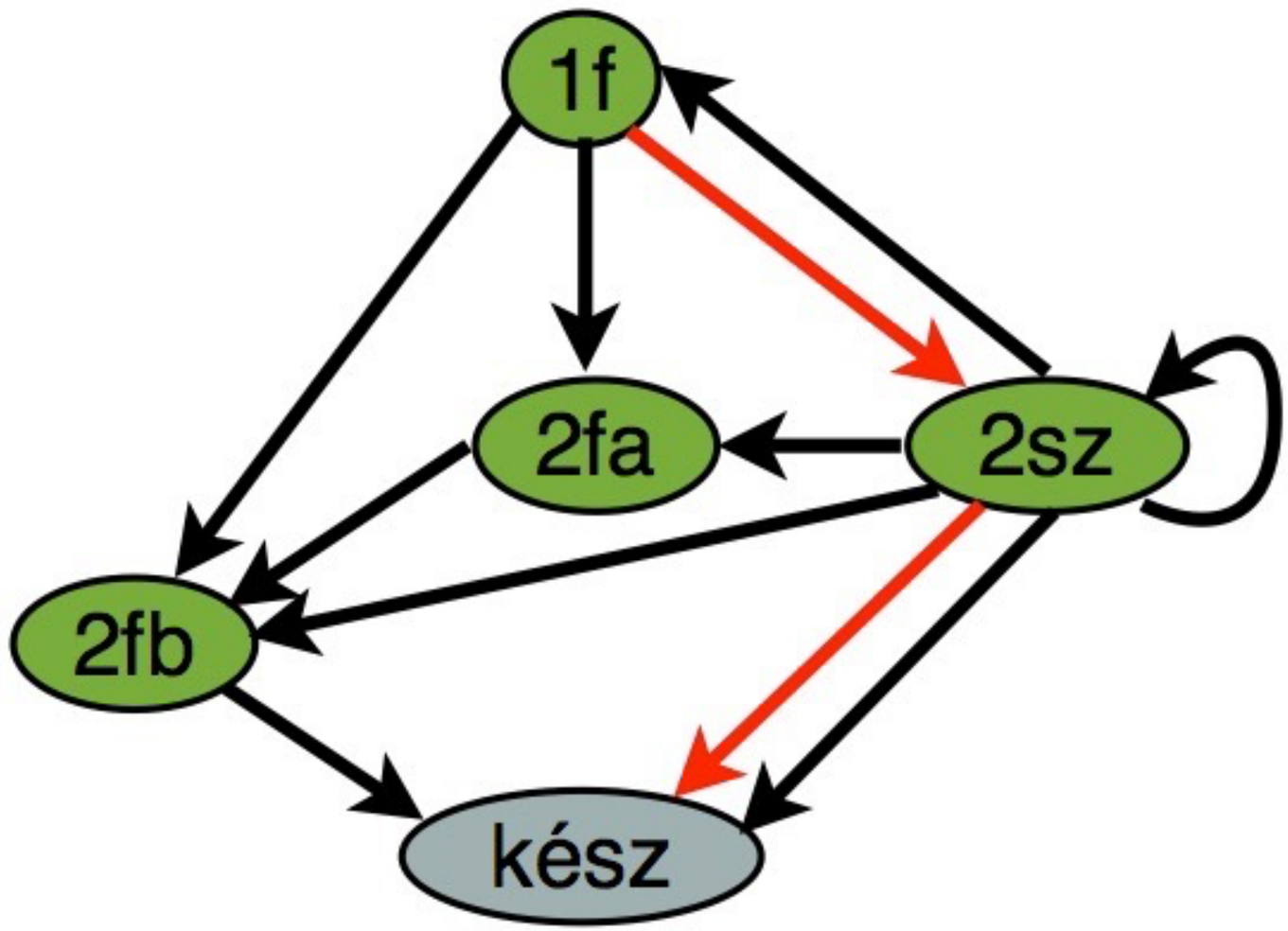

B
x
A
F
y
cd
G
1f
F
B
G
x
A
cd

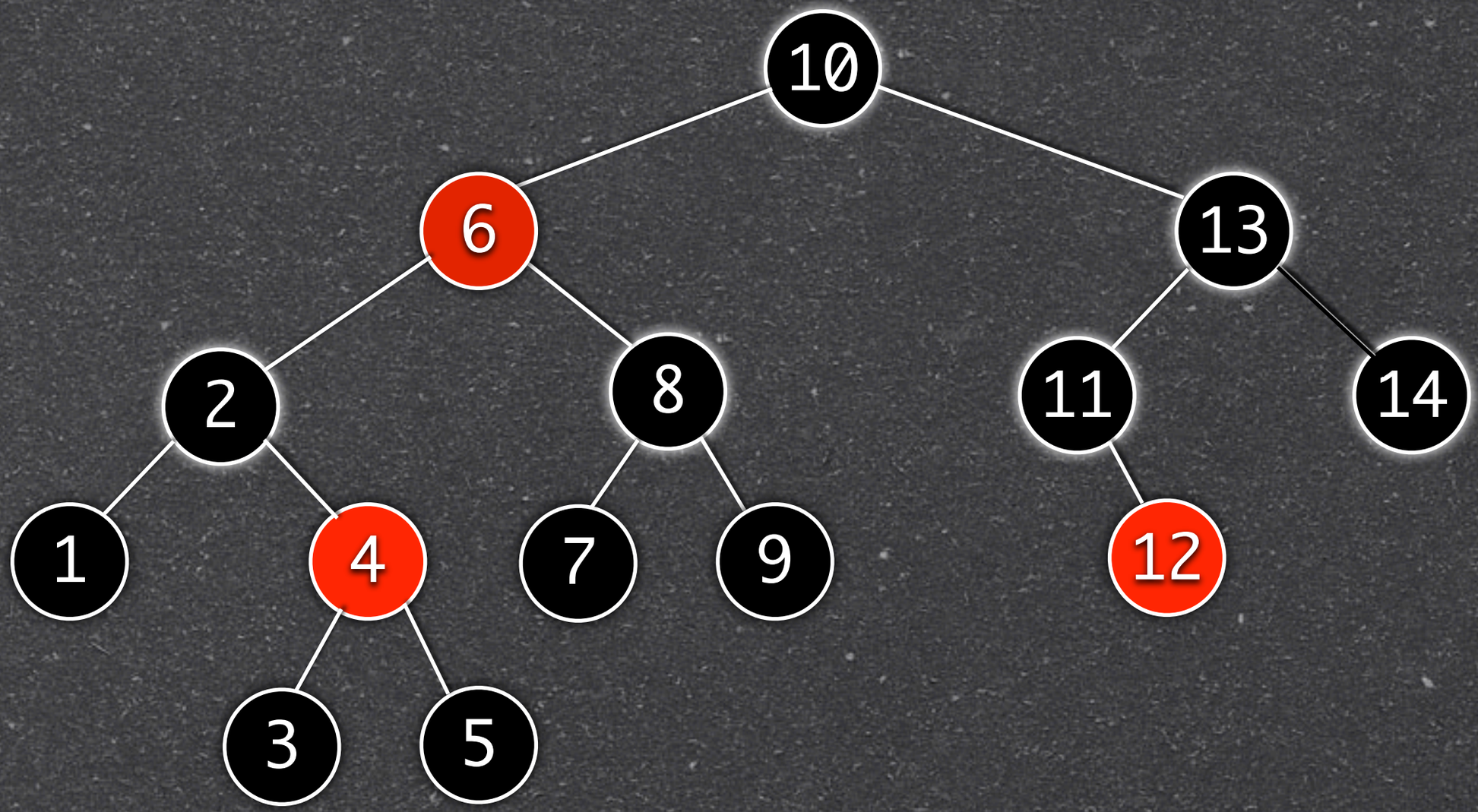

6
2
10
1
4
8
12
3
5
7
9
11

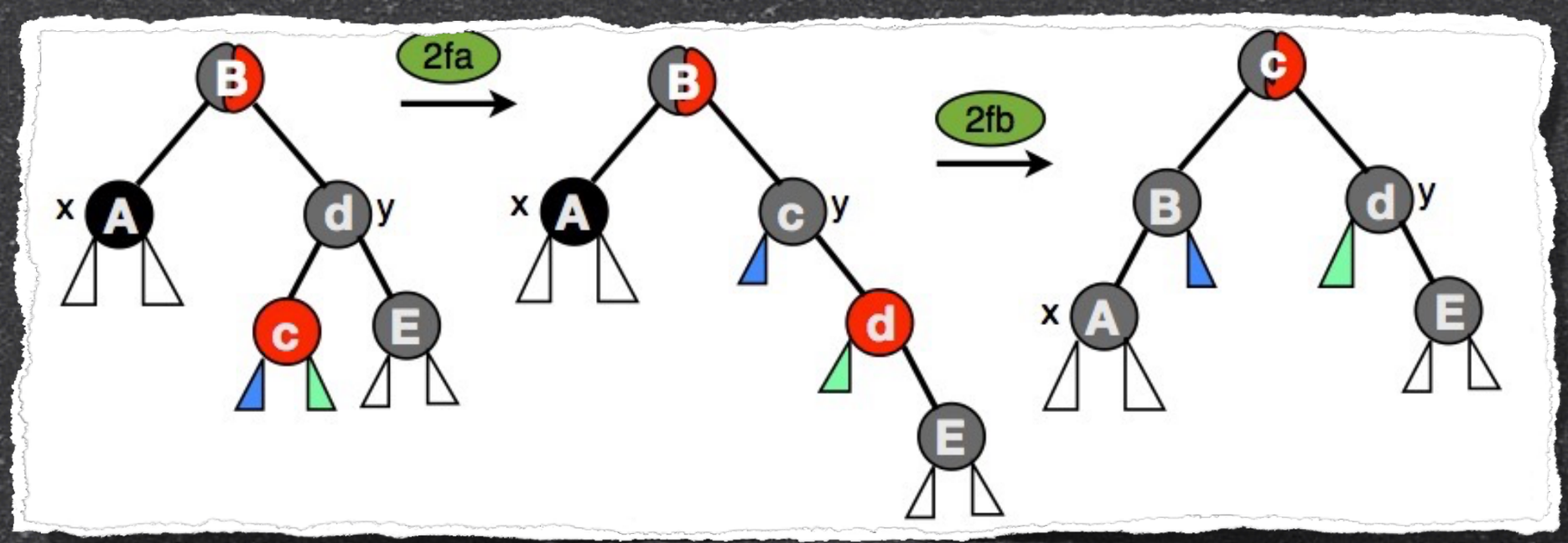

B
x
A
cd
y
2sz
x
B
A
cd
y

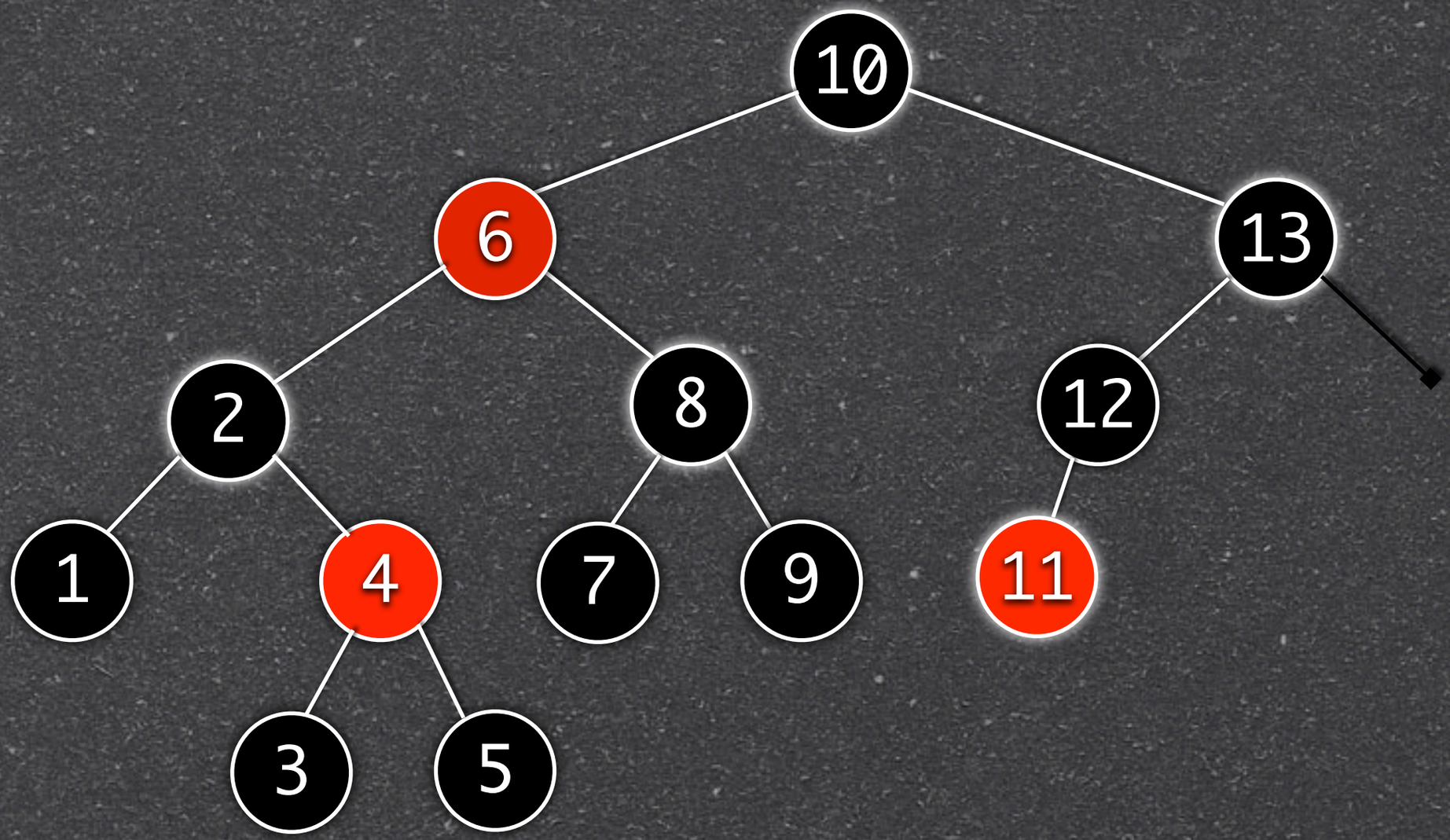

6
2
10
1
4
8
12
3
5
7
9
11

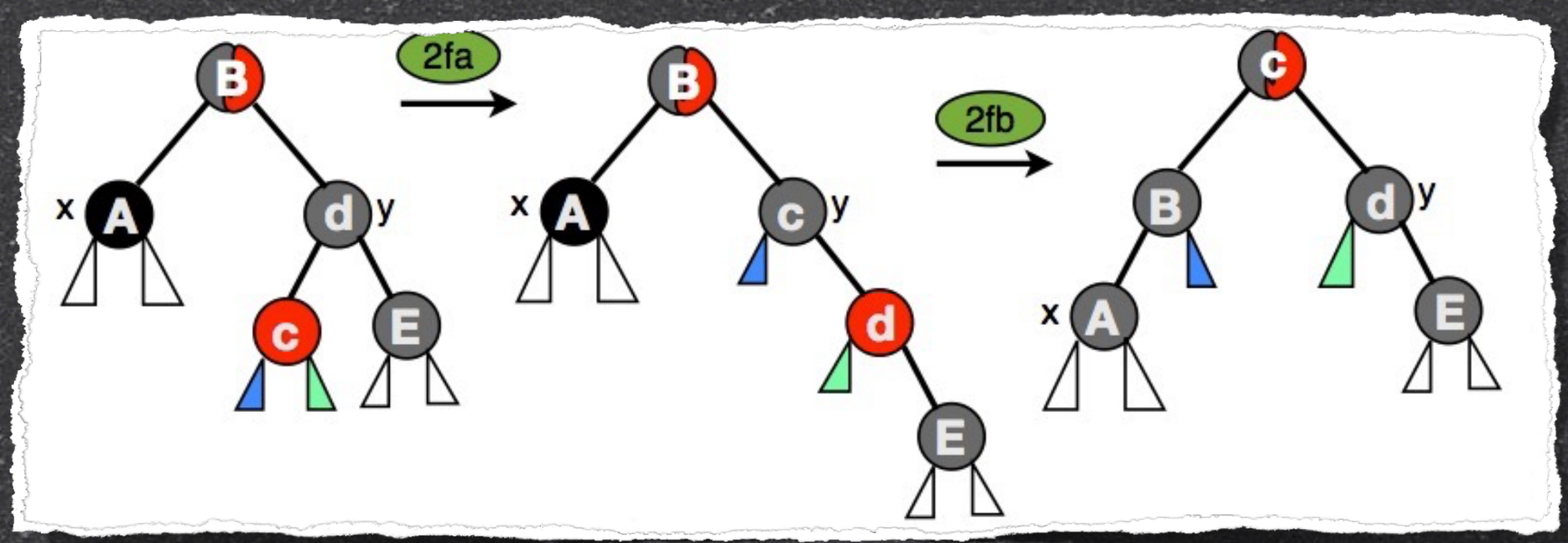

B
x
A
cd
y
2sz
x
B
A
cd
y

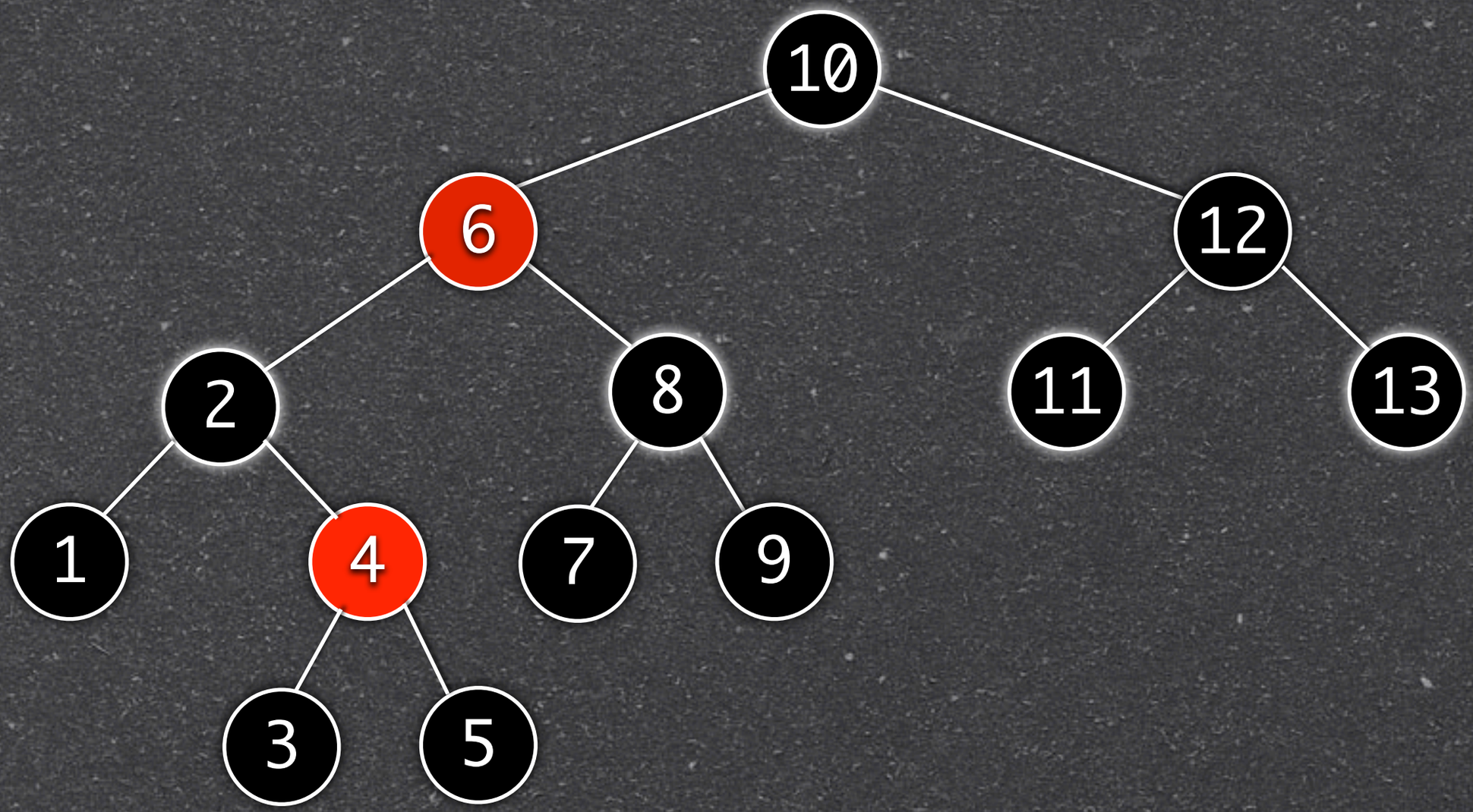

1f
2fa
2sz
2fb
kész

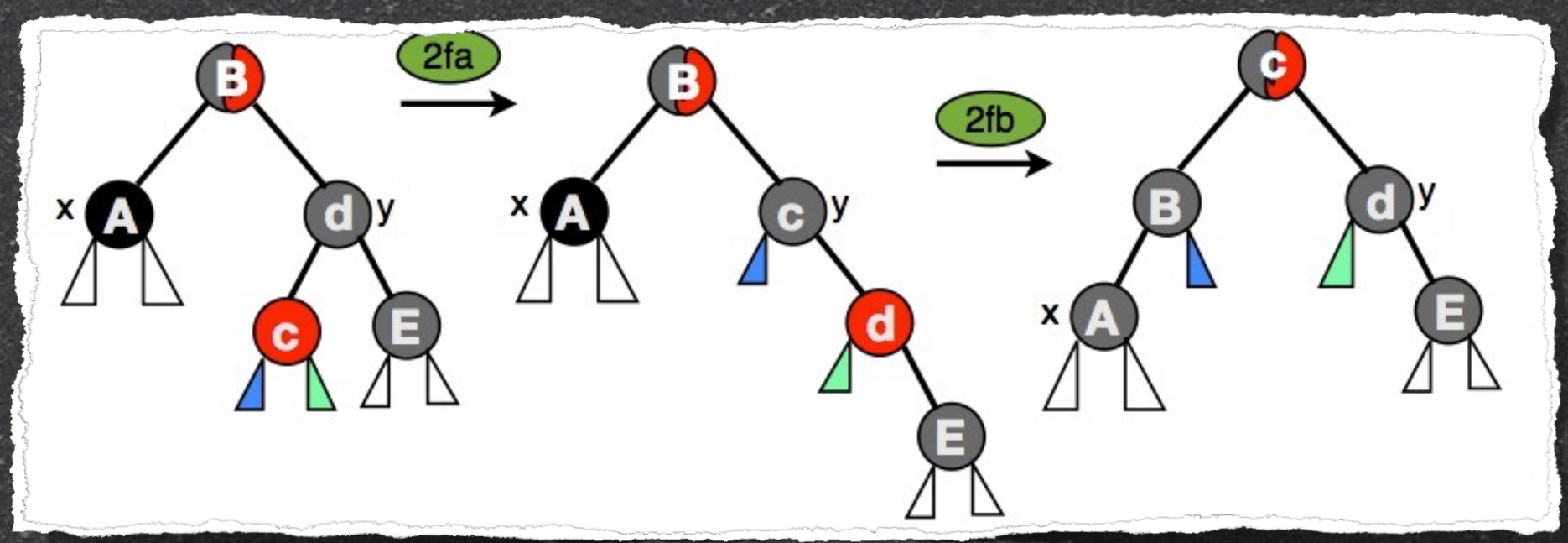

10
6
13
2
8
11
14
1
4
7
9
12
3
5

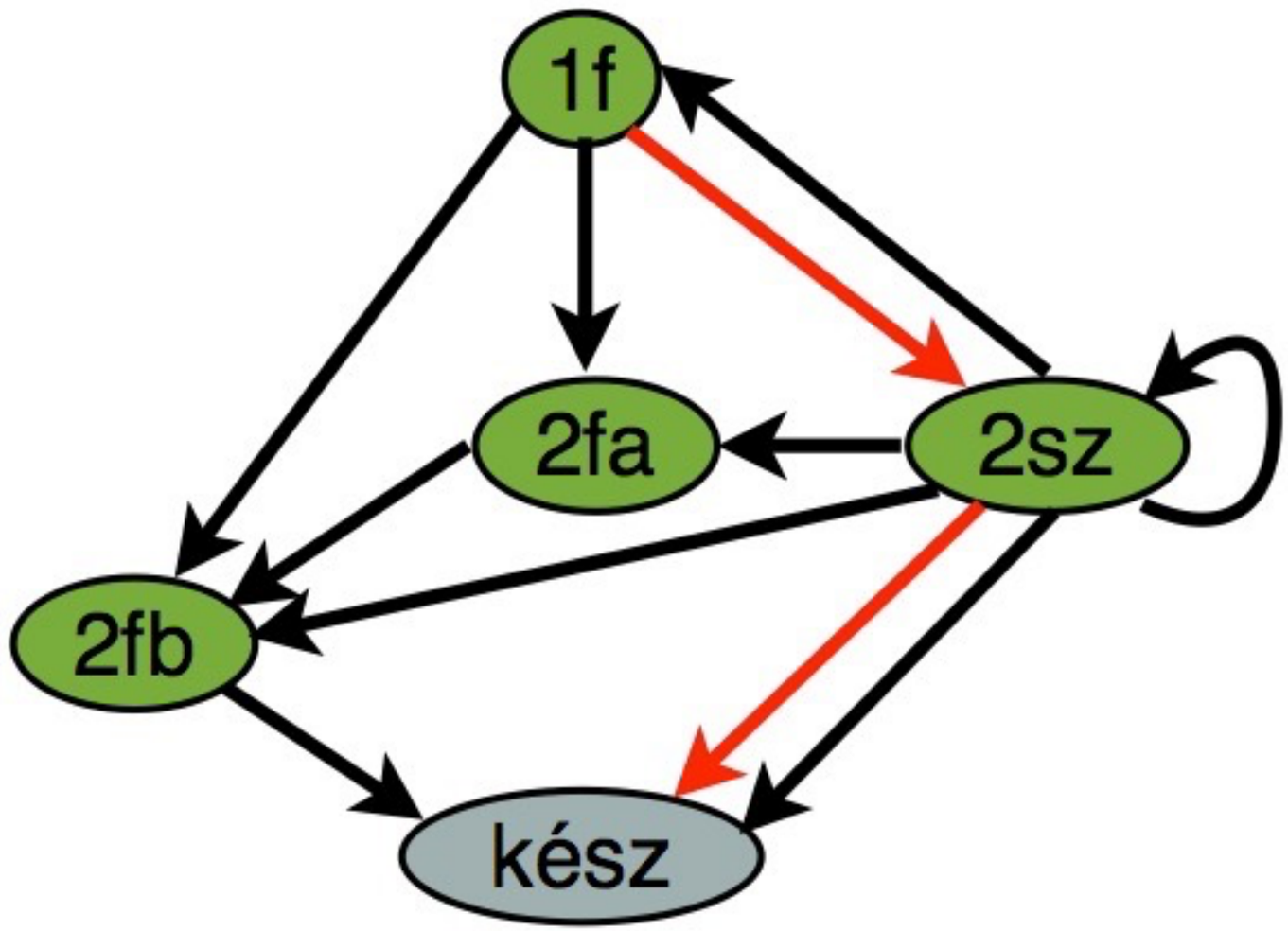

2fa
2fb
B
x
A
d
y
c
E
B
x
A
c
y
d
E
c
B
d
y
x
A
E

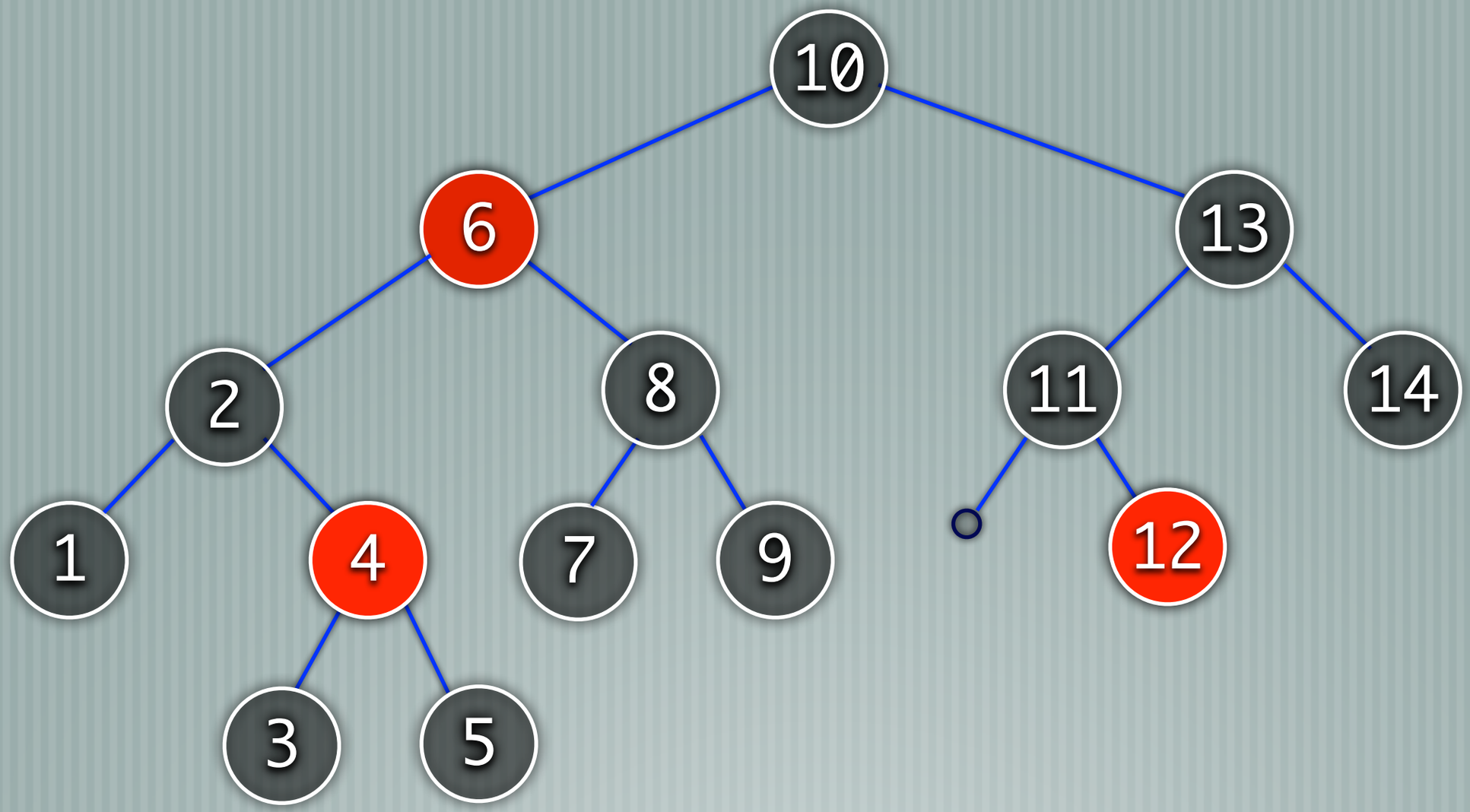

10
6
13
2
8
12
1
4
7
9
11
3
5

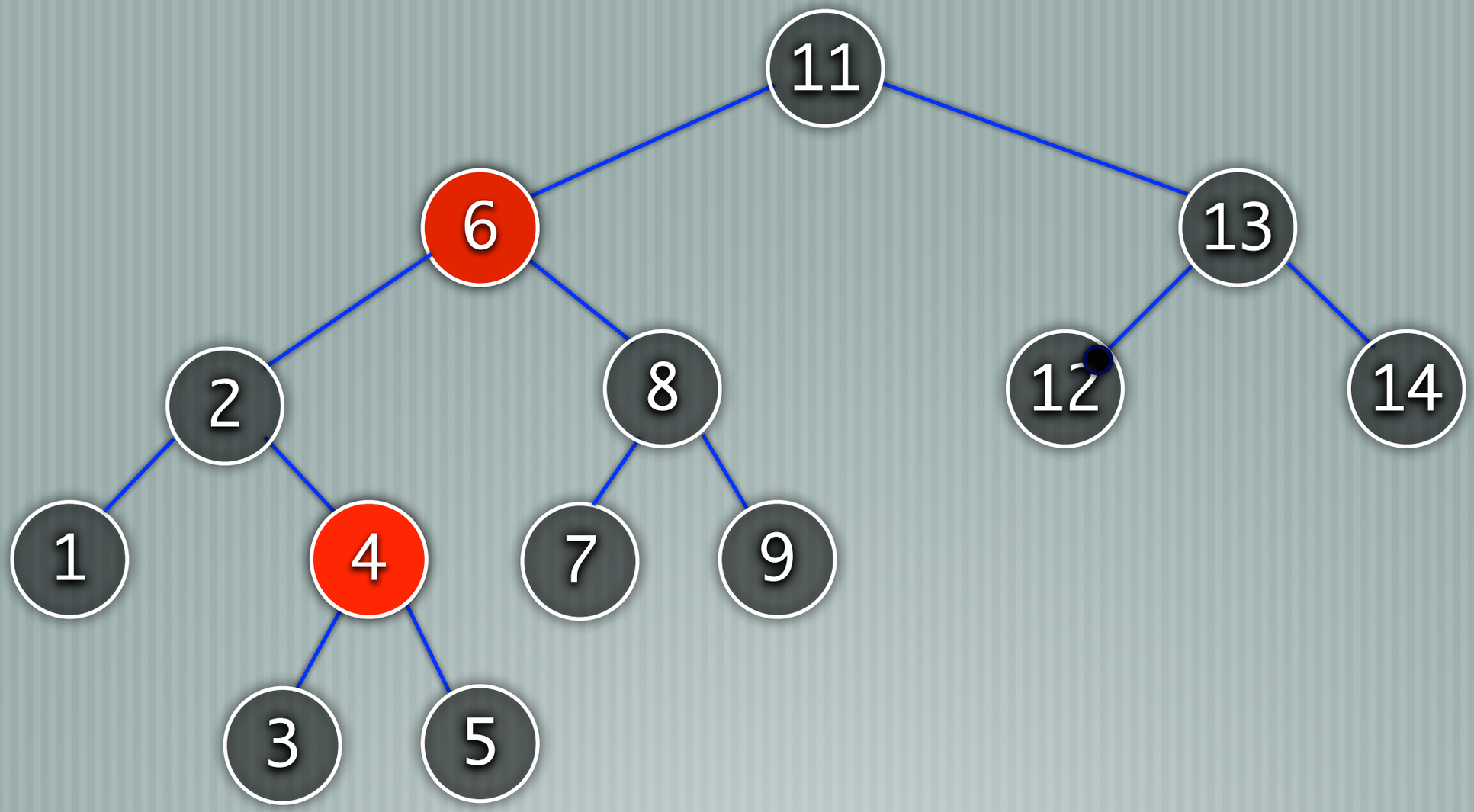

2fa
2fb
B
x
A
d
y
c
E
B
x
A
c
y
d
E
c
B
d
y
x
A
E

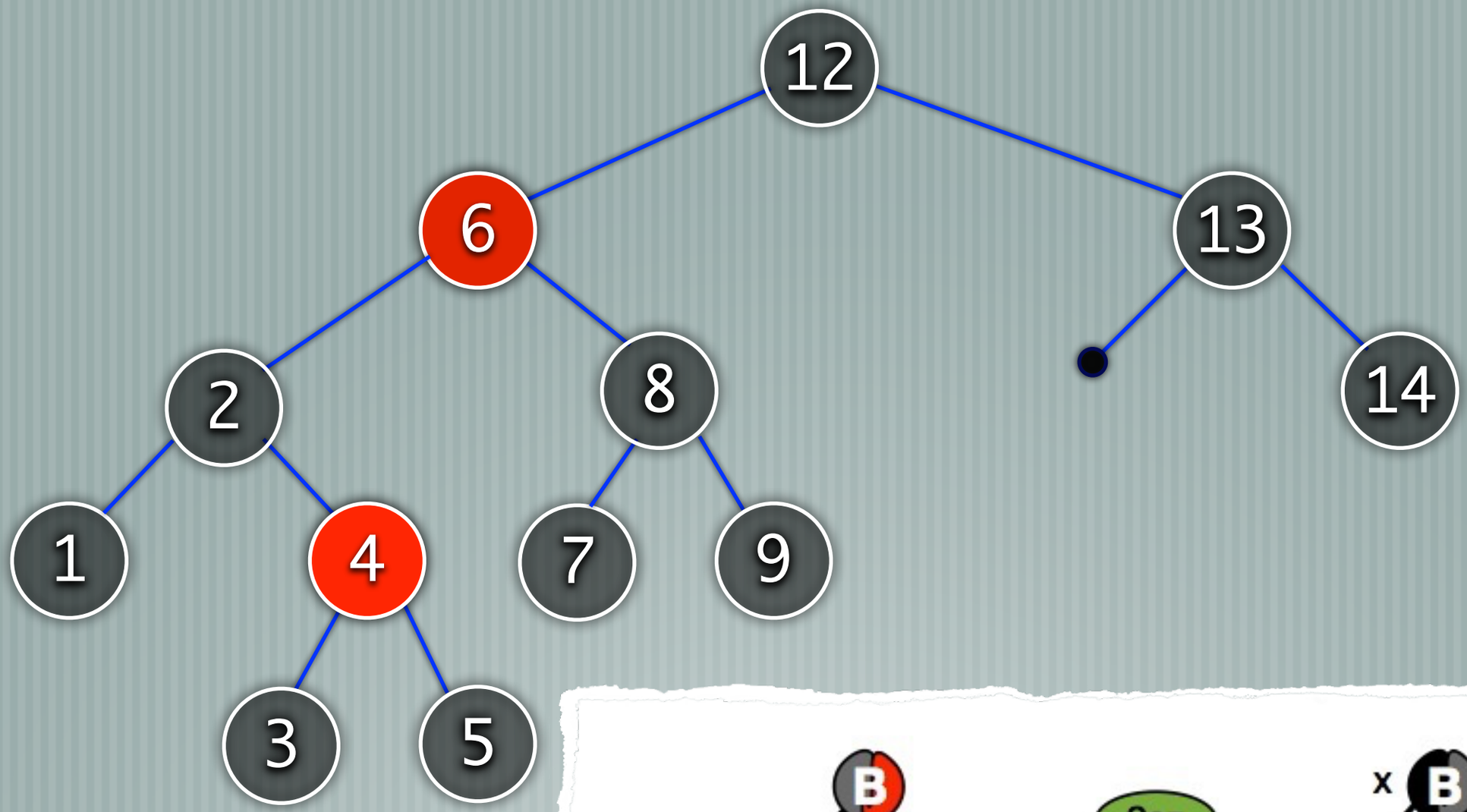

10
6
12
2
8
11
13
1
4
7
9
3
5

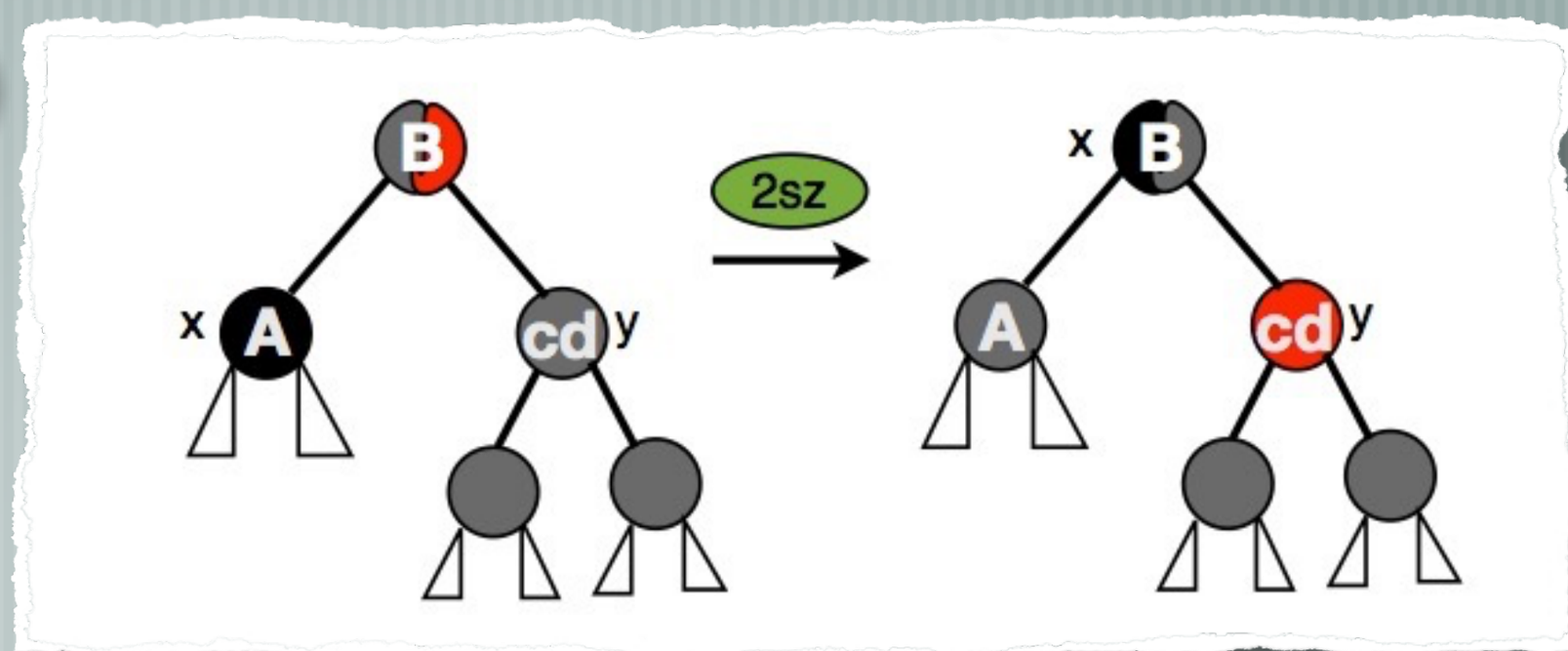

2fa
2fb
B
x
A
d
y
c
E
B
x
A
c
y
d
E
c
B
d
y
x
A
E

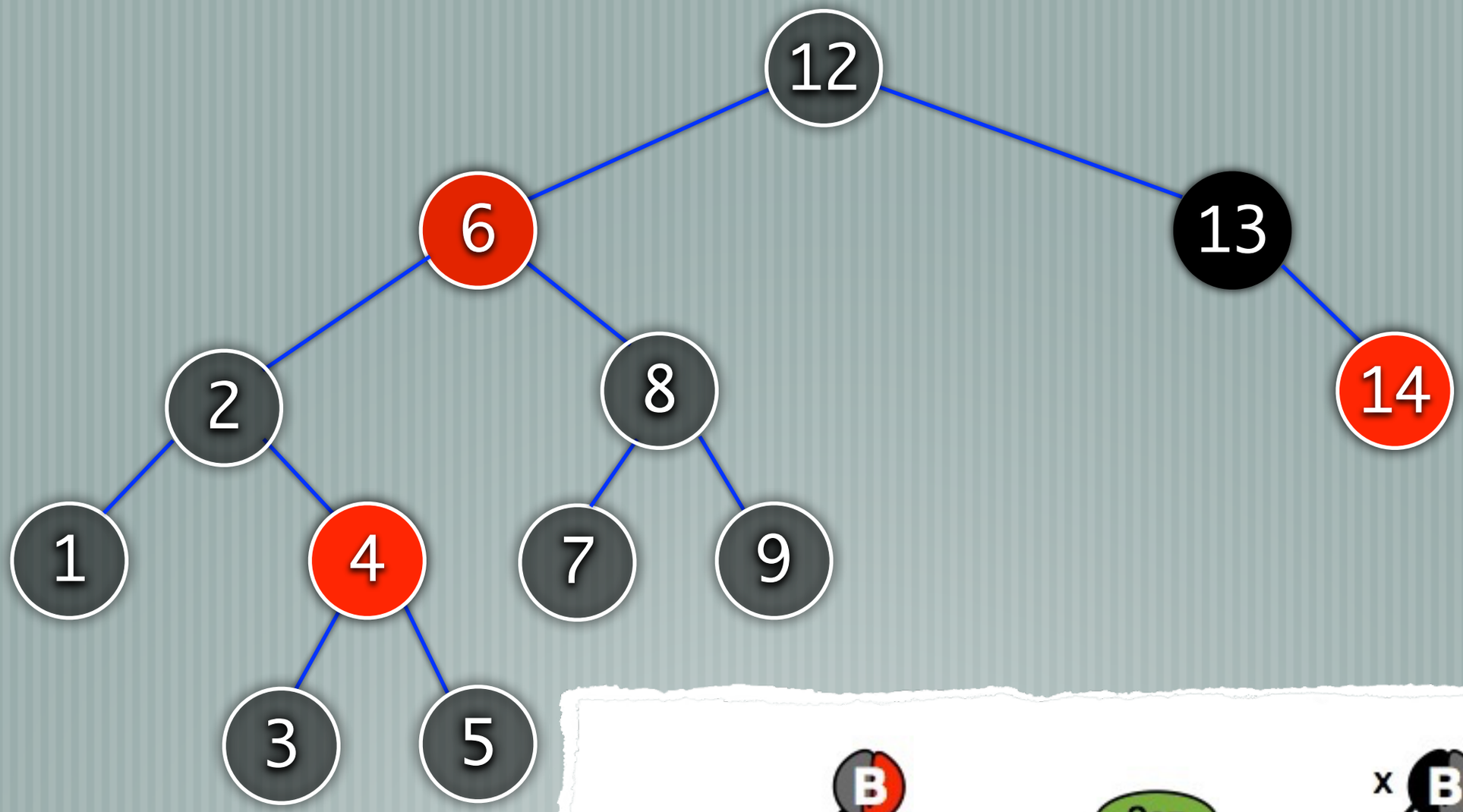

1f
2fa
2sz
2fb
kész

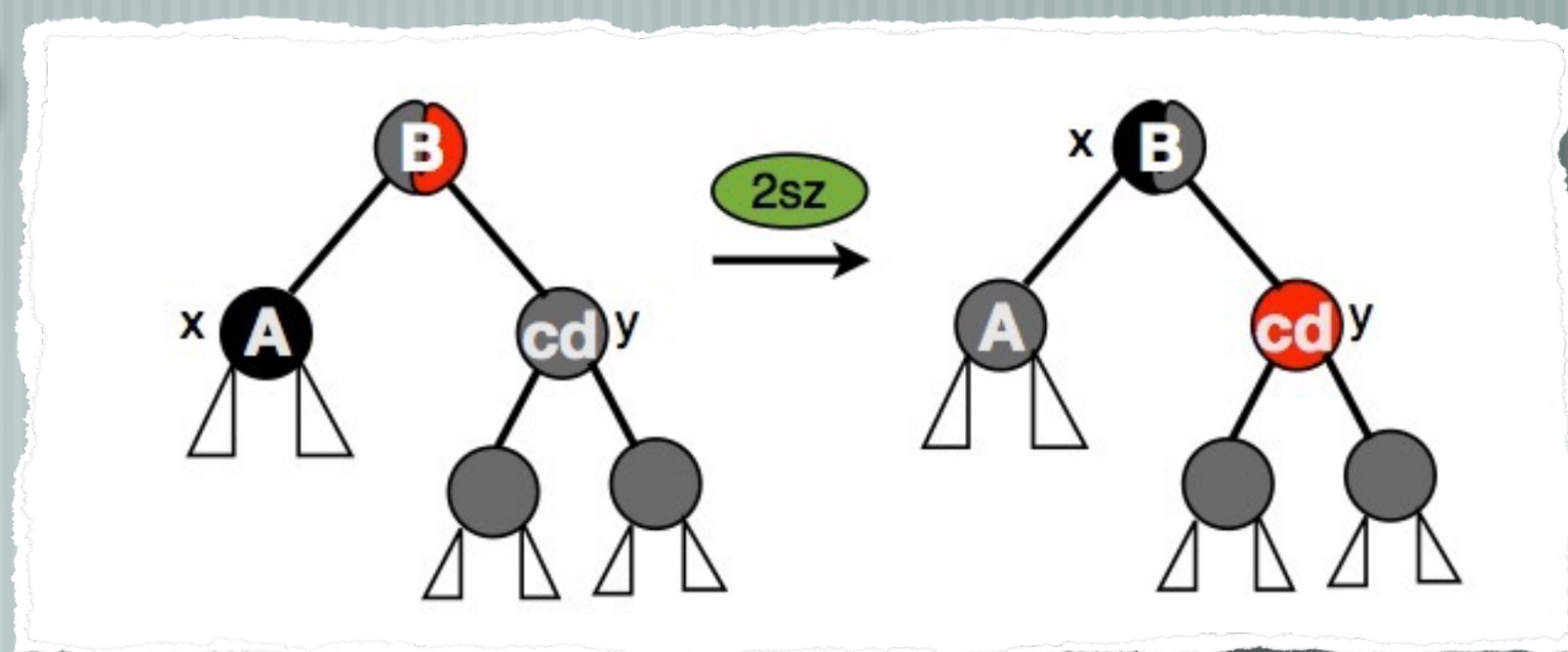

10
6
13
2
8
11
14
1
4
7
9
12
3
5

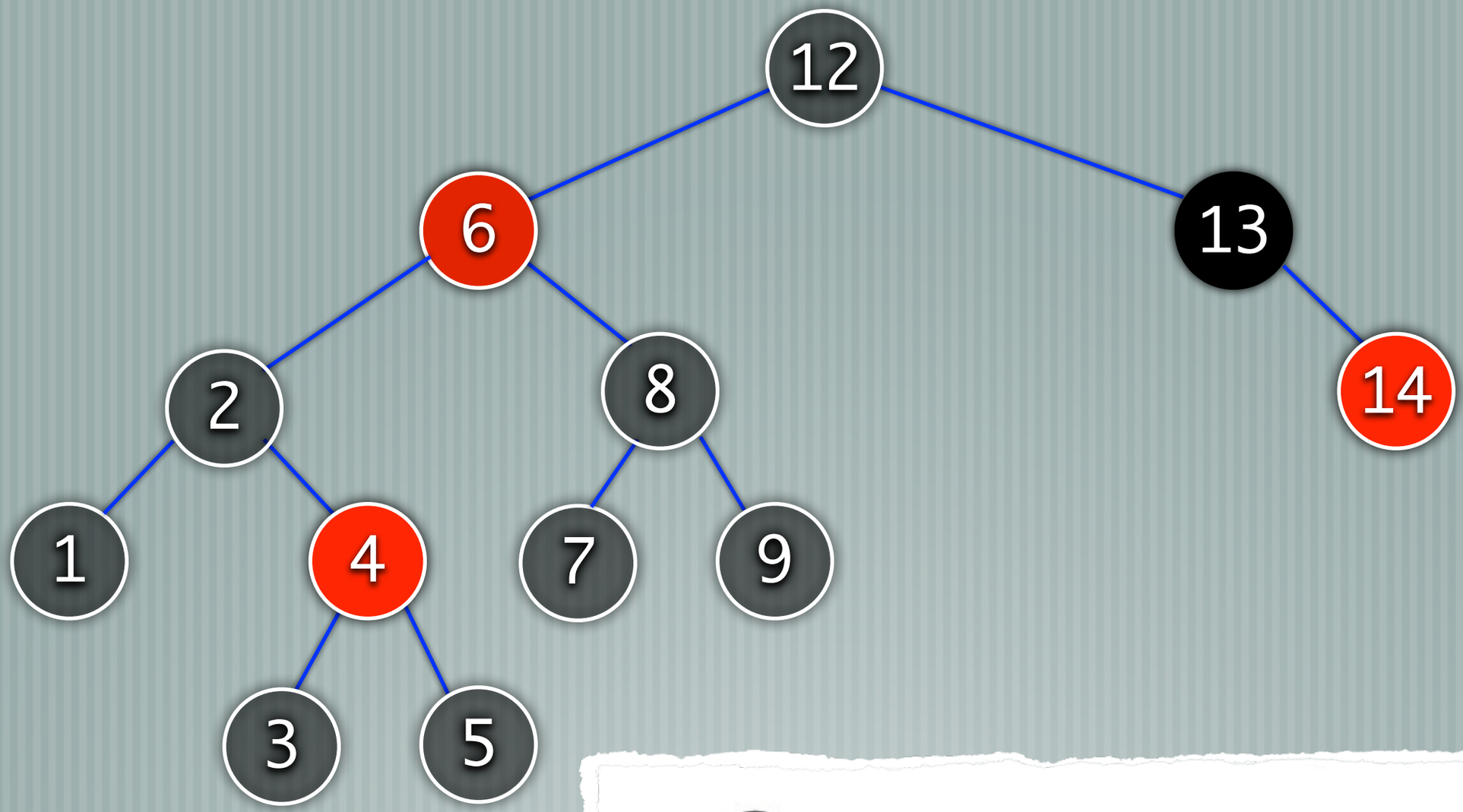

11
6
13
2
8
12
14
1
4
7
9
3
5

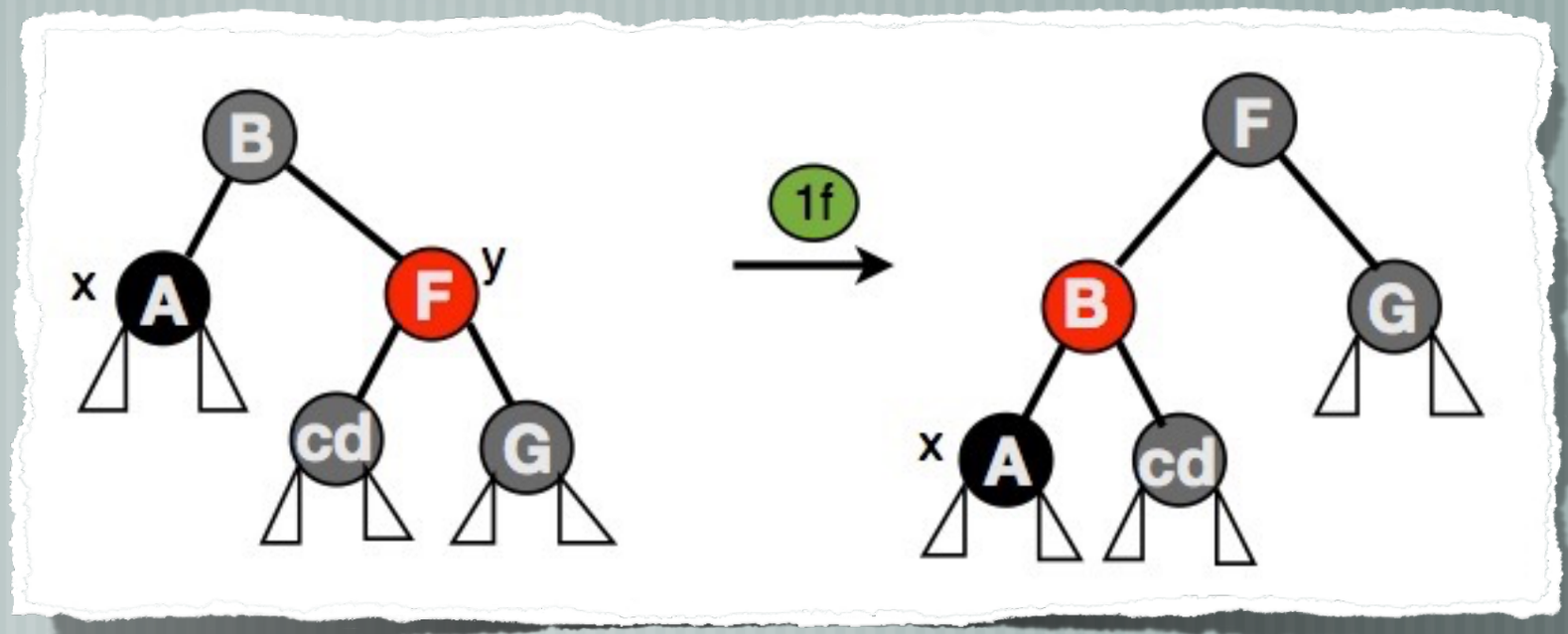

12
6
13
2
8
14
1
4
7
9
3
5

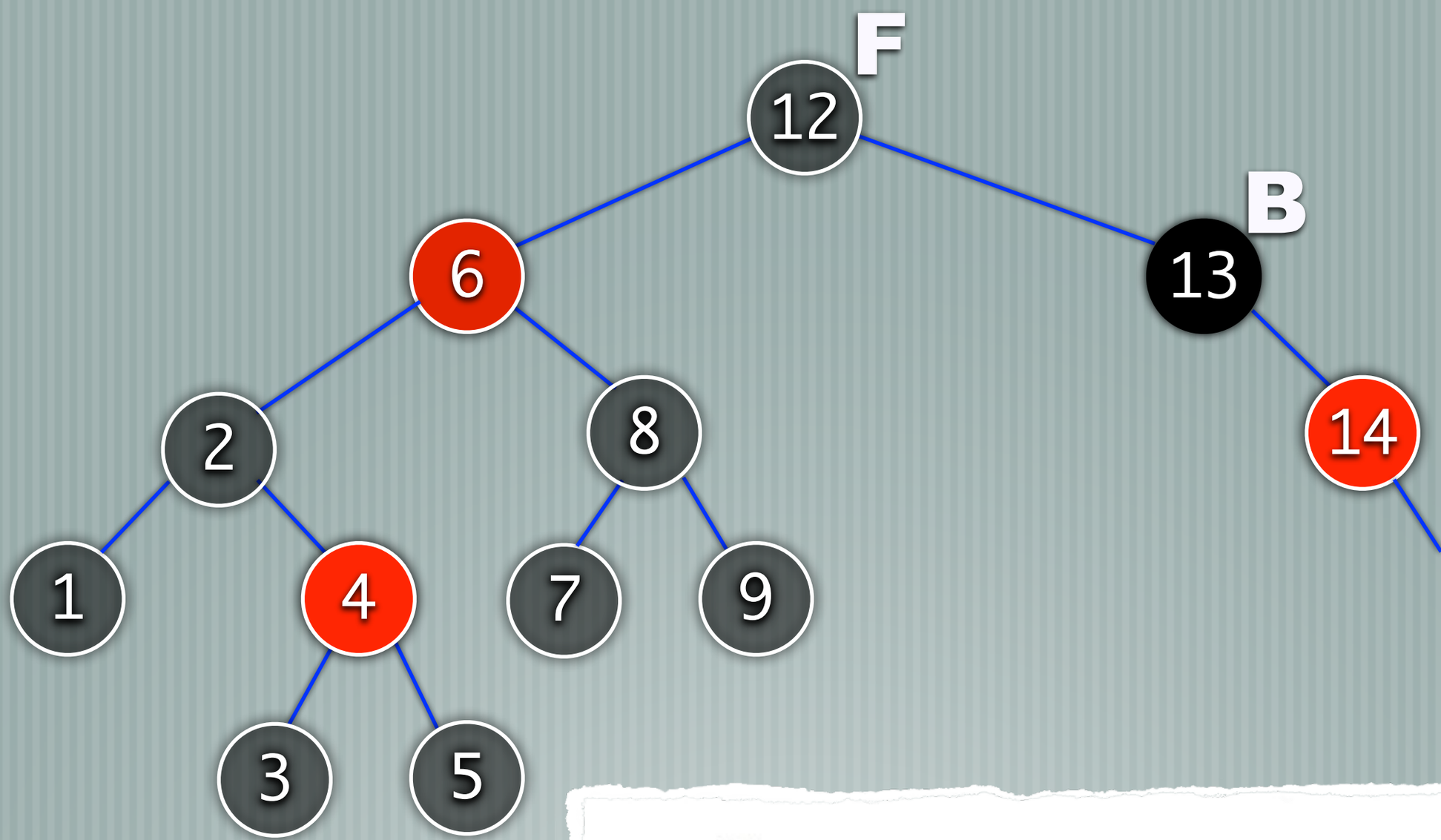

B
x
A
cd
y
2sz
x
B
A
cd
y

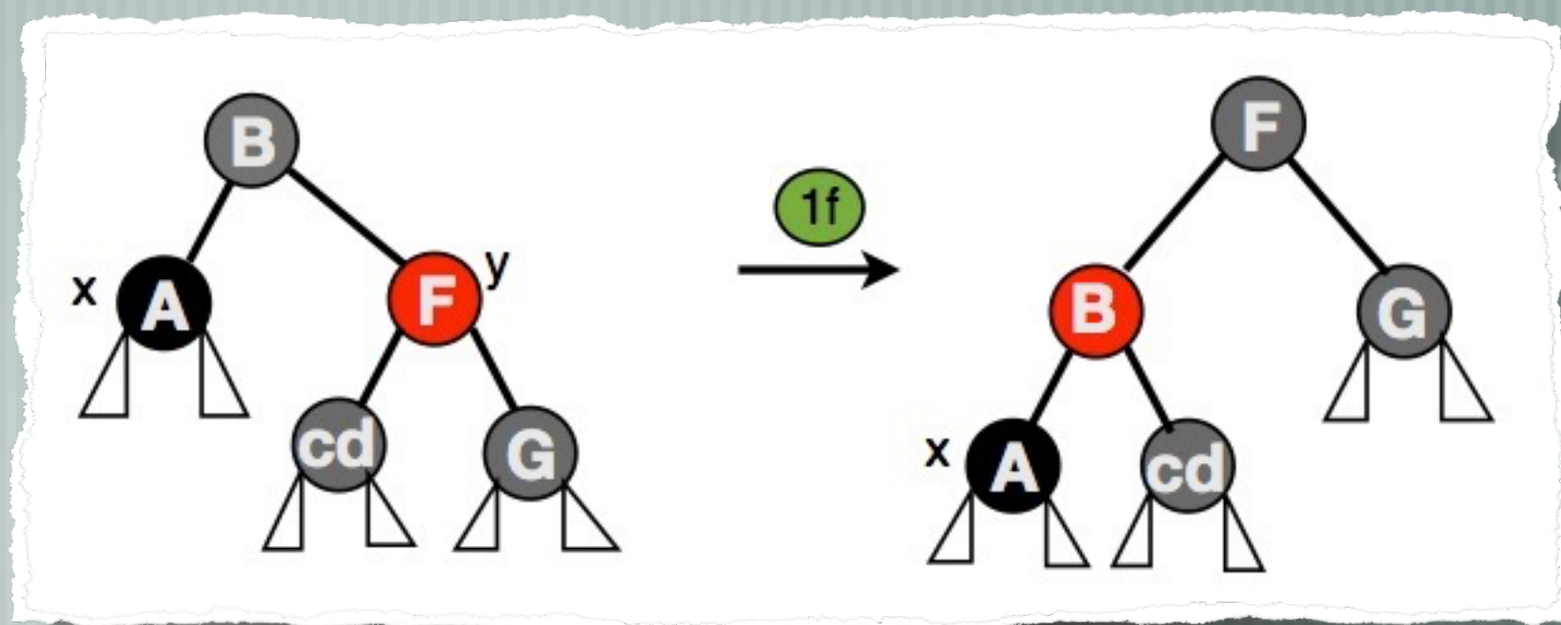

12
6
13
2
8
14
1
4
7
9
3
5

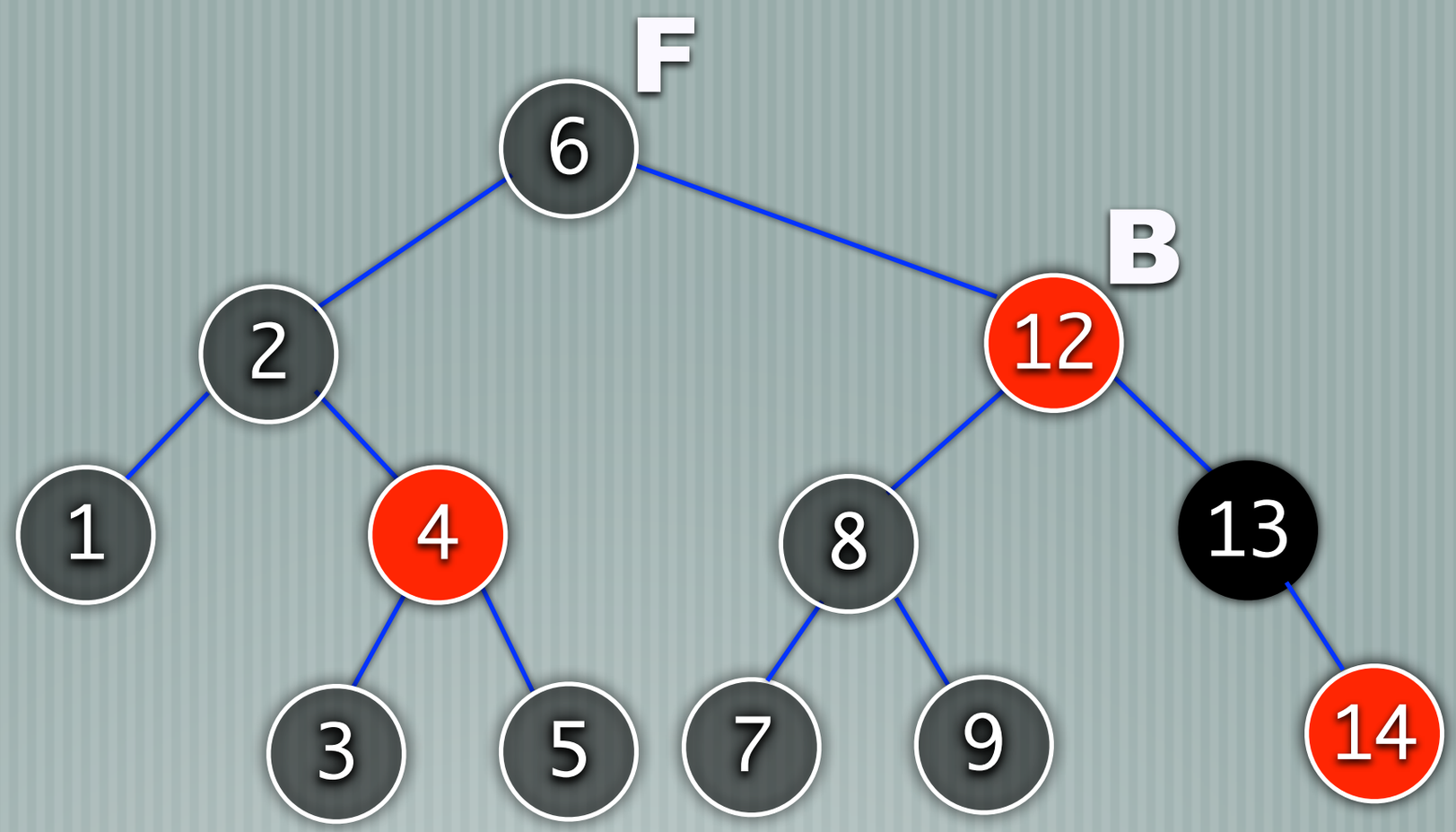

B
x
A
cd
y
2sz
x
B
A
cd
y

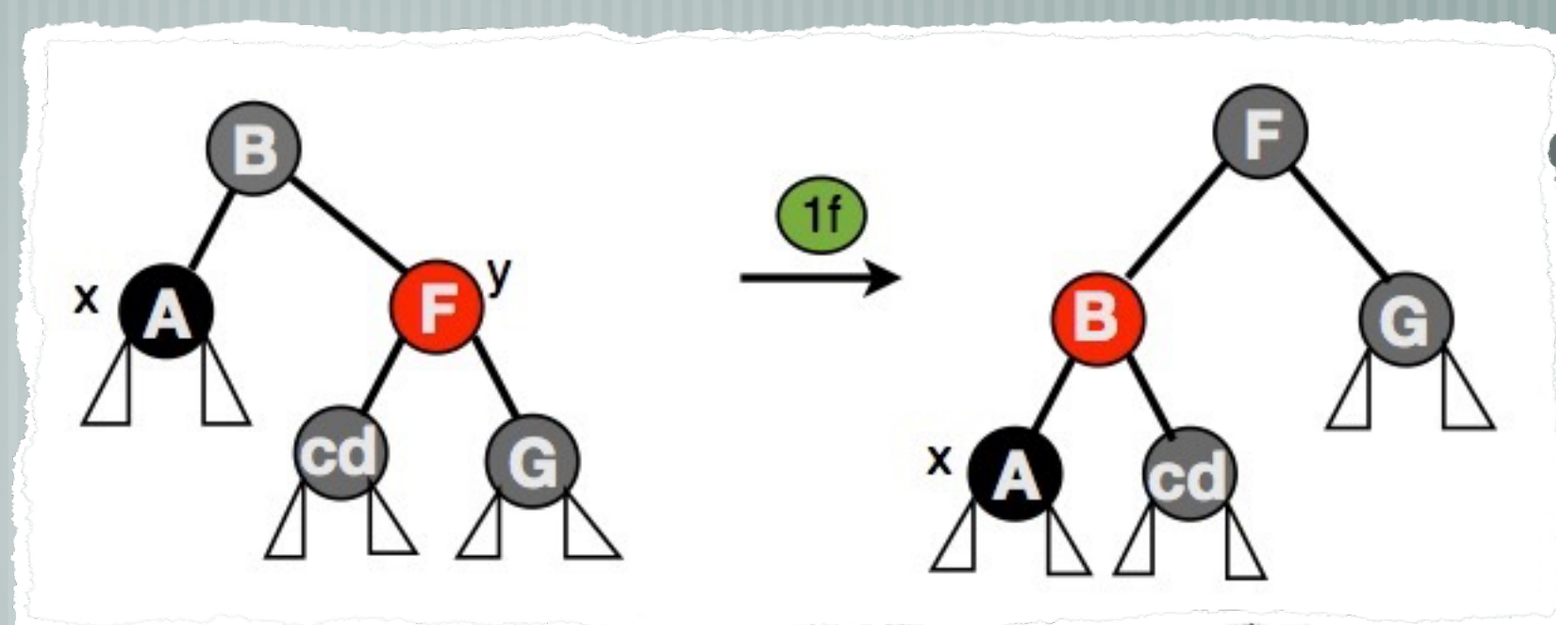

12
6
13
2
8
14
1
4
7
9
3
5

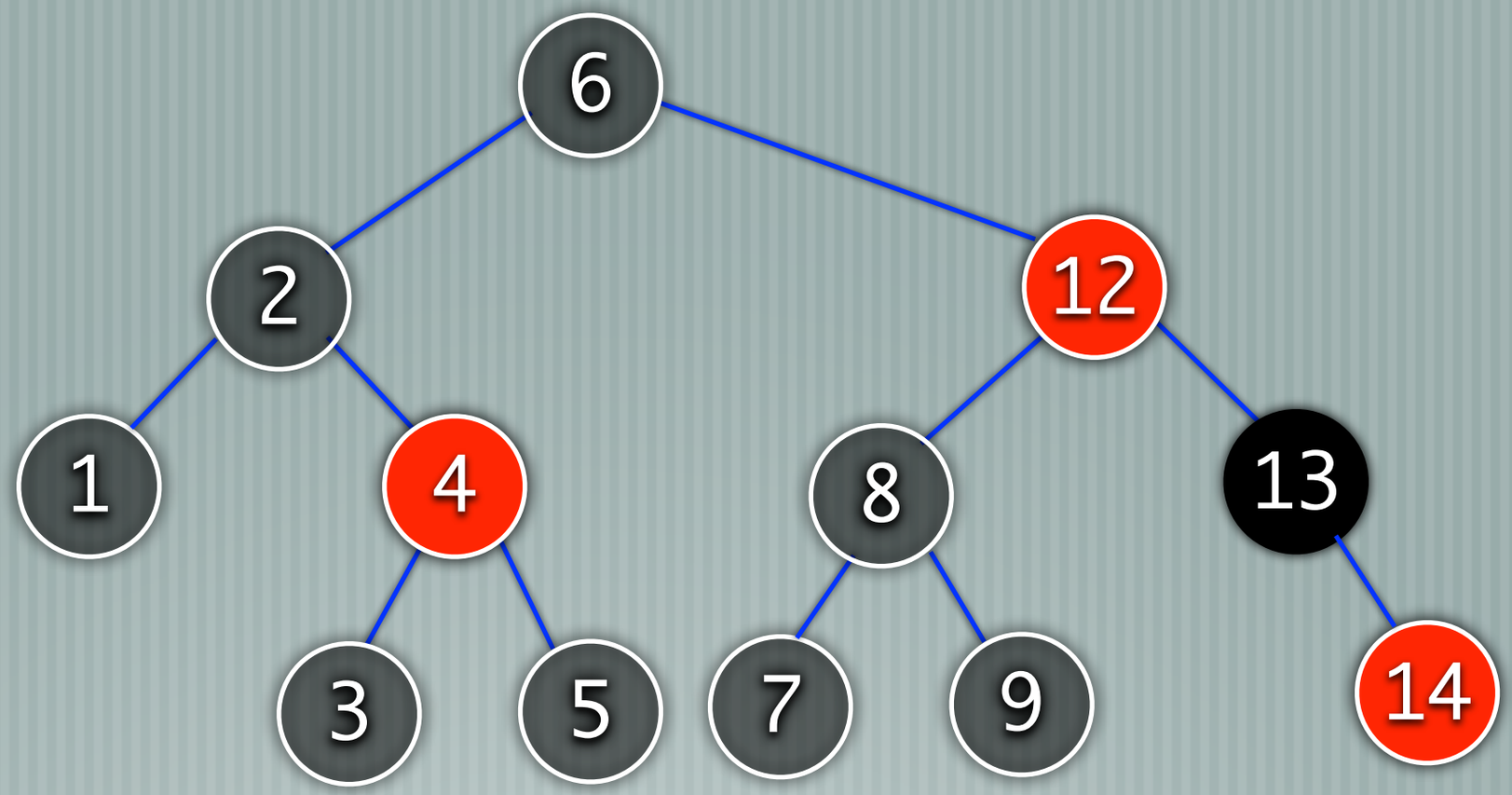

B
x
A
F
y
cd
G
1f
F
B
G
x
A
cd

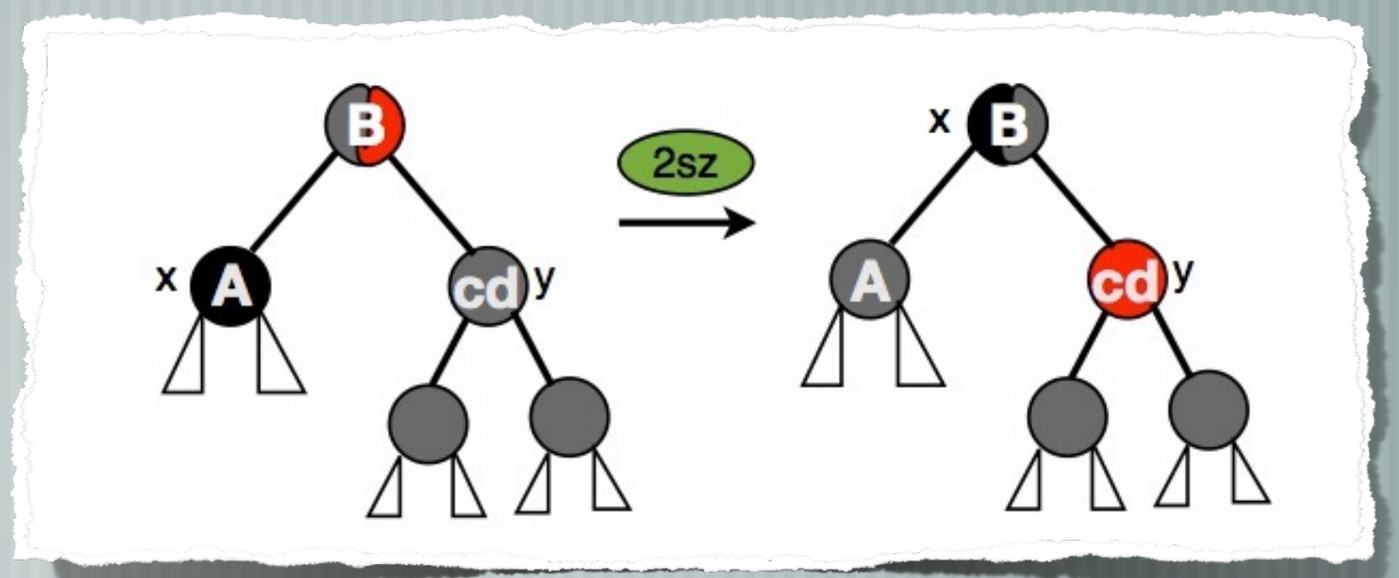

F
12
B
13
6
2
8
14
1
4
7
9
3
5

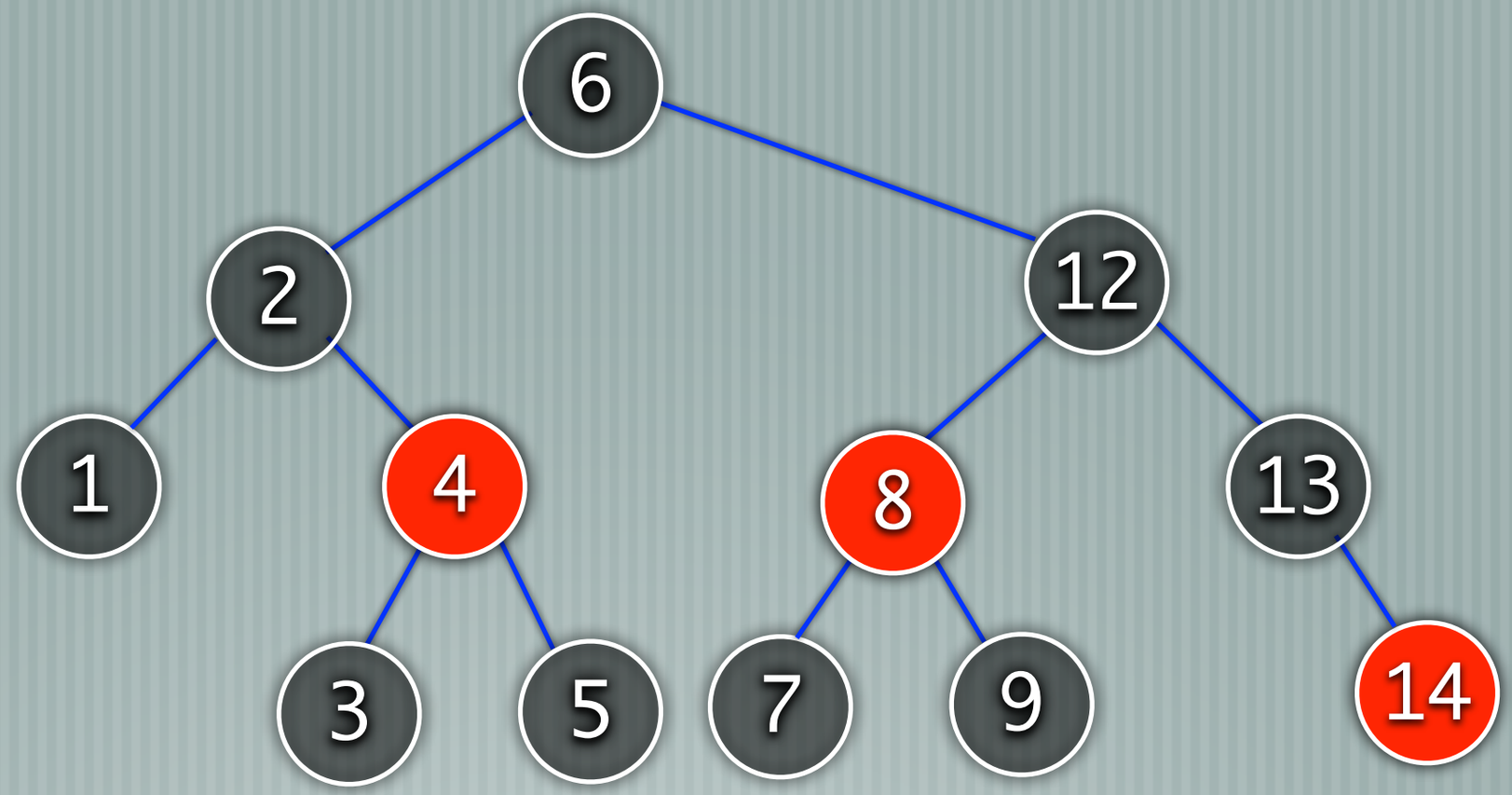

B
x
A
F
y
cd
G
1f
F
B
G
x
A
cd

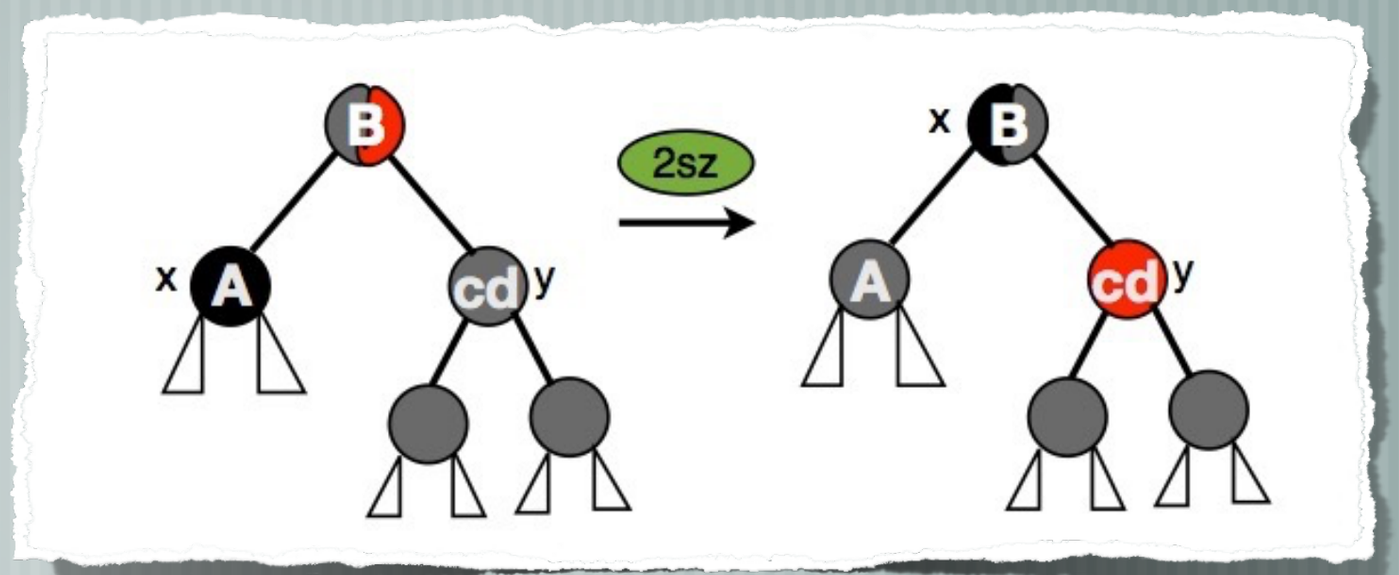

F
6
B
2
12
1
4
8
13
3
5
7
9
14


B
x
A
F
y
cd
G
1f
F
B
G
x
A
cd


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


B
x
A
cd
y
2sz
x
B
A
cd
y


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


B
x
A
cd
y
2sz
x
B
A
cd
y

ÖNSZERVEZŐ BINÁRIS KERESŐFÁK

vágható-egyesíthető halmaz adattípus

VÁGHATÓ-EGYESÍTHETŐ HALMAZ ADATTÍPUS

$$Vag(H,k): H_1, H_2$$

$$H_1 = \{x \in H : x < k\}$$
$$H_2 = \{x \in H : x \geq k\}$$

$$\max(H_1) \leq \min(H_2)$$

$$Egyesit(H_1, H_2): H$$

$$H = H_1 \cup H_2$$

$$H=\{2,5,7,11,23,45,75\} \underset{Egyesit(H_1,H_2)}{\overset{Vag(H,23)}{\rightleftarrows}} \begin{array}{l} H_1=\{2,5,7,11\} \\ H_2=\{23,45,75\} \end{array}$$

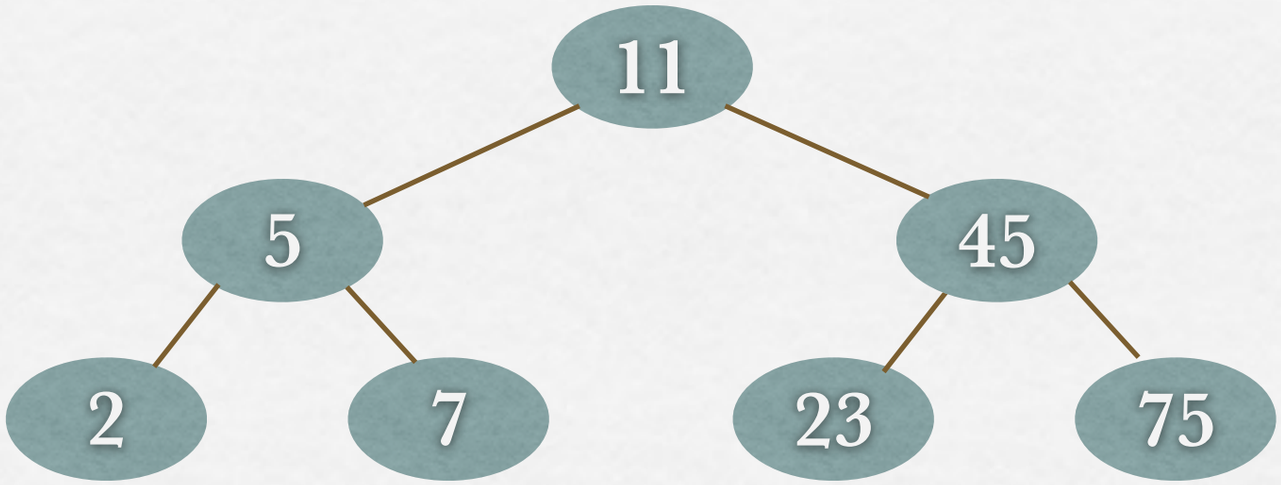
VÁGÁS

A keresési útvonal mentén feldaraboljuk a fát, majd a megfelelő részfákat összekapcsoljuk.

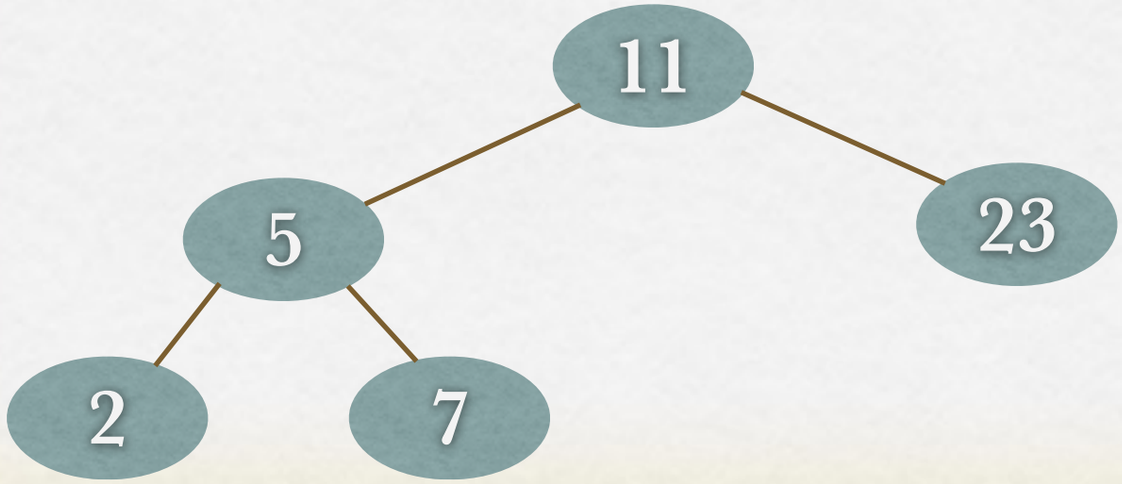
A vágás futási ideje legrosszabb esetben O(h(F))

$H=\{2,5,7,11,23,45,75\}$

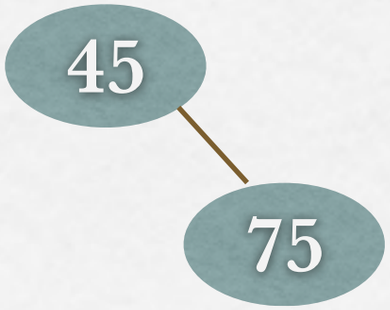
$Vag(H,24)$



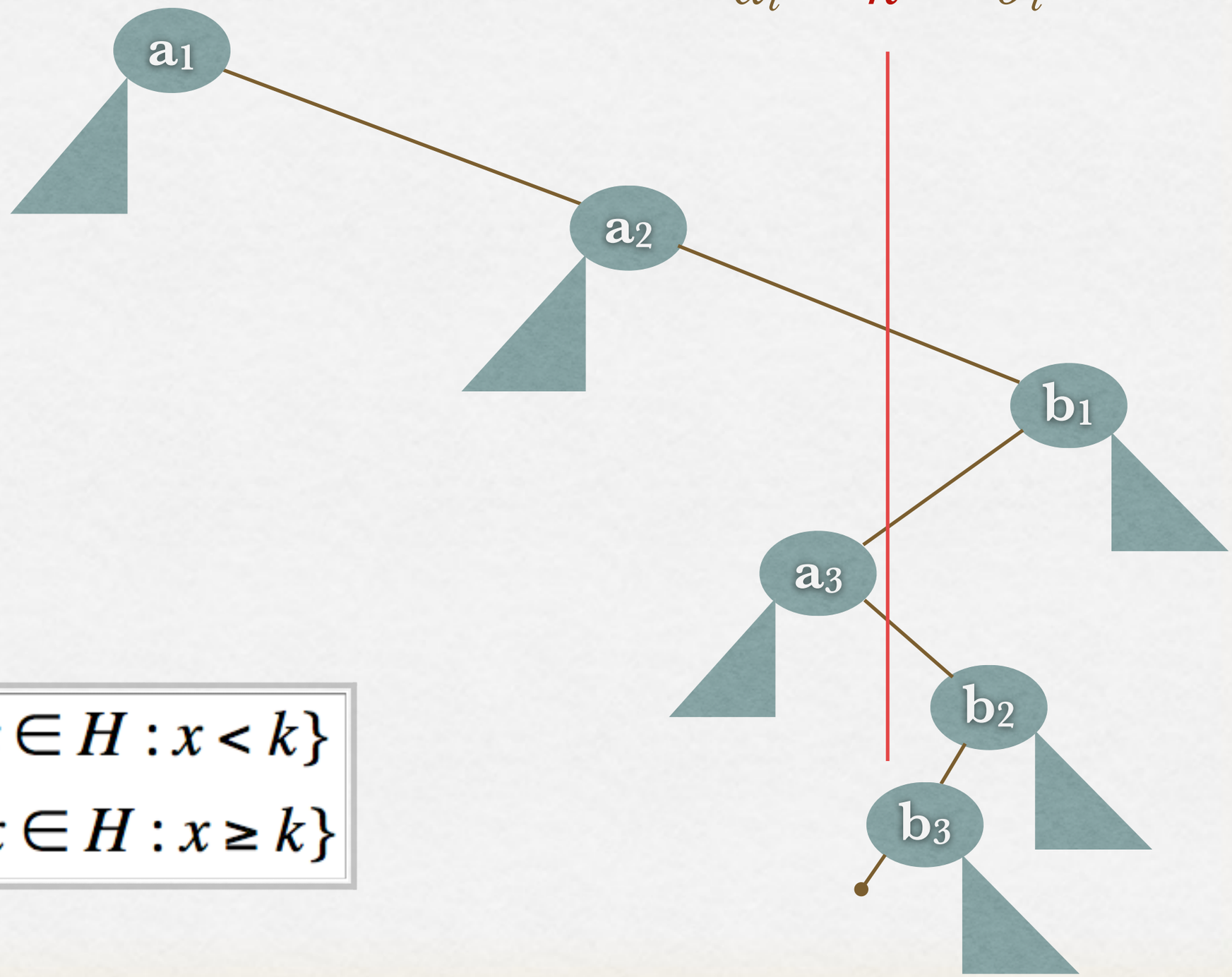
$H_1=\{2,5,7,11,23\}$



$H_2=\{45,75\}$

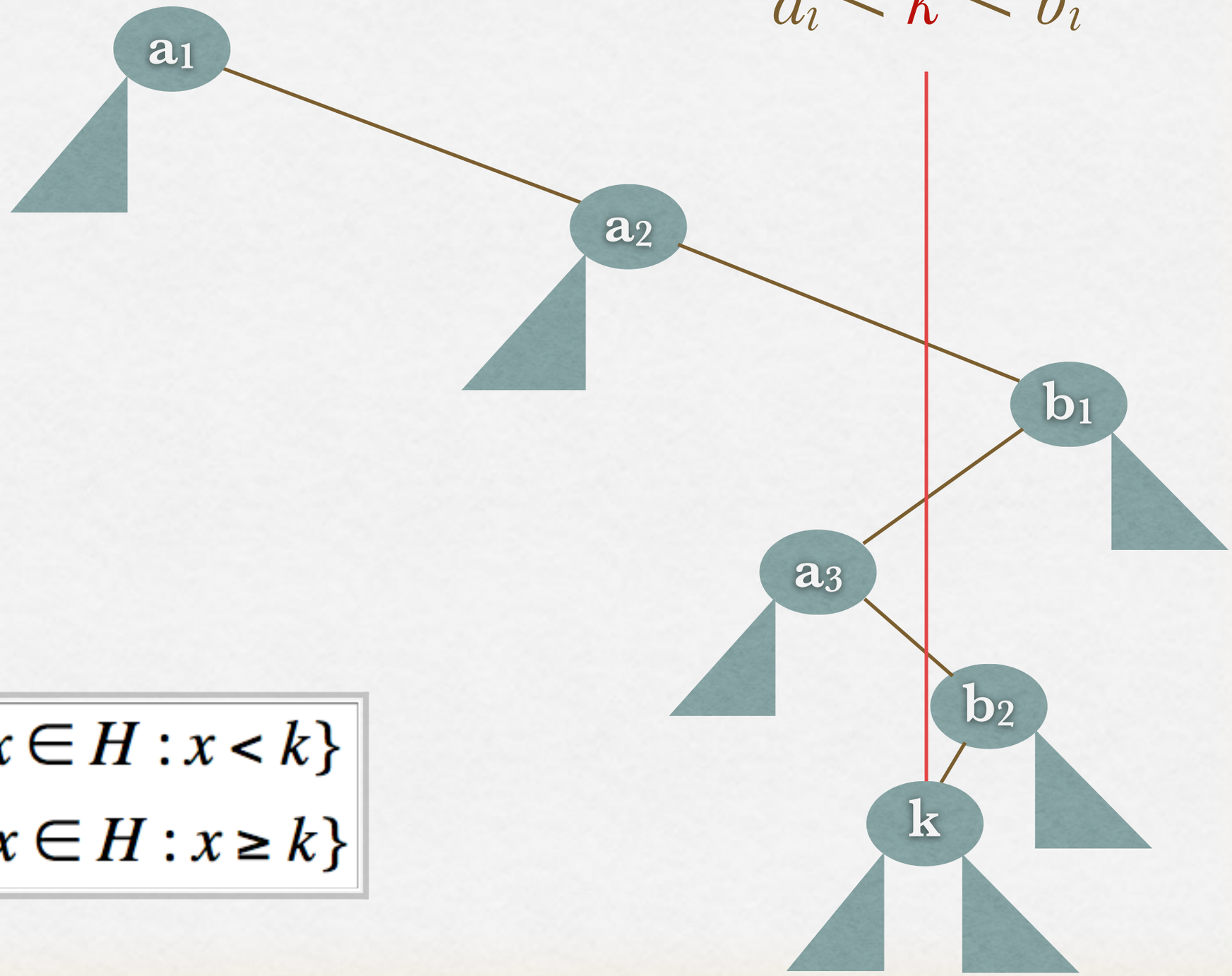


$$a_i < k < b_i$$



$$H_1 = \{x \in H : x < k\}$$
$$H_2 = \{x \in H : x \geq k\}$$

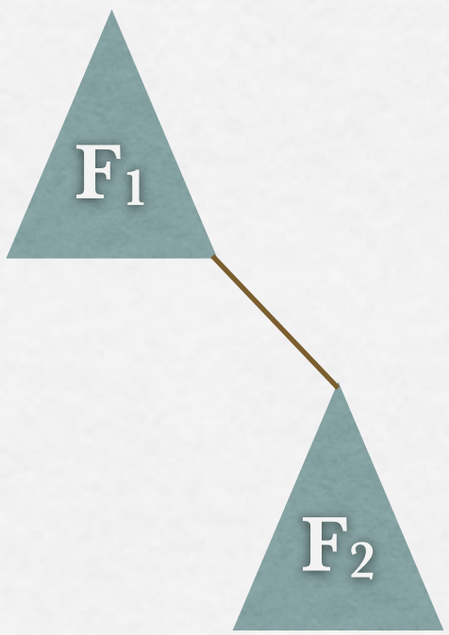
$$a_i < k < b_i$$



$$H_1 = \{x \in H : x < k\}$$
$$H_2 = \{x \in H : x \geq k\}$$

EGYESÍTÉS

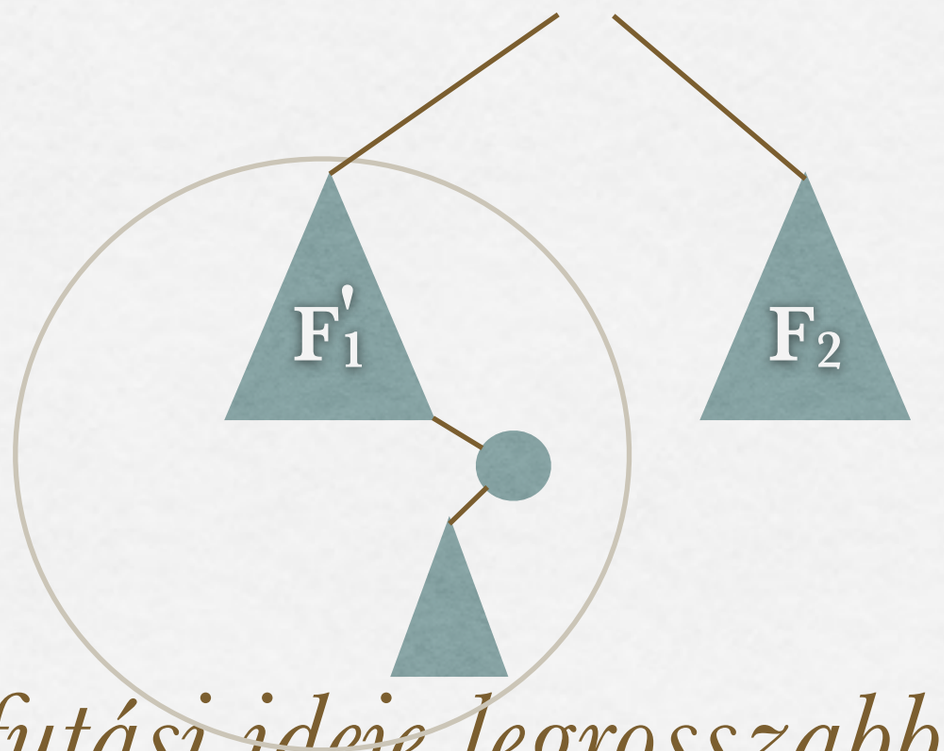
Ha a $max(F_1) \leq min(F_2)$ feltétel teljesül:



Az egyesítés futási ideje legrosszabb esetben O(1)

EGYESÍTÉS2

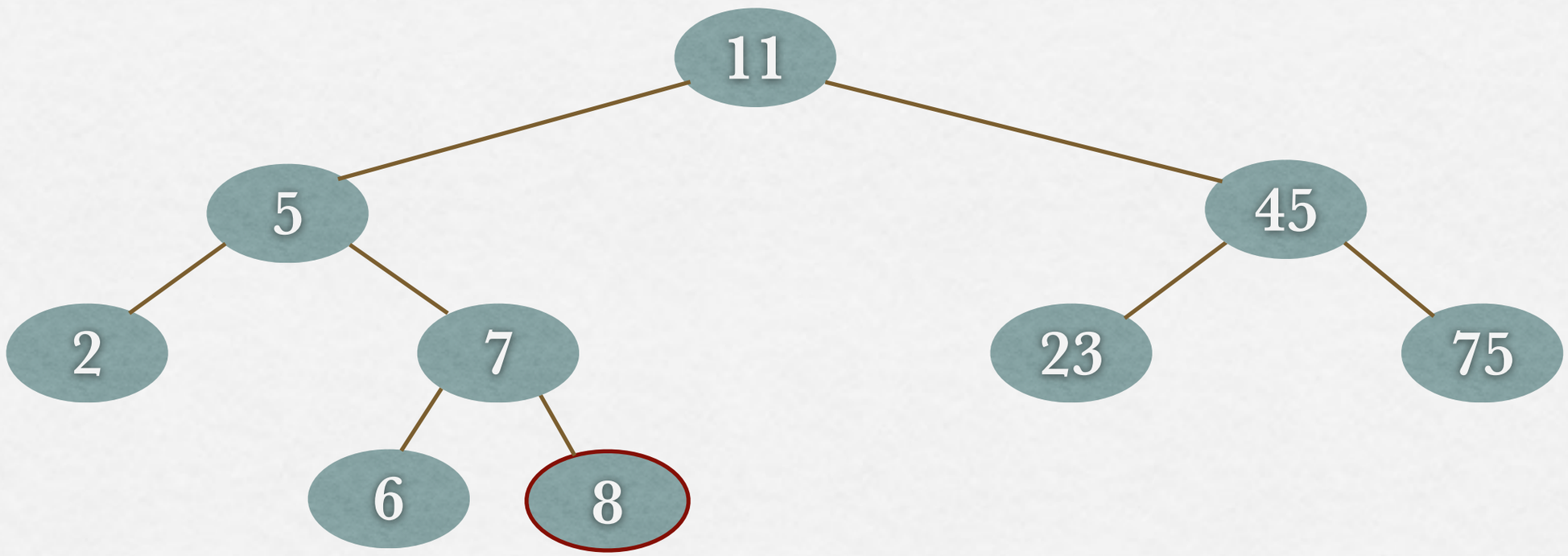
Ha a max(F_1)≤min(F_2) feltétel teljesül:



Az egyesítés futási ideje legrosszabb esetben O(h(F))

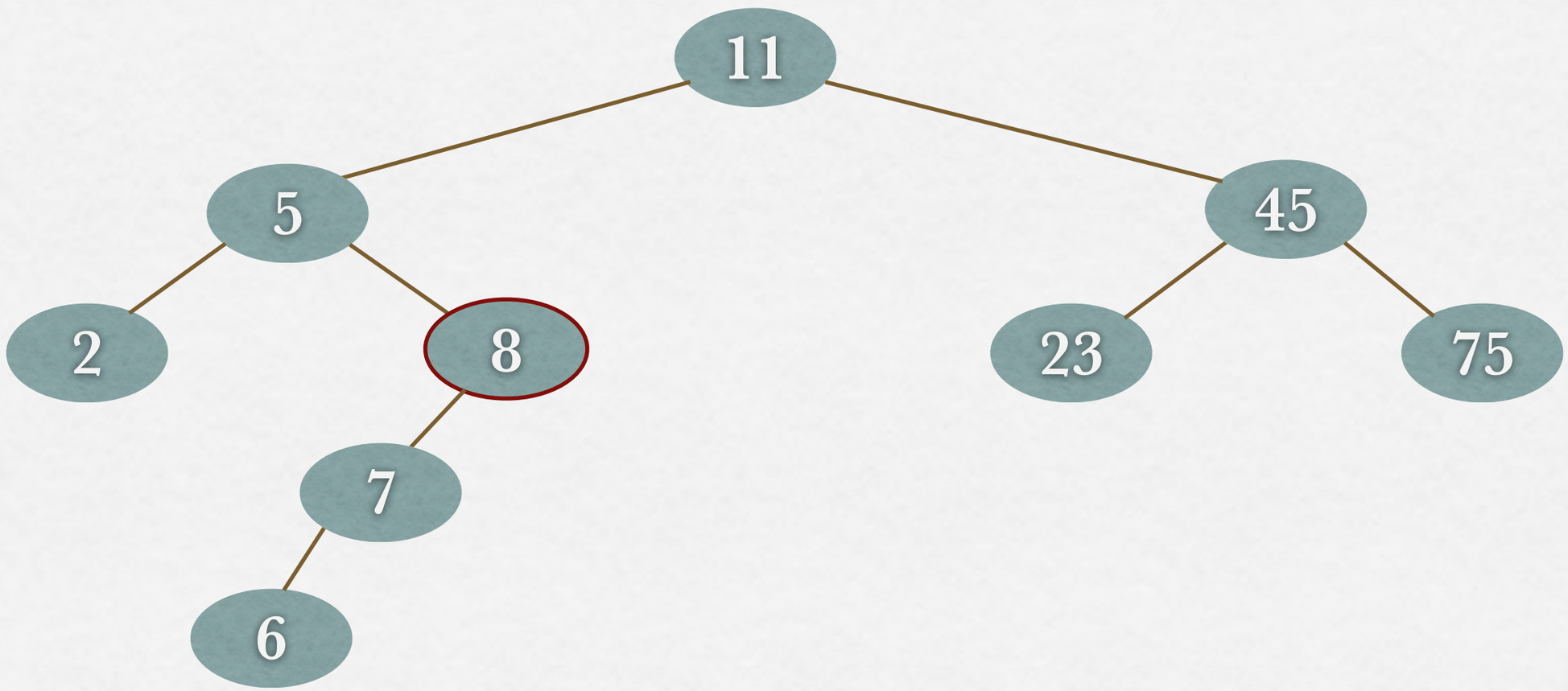
FELFORGAT

F.ff(fapont p) ;



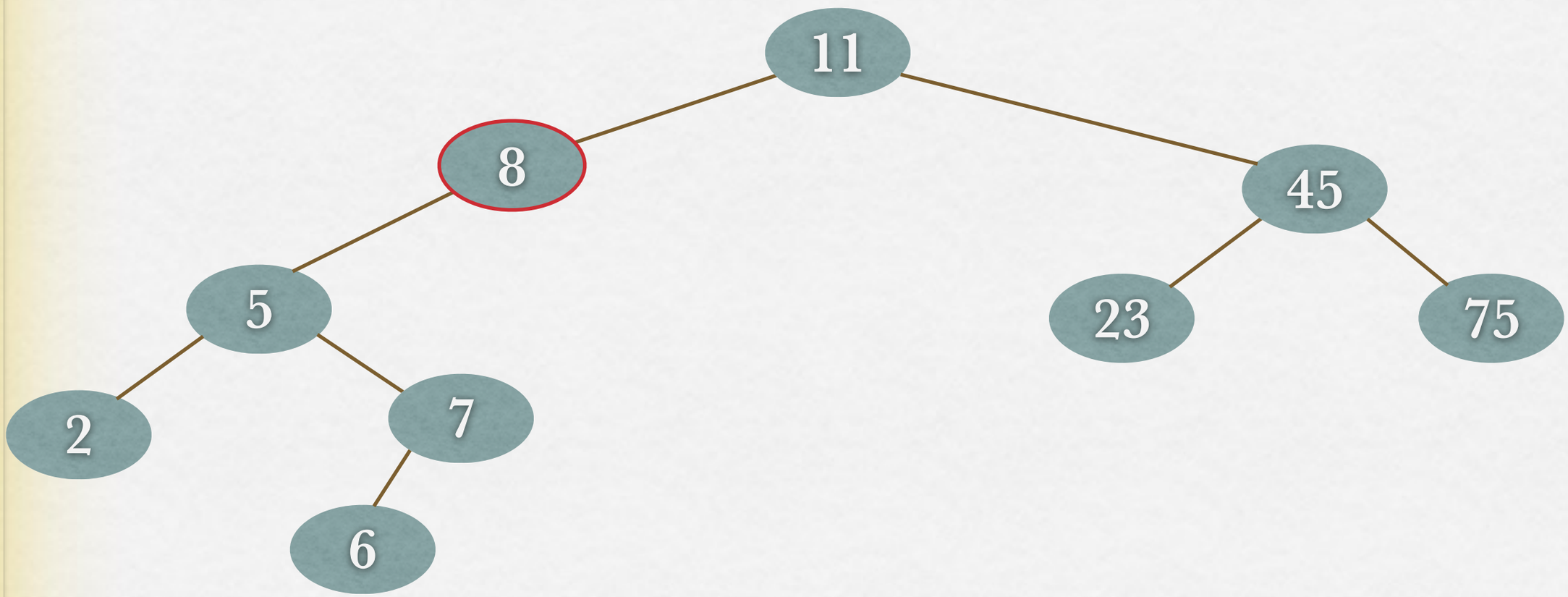
FELFORGAT

F.ff(fapont p) ;



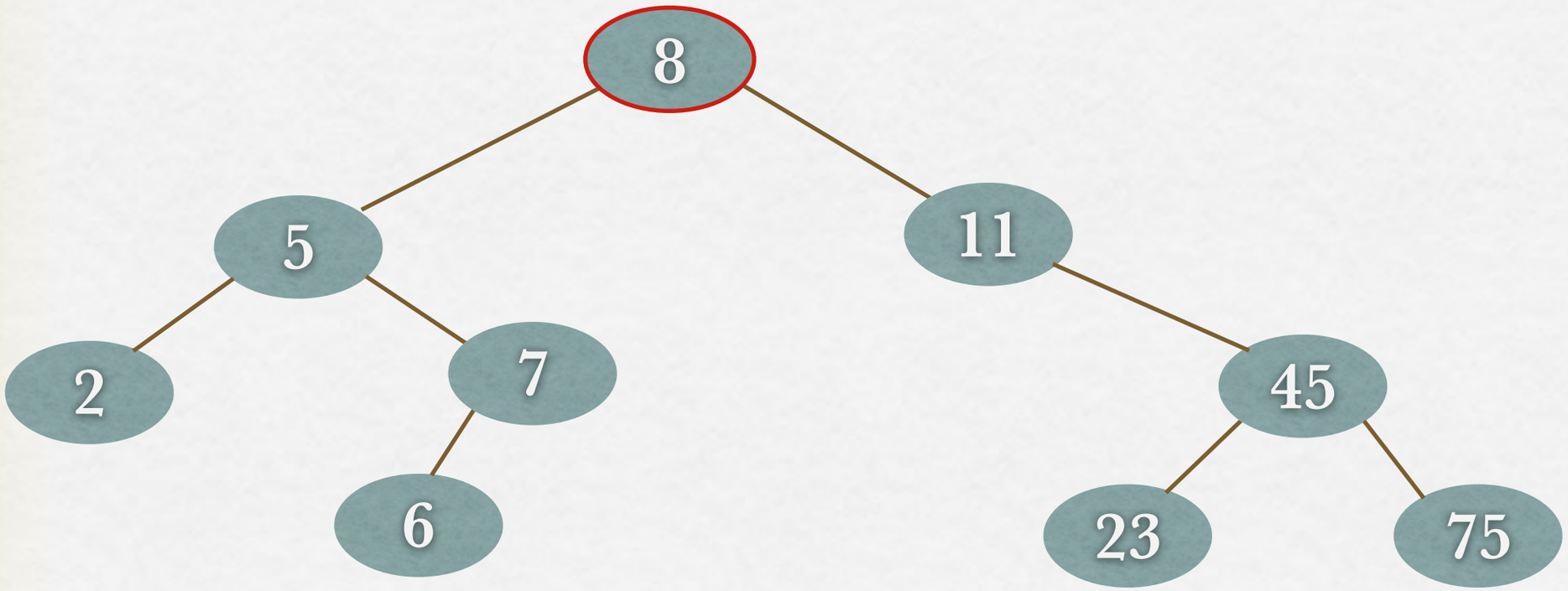
FELFORGAT

F.ff(fapont p) ;



FELFORGAT FF(P)

F.ff(fapont p) ;

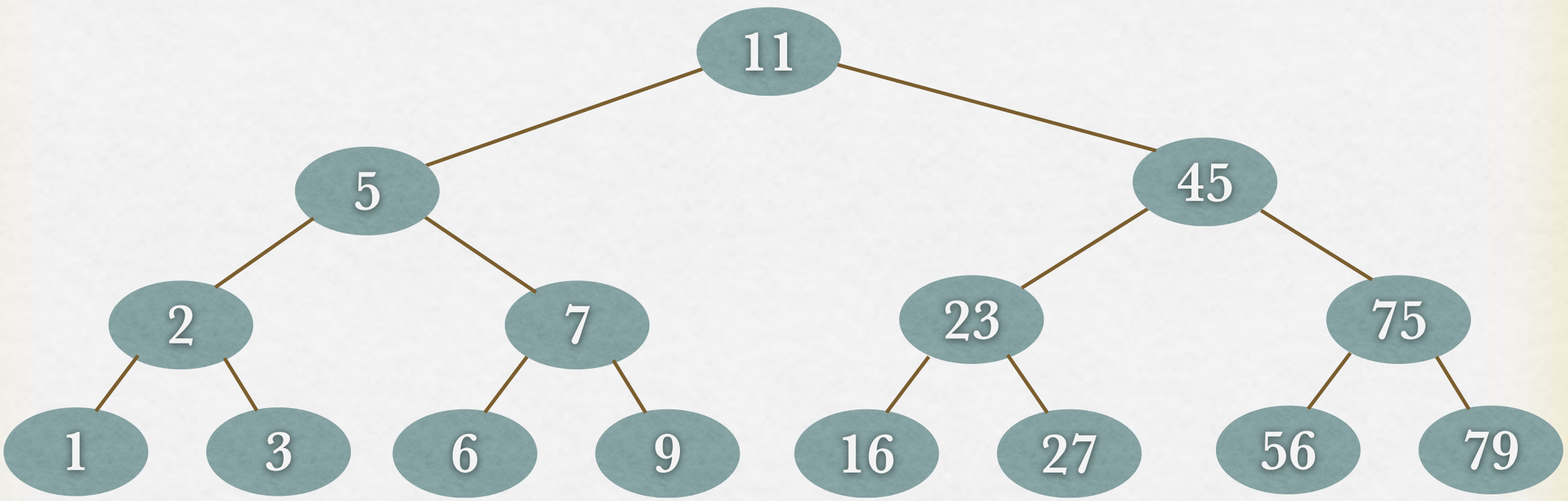


A felforgat futási ideje legrosszabb esetben O(h(F))

FF(K)

F.ff(int k) ;

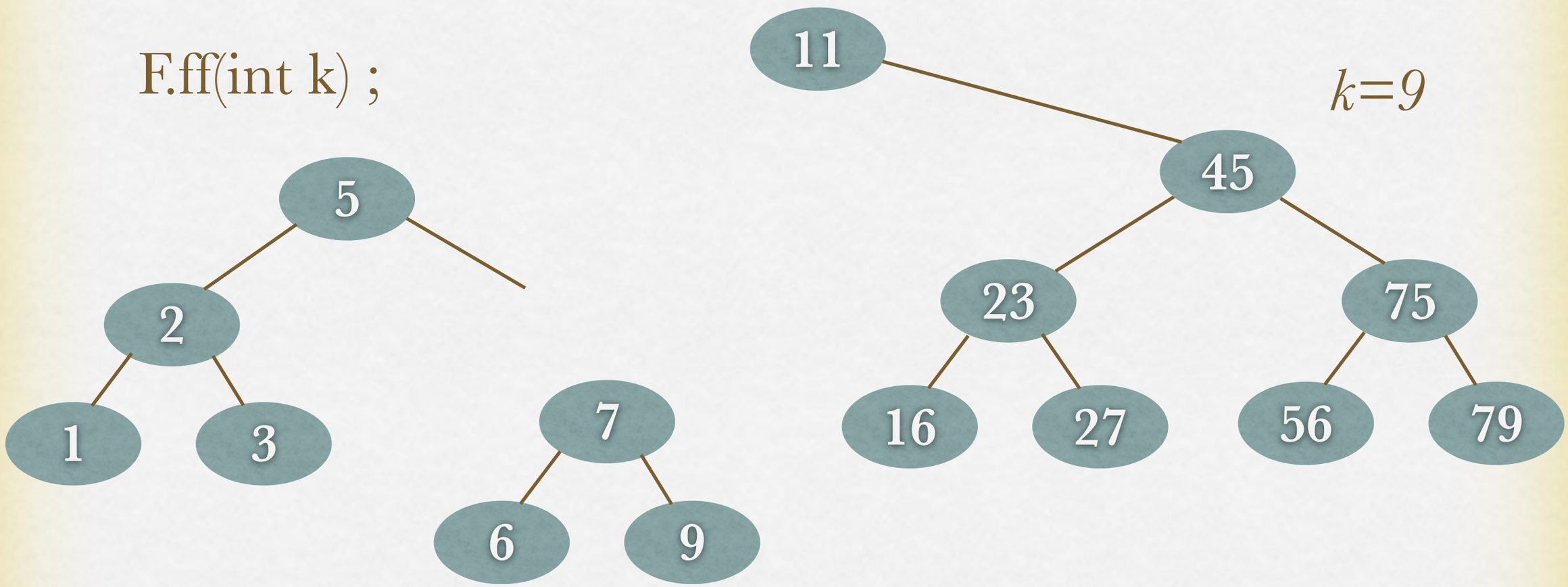
k=9



FF(K)

F.ff(int k) ;

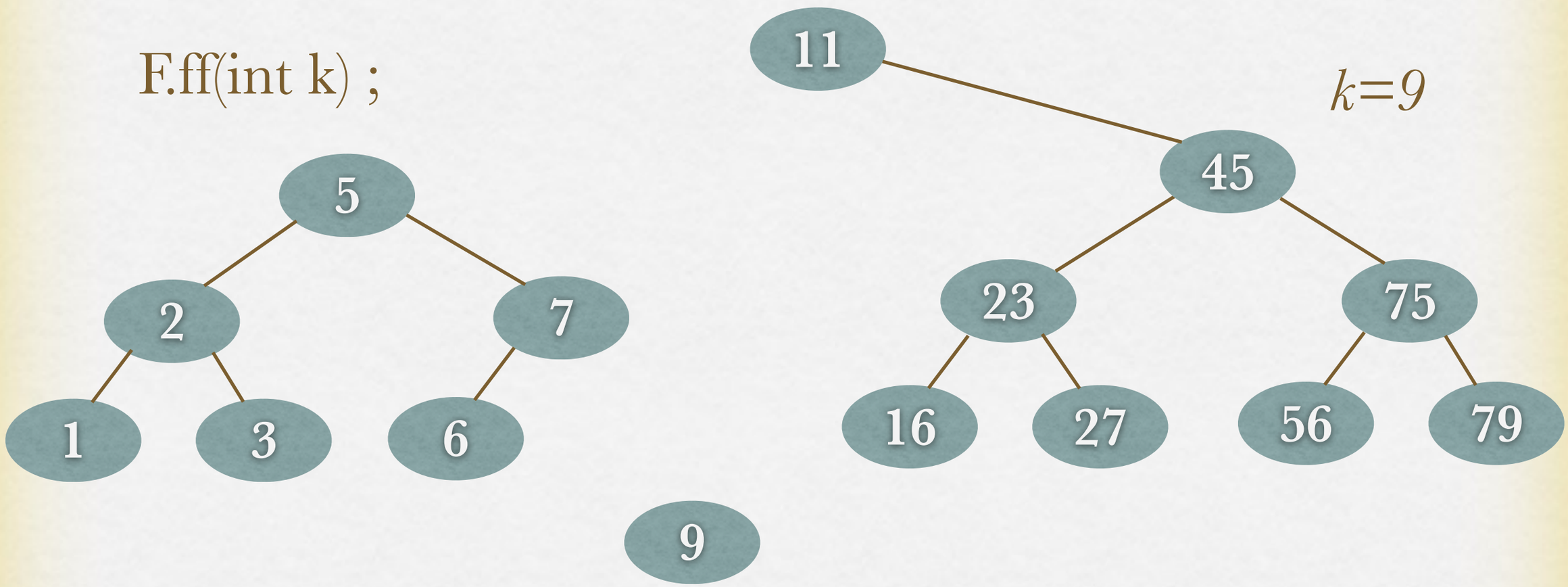
k=9



FF(K)

F.ff(int k) ;

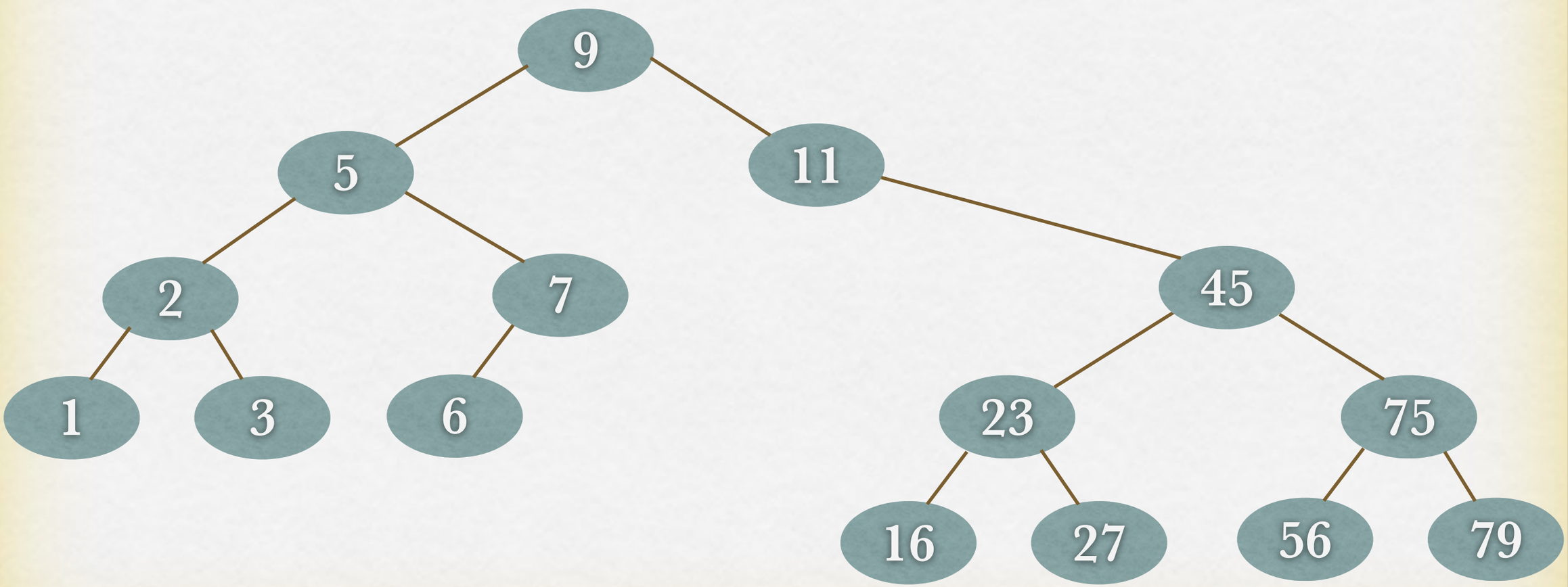
k=9



FF(K)

F.ff(int k) ;

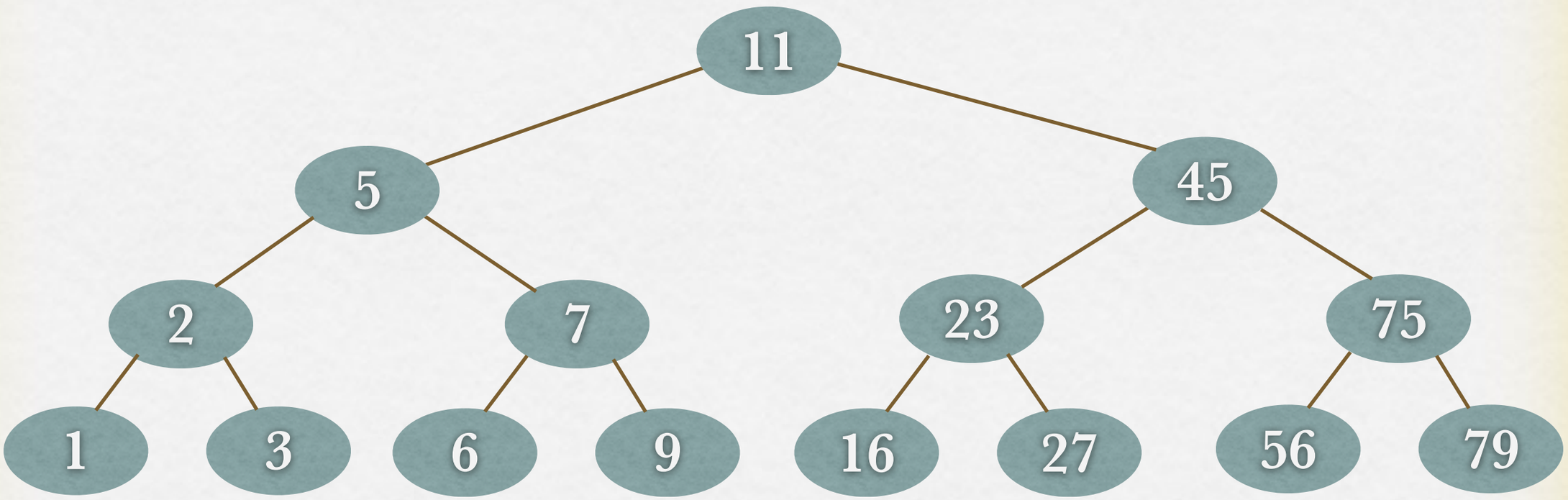
k=9



FF(K)

F.ff(int k) ;

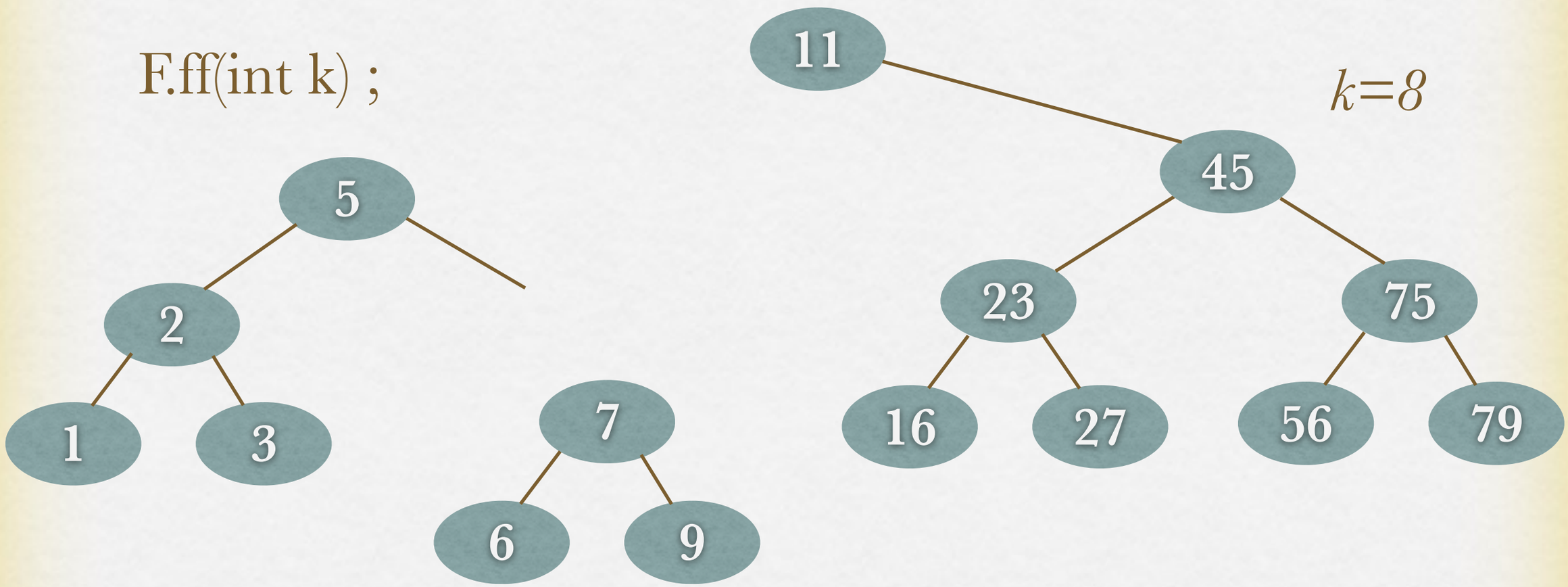
k=8



FF(K)

F.ff(int k) ;

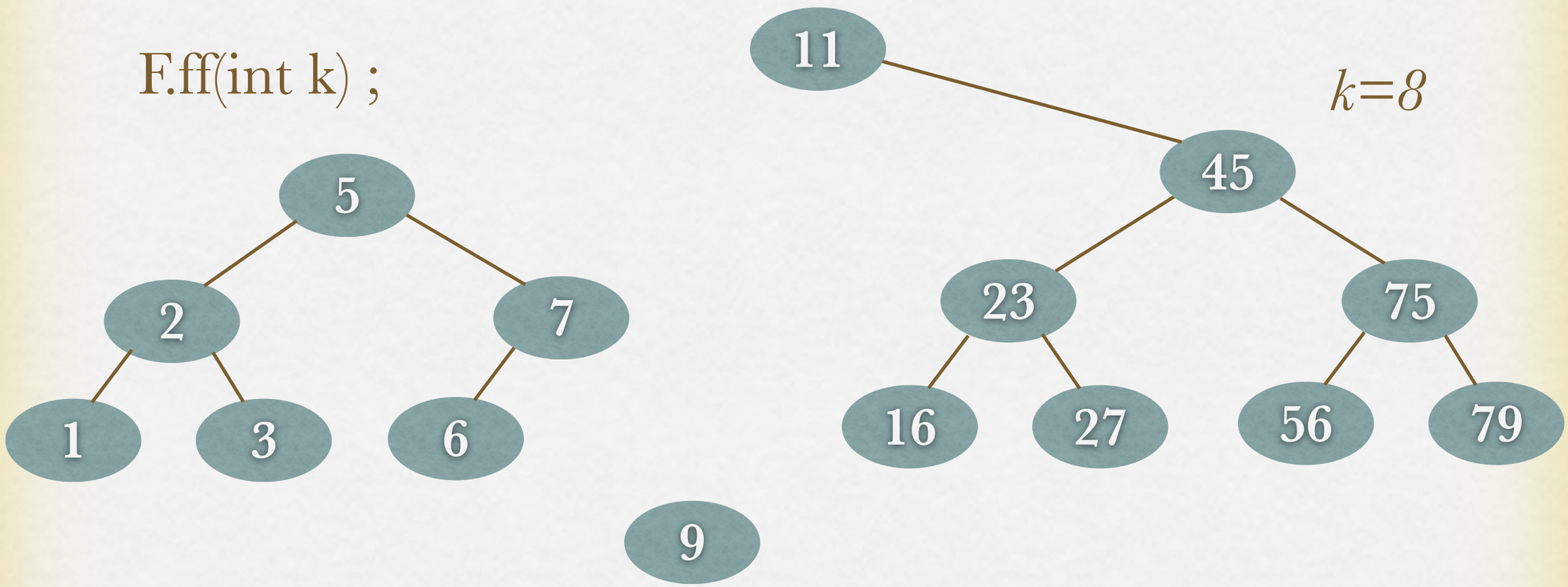
k=8



FF(K)

F.ff(int k) ;

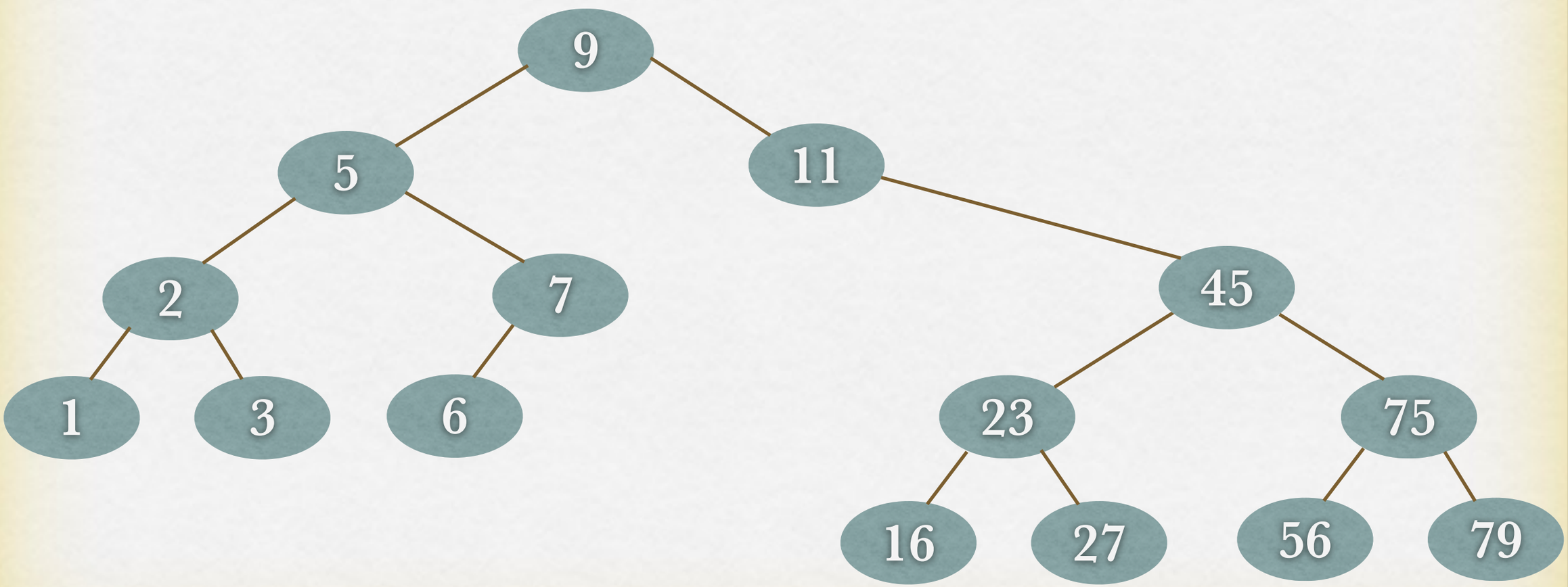
k=8



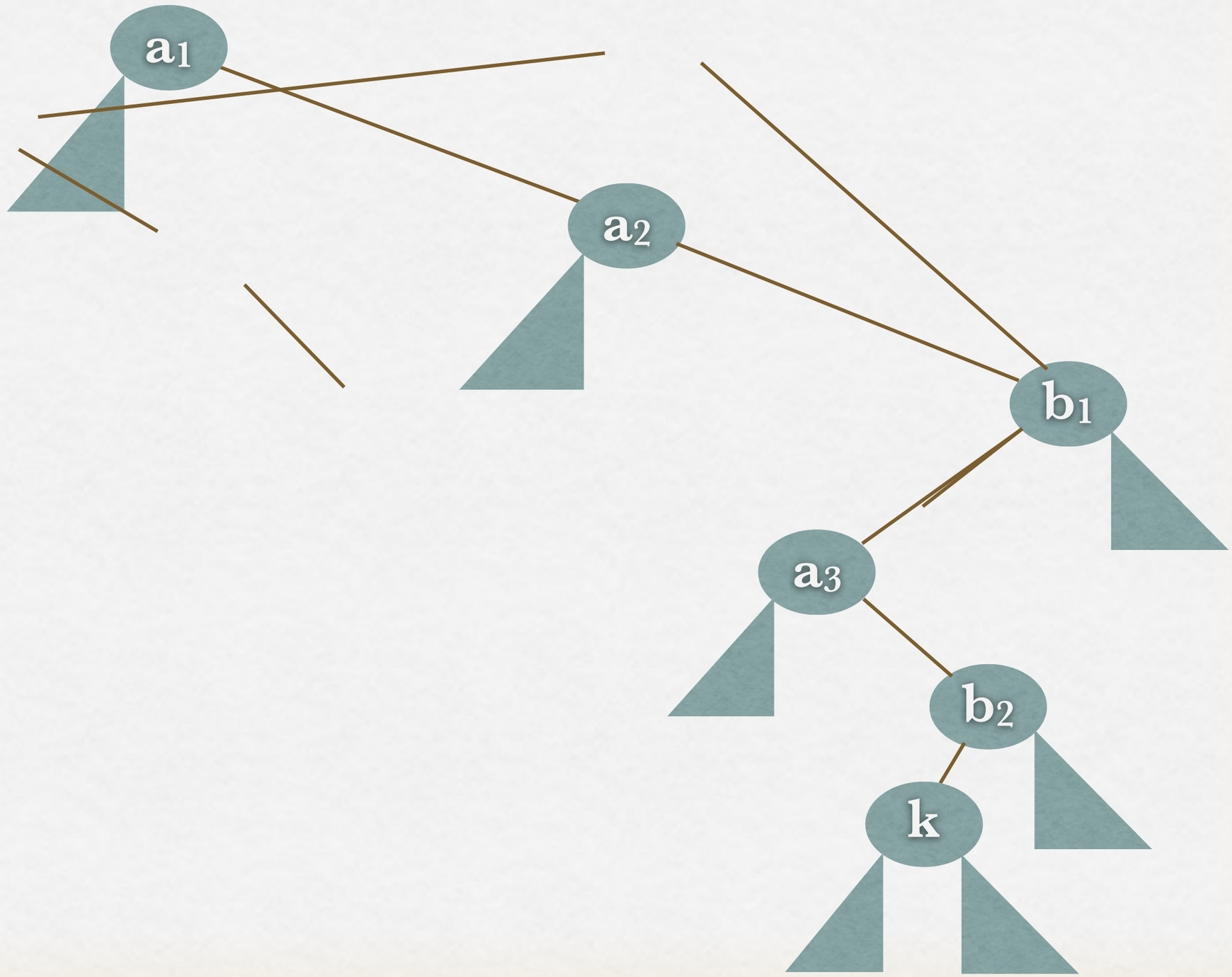
FF(K)

F.ff(int k) ;

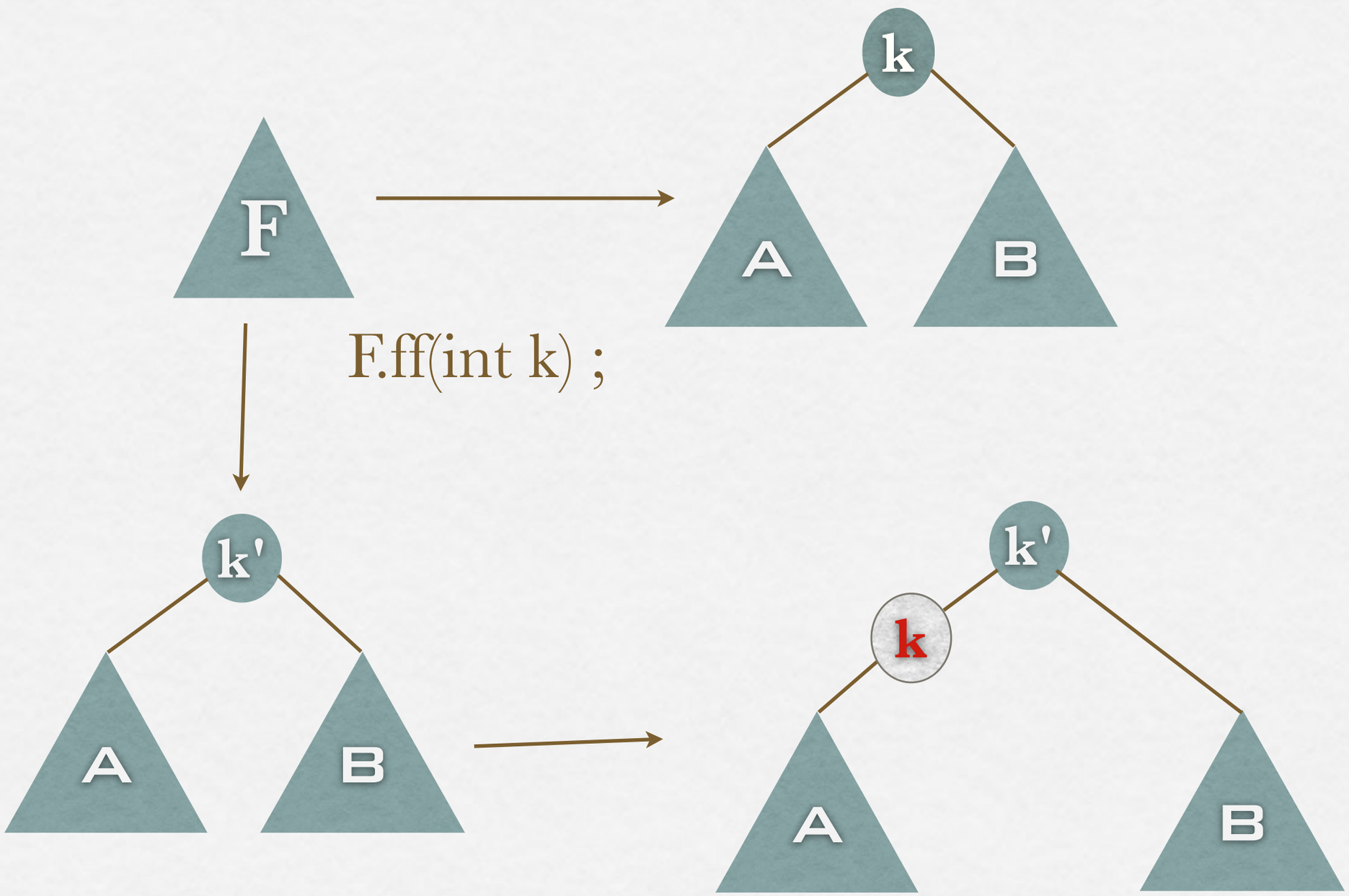
k=8



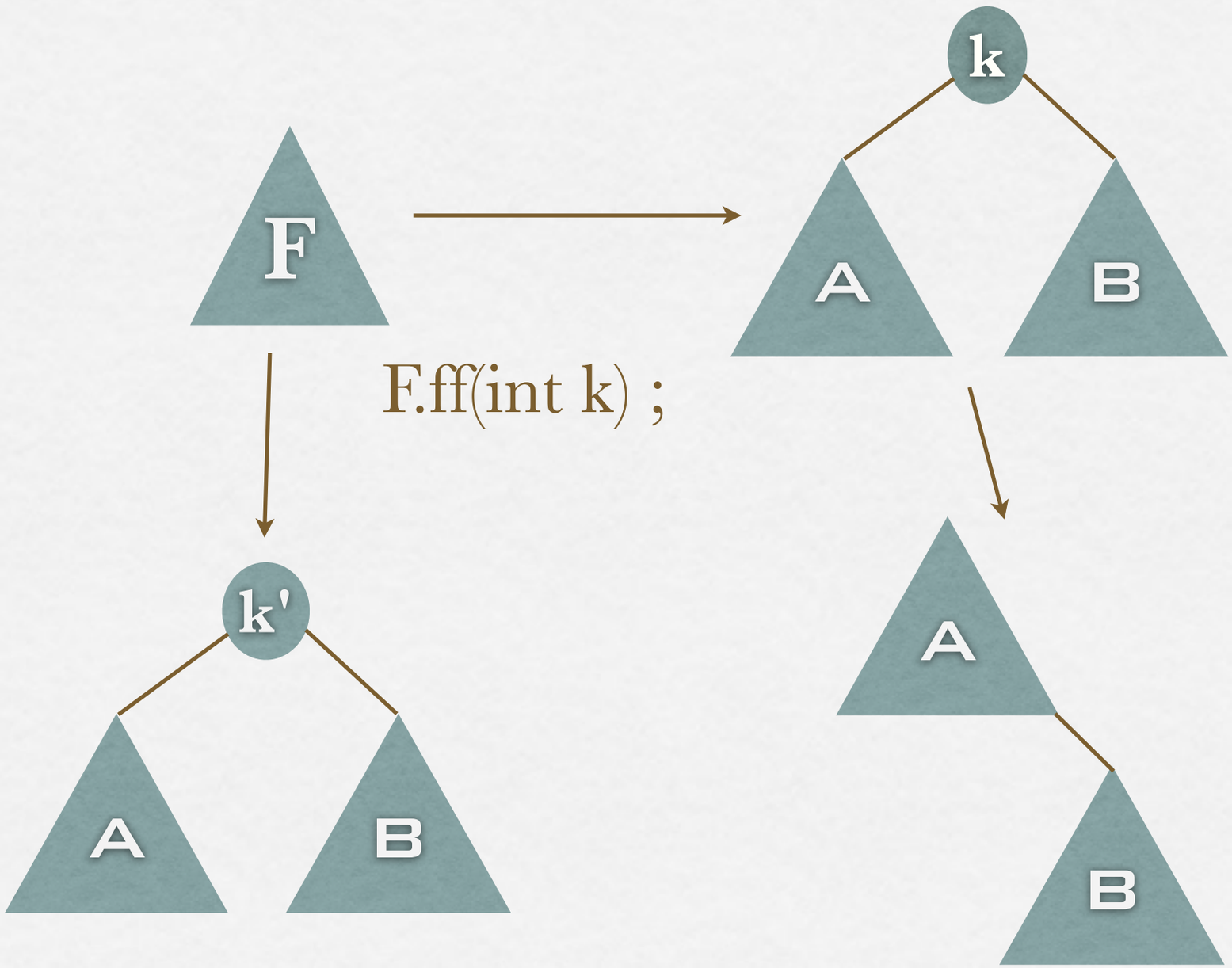
$a_i < k < b_i$



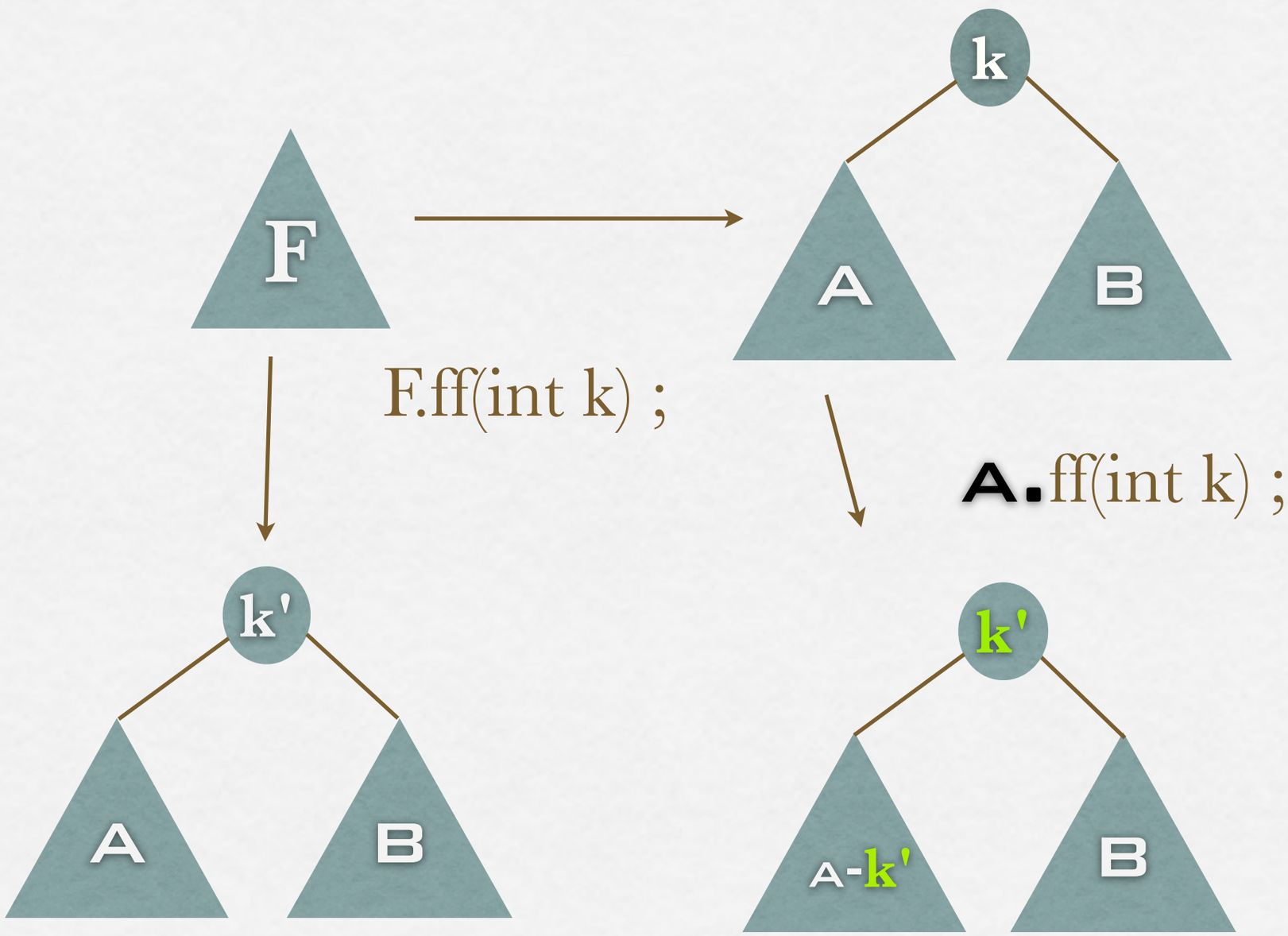
F.bovit(k) ;



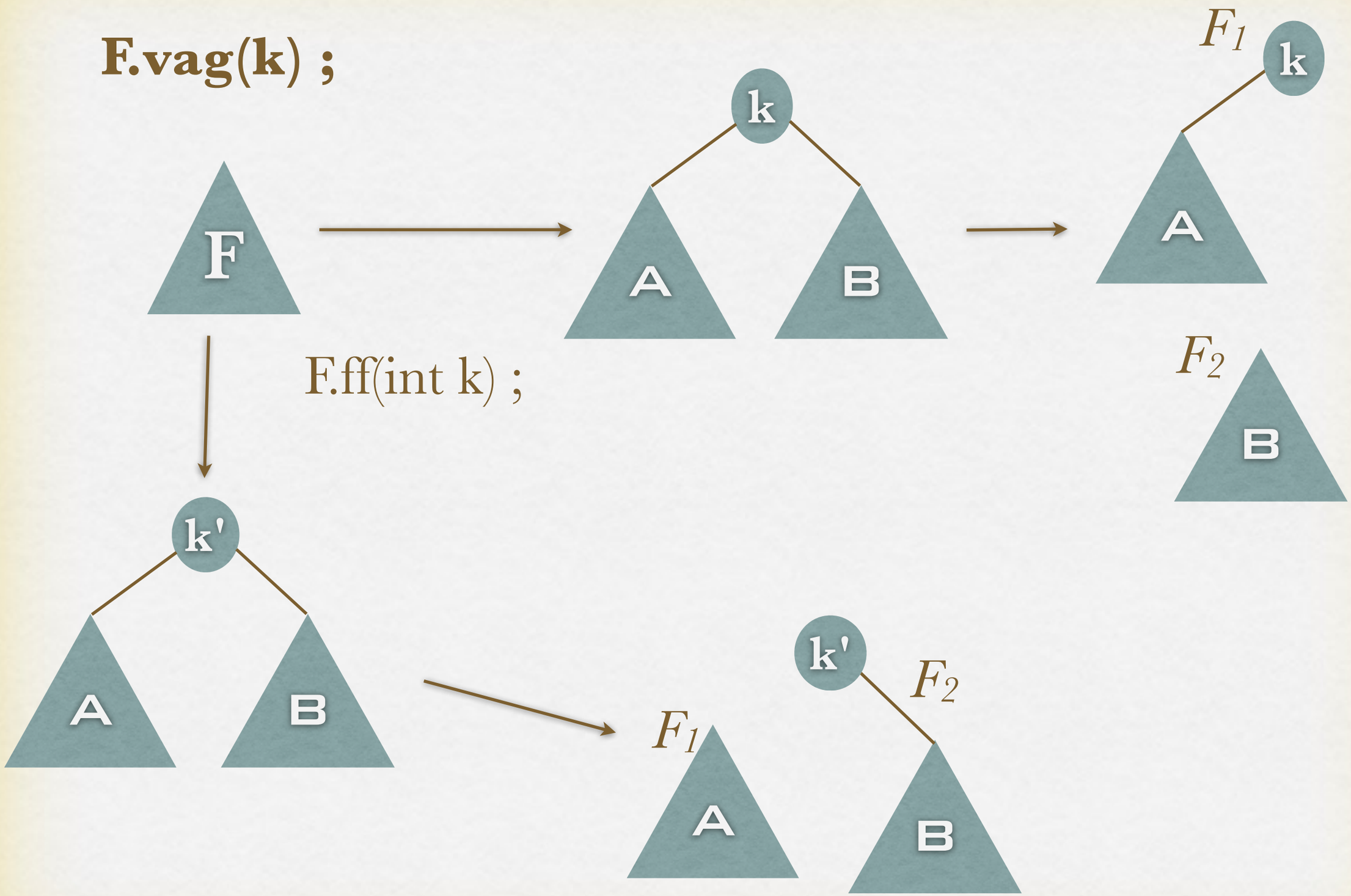
F.torol(k) ;



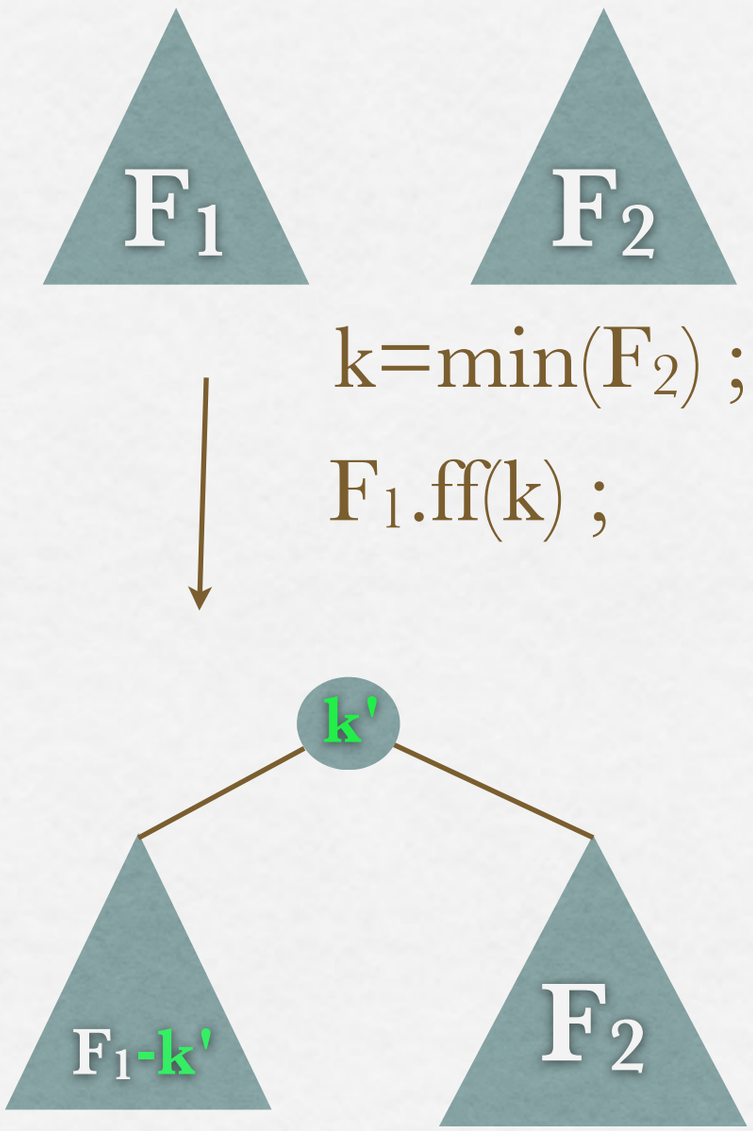
F.torol(k) ;



F.vag(k) ;



F_1.egyesit(F_2) ;



ÖNSZERVEZŐ BINÁRIS KERESŐFÁK HATÉKONYSÁGA

Tétel: Ha a halmazok ábrázolására önszervező bináris keresőfát használunk, akkor minden $\alpha_1,\ldots,\alpha_m$ műveletsor, ahol $\forall i \in \{1..m\}$: $\alpha_i \in \{$*keres*;*bovit*;*torol*;*vag*;*egyesit*$\}$ összesített futási ideje $O(m \lg n)$, ahol n a BOVIT műveletek száma és a műveletsor előtt üres halmazzal indulunk.

AMORTIZÁCIÓS KÖLTSÉGELEMZÉS

A műveletsor futási idejének meghatározásához fejezzük ki minden egyes művelet futási idejét, és azt összegezzük!

Verem adattípus ***betesz(x); kivesz();***

O(1) ***O(1)***

BVEREM *adattípus:*

VEZESSÜK BE ***TOROL(K);*** ÚJ MŰVELETET!

```
void torol (k) {
   while (--k>=0 && not ures()) kivesz();
}
```

torol(k); ***O(k)***

*Legyen **betesz(x); kivesz(); torol(k);** műveletekből tetszőleges sorrendben **m** darab, és kezdetben üres verem!*

*Mivel kivesz(k) futási ideje legrosszabb esetben **m** és a műveletekből **m** darab van, a futási idő legrosszabb esetben **m^2**?*

Legrosszabb esetre ez a módszer nem mindig éles!

Összesítéses elemzés

A teljes műveletsor összesített költségét számítjuk, nem az egyes műveletekét!

Minden elem a verembe egyszer kerülhet be és egyszer ki.

*M művelet esetén maximum **m** elem kerül a verembe,*

*így a futási idő **O(m)!***

A műveletekre átlagolt T(n)/m költséget számítjuk. Ezt az egyes műveletek átlagolt költségének nevezzük.

Könyvelési módszer

A könyvelési módszer esetén különböző amortizált költséget számítunk ez egyes műveletekre, megengedve azt is, hogy egyes műveletek esetén a tényleges költségnél többet, másoknál kevesebbet számlázzunk.
Az az összeg, amit egy adott műveletre felszámítunk, a művelet amortizációs költsége. Ha felső korlátot akarunk adni (a legrosszabb esetre), akkor az amortizációs összköltség a tényleges összköltség felső korlátja kell legyen minden n-re.

$$\sum_{i=1}^{n} \widehat{c_i} \geq \sum_{i=1}^{n} c_i$$

Potenciál módszer

Feltételezzük, hogy minden művelet ugyanazon D adatszerkezeten hajt végre műveletet.
Az adatszerkezet kezdeti helyzete D_0, az i-edik művelet végrehajtása után D_i.

Az előre kifizetett munkát nem az adatszerkezet egyes elemeihez rendeljük, hanem mint potenciális energiát (potenciált), a jövőbeni műveletek kifizetésére használhatjuk.

Ezt a potenciált az adatszerkezet egészéhez rendeljük hozzá. (pl. az adatszerkezetben tárolt elemek száma)

A φ potenciálfüggvény a
D_i adatszerkezethez egy
$\varphi(D_i)$ valós számot rendel,
ami a D_i adatszerkezethez rendelt potenciál.

Az i-edik művelet $\hat{c}_i$ amortizációs költségét
a φ potenciálfüggvényre vonatkozóan
a következő egyenlettel definiáljuk:

$$\hat{c}_i = c_i + \varphi(D_i) - \varphi(D_{i-1})$$

$$\hat{c}_i = c_i + \varphi(D_i) - \varphi(D_{i-1})$$

A teljes amortizációs költség:

$$\sum_{i=1}^{m} \hat{c}_i = \sum_{i=1}^{m} (c_i + \varphi(D_i) - \varphi(D_{i-1})) =$$

$$= \sum_{i=1}^{m} c_i + \varphi(D_1) - \varphi(D_0) + \varphi(D_2) - \varphi(D_1) + \ldots + \varphi(D_m) - \varphi(D_{m-1})$$

$$= \sum_{i=1}^{m} c_i + \varphi(D_m) - \varphi(D_0)$$

$$\sum_{i=1}^{m} \hat{c}_i = \sum_{i=1}^{m} c_i + \varphi(D_m) - \varphi(D_0)$$

Legyen

$$\varphi(D_m) - \varphi(D_0) \geq 0$$

azaz

$$\varphi(D_m) \geq \varphi(D_0)$$

$$\sum_{i=1}^{m} \hat{c}_i \geq \sum_{i=1}^{m} c_i$$

Az amortizált költség felső korlátja a tényleges költségnek!

pl.: BVEREM adattípus amortizációs költségelemzése

Definiáljuk φ potenciálfüggvényt a veremben lévő elemek számaként!

$$\varphi(D_0) = 0 \qquad \varphi(D_m) \geq \varphi(D_0) = 0$$

betesz(x);

$$\Delta\varphi: \quad \varphi(D_i) - \varphi(D_{i-1}) = 1$$

$$\hat{c}_i = c_i + \varphi(D_i) - \varphi(D_{i-1}) = 1 + 1 = 2 \qquad O(1)$$

kivesz();

$$\Delta\varphi: \quad \varphi(D_i) - \varphi(D_{i-1}) = -1$$

$$\hat{c}_i = c_i + \varphi(D_i) - \varphi(D_{i-1}) = 1 - 1 = 0 \qquad O(1)$$

betesz(x) művelet költsége *O(1)*.
kivesz() művelet költsége *O(1)*.

torol(k); művelet végrehajtásakor a potenciális változás:

$$\Delta\varphi:\ \varphi(D_i)-\varphi(D_{i-1})=-k'$$

$$k'=\min(|V|,k)$$

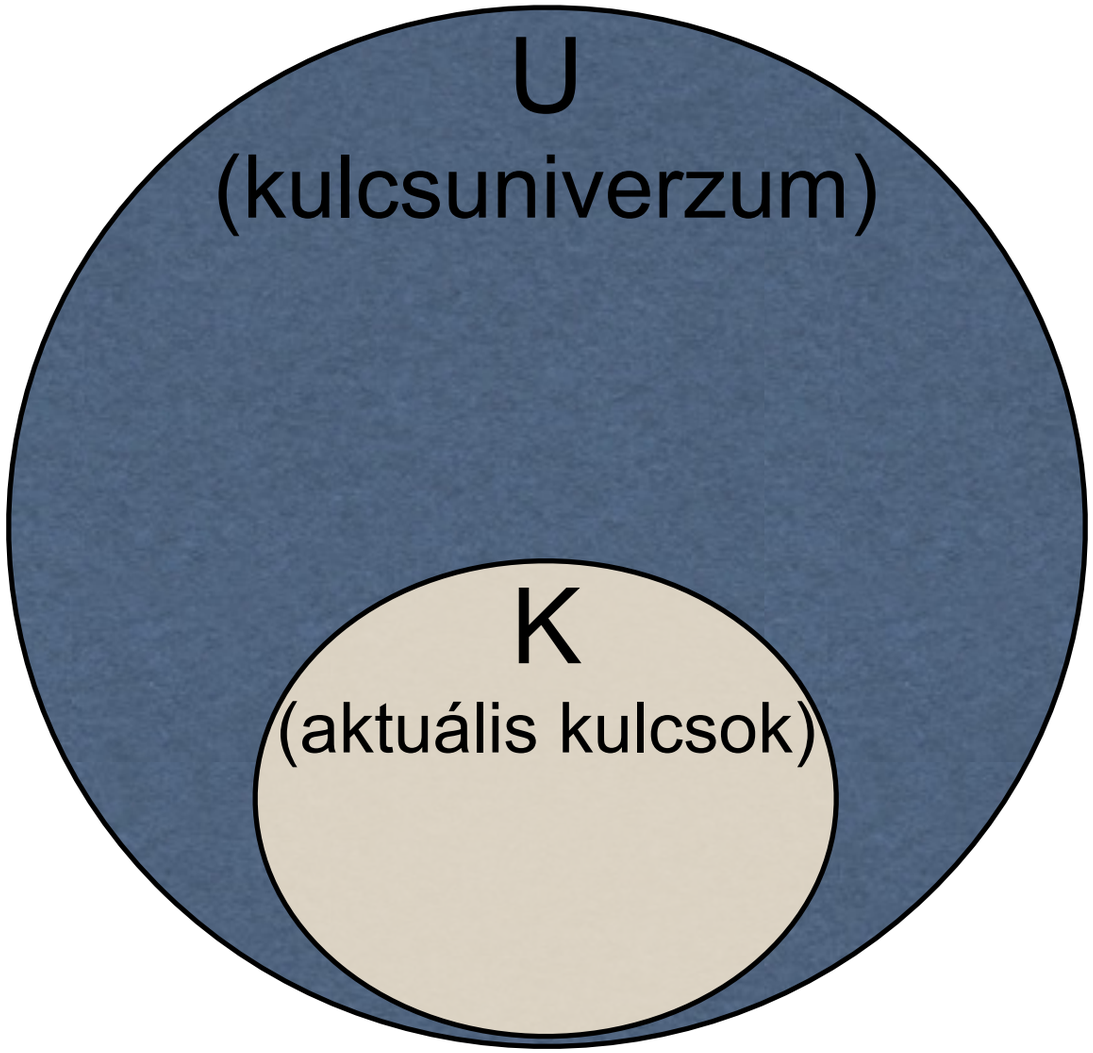
$$\hat{c}_i = \overset{k'}{c_i} + \overbrace{\varphi(D_i)-\varphi(D_{i-1})}^{-k'} = 0$$

O(1)

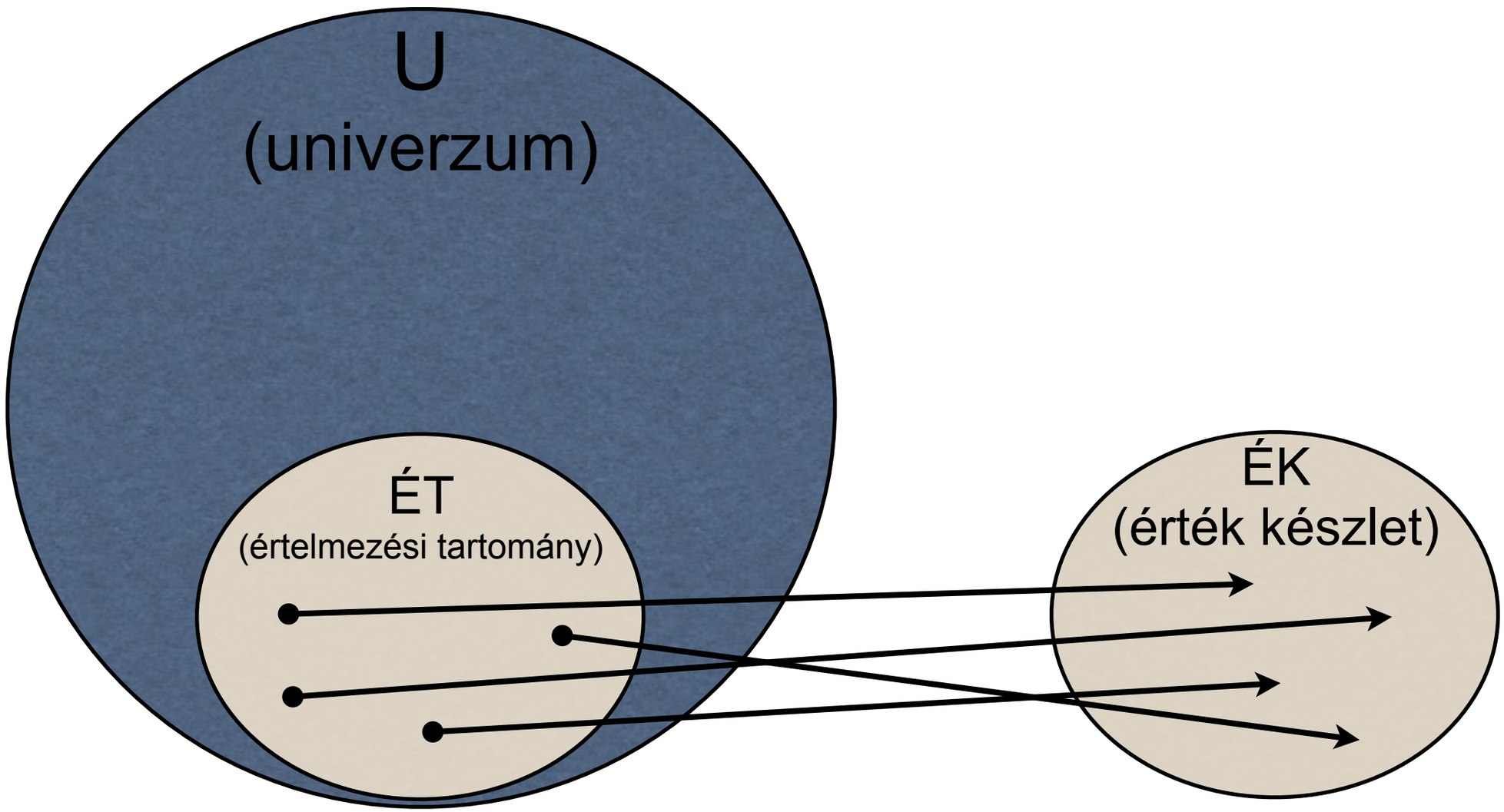
Mindhárom művelet amortizált költsége O(1) ezért tetszőleges m hosszú BVEREM műveletsor költsége O(m)

Hasítótáblák

Halmaz adattípus



Függvény adattípus



Milyen az univerzum?

Rendezések

felelevenítés

Kupacrendezés:
n elemet mozgat a kupacba
majd n elemet vesz ki a kupacból

1 elem kupacba tétele O(logn) n elem kupacba tétele O(nlogn)

1 elem kupacból kivétele O(logn) n elem kupacból kivétele O(nlogn)

Külső rendezés

1 elem halmazba tétele O(logn) n elem halmazba tétele O(nlogn)
például piros-fekete fa

n elem kiírása O(n)

O(n+nlogn)=O(nlogn)

Leszámláló rendezés

felelevenítés

Csak speciális esetre!

U={1,2,..,10}

T

1	2	3	4	5	6	7	8	9	10
I I	I I	I		I		I	I I I	I	

8 2 8 1 7 3 2 9 8 5 1

1 1 2 2 3 5 7 8 8 8 9

O(n)

Leszámláló rendezés

felelevenítés

n elemet mozgat (striguláz) a tömbbe
majd n elemet vesz ki a tömbből

1 elem tömbbe tétele O(1)

1 elem tömbből kivétele O(1)

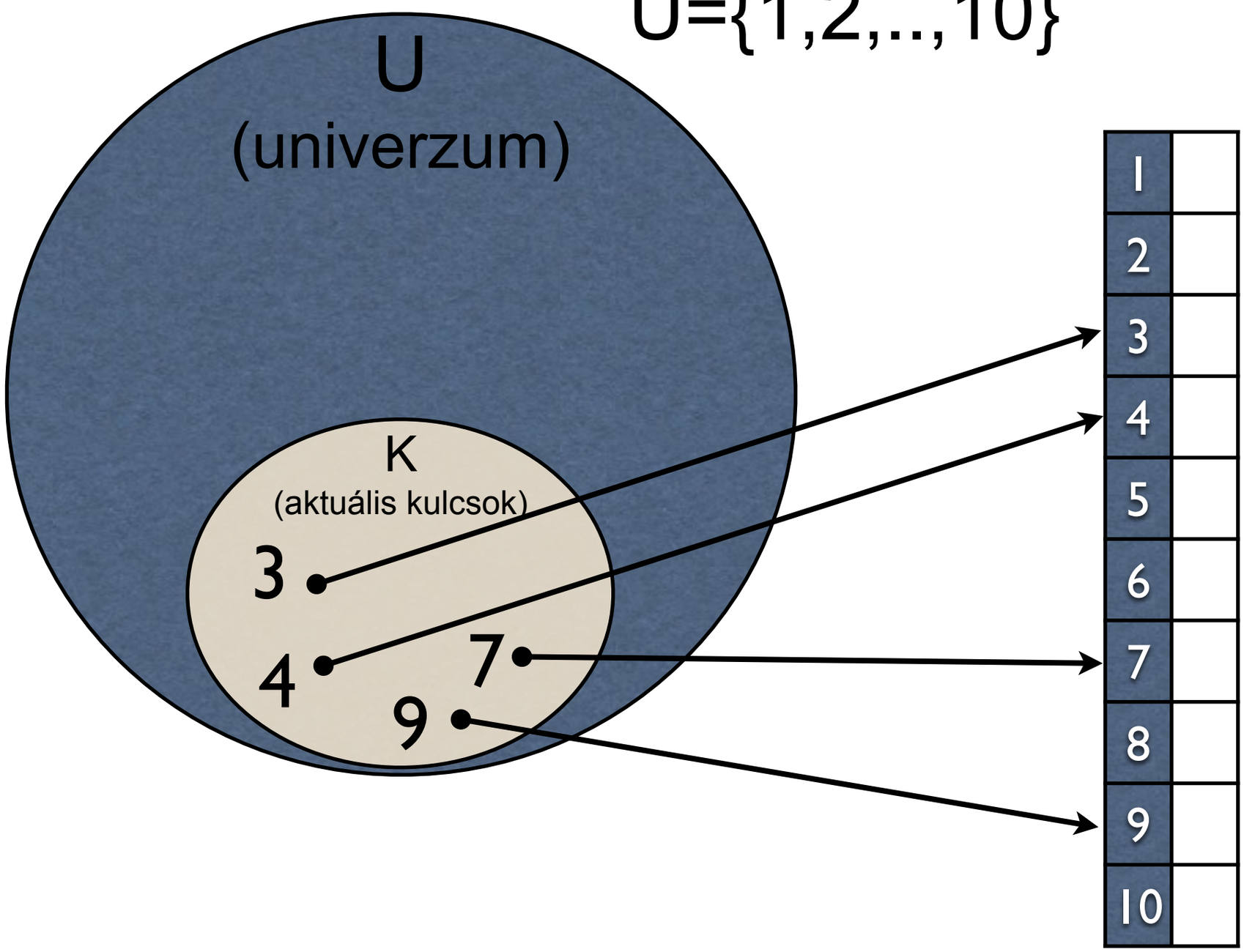
n elem tömbbe tétele O(n)

n elem tömből kivétele O(n)

Közvetlen címzésű táblázatok

Erősen korlátozott felhasználhatóság!

U={1,2,..,10}



Elem betesz: O(1)

Elem keres: O(1)

Elem kivesz: O(1)

Megvalósítás

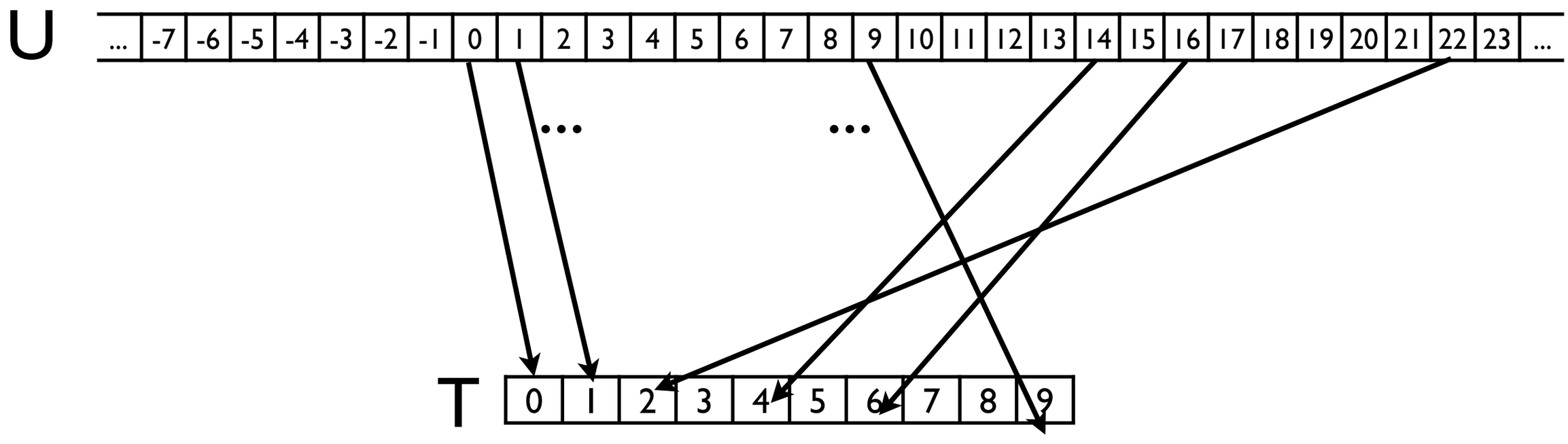
U={0,1,2,..,99}

```
int i,T[99] ;
for (i=0;i<100;i++) T[i]=0 ;
int betesz(int a) {
   if (T[a]==0) {
      T[a]=1 ;
      return 1 ;
   } else return 0 ;
}
boolean keres(int a) {
   if (T[a]==1) return true ;
   else return false ;
}
int torol(int a) {
   if (T[a]==1) {
      T[a]=0 ;
      return 1 ;
   } else return 0 ;
}
```

Korlátozott felhasználhatóság megszűnik?

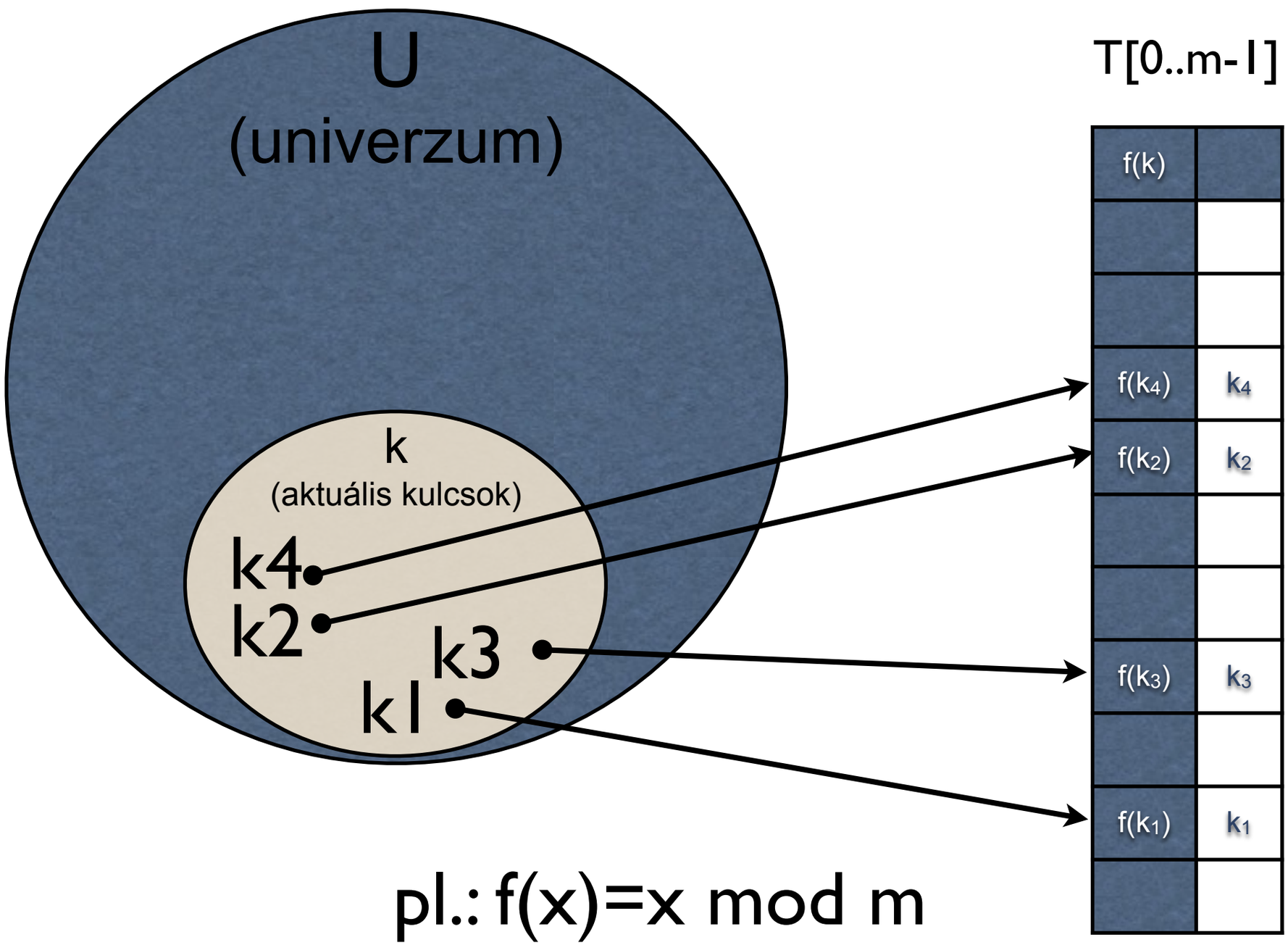
U={0,1,2,..} U={..,-1,0,1,..}

nem használhatunk végtelen tömböt!



Hasító táblázat

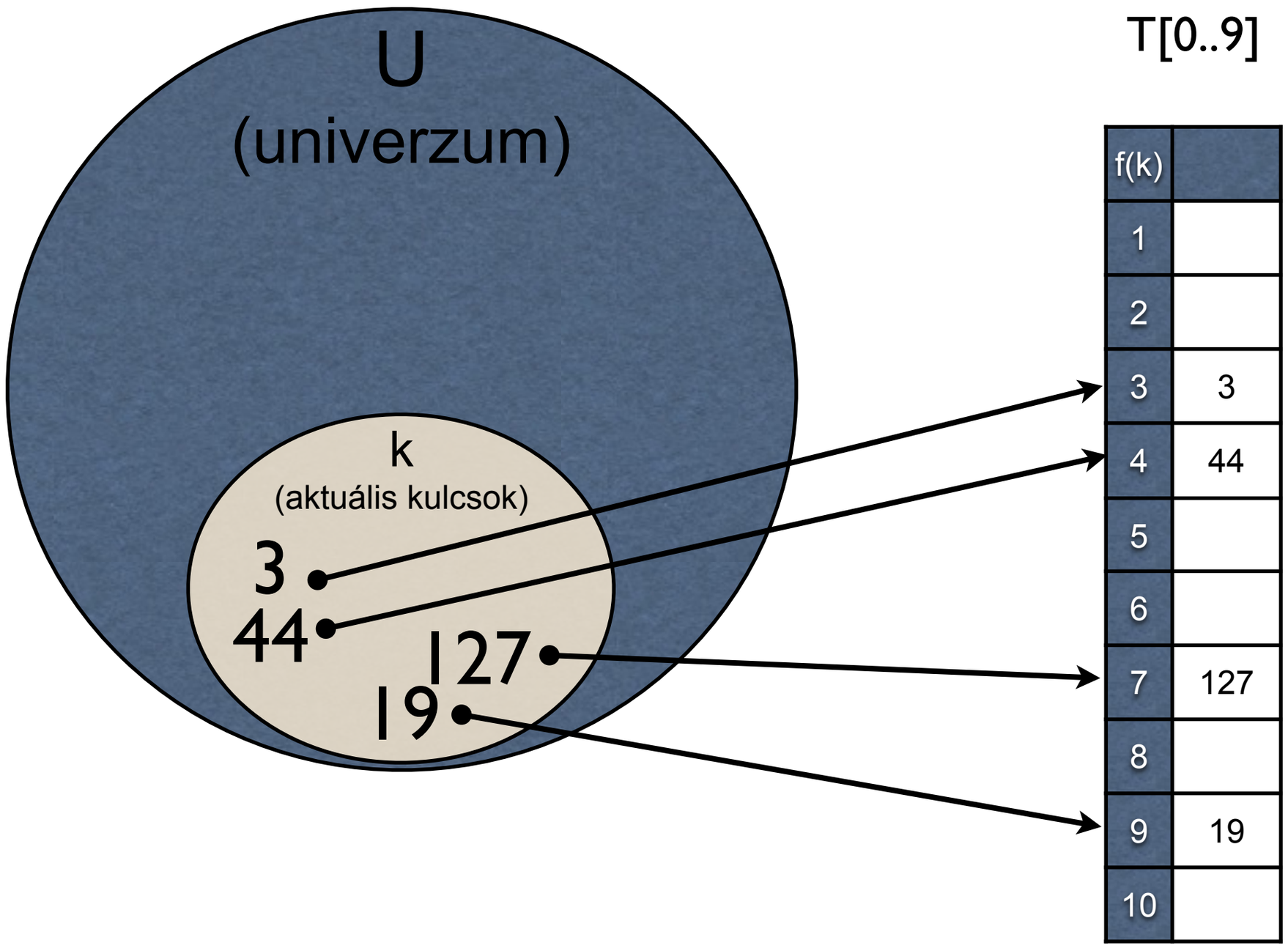
f(x) hasító függvény



pl.: f(x)=x mod m

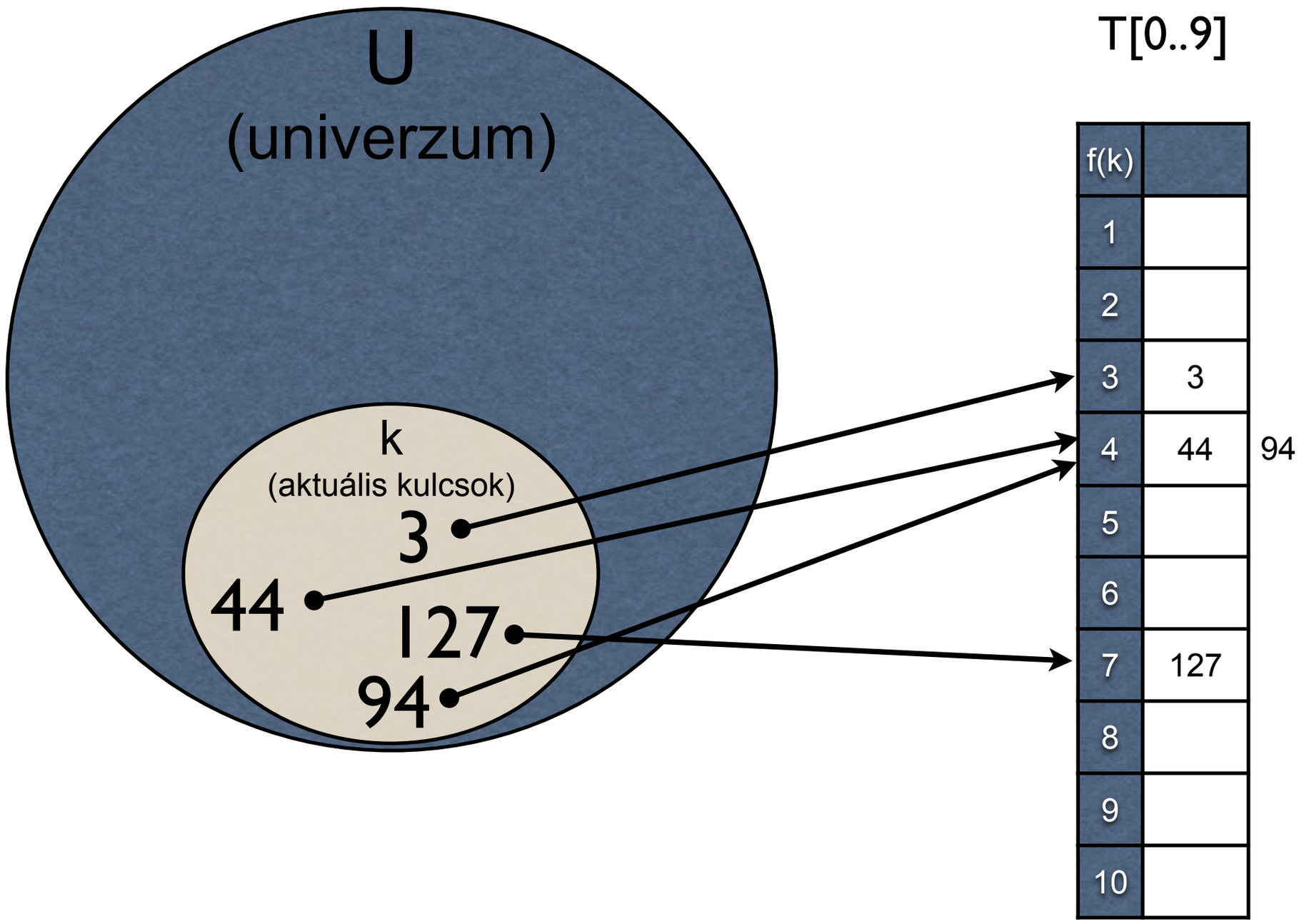
Hasító táblázat példa

pl.: f(x)=x mod 10

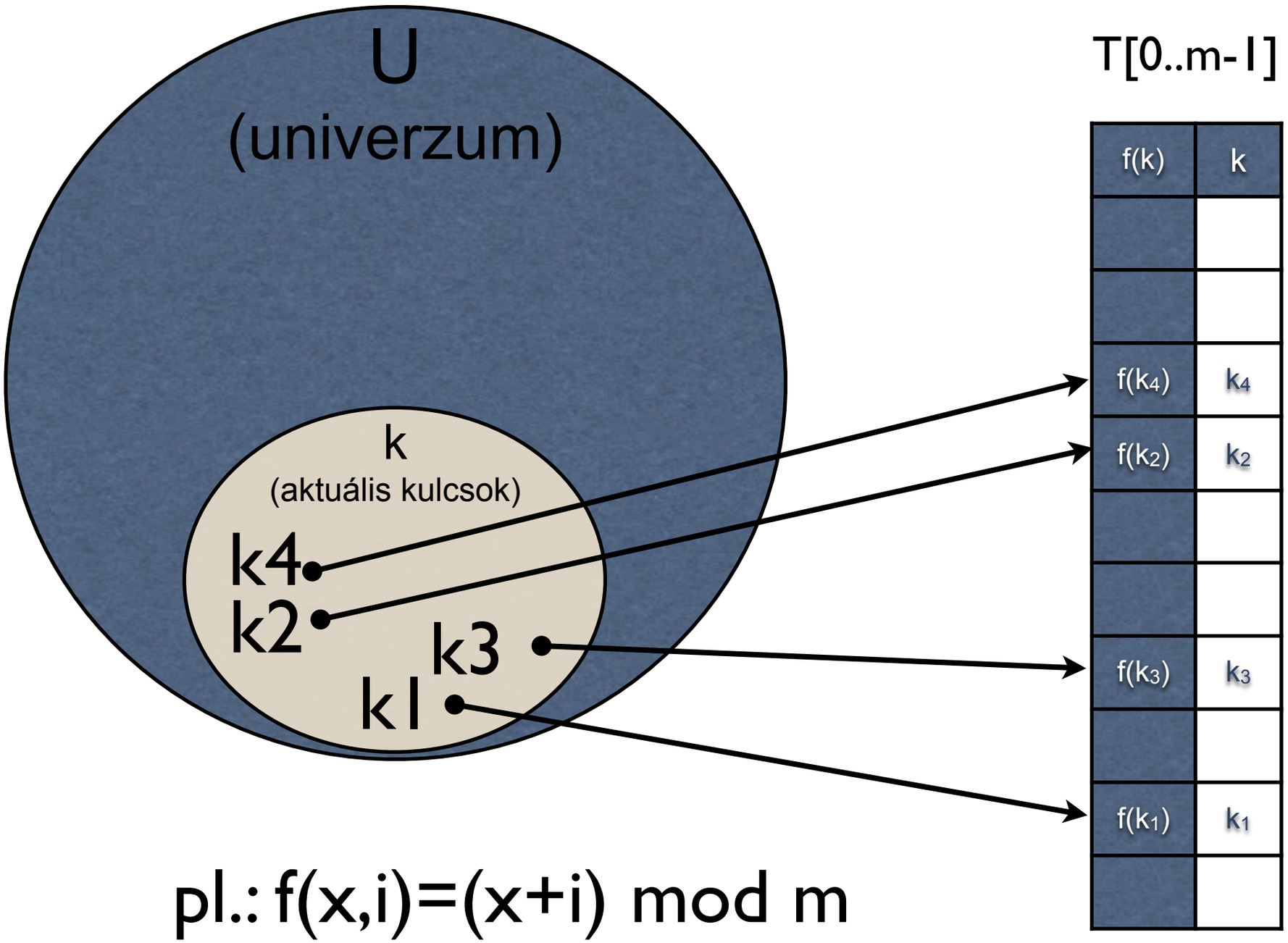


Hasító táblázat példa

pl.: f(x)=x mod 10



Ütközésfeloldás nyílt címzéssel



pl.: $f(x,i)=(x+i) \bmod m$

$i=0,1,2,\ldots,m-1$

Ütközésfeloldás nyílt címzéssel

pl.: $f(x,i)=(x+i) \bmod 10$

$i=0,1,2,\ldots,m-1$

T[0..9]

f(k)	k
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

37 93

87

Adott kulcsú elem törlése

pl.: f(x,i)=(x+i) mod 10

i=0,1,2,...,m-1

T[0..9]

f(k)	k
0	
1	
2	
3	93
4	23
5	53
6	
7	87
8	37
9	

18 87 37 17

Adott kulcsú elem megkeresése

T[0..9]

f(k)	k
0	-
1	-
2	-
3	93
4	T
5	53
6	T
7	87
8	37
9	-

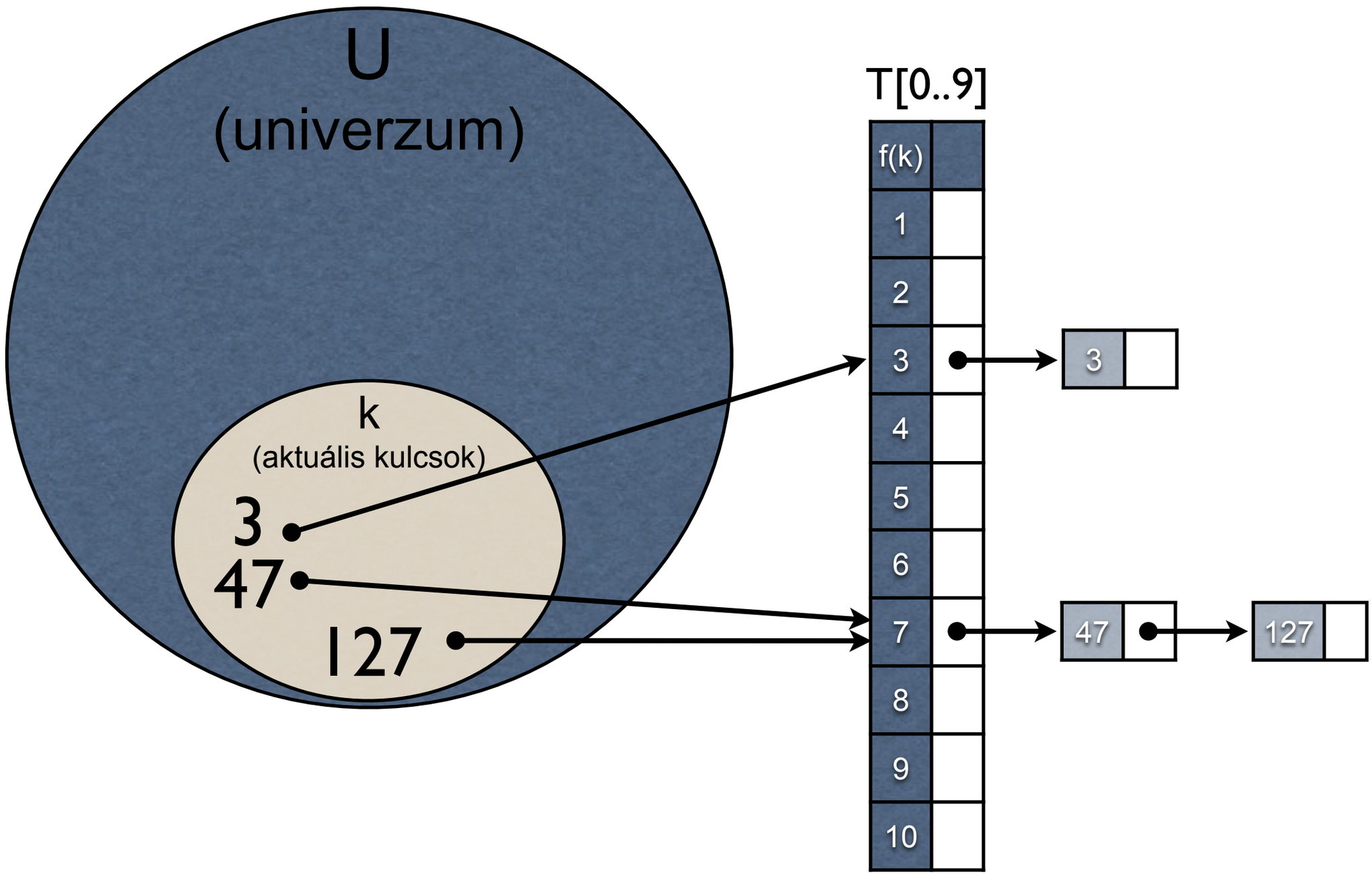
24

```
keres(k) {
   i=0 ;
   do {
      j=h(k,i++) ;
      if (T[j]==k) return j ;
   } while (T[j]!=nil && i!=m) ;
   return nil ;
}
```

Törlést kiegészítő művelet?

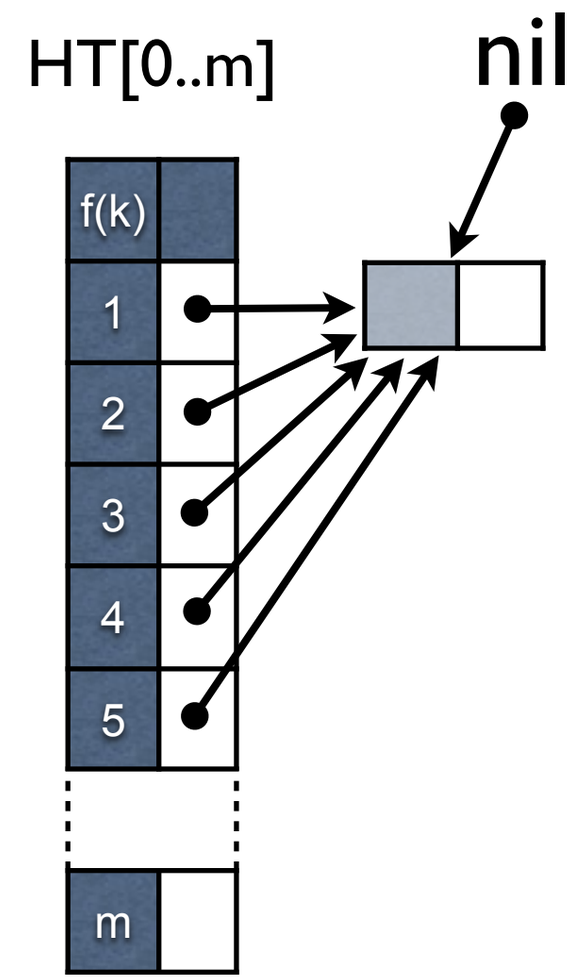
Hasítótáblázatok

pl.: f(x)=x mod 10



Adatszerkezet

```
public class HashT {
   int elemszam=0 ;
   class hastomb {
      elemtar elso ;
   }
   class elemtar {
      int kulcs ;
      elemtar kovetkezo ;
   }
   elemtar nil ;
   hastomb[] HT=new hastomb[m] ;
   HashT() {
      nil=new elemtar() ;
      int i ;
      for (i=1;i<n;i++) HT[i].elso=nil ;
   }
   ...
}
```



Adott kulcsú elem megkeresése

```
public class HashT {
   ...
   int hasit(int k) {
      return k % m ;
   }

   elemtar keres(int k) {
      elemtar q=HT[hasit(k)].elso ;
      while (q!=nil && q.kulcs!=k) q=q.kovetkezo ;
      return q ;
   }
   ...
}
```

Mennyi ideig tart egy adott kulcsú elem megkeresése?

Legyen T egy m rést tartalmazó hasító táblázat amelyikben n elem van

T[0..m-1]

f(k)	
1	
2	
3=f(k_1)	k_1
4=f(k_2)	k_2
5	
6	
7=f(k_3)	k_3
m-2=f(k_n)	k_n
m-1	

α kitöltési tényező $\alpha = \frac{n}{m}$

az egy láncba fűzött elemek átlagos száma

Legrosszabb eset elemzés:

mind az n elem egy láncba képződik le

keresés végrehajtási ideje: O(n)

Adott kulcsú elem beszúrása

```
int beszur(int k) {
   elemtar elozo ;
   elemtar q=HT[hasit(k)].elso ;
   while (q!=nil && q.kulcs!=k) {
      elozo=q ;
      q=q.kovetkezo ;
   }
   if (q!=nil) return -1;
   else {
      elemtar p=new elemtar() ;
      elozo.kovetkezo=p ;
      p.kulcs=k ;
      p.kovetkezo=nil ;
   }
}
```

Adott kulcsú elem törlése

```
int torol(int k) {
   elemtar elozo ;
   elemtar q=HT[hasit(k)].elso ;
   while (q!=nil && q.kulcs!=k) {
      elozo=q ;
      q=q.kovetkezo ;
   }
   if (q==nil) return -1;
   else {
      elozo.kovetkezo=q.kovetkezo ;
      return 0 ;
   }
}
```

Feltételezhetjük, hogy minden elem egyforma valószínűséggel képződik le bármely résre, függetlenül attól, hogy a többiek hová kerültek

Ezt ***egyszerű egyenletes hasítási feltétel***nek nevezzük

T[j] lista hosszát jelöljük n_j-vel. (j=0,1,..., m-1)

ekkor $n = n_0 + n_1 + ... + n_{m-1}$ és n_j várható értéke:

$$E(n_j) = \frac{n}{m} = \alpha$$

Hasítófüggvény megválasztása

$f : Z_k^3 \rightarrow Z_m$ pl. rgb(240,0,127)

rgb(r,g,b) ⇒ (65536r + 256g + b) mod m

rgb(r,g,b) ⇒ (66563r + 257g + b) mod m

$f : [R] \rightarrow Z_m$ pl. float

$f : [R]^2 \rightarrow Z_m$ pl. float interval

<u>Tétel:</u> **Láncolásos ütközésfeloldás**nál, ha a hasítás **egyszerű egyenletes**, akkor a *sikertelen keresés* **átlagos** ideje: $O(1+\alpha)$.

<u>Biz.:</u> A sikertelen keresés átlagos ideje megegyezik annak átlagos idejével hogy a $T[f(k)]$ listát végigkeressük.
Ennek a listának az átlagos hossza α. $f(k)$ kiszámítási ideje 1.
Így a sikertelen keresés átlagos ideje: $O(1+\alpha)$

$\square$

<u>Tétel:</u> **Láncolásos ütközésfeloldás**nál, ha a hasítás **egyszerű egyenletes**, akkor a *sikeres keresés* **átlagos** ideje: $O(1+\alpha)$.

<u>Megjegyzés:</u> A sikeres keresés abban különbözik a sikertelentől, hogy nem feltétlenül kell a $T[f(k)]$ listát végigkeressük, így az átlagos idő nem lehet több mint a *sikertelen keresés* **átlagos** ideje.

$\square$

Következtetések

Ha α konstans, a keresés, beszúrás és törlés ideje O(1)!

Hogy lehet biztosítani, hogy az α konstans legyen?

Ha m arányos n-el azaz tudjuk előre az adatszerkezetbe kerülő adatok maximális számát (vagy legalább annak nagyságrendjét)

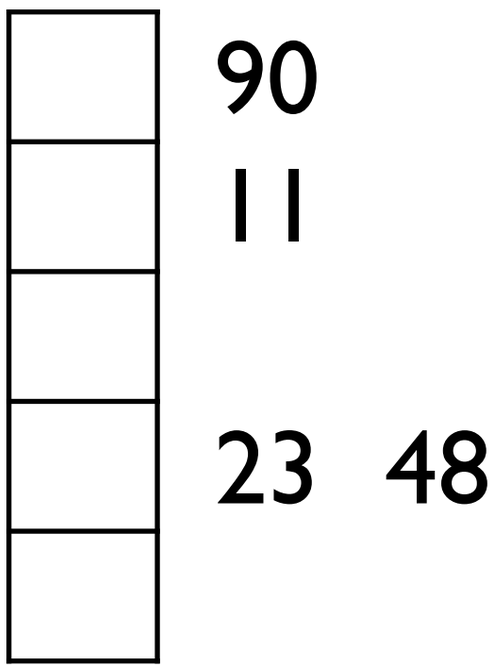
Korlátozott felhasználhatóság nem szűnt meg!

Futási idő

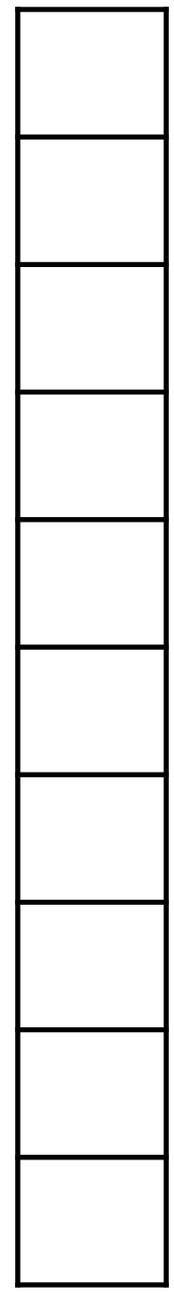
n	O(logn)	O(1)
1	1	1
1 000	11	1
1 000 000	21	1
1 000 000 000	31	1
1 000 000 000 000	41	1

37

x mod 5



x mod 10



1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 10 ...n

5 5 10 10

O(1)

Dinamikus táblák

```
Type
  Elemtip=???;
  Tabla=Record
            meret:Longint;
            eszam:Longint;
            Tar:^Array[1..MaxLongint] of Elemtip;
          End;
```

Tábla kiterjesztése

```
Procedure Beszur(Var T:Tabla; x:Elemtip);
Var UT:Tabla;
  i:Longint;
Begin
  If T.meret=0 Then Begin
    GetMem(T.Tar,Sizeof(Elemtip));
    T.meret:=1;
    T.eszam:=0;
  End;
  If T.szam=T.meret Then Begin
    UT.meret:=2*T.meret;
    Getmem(UT.Tar, UT.meret*Sizeof(Elemtip));
    UT.eszam:=T.eszam;
    For i:=1 To T.eszam Do
      UT.Tar^[i]:=T.Tar^[i];
    MreeMem(T.Tar,T.meret*Sizeof(Elemtip))
    T:=UT;
  End;
  Inc(T.eszam);
  T.Tar^[T.eszam]:=x;
End;
```

$$c_i = \begin{cases} i & \text{if } i-1 \text{ 2-hatvány ,} \\ 1 & \text{különben .} \end{cases}$$

n Beszur teljes költsége

$$\Phi(T) = 2T.eszam - T.meret$$

Minden kiterjesztés után közvetlenül T.eszam = T.meret/2+1, vagyis $\Phi(T) = 2$, közvetlenül előtte pedig T.eszam = T.meret, így $\Phi(T)$ = T.eszam. $\Phi(T)$ mindig nemnegatív.

Legyen az i-edik művelet után közvetlenül $szam_i$ a tárolt elemek száma, $meret_i$ a tábla mérete és Φ_i a potenciálfüggvény értéke. Kezdetben $szam_0 = meret_0 = \Phi_0 = 0$.

Ha az i-edik Beszur művelet nem váltja ki a tábla kiterjesztését, akkor $meret_i = meret_{i-1}$, így az amortizációs költsége:

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot eszam_i - meret_i) - (2 \cdot eszam_{i-1} - meret_{i-1}) \\
&= 1 + (2 \cdot eszam_i - meret_i) - (2(eszam_i - 1) - meret_i) \\
&= 3\,.
\end{aligned}
$$

Ha az i-edik Beszur művelet kiváltja a tábla kiterjesztését, akkor

$$
\begin{aligned}
meret_i &= 2 \cdot meret_{i-1} \\
meret_{i-1} &= eszam_{i-1} = eszam_i - 1 \\
meret_i &= 2 \cdot (eszam_i - 1)
\end{aligned}
$$

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= eszam_i + (2 \cdot eszam_i - meret_i) - (2 \cdot eszam_{i-1} - meret_{i-1}) \\
&= eszam_i + (2 \cdot eszam_i - 2 \cdot (eszam_i - 1)) - (2(eszam_i - 1) - (eszam_i - 1)) \\
&= eszam_i + 2 - (eszam_i - 1) \\
&= 3
\end{aligned}
$$

Sor, Verem

Láncolt listák

Halmaz, Map

Hasítótáblák, Keresőfák

Prioritási sor:

Egyesíthető prioritási sor

Módosítható prioritási sor

Módosítható, egyesíthető prioritási sor

Egyesíthető prioritási sor

Egyesíthető prioritási sor műveletek:

Eprisor s1,s2 ;

void s.sorba(T x) ; O(log(n)) ?

T s.sorból() ; O(log(n)) ?

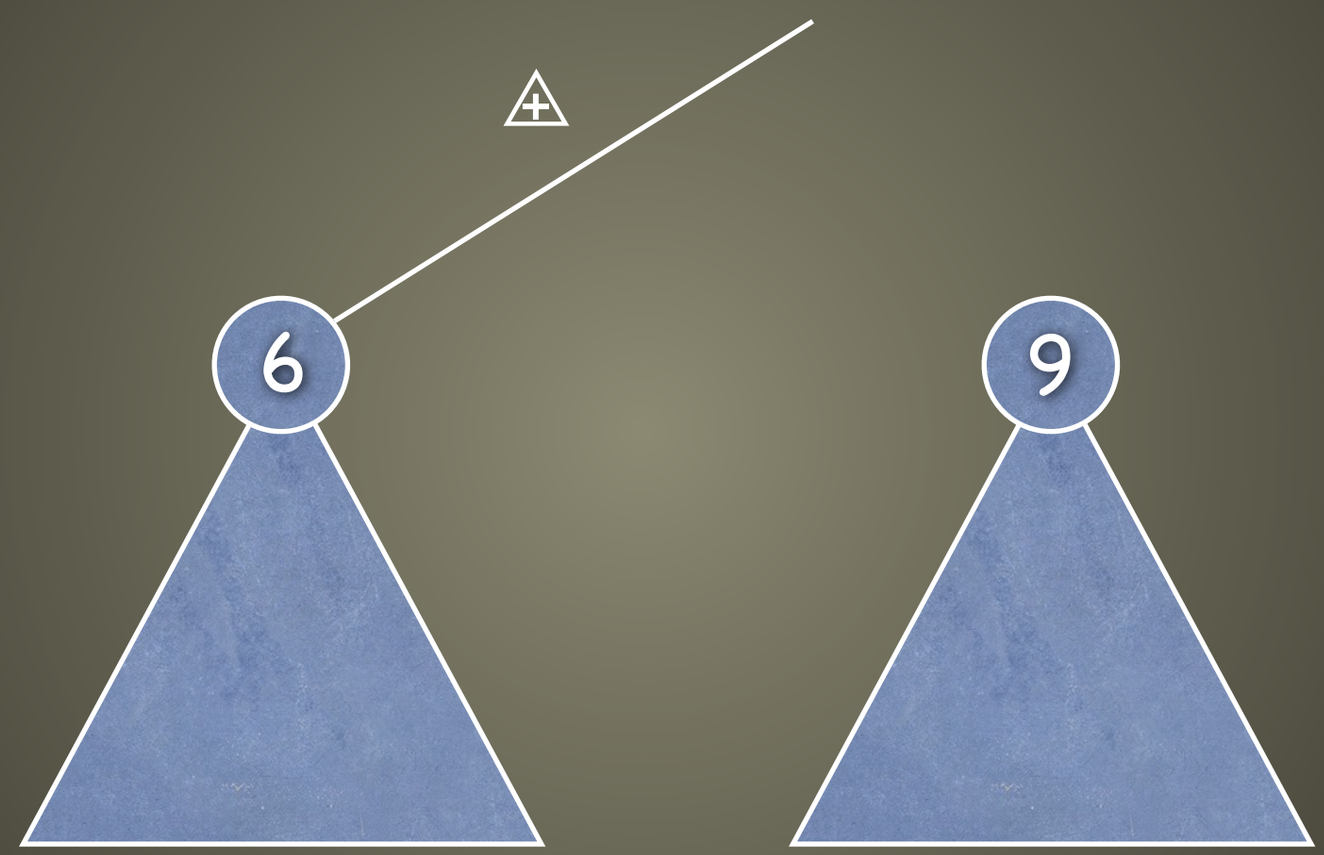
Eprisor s1.egyesit(Eprisor s2) ; ?

Binomiális fa

maximum kupac

Prisor S_1, S_2 ;

$S = S_1 \triangle S_2$



def.: $\triangle$ művelet a nagyobb kulcsot tartalmazó gyökérelemhez kapcsoljuk a kisebb kulcsot tartalmazó gyökérelemet

def.: legyen $\mathcal{B}$ (r) r-ed rangú binomiális fa

legyen $\mathcal{B}$ (0) egy egyetlen pontból

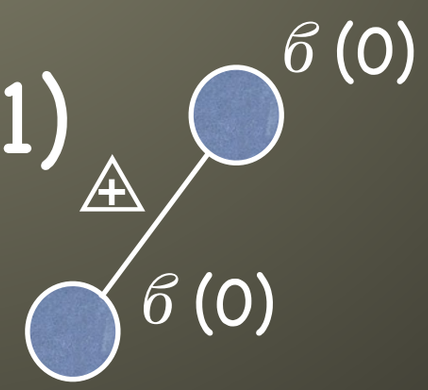
álló binomiáris fa

és $\mathcal{B}$ (r)= $\mathcal{B}$ (r-1) $\triangle\!\!\!+$ $\mathcal{B}$ (r-1)

példa:

$\mathcal{B}$ (0)

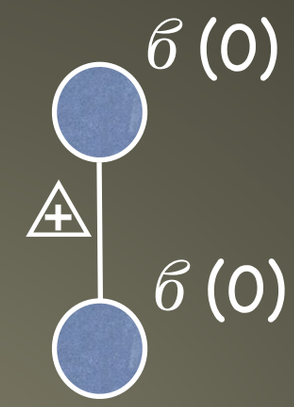
$\mathcal{B}$ (1)



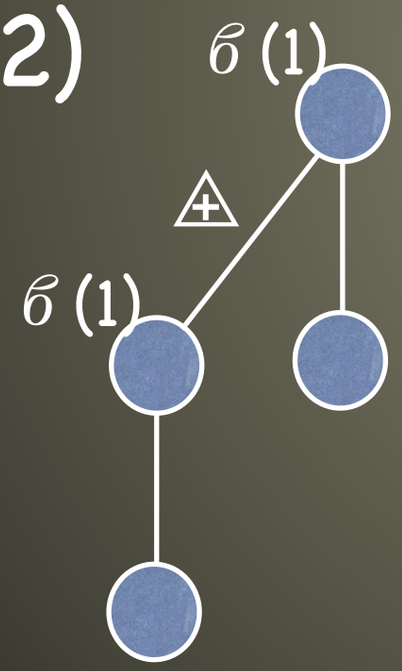
példa:

$\mathcal{B}(0)$

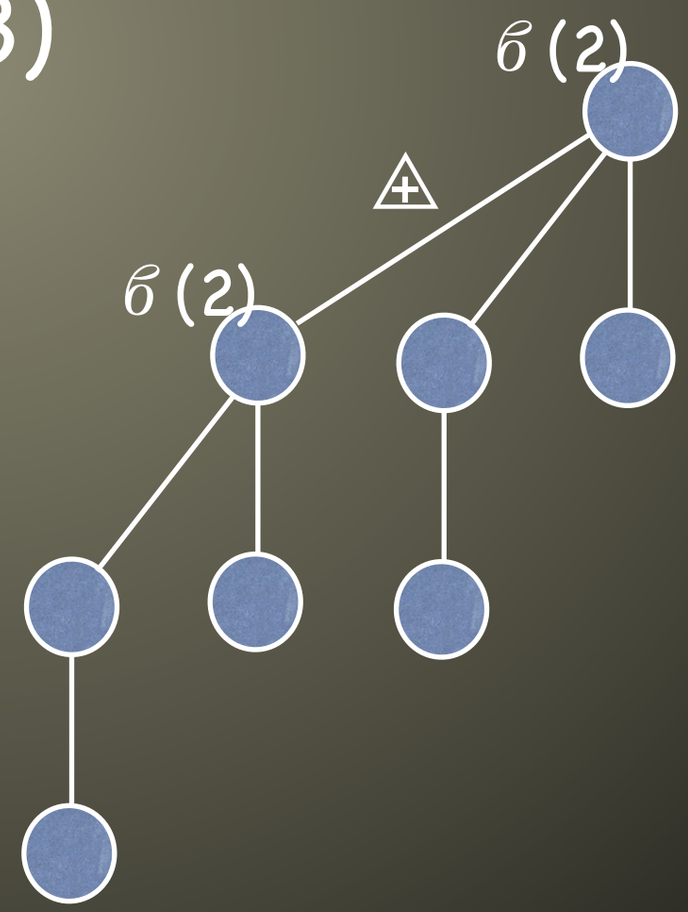
$\mathcal{B}(1)$



$\mathcal{B}(2)$



$\mathcal{B}(3)$



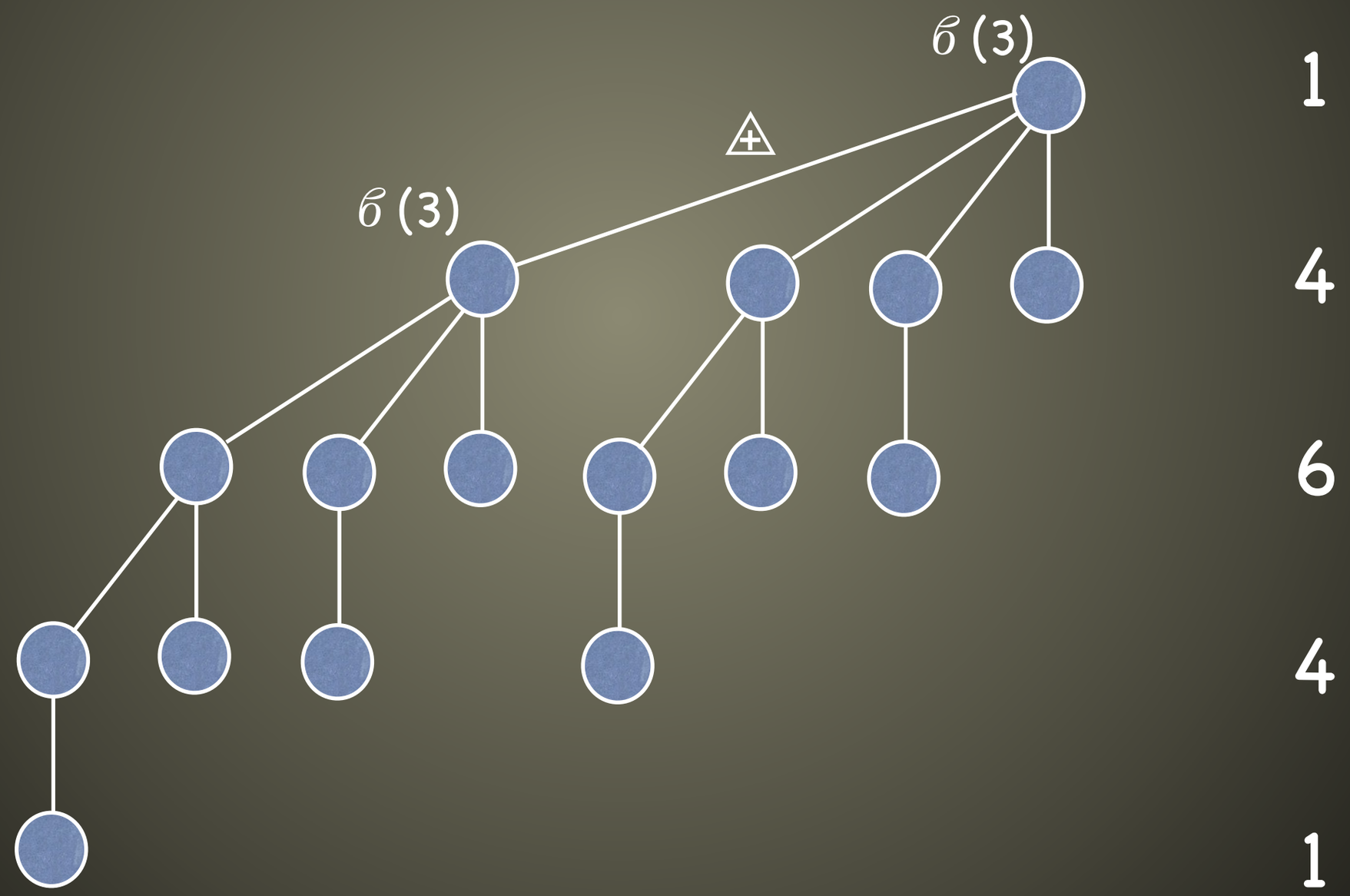
1

3

3

1

$\mathcal{B}(4)$



tétel: $h=O(\log(n))$

biz.:

$h=h(\mathcal{B}(r))=r+1$

$n=N(\mathcal{B}(r))=2^r$

$\log_2(n)=\log_2(2^r)=r\cdot\log_2(2)=r=h-1$

$h-1=\log_2(n)$

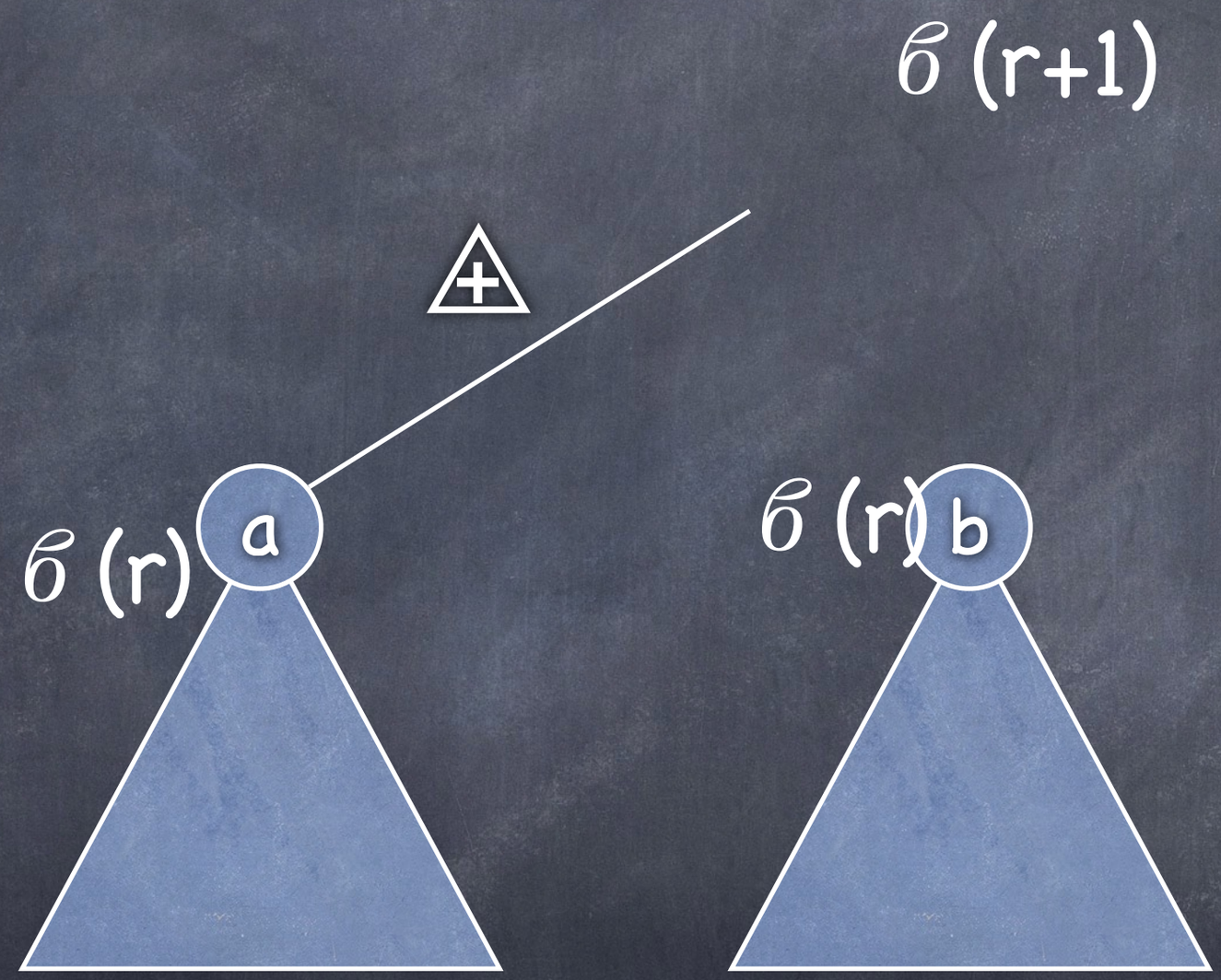
$h=\log_2(n)+1=O(\log(n))$

r	h	n
0	1	1
1	2	2
2	3	4
3	4	8

$\square$

Binomiális kupac

$b \geq a$



Mennyi adatot kell tárolunk?

r	h	n
0	1	1
1	2	2
2	3	4
3	4	8

n=3

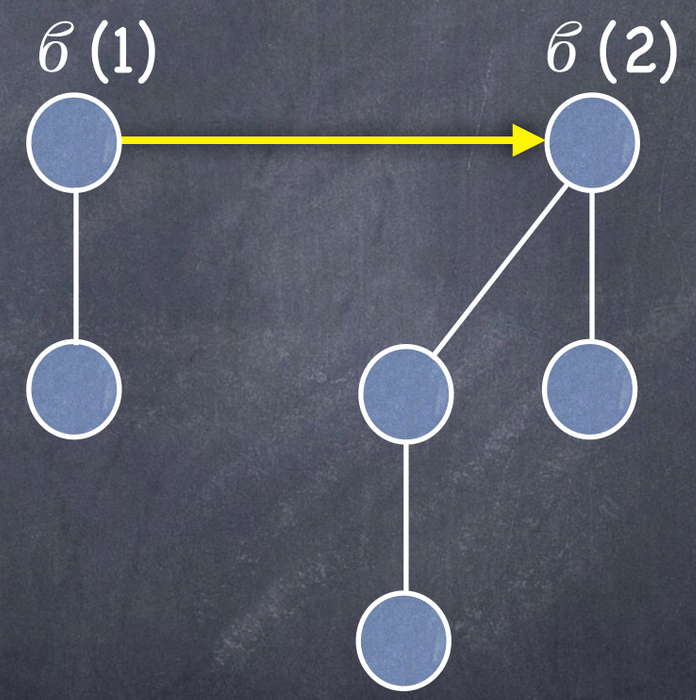
n=5 n=6 n=7

csak binomiáris fákat használjunk!

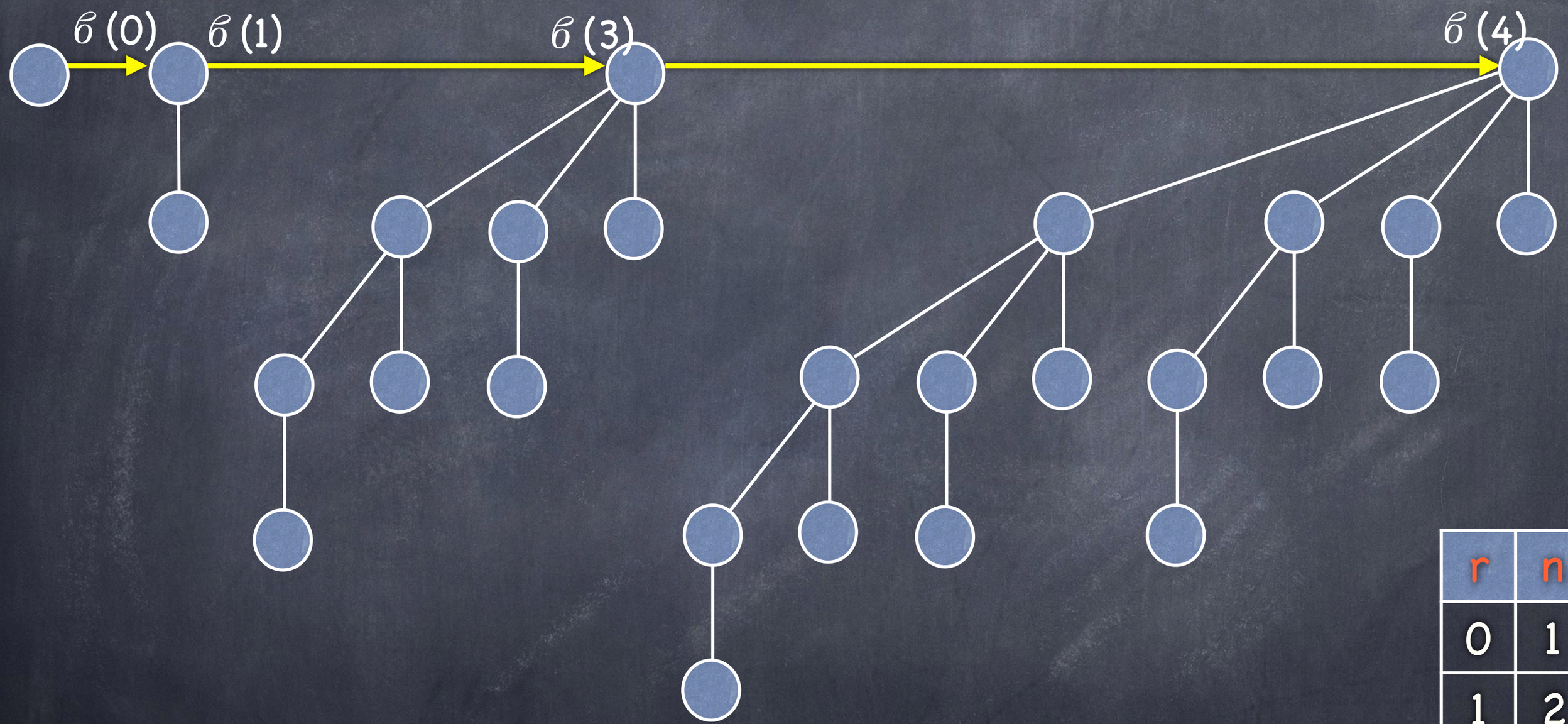
pl.:

n=6

r	h	n
0	1	1
1	2	2
2	3	4
3	4	8



pl.: n=27



r	n
0	1
1	2
2	4
3	8
4	16

n	n②	l	bh(n)
1	1	1	1
2	10	1	2
3	11	2	
4	100	1	3
5	101	2	
6	110	2	
7	111	3	
8	1000	1	4
9	1001	2	
10	1010	2	
11	1011	3	
12	1100	2	
13	1101	3	
14	1110	3	
15	1111	5	
16	10000	1	5

tétel: $l \leq \log_2(n)$

biz: l egyenlő az n bináris alakjában szereplő 1-ek számával, ami kisebb, mint a bináris alak összes számjegyeinek száma

$\square$

Egyesíthető prioritási sor műveletek:

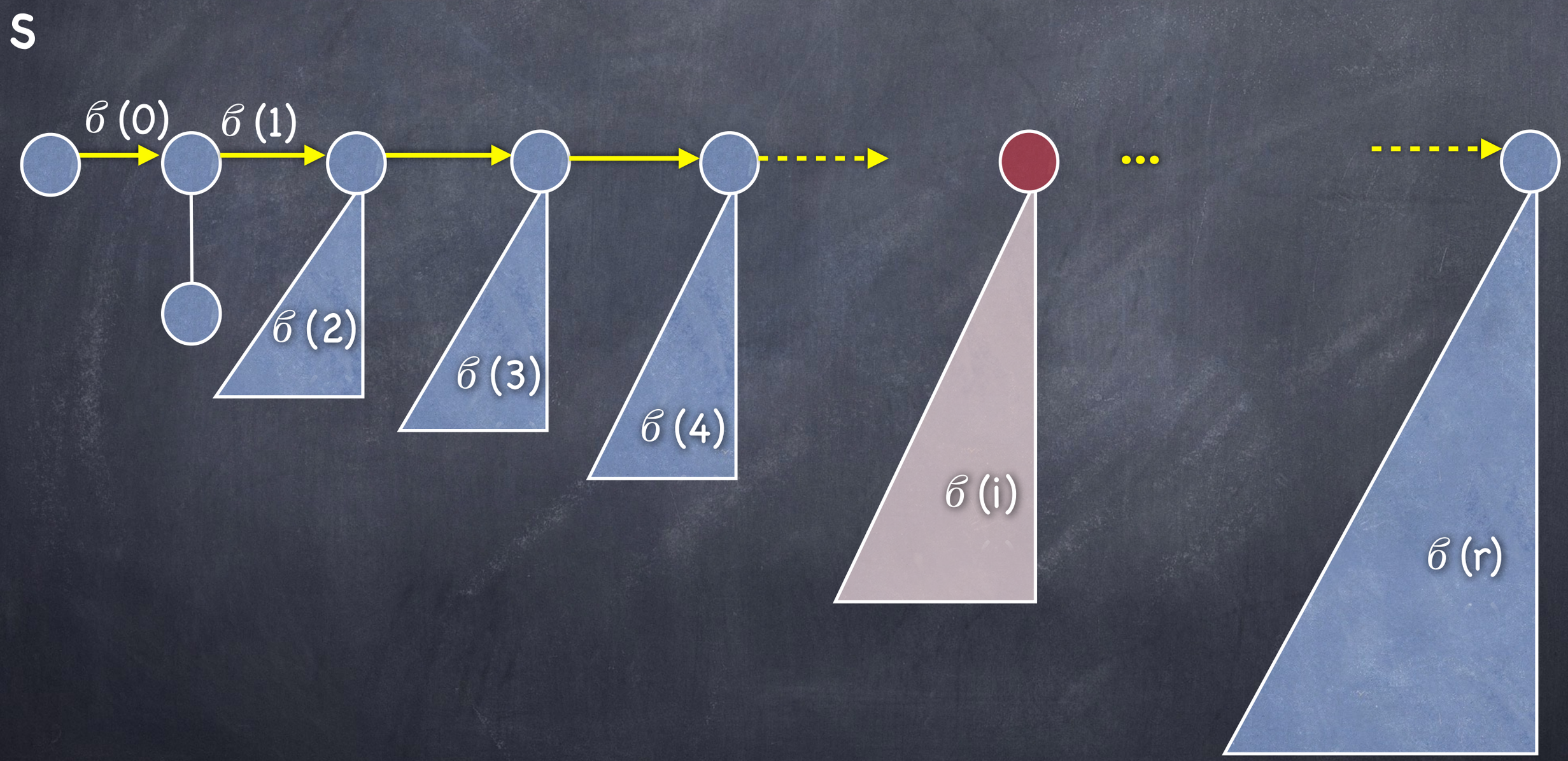
Eprisor s1,s2 ;

void s.sorba(T x) ;

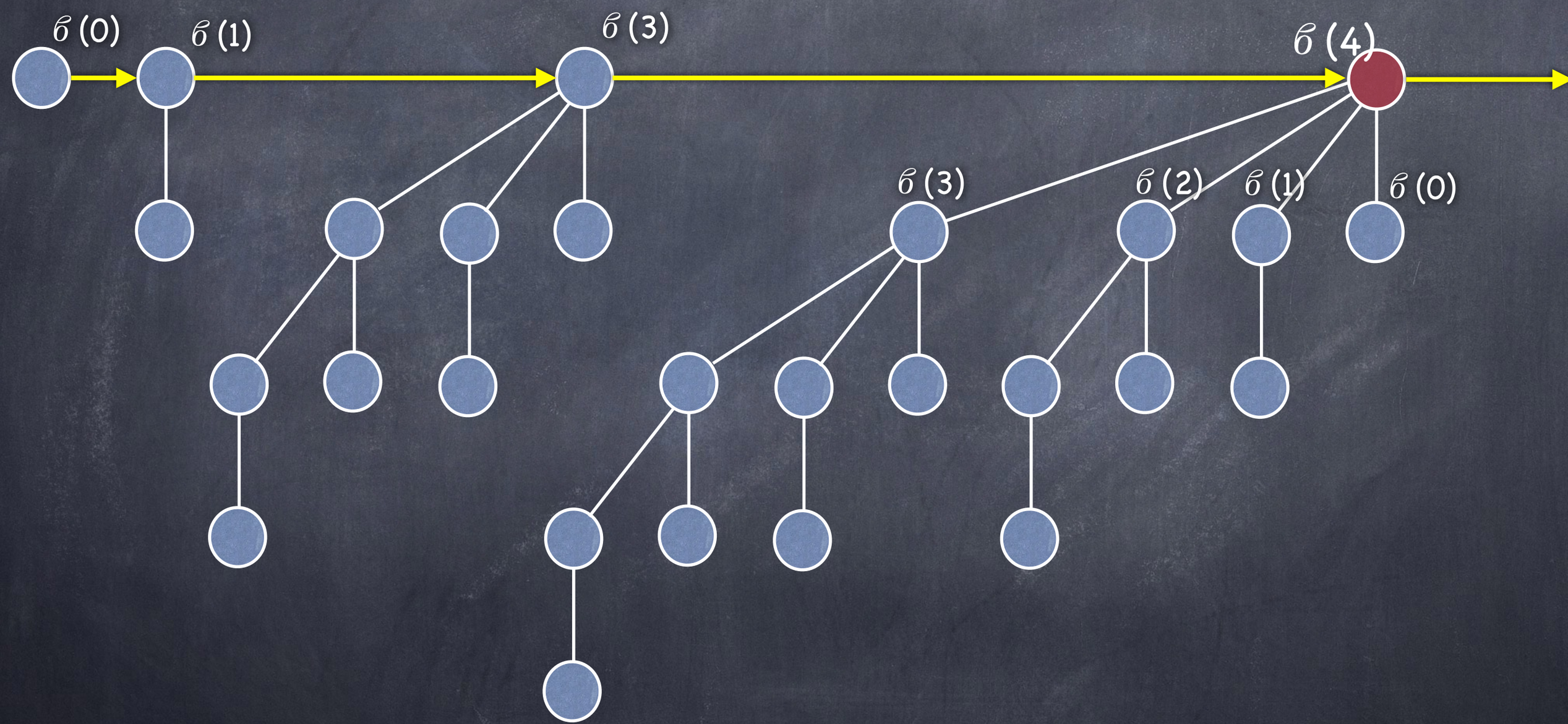
T s.sorbol() ;

Eprisor s1.egyesit(Eprisor s2) ;

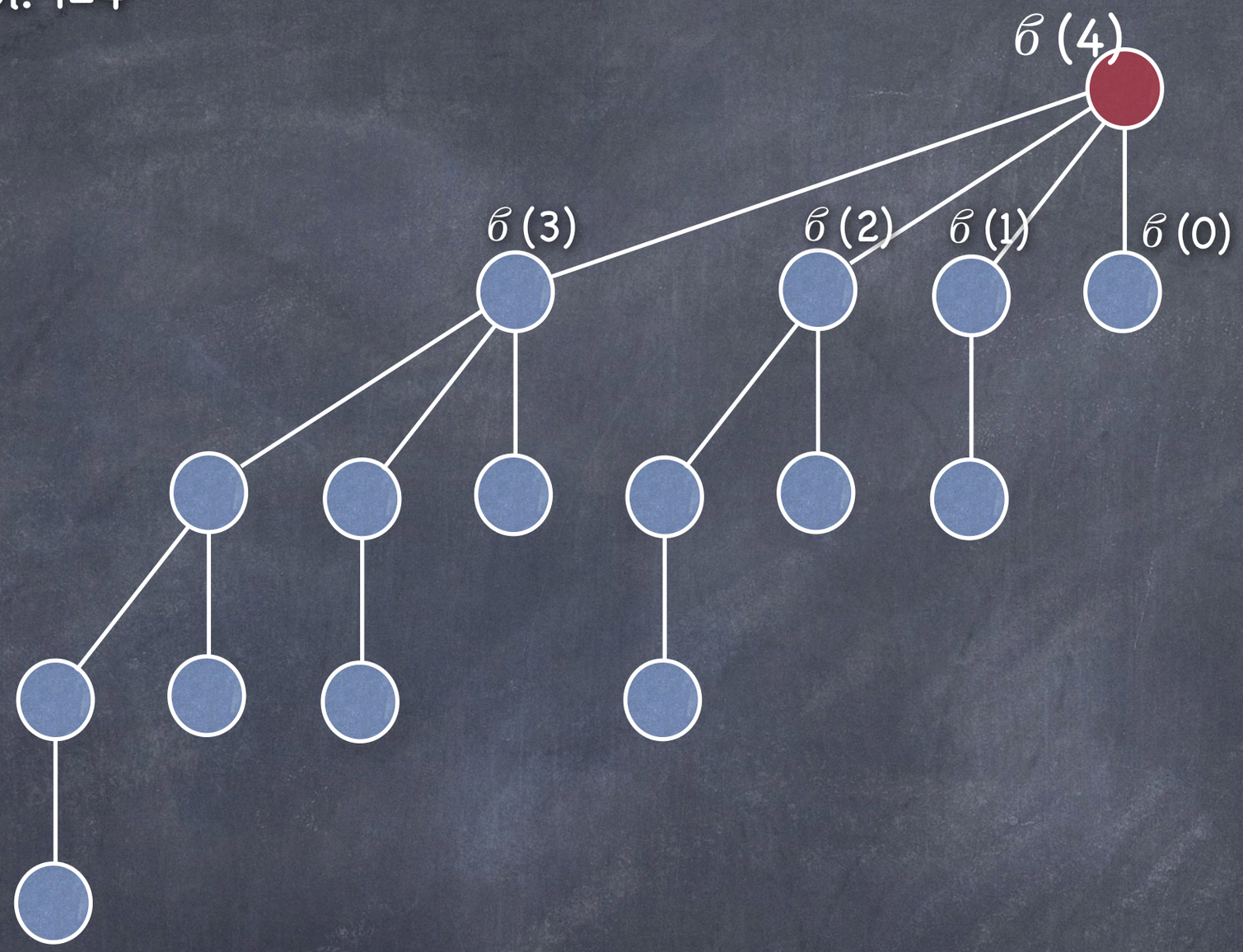
T s.sorbol()

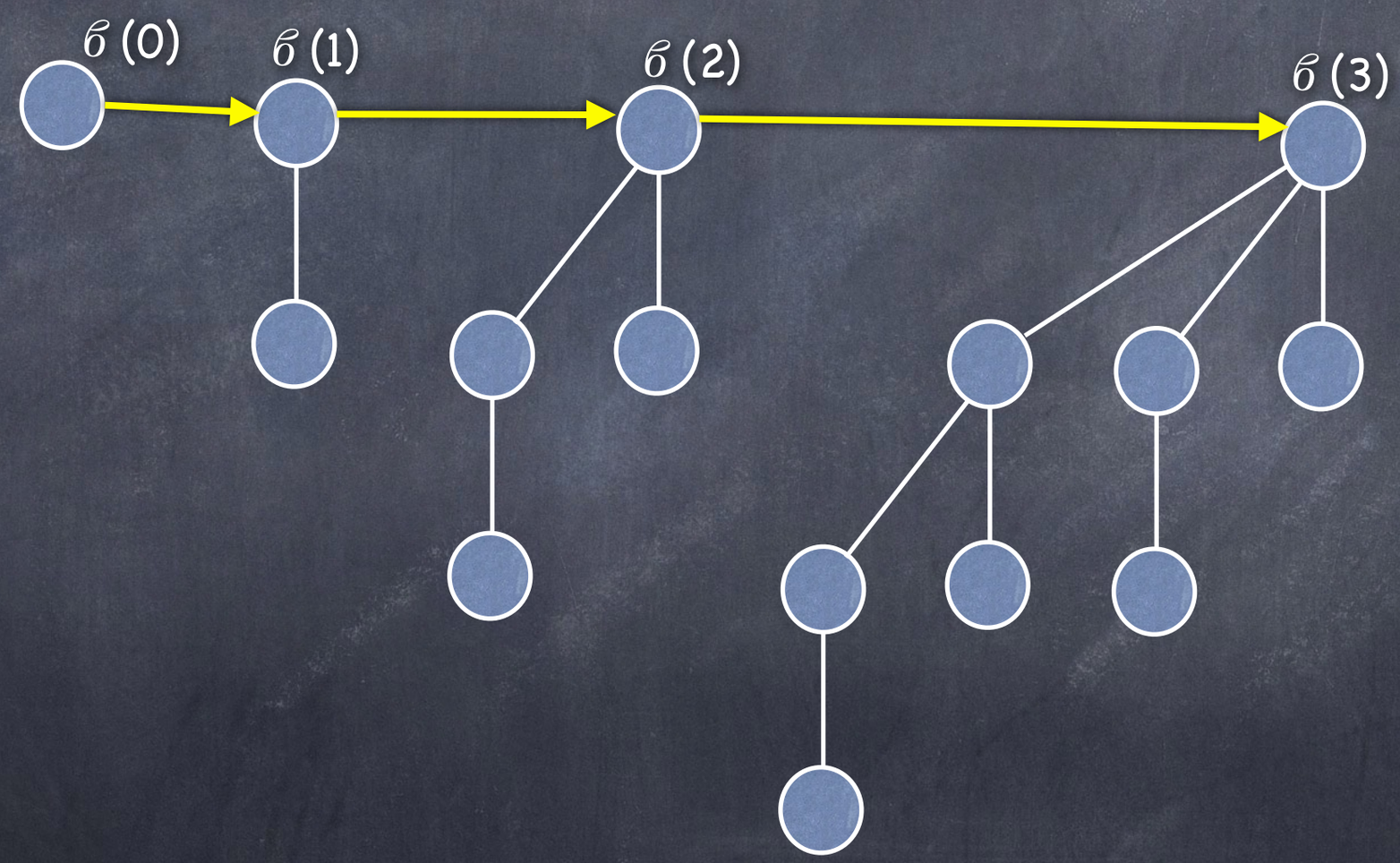

s
$\mathcal{b}$ (0)
$\mathcal{b}$ (1)
$\mathcal{b}$ (2)
$\mathcal{b}$ (3)
$\mathcal{b}$ (4)
$\mathcal{b}$ (i)
...
$\mathcal{b}$ (r)

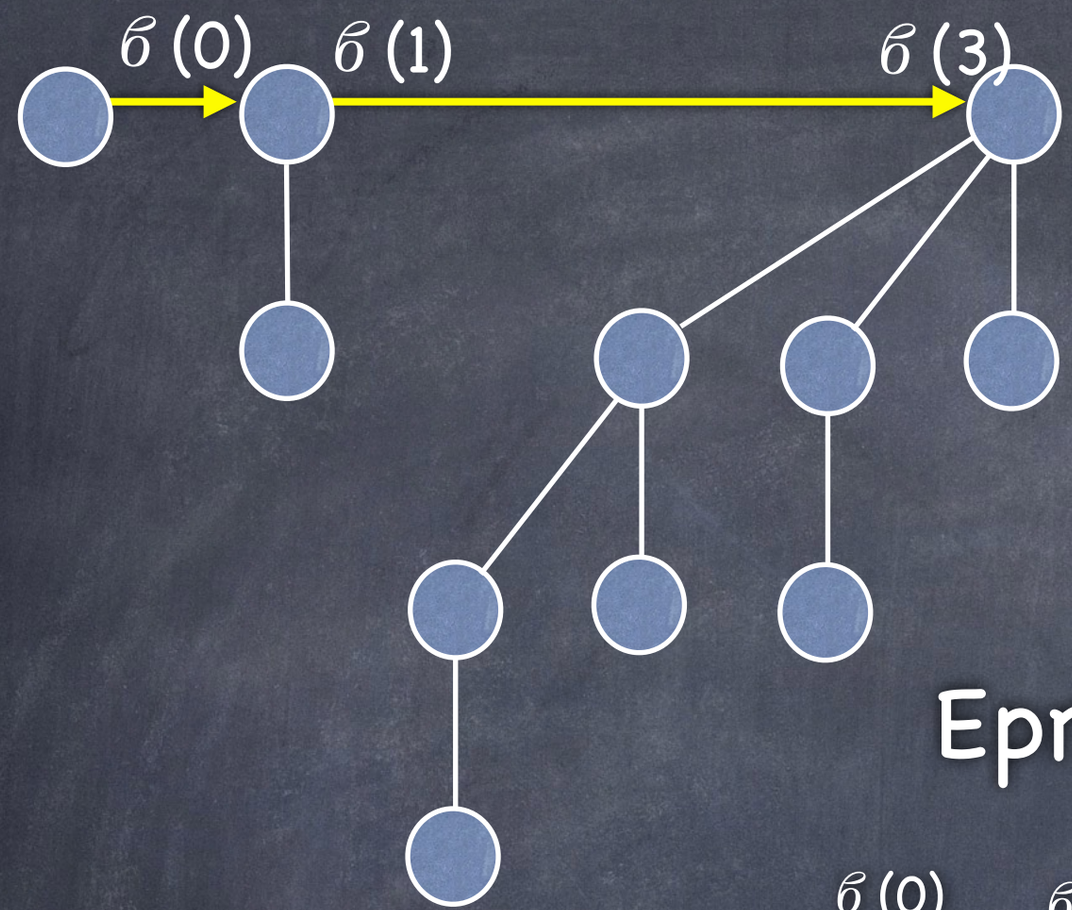
pl. i=4


$\mathcal{B}$ (0)
$\mathcal{B}$ (1)
$\mathcal{B}$ (3)
$\mathcal{B}$ (4)
$\mathcal{B}$ (3)
$\mathcal{B}$ (2)
$\mathcal{B}$ (1)
$\mathcal{B}$ (0)

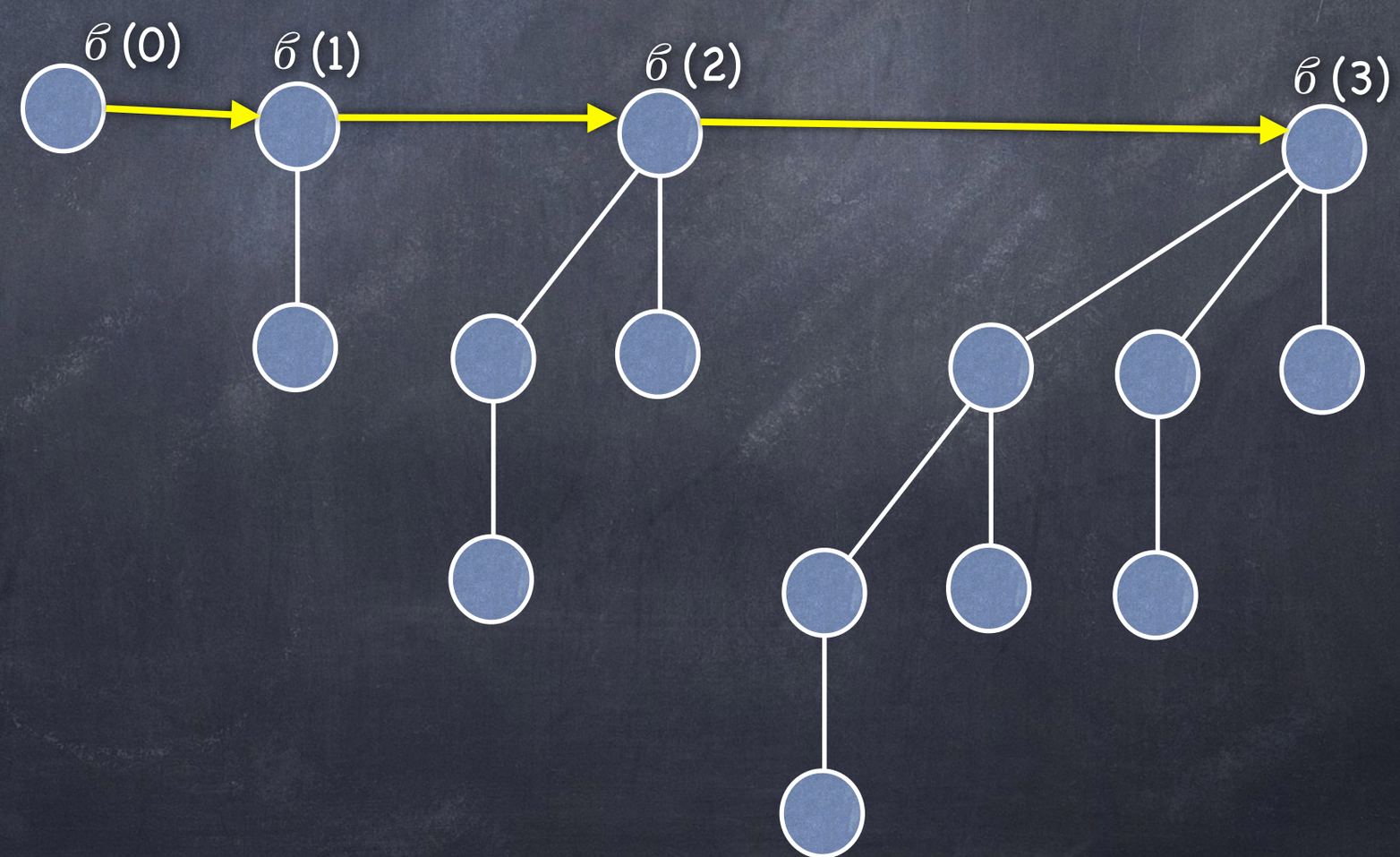
$\mathcal{B}$ (i) pl. i=4

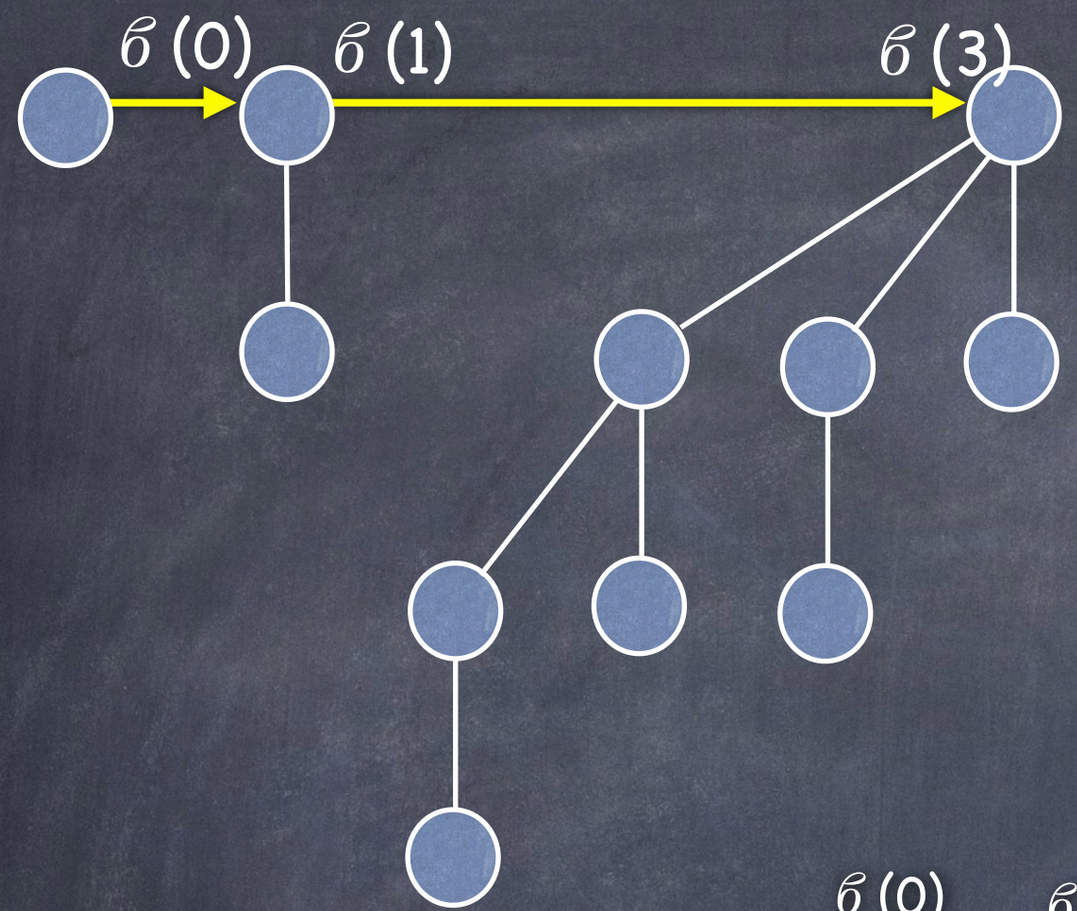


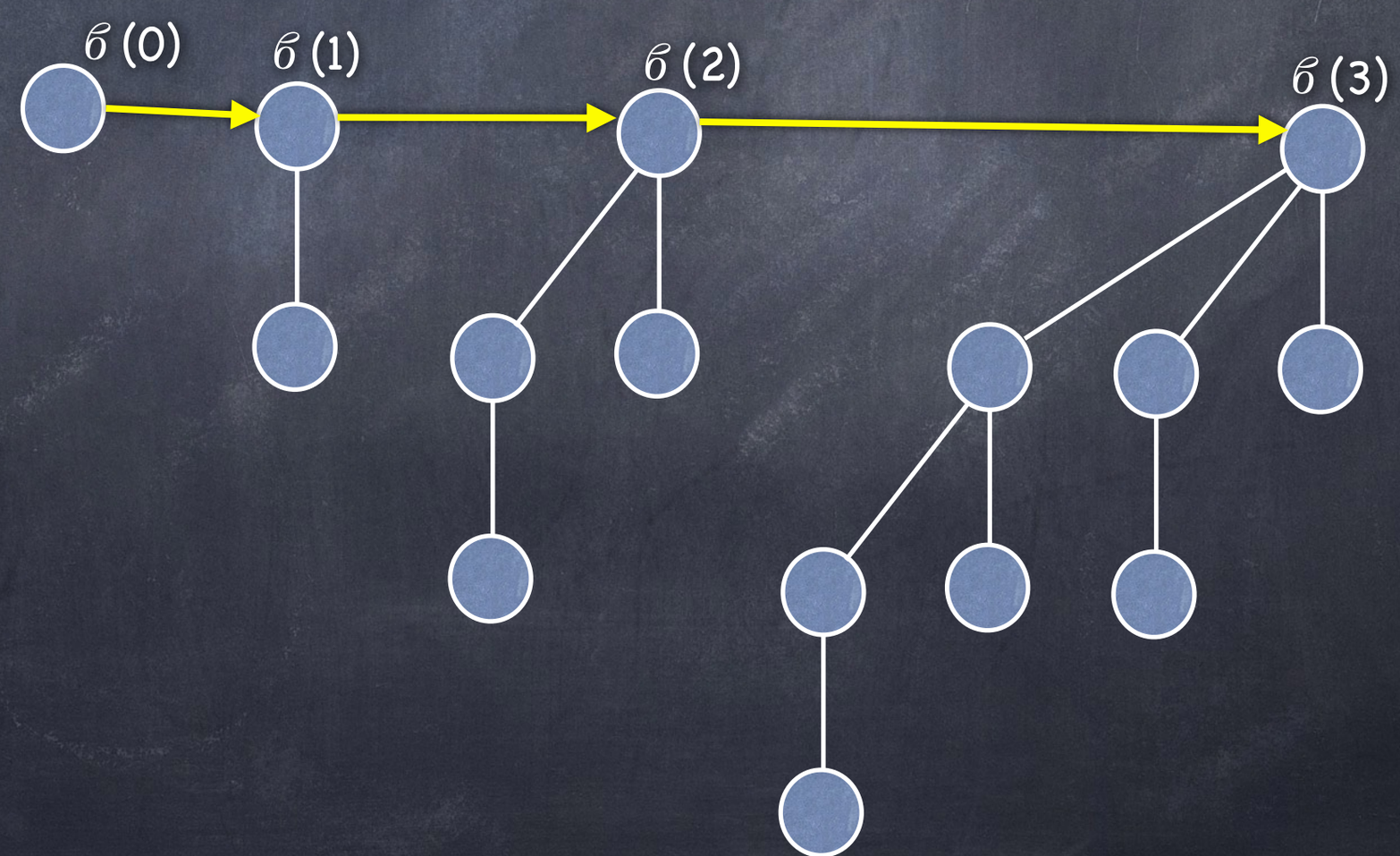

$\mathcal{B}(0)$
$\mathcal{B}(1)$
$\mathcal{B}(2)$
$\mathcal{B}(3)$

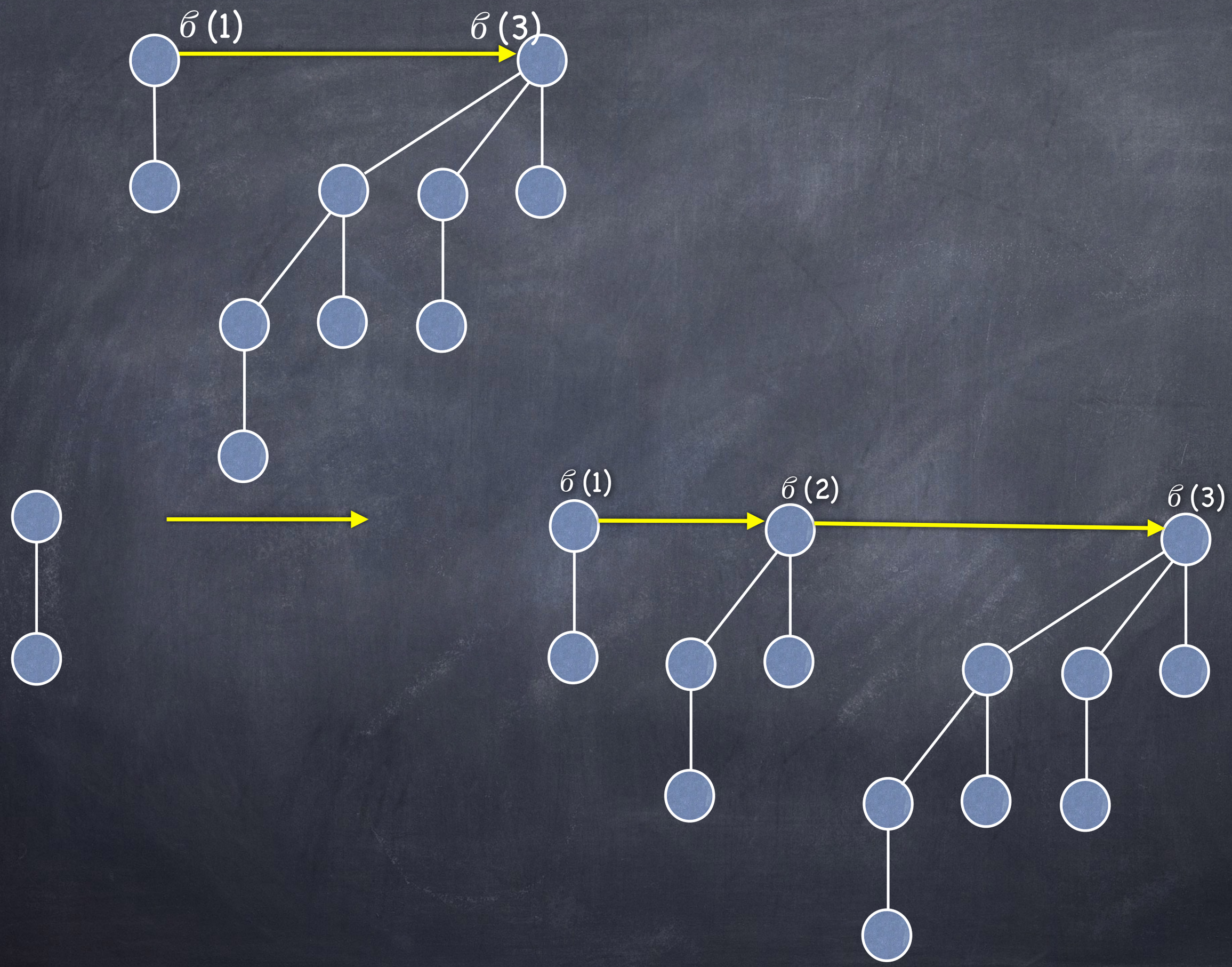


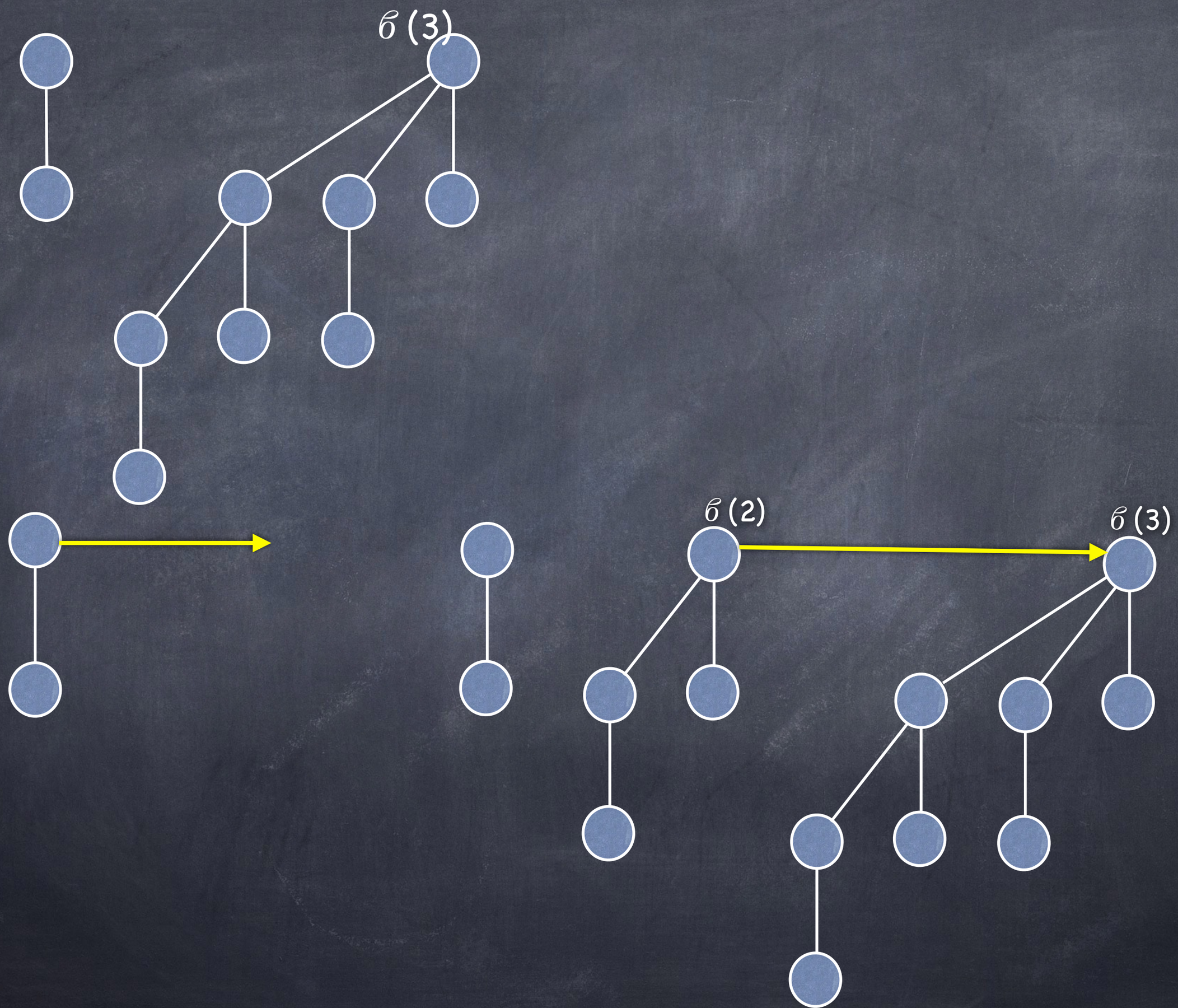
Eprisor s1.egyesit(Eprisor s2) ;

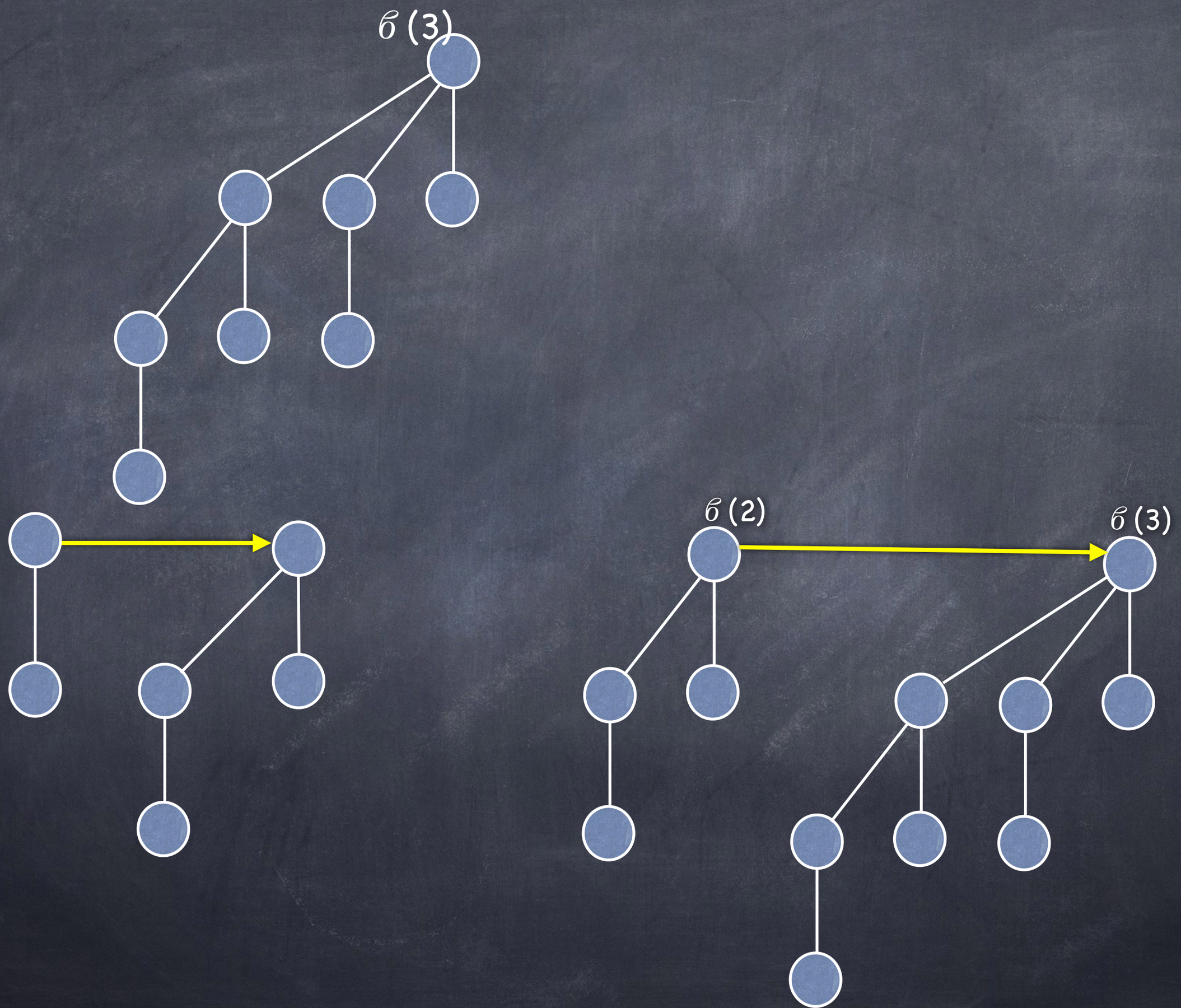


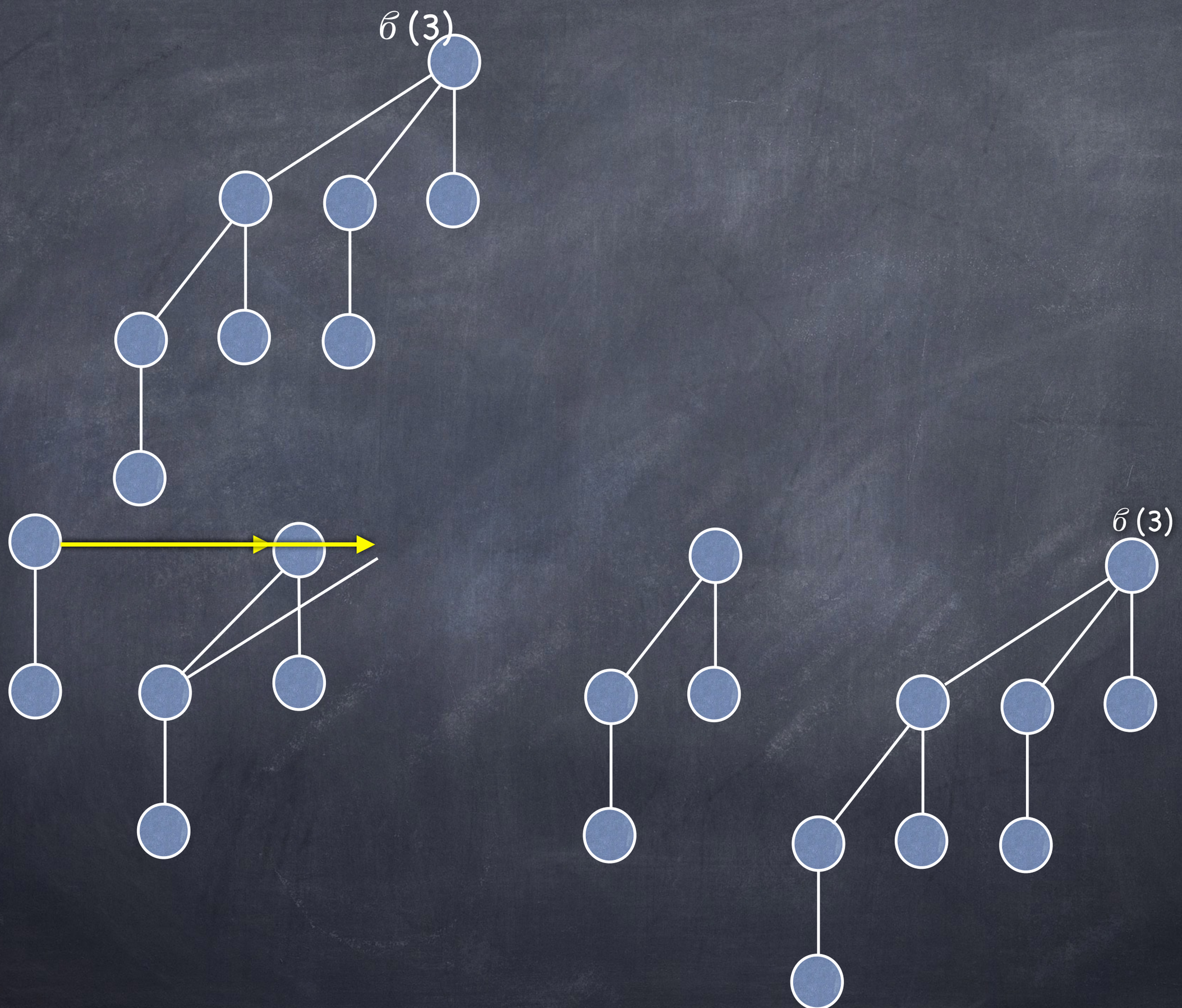

$\mathcal{b}$ (0)
$\mathcal{b}$ (1)
$\mathcal{b}$ (3)

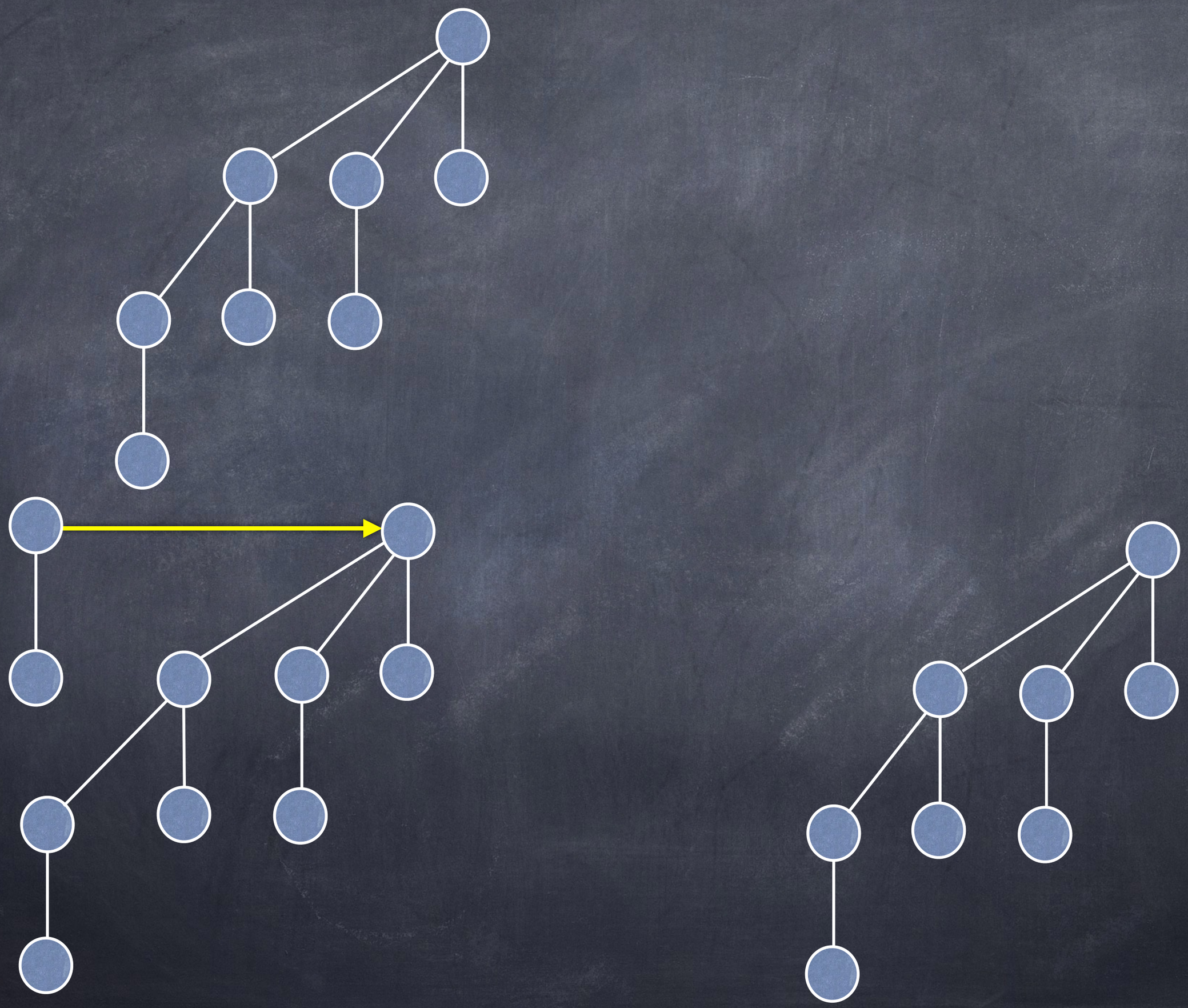

$\mathcal{b}$ (0)
$\mathcal{b}$ (1)
$\mathcal{b}$ (2)
$\mathcal{b}$ (3)

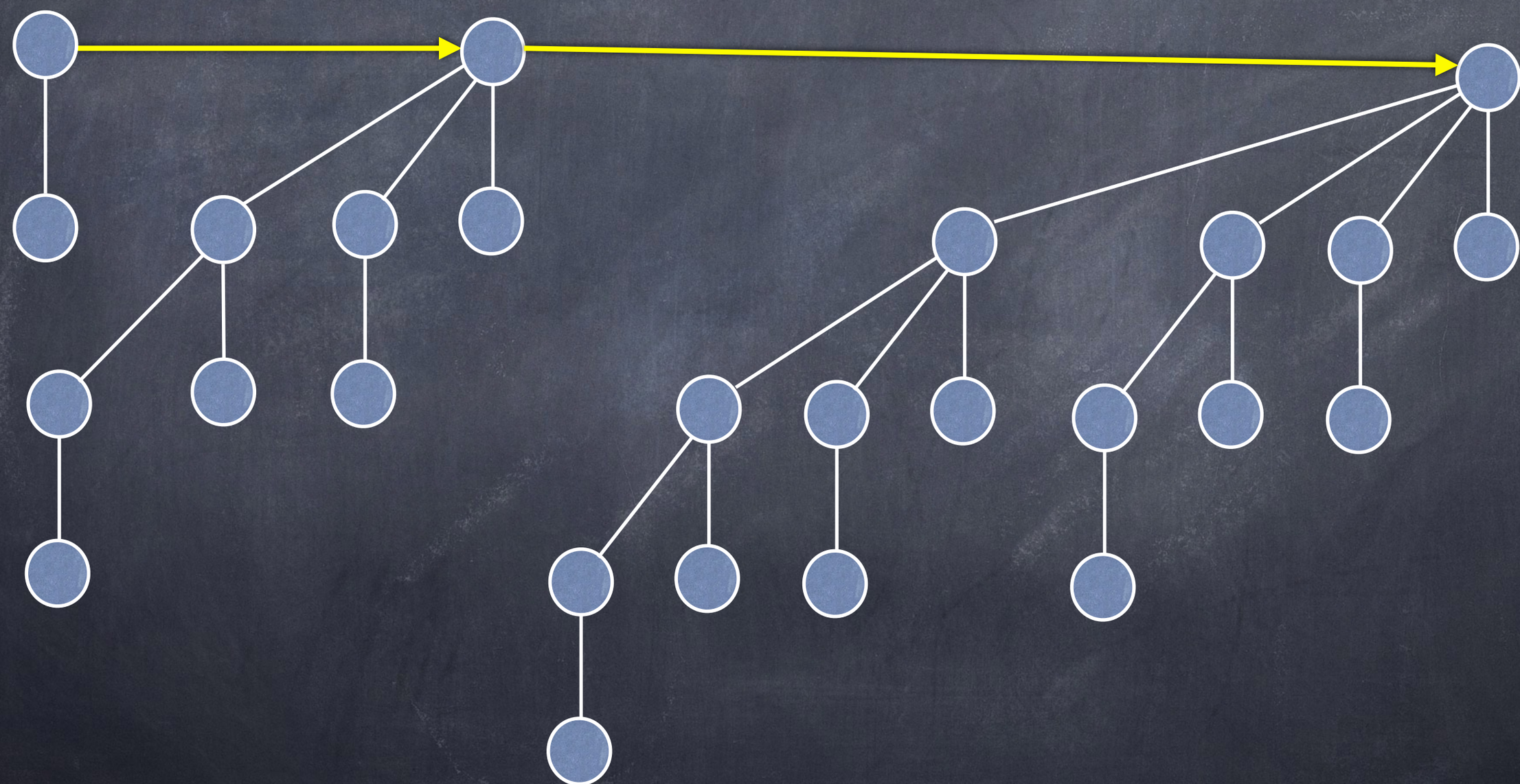

$\mathcal{b}$ (1)
$\mathcal{b}$ (3)
$\mathcal{b}$ (1)
$\mathcal{b}$ (2)
$\mathcal{b}$ (3)


$\mathcal{B}(3)$
$\mathcal{B}(2)$
$\mathcal{B}(3)$

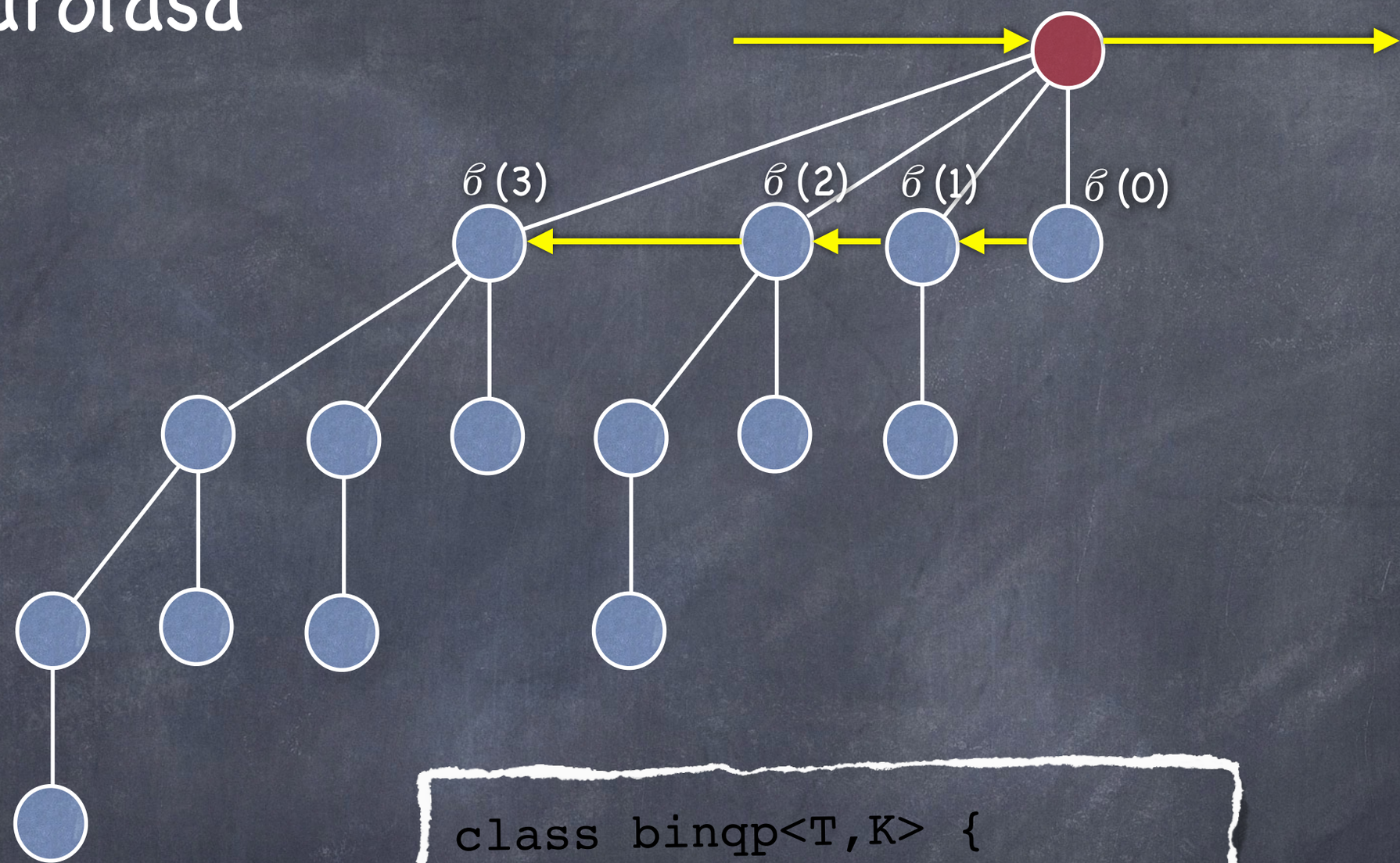

$\mathcal{B}(3)$
$\mathcal{B}(2)$
$\mathcal{B}(3)$


$\mathcal{B}(3)$
$\mathcal{B}(3)$



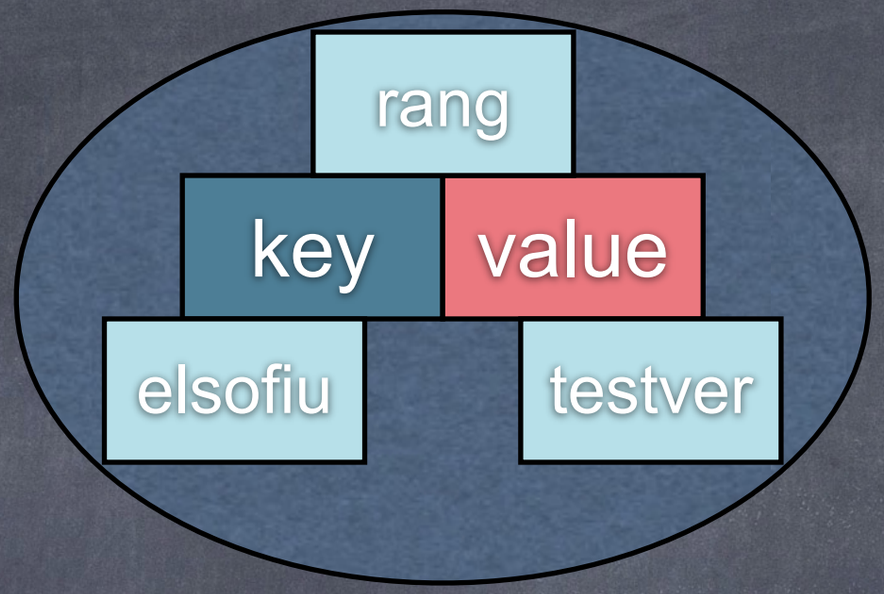


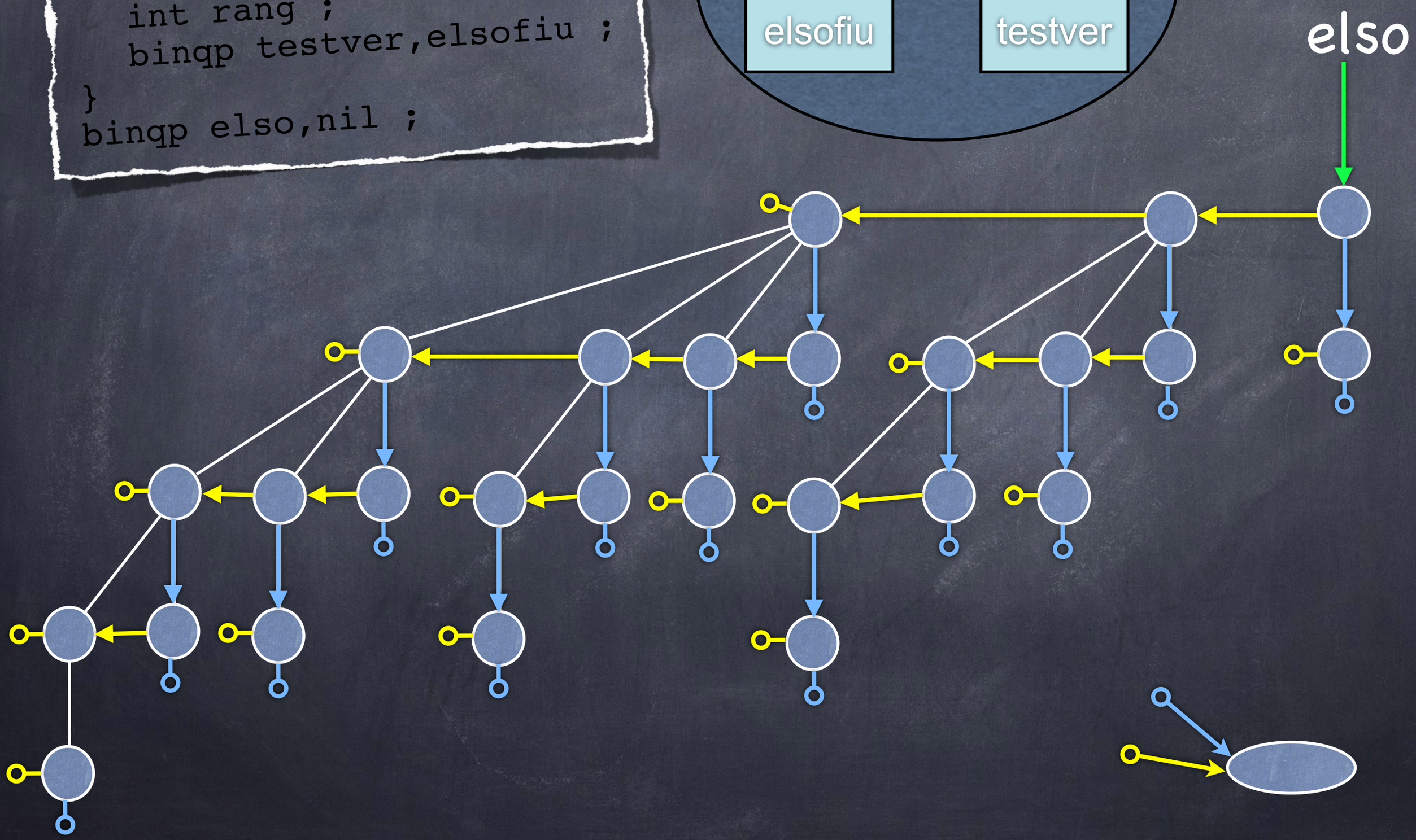
Pont tárolása




```
class binqp<T,K> {
   T key ;
   K value ;
   int rang ;
   binqp testver,elsofiu ;
}
```

```
class binqp<T,K> {
   T key ;
   K value ;
   int rang ;
   binqp testver,elsofiu ;
}
binqp elso,nil ;
```





Egyesíthető prioritási sor műveletek:

egyesít	O(logn)
sorból	O(logn)
sorba	O(logn)

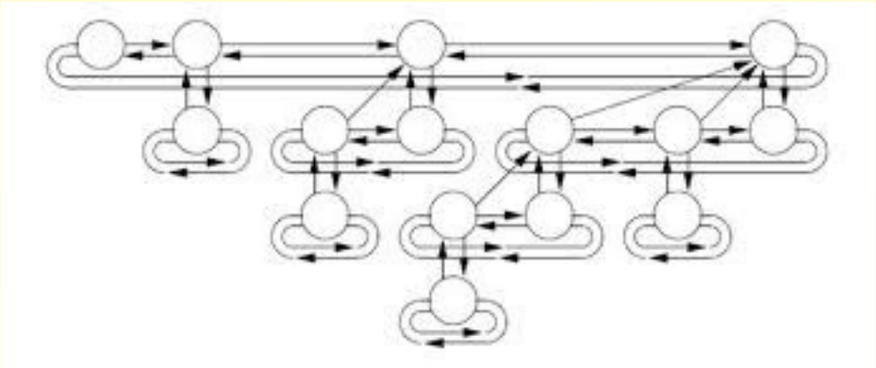
- létrehozunk egy 1 elemű kupacot
- egyesítjük az s kupaccal

létrehoz	O(1)
megszüntet	O(n)

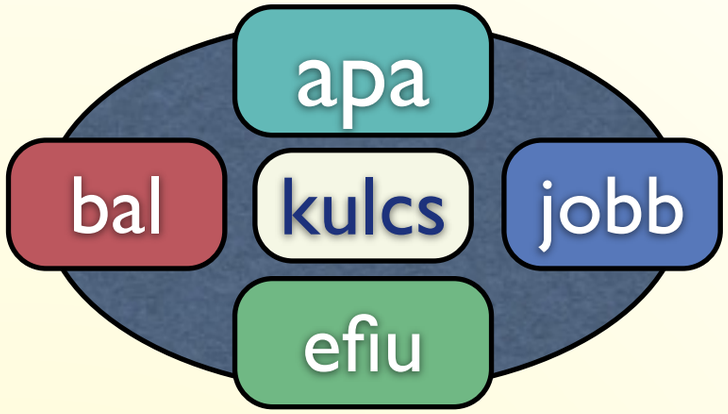
```
módosít(x,k) {  O(log(n))
  if (k>x.kulcs) {
    x.kulcs=k ;
    y=x
    z=x.apa ;
    while(z!=nil and y.kulcs<z.kulcs) {
      csere(y,z) ;
    y=z ;
    z=y.apa ;
    }
  } else ... (HF!)
}
```

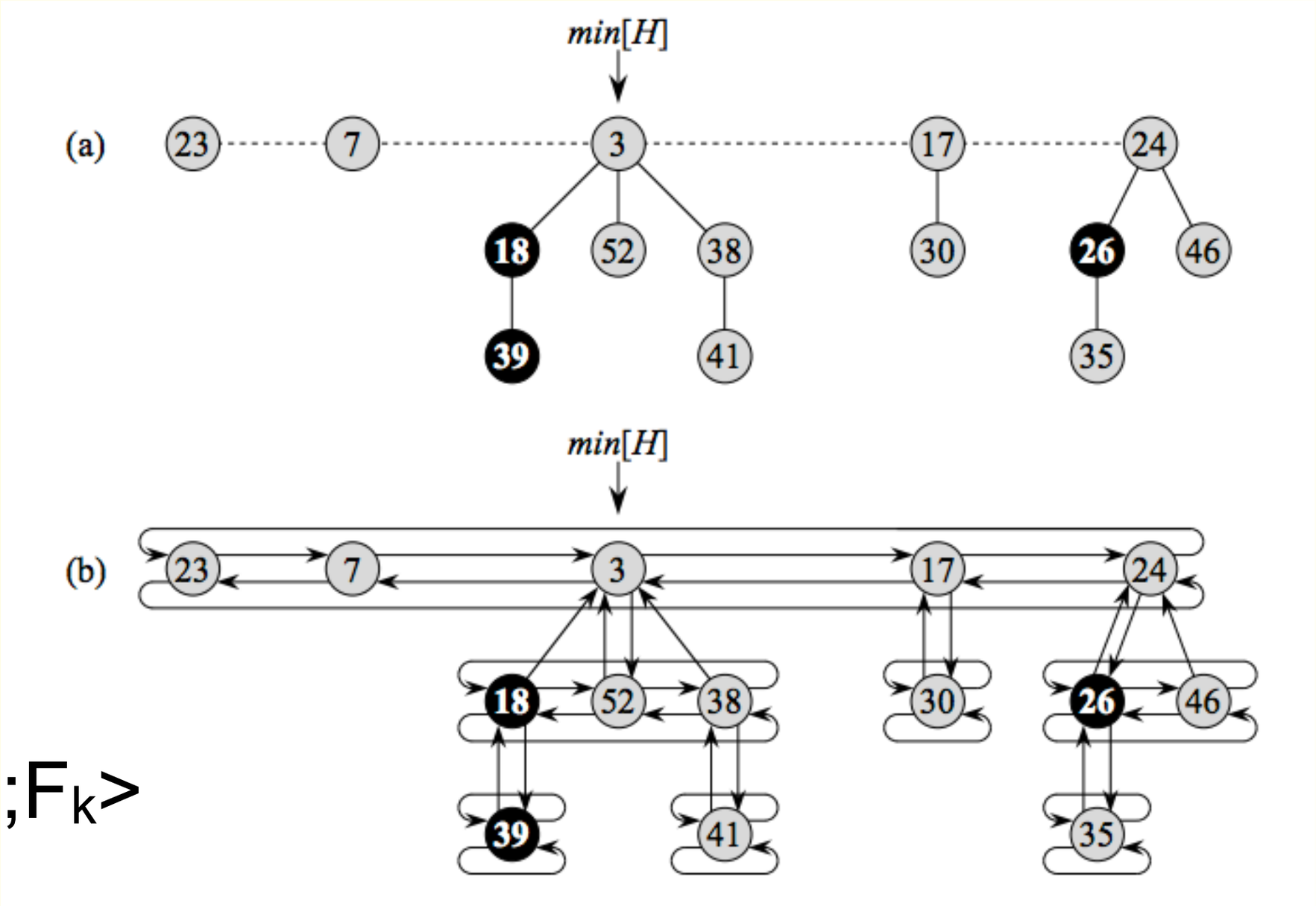
```
töröl(x) {  O(log(n))
  módosít(x,INF) ;
  sorból() ;
}
```

Fibonacci kupac



```
private class FibFaPont<E>{
        E kulcs;
        FibFaPont<E> bal, jobb, apa, efiu;
        byte fokszam;
        boolean megjelol=false;
        FibFaPont(E k){
                kulcs=k;
                this.bal=this;
                this.jobb=this;
        }
}
```





$H = \langle F_1;\ldots;F_k \rangle$

Jelölje t(H) a H sorozatbeli fák számát, m(H) pedig a megjelölt pontok számát!

$$\varphi(H) = t(H) + 2m(H)$$

Műveletek

H=egyesit(H_1,H_2);

A művelet egyszerűen megvalósítható két körlánc közönséges egyesítésével. Egyesítsük a ***H_1*** és ***H_2*** körláncot, majd ***H_1***:min és ***H_2***:min közül a kisebb kulcsot tartalmazó legyen az egyesített sorozat kijelölt minimum pontja.

$$\Delta\varphi :$$

$$\varphi(H) - (\varphi(H_1) + \varphi(H_2)) =$$

$$= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2)))$$

$$= t(H) - ((t(H_1) + t(H_2)) + 2m(H) - (2m(H_1) + 2m(H_2))$$

$$= 0$$

O(1)

$$\hat{c}_e = c_e + \Delta\varphi((H_1, H_2) \Rightarrow H)$$

O(1)

sorba(x);

Képezzünk a beszúrandó x elemből egypontú fát, majd egyesítsük a H-t ábrázoló kupaccal.

$\Delta\varphi$: $((t(H)+1)+2m(H))-(t(H)+2m(H))=1$.

O(1)

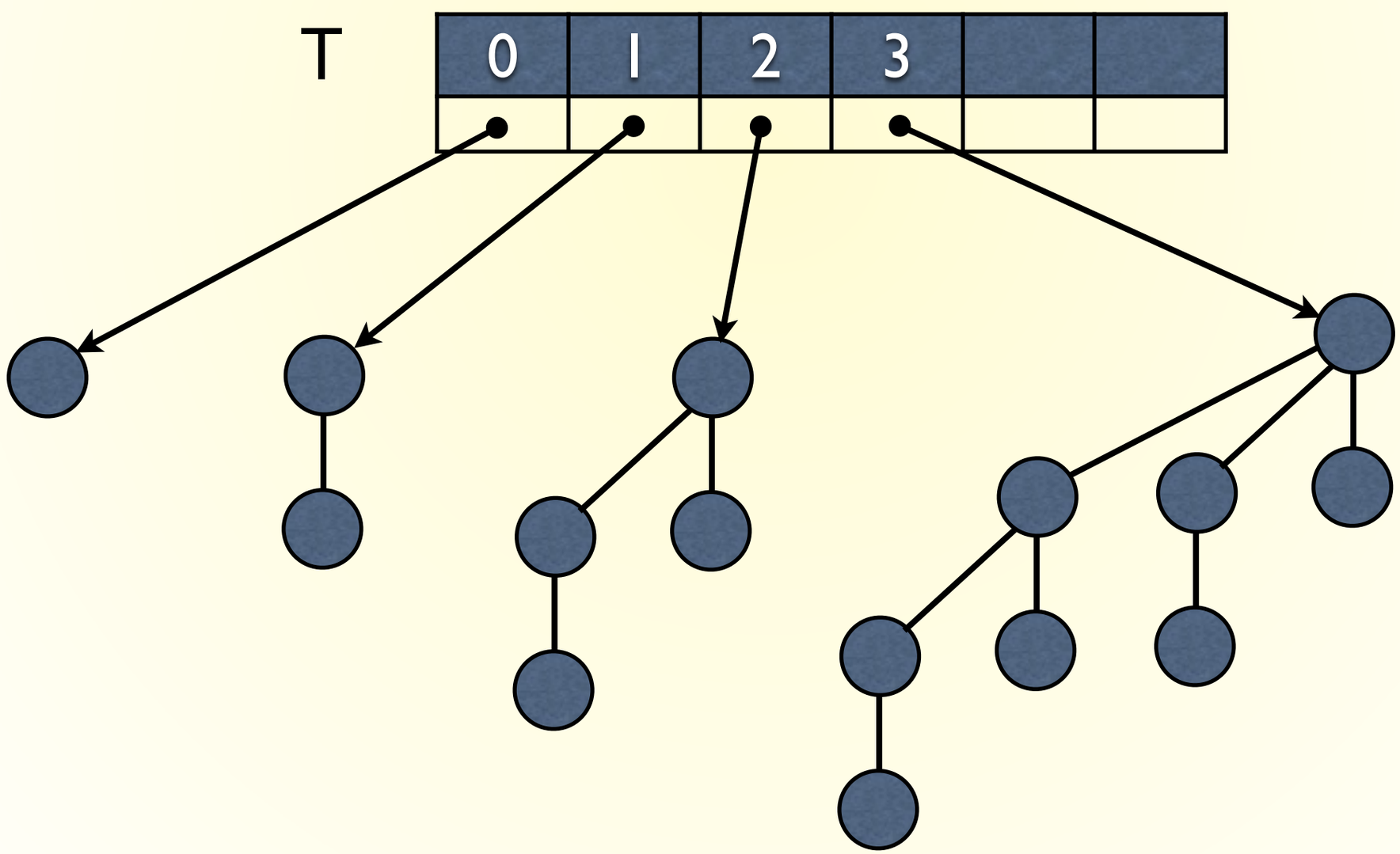
x=sorbol();

Az amortizált elemzés során feltételezzük, hogy adott egy D(n) felső korlát az n pontot tartalmazó Fibonacci kupacban szereplő csúcsok maximális fokszámára. Később az is igazolni fogjuk, hogy D(n) = O(log(n)).

Az algoritmus a minimum elem fiait átrakja a gyökérlistába, majd végrehajtja a ***kiegyenlit*** eljárást.

```
sorbol(x) {
  z=max ;
  if (z==nil) return z ;
  for (z minden x fiára) {
    x-et tegyük a gyökérlistába
    x.apa=nil ;
  }
  vegyük ki z-t a gyökérlistából
  if (z==z.jobb) max=nil ;
  else max=z.jobb ;
  kiegyenlit() ;
  elemszam-- ;
}
```

A ***kiegyenlit*** használja a ***kupszerk(y,x)*** eljárást, ami az y gyökerű fát berakja az x gyökerű fa alá. Továbbá felhasznál egy T tömböt, amelynek a mérete D(n) és amelyre T[i] az i fokszámú fát fogja tartalmazni.



```
kiegyenlit() {
   for (i=1;i<D(n);i++) A[i]=nil ;
   for (S minden w fájára) {
      x=w;
      d=x.fokszam;
      while (A[d]!=nil) {
         y=A[d] ;
         if (x.kulcs > y.kulcs) csere(x,y)
         kupszerk(y,x);
         A[d]=nil;
         d++;
      }
      A[d]=x;
   }
   max=nil ;
   for (i=1;i<D(n);i++)
      if (A[i]!=nil) {
         "tegyük A[i]-t a gyökérlistába"
         if (max==nil or A[i].kulcs>max.kulcs) min=A[i] ;
      }
}
```

```
kupszerk(y,x) {
   "vegyük ki y-t S-ből"
   "tegyük y-t x egy fiává"
   "növeljük x fokszámát"
   x.megjelolt=false;
}
```

A gyökérlista mérete legfeljebb $D(n) + t(H) - 1$,
így az aktuális költség $O(D(n) + t(H))$

A potenciál a min. pont kivágása előtt: $t(H) + 2m(H)$

A kivágás után $D(n) + 1 + 2m(H)$

$$O(D(n)+t(H)) + ((D(n)+1)+2m(H))-(t(H)+2m(H))$$

$$= O(D(n)) + O(t(H))-t(H) = O(D(n))$$

`módosít(x,k)`

Az algoritmus kivágja az adott elemet ha sérül a kupactulajdonság, továbbá a KASZKÁD-VÁGÁS algoritmus segítségével gondoskodik arról, hogy ne legyen olyan pont, ami több fiát is elveszti.

```
modosit(x,k) {
  if (k>x.kulcs) {
    x.kulcs=k ;
    y=x.apa ;
    if (y!=Nil and x.kulcs<y.kulcs) {
      kivag(x) ;
      kaszkadvag(y) ;
    }
    if (x.kulcs<max.kulcs) min=x ;
  } else ...
}
```

```
kivag(x) {
  vegyük ki x-et x.apa fiainak listájából
  tegyük bele x-et a gyökérlistába
  x.apa=nil ;
  x.megjelolt=False ;
}


kaszkadvag(y) {
  z=y.apa ;
  if (z!=nil) {
    if (y.megjelolt==False) y.megjelolt=True ;
    else kivag(y) ;
    kaszkadvag(z) ;
  }
}
```

A ***módosít*** algoritmus futási idejének elemzése:

1. A tényleges költség kiszámítása.

   Tegyük fel, hogy ***módosít*** adott hívására c számú ***kaszkádvág*** hajtódik végre. Ekkor ***módosít*** tényleges futási ideje O(c).

2. A potenciál változás:

   A ***kaszkádvág*** minden hívása, az utolsó kivételével, kivesz egy fapontot a fából és beteszi azt H gyökérlistájába. Így a ***kaszkádvág***-ok végrehajtása után t(H)+c fa lesz a gyökérlistában. (t(H) volt eredetileg, az x pont, és még másik c-1 pont került be a vágások eredményeként). Legfeljebb m(H)-c+2 pont lesz megjelölve, mert c-1 pont jelöletlenné vált a ***kaszkádvág***ások miatt, és esetleg egyet még jelöletlenné tesz.

A potenciál változása: ((t(H) +c) +2(m(H)-c+2))-(t(H) +2m(H)) = 4-c

Tehát a ***módosít*** eljárás amortizált futási ideje legfeljebb O(c) +4-c = O(1)

A gyökérlista mérete legfeljebb D(n).

A maximális fokszám felső korlátja:

Lemma.
Legyen x a H Fibonacci-kupac tetszőleges pontja, amelynek fokszáma k.
Jelölje az $y_1, y_2, \ldots, y_k$ sorozat x fiait abban az időrendi sorrendben, ahogyan x fiai lettek!
Ekkor
y_1.fokszam ≥ 0 és
y_i.fokszam ≥ i - 2 , ∀i = 2, ..., k-ra.

Bizonyítás.
Az y_1.fokszam ≥ 0 állítás nyilvánvaló.
$i \geq 2$ esetén vegyük észre, hogy amikor y_i-t x-hez kapcsoltuk annak fiaként, akkor már az y_1, ..., y_{i-1} pontok x fiai voltak, így az x.fokszam = i -1 fennállt.
Az y_i pontot csak akkor kapcsoltuk x-hez, ha y_i.fokszam = x.fokszam, tehát y_i.fokszam = i - 1 is teljesült.
Azóta y_i legfeljebb egy fiát veszthette el, mert egyébként x-ből kivágtuk volna.

Tehát y_i.fokszam ≥ i - 2.

Tekintsük a következő Fibonacci számsorozatot:

$$F_k = \begin{cases} 0 & \text{ha } k = 0\,, \\ 1 & \text{ha } k = 1\,, \\ F_{k-1} + F_{k-2} & \text{ha } k \geq 2\,. \end{cases}$$

k	F_k	$\sum_{i=0}^{k} F_i$	$1+\sum_{i=0}^{k} F_i$
1	1	1	2
2	1	2	3
3	2	4	5
4	3	7	8
5	5	12	13
6	8	20	21
7	13	33	34
8	21	54	55

Lemma.

Minden k ≥ 0 egész számra:

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i\,.$$

Bizonyítás

k-szerinti indukcióval bizonyítunk.

k = 0-ra:

$$1+\sum_{i=0}^{0} F_i \;=\; 1+F_0 \;=\; 1+0 \;=\; 1 \;=\; F_2$$

Az indukciós feltétel szerint $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$

tehát:

$$F_{k+2} \;=\; F_k + F_{k+1} \;=\; F_k + \left(1+\sum_{i=0}^{k-1} F_i\right) \;=\; 1+\sum_{i=0}^{k} F_i$$

Lemma

Minden $k \geq 0$ *egész számra* $F_{k+2} \geq \Phi^k$.

Bizonyítás

Bizonyítás k-szerinti indukcióval.

$$F_2 = 1 = \Phi^0, F_3 = 2 \geq (1+\sqrt{5})/2 = \Phi^1.$$

$$F_{k+2} = F_k + F_{k+1} \geq \Phi^{k-2} + \Phi^{k-1} = \Phi^{k-2}(1+\Phi)$$

$$\Phi^2 = (1+\Phi)$$

$$F_{k+2} \geq \Phi^{k-2}\Phi^2 = \Phi^k$$

Lemma.

Legyen x a H Fibonacci-kupac tetszőleges pontja, amelynek fokszáma k. Ekkor $méret(x) \geq \Phi^k$

Bizonyítás

Jelölje s_k a legkevesebb pontot tartalmazó Fibonacci-kupac k fokszámú fájának pontjai számát!

$$s_0 = 1, s_1 = 2, s_2 = 3$$

s_k értéke k-szerint monoton nő

Jelölje az $y_1, y_2, ..., y_k$ sorozat x fiait abban az időrendi sorrendben, ahogyan x fiai lettek!

$$méret(x) \geq s_k = 2 + \sum_{i=2}^{k} s_{y_i.fokszam} \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}$$

Tehát $méret(x) \geq s_k \geq F_{k+2} \geq \Phi^k$.

következmény:

Egy n pontot tartalmazó Fibonacci-kupac tetszőleges pontjának maximális fokszáma: $D(n) = O(\lg n)$.

Bizonyítás:

$$n \geq \text{méret}(x) \geq \Phi^k$$

$$k \leq \log_\Phi n$$

Így bármely pont maximális fokszáma $D(n) = O(\lg n)$

```
töröl(x) {
  módosít(x,INF) ;
  sorból() ;
}
```

O(log(n))

Bin.Qpac vs. Fib.Qpac

	Bin.Qpac	Fib.Qpac
sorba	O(logn)	*O(1)
sorbol	O(logn)	*O(logn)
egyesit	O(logn)	*O(1)
modosit	O(logn)	*O(1)
torol	O(logn)	*O(logn)

Mintaillesztés

Egy Σ ábécé feletti szavakon a Σ^* (Σ elemeiből képzett véges sorozatok) elemeit értjük. Az $X \in \Sigma^*$ szó hossza a sorozat elemeinek a száma, jele $|X|$.

A továbbiakban feltételezzük, hogy a szavakat tömbökkel ábrázoljuk, tehát az X szó i-edik elemére az X[i] értékkel (x_i) hivatkozunk.

Az $X = x_1,\ldots,x_m$, és $Y = y_1,\ldots,y_n$ szavak konkatenációja: $XY = x_1,\ldots,x_m,y_1,\ldots,y_n$.

Az U szó kezdőszelete, vagy prefixe V-nek, jele $U \sqsubset V$, ha $\exists W \in \Sigma^*$, amelyre $UW = V$.

Az U szó végződése, vagy szuffixe V-nek, jele $U \sqsupset V$, ha $\exists W \in \Sigma^*$, amelyre $WU = V$.

Az $X \in \Sigma^*$ szó első i betűjéből álló prefixére az $X_i = X[1...i]$ $(x_1 ... x_i)$ jelölést használjuk.

Mintaillesztési probléma

Bemenet: $A,S \in \Sigma^*$, A a szöveg, S a minta.

Kimenet: A legkisebb (vagy másik problémában az összes) olyan i, amelyre $A[i+1..i+m] = S$, ahol $m = |S|$ a minta hossza.

A kimeneti feltétel esetén azt mondjuk, hogy az S minta i eltolással illeszkedik az A-ra.

A feladatnak nyilvánvalóan megoldását adja a következő algoritmus, amely az összes lehetséges eltolás végigpróbálásán alapul:

```
vegignez(A,S) {
   n=hossz(A) ;
   m=hossz(S) ;
   for (i=0;i<n-m;i++) {
      egyezik=true ;
      for (j=0;j<m and egyezik;j++) {
         if (A[i+j]!=S[j-1]) {
            egyezik=false ;
            exit ;
         }
         if (egyezik) return i-1;
      }
   }
   return -1 ;
}
```

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d
d o r t m

b l a b d o r t d o d o r t m u n d
d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

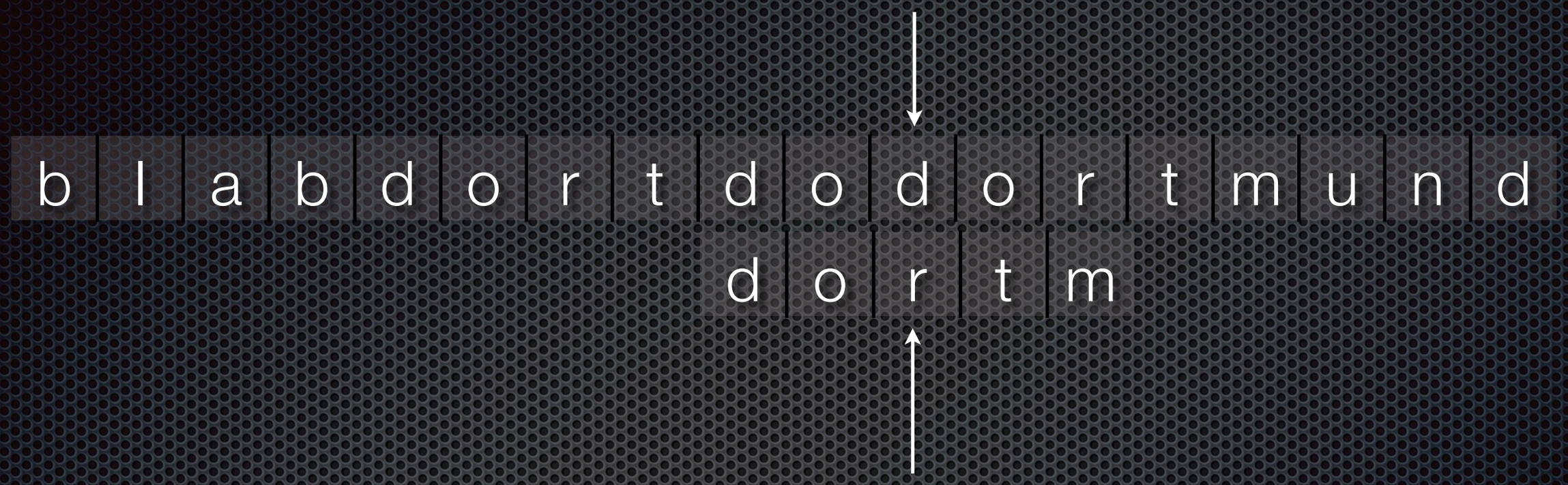
d o r t m

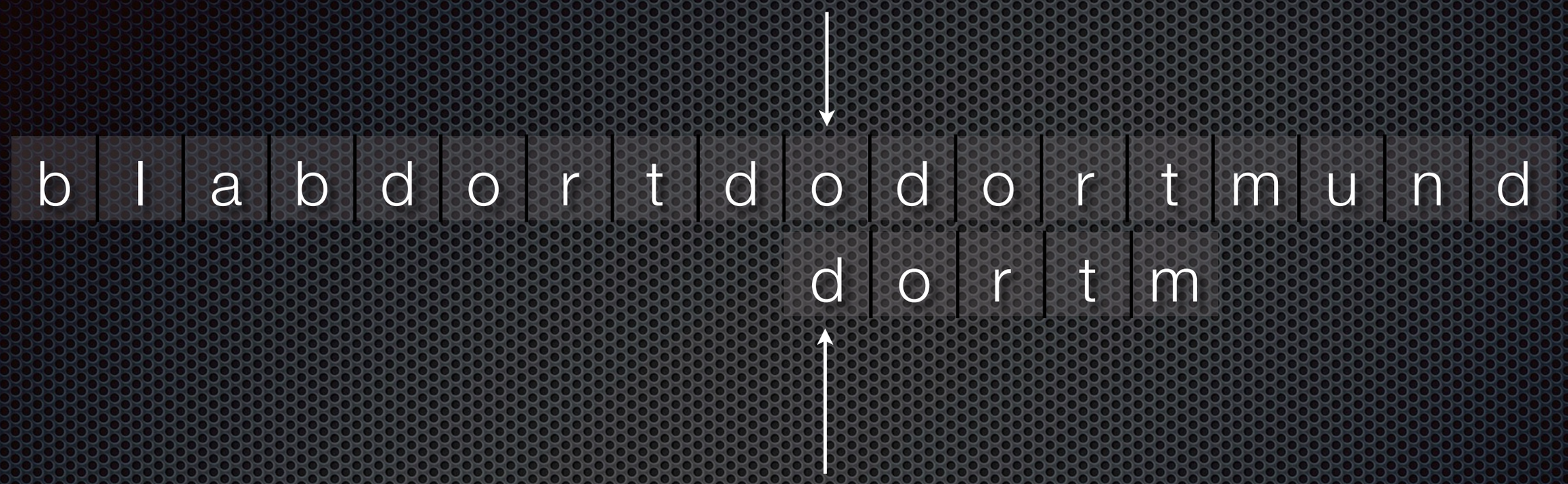
b l a b d o r t d o d o r t m u n d

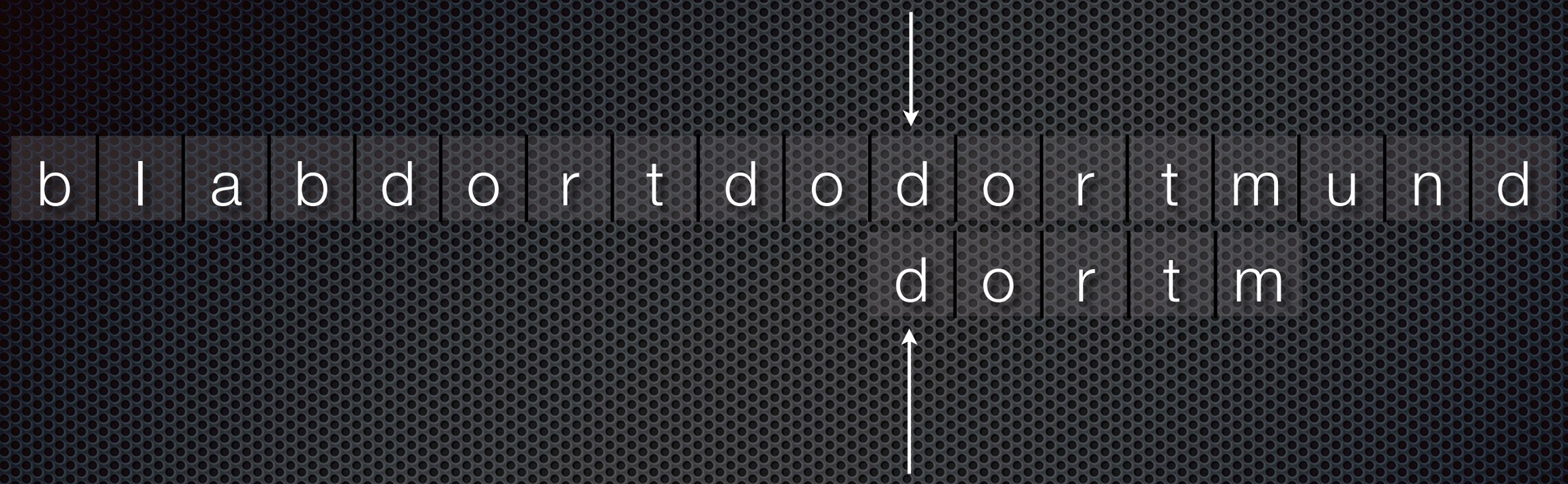
d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

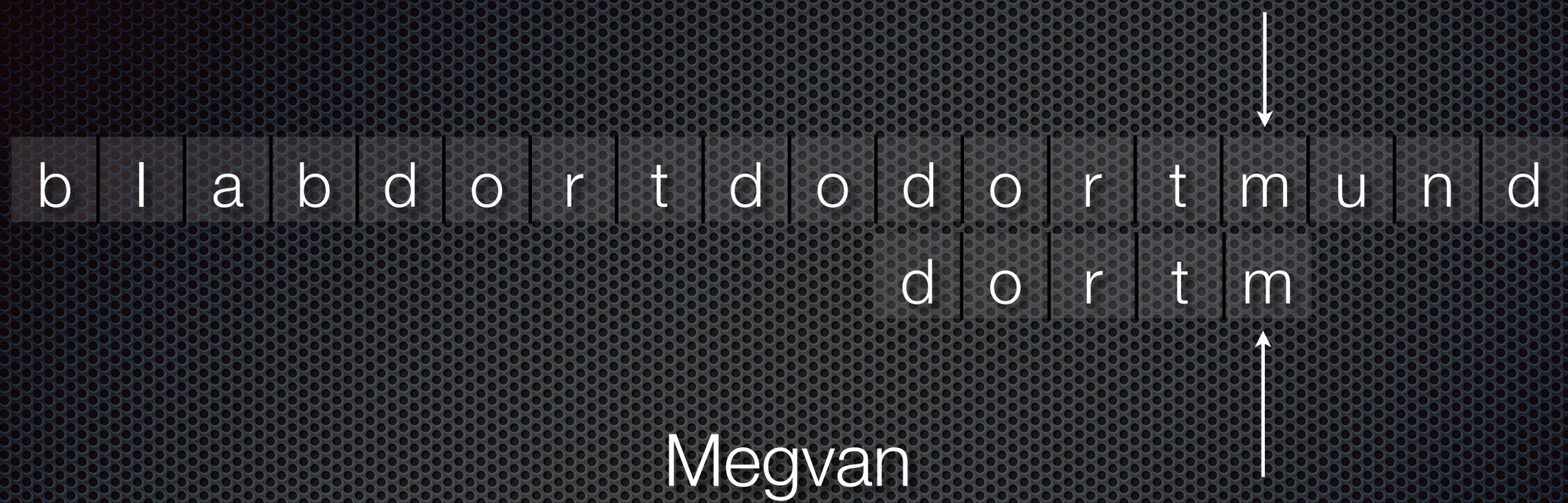
d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m
Megvan

Mennyi a futási idő legrosszabb esetre?

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

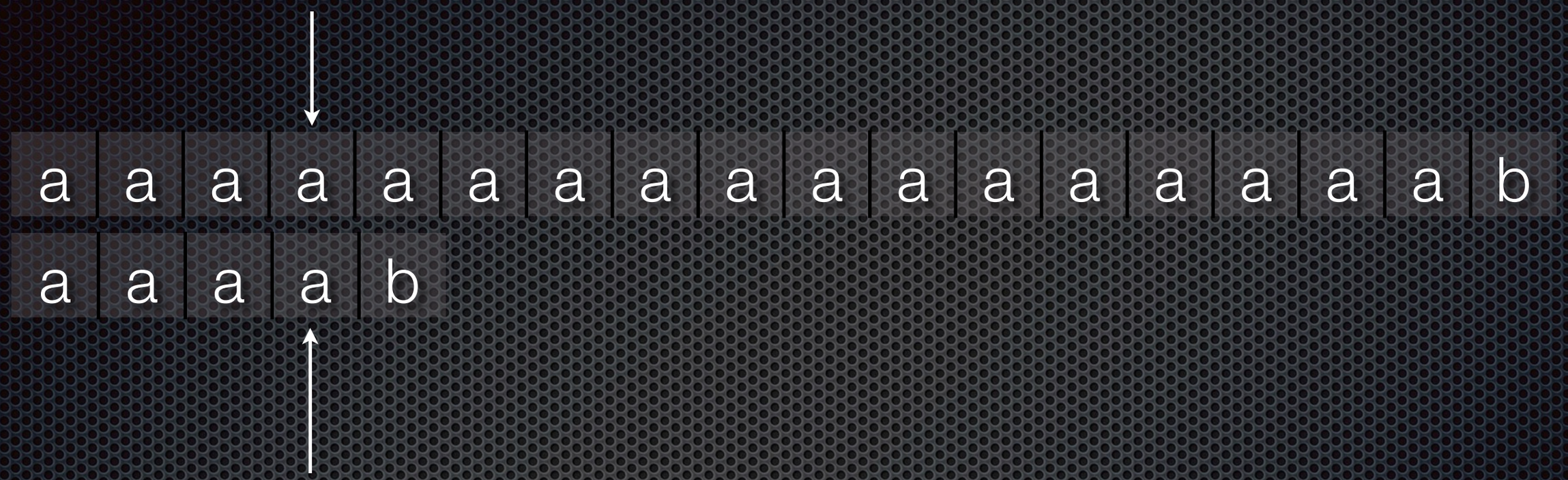
a	a	a	a	b

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

a	a	a	a	b

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

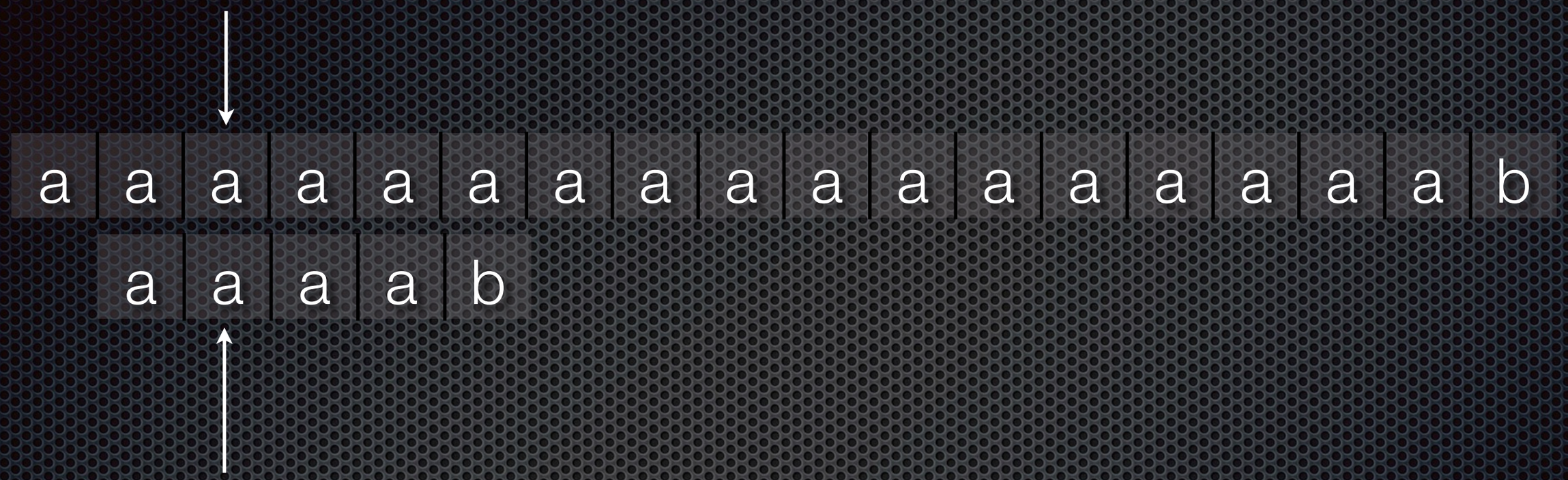
a	a	a	a	b


a a a a a a a a a a a a a a a a a a b
a a a a b

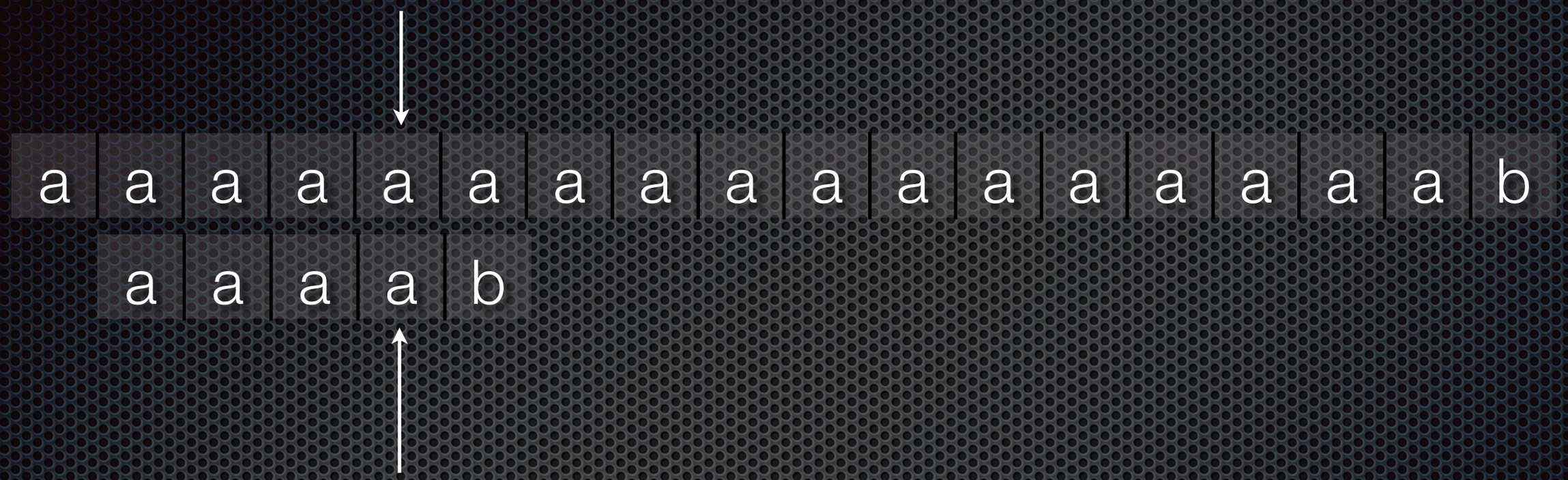
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

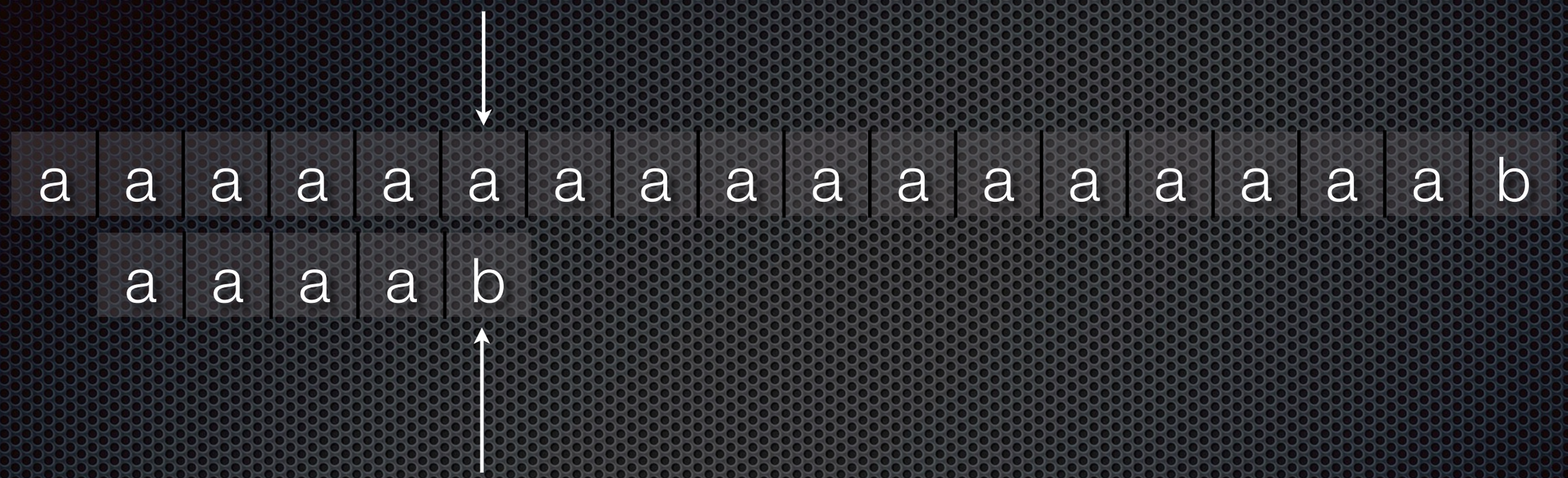
a	a	a	a	b

a a a a a a a a a a a a a a a a a a b
a a a a b


a a a a a a a a a a a a a a a a a a b
a a a a b

a a a a a a a a a a a a a a a a a a b
a a a a b


a a a a a a a a a a a a a a a a a a b
a a a a b

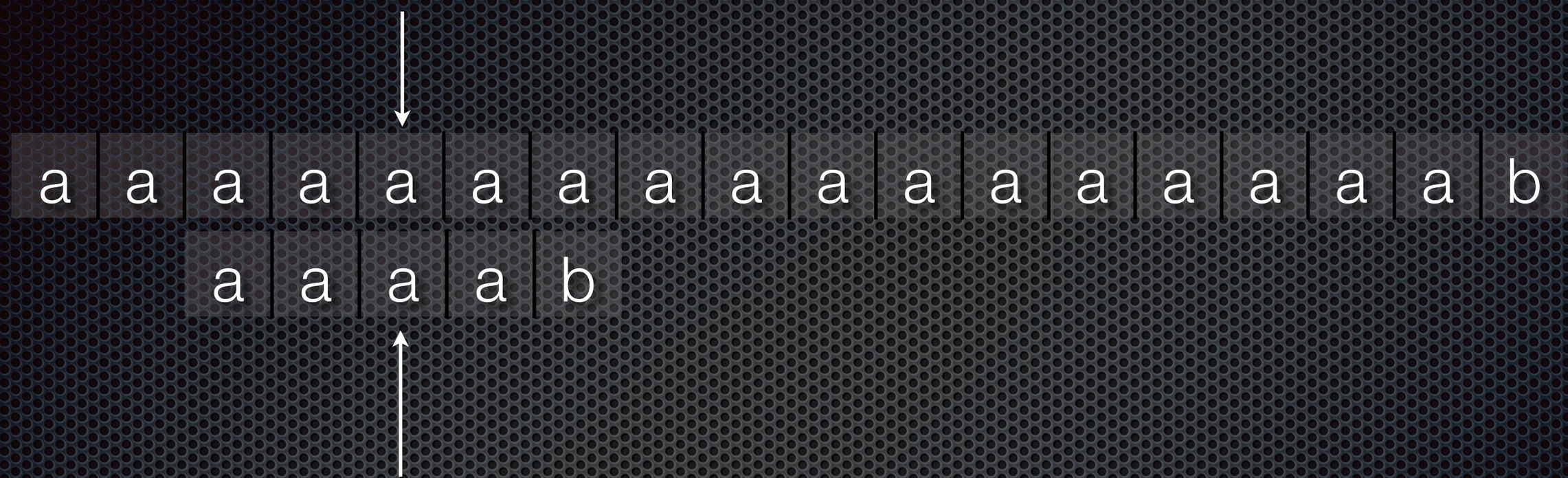

a a a a a a a a a a a a a a a a a a b
a a a a b

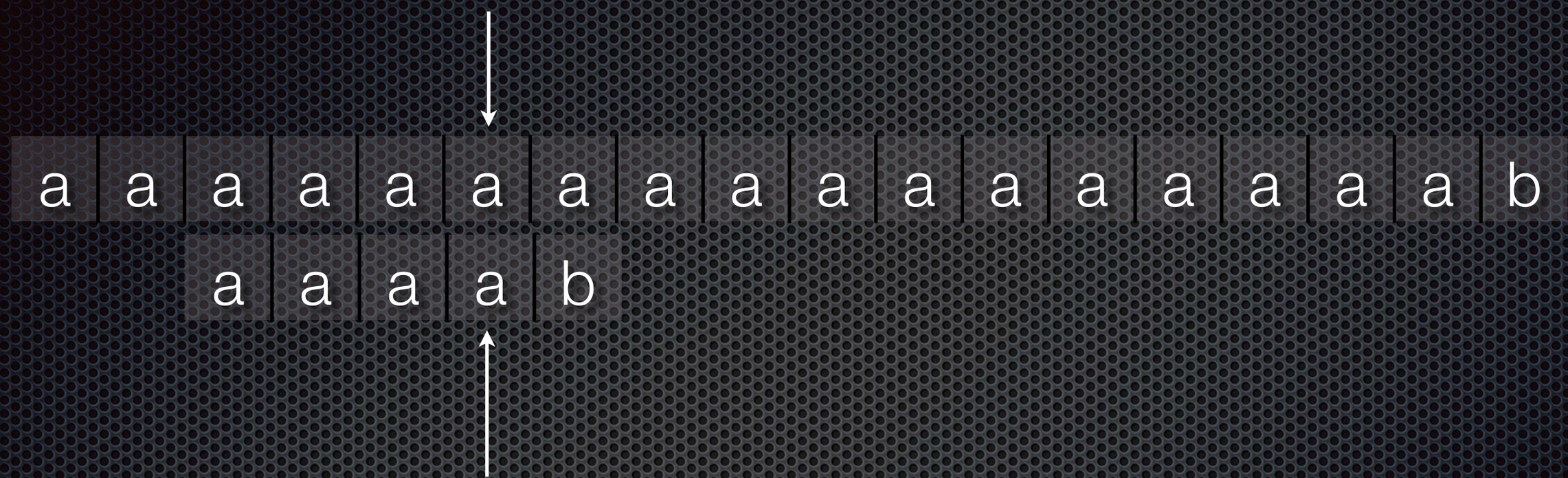
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

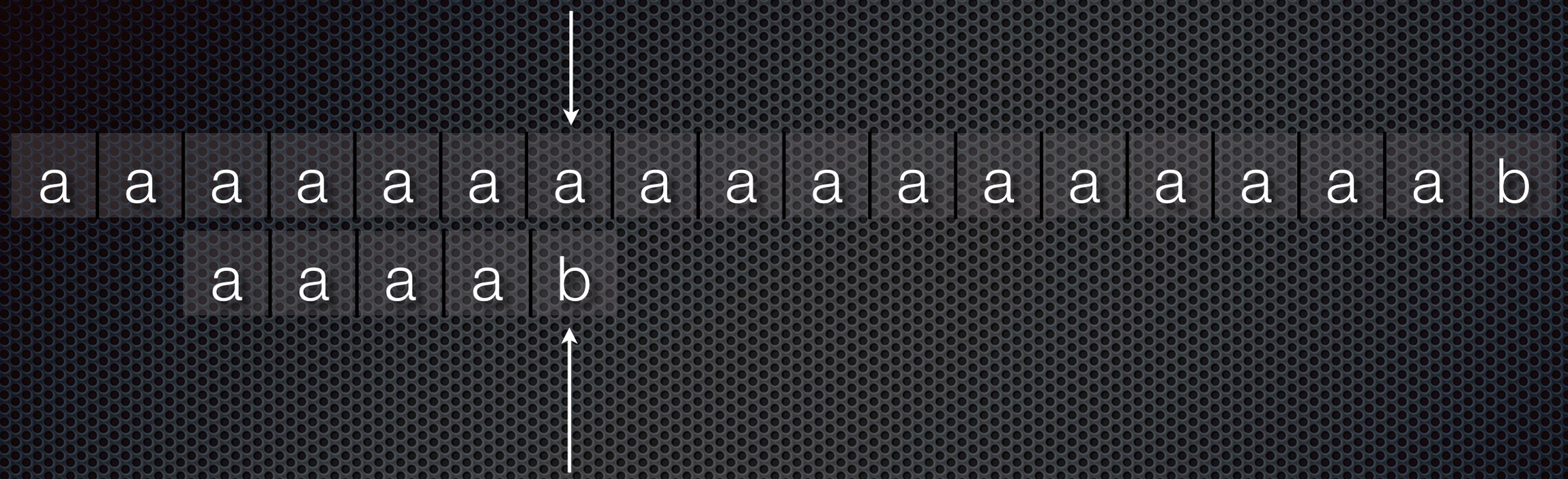
a	a	a	a	b

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

a	a	a	a	b


a a a a a a a a a a a a a a a a a a b
a a a a b


a a a a a a a a a a a a a a a a a a b
a a a a b


a a a a a a a a a a a a a a a a a a b
a a a a b

O(nm)

Mintaillesztés véges determinisztikus automatával

Véges determinisztikus automata olyan $M = (Q, q_0, F, S, \delta)$ rendezett ötös, ahol

- Q véges halmaz, az állapotok halmaza,
- $q_0 \in Q$ a kezdőállapot,
- $F \subseteq Q$ a végállapotok halmaza,
- Σ véges halmaz, a bemeneti jelek halmaza,
- $\delta : Q \times \Sigma \to Q$, az automata átmenetfüggvénye.

A δ átmenetfüggvényt a szokásos módon iterálással kiterjesztjük a $Q \times \Sigma^*$ halmazra.

Legyen $M = (Q,q_0,F,S,\delta)$ olyan véges determinisztikus automata, amely a Σ^*S nyelvet ismeri fel!

Mivel Σ^*S az összes olyan szavak halmaza, amelyek az S mintára végződnek, ezért A[0..i]-t pontosan akkor ismeri fel az automata, ha az S minta [i-m] eltolással előfordul az A szövegben.

Ekkor a következő algoritmus megoldja a problémát.

$M = (Q, q_0, F, S, \delta)$

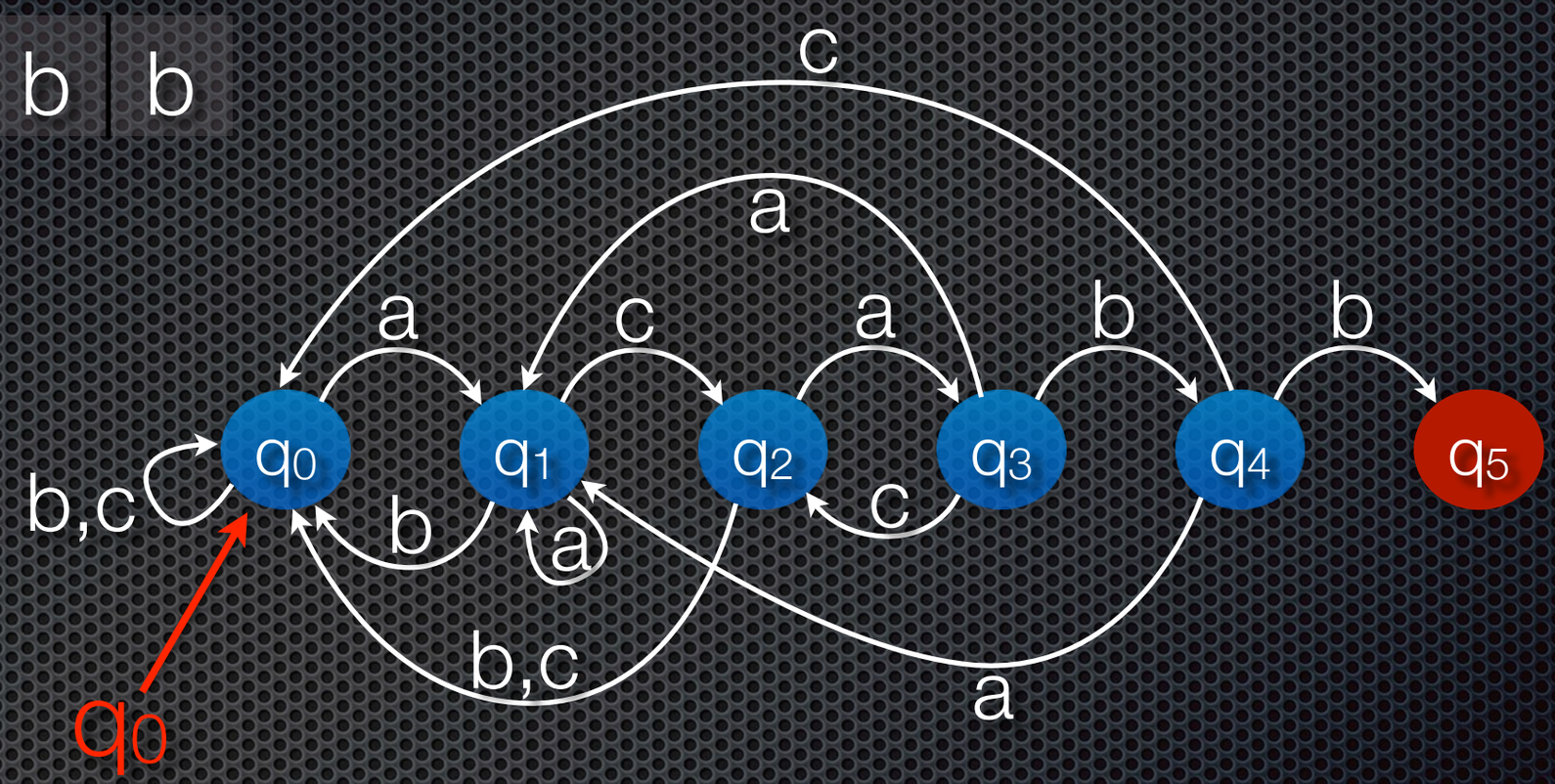
```
q = q0;
n = |A|;
m = |S|;
for (i=1;i<=n;i++) {
    q = δ(q,A[i]);
    if (q ∈ F) return i-m+1;
}
```

Az algoritmus futási ideje adott automata mellett O(n).

Példa

$\Sigma=\{a,b,c\}$

S	a	c	a	b	b



A	a	c	a	b	a	c	b	b	a	a	c	a	b	b	a
	q_1	q_2	q_3	q_4	q_1	q_2	q_0	q_0	q_1	q_1	q_2	q_3	q_4	q_5	

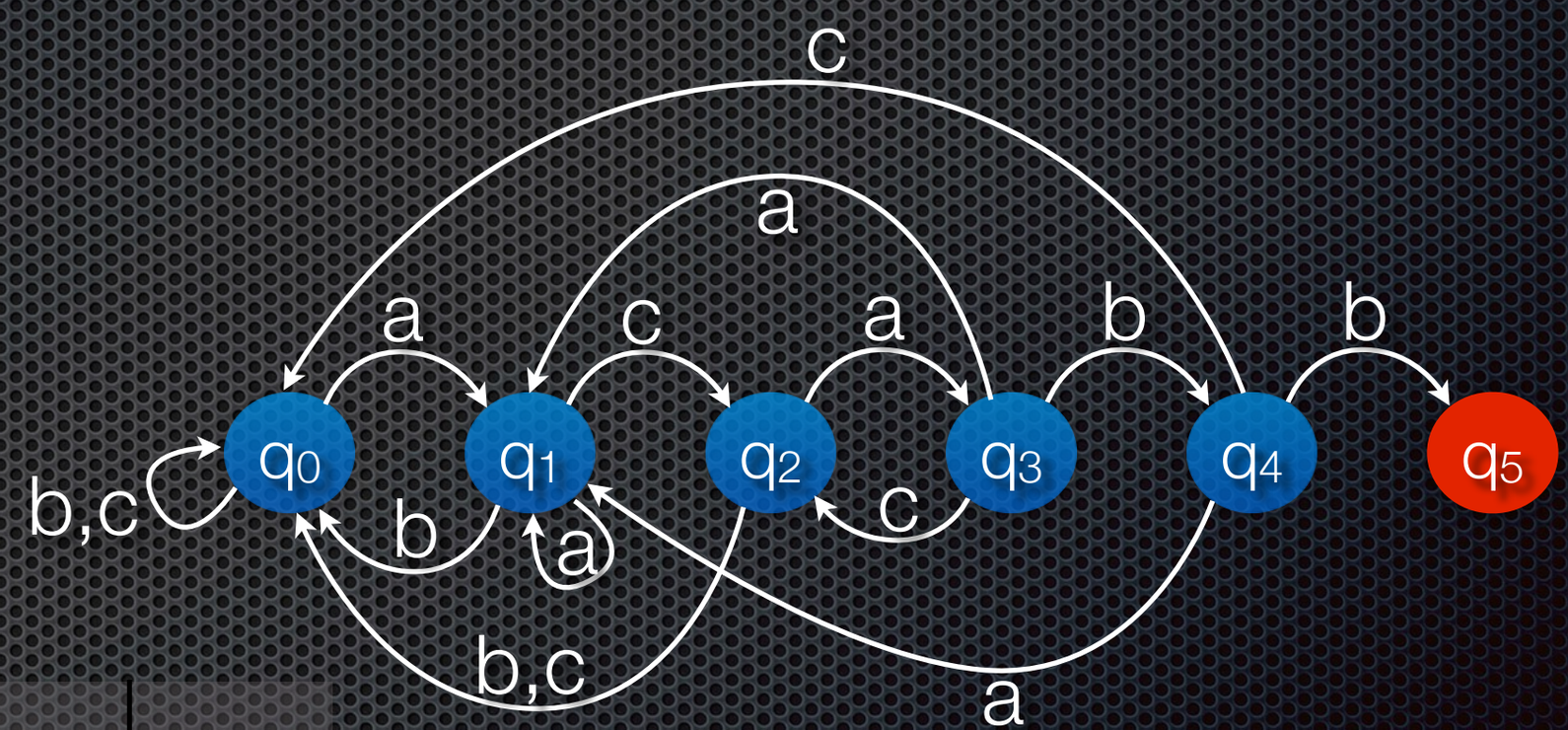
Hogyan tudjuk előállítani az M automata átmenetfüggvényét?

Vegyük azt az automatát, amelyben az állapotok halmaza 0,...,m és minden $q < m$ állapot és $x \in \Sigma$ esetén

$$\delta(q,x) = \begin{cases} q+1, & \text{ha } x = S[q+1], \\ \max\{k : S_k \sqsupset S_q x\}, & \text{ha } x \neq S[q+1], \end{cases}$$

```
m = |S| ;
for (q:= 0; q<=m;q++)
    for (∀x: x ∈ Σ) {
        for (k=min(m,q+1);Sk!⊐Sqx;k--) ;
        δ(q,x) = k ;
    }
}
```

	0	1	2	3	4
S	a	c	a	b	b



	q_0	q_1	q_2	q_3	q_4
a	q_1	q_1	q_3	q_1	q_1
b	q_0	q_0	q_0	q_4	q_5
c	q_0	q_2	q_0	q_2	q_0

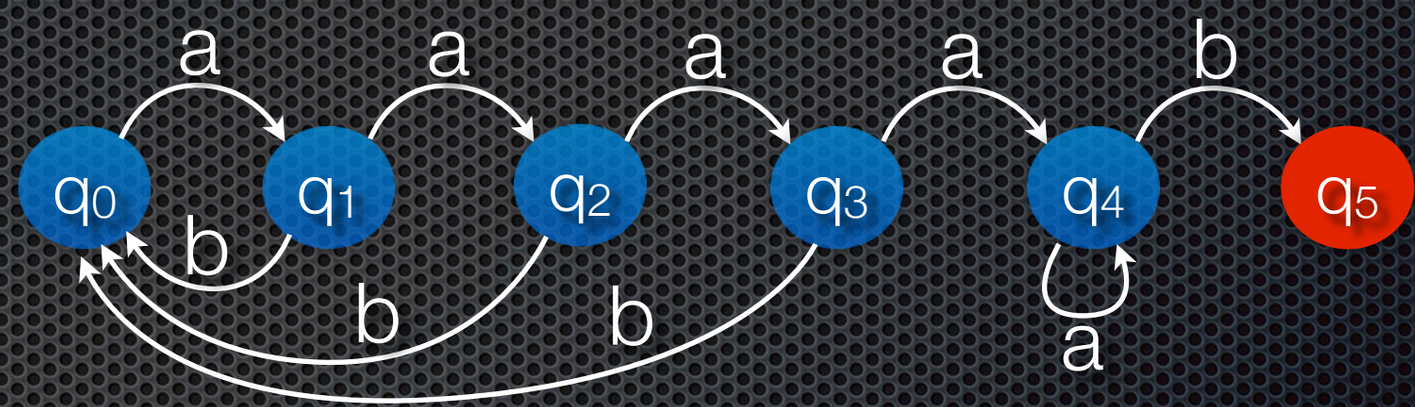
$O(|\Sigma|m^2) \leq O(m^3)$

Példa

$\Sigma=\{a,b\}$

S

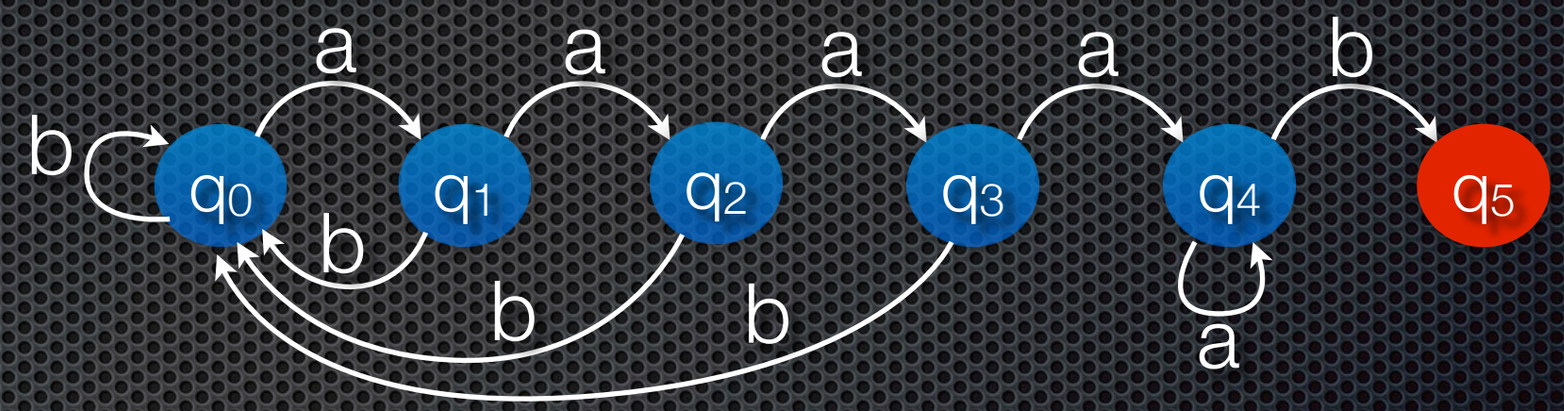
a	a	a	a	b



Példa

$\Sigma=\{a,b\}$

S | a | a | a | a | b |



A | a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | b |

Knuth Morris Pratt algoritmus

Ha egy helyen a minta egy kezdőszelet illeszkedik, akkor a következő néhány elemnél nem feltétlen kell ellenőrízni az illeszkedést, hanem a mintától és az aktuális illeszkedéstől függően lehet előre léptetni az összehasonlító algoritmust.
A léptetés mértéke előre kiszámítható a mintára.

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

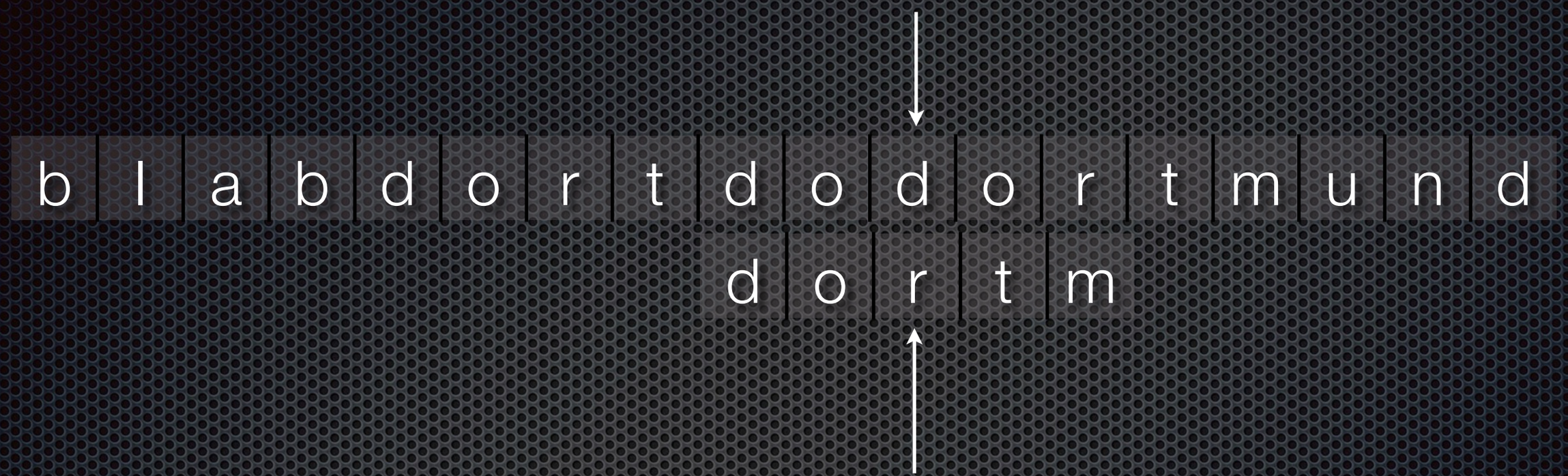
d o r t m

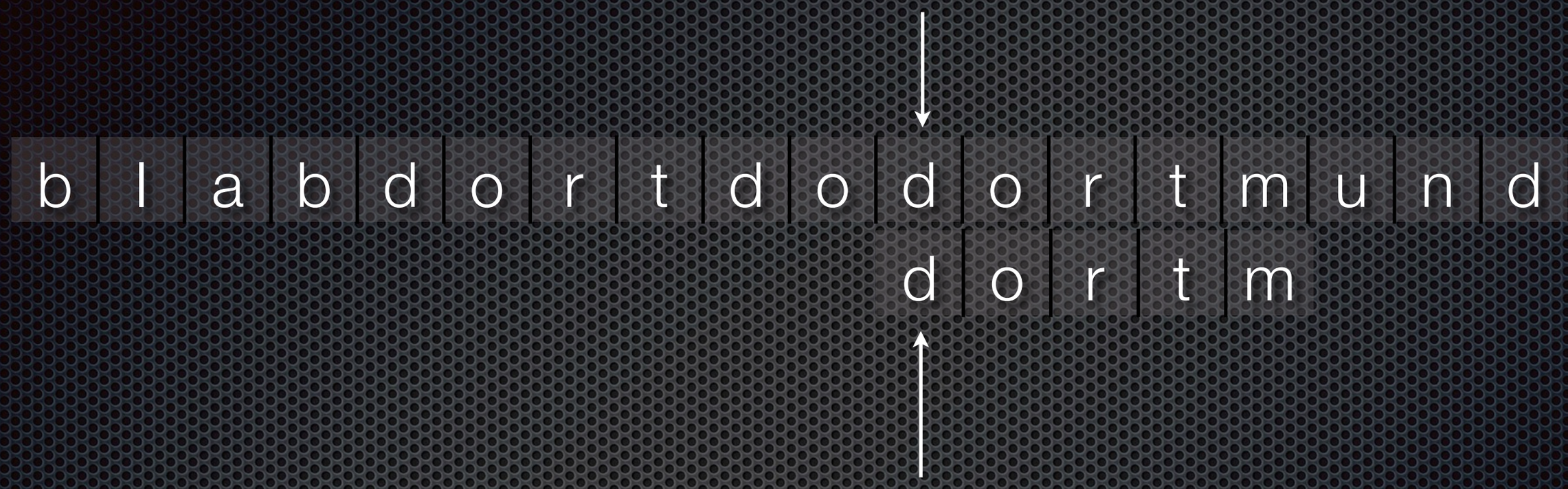
b l a b d o r t d o d o r t m u n d

d o r t m

b l a b d o r t d o d o r t m u n d

d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m


b l a b d o r t d o d o r t m u n d
d o r t m

A minta prefix függvénye PRE : {1,...,m} → {0,...,m–1} a következoképpen adható meg:

$$PRE(q) = \max\{k : k < q \wedge S_k \sqsupset S_q\} \quad q = 1, \ldots, m$$

```
Prefix(S) {
    m=hossz(S) ;
    pre[1]=0 ;
    k=0 ;
    for (q=2;q<=m;q++) {
        while (k>0 and S[k+1]!=S[q])
            k=PRE(k) ;
        if (S[k+1]==S[q]) k++ ;
        pre[q]=k ;
    }
    return pre ;
}
```

A Prefix eljárás futási ideje legrosszabb esetben O(m).

```
KMP(A,S) {
   n=hossz(A) ;
   m=hossz(S) ;
   pre=Prefix(S) ;
   q=0 ;
   for (i=1;i<=n;i++) {
      while (q>0 and S[q+1]!=A[i]) q:=pre[q] ;
      if (S[q+1]==A[i]) q++ ;
      if (q==m)  {
         illeszkedik(i-m+1) ;
         q:=pre[q] // következő illeszkedés keresése
      }
}
```

Rabin-Karp algoritmusa

Az algoritmus a Σ feletti szavakat, egy $|\Sigma| = d$ alapú számrendszerben felírt számként tekinti. Ekkor az S mintát egyértelműen megadja a hozzárendelt szám értéke, amit jelöljön s. Ez az s érték kiszámítható O(m) időben. Az algoritmus alapötlete, hogy az A szövegre, minden i-re kiszámolva az $A[i + 1,\ldots,i + m]$ számértéket, azon esetekben, ahol ez a szám s illeszkedik a minta. Ezt hatékonyan számíthatjuk, mivel $A[i+2,\ldots,i+m+1] = (A[i+1,\ldots,i+m]-d^m A[i+1])d + A[i+m+1]$, így az első érték után a többiek mindegyike konstans időben kiszámítható.

A módszerrel az a probléma, hogy a létrejövő számok túl nagyok így nem ábrázolhatóak a számítógépen a szokásos számtípusokkal.
A megoldás az, hogy a számok helyett csak egy megfelelően nagy, de még kezelhető q érték szerinti osztásmaradékukat használjuk.

Természetesen így egy olyan algoritmust kapunk, amely esetén a minta illeszkedik válasz hamis lehet, ha a két szám maradéka megegyezik.
Ezért a maradékosztályok megegyezése esetén, a helyességet ellenőrizni kell.

Viszont az algoritmus gyors elutasítási heurisztikaként jól használható, mivel ha azt kapjuk, hogy a minta nem illeszkedik, akkor valóban nincs illeszkedés.

Geometriai algoritmusok

pont helyzete, konvex burok, ponthalmaz legközelebbi és legtávolabbi pontpárja, metsző szakaszpárok keresése

Pont

$$p = (x, y)$$

$$x, y \in R$$

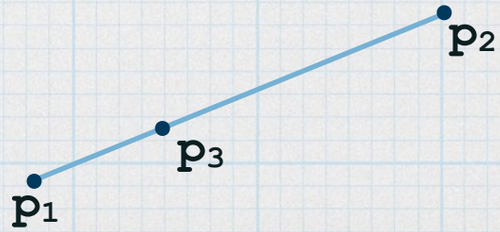
```
class pont {
   double x,y ;
   pont(double ix,iy) {
      x=ix ;
      y=iy ;
   }
}
pont p=new pont(5.2,7.47) ;
```

$p_1 = (x_1, y_1)$ $p_2 = (x_2, y_2)$ pontok

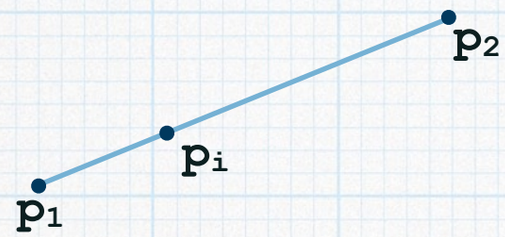
konvex kombinációja $p_3 = (x_3, y_3)$ pont, ha

$$\exists \alpha \in R : 0 \le \alpha \le 1, x_3 = \alpha x_1 + (1-\alpha) x_2 \wedge y_3 = \alpha y_1 + (1-\alpha) y_2$$

azaz $p_3 = \alpha p_1 + (1-\alpha) p_2$



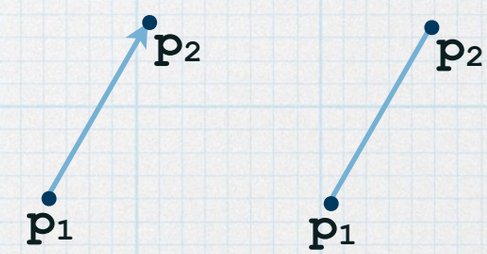
Szakasz



Két különböző p_1 és p_2 pont esetén $\overline{p_1p_2}$ szakasz p_1 és p_2 konvex kombinációinak halmaza.

p_1 és p_2 pontokat a $\overline{p_1p_2}$ szakasz végpontjainak nevezzük.

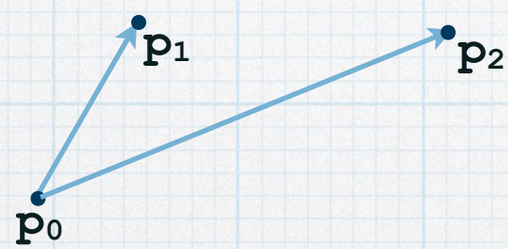
Néha p_1 p_2 sorrendje is számít, ilyenkor $\overrightarrow{p_1p_2}$. irányított szakaszról beszélünk.



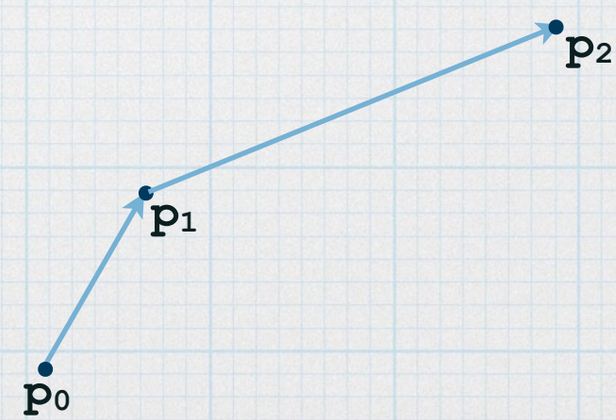

```
class szakasz {
   pont p1,p2 ;
   szakasz(pont ip1,ip2) {
      p1=ip1;
      p2=ip2 ;
   }
}
szaksz s=new szakasz(p15,p26) ;
```

Forgásirány

1. Adott két irányított szakasz: $\overrightarrow{p_0p_1}$ és $\overrightarrow{p_0p_2}$.
   A közös p_0 végpont körül $\overrightarrow{p_0p_1}$-hez viszonyítva p_0p_2 jobbra (🕑), vagy balra fordul?



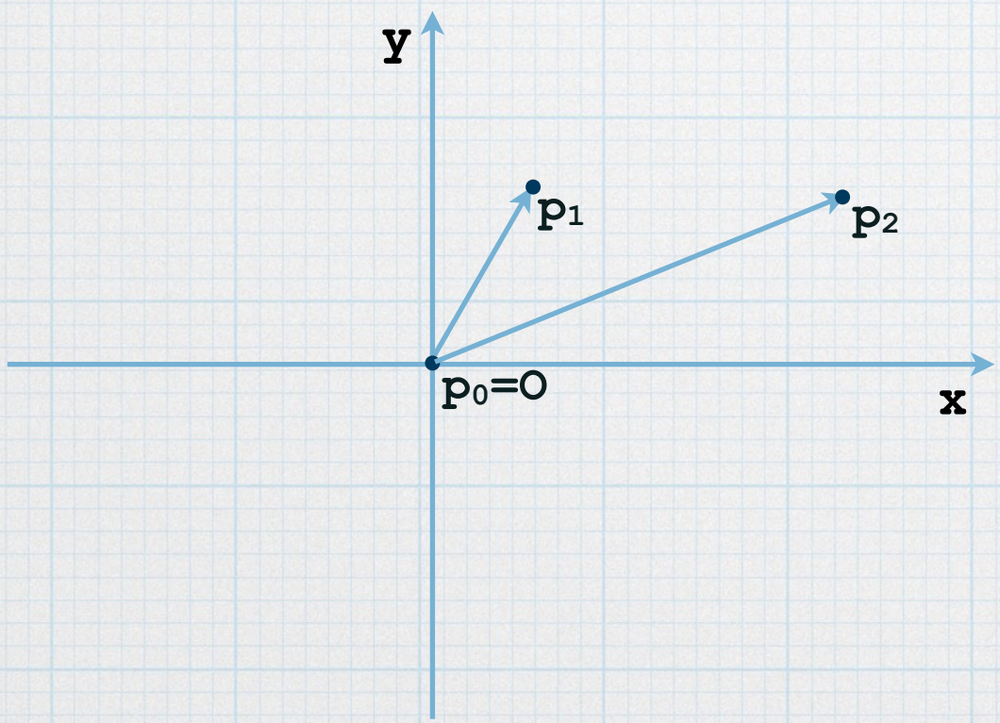
2. Adott $\overrightarrow{p_0p_1}$ és $\overrightarrow{p_1p_2}$ szakasz.
   Ha folyamatosan bejárjuk $\overrightarrow{p_0p_1}$-et majd $\overrightarrow{p_1p_2}$-t, balra fordulunk-e a p_1 pontban?

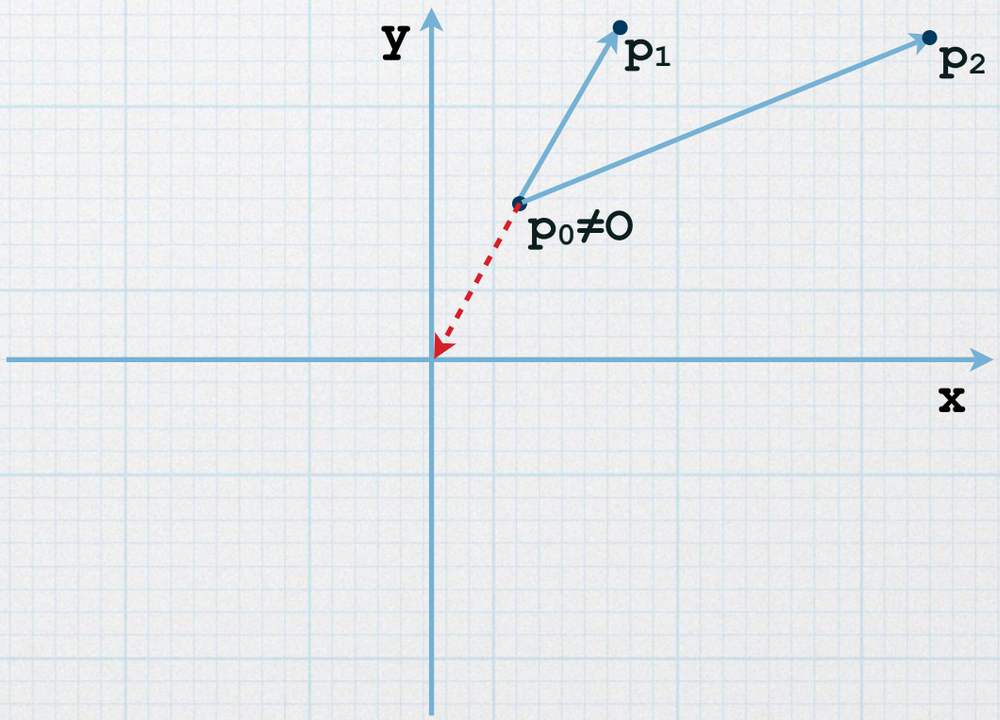


1. Forgásirány

$$\overrightarrow{p_1} \times \overrightarrow{p_2} = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$$

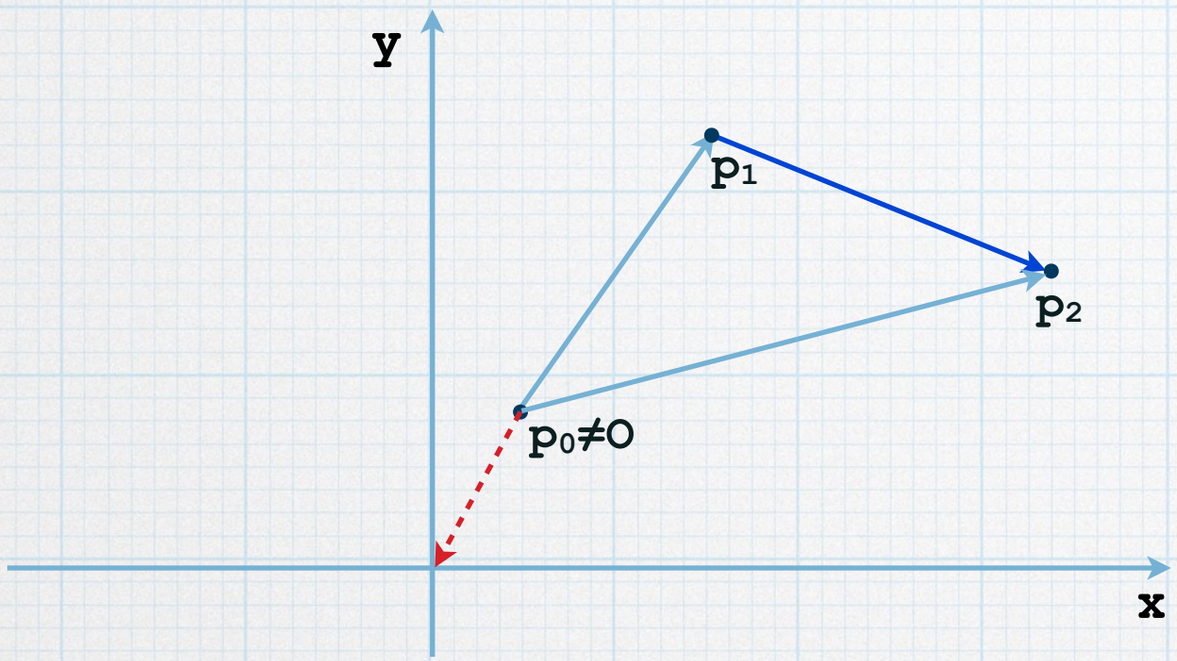
Ha $\overrightarrow{p_1} \times \overrightarrow{p_2}$ pozitív, $\overrightarrow{p_1}$ jobbra fordul $\overrightarrow{p_2}$-höz képest.





$$f = (p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

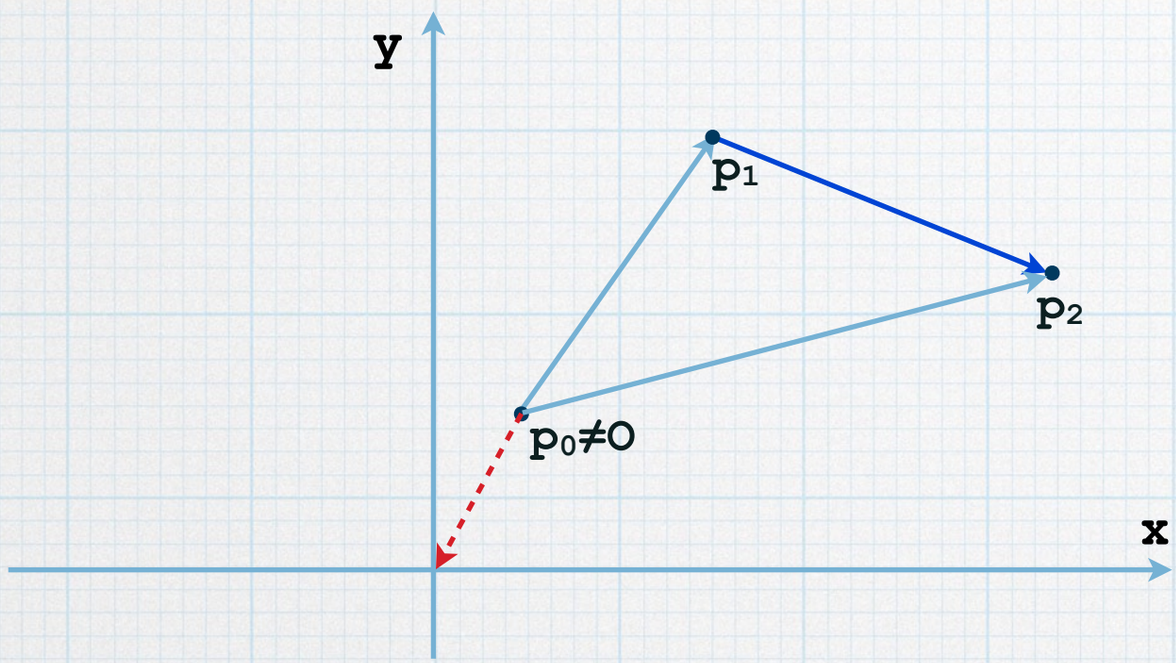
2. Forgásirány

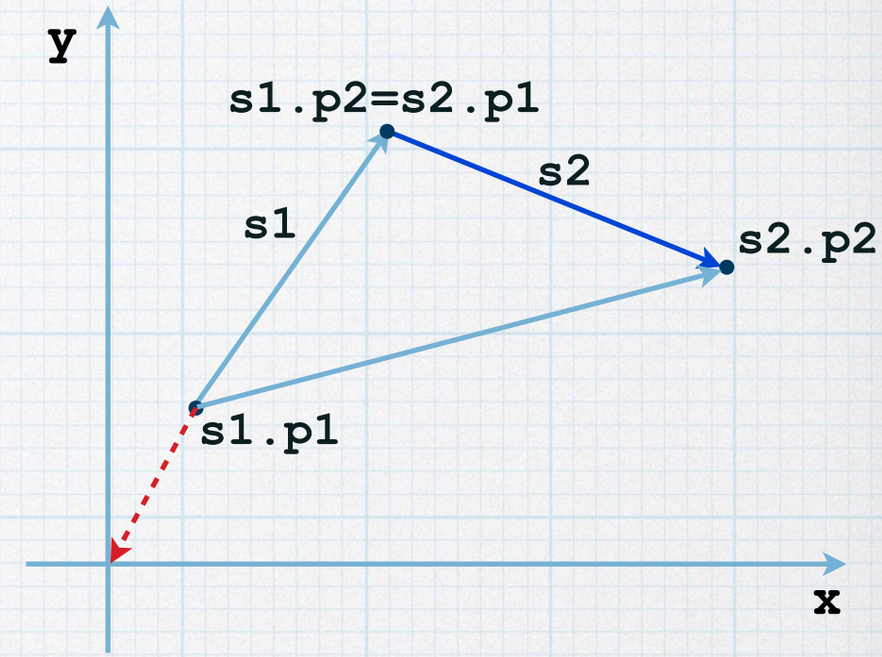


$$f = (p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

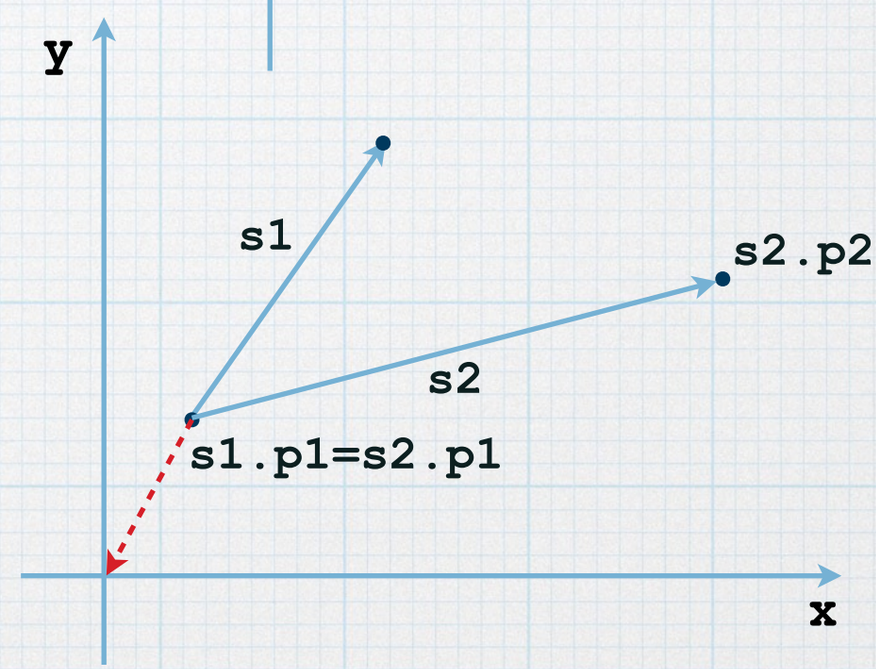
f>0⇒balra fordul
f<0⇒jobbra fordul
f=0⇒egy egyenesbe esik (nem fordul)

Forgásirány





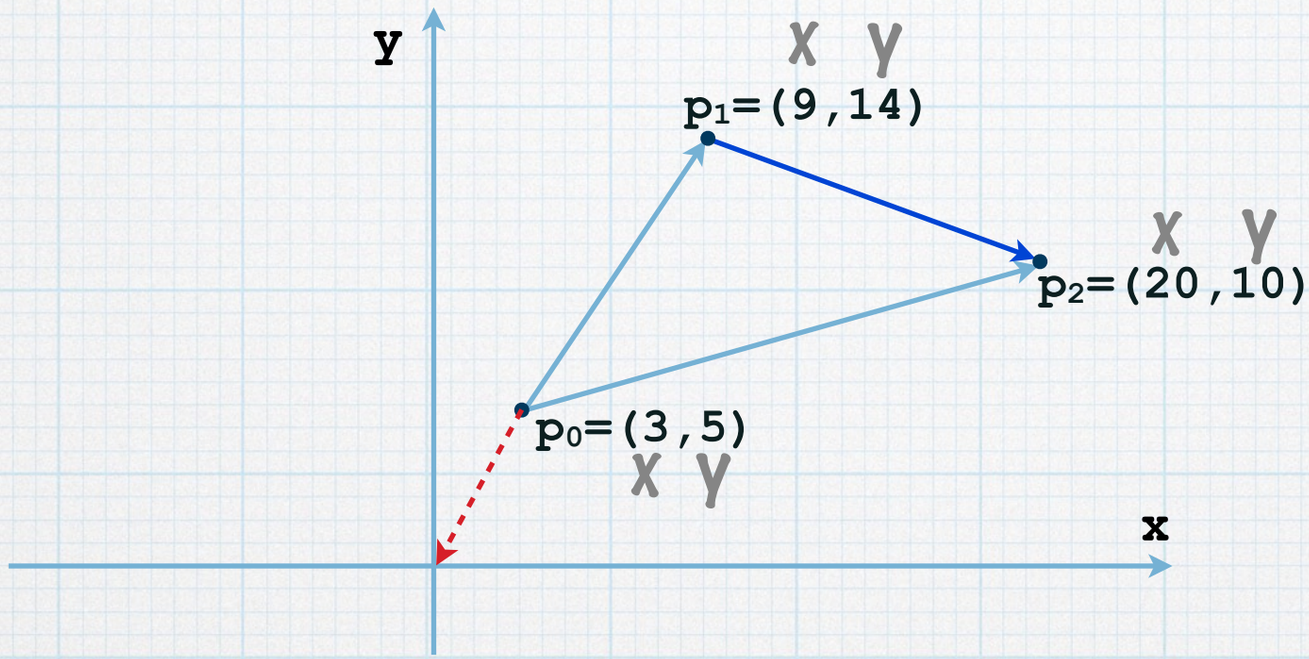
$$f=(p_1-p_0)\times(p_2-p_0)=(x_1-x_0)(y_2-y_0)-(x_2-x_0)(y_1-y_0)$$




```
forgasirany(pont p0,p1,p2) {
   f=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y) ;
   if (f>0) return 1 ;
   if (f<0) retun -1;
   retun 0;
}
```

Forgásirány

példa:




```
forgasirany(pont p0,p1,p2) {
   f=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y) ;
```

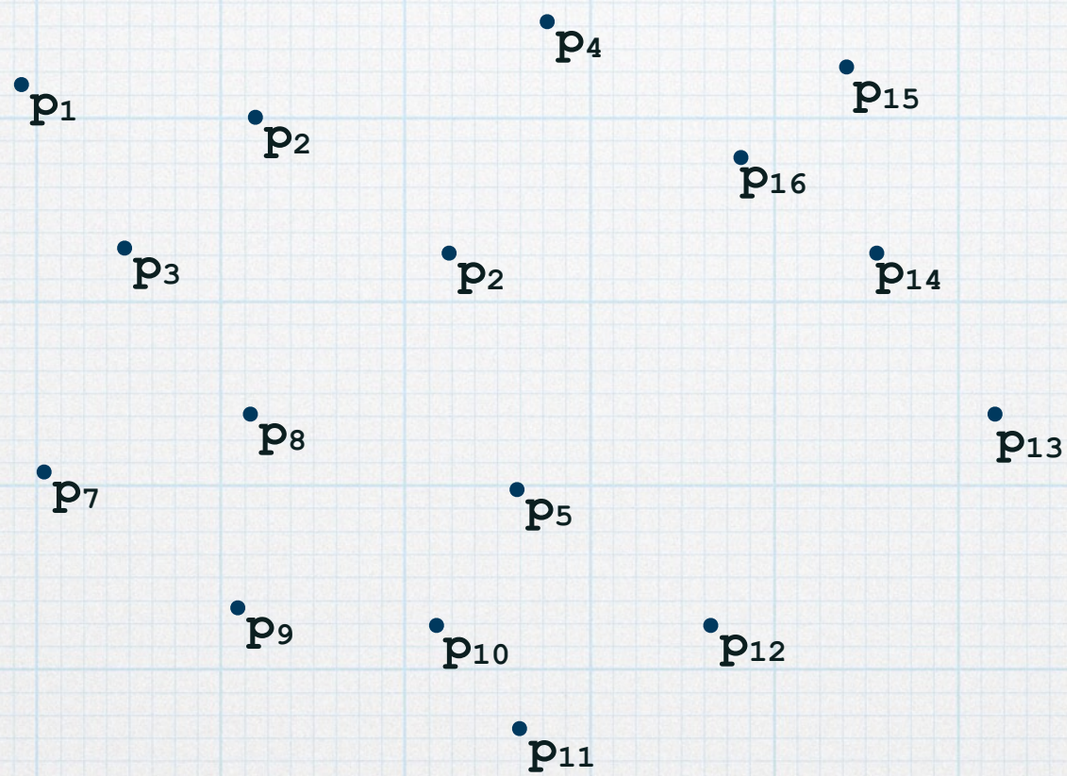
9 3 10 5 20 3 14 5

6 5 17 9

30 153 -123

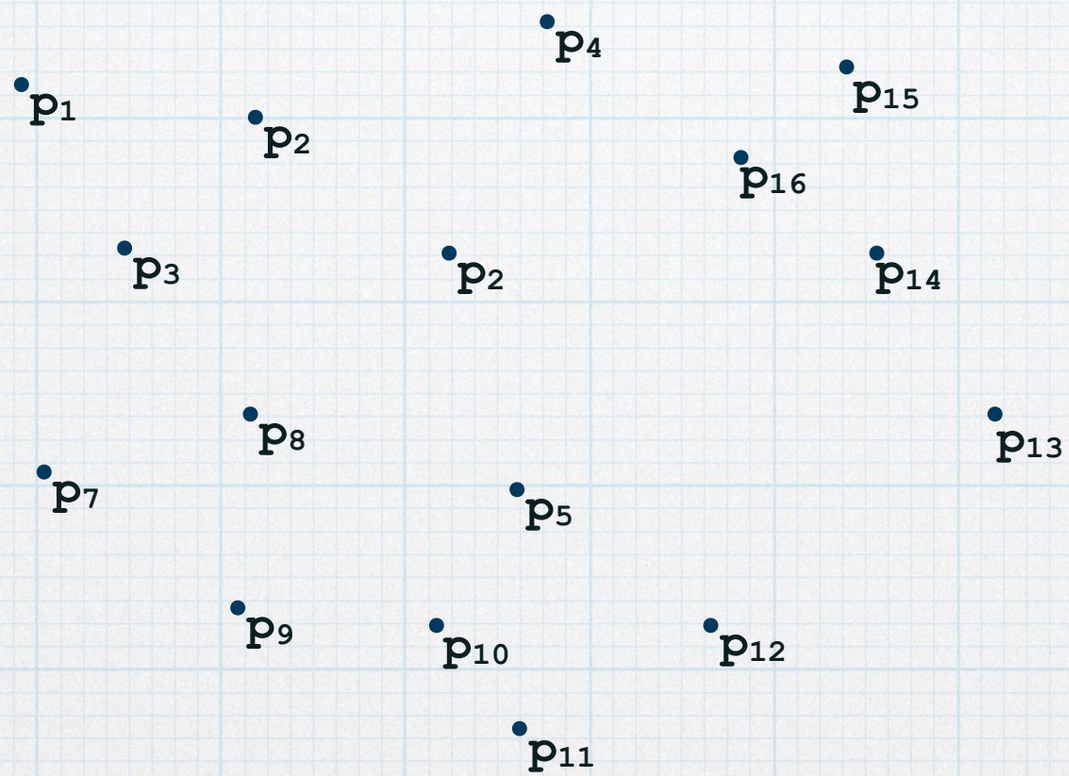
```
   if (f>0) return 1 ; // balra fordul
   if (f<0) retun -1; // jobbra fordul
   retun 0;
}
```

Adott pontok egy $\{p_1, \ldots, p_n\}$ halmaza.



$\{p_1, \ldots, p_n\}$ ponthalmaz polárszög szerinti rendezése

1. Válasszuk ki a legkisebb x koordinátájú pontot, ha több ilyen van, akkor válasszuk ezek közül a legkisebb y koordinátájút és jelüljük **q**-val.

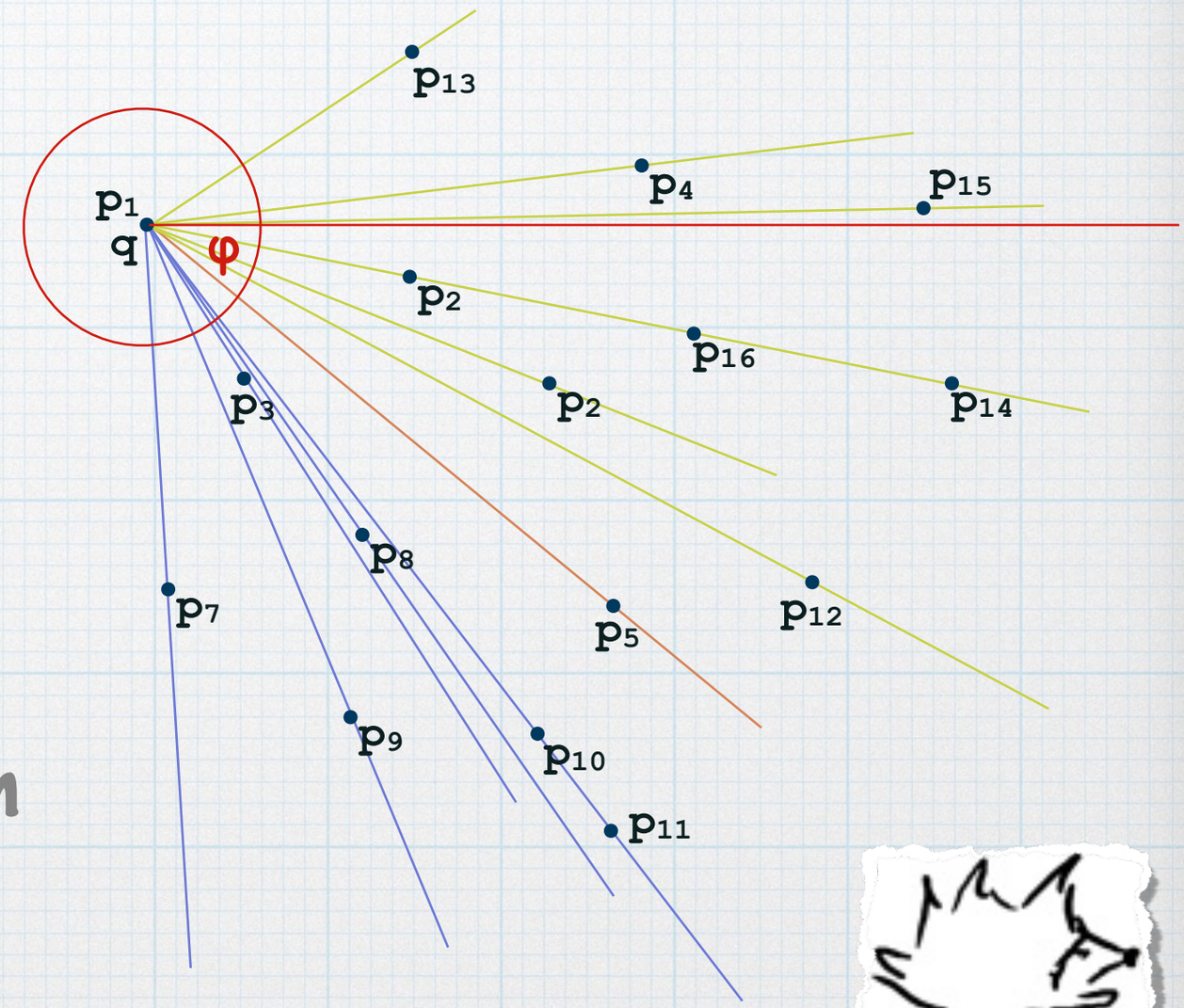


$\{p_1,...,p_n\}$ ponthalmaz polárszög szerinti rendezése

1. Válasszuk ki a legkisebb x koordinátájú pontot, ha több ilyen van, akkor válasszuk ezek közül a legkisebb y koordinátájút és jelüljük **q**-val.

2. Húzzunk félegyeneseket a **q**-ból minden ponthoz.

   Rendezzük az egyeneseket a **q** ponton áthaladó, x-tengellyel párhuzamos egyenessel bezárt (előjeles) szög alapján. $\varphi(q,p)$



3. Rendezzük a pontokat úgy, hogy **q** legyen az első és **p** előbb legyen mint **r** ha:
   $\varphi(q,p)<\varphi(q,r)$ vagy
   $\varphi(q,p)=\varphi(q,r)$ és **p** közelebb van **q**-hoz mint **r**.

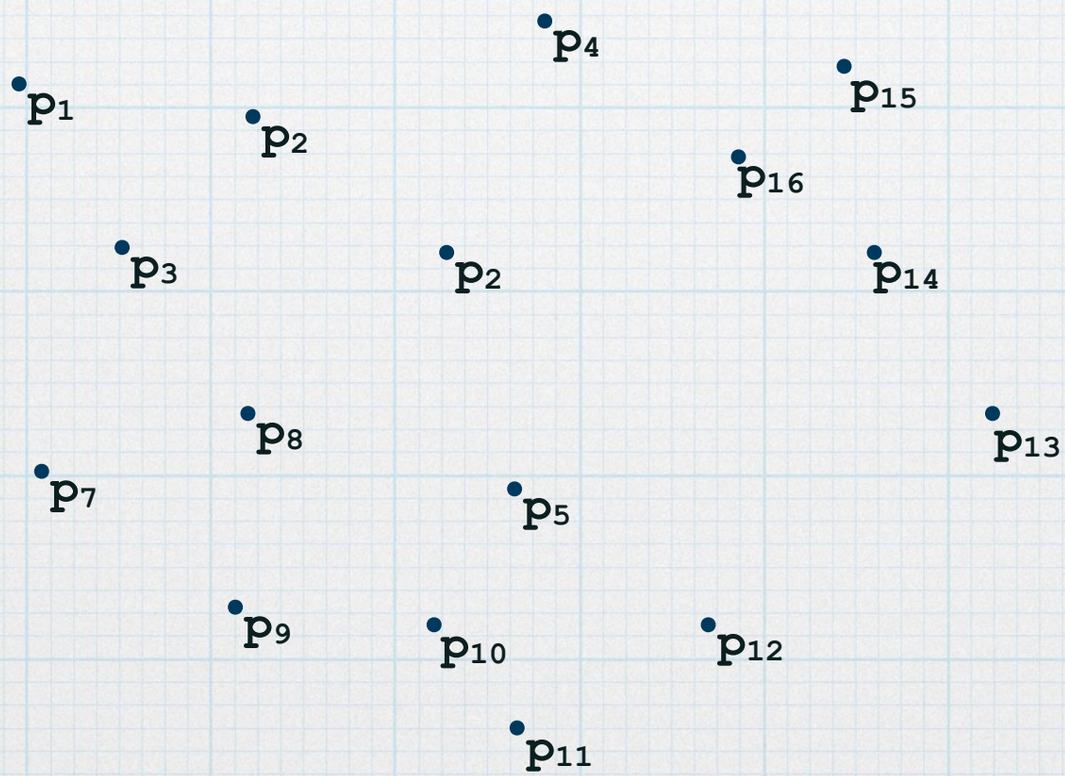
$p<r \iff$ `forgasirany(q,p,r)==1`

O(nlogn)

Feladat

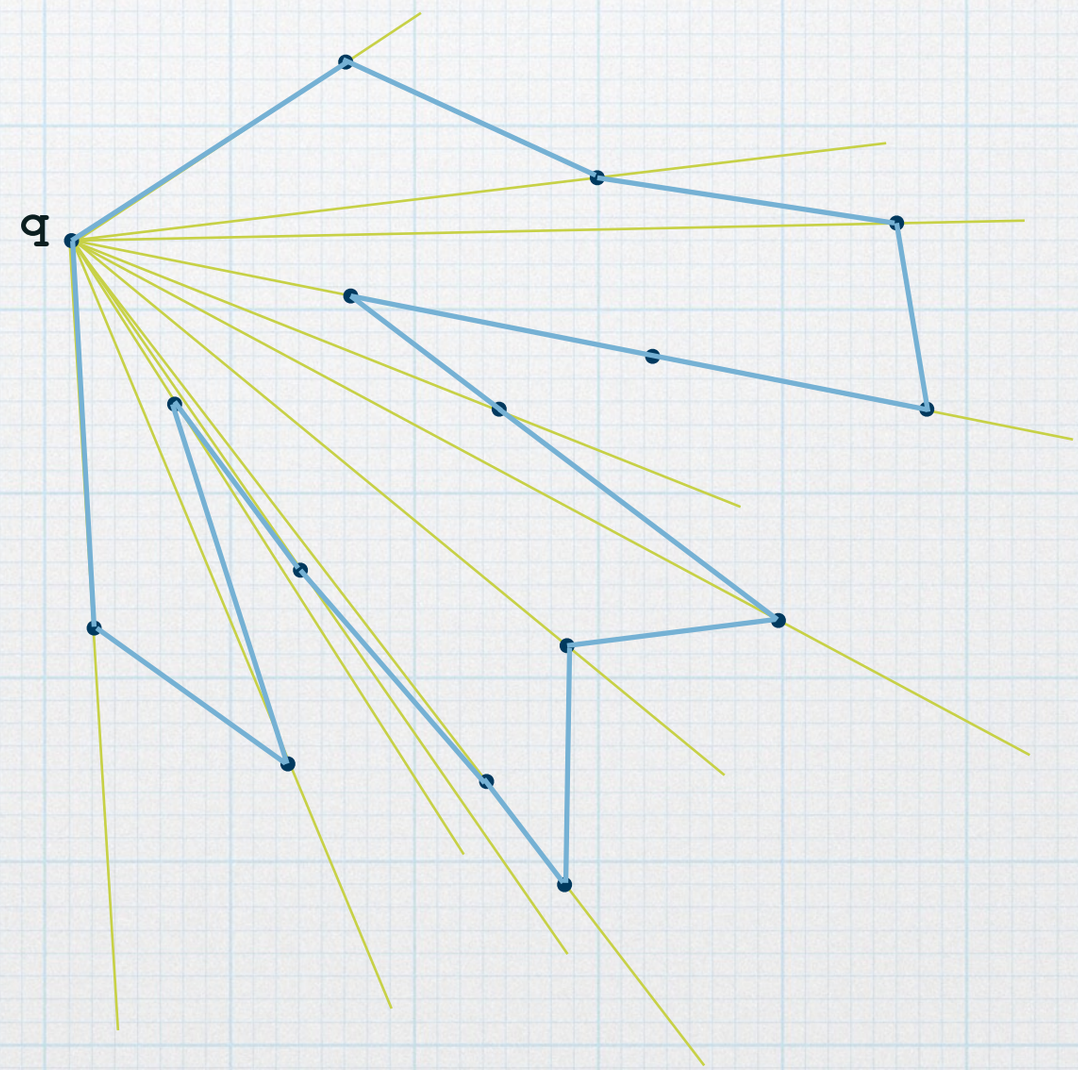
Pontok összekötése zárt nem metsző poligonná

Adott $\{p_1, \ldots, p_n\}$ ponthalmaz.
Kössük össze $p_1, \ldots, p_n$ pontokat zárt, nem metsző poligonná!



Megoldás

1. A ponthalmaz polárszög szerinti rendezése.
2. Kössük össze a pontokat a rendezés szerinti sorrendben.



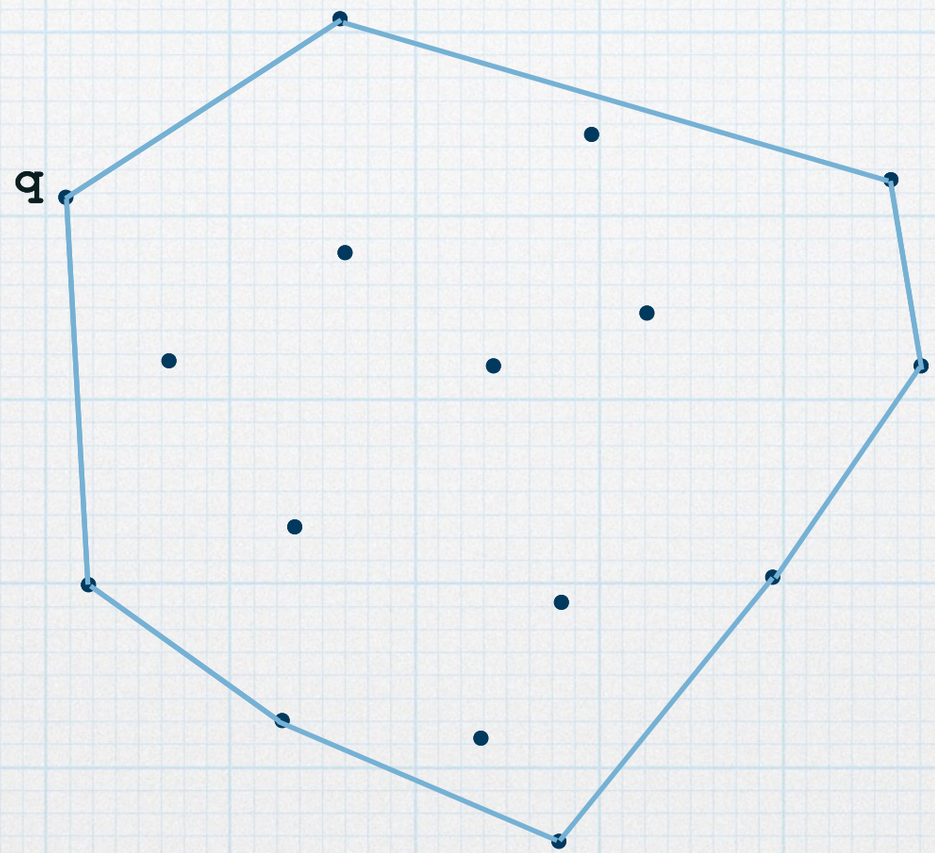
Feladat

Konvex burok meghatározása

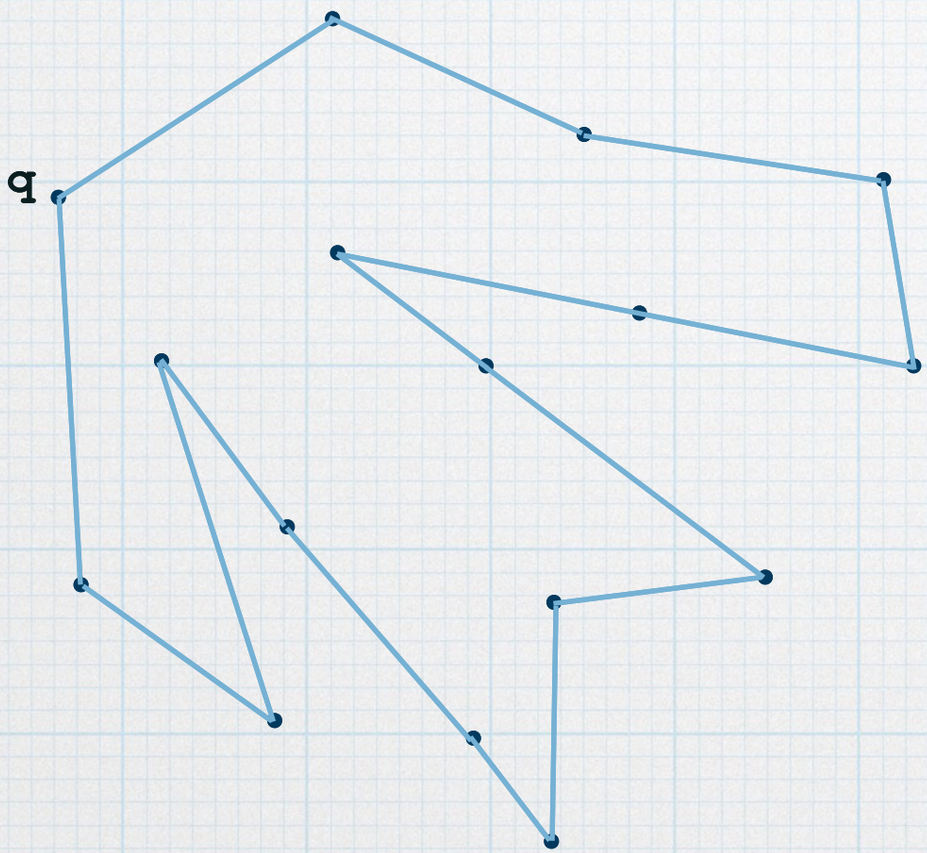
Adott $\{p_1, \ldots, p_n\}$ ponthalmaz.
Határozzuk meg $p_1, \ldots, p_n$ ponthalmaz konvex burkát!

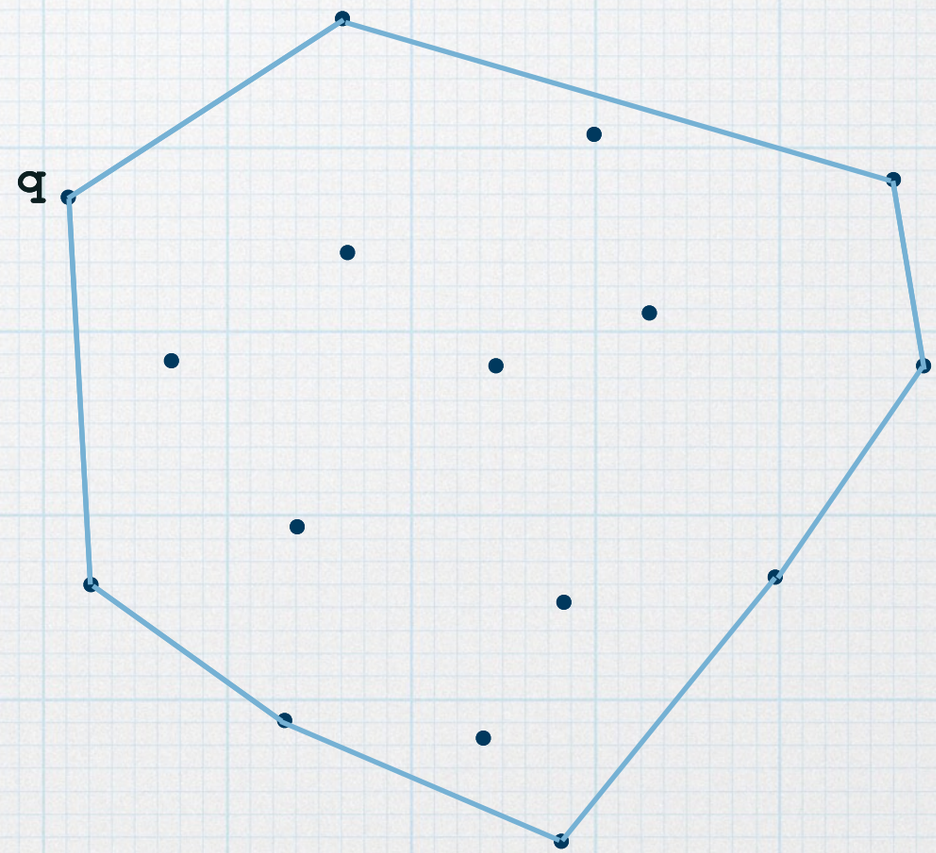
Definíció:
Egy egyszerű (nem metsző) poligon konvex, ha bármely két belső pontját összekötő szakasz minden pontja a poligon belsejében, vagy határán van.

Definíció:
Egy $P=\{p_1, \ldots, p_n\}$ ponthalmaz konvex burka az a legszűkebb Q konvex poligon, amely a ponthalmaz minden pontját tartalmazza.


q

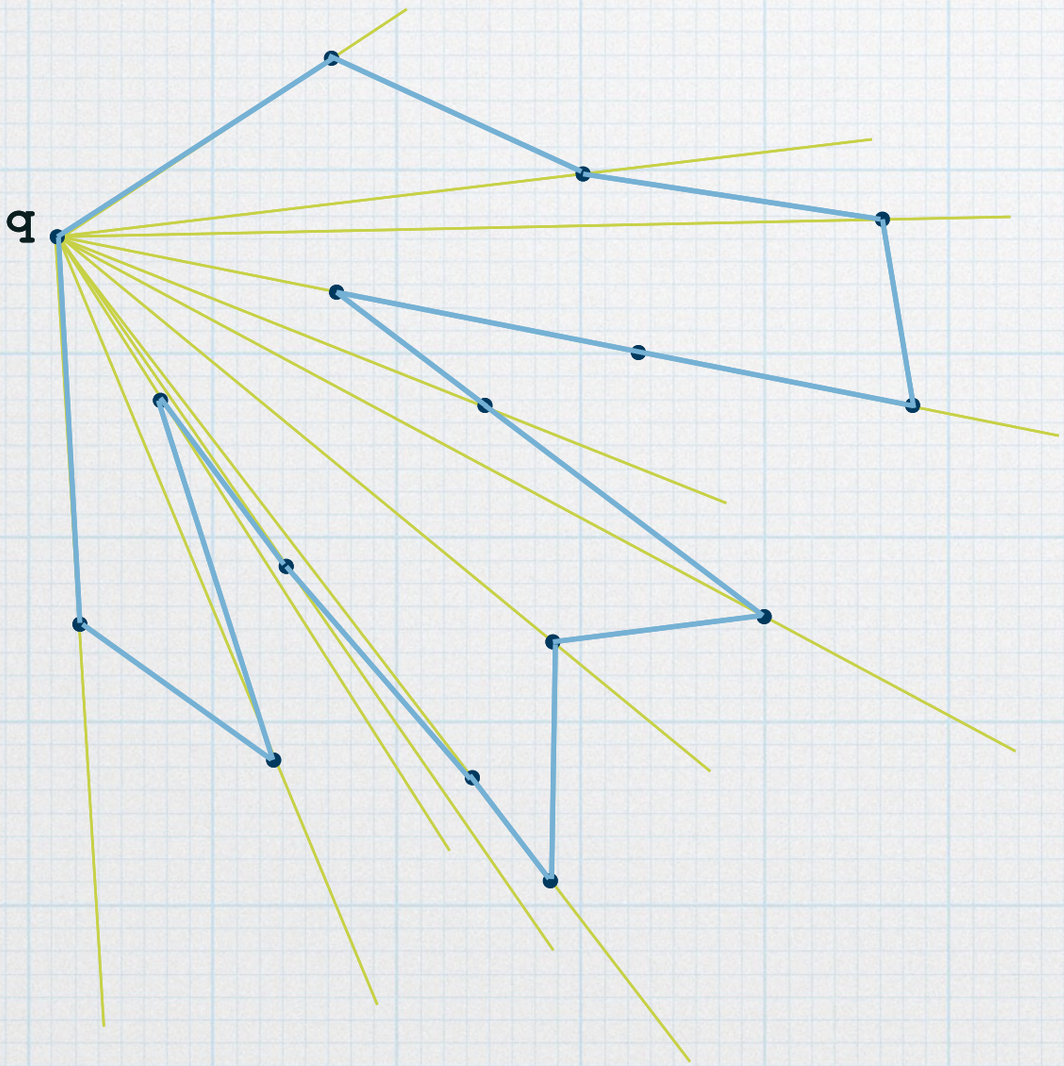
Megoldás

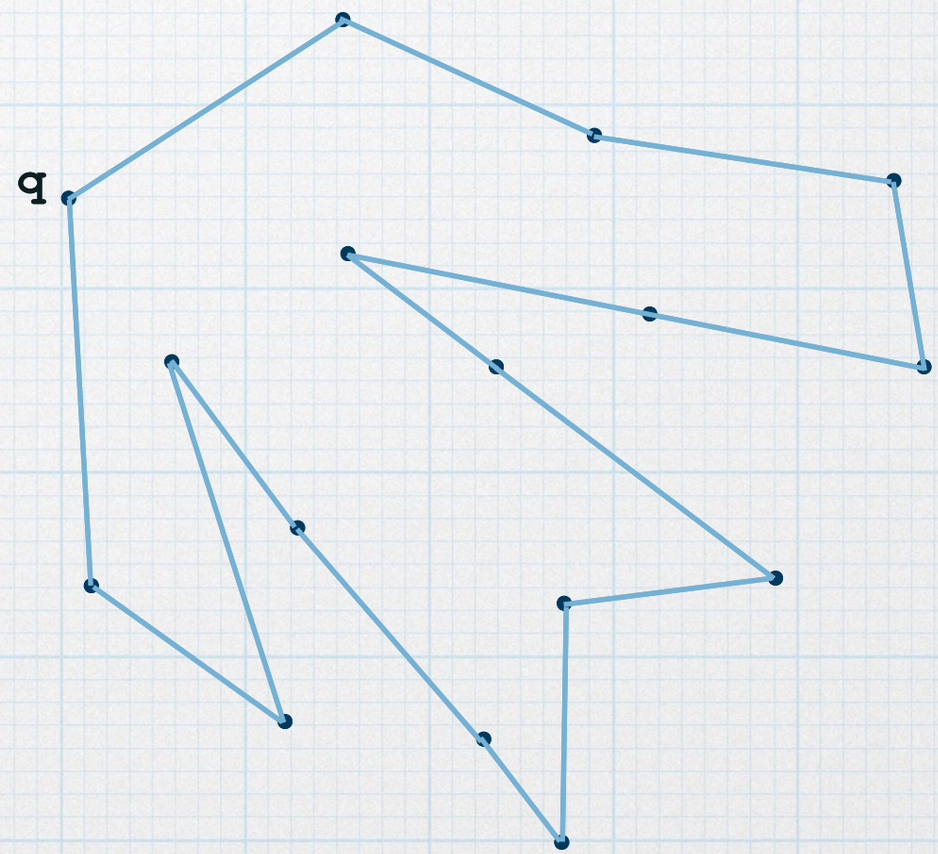




Megoldás

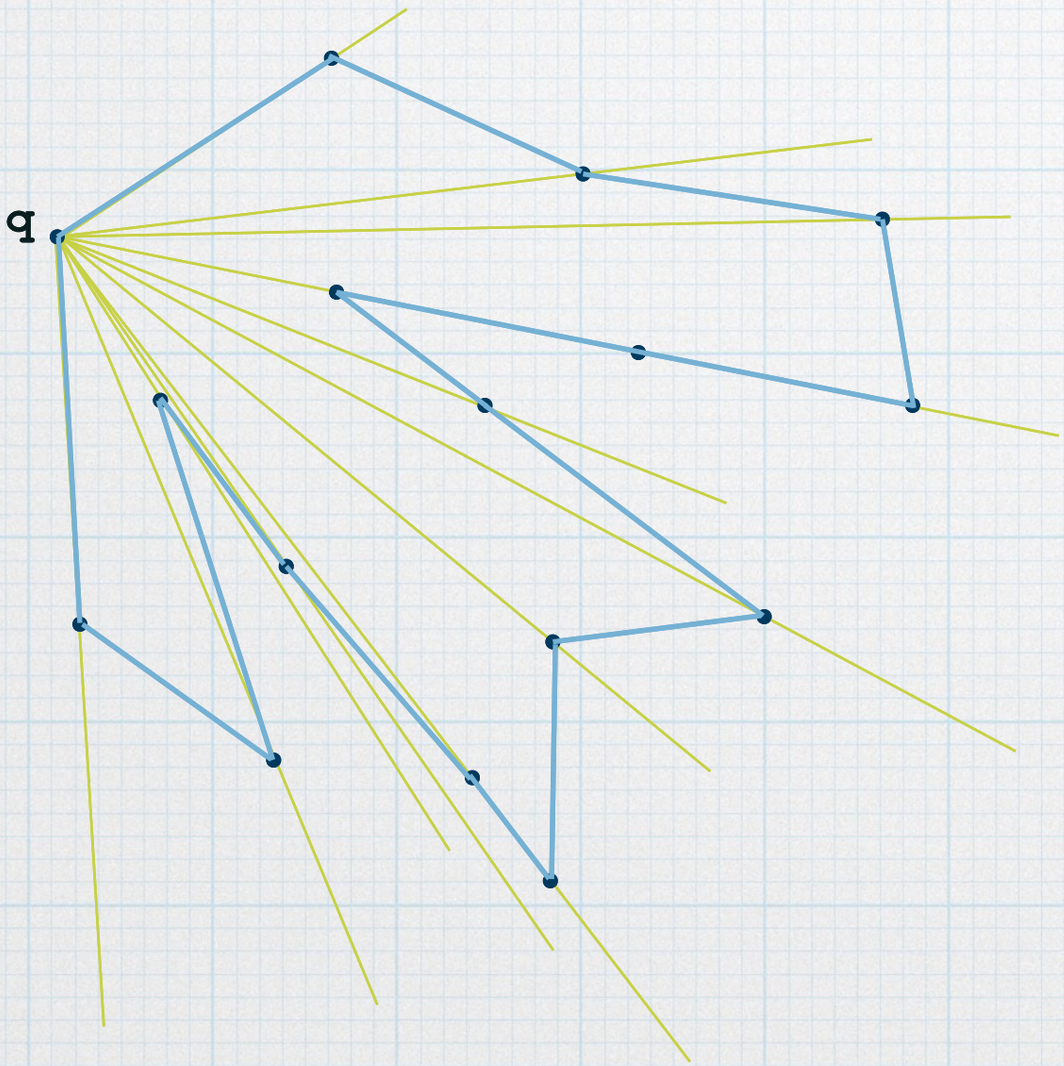
1. a ponthalmaz polárszög szerinti rendezése.

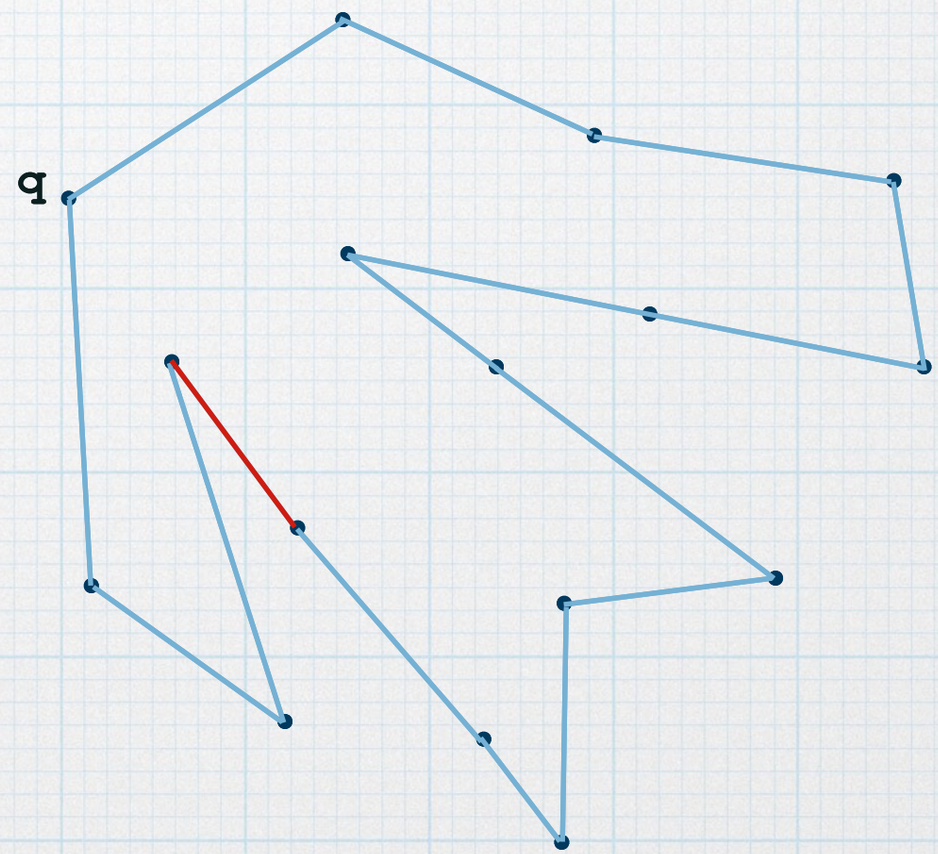




Megoldás

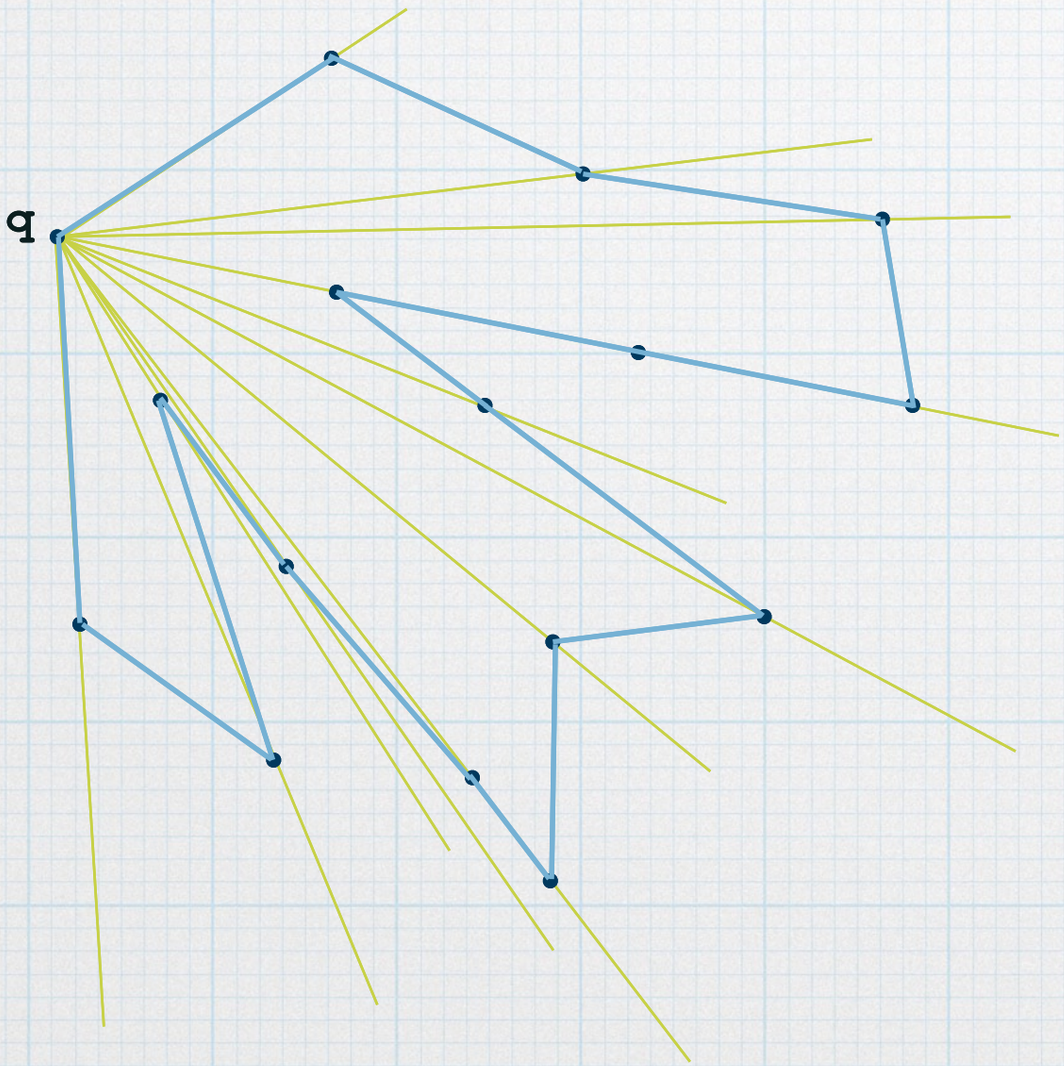
1. a ponthalmaz polárszög szerinti rendezése.

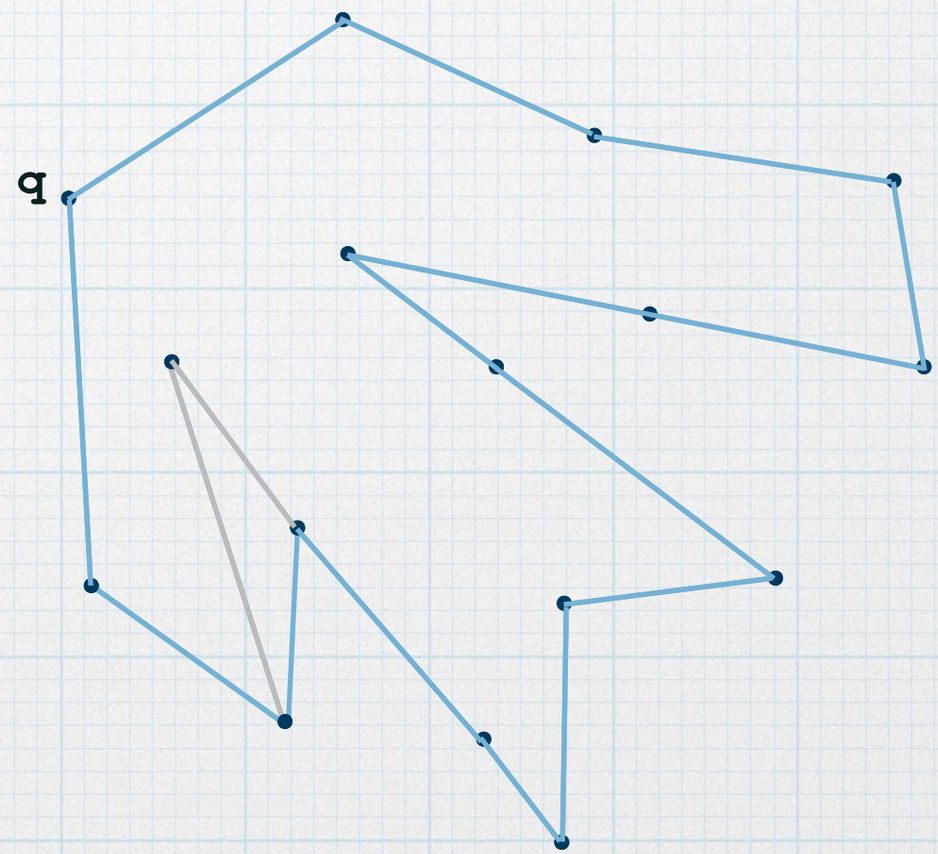




Megoldás

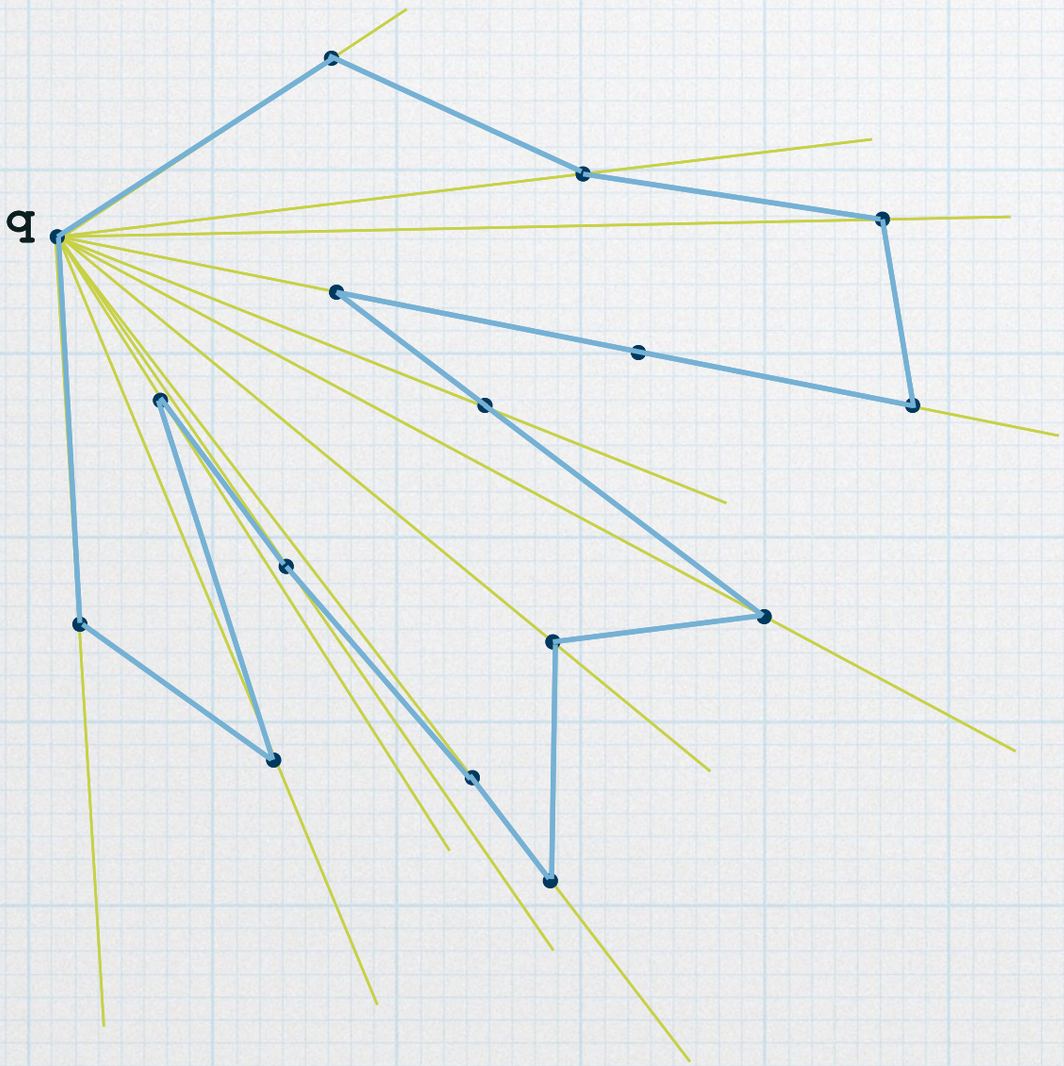
1. a ponthalmaz polárszög szerinti rendezése.

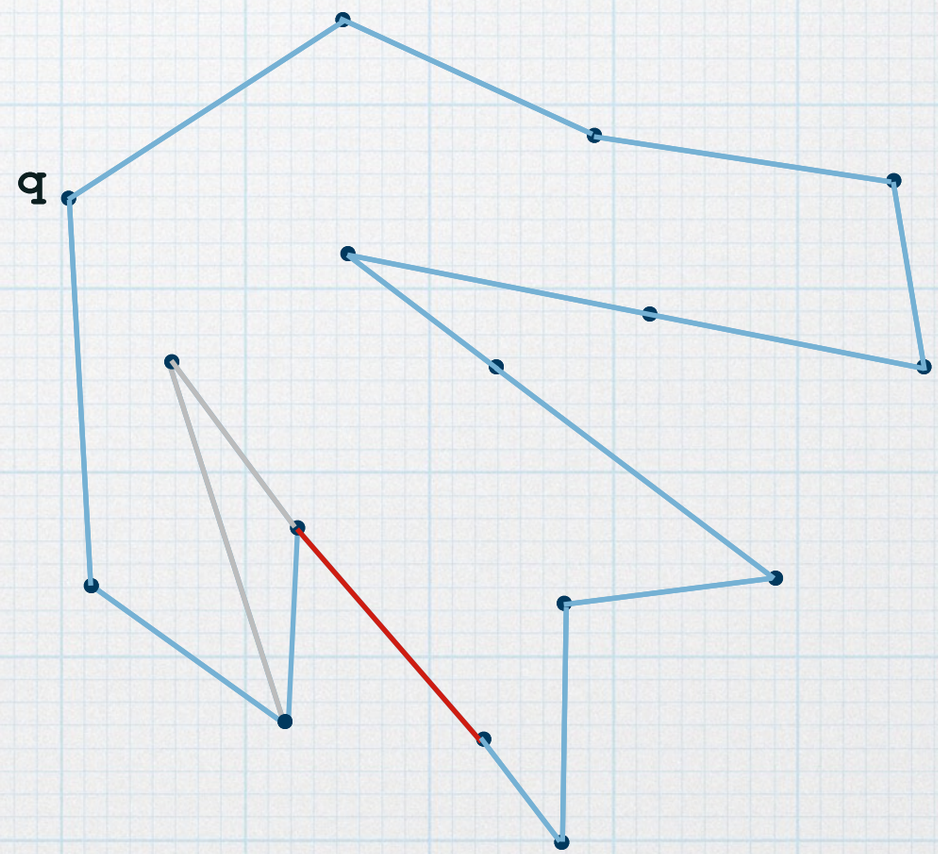




Megoldás

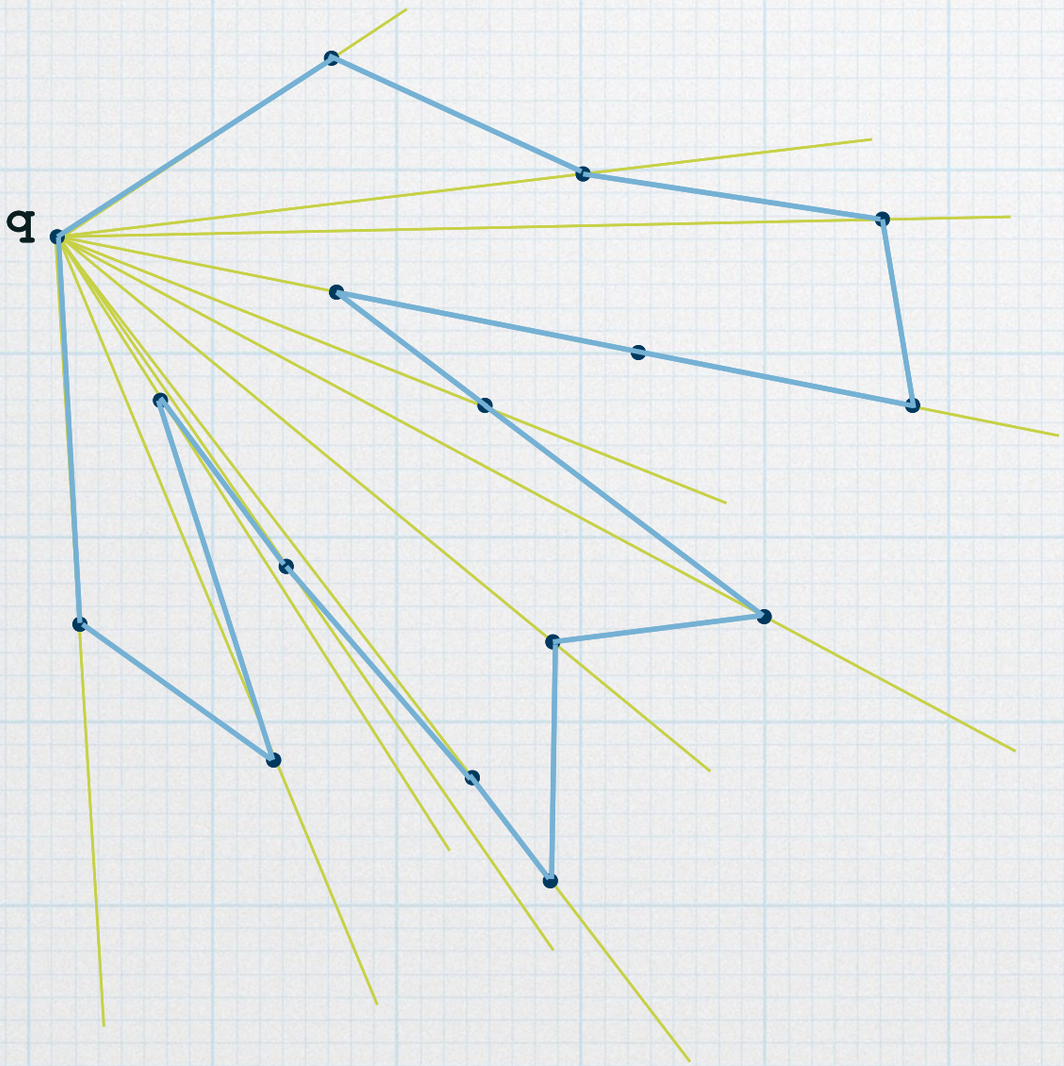
1. a ponthalmaz polárszög szerinti rendezése.

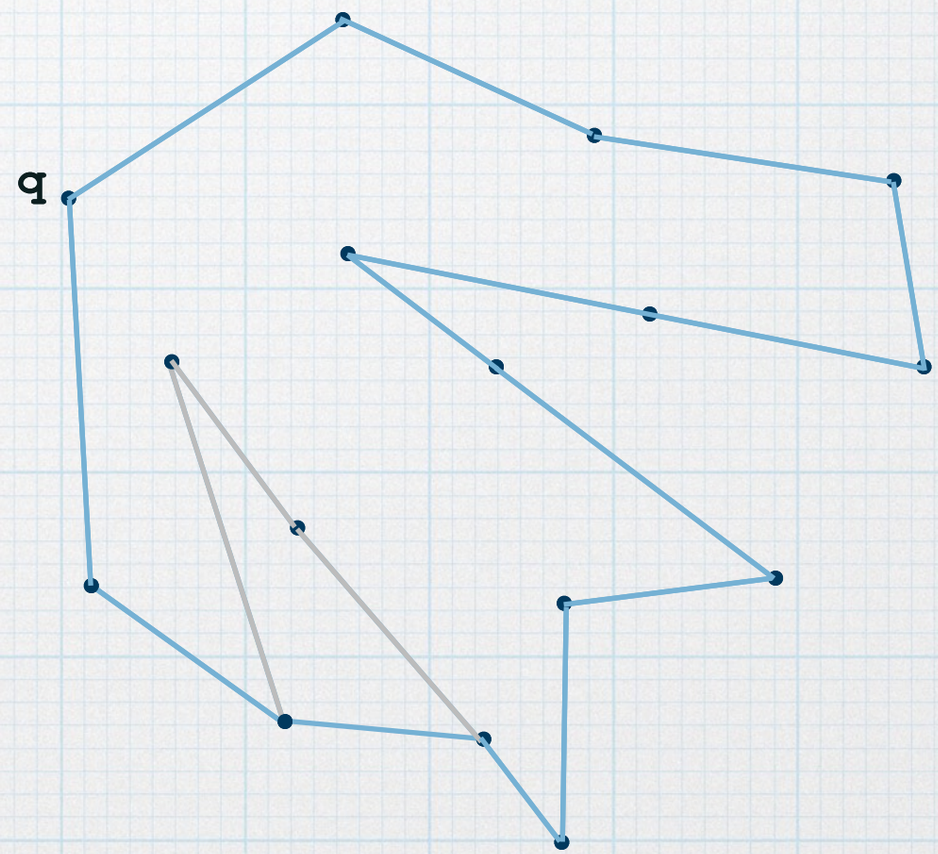




Megoldás

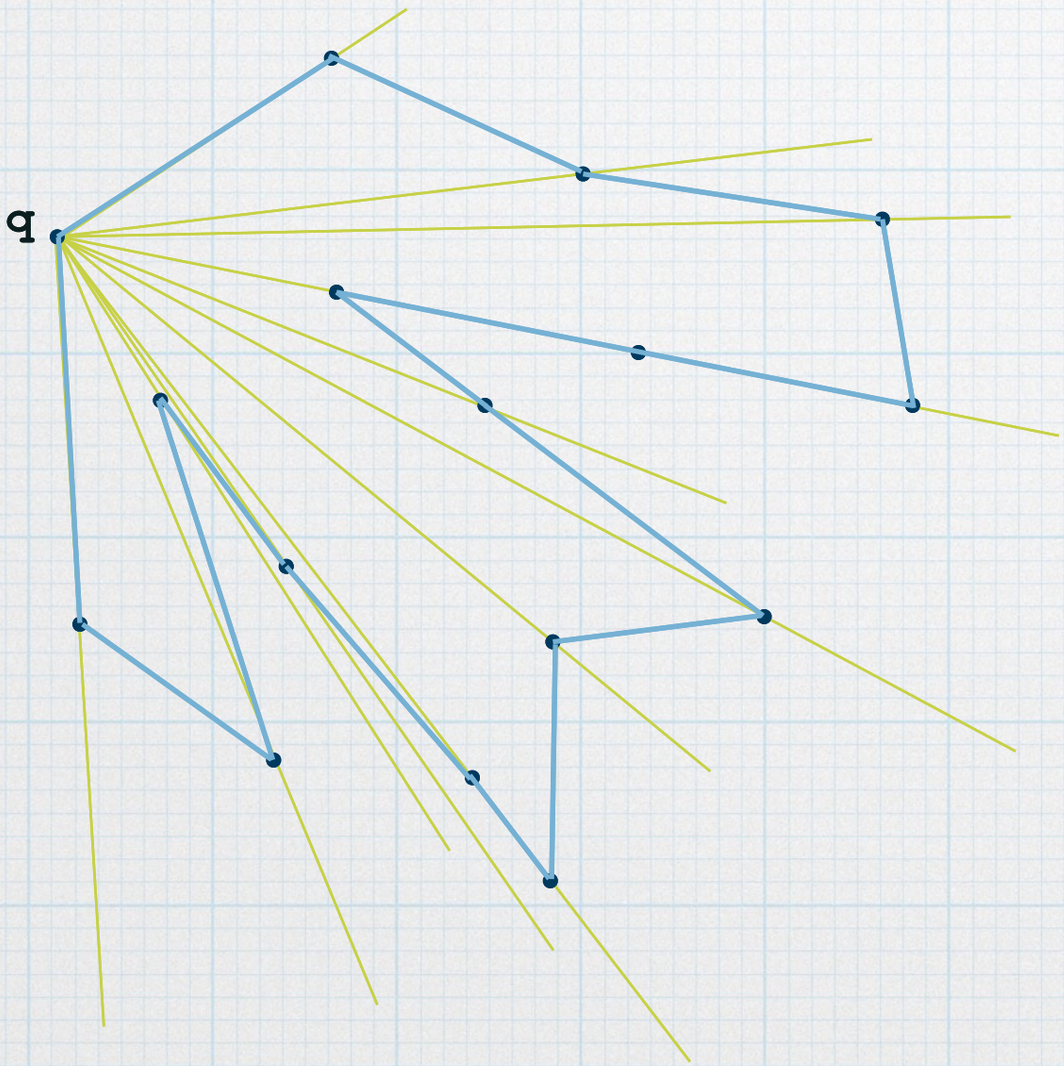
1. a ponthalmaz polárszög szerinti rendezése.

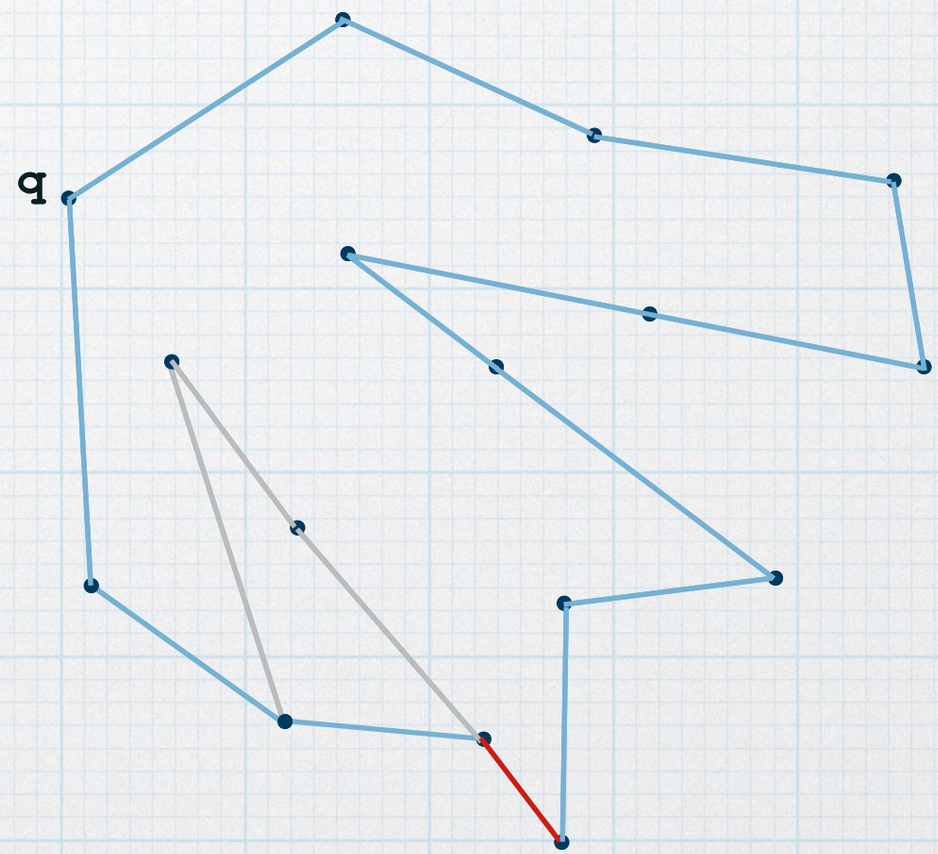




Megoldás

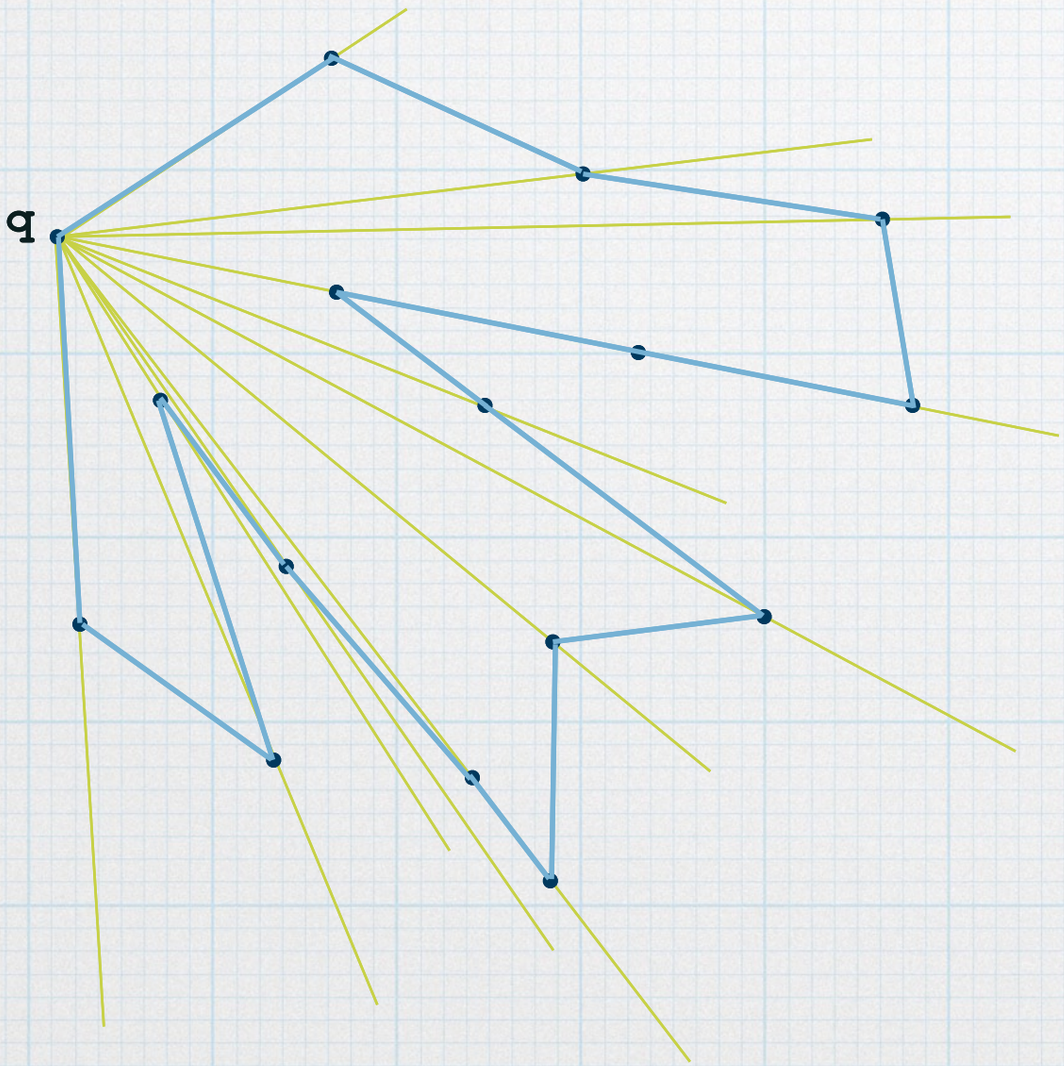
1. a ponthalmaz polárszög szerinti rendezése.

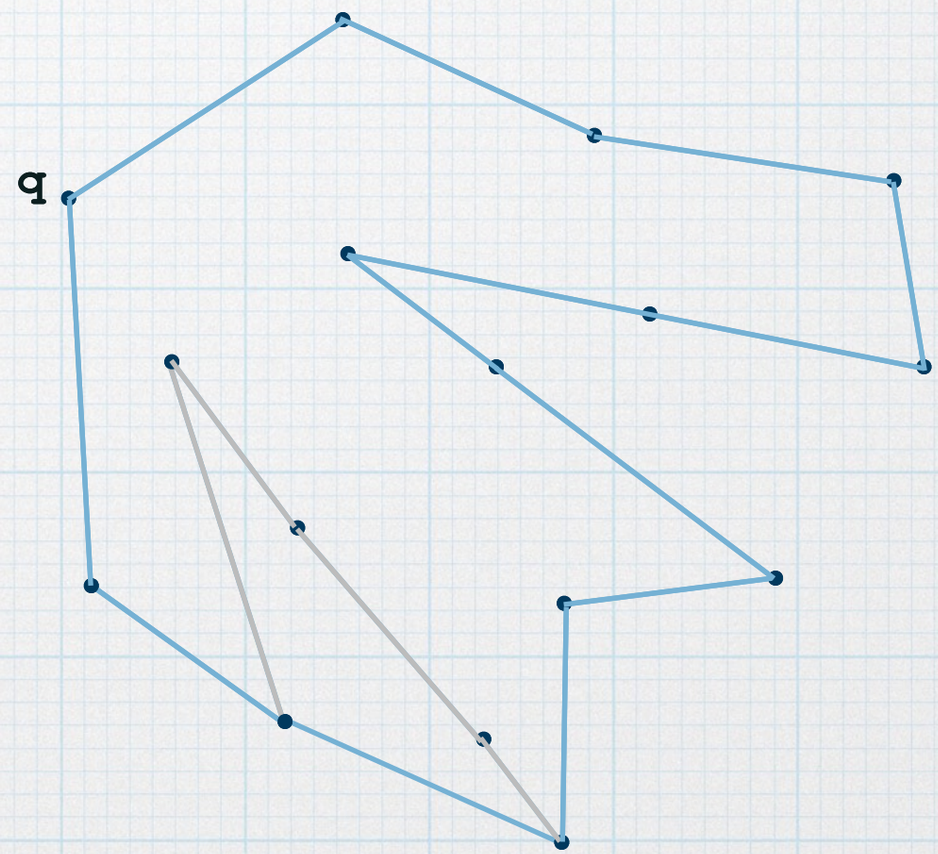




Megoldás

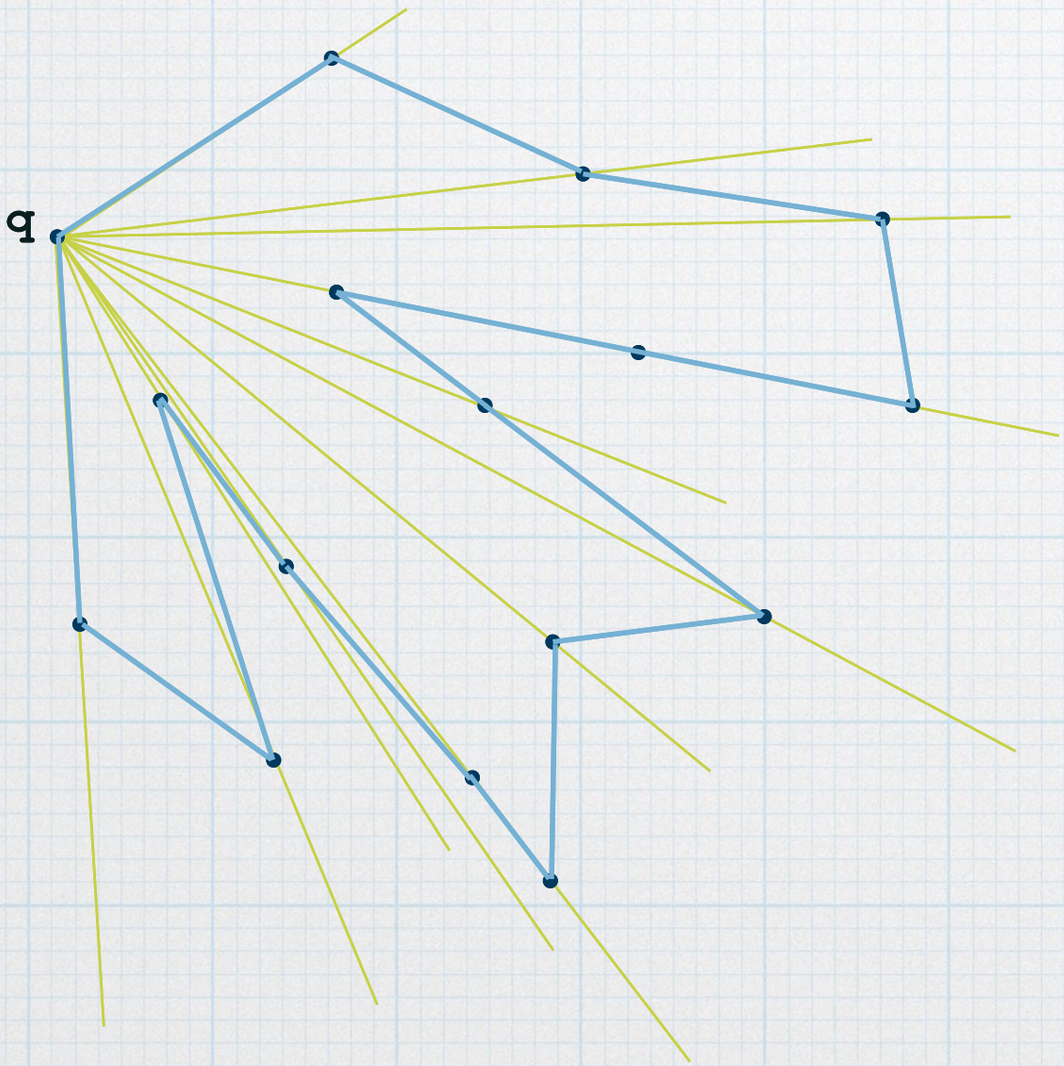
1. a ponthalmaz polárszög szerinti rendezése.

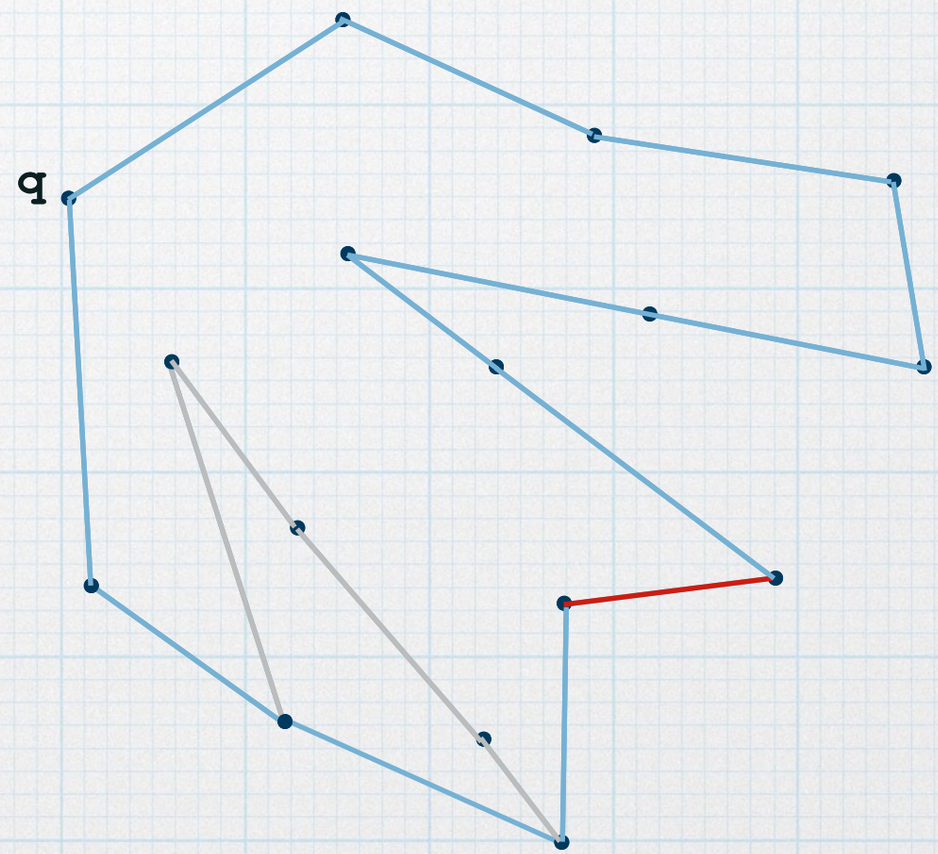




Megoldás

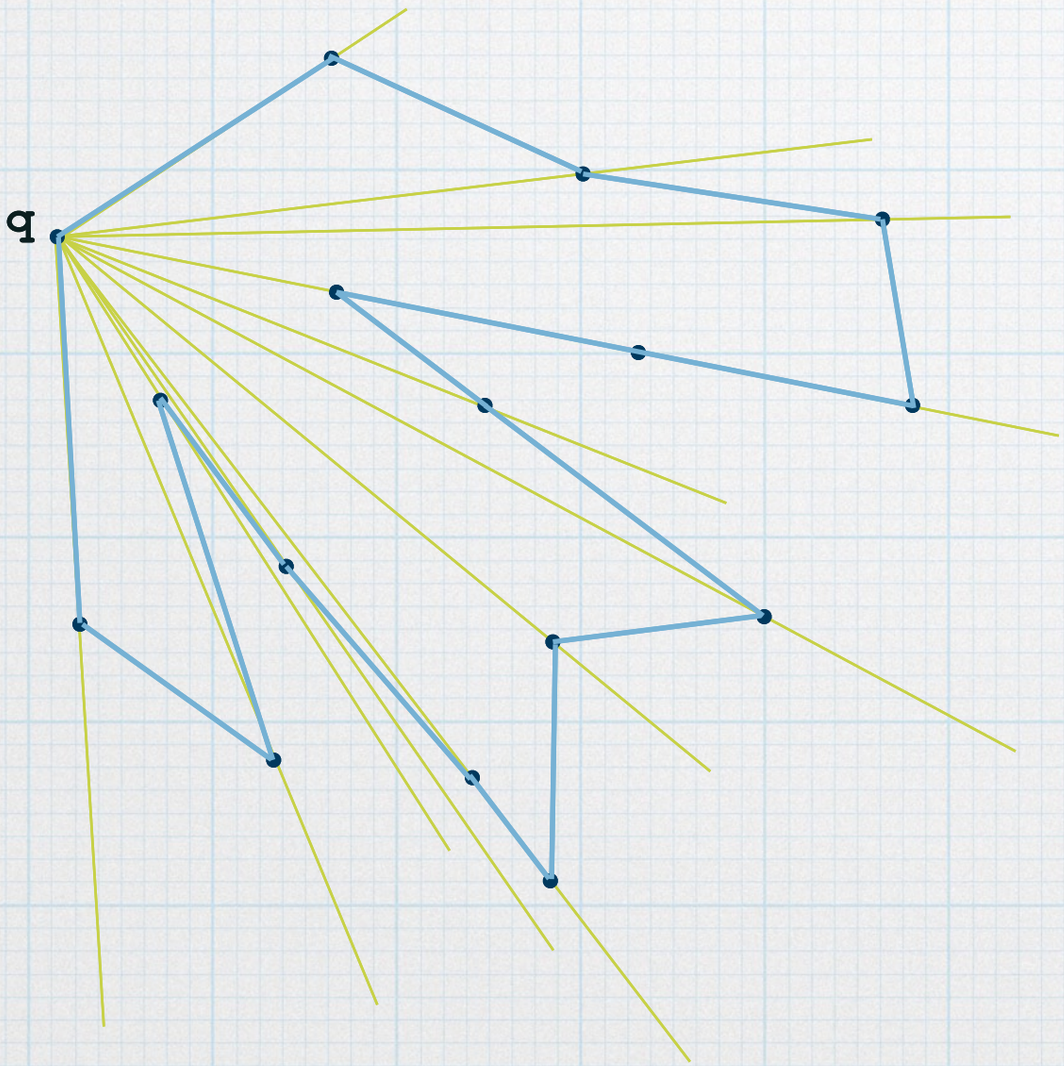
1. a ponthalmaz polárszög szerinti rendezése.

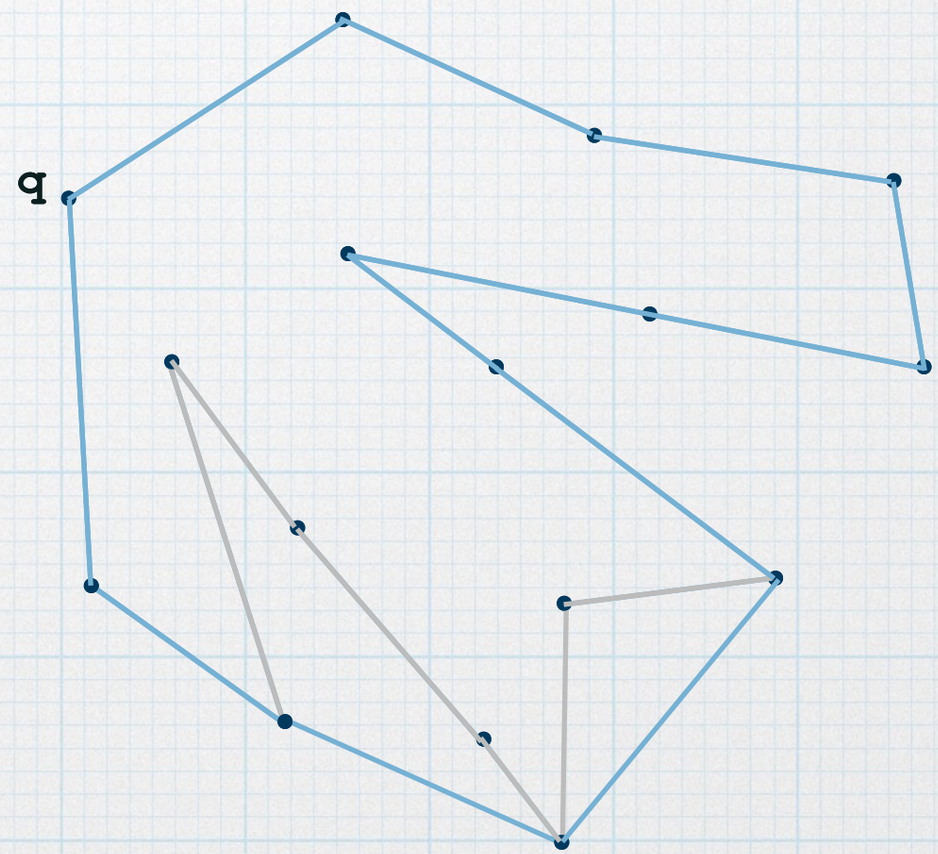




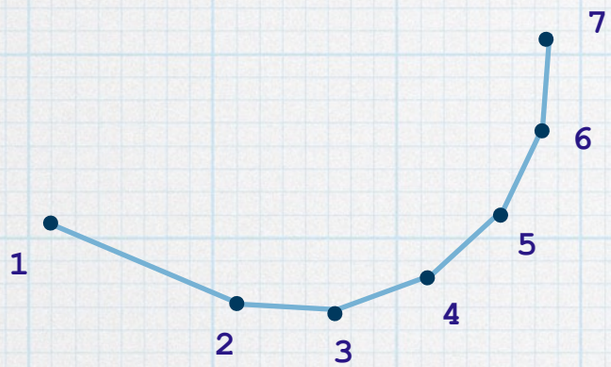
Megoldás

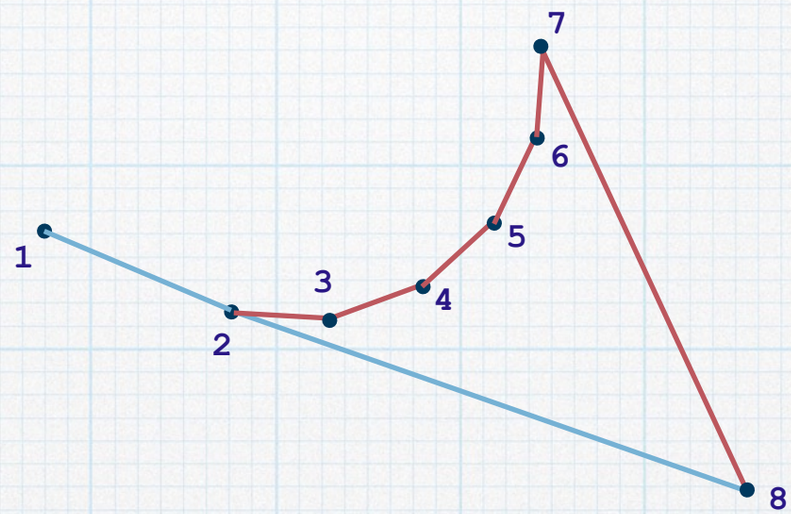
1. a ponthalmaz polárszög szerinti rendezése.

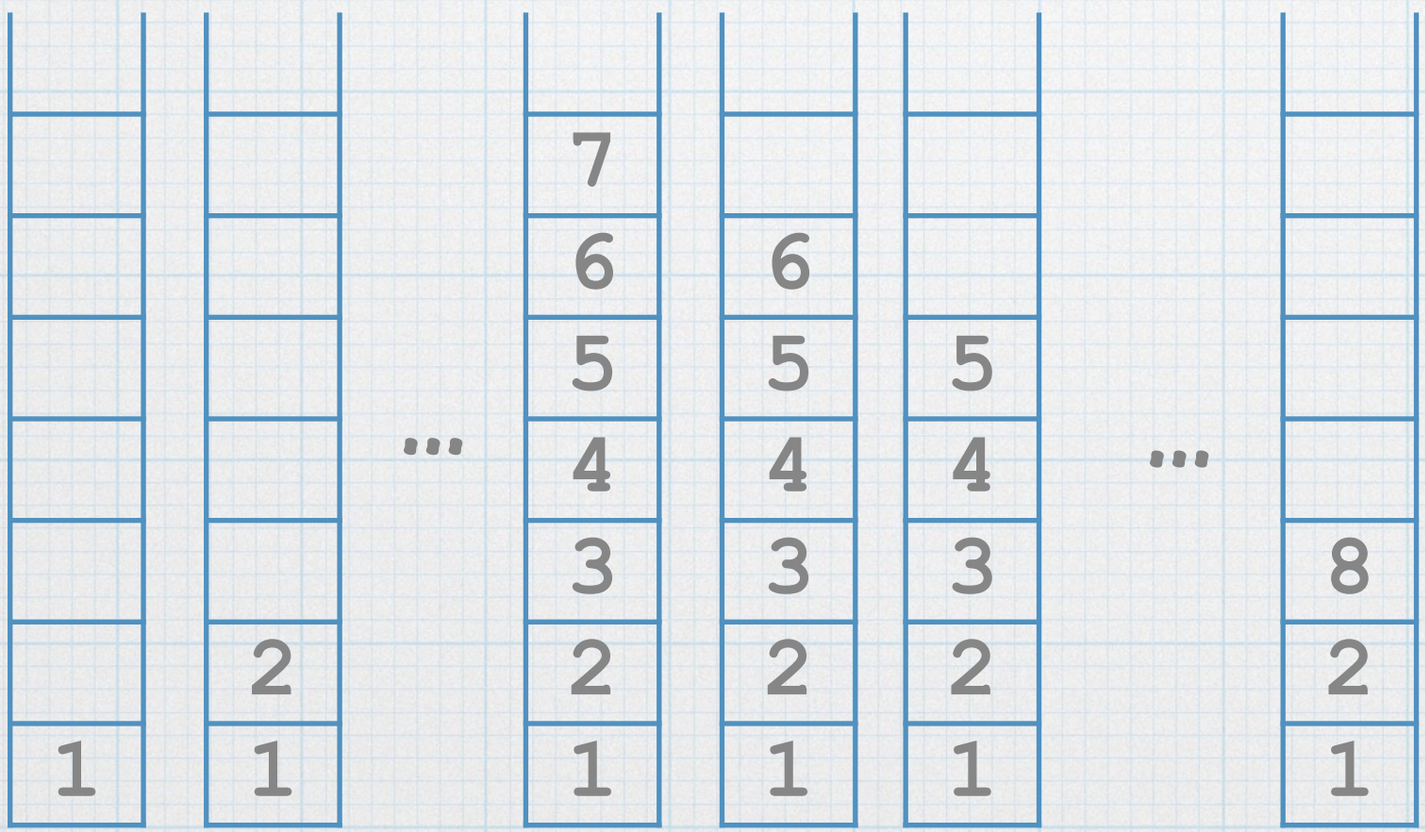




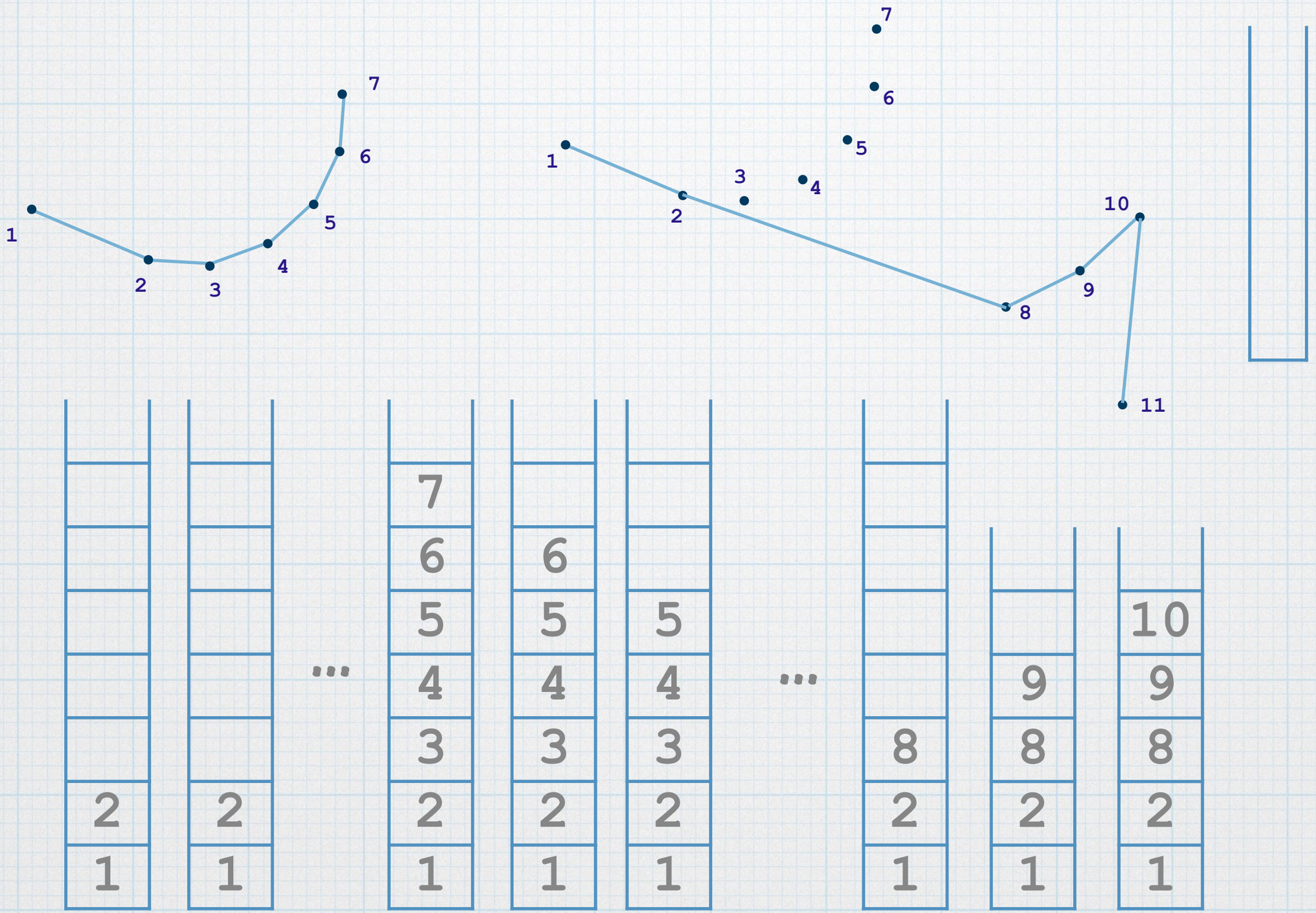
Megoldás



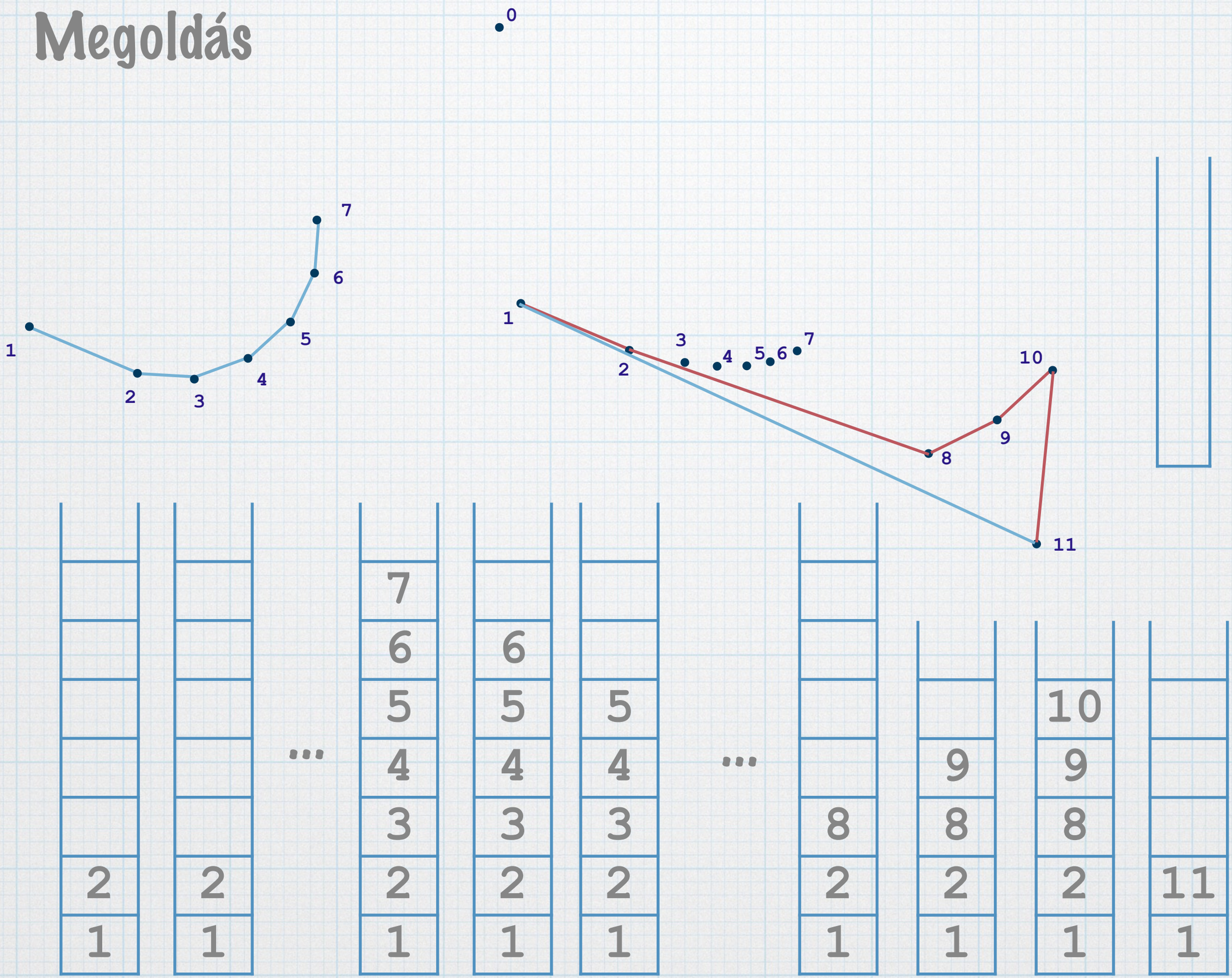




Megoldás


1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
10
11
2
1
2
1
...
7
6
5
4
3
2
1
6
5
4
3
2
1
5
4
3
2
1
...
8
2
1
9
8
2
1
10
9
8
2
1

Megoldás


0
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
10
11
...
...
2
1
2
1
7
6
5
4
3
2
1
6
5
4
3
2
1
5
4
3
2
1
8
2
1
9
8
2
1
10
9
8
2
1
11
1

Megoldás

Konvex burok meghatározása

1. A ponthalmaz polárszög szerinti rendezése. O(nlogn)
2. A pontokat sorra vesszük a rendezés sorrendjében. + O(n)

3. Ha a verem tetején lévő pont, az aktuális és a következő pont forgásiránya + O(n)

   1 vagy 0: az aktuális pontot betesszük a verembe

   -1: kivesszük a pontot a veremből és visza 3.-hoz

4. Az utolsó pont után mégegyszer vesszük q-t, majd megállunk.
   A veremben lévő pontok a konvex burok pontjai.

= O(nlogn)

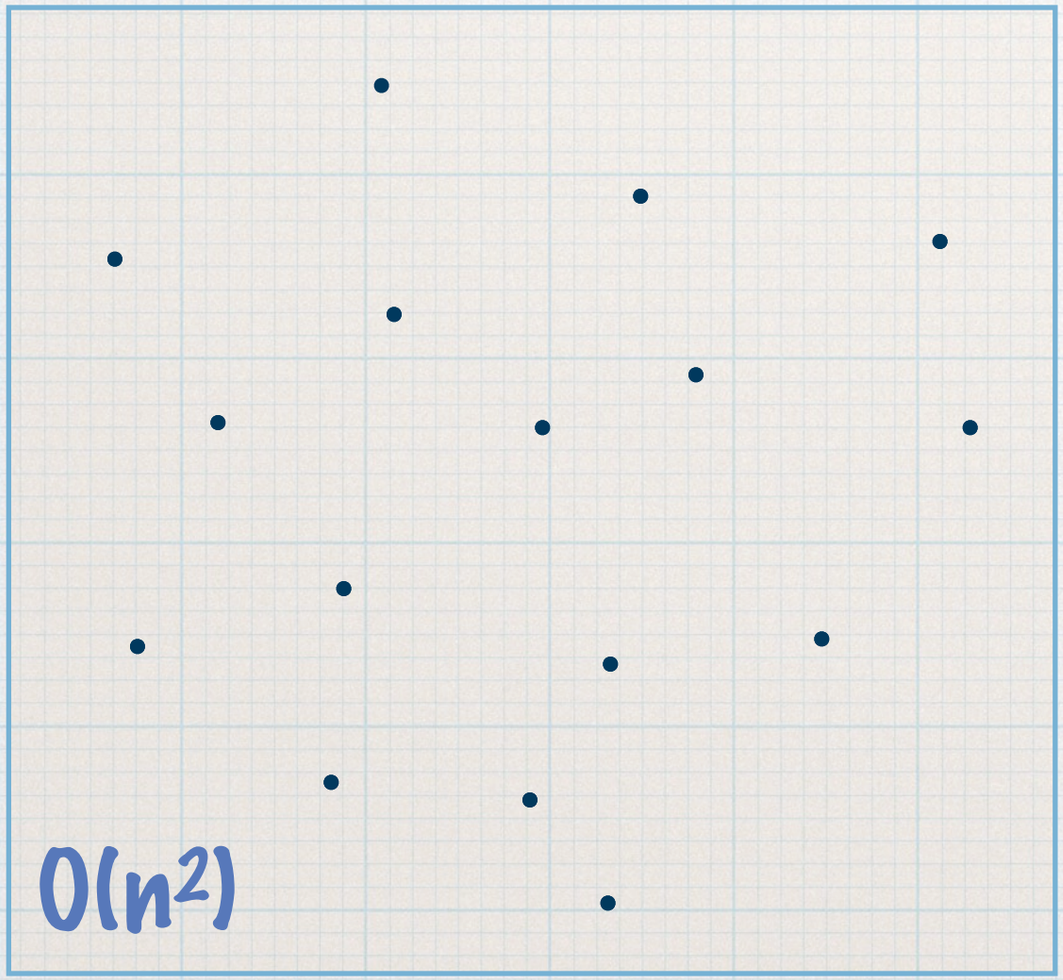
Feladat

Legtávolabbi pontpár meghatározása

Adott egy $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ ponthalmaz a síkon. Szokásos módon, a $p_i = (x_i, y_i)$ és $p_j = (x_j, y_j)$ pontok távolsága

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Adjunk hatékony algoritmust a két legtávolabbi pont meghatározására!



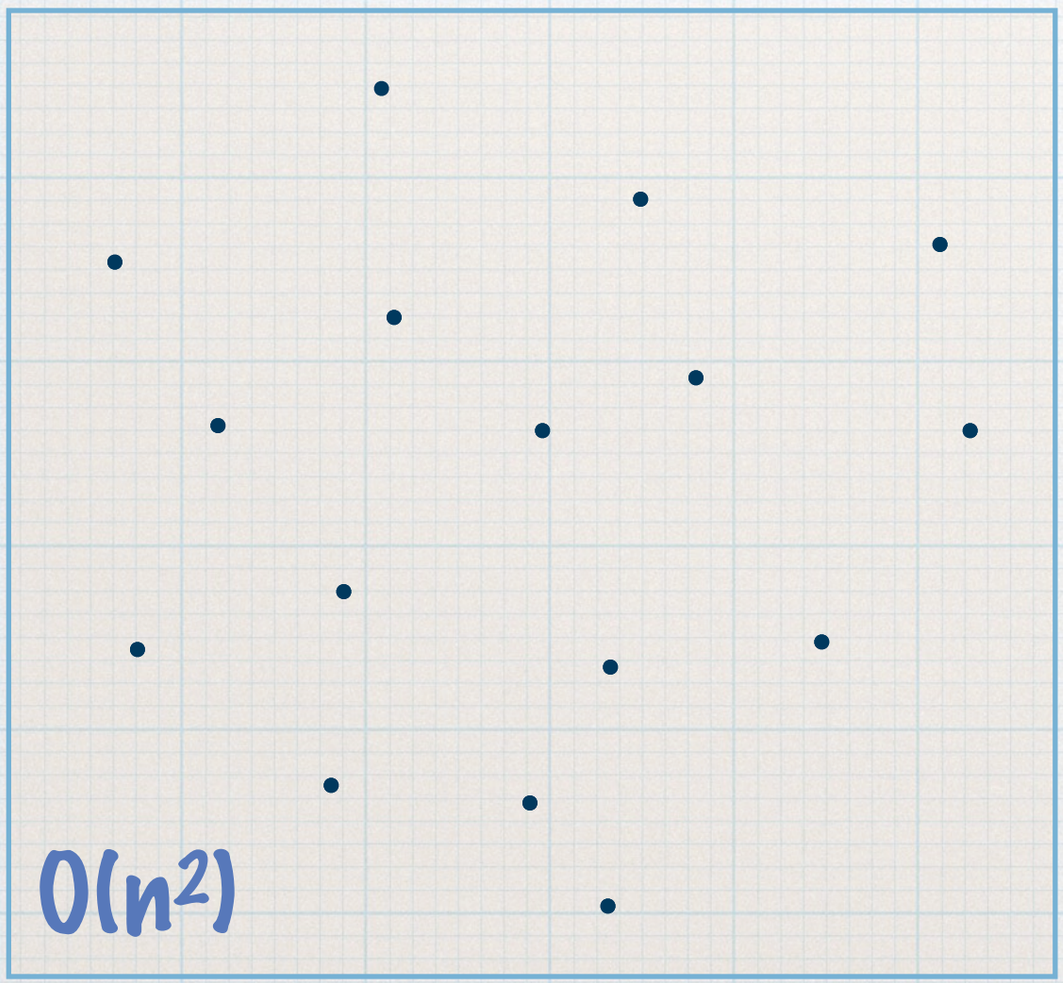
Feladat

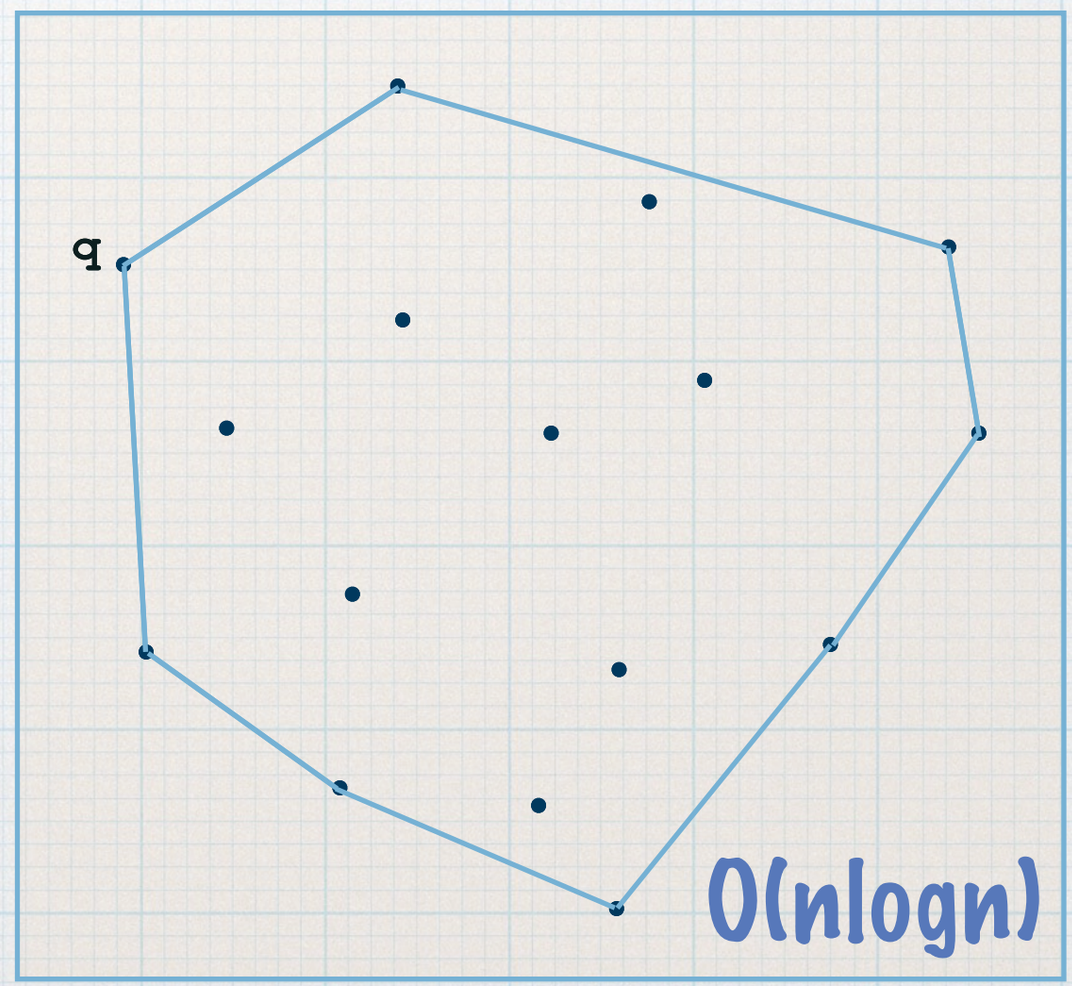
Legtávolabbi pontpár meghatározása

Adott egy $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ ponthalmaz a síkon. Szokásos módon, a $p_i = (x_i, y_i)$ és $p_j = (x_j, y_j)$ pontok távolsága

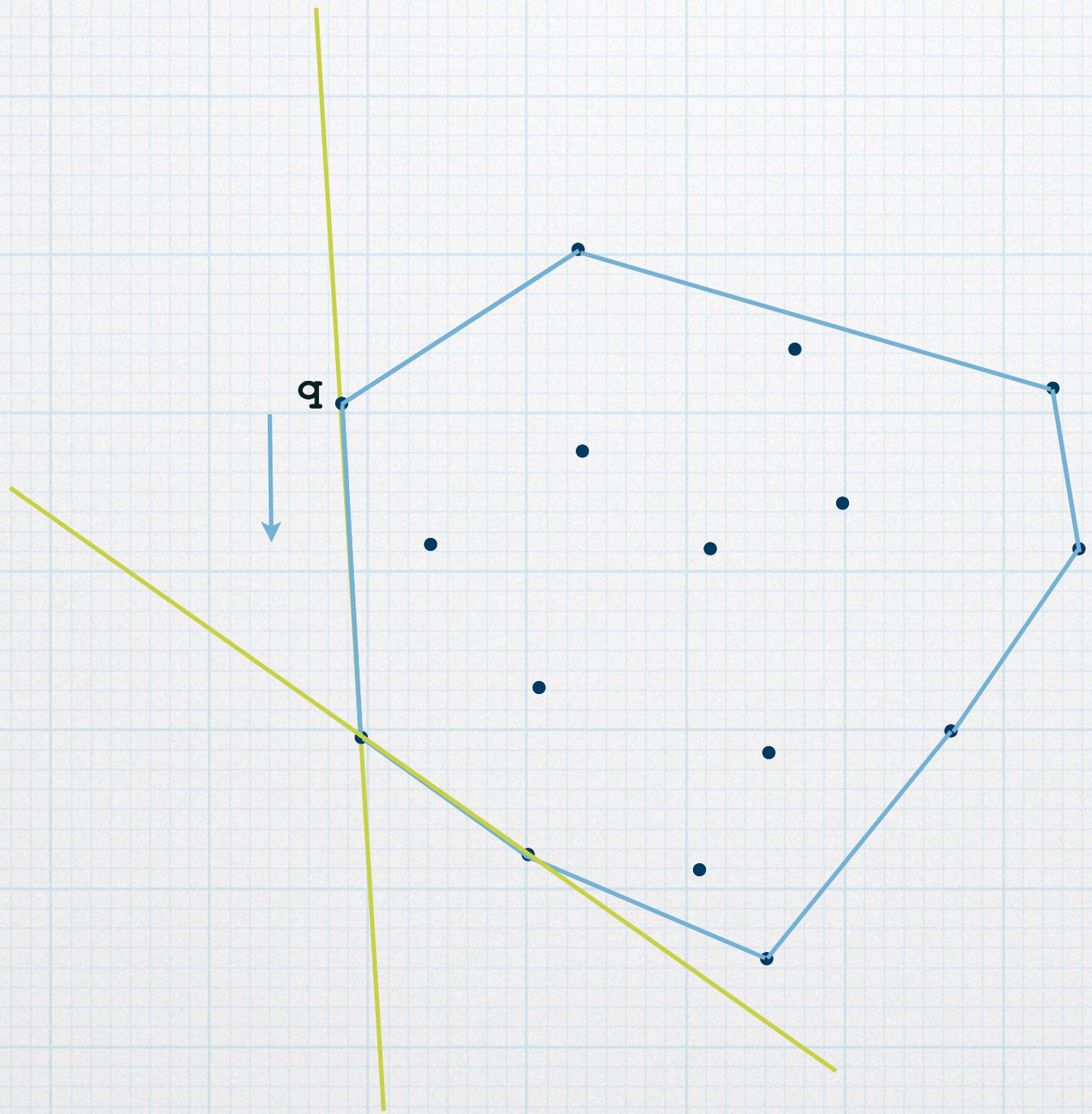
$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Adjunk hatékony algoritmust a két legtávolabbi pont meghatározására!

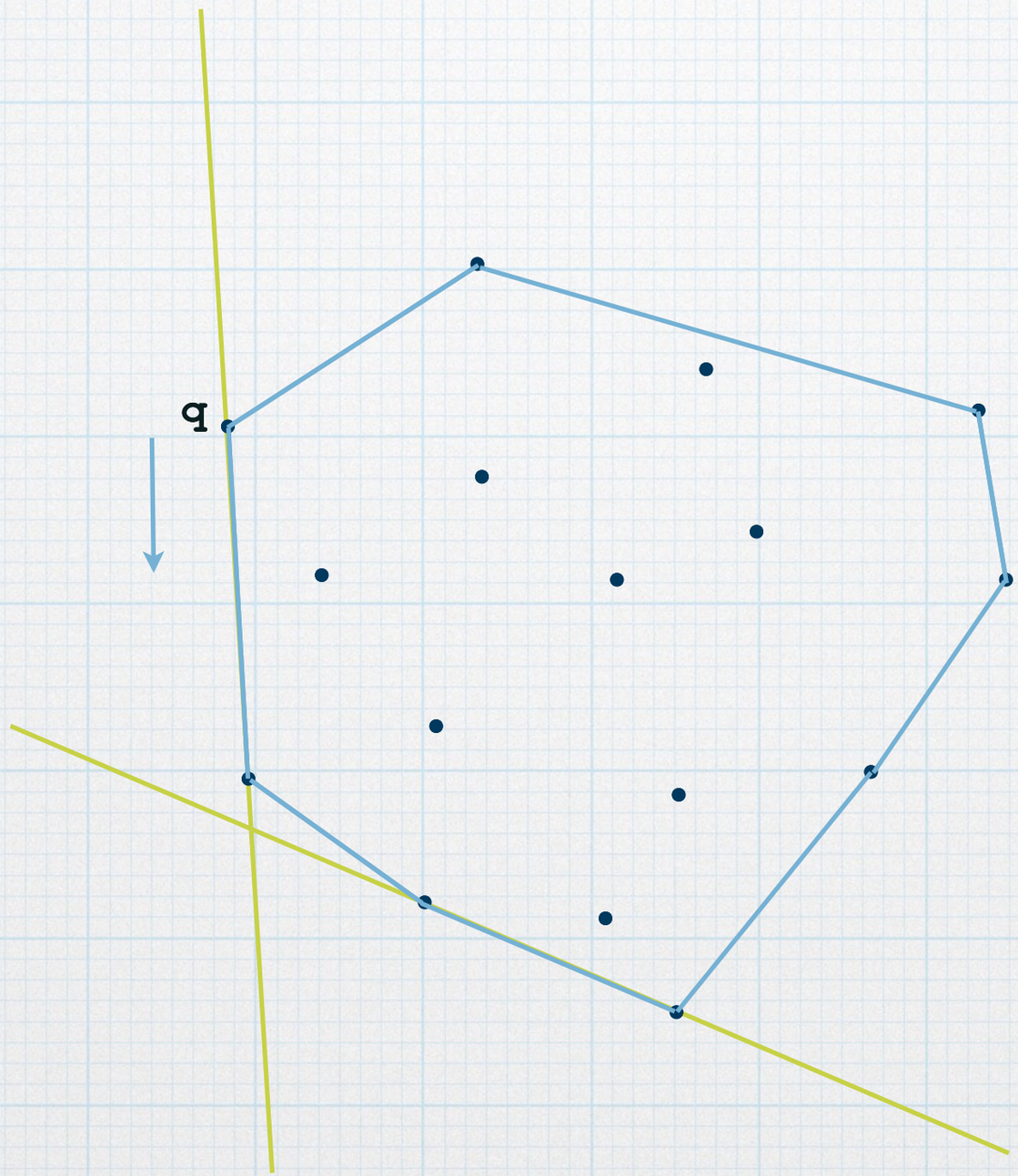




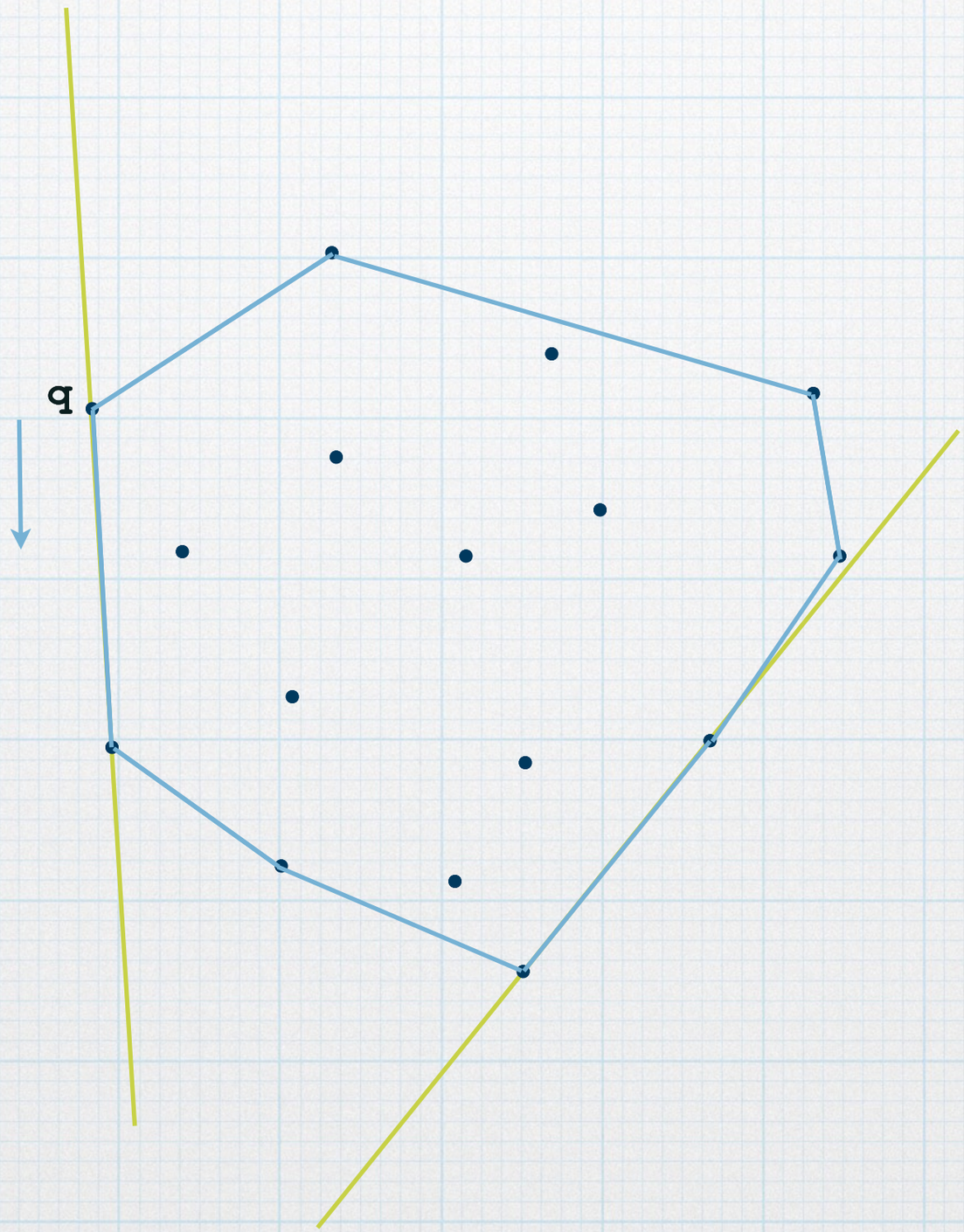
Megoldás



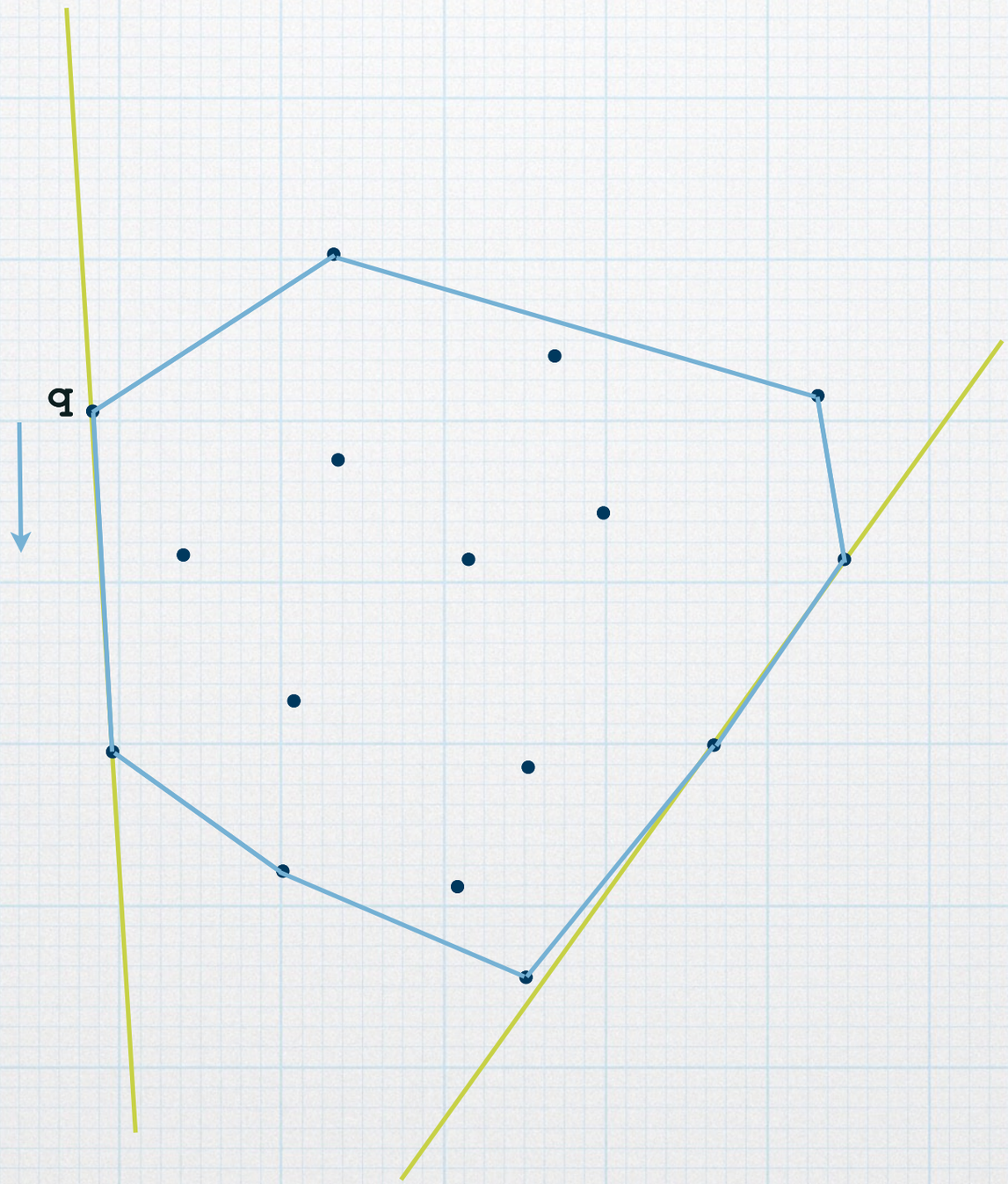
Megoldás



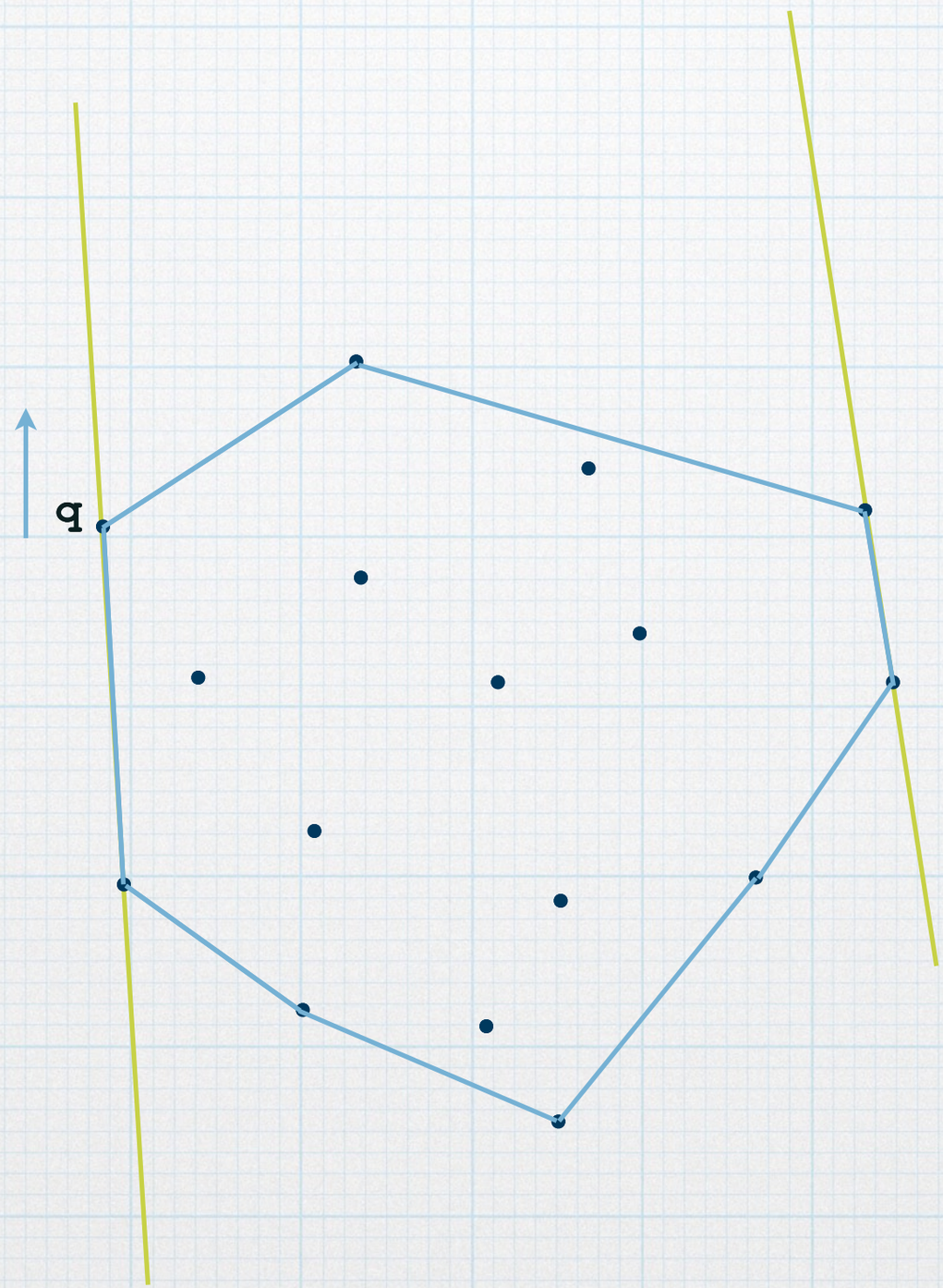
Megoldás



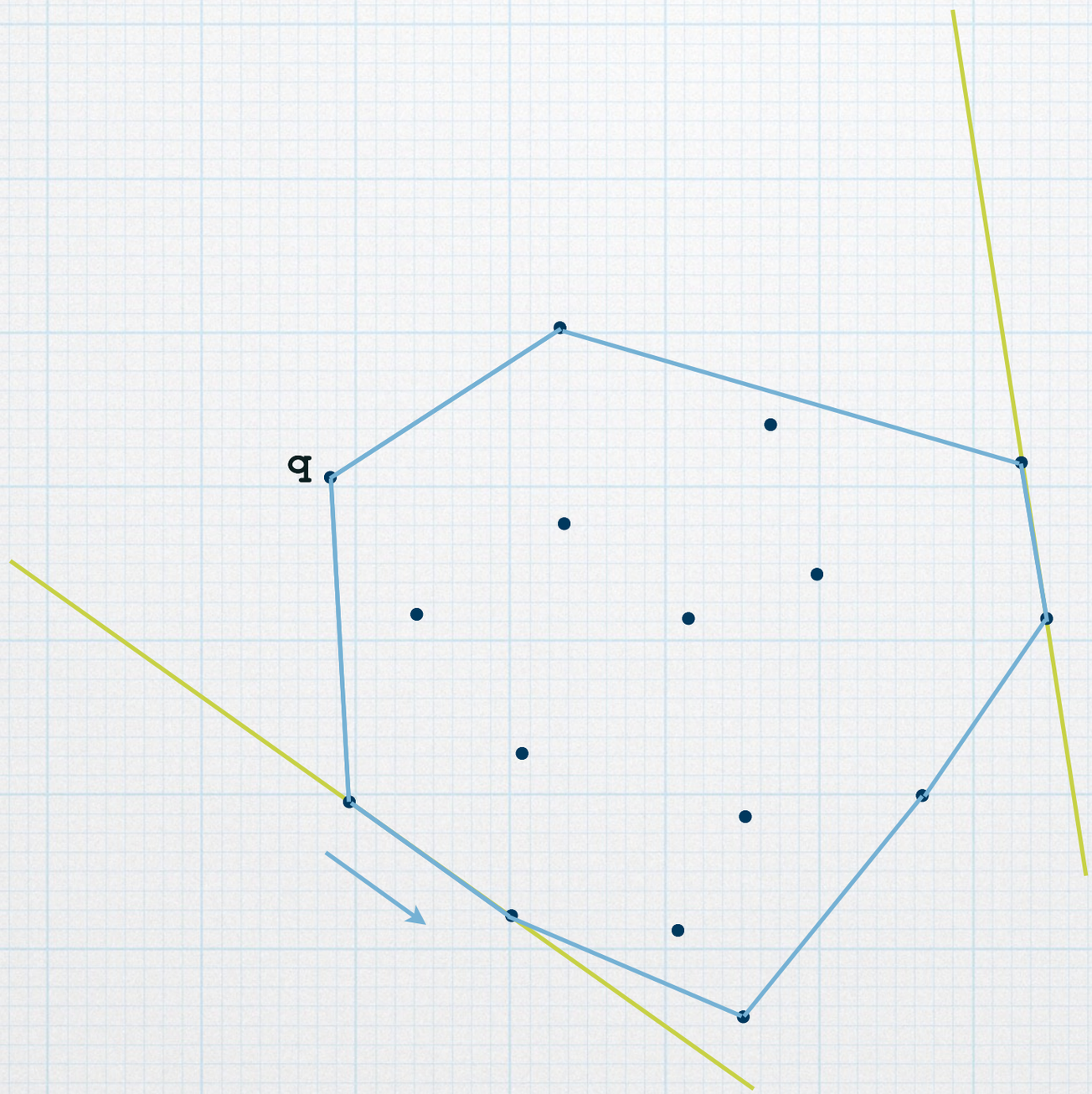
Megoldás



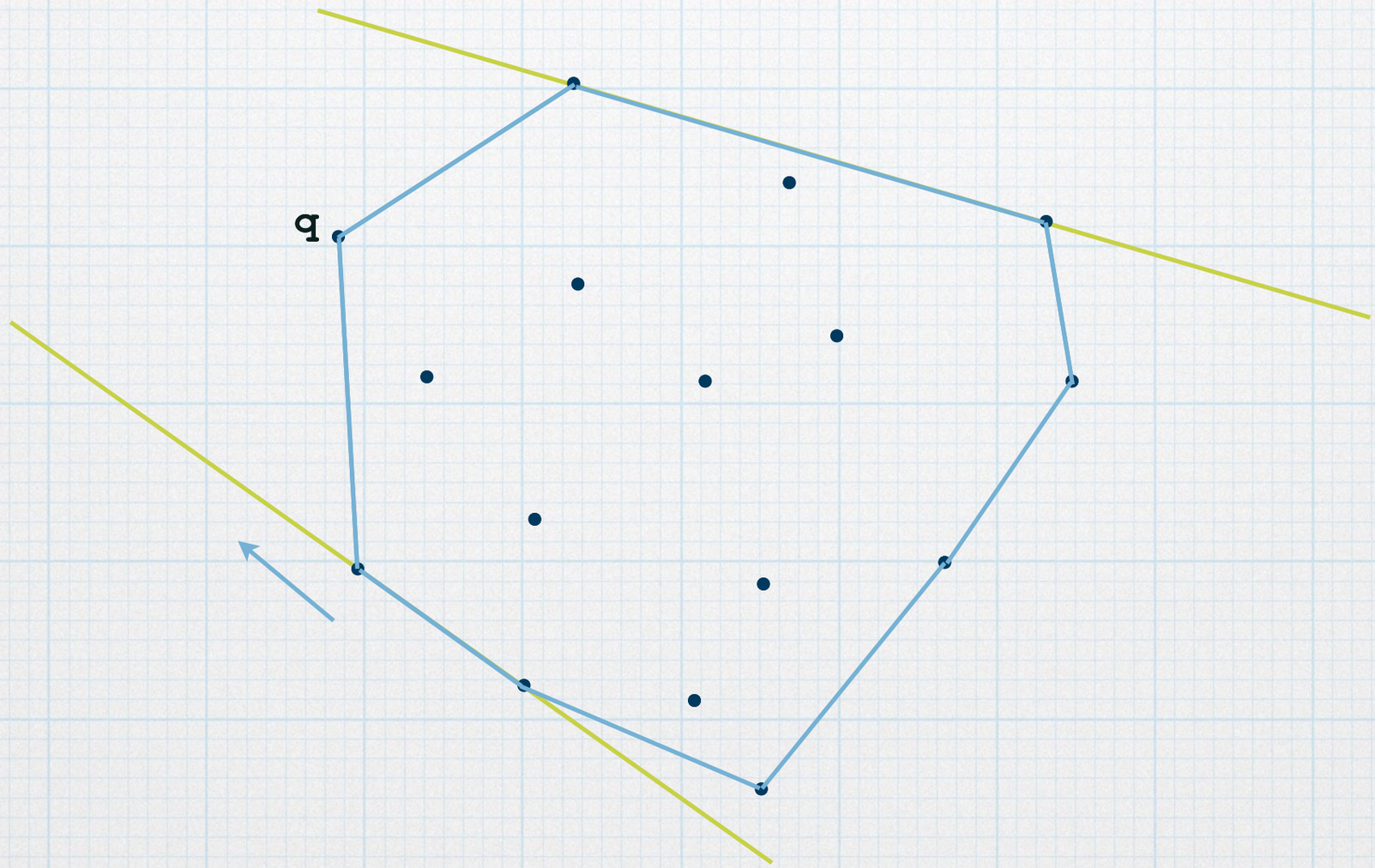
Megoldás



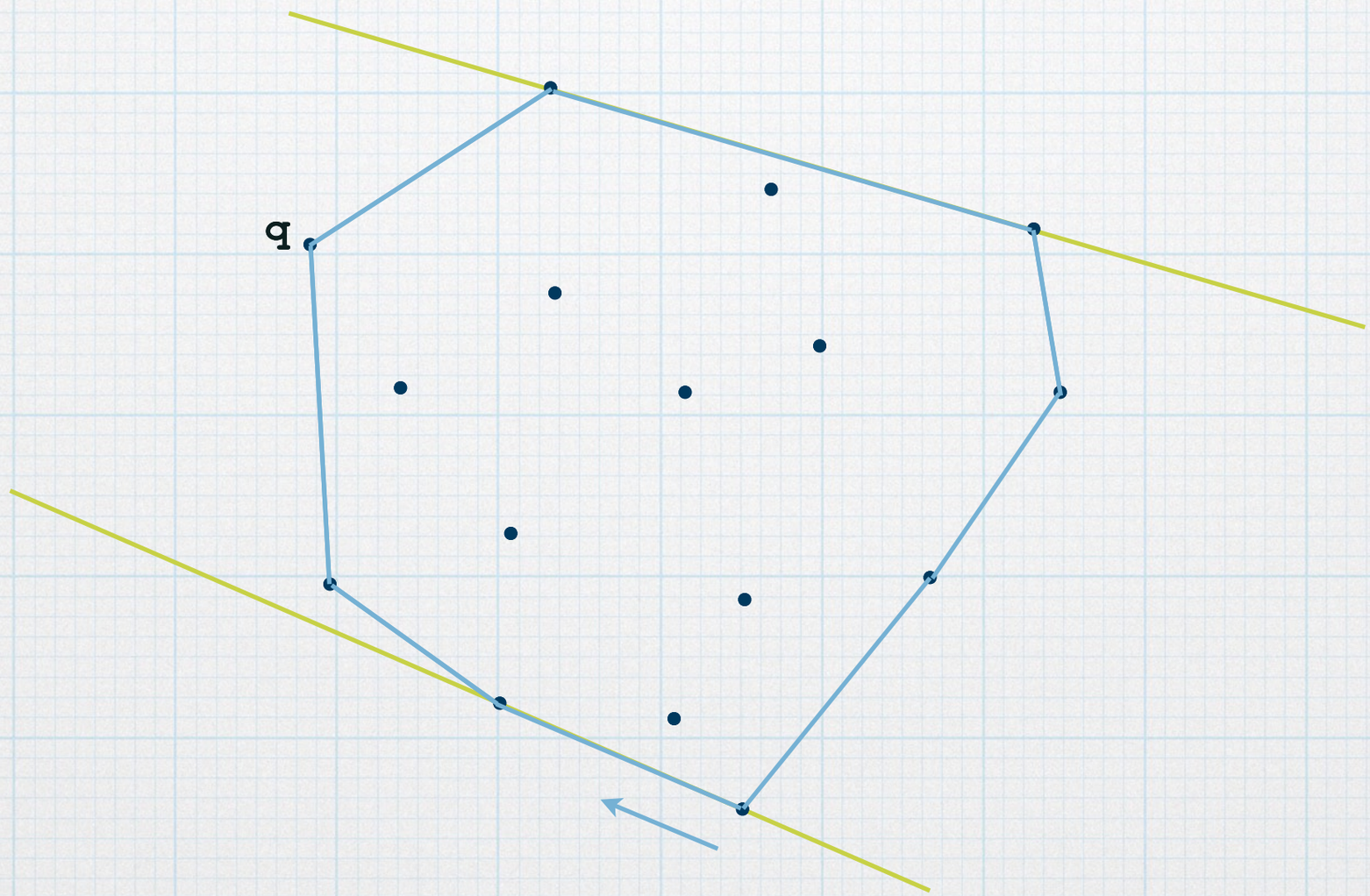
Megoldás



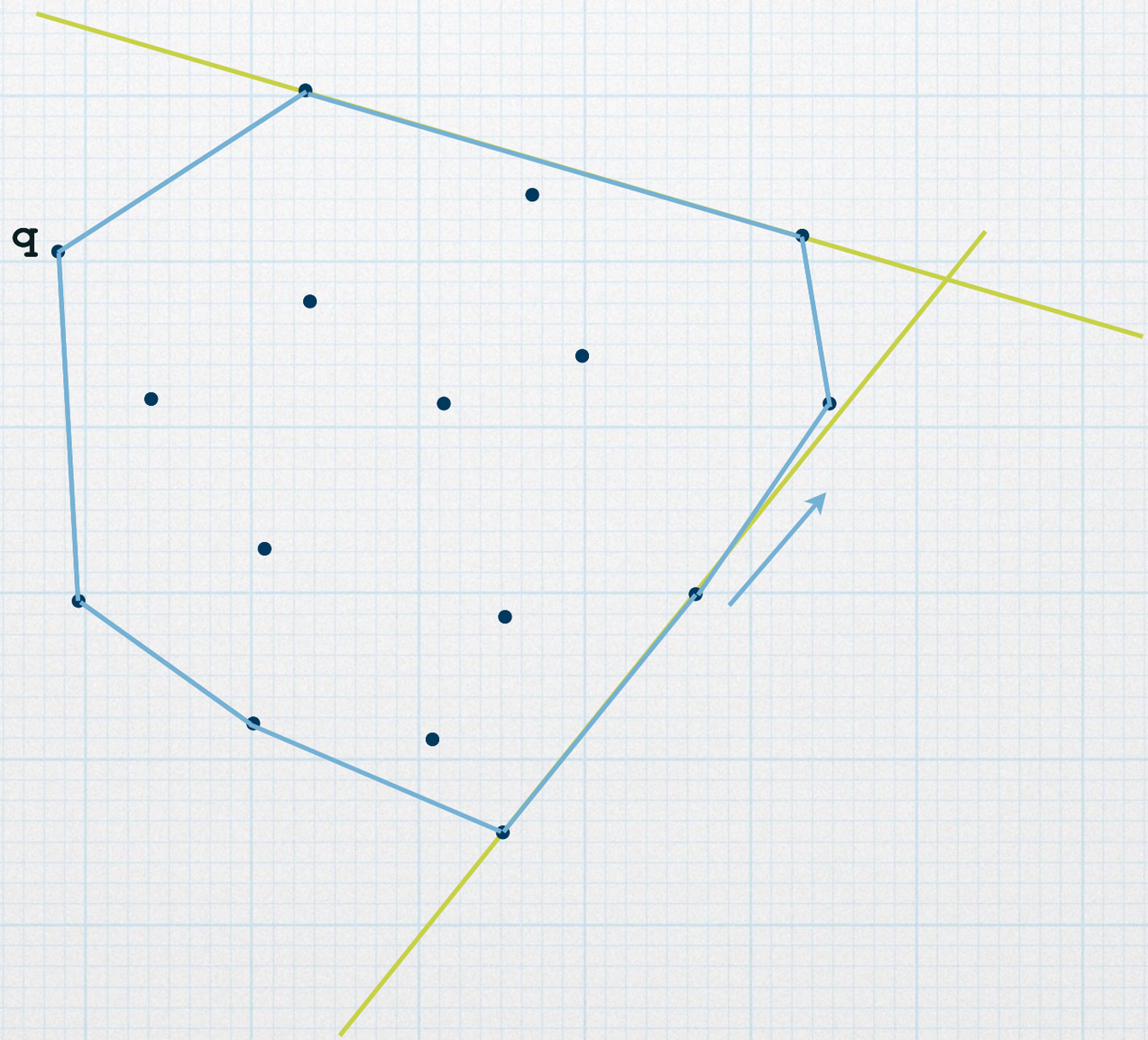
Megoldás



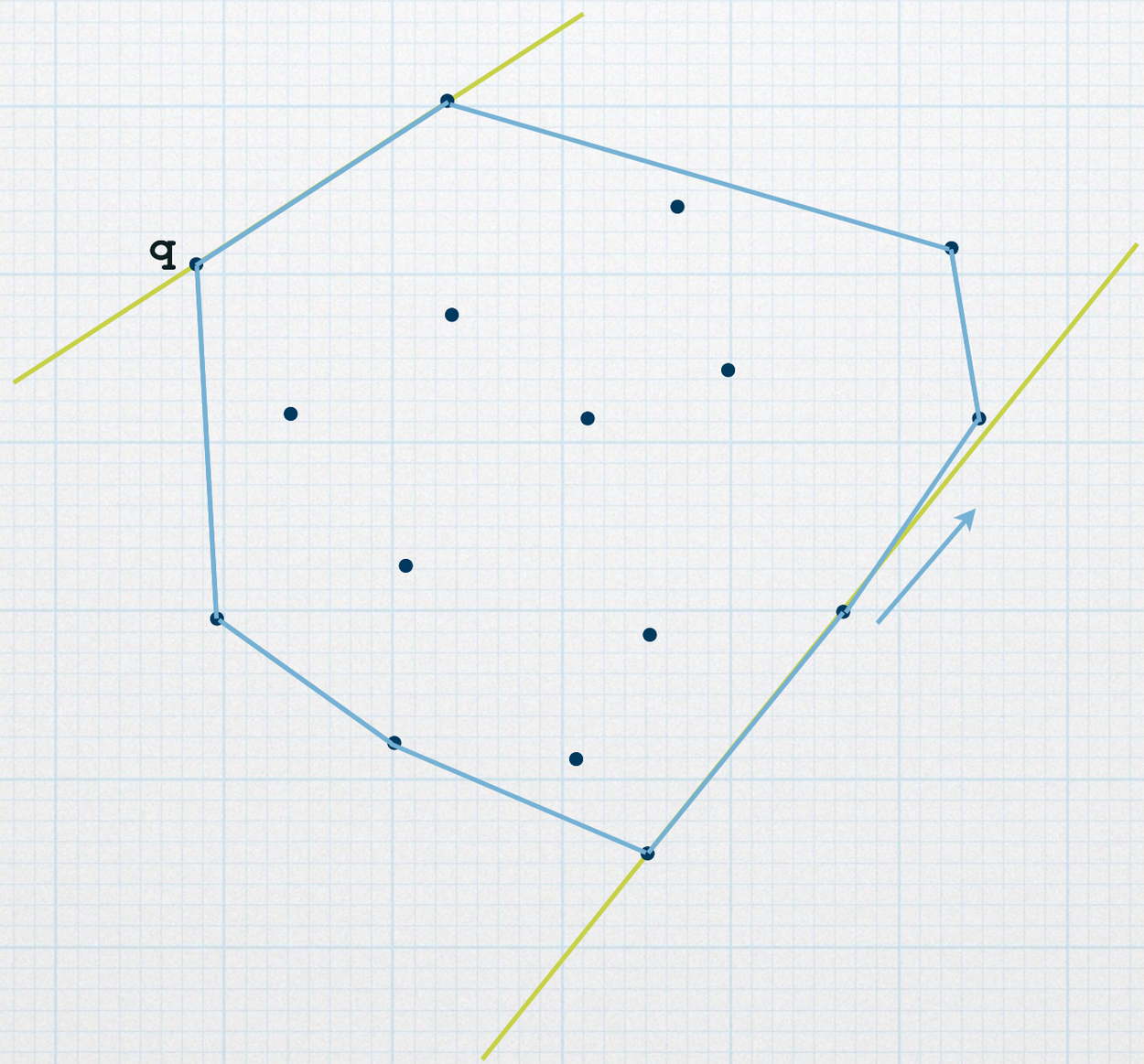
Megoldás



Megoldás

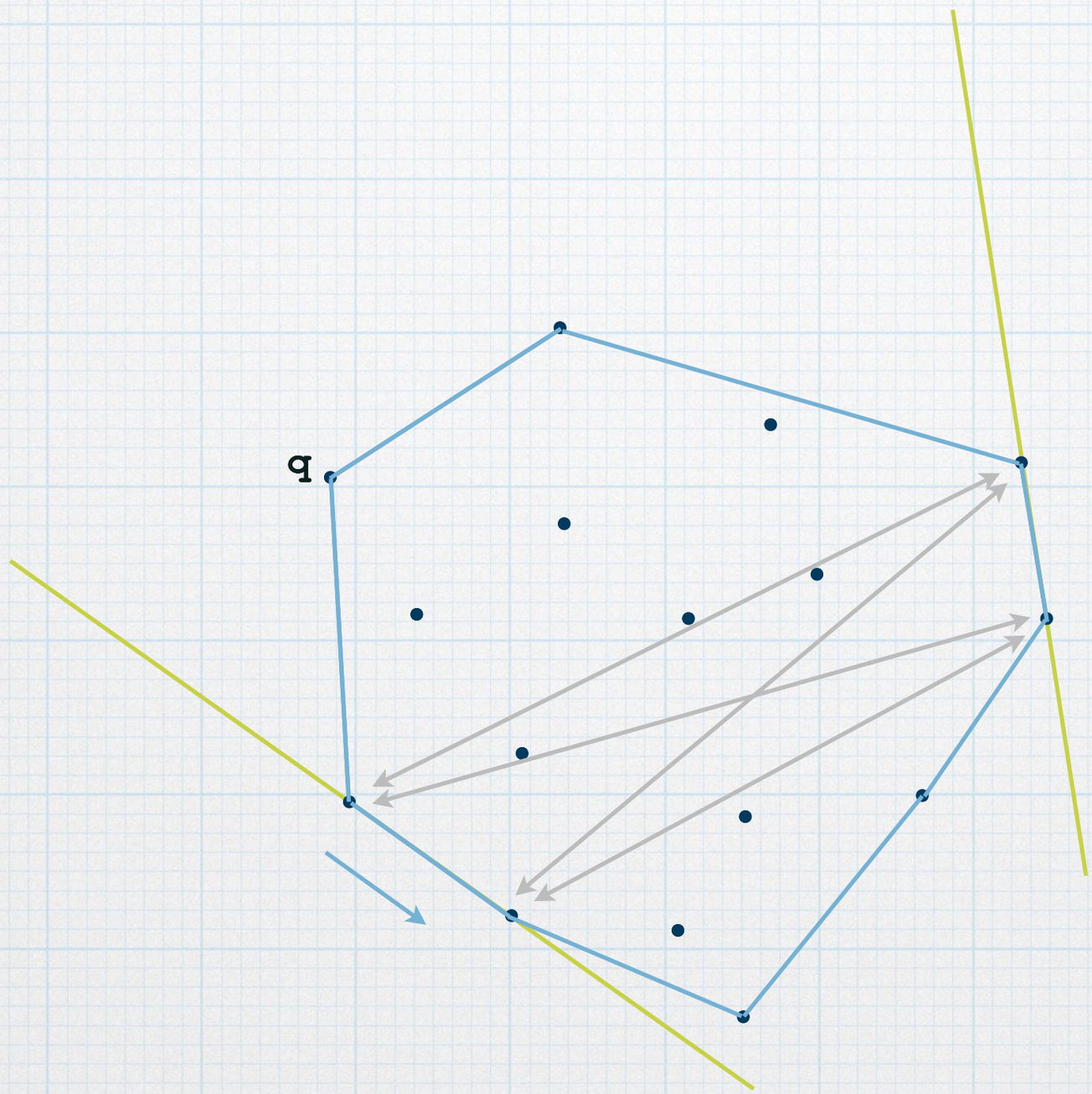


Megoldás



O(n)

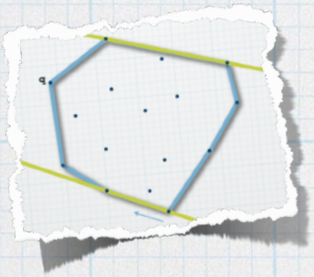
Megoldás



Megoldás

Legtávolabbi pontpár meghatározása

1. A konvex burok meghatározása. O(nlogn)



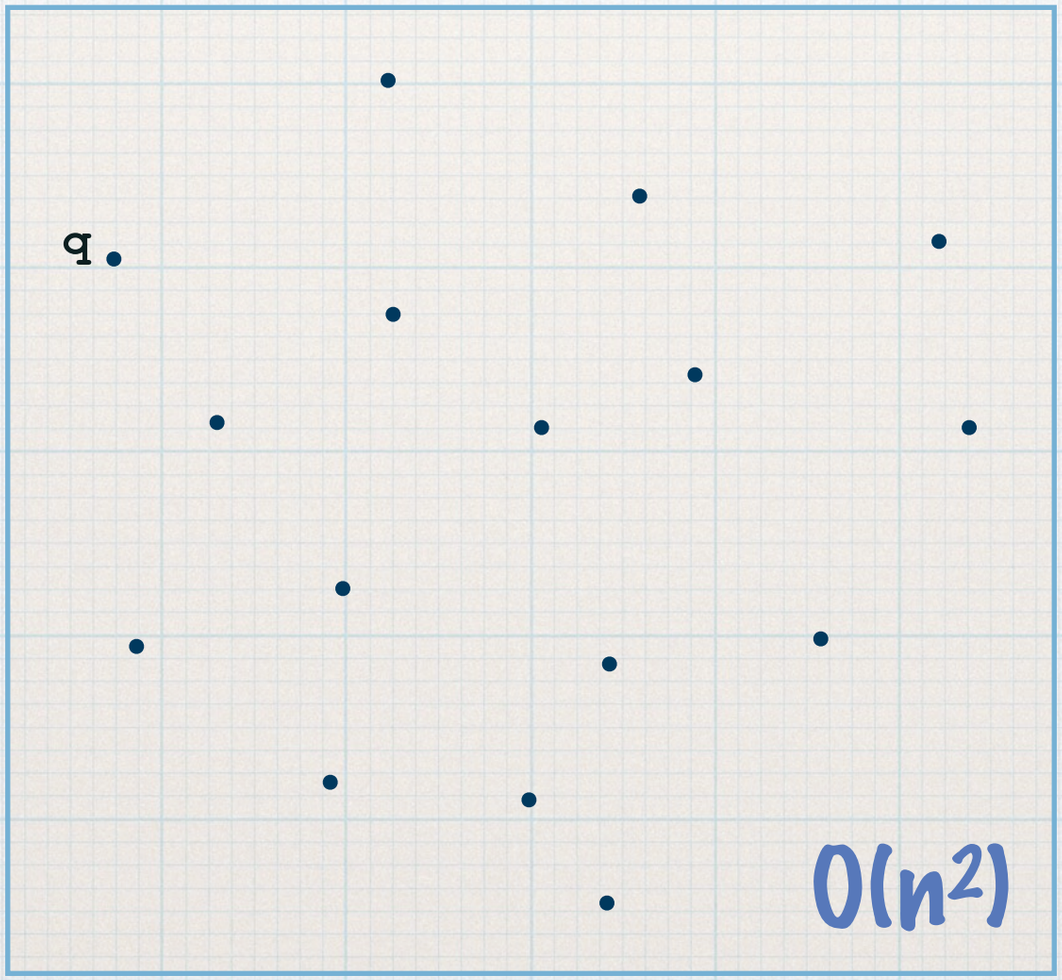
2. A 2 egyenes végiggörgetése és közben maximumkeresés az érintett pontok távolságnégyzetein. + O(n)

O(nlogn)

Feladat

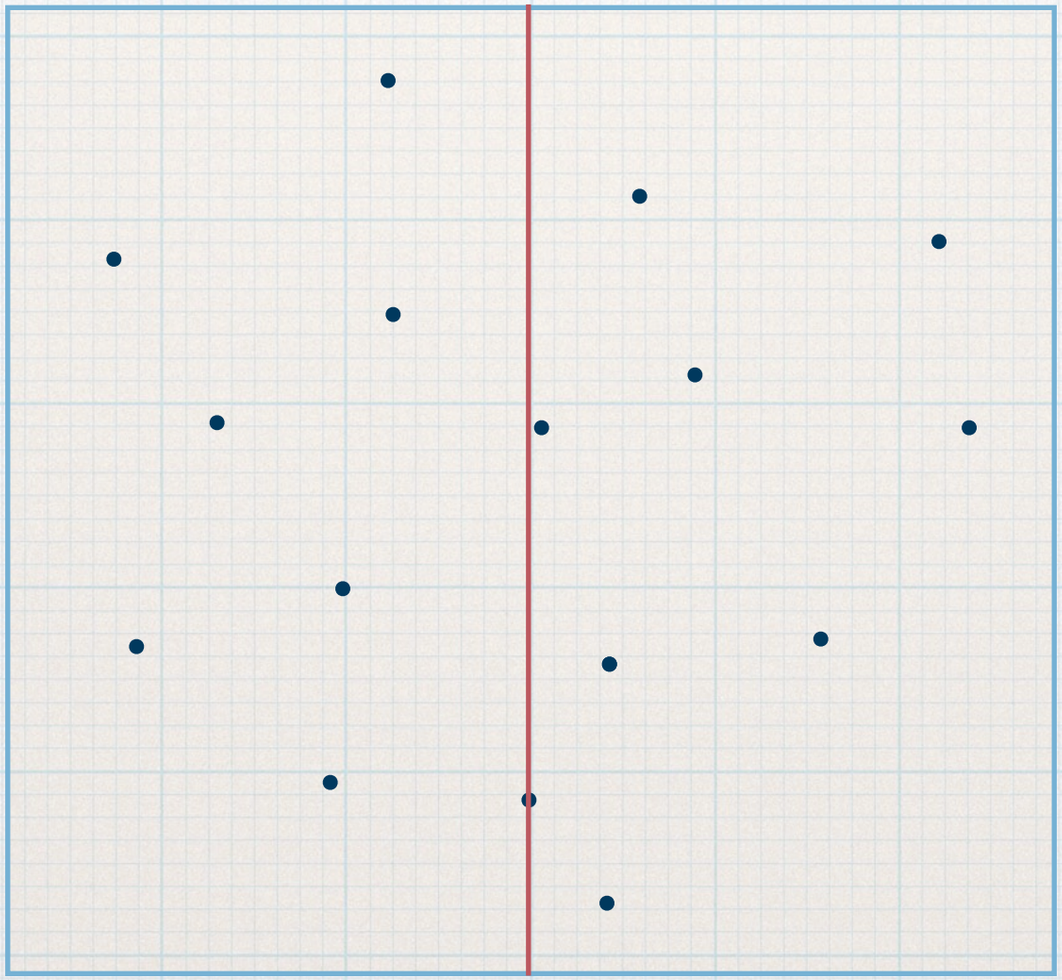
Legközelebbi pontpár meghatározása

Nézzük végig az összes pontpárt és válasszuk ki hol lesz a távolság (négyzete) minimális?

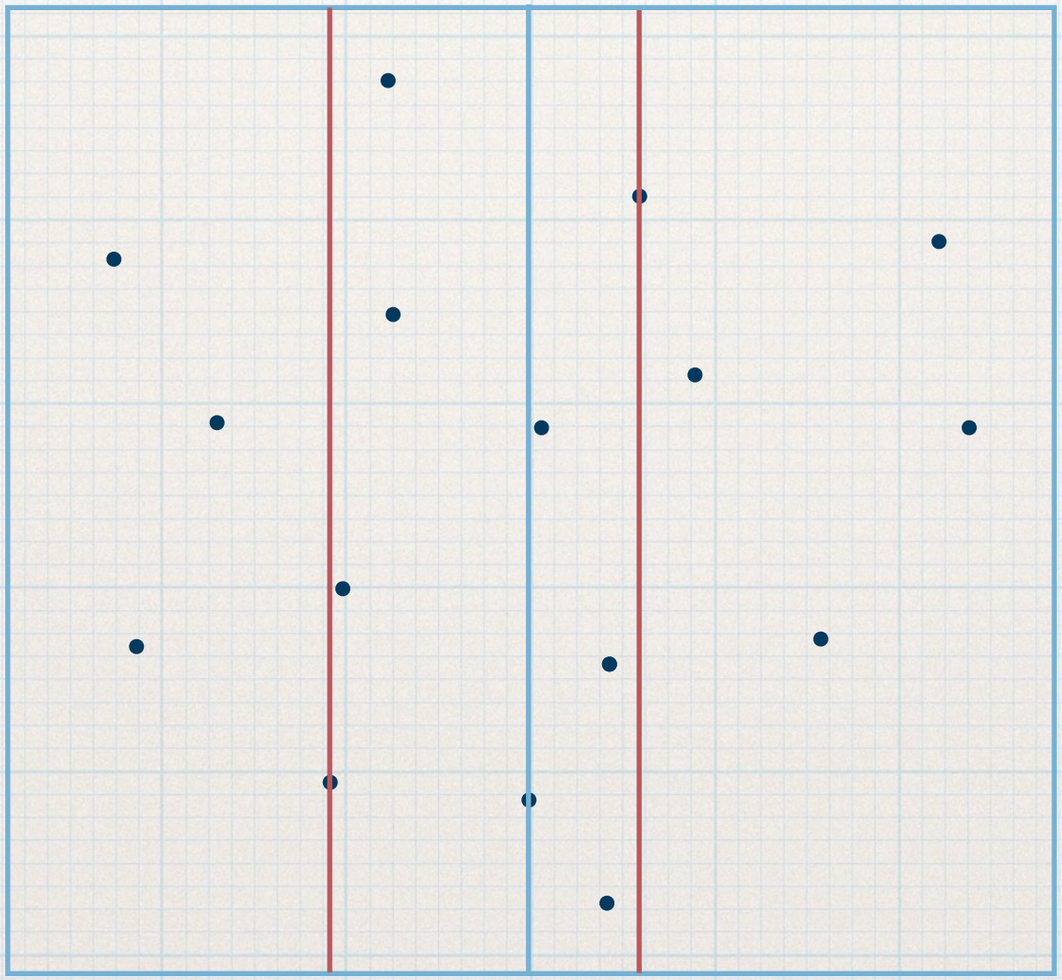


Megoldás

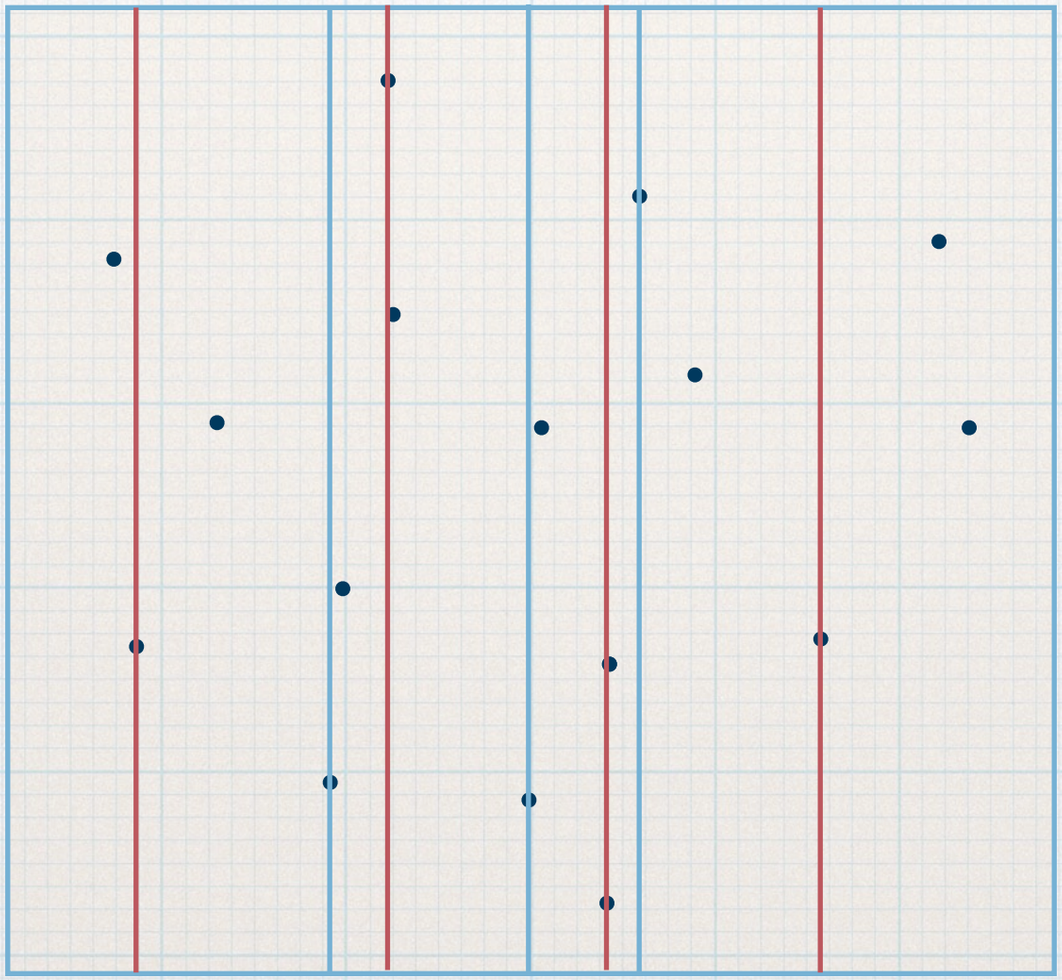
Oroszlán keresése a sivatagban



Megoldás



Megoldás



Minden rekurzív hívásnál átadásra kerülnek a P ponthalmaz egy Q részhalmazának pontjai egy az x koordináták szerint monoton növekvően rendezett X tömbben (azonos x koordinátájú pontok esetén a kisebb y koordinátájú jön előbb), valamint egy az y koordináták szerint monoton növekvően rendezett Y tömbben (azonos y koordinátájú pontok esetén a kisebb x koordinátájú jön előbb).
Hogy ne kelljen minden egyes rekurzív hívás előtt rendezésre pazarolni az időt, ezért már az algoritmus elején előállítjuk P pontjainak az x, illetve az y koordináták szerint monoton növekvően rendezett tömbjeit, így a rekurzív hívások előtt csak kiválogatásokat kell végezni.

Egy adott rekurzív lépés a következő.
Ha $|Q| < 3$, akkor páronkénti vizsgálattal határozzuk meg a minimális távolságot.
Természetesen az érdekes eset, amikor $|Q| > 3$, ilyenkor az alábbiak szerint járunk el:

Először egy olyan **e** függőleges egyenest keresünk, amely Q-t két olyan Q_L és Q_R részre osztja, amelyekre:

- $|Q_L| = \lceil |Q|/2 \rceil$ és $|Q_R| = \lfloor |Q|/2 \rfloor$
- Q_L minden pontja az egyenes bal oldalán van, vagy illeszkedik az egyenesre
- Q_L minden pontja az egyenes jobb oldalán van, vagy illeszkedik az egyenesre

Válogassuk szét a pontokat az X tömbből az X_L és X_R tömbökbe: X_L-be kerüljenek a Q_L-beli pontok, X_R-be pedig a Q_R-beliek. Természetesen az X_L és X_R tömbök is rendezettek az x koordináták szerint.
Válogassuk szét a pontokat az Y tömbből is az Y_L és Y_R tömbökbe a fentiekhez hasonlóan!

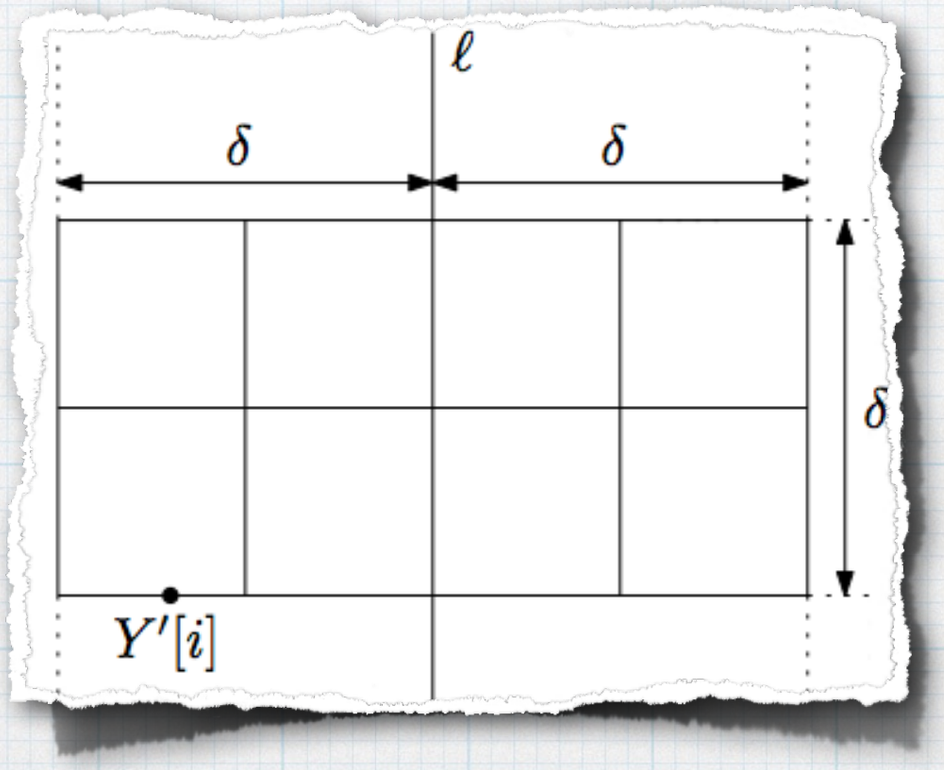
Ezután keressük meg rekurzívan a Q_L és Q_R halmazokban a legközelebbi pontpárt.
Az első hívás bemenete az X_L és Y_L tömbök, míg a második hívásé az X_R és Y_R tömbök. Legyen Q_L-ben, illetve Q_R-ben a minimális távolság δ_L, illetve δ_R, és legyen $\delta = \min(\delta_L, \delta_R)$

Most a legközelebbi pontpár:

- vagy az egyik rekurzív hívás által megtalált δ távolságú páros,
- vagy egy olyan pár, amelynek egyik pontja Q_L-ben, a másik Q_R-ben van.

A következő lépés annak meghatározása, hogy ez utóbbi párok között van-e olyan, amelyben a pontok távolsága kisebb, mint δ. Vegyük észre, hogy ha van ilyen pár, akkor annak egyik pontja sem lehet δ-nál nagyobb távolságra e-től.
Válogassuk ki az Y tömbből azokat a pontokat, amelyek e-től mért távolsága legfeljebb δ. Jelölje a kapott tömböt Y'.
Természetesen az Y' tömb is rendezett az y koordináták szerint.

Most minden egyes $p \in Y'$ ponthoz keressük meg Y' azon pontjait, melyek p-től mért távolsága legfeljebb δ, és amelyek a p-n átmenő vízszintes egyenesen vagy az fölött vannak.
Legfeljebb 7 ilyen pont jöhet számításba:
Ha $i < j$ és $Y'[i]$ valamint $Y'[j]$ távolsága legfeljebb δ, akkor ez a két pont az ábrán látható $\delta \times 2\delta$ oldalú téglalapban van.



Mivel a téglalapban lévő 8 kis négyzet egyikében sem lehet egynél több pont Y'-ből, az állítás adódik.

Így minden $p \in Y'$ pontra elég kiszámolni p távolságát az Y'-ben utána következő 7 ponttól. Az így kapott távolságok minimuma legyen δ'. Ha δ' < δ, akkor Q-ban a minimális távolság δ', egyébként a minimális távolság δ.

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{ha } n > 3, \\ O(1) & \text{ha } n \leq 3. \end{cases}$$

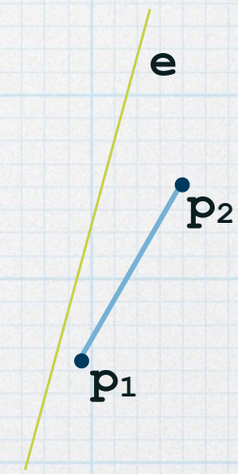
$$T(n) = O(n \log n).$$

Mivel az előrendezések is megvalósíthatók O(n log n) költséggel, ezért az algoritmus teljes költsége szintén O(n log n).

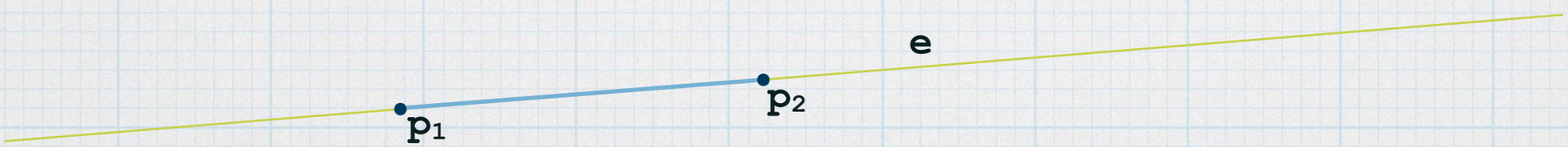
```
TAVKERES(P,X,Y) {
   if |P| ≤ 3 then return NAIV(P)
   Legyen PL, P ⌈|P|/2c⌉darab legkisebb x koordinátájú eleme.
   Legyenek XL,YL a PL beli pontok rendezett halmazai.
   Legyen PR P többi eleme
   Legyenek XR,YR a PR beli pontok rendezett halmazai.
   Legyen x0 PL-t és PR szeparáló x koordináta
   dL := TAVKERES(PL,XL,YL)
   dR := TAVKERES(PR,XR,YR)
   d := min(dL,dR)
   Yd := azon P-beli pontok y koordináta szerint rendezett
halmaza, amelyek x-koordinátájára |x0-x| ≤ d (max. 7 pont)
   Ha van Yd-ben d-nél közelebbi pár, akkor azt adjuk vissza,
egyébként d-t a megfelelo pontpárral
}
```

Szakaszok metszése

A p_1p_2 szakasz átfog egy egyenest,
ha p_1 az egyenes egyik, p_2 az egyenes másik oldalára esik.



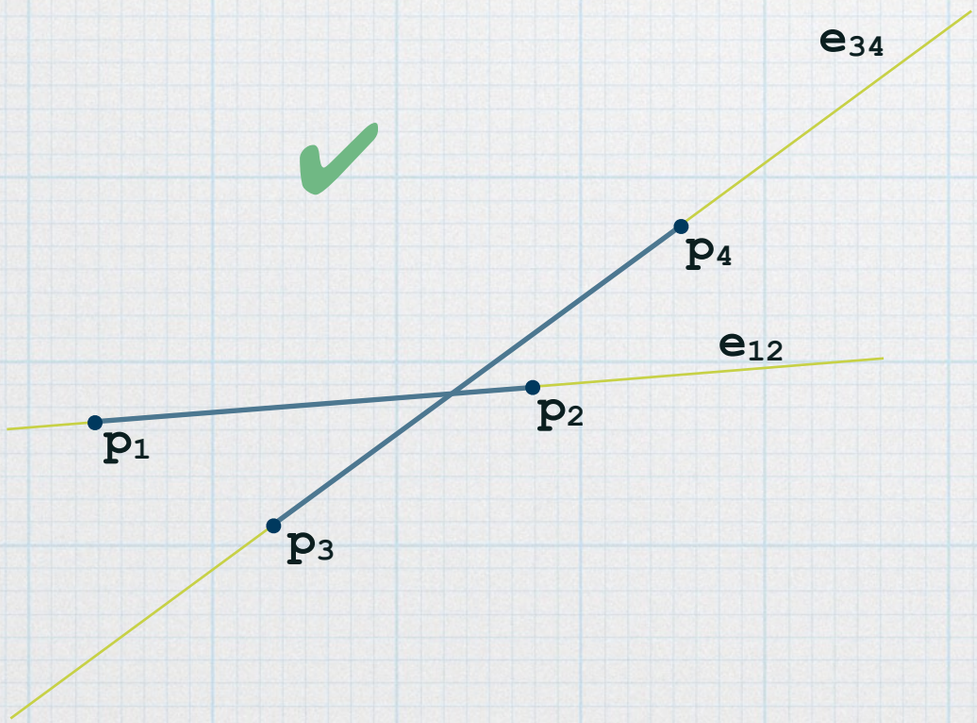
A p_1p_2 szakasz egyenese e, ha $p_1 \in e \wedge p_2 \in e$

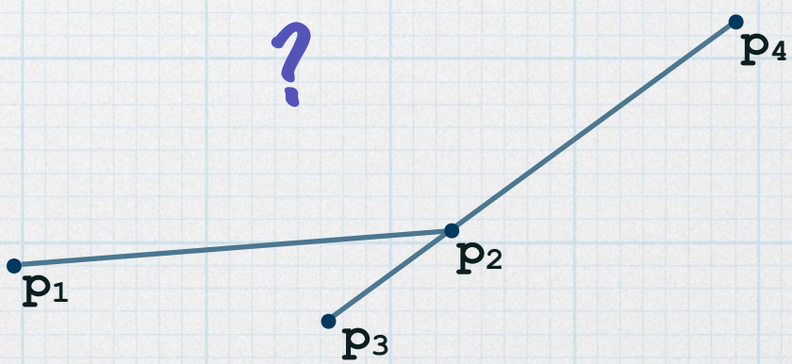


Szakaszok metszése

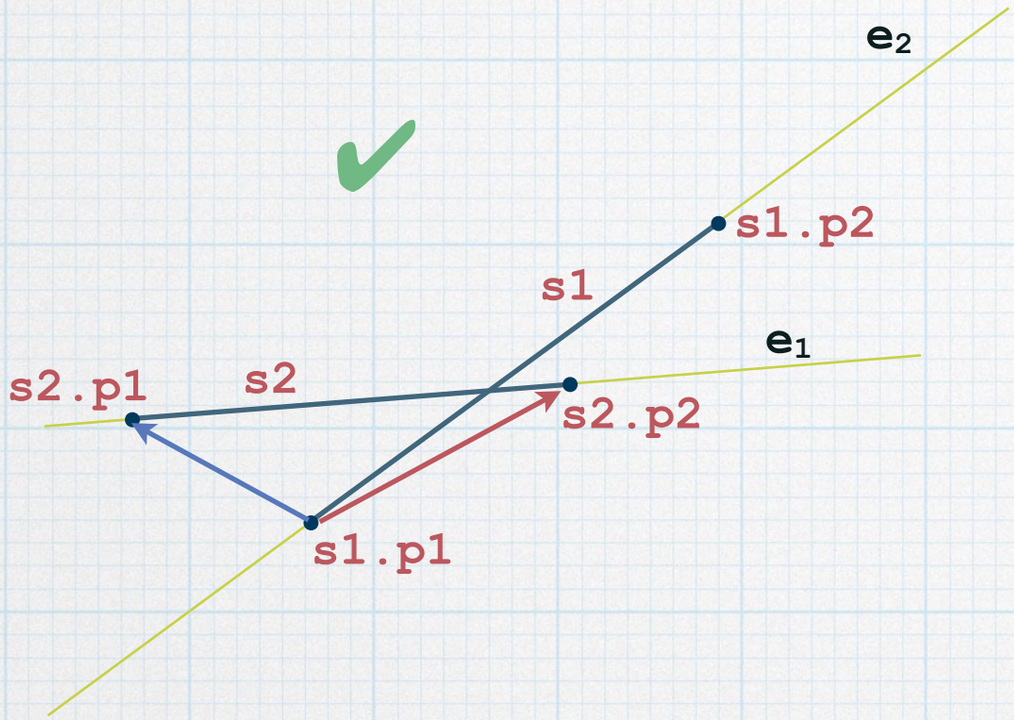
Két szakasz pontosan akkor metszi egymást, ha az alábbi feltételek egyike teljesül:

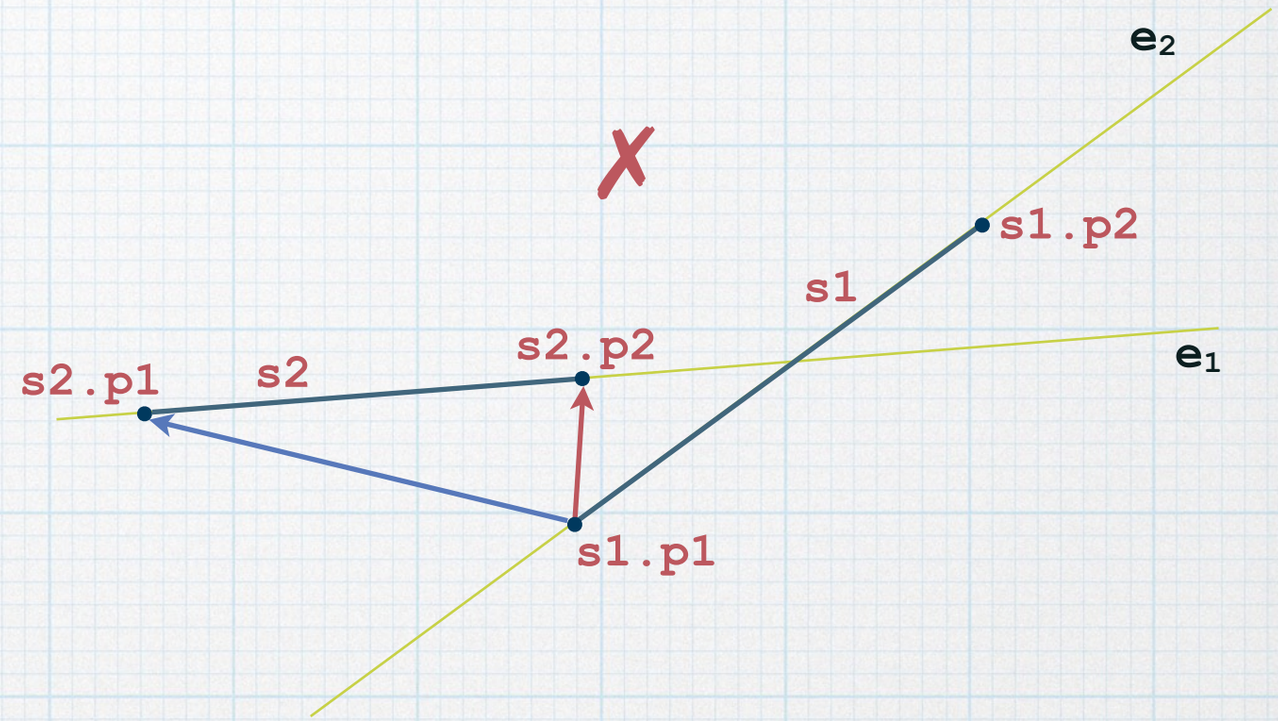
- ✔ mindkét szakasz átfogja a másik egyenesét
- ? az egyik szakasz valamelyik végpontja illeszkedik a másik szakaszra





Szakaszok metszése

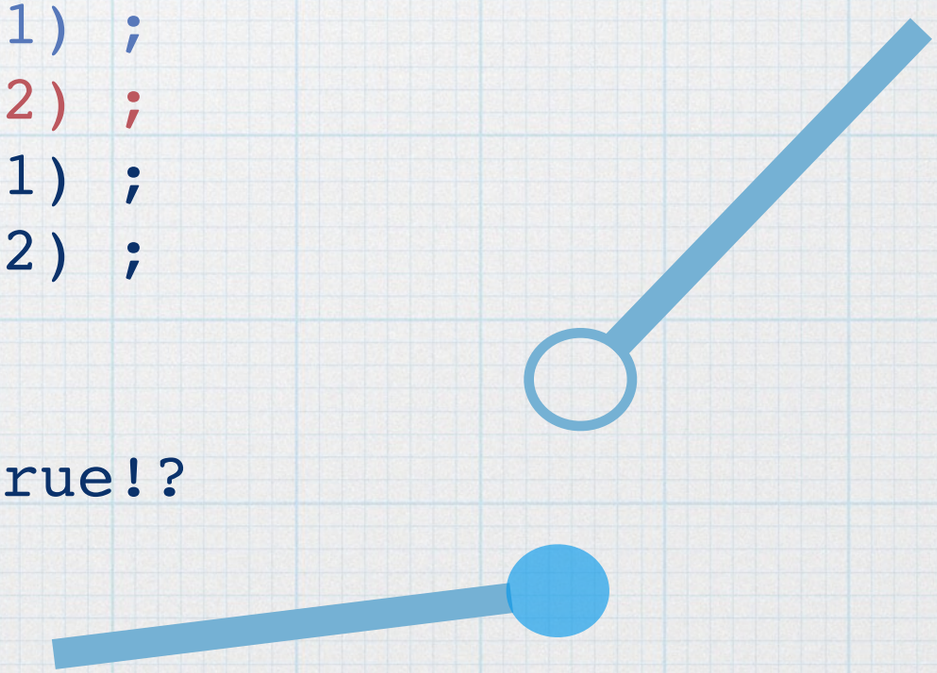





```
szmetsz(szakasz s1,s2) {
    int e=forgasirany(s1.p1,s1.p2,s2.p1) ;
    int f=forgasirany(s1.p1,s1.p2,s2.p2) ;
    int g=forgasirany(s2.p1,s2.p2,s1.p1) ;
    int h=forgasirany(s2.p1,s2.p2,s1.p2) ;
    if (e*f<0 and g*h<0) return true ;

    ... // és ha rajta van is return true!?

    return false ;
}
```

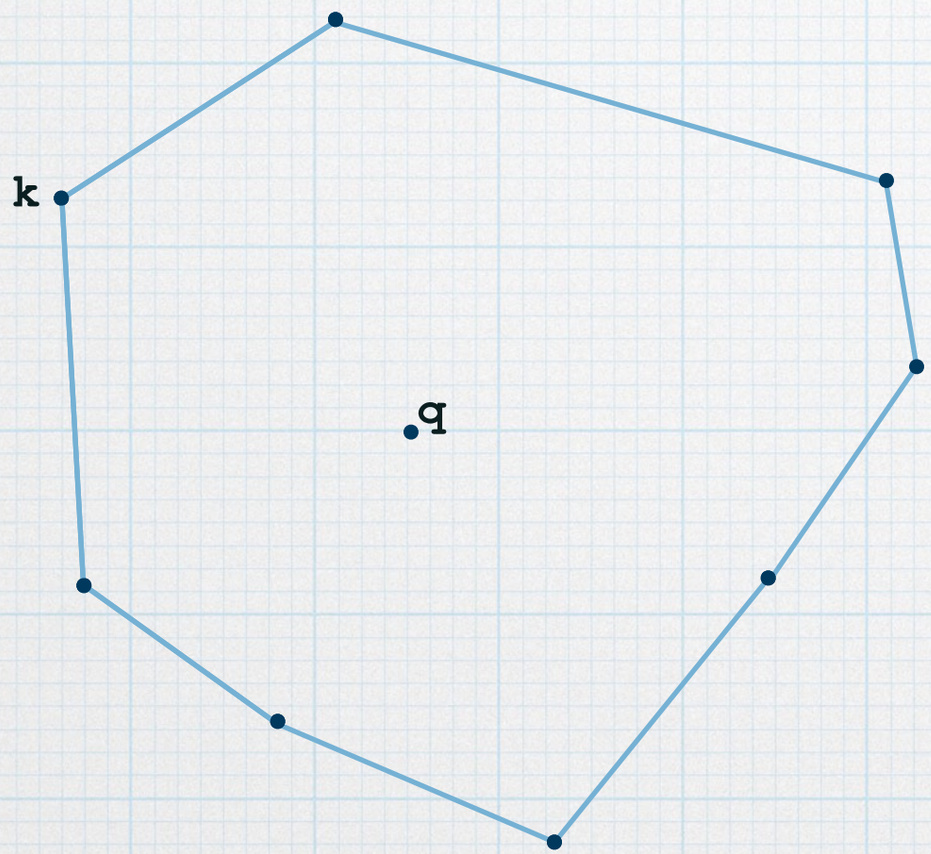


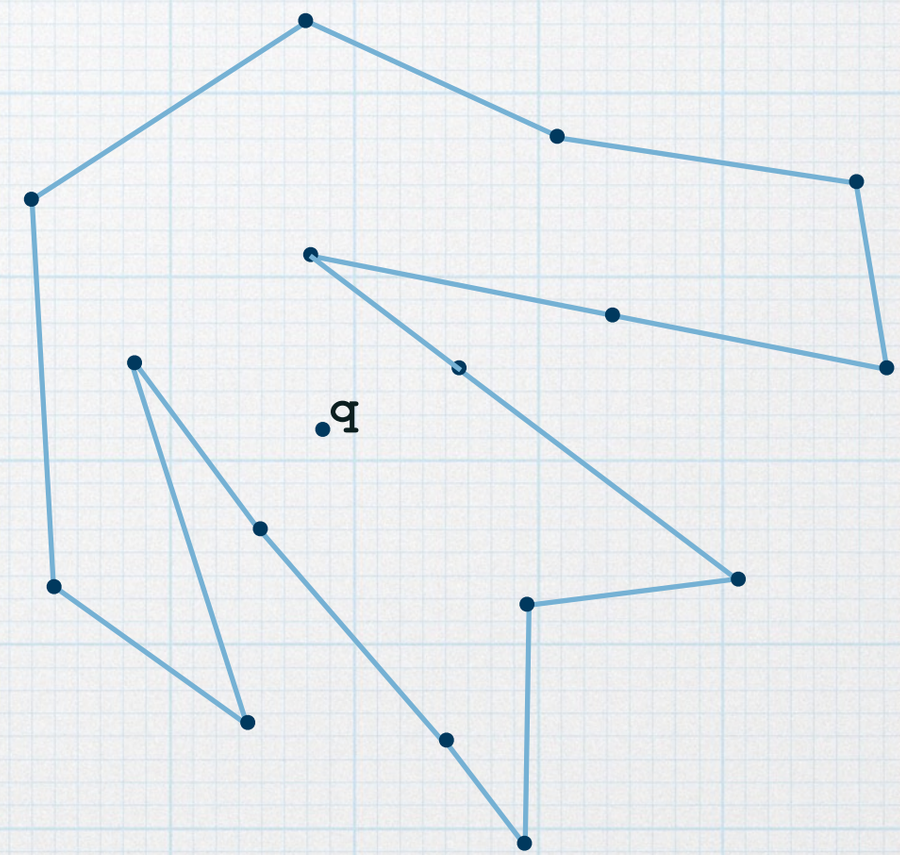
Feladat

Pont helyzetének meghatározása

Adott a síkon egy zárt, nem-metsző poligon a csúcsainak $\{p_1, \ldots, p_n\}$ órajárással ellentétes felsorolásával és adott egy q pont.
Eldöntendo a q pontnak a poligonhoz viszonyított helyzete, azaz, hogy a belsejéhez tartozik-e, az oldalán van-e, avagy külső pont?

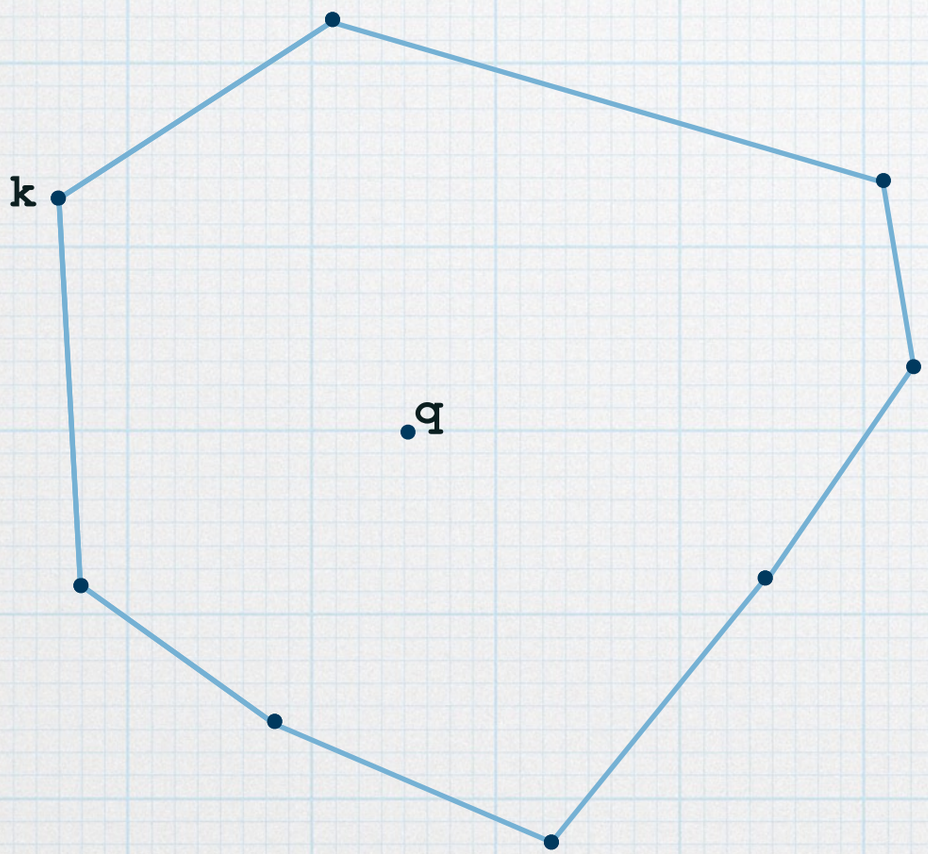
Megoldás



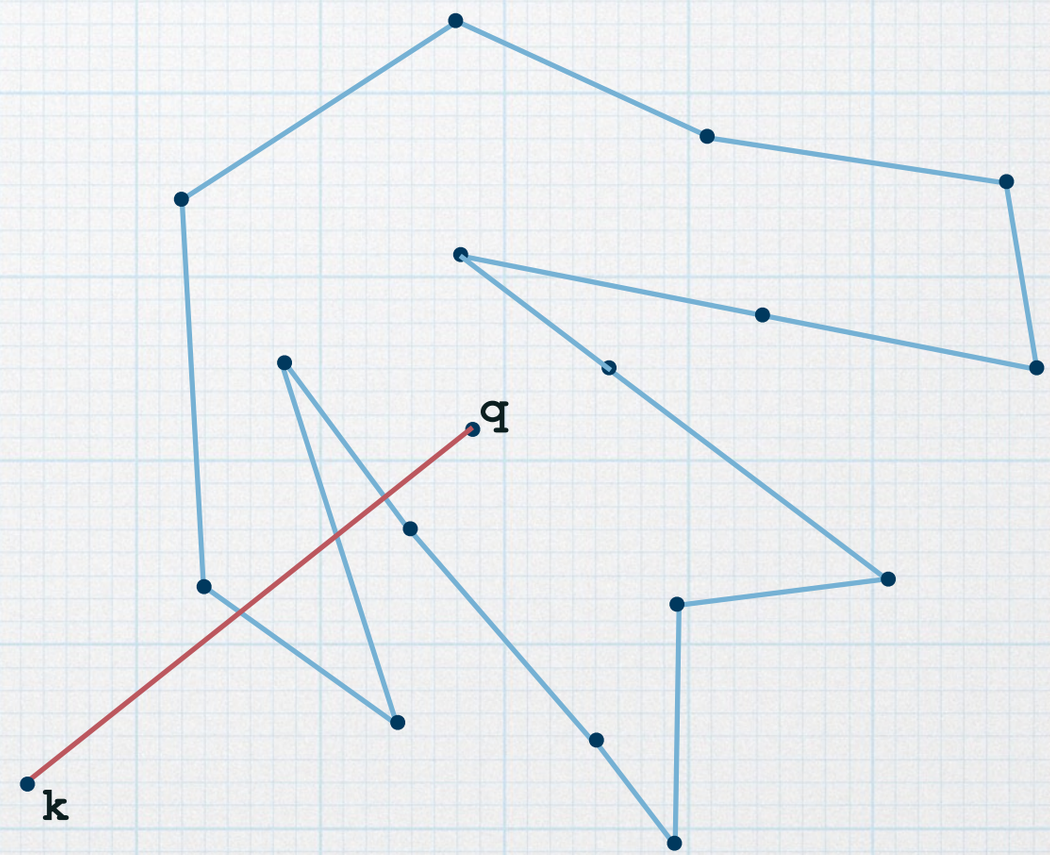


O(n)

Megoldás



O(n)



O(n)

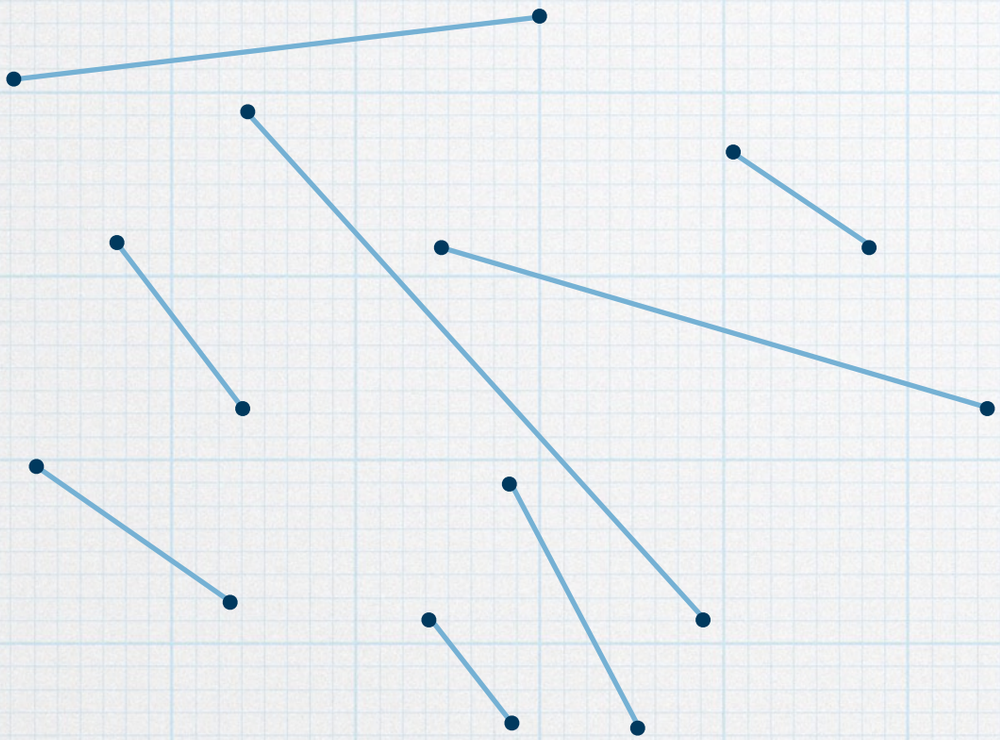
Feladat

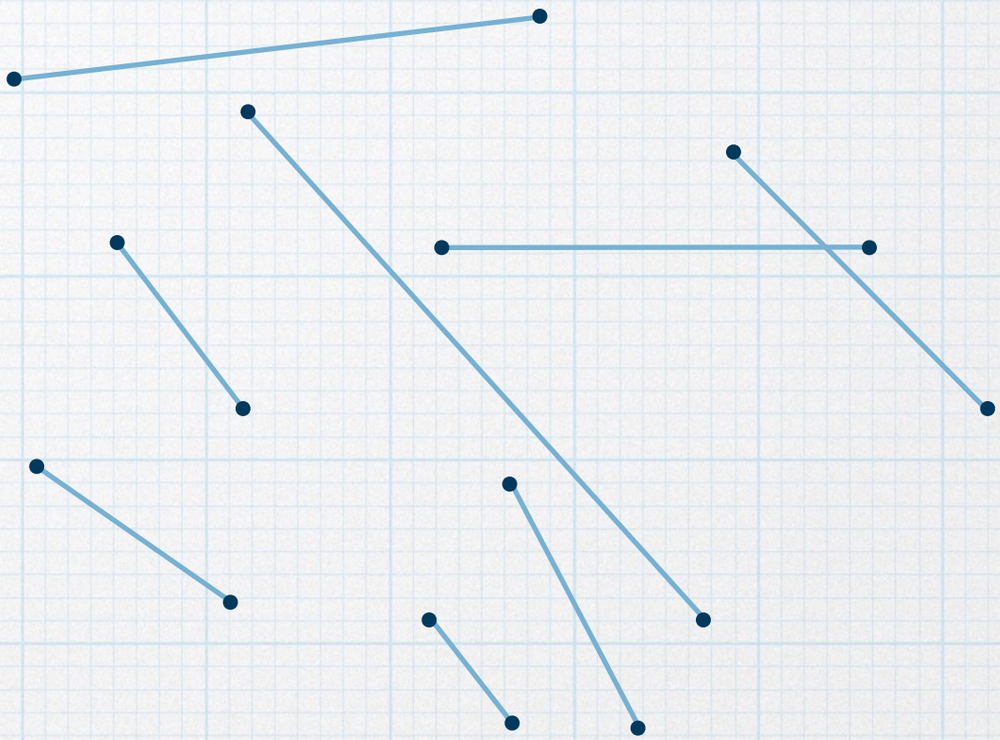
Metsző szakaszpárok keresése

A feladatban adott n darab szakasz $\{s_1, \ldots, s_n\}$ (a szakaszok az s_i.bal és s_i.jobb végpontjaik által adottak) és meg kell határozni vane-e közöttük kettő olyan, amely metszi egymást.

Egyszerűsítő feltételek:
1. Egyik bemeneti szakasz sem függőleges.
2. Nincs három olyan szakasz, amelyek egy pontban metszenék egymást

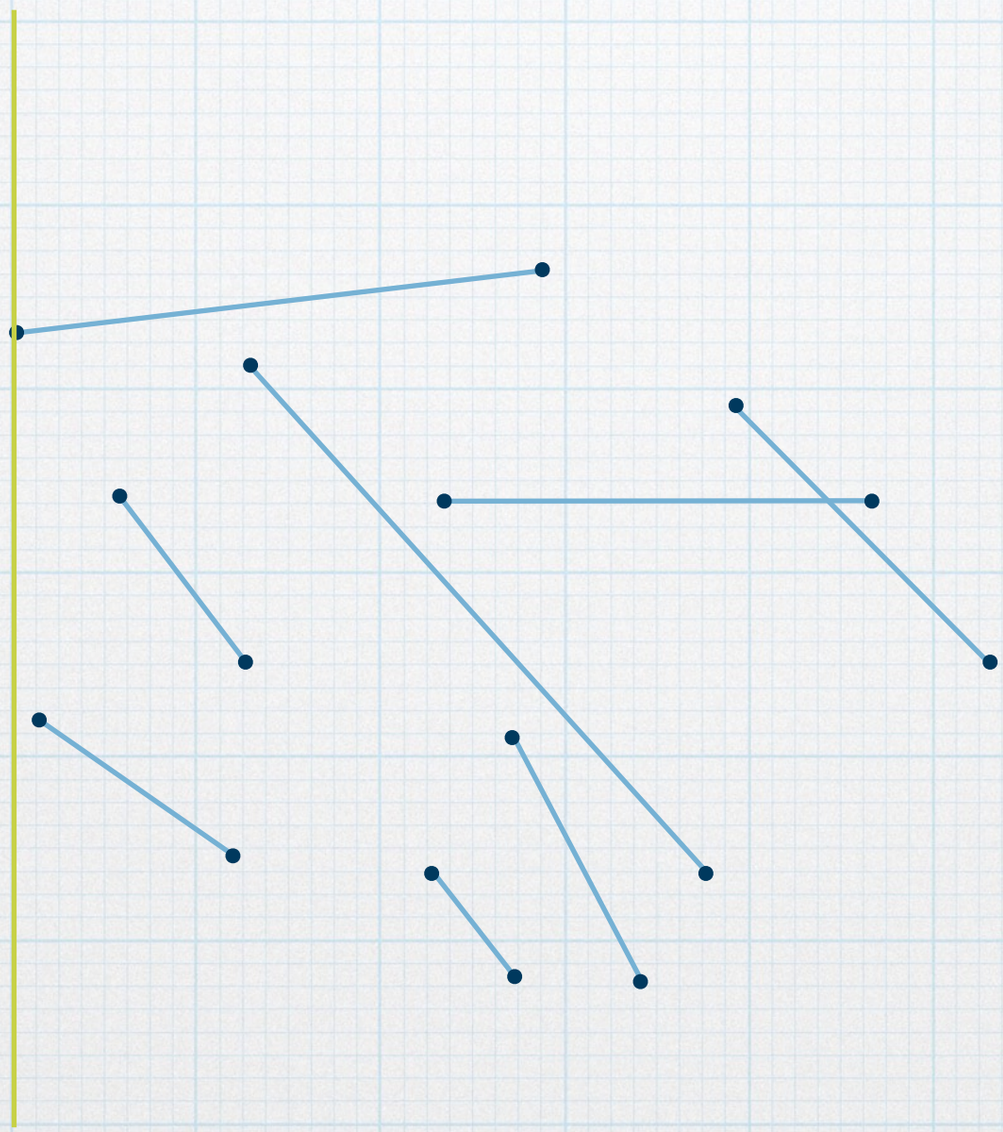
Megoldás



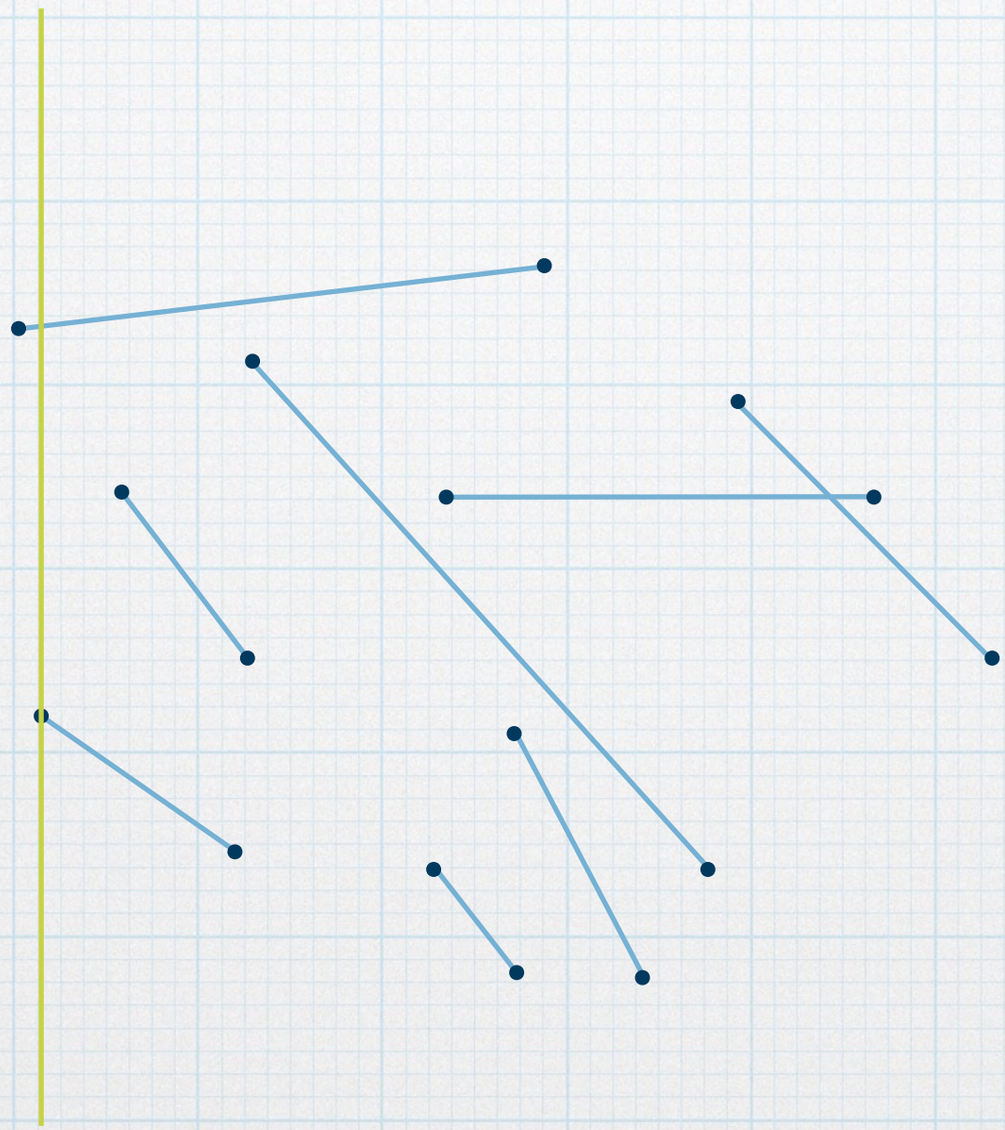


Seprő egyenes
Seprés során egy képzeletbeli seprő egyenest mozgatunk a geometriai elemek halmazán. Azt a dimenziót, amelyben a seprő egyenes halad, idődimenziónak hívjuk.
Jelen esetben a seprő egyenes az y-tengellyel párhuzamos egyenes, az idodimenzió az x-koordináta lesz.

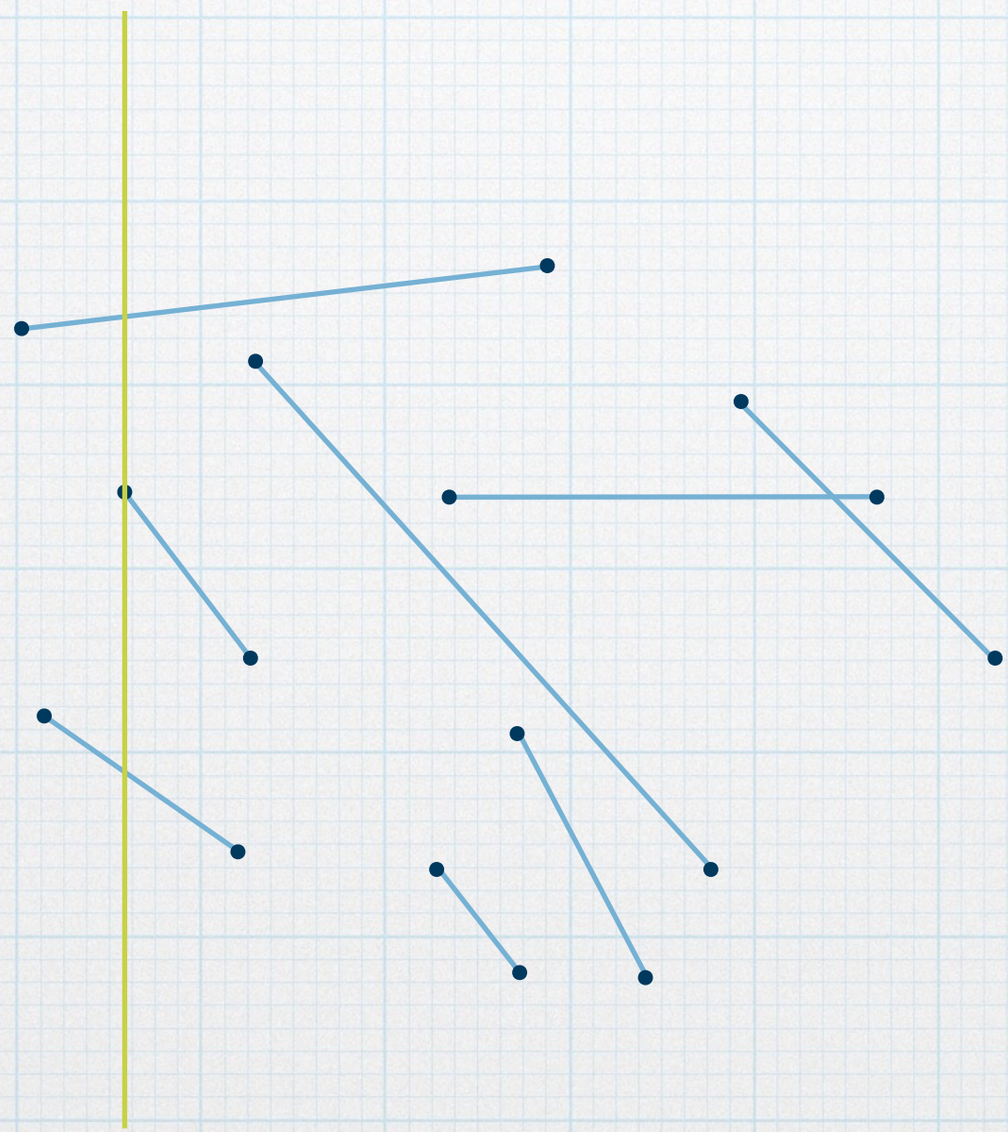
Megoldás



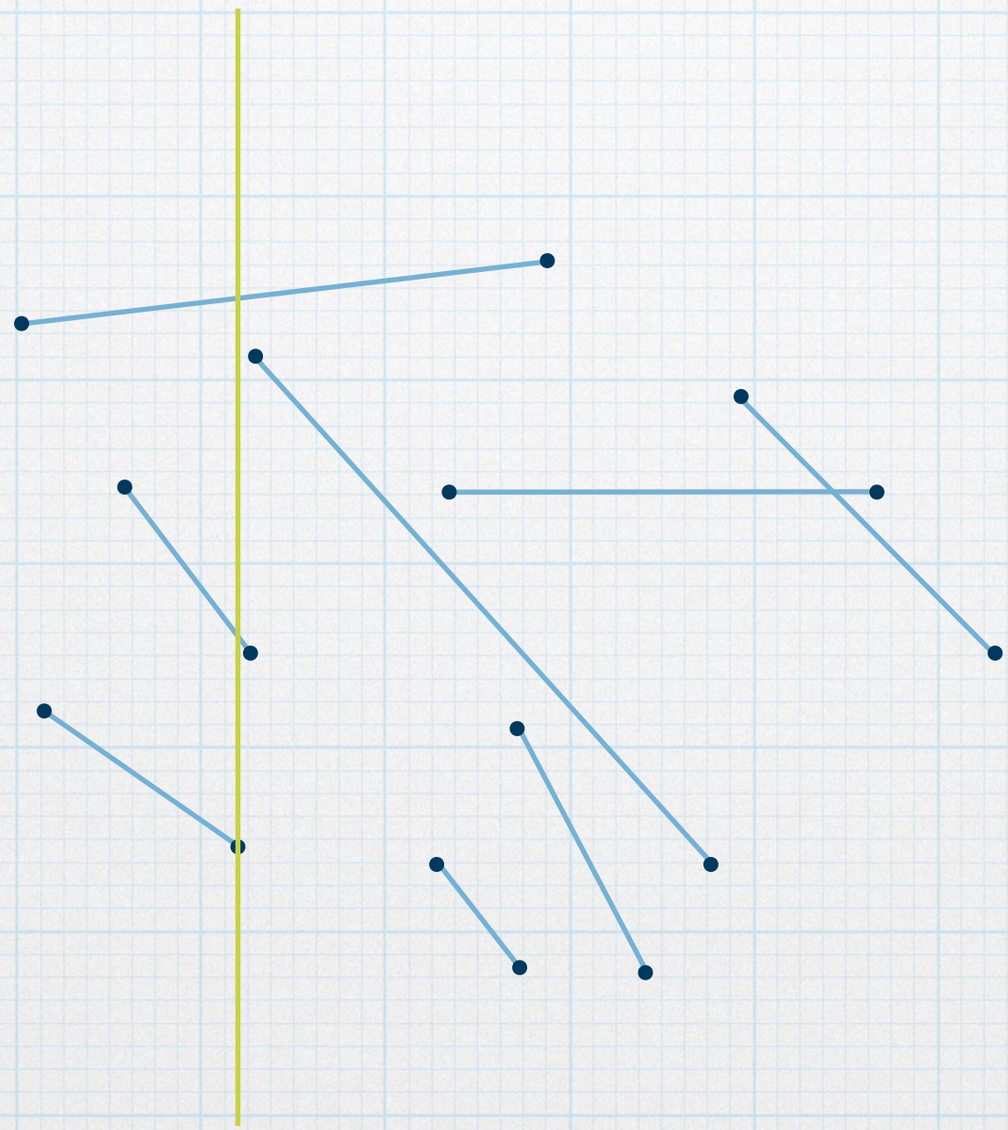
Megoldás



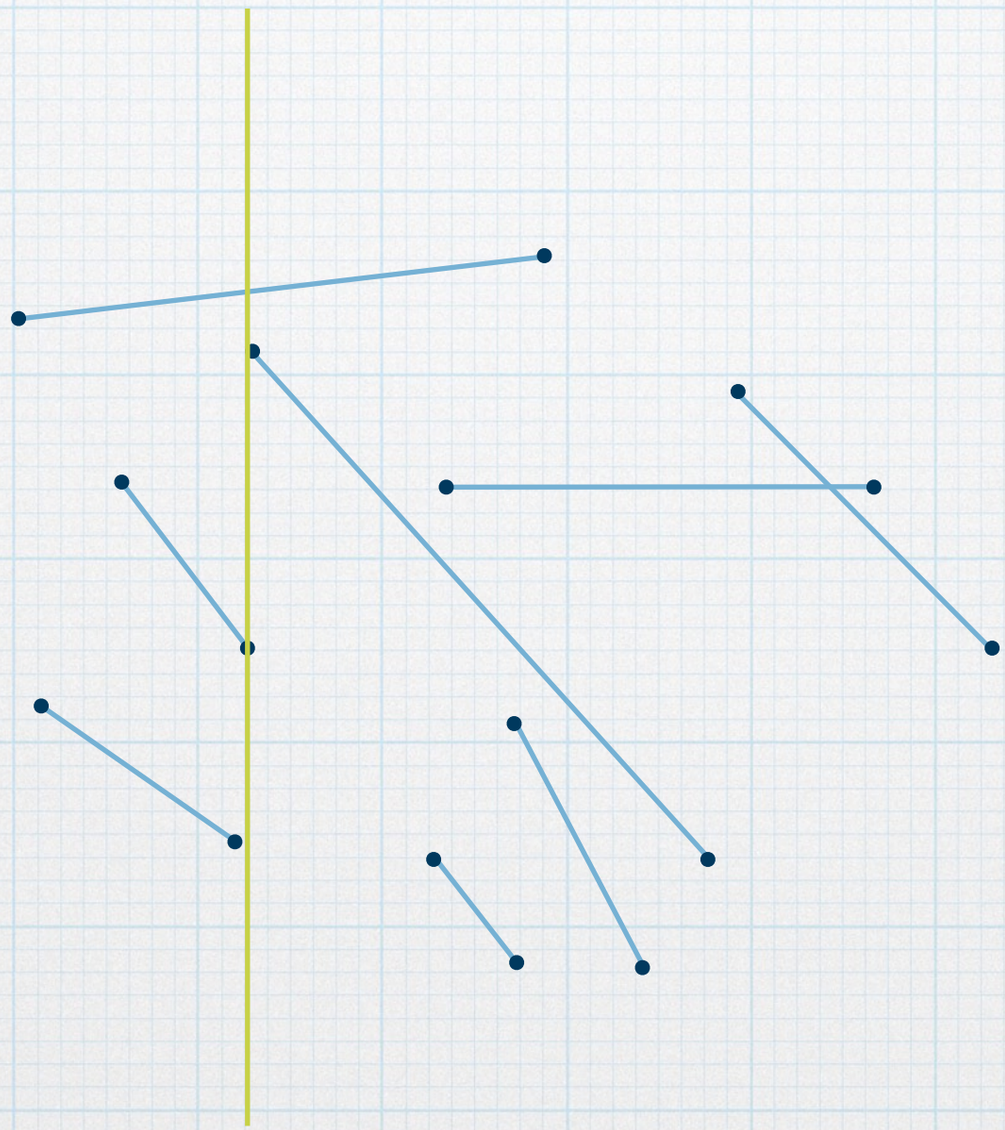
Megoldás



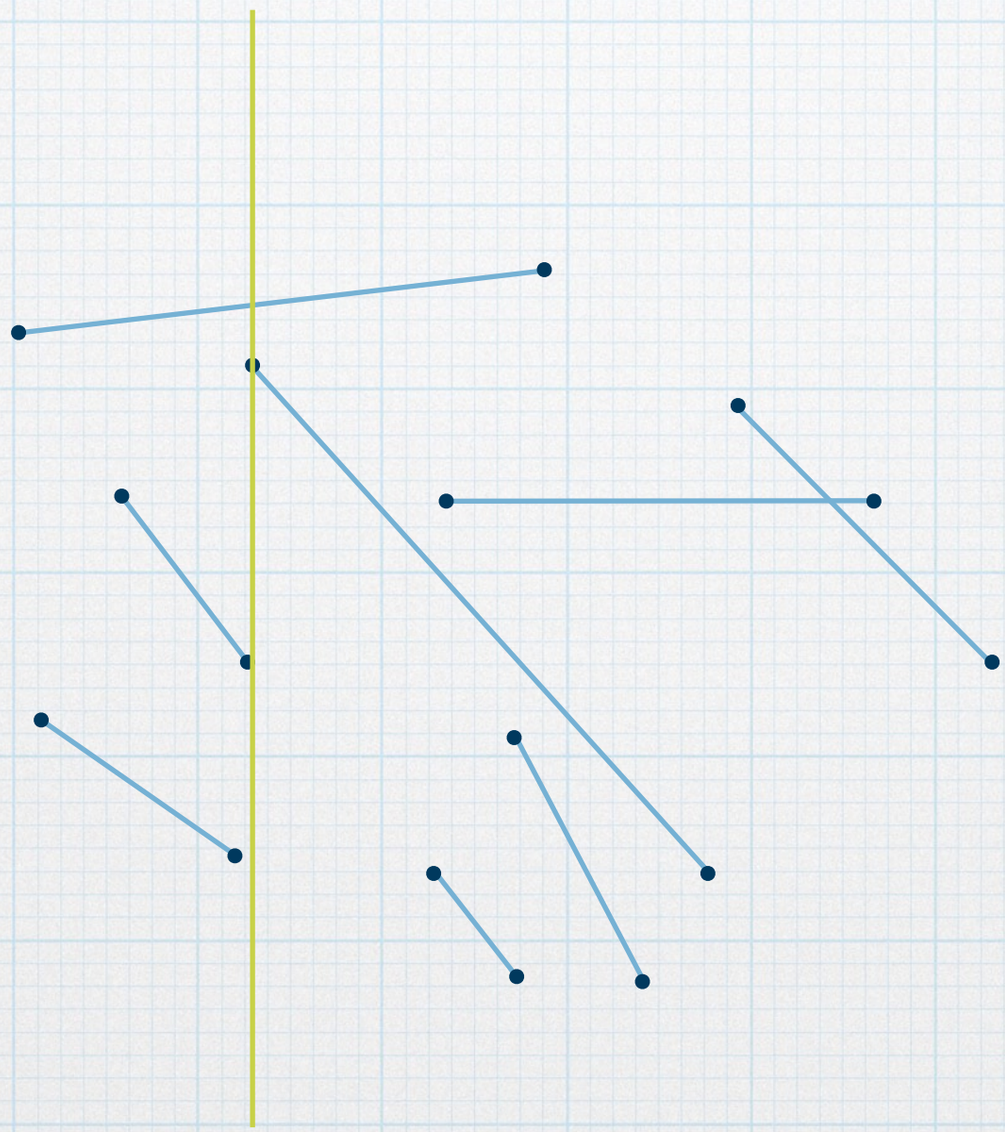
Megoldás



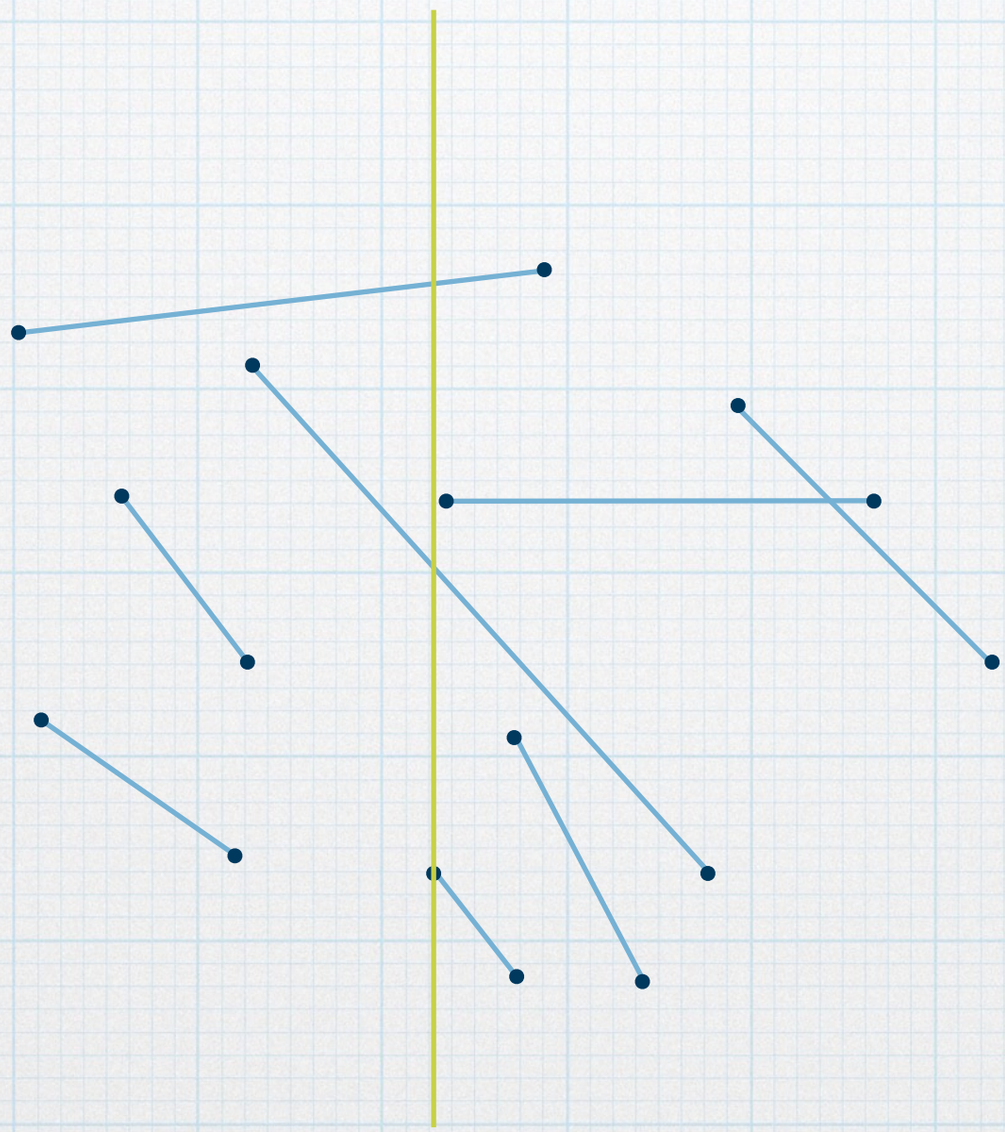
Megoldás



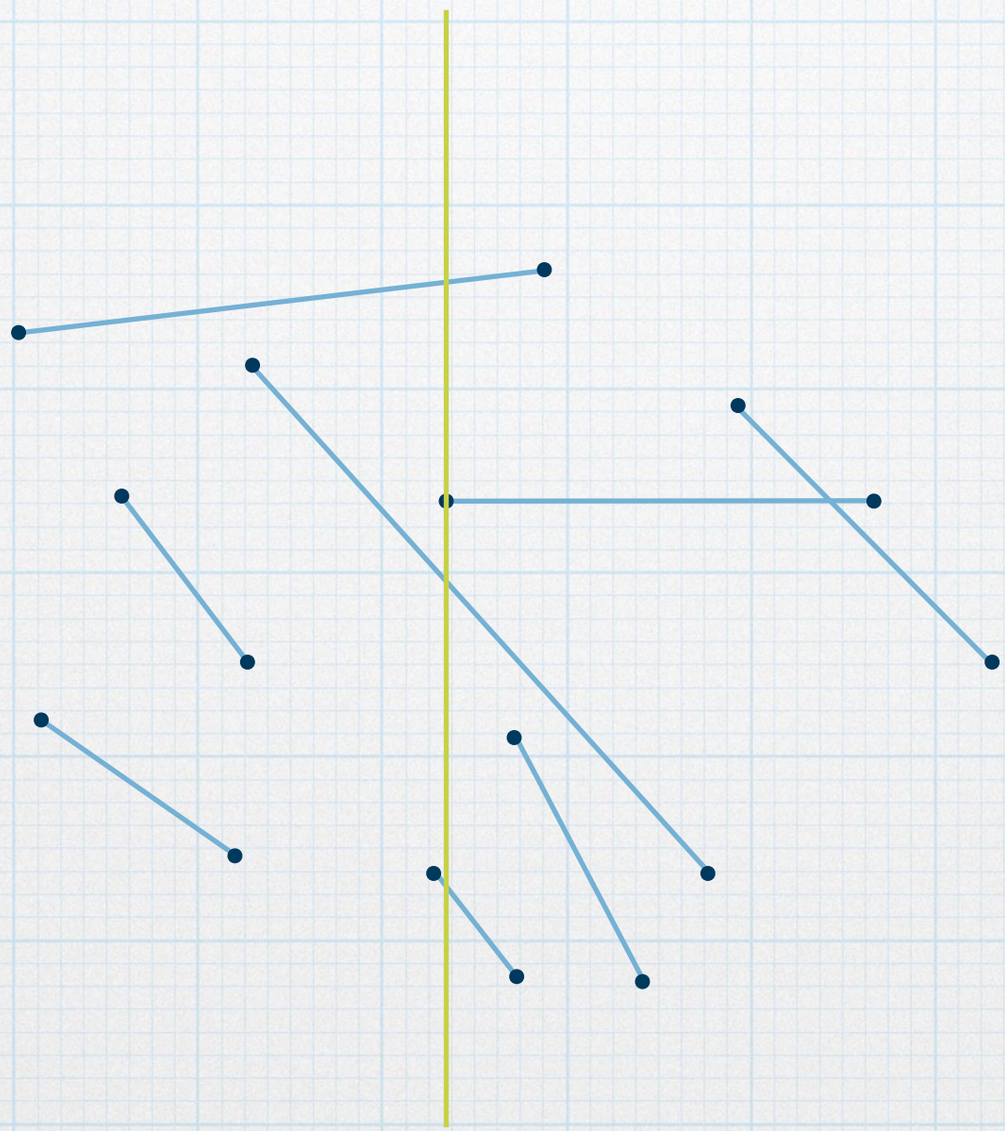
Megoldás



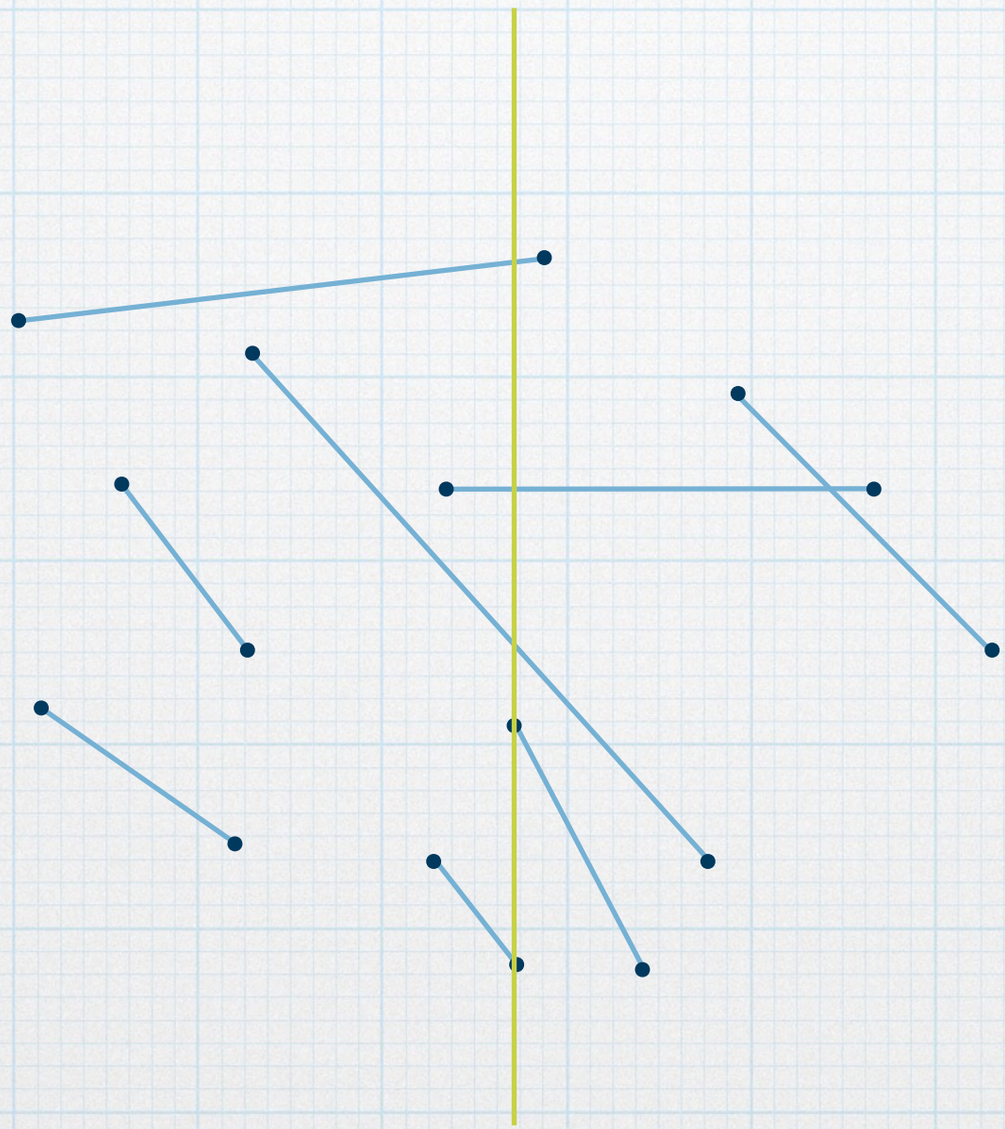
Megoldás



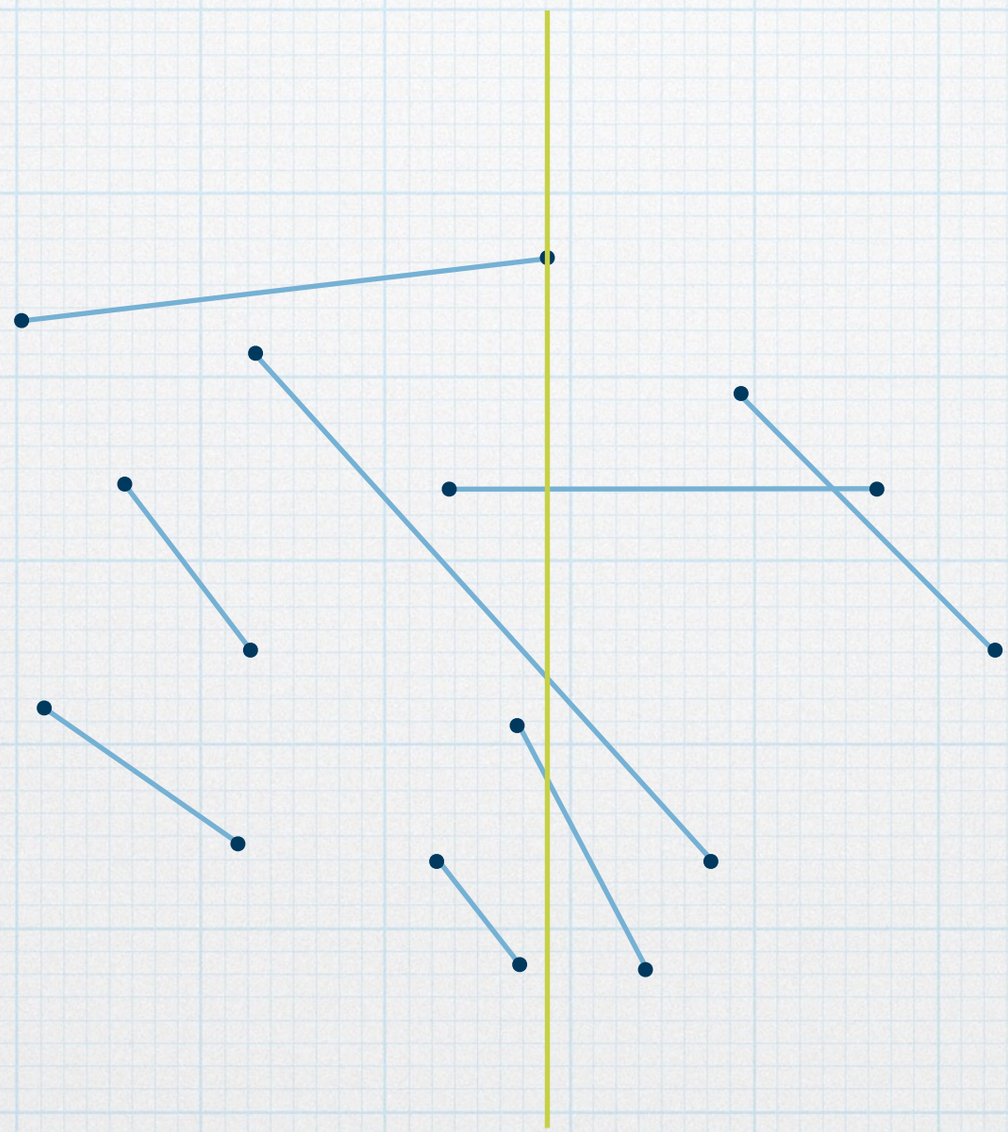
Megoldás



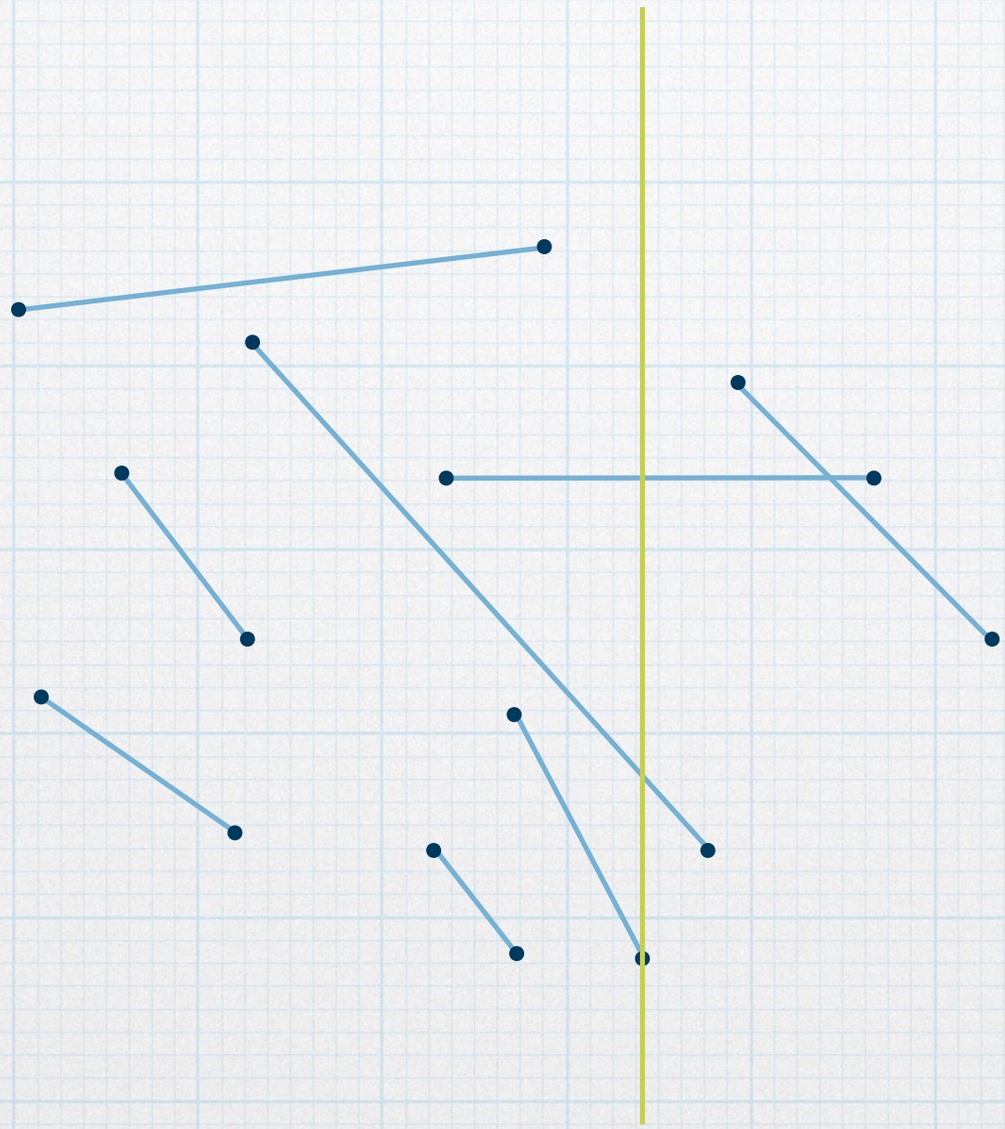
Megoldás



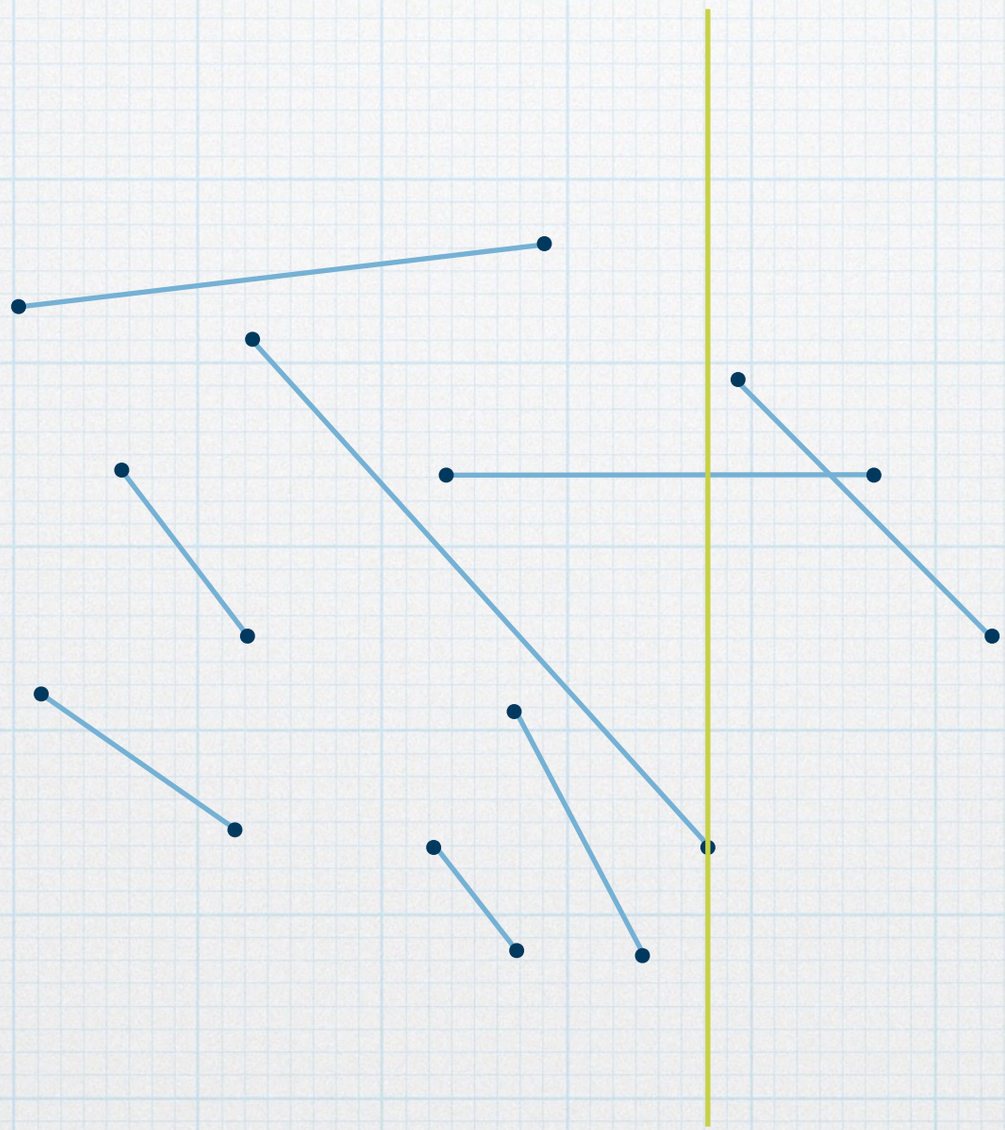
Megoldás



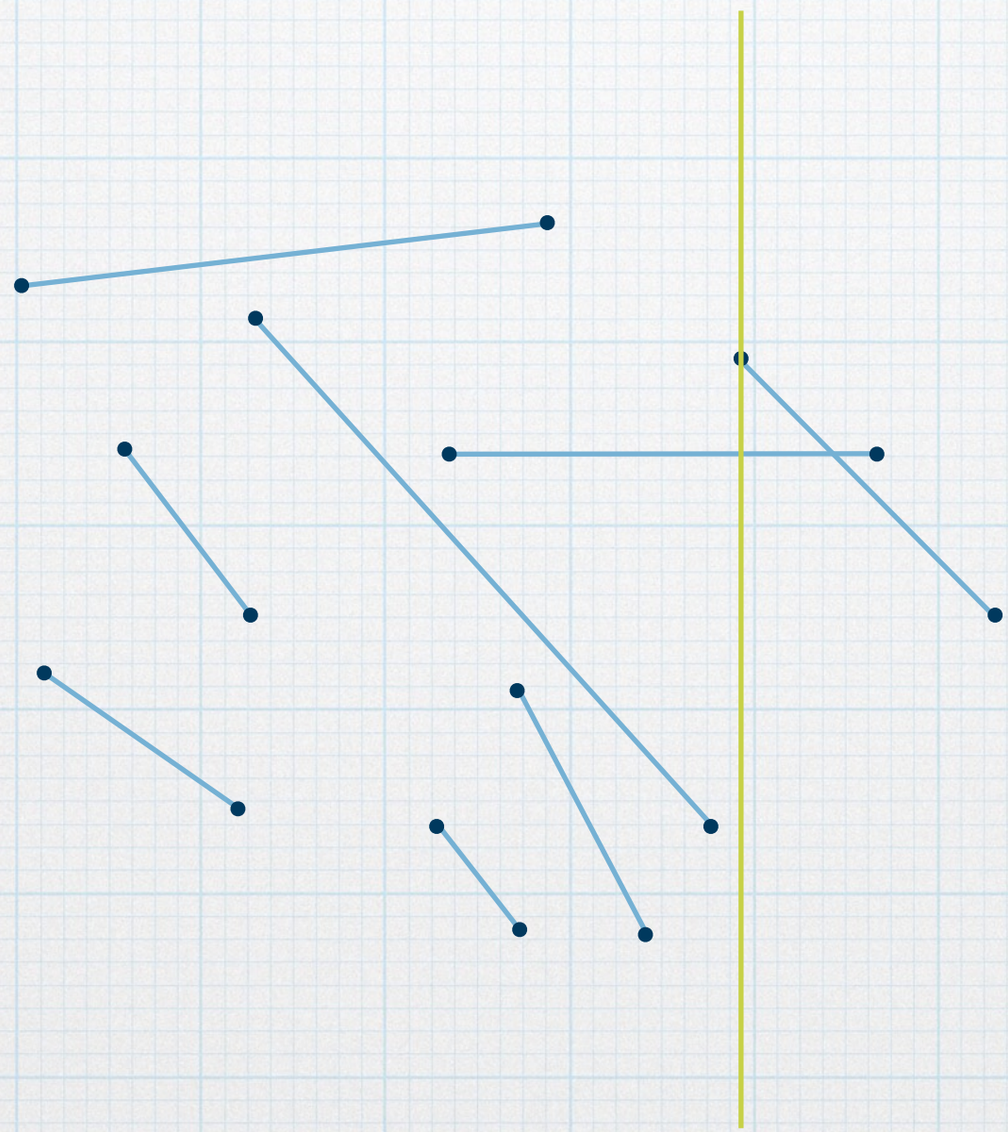
Megoldás



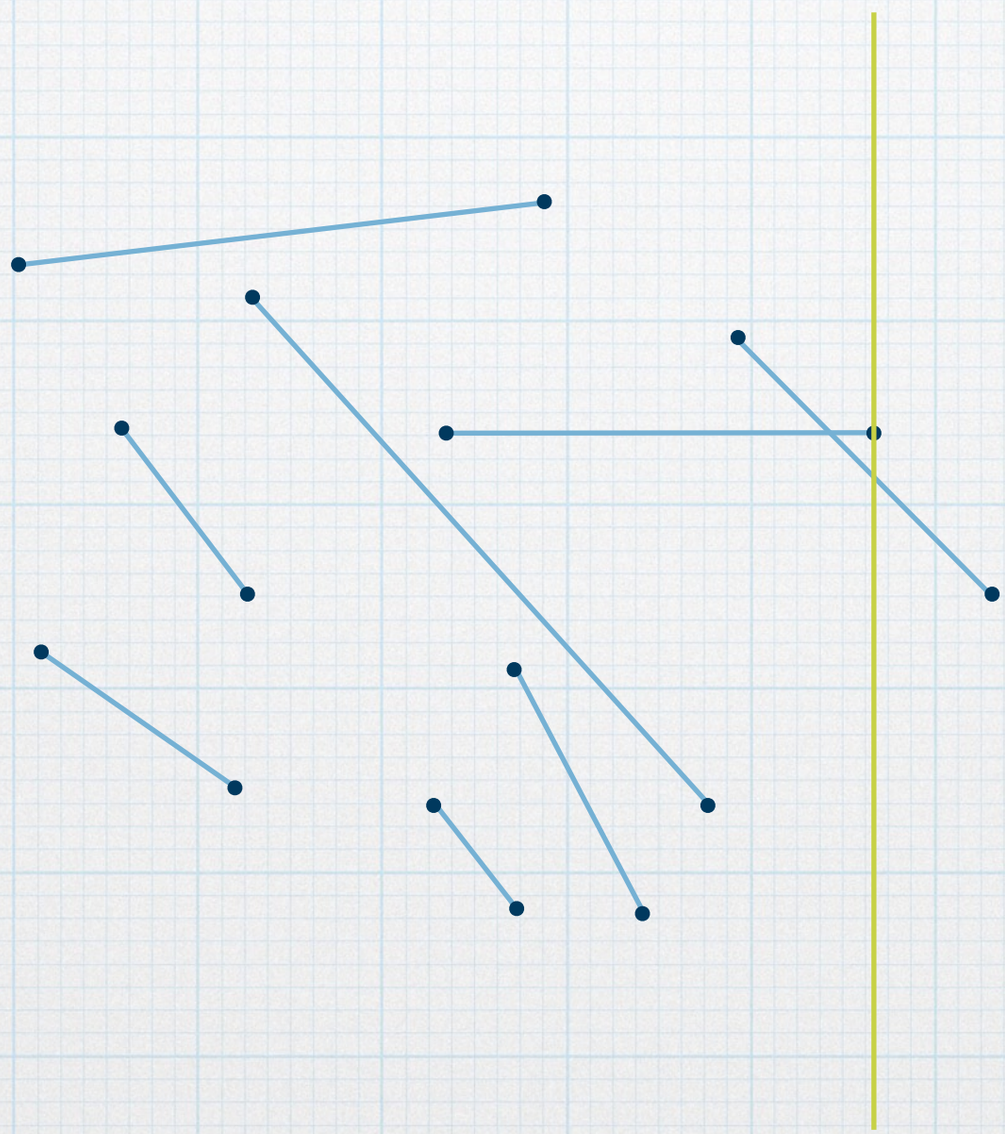
Megoldás



Megoldás



Megoldás



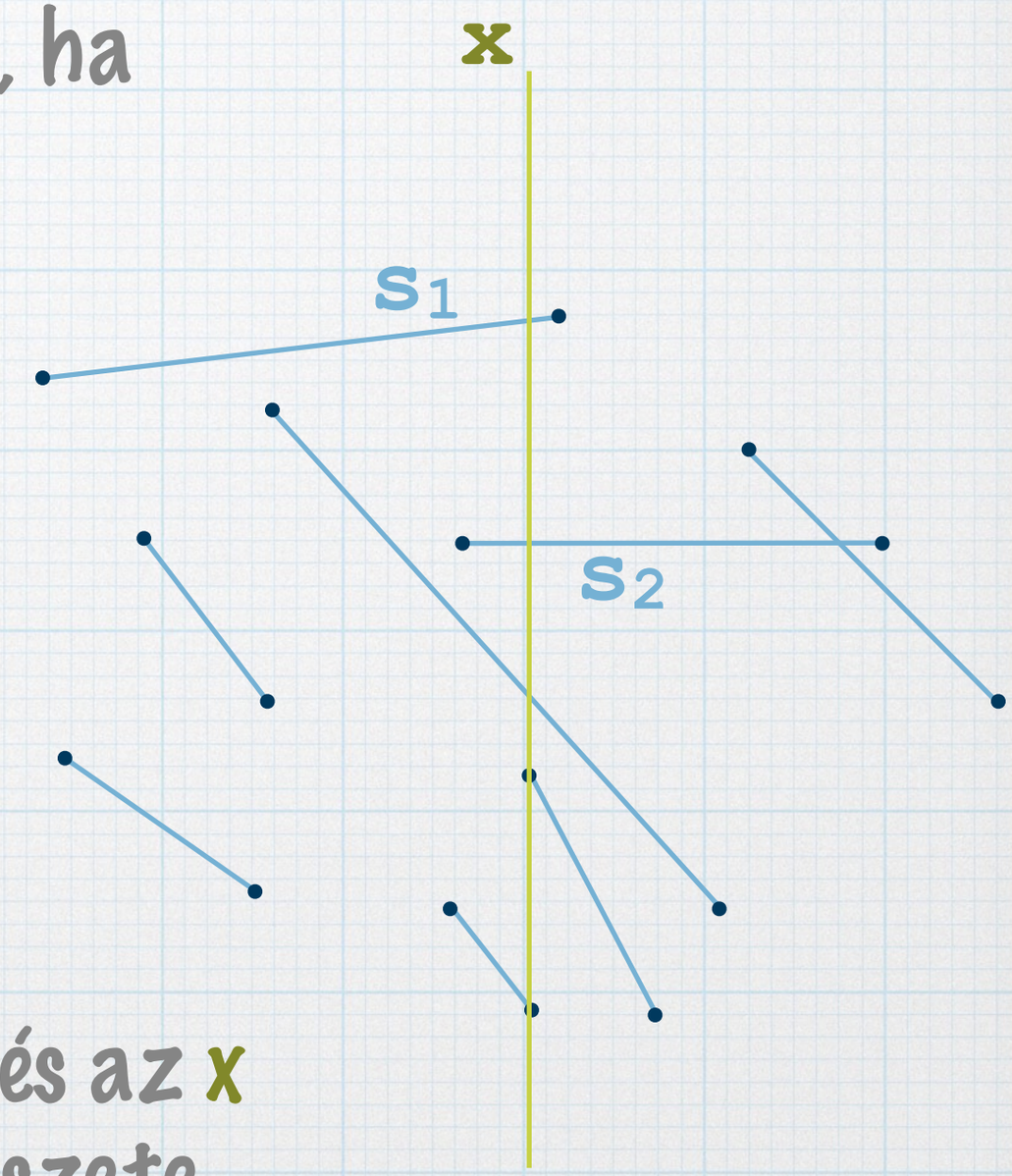
Legyen s_1 és s_2 szakasz és $x \in R$

s_1 összehasonlítható s_2-vel x-nél, ha

$s_1.bal.x \leq x \leq s_1.jobb.x$

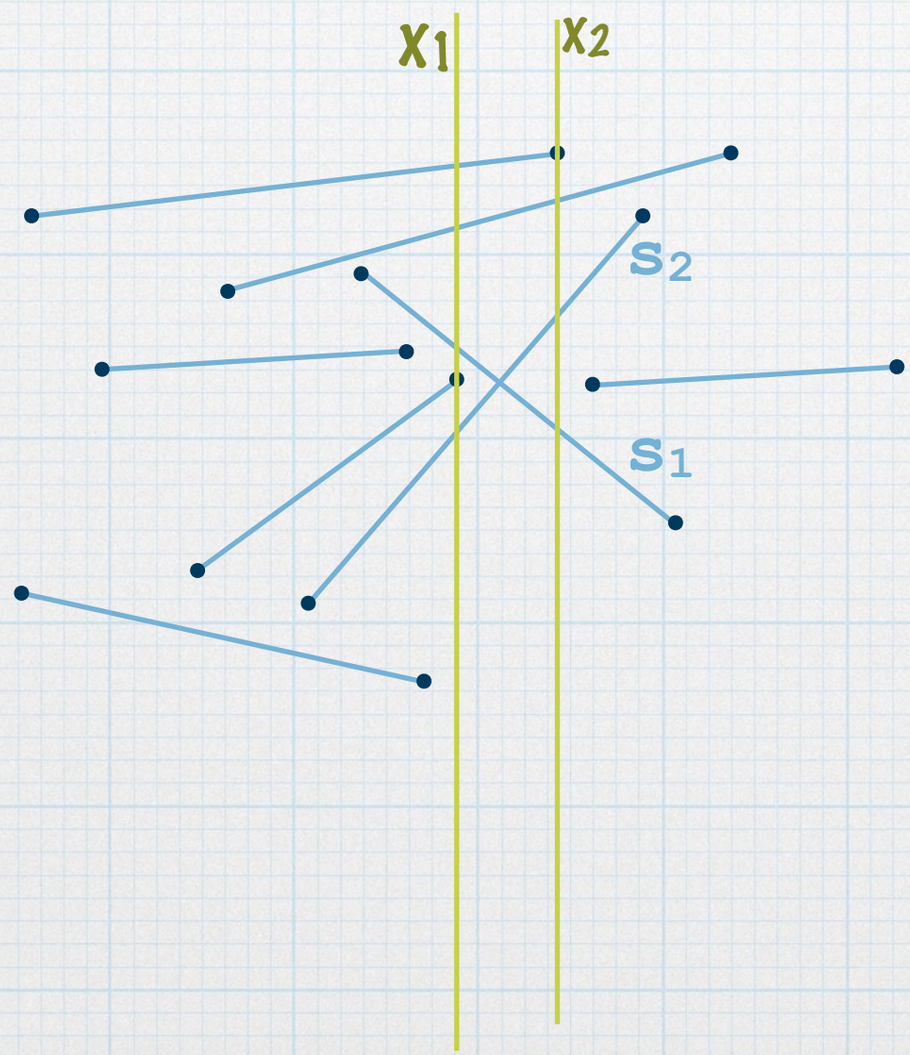
és

$s_2.bal.x \leq x \leq s_2.jobb.x$

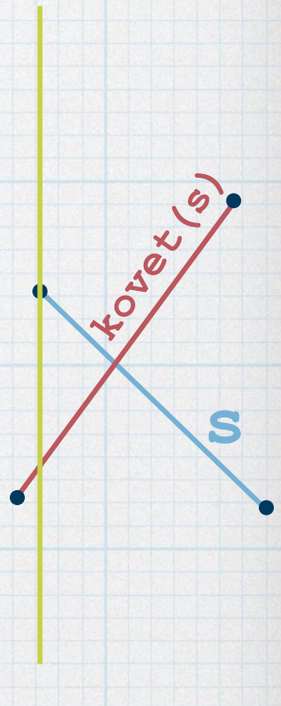


Az s_1 szakasz felette van az s_2 szakasznak, ($s_1 >_x s_2$), ha s_1 összehasonlítható s_2-vel x-nél és az x seperő egyenesnek s_1-el vett metszete magasabban van, mint s_2-vel vett metszete.

Ha $x_1 < x_2$-re $s_1 >_{x_1} s_2$ és $s_2 >_{x_2} s_1$, akkor az s_1 és s_2 szakasz metszi egymást x_1 és x_2 között. Továbbá, ha x_1 és x_2 a seprő egyenes két egymást követő megállási helye, akkor x_1-ben (után) a két szakasz egymást követő lesz a rendezésben.

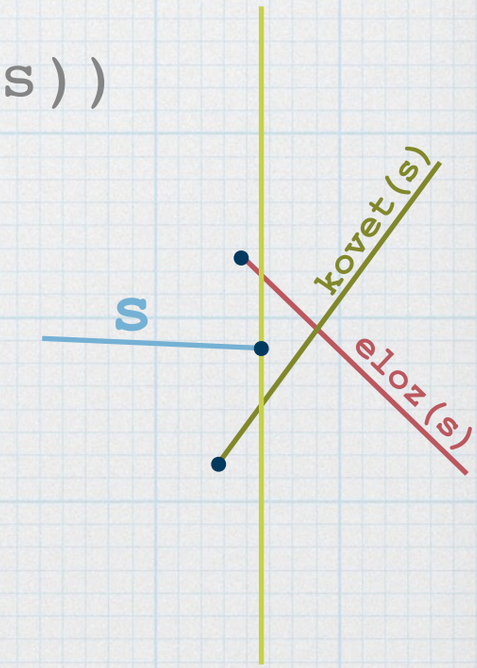


Tehát elegendő ellenőrizni minden szakasz berakásakor, hogy a rendezésben őt megelőző, és követő szakasz metszi-e a berakottat, és kivételkor ellenőrizni, hogy a kivett szakaszt megeloző és követő szakaszok metszik-e egymást.



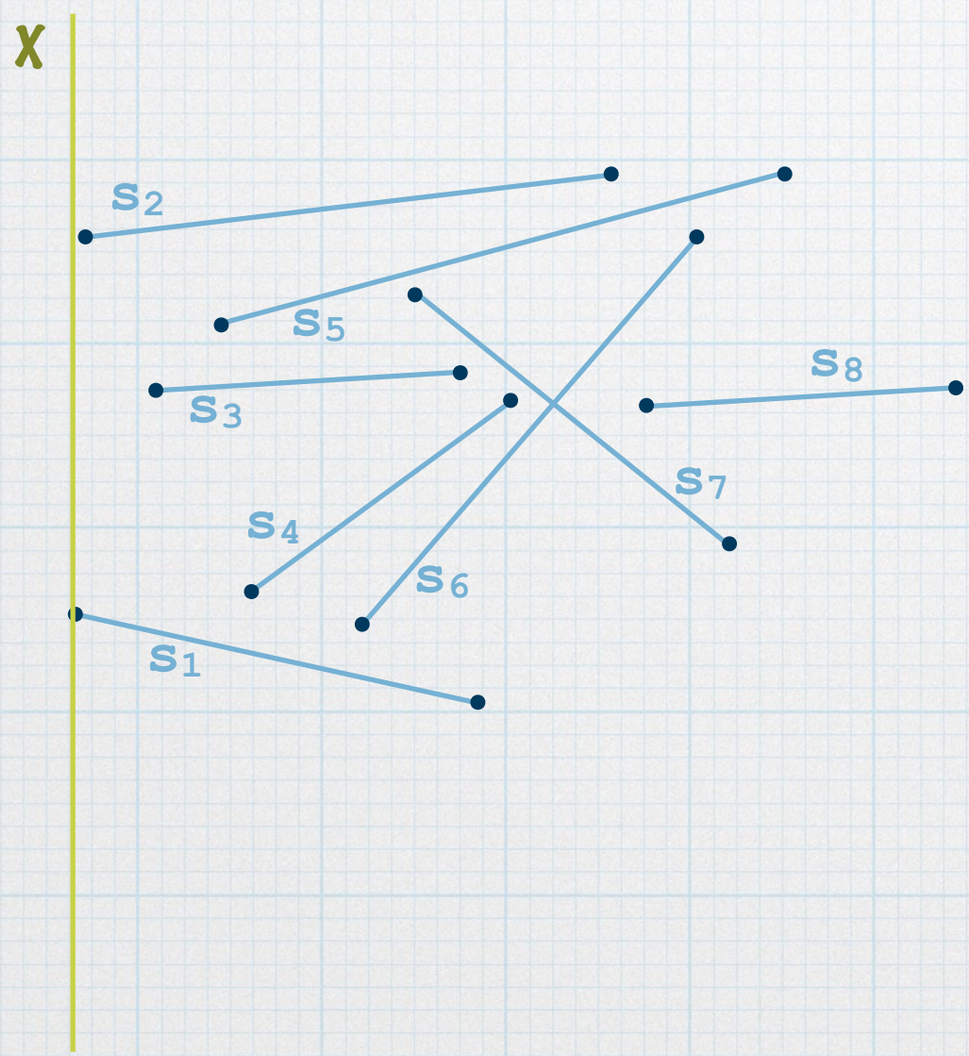
Használjunk Rhalmaz adattípust! (pl. piros-fekete fa)

```
MSZPkeres() {
   minden p-re az R beli sorrendben
   if (p==s.bal) {
      betesz(s) ;
      if (szmetsz(kovet(s),s) or szmetsz(eloz(s),s))
         return True ;
   }
   if (p=s.jobb) {
      if szmetsz(kovet(s),eloz(s)) return True ;
      kivesz(s) ;
   }
}
```



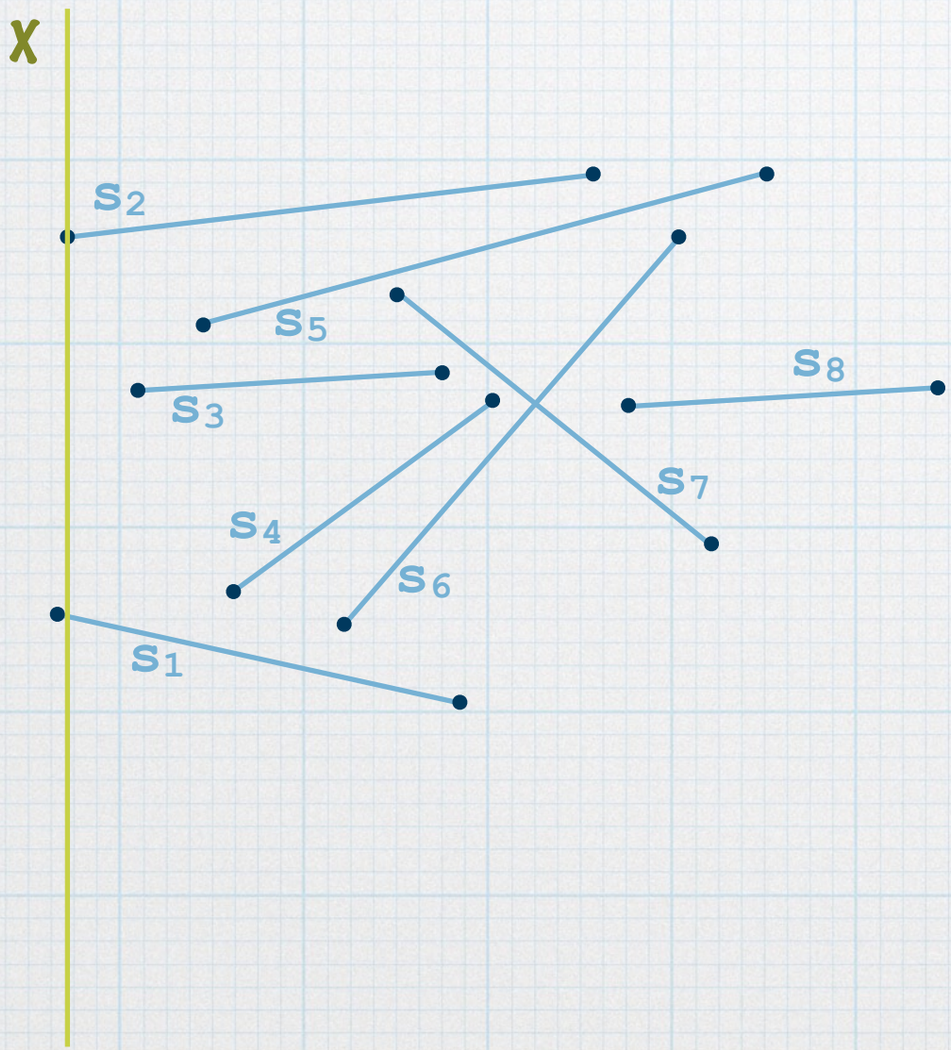
O(nlogn)

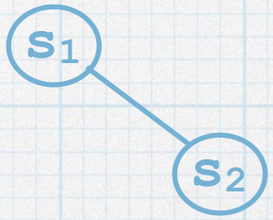
Példa



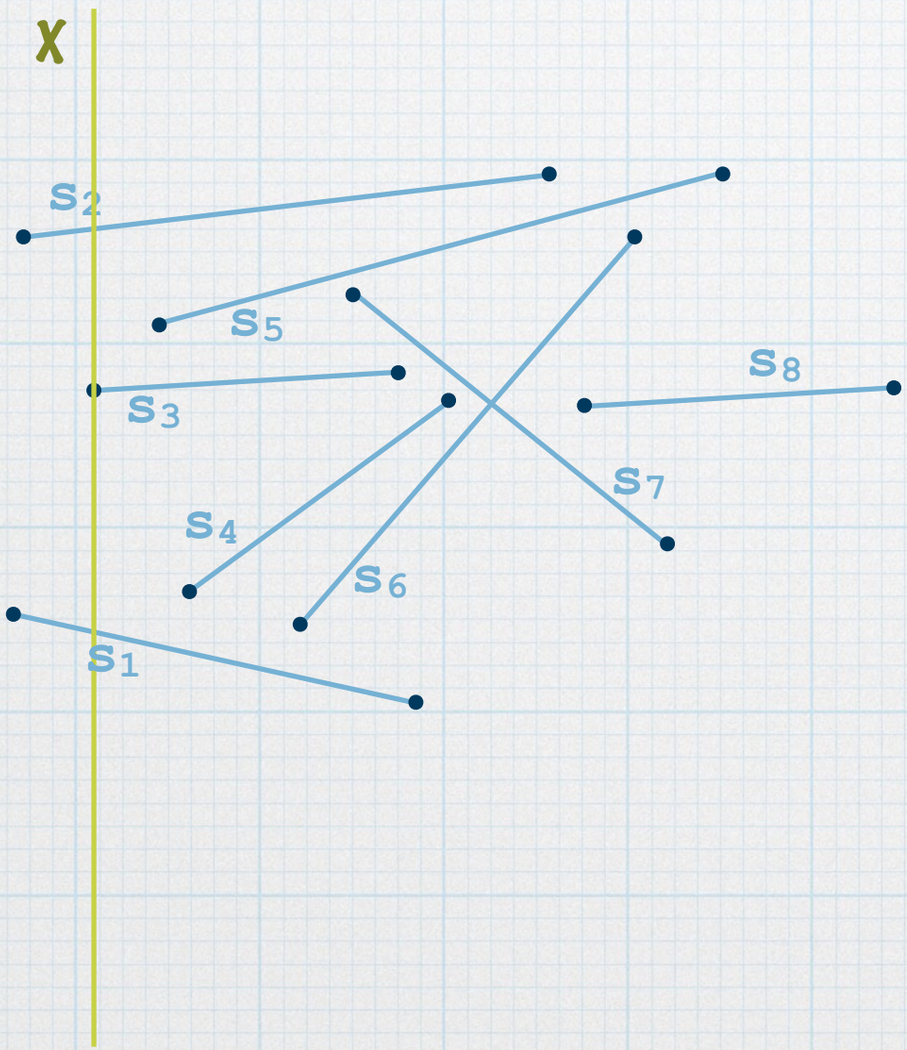
s_1

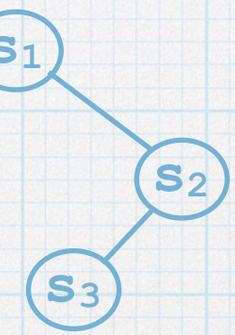
Példa



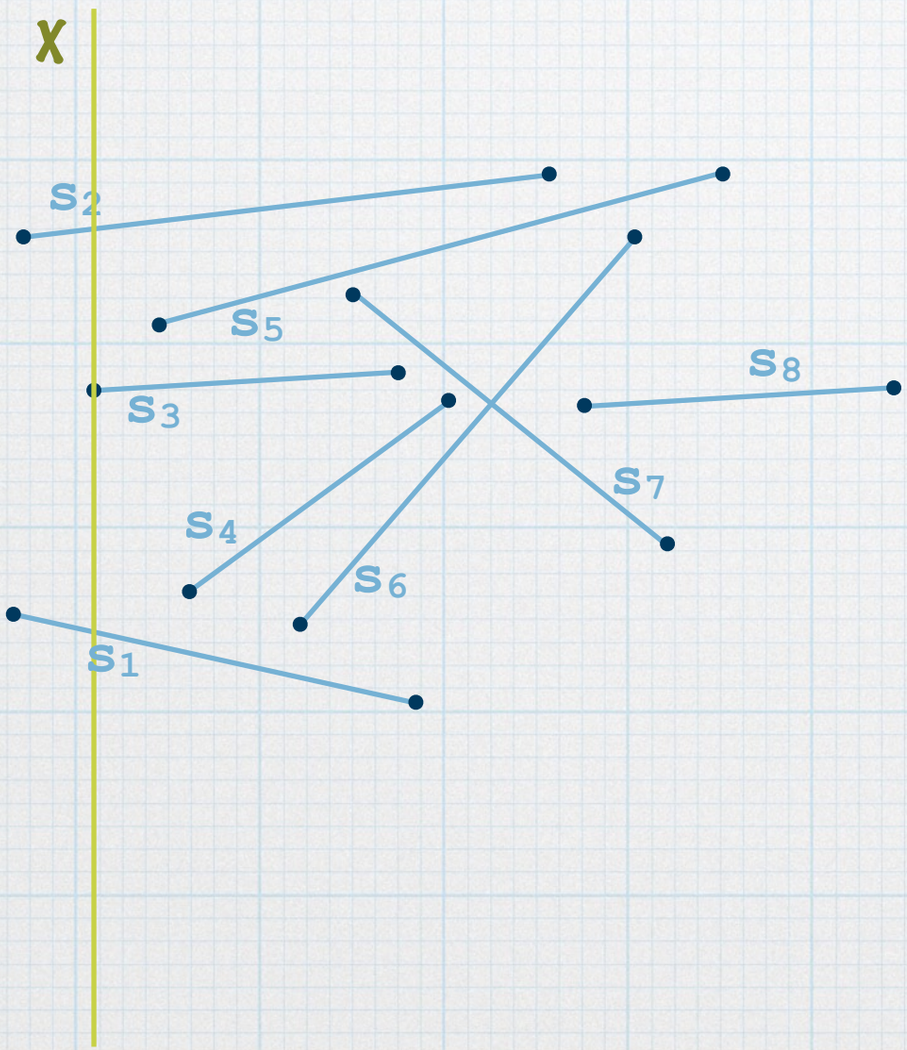


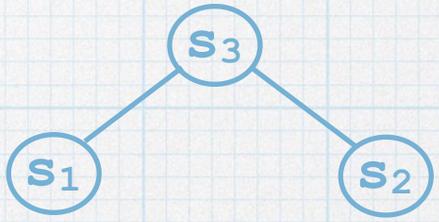
Példa



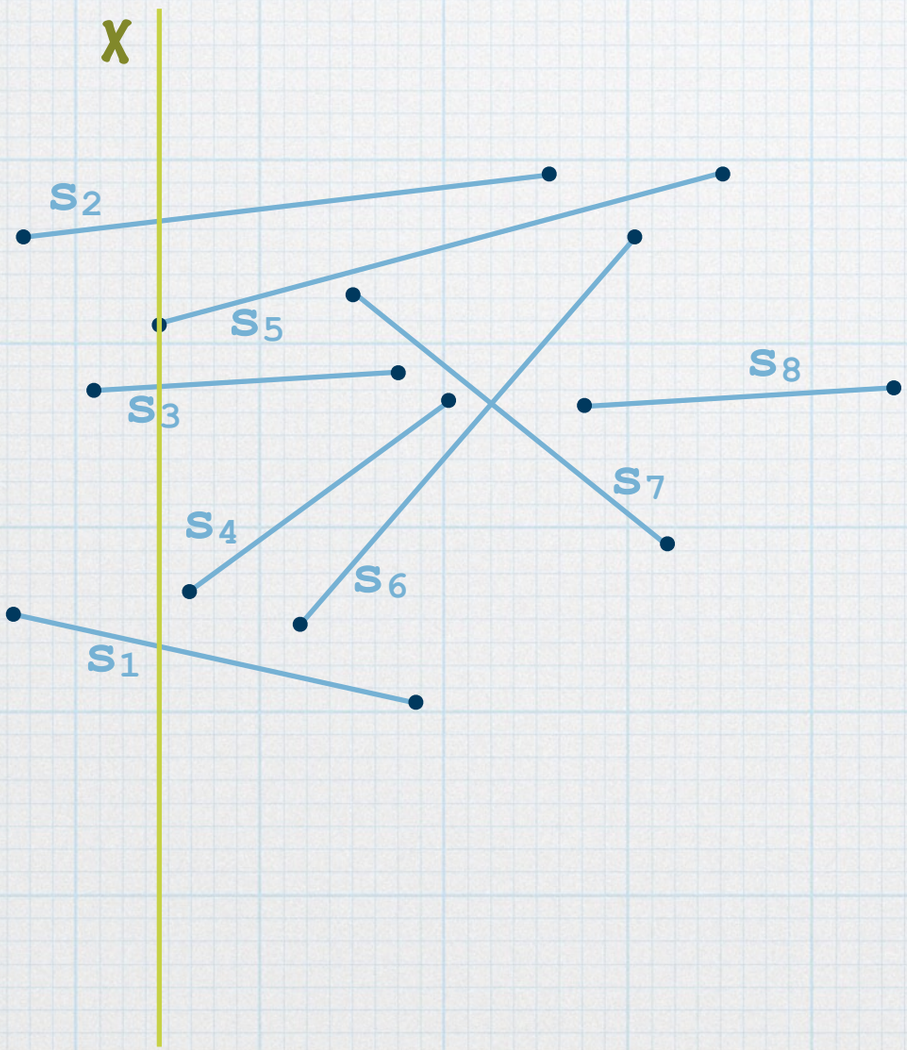


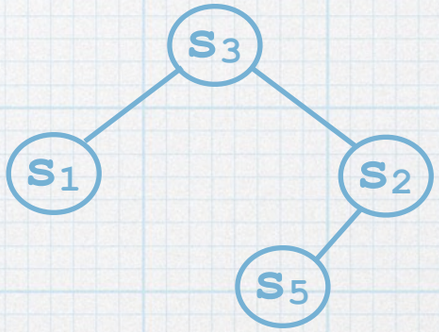
Példa



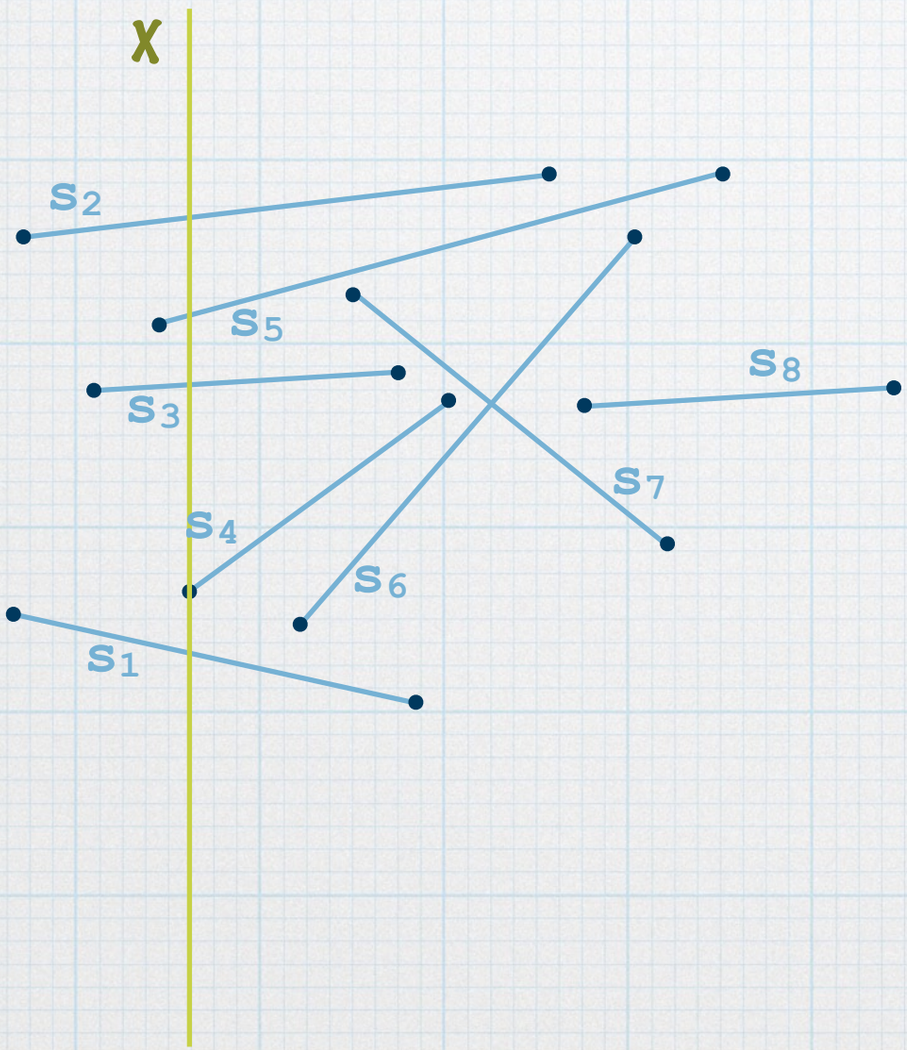


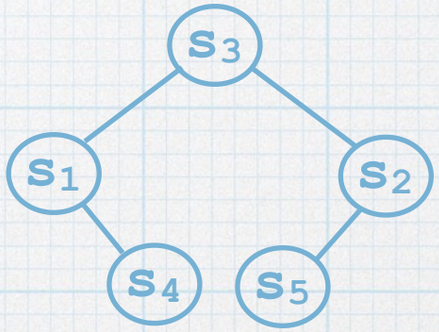
Példa



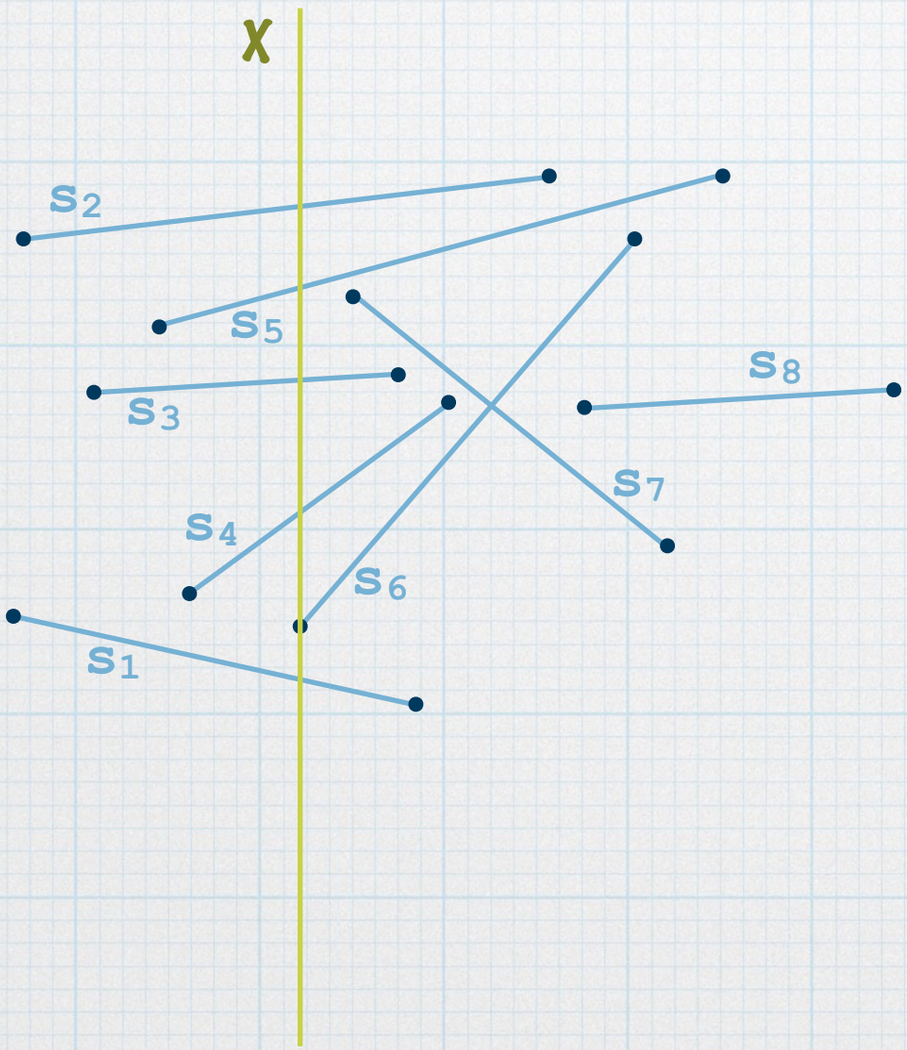


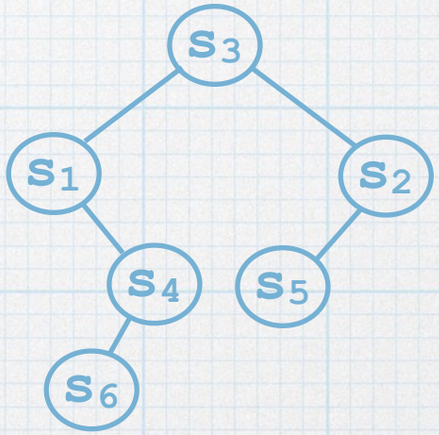
Példa



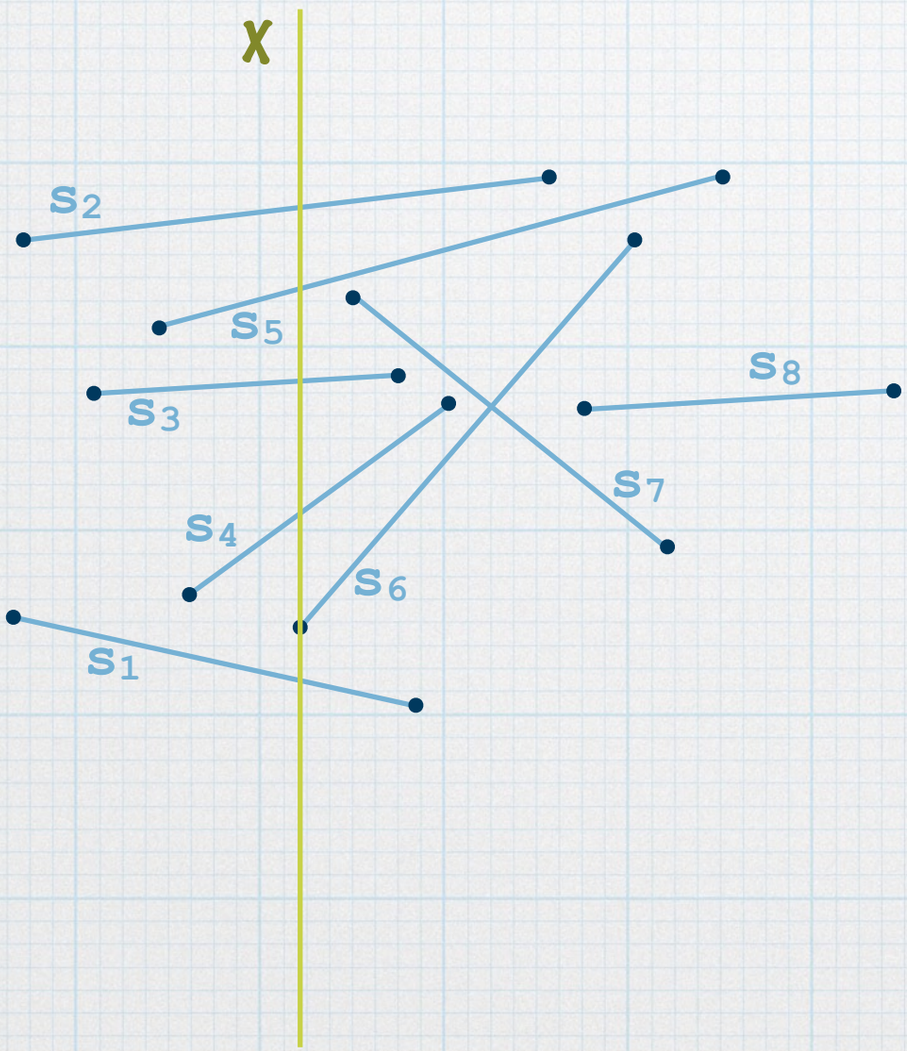


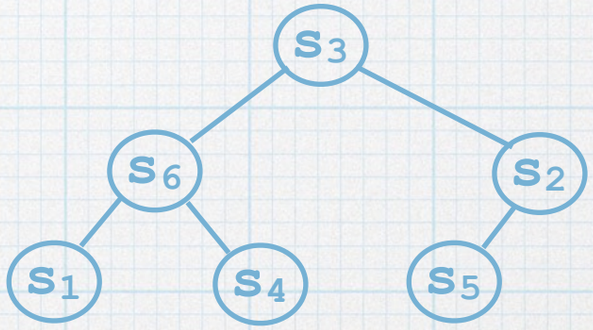
Példa



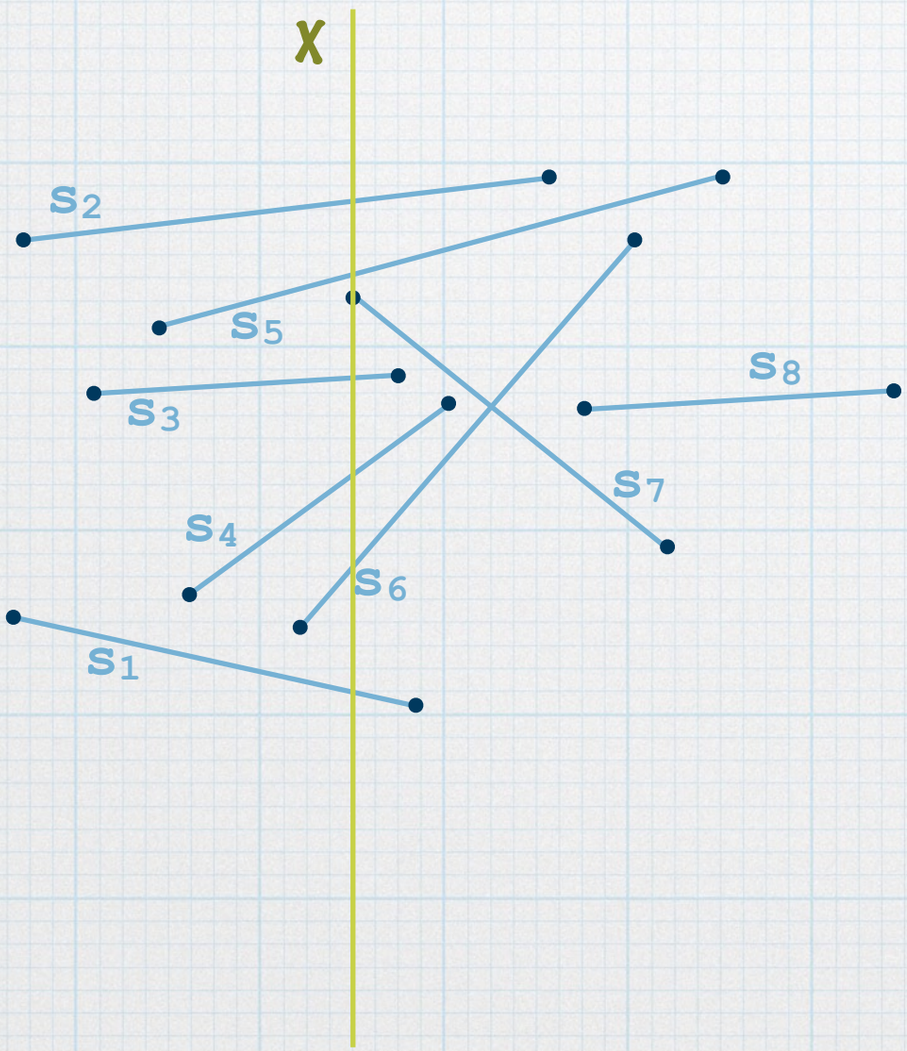


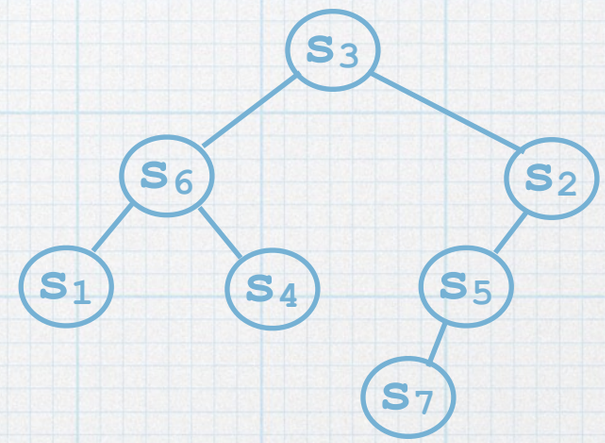
Példa



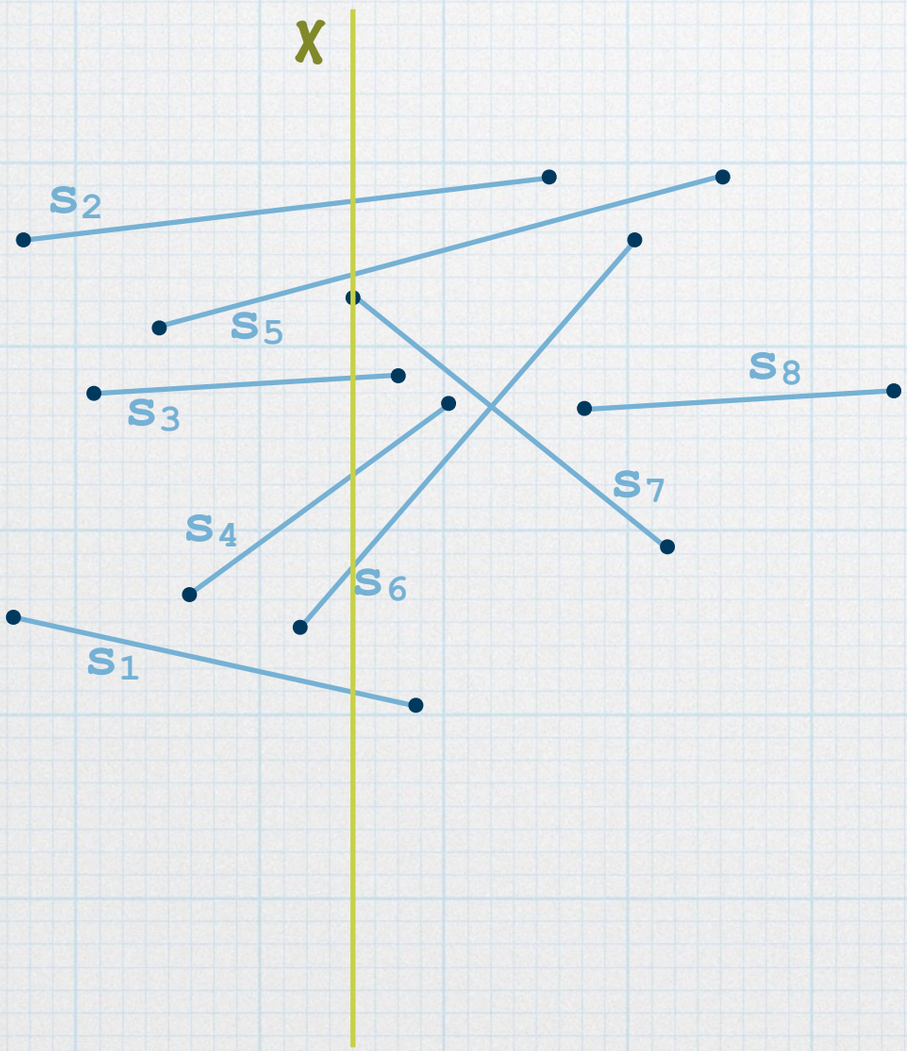


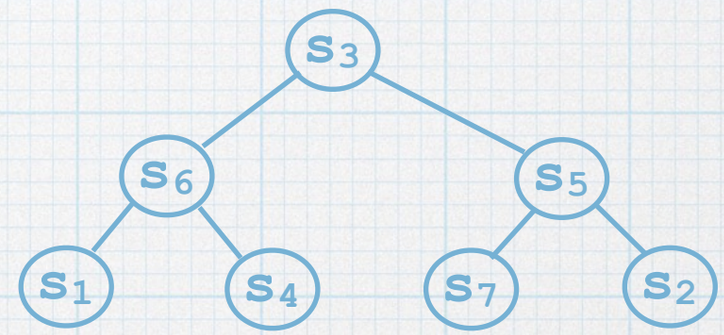
Példa



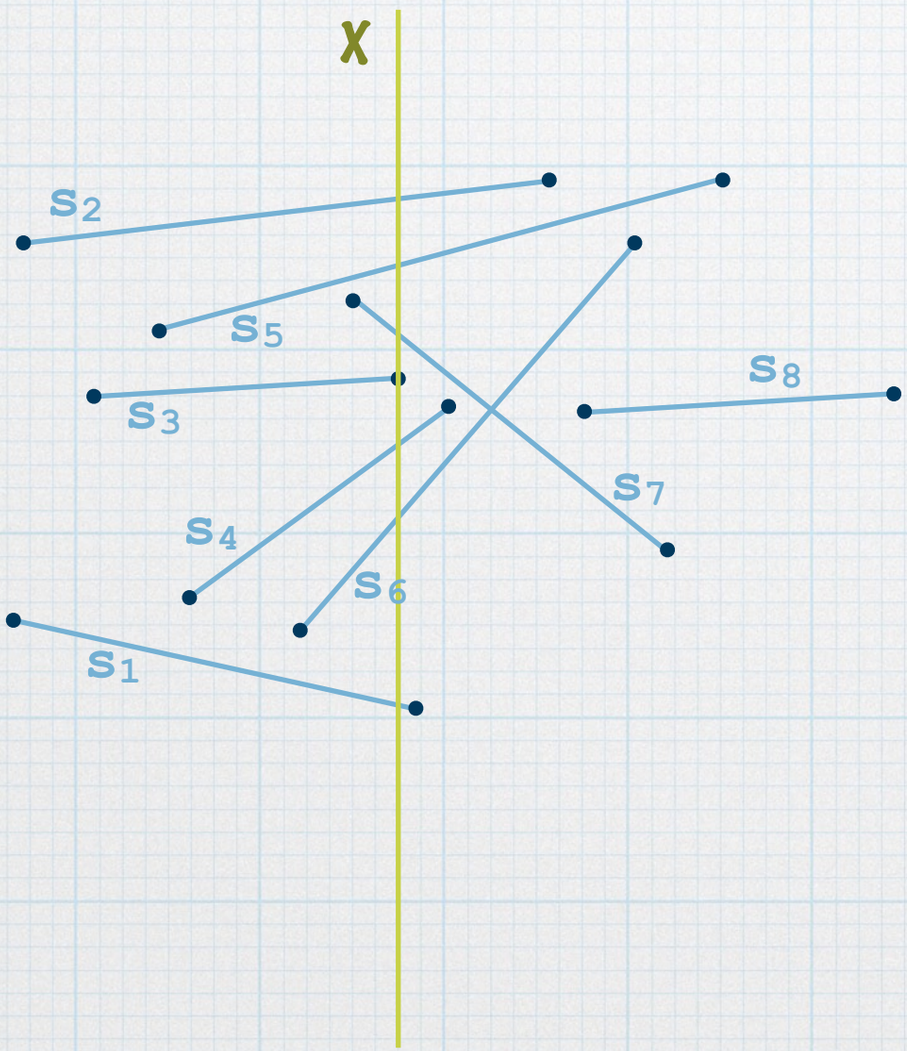


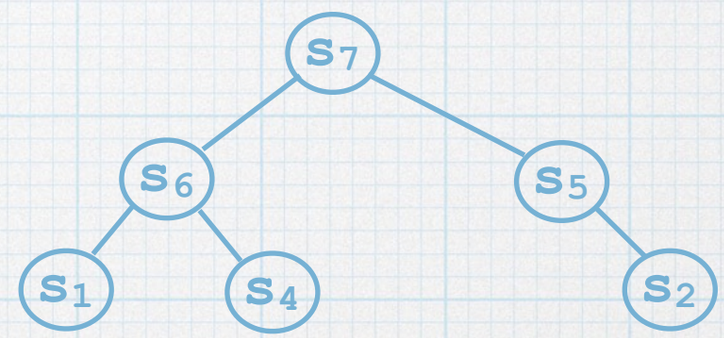
Példa



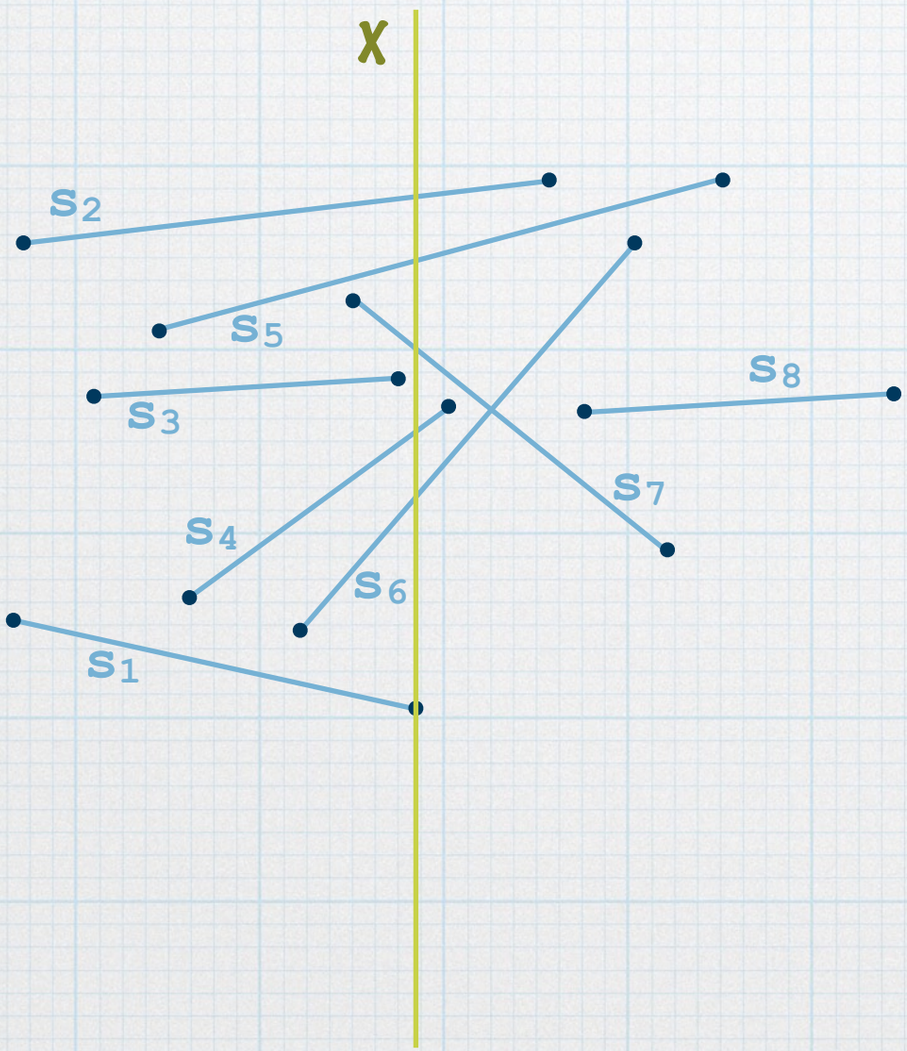


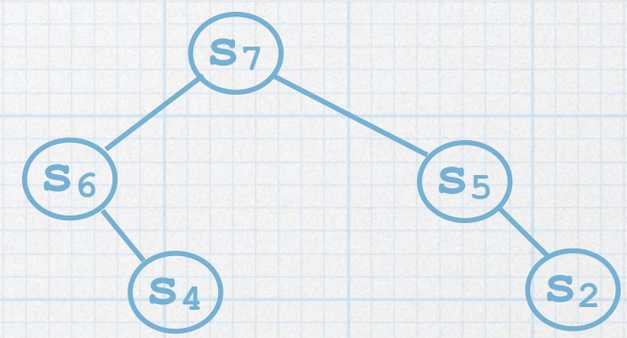
Példa



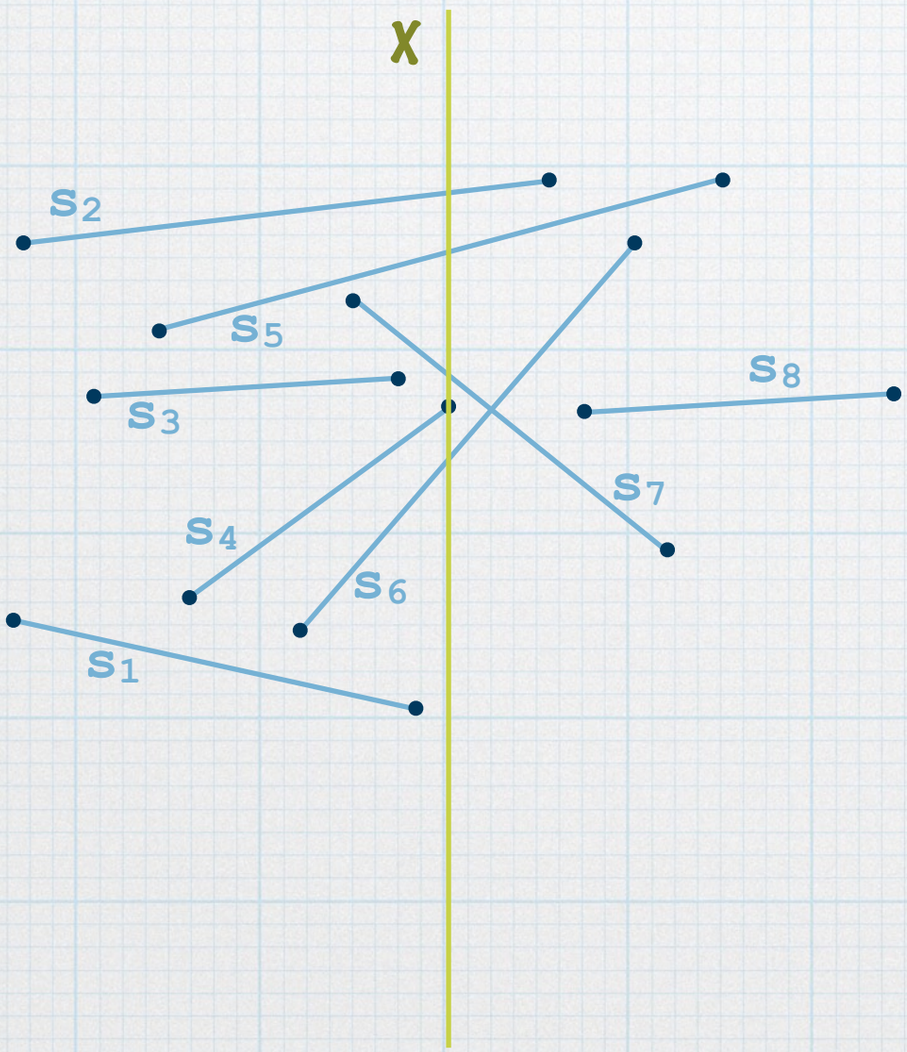


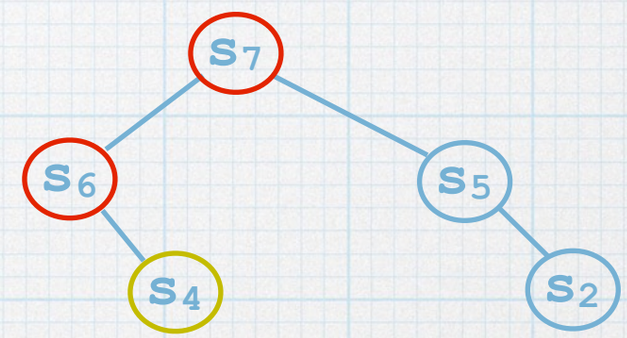
Példa





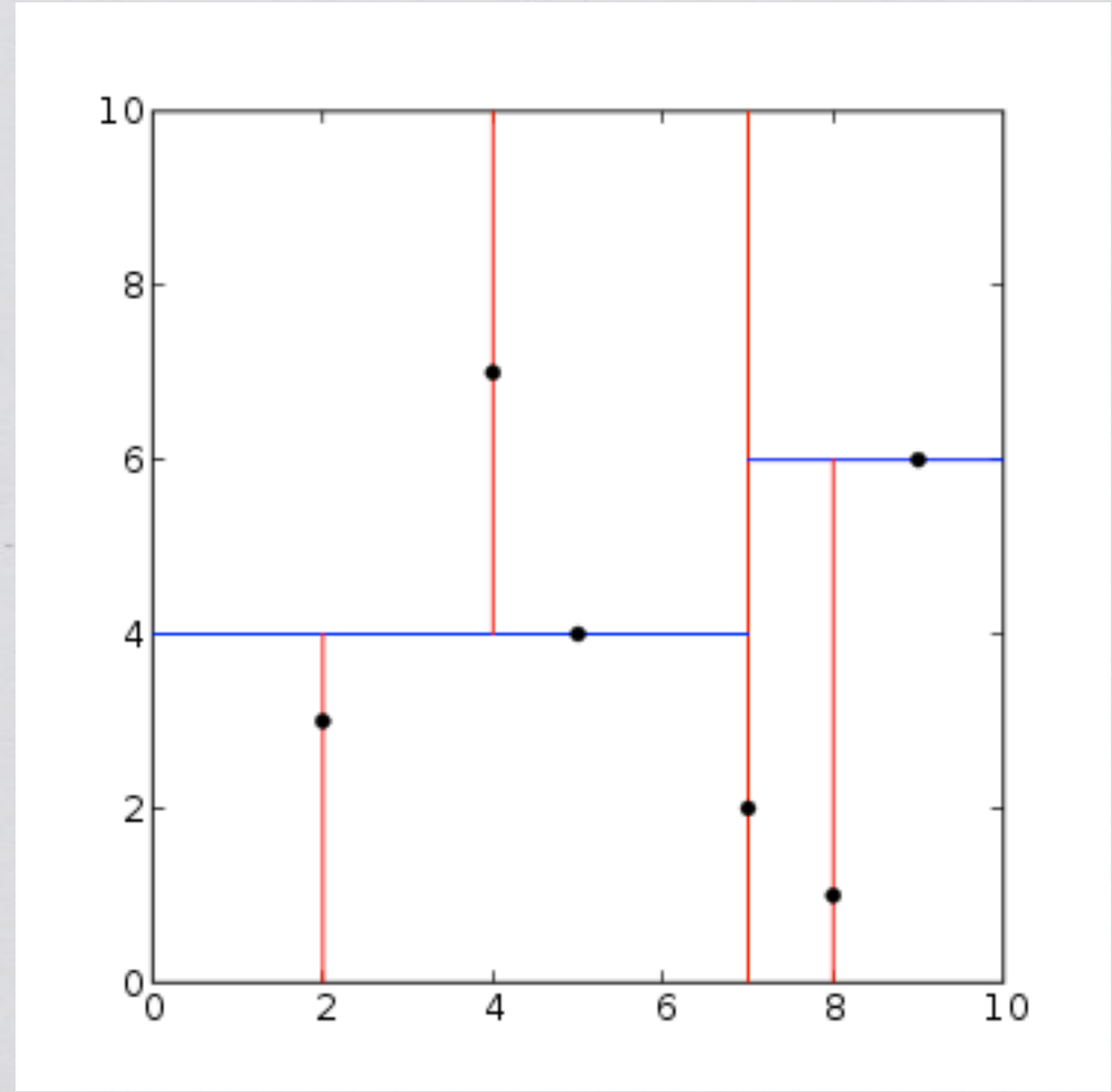
Példa

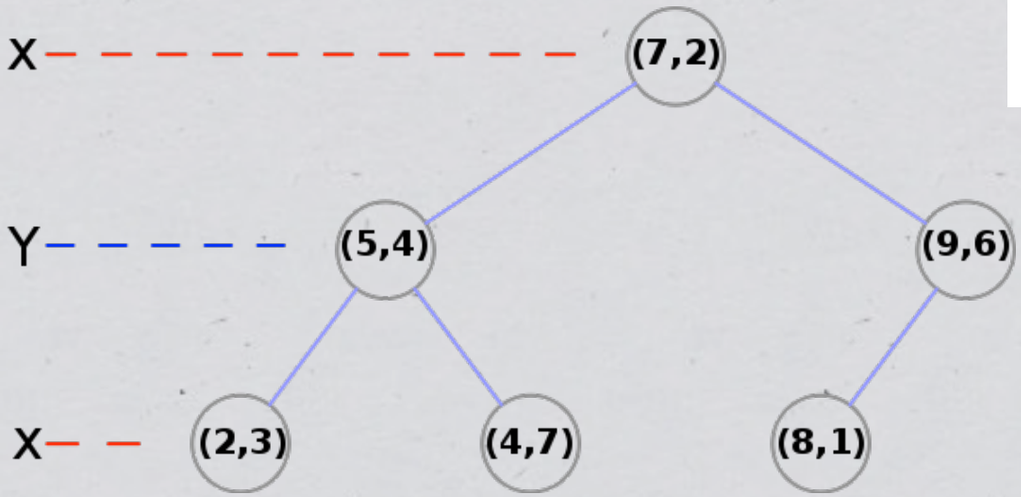


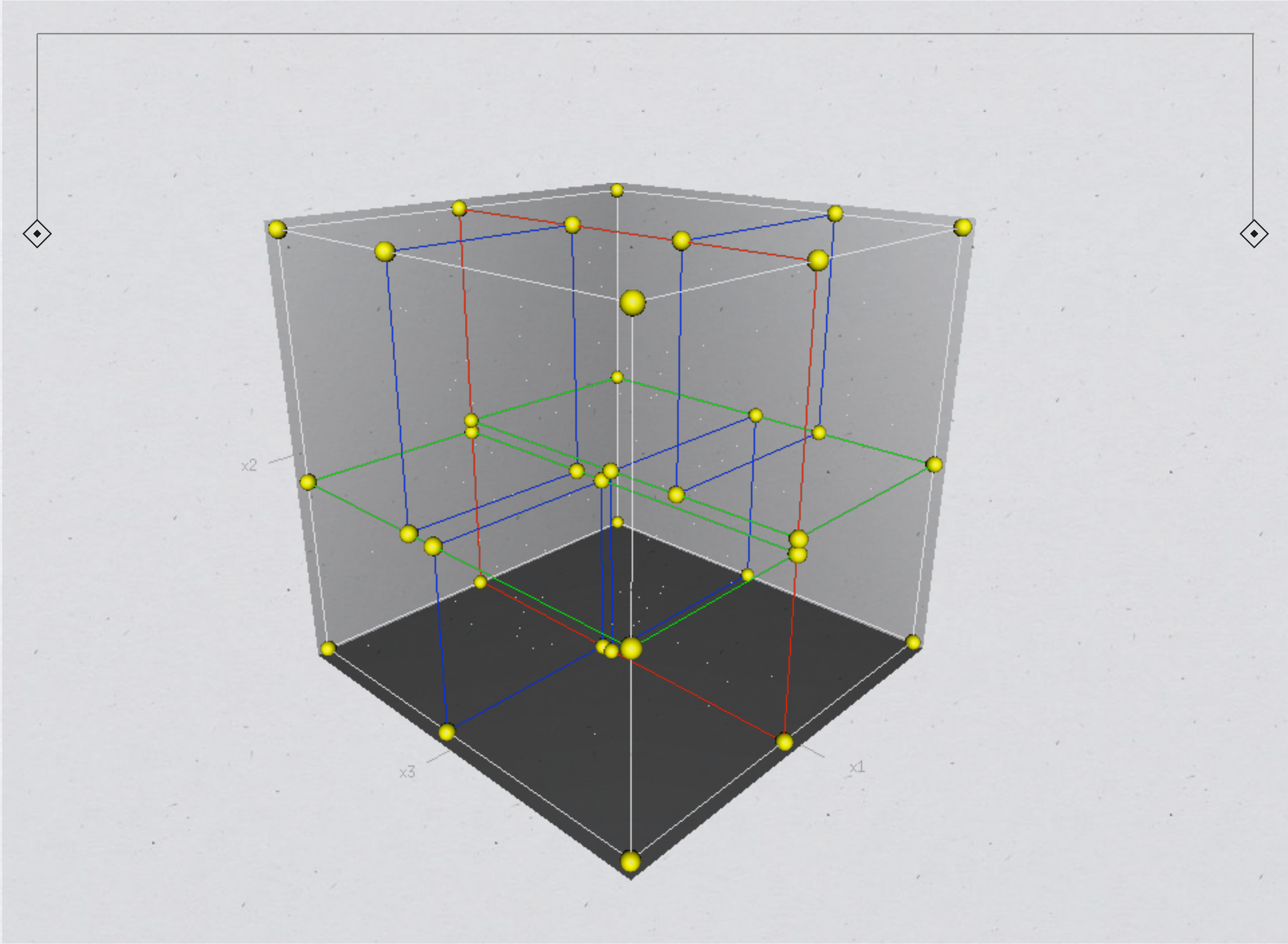


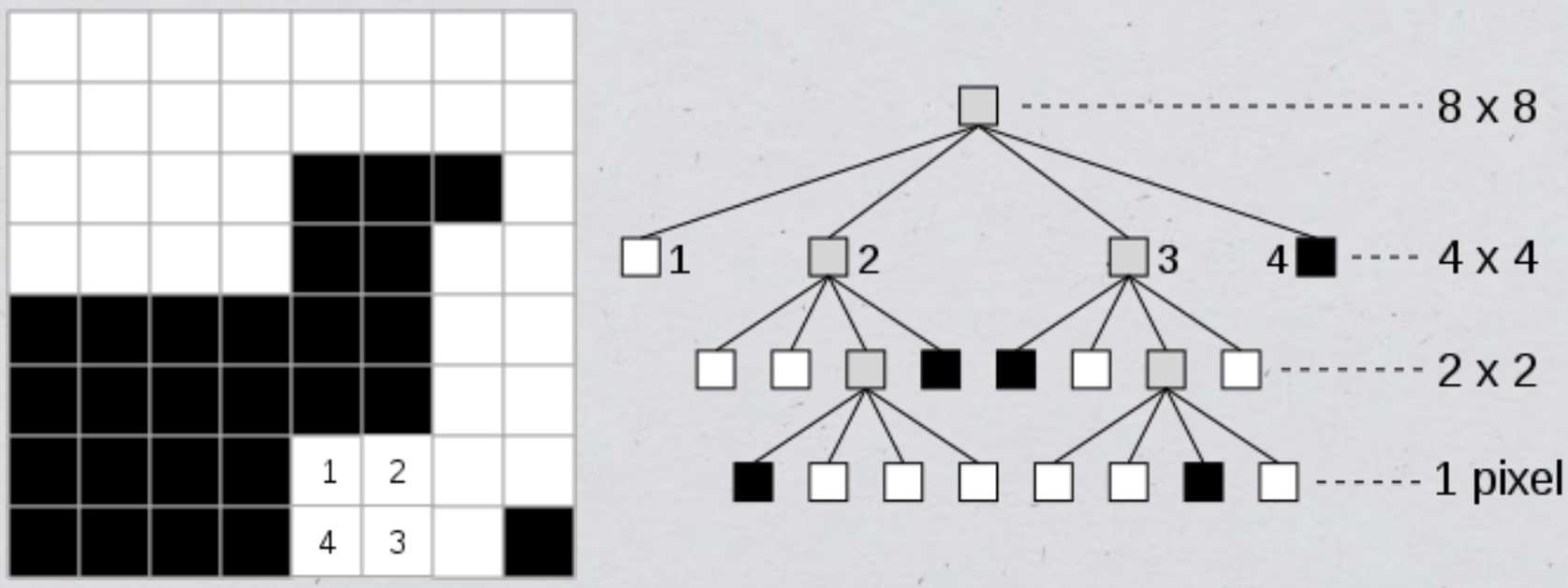
GEOMETRIAI ADATSZERKEZETEK

4-es fa, 8-as fa, k-d fa


10
8
6
4
2
0
0
2
4
6
8
10


X
Y
X
(7,2)
(5,4)
(9,6)
(2,3)
(4,7)
(8,1)


x2
x3
x1


1
2
4
3
8 x 8
1
2
3
4
4 x 4
2 x 2
1 pixel

Véletlenített algoritmusok

Gyorsrendezés véletlenül generált felosztó elemmel

RandomGyorsrendez

1. Ha csak egy elem van return.
2. Egyébként választunk egyenletes eloszlás alapján egy felosztó elemet.
3. A felosztó elem elé helyezzük a nála kisebb, mögé a nála nagyobb elemeket.
4. RandomGyorsrendez a felosztó elemnél kisebb elemekre
5. RandomGyorsrendez a felosztó elemnél nagyobb elemekre

A futási idő várható értéke legrosszabb esetben is O(nlogn)

Az olyan algoritmusokat, amelyek biztosan helyes eredményt adnak Las Vegas típusú algoritmusnak nevezzük (ez esetben az algoritmus futási ideje függ a véletlen döntésektől és az egy valószínűségi változó).

Az olyan algoritmusokat, amelyek nem biztos, hogy helyes eredményt adnak (viszont általában kicsi a futási idejük, és gyakran független a véletlen döntésektől) Monte Carlo típusú algoritmusoknak hívjuk.

- Egyoldali hibás Monte Carlo algoritmus: ekkor az algoritmus az egyik (vagy az igen vagy a nem) válasz esetén nem hibázhat, a másik válasz esetén a hiba valószínűsége alulról korlátozott.
- Kétoldali hibás Monte Carlo algoritmus: mindkét válasz esetén hibázhat az algoritmus, de adott alsó korlát a hibázás valószínűségére.

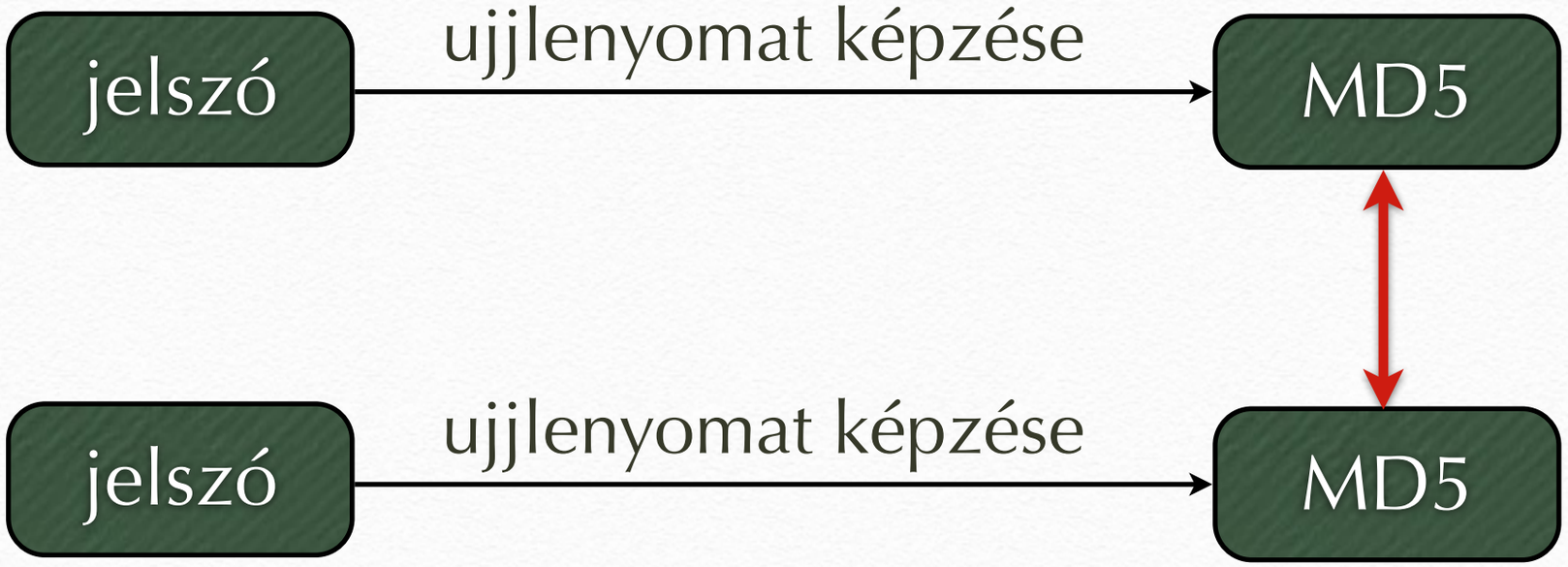
Polinomok egyenlőségének ellenőrzése

Legyen P(x) és Q(x) két n-edfokú polinom.
A feladat ellenőrízni, hogy azonosak-e.
Ha a polinomok az együtthatók által vannak megadva, akkor nyilvánvaló megoldás az együtthatók egyenlőségének ellen őrzése.
De el őfordulhatnak olyan alkalmazások, ahol a polinomok nem az együtthatókkal vannak megadva, hanem egy jóval bonyolultabb formában, esetleg csak csak egy eljárást használhatunk, amely adott helyen megadja a polinom értékét.

A feladat tulajdonképpen az, hogy a $P(x)-Q(x) = 0$ polinom egyenlőséget kell ellenőrízni. Mivel egy n-edfokú polinomnak legfeljebb n gyöke van, ezért egy megoldás, hogy veszünk n+1 darab értéket $x1,x2,...,xn+1$ és minden i-re teszteljük a $P(xi)-Q(xi) = 0$ egyenlőséget. Ha minden esetben egyenlőség áll fenn, akkor tudjuk, hogy a polinomok megegyeznek egyébként pedig különbözőek.

Vegyünk S elemet, amelyeken a polinomok értelmezve vannak, válasszunk véletlenszerűen egyenletes eloszlás alapján egy x elemet. Ha $P(x) \neq Q(x)$, akkor a polinomok különbözőek, egyébként valószínűleg megegyeznek.

Ujjlenyomat módszer alkalmazásai



Közelítő algoritmusok

Nehéz (pl. NP teljes) problémák optimális megoldásának meghatározására nem tudunk (garantáltan) polinom idejű algoritmust adni.

Lehetőségek:

- exponenciális futási idejű algoritmus
- polinomiális futási idejű algorimus amely nem feltétlenül az optimumot határozza meg, de garantáltan az optimálishoz közeli megoldást ad.

garantáltan az optimálishoz közeli megoldás

minden I inputra OPT(I) jelölje az optimális megoldás célfüggvényértékét

egy A algoritmusra A(I) jelölje az algoritmus által az A inputon felvett célfüggvényértéket

Minimalizálási feladatok esetén egy polinomiális idejű A algoritmusra akkor mondjuk, hogy C-approximációs, ha minden I inputra:

$$A(I) \leq C \cdot OPT(I)$$

Gyakran a definícióban megengednek egy inputtól független D additív konstanst, azaz az $A(I) \leq C \cdot OPT(I) + D$ egyenlotlenségnek kell teljesülnie minden inputra.

Egy algoritmus approximációs hányadosának a legkisebb C számot nevezzük, amire az C-approximációs.

Maximalizálási feladatnál akkor beszélünk C-approximációs algoritmusról, ha minden I-re $OPT(I) \leq C \cdot A(I)$.

A polinomiális approximációs séma (PTAS) olyan algoritmuscsalád, amely minden $\varepsilon > 0$ számhoz tartalmaz egy $1+\varepsilon$-approximációs algoritmust.

Bár az algoritmus polinimiális idejű ε-t konstansnak tekintve, a futási ideje nagyon nagy lehet (pl. $\Theta(n^{1/\varepsilon})$).

Ládapakolási feladat

Adottak tárgyak a méretük által $a_1,\ldots,a_n$.
Cél a tárgyakat elhelyezni a lehető legkevesebb egység méretű ládába.
Egy ládába egy L halmaz pakolása akkor lehetséges, ha

$$\sum_{i \in L} a_i \leq 1$$

Ládapakolási feladat

FF algoritmus: Az a_i elemet az első olyan ládába teszi, amelybe elfér, ha egyikben sem fér el új ládát nyit.

Tétel: A FF algoritmus 2-approximációs.

Bizonyítás:
Vegyünk két egymás utáni ládát a pakolásban. A ládákba helyezett tárgyak összsúlya nagyobb, mint 1. (Ez bármely két ládára igaz.)
Tehát, ha FF k darab ládát használt, akkor a tárgyak összsúlya nagyobb, mint k\2.
Másrészt az optimális megoldás legalább annyi ládát használ mint a ládák összsúlya, így OPT > k\2, amiből 2OPT ≥ k adódik.

□

Ládapakolási feladat

Az FFD algoritmus elsőként rendezi a tárgyakat méret szerint monoton csökkenő sorrendbe majd ezt követően hajtja végre az FF algoritmust.

Tétel Az FFD algoritmus aszimptotikus hányadosa 11/9.

Online algoritmusok

Online problémáról beszélünk azokban az esetekben, ahol nem ismert az egész input, hanem az algoritmus az inputot részenként kapja meg, és a döntéseit a megkapott részletek alapján a további részekre vonatkozó információk nélkül kell meghoznia.

Online (vagy semi-online) algoritmusokat sok probléma esetén alkalmaznak az operációkutatás, az elméleti számítástudomány és a közgazdaságtan különböző területein.

Az algoritmusok hatékonyságát általában a versenyképességi analízis alapján mérik (egy másik használt elemzési módszer az átlagos eset analízis).

Versenyképességi analízis

Optimalizálási feladatok esetén minden I inputra jelölje OPT(I) az optimális megoldás célfüggvényértékét és egy A algoritmusra A(I) az algoritmus által az I inputon felvett célfüggvényértéket.

Minimalizálási feladatok esetén egy online A algoritmusra azt mondjuk C-versenyképes, ha minden I inputra teljesül, hogy $A(I) \leq C \cdot OPT(I)$.

Egy A algoritmus aszimptotikus versenyképességi hányadosa: R_A^∞

$$R_A^n = \max\{A(L)/OPT(L) \mid OPT(L) = n\}$$

$$R_A^\infty = \limsup_{n\to\infty}(R_A^n)$$

Az aszimptotikus hányados fő tulajdonsága az, hogy azt vizsgálja, miként viselkedik az algoritmus akkor, ha a bemenet mérete nő, pontosabban ha az optimális költség a végtelenhez tart.

Online ládapakolási feladat

Ha a nyitott ládák száma csak egy lehet, akkor az egyetlen algoritmus, amely használható, az NF algoritmus:

Amennyiben a tárgy elfér a nyitott ládában tegyük oda! Ellenkező esetben zárjuk be a nyitott ládát, nyissunk egy új ládát és tegyük abba a tárgyat!

Tétel: Az NF algoritmus aszimptotikus versenyképességi hányadosa 2.

A nyugtázási probléma

költségfüggvény $$k + \sum_{j=1}^{k} v_j$$

ahol k a nyugták száma

és $$v_j = \sum_{t_{j-1} < a_i \leq t_j} t_j - a_i$$

a j-edik nyugta által összegyűjtött teljes késedelem.

A probléma online, azaz egy adott t időpontban csak a t-ig megérkezett csomagok érkezési idejeit ismerjük és nincs semmi információnk a további csomagokról.

Szimmetrikus kulcsú titkosítás

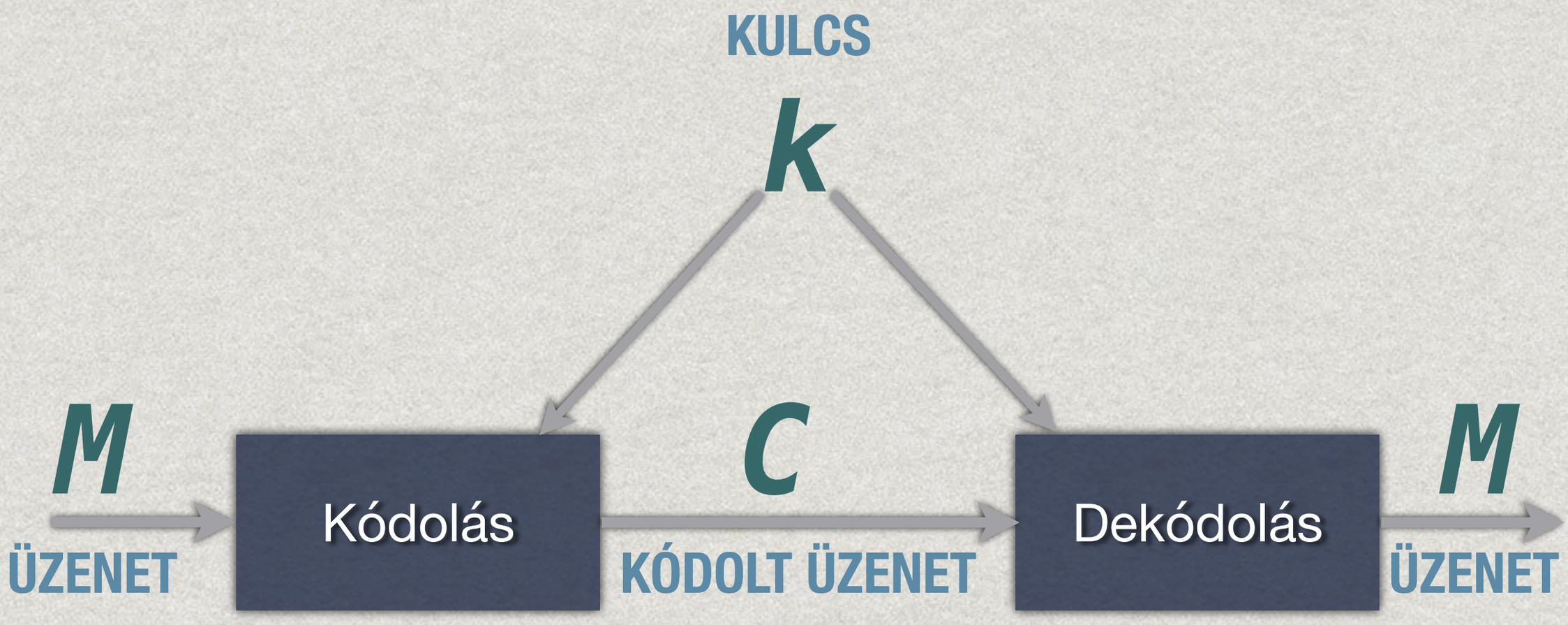
a kulcsmegosztás problémája

A titkosítandó szöveget a közös titkosítási kulcsot felhasználva átalakítjuk, az így kapott információt továbbítjuk, majd a fogadó fél ugyanazt a közös titkosító kulcsot használva fejti azt meg.

Problémát okoz, hogy a kulcs egyértelműen feloldja a védett információt.
Speciális feladat a titkos kulcs küldő és fogadó fél közötti cseréjének a megoldása.
További probléma a titkosító kulcsok biztonságának védelme a helyi számítógépeken.

AES IDEA

Szimmetrikus kulcsú titkosítás



Nyilvános kulcsú titkosítás

RSA algoritmus

OpenPGP

This mail will be signed and encrypted

OpenPGP

To: GPGTools Project <gpgtools-org@lists.gpgtools.org>

Cc:

Bcc:

Reply To:

Subject: This mail will be signed and encrypted

From: GPGTools Project... Signature: Signature #3

This mail will be signed and encrypted...

This mail will be signed and encrypt

Delete Junk Reply Reply All Forward

From: GPGTools Project <gpgtools-org@lists.gpgtools.org>
Subject: This mail will be signed and encrypted
Date: 9. November 2011 19:35:21 MEZ
To: GPGTools Organization
Security: Encrypted, Signed (gpgtools-org@lists.gpgtools.o

This mail will be signed and encrypted...

GPGMail

General Accounts RSS Junk Mail Fonts & Colors Viewing Composing Signatures Rules GPGMail

GPGMail
Version: 2.0a4

FAQ
Donate
Contact

Composing
- [x] Enable OpenPGP/MIME
- [ ] Optionally encrypt drafts

Reading
- [x] Enable OpenPGP/MIME

Updates
- [x] Download beta versions
- [x] Automatically check for updates
- [ ] Download updates automatically

Check Now

NYILVÁNOS KULCSÚ TITKOSÍTÁS

Legyen D a titkosítandó üzenetek halmaza.

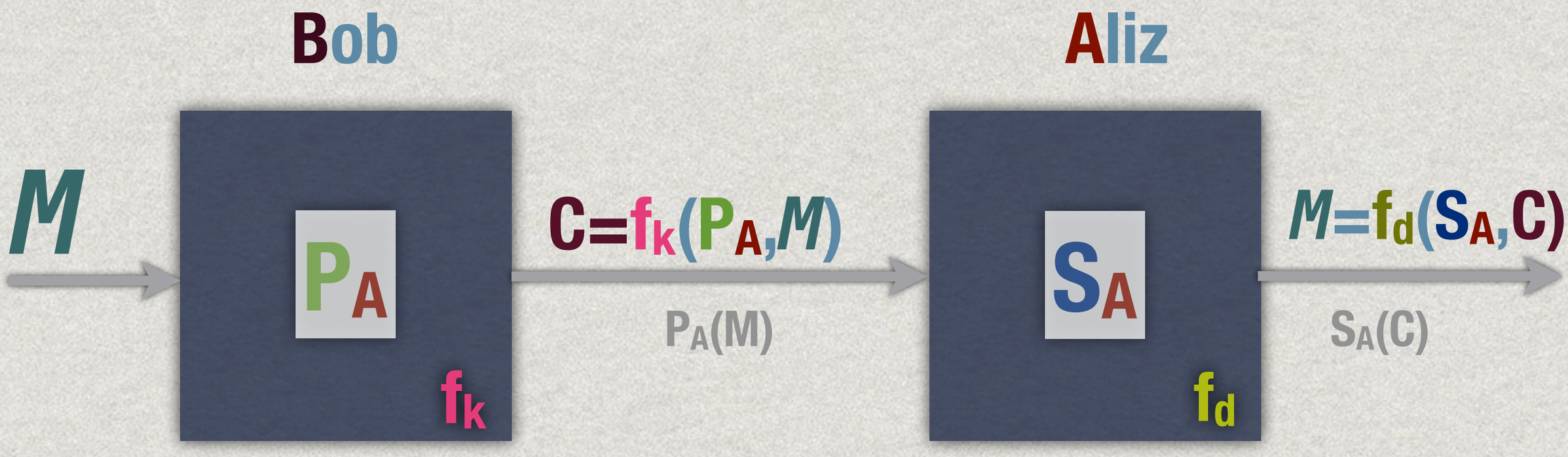
Tegyük fel, hogy **B**ob titkosítottan szeretné elküldeni **A**liznak az $M \in D$ üzenetet.

A nyilvános kulcsú titkosítás esetén **A**liznak van egy S_A titkos (**S**aját) és egy P_A nyilvános (**P**ublikus) kulcsa, továbbá van egy $f_k : D \rightarrow D'$, $f_d : D' \rightarrow D$ függvénypár (a kódoló és dekódoló függvények), amelyekre teljesül, hogy

$$f_d(S_A, f_k(P_A, M)) = M$$

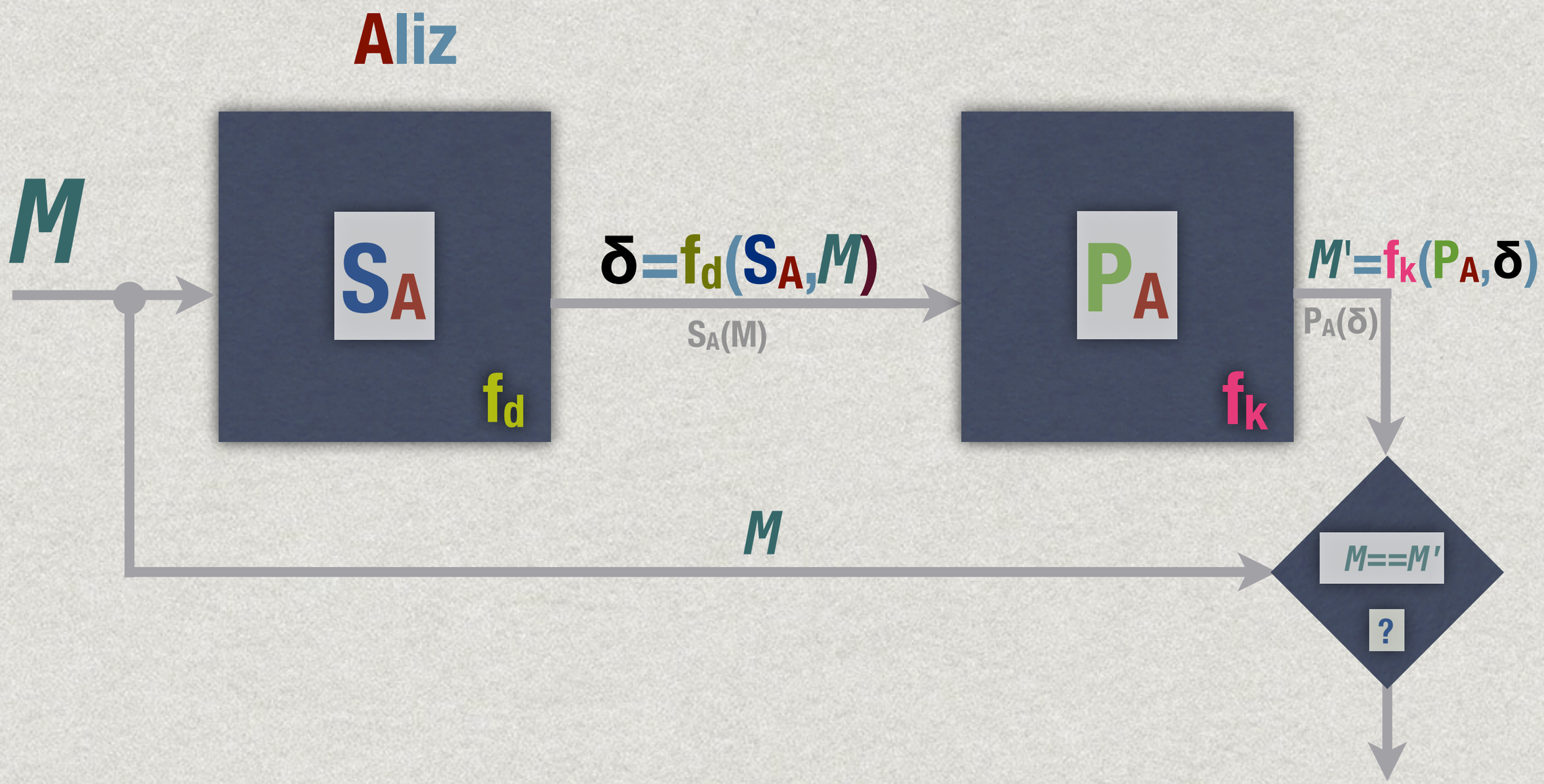
NYILVÁNOS KULCSÚ TITKOSÍTÁS

$$f_d(S_A, f_k(P_A, M)) = M$$



DIGITÁLIS ALÁÍRÁS

$$f_k(P_A, f_d(S_A, M)) = M$$



RSA ALGORITMUS

1976, Ronald L. **R**ivest, Adi **S**hamir és Len **A**dleman

1. Vegyünk véletlenszerűen két különböző nagy prímszámot, p-t és q-t.
2. Legyen $n = pq$.
3. Vegyünk egy olyan kis páratlan e számot, amely relatív prím $\phi(n) = (p-1)(q-1)$-hez.
4. Keressünk egy olyan d számot, amelyre $ed = 1 \bmod \phi(n)$.
5. Az RSA nyilvános kulcs a $P = (e,n)$ pár lesz.
6. Az RSA titkos kulcs az $S = (d,n)$ pár lesz.

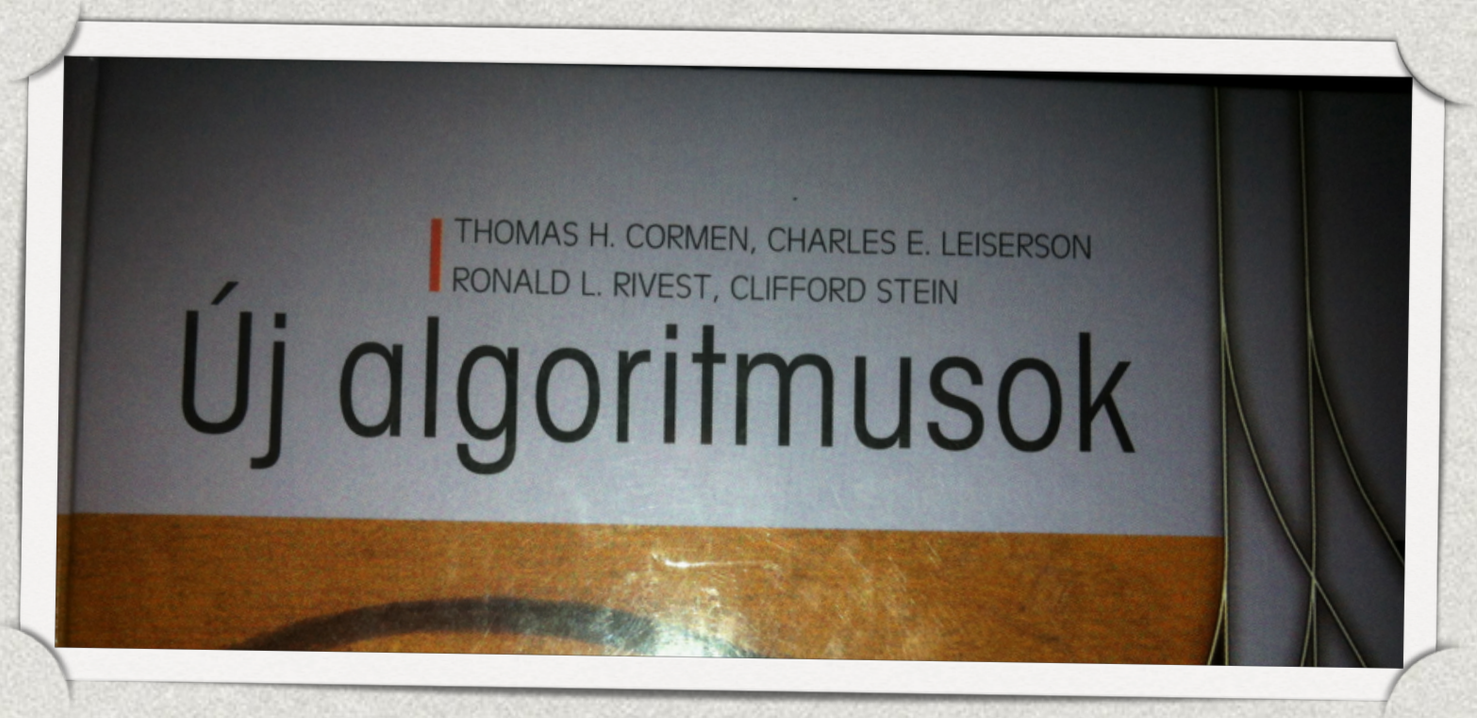
$2^m < n < 2^{m+1}$

m=128, 256, 512, 1024

65537

RSA ALGORITMUS

1976, Ronald L. **R**ivest, Adi **S**hamir és Len **A**dleman



Ebben a sémában az elküldhető üzenetek halmaza $Z_n = \{0,1,\ldots,n-1\}$.

A kódolás a $P = (e,n)$ nyilvános kulccsal:

$$P(M) = M^e \bmod n.$$

A dekódolás a titkos kulccsal:

$$S(C) = C^d \bmod n.$$

PÉLDA:

1. Vegyünk véletlenszerűen két különböző nagy prímszámot, p-t és q-t. $p = 5, q = 7$
2. Legyen $n = pq$.
3. Vegyünk egy olyan kis páratlan e számot, amely relatív prím $\phi(n) = (p-1)(q-1)$-hez.
4. Keressünk egy olyan d számot, amelyre $ed = 1 \bmod \phi(n)$.
5. Az RSA nyilvános kulcs a $P = (e,n)$ pár lesz.
6. Az RSA titkos kulcs az $S = (d,n)$ pár lesz.

$n = p \cdot q = 5 \cdot 7 = 35$ 3?

$\phi(n) = (5-1)(7-1) = 4 \cdot 6 = 24$

$e = 7$ // relatív prím 24-hez

$9 \cdot 24 = 216$

$e \cdot d = 7 \cdot 31 = 217 = 1 \bmod 24$ 2?

$P = (e,n) = (7,35)$

$S = (d,n) = (31,35)$

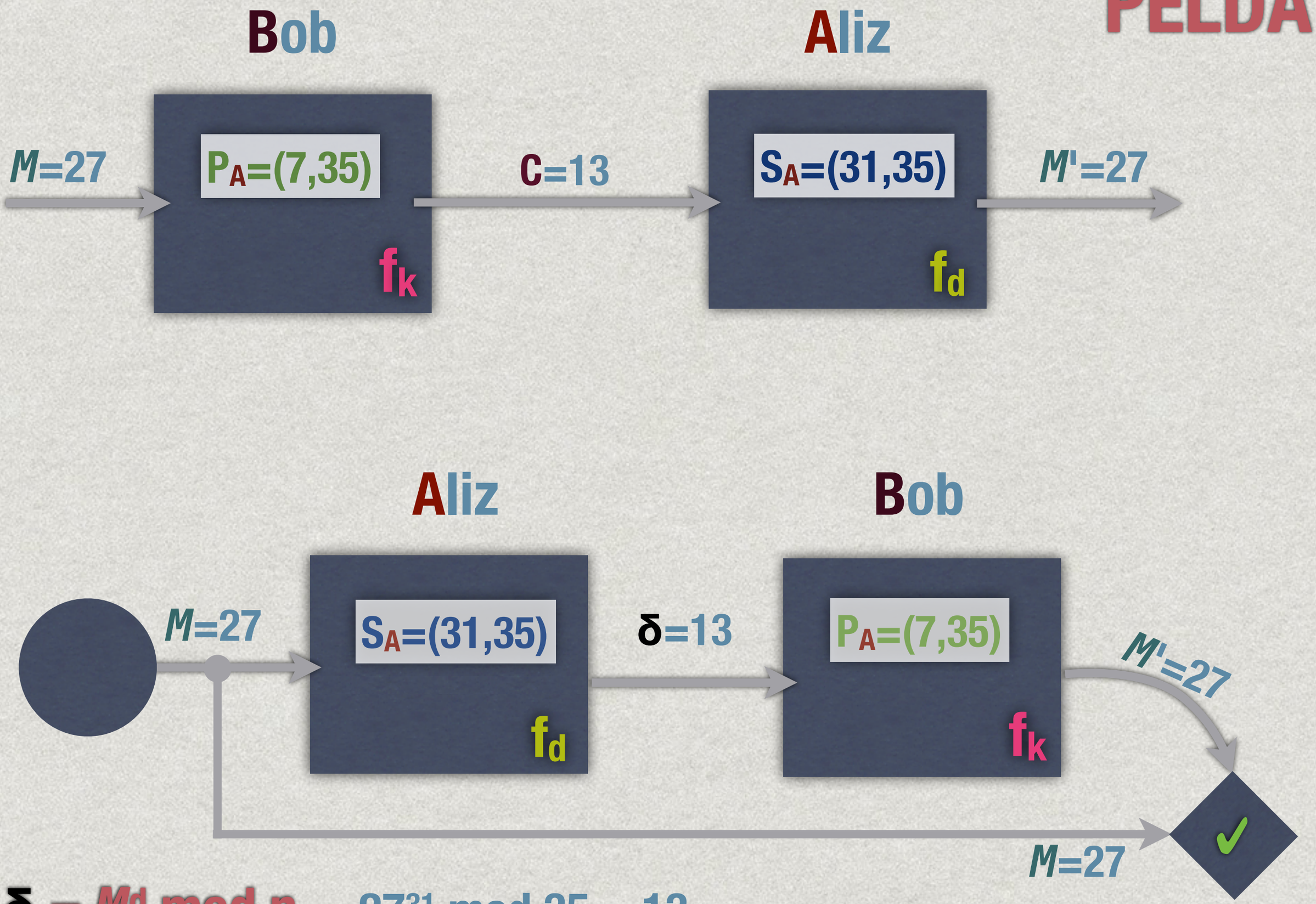
$M = 27$

1?

$C = M^e \bmod n = 27^7 \bmod 35 = 10460353203 \bmod 35 = 13$

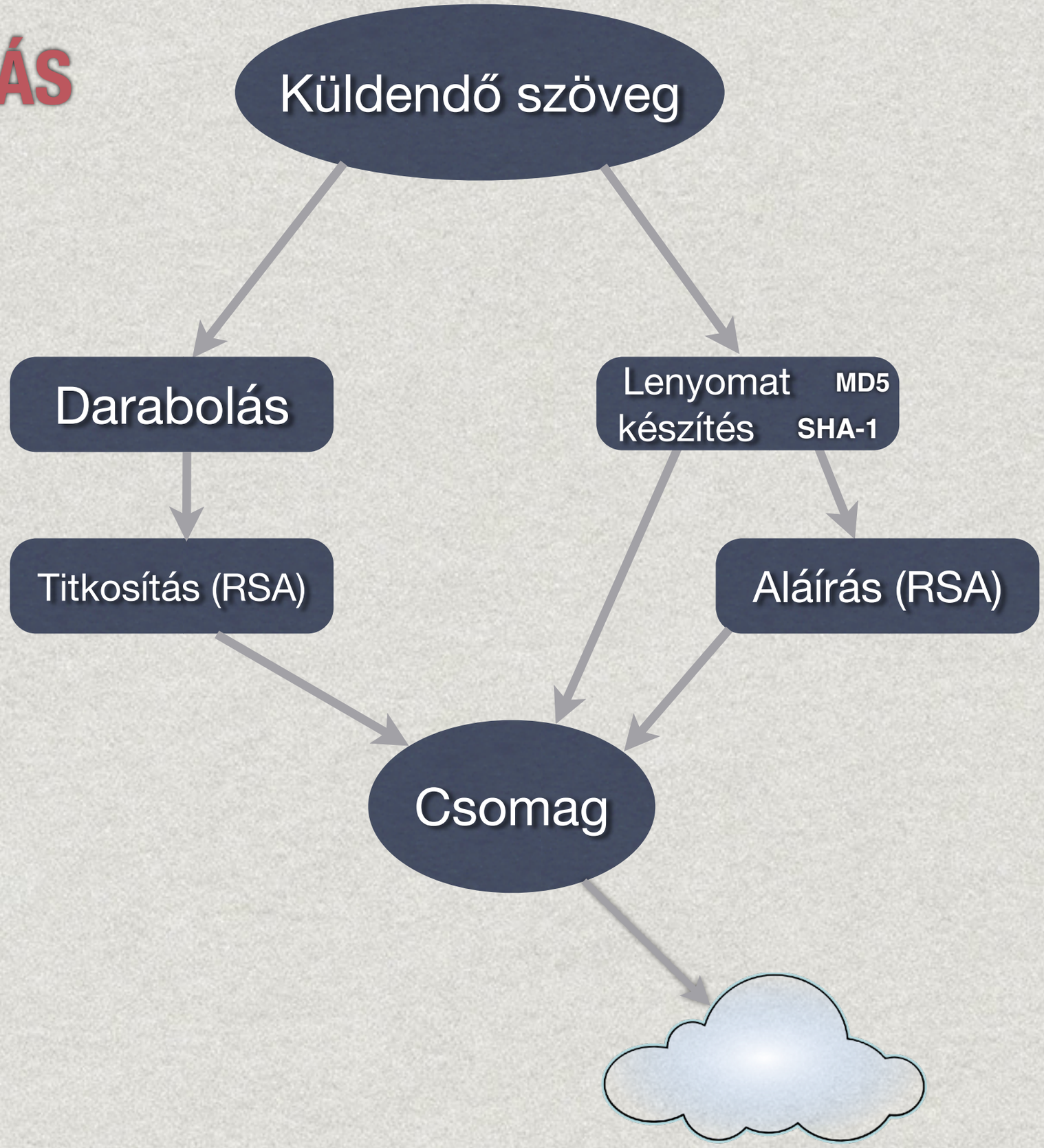
$M' = C^d \bmod n = 13^{31} \bmod 35 = 27$

PÉLDA



$$\delta = M^d \bmod n = 27^{31} \bmod 35 = 13$$

PÉLDA ALKALMAZÁS



HATVÁNYOZÁS

$P(M) = M^e \bmod n.$

$3^{11}=3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$

$11=8+2+1=2^3+2^1+2^0$

$b = b_k 2^k+\ldots+b^1 2^1+b_0$

$3^{11}=3^{2^3+2^1+2^0} = 3^{2^0} \cdot 3^{2^1} \cdot 3^{2^3}$

```
hatv(a,b) {
   d = a;
   legyen <bk;bk-1;...;b1;b0> b bináris alakja;
   if (b0==0) e=1 ; else e = a ;
   for (i=1;i<=k;i++) {
      d = d·d;
      if (bi == 1) e = d·e;
   }
   return e;
}
```

HATVÁNYOZÁS

$P(M) = M^e \bmod n.$

```
static long Hatv(int a, int b){
   long e=(b%2==1)? a:1;
   long d=a;    //2-hatványok
   b>>=1;
   while (b>0){
      d*=d;
      if (b%2==1)
      e*=d;
      b>>=1;
   }
   return e;
}
```

MODULÁRIS HATVÁNYOZÁS

$P(M) = M^e \bmod n.$

$(x \cdot y)/n = (x/n) \cdot (y/n)$

$(x \cdot y) \backslash n = (x \backslash z) \cdot (y \backslash n)$

$(x \cdot y) \% n = ((x \% n) \cdot (y \% n)) \% n$

```
mhatv(a,b,n) {
   d = a;
   legyen <b_k;b_{k-1};...;b_1;b_0> b bináris alakja;
   if (b_0==0) e=1 ; else e = a ;
   for (i=1;i<=k;i++) {
      d = (d·d) mod n;
      if (b_i == 1) e = (d·e) mod n;
   }
   return e;
}
```

Z_n

$3^6 \bmod 5$

d=3, e=3, b=<110>, e=1

i=1: d=4, e=4

i=2: d=1, e=4

$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$

$4 \cdot 4 \cdot 4$

$1 \cdot 4$

$3^6 \bmod 5 = 4$

$3^6 = 729$

1?

MODULÁRIS HATVÁNYOZÁS

$P(M) = M^e \bmod n.$

```
static long ModHatv(int a, int b, int n){
   long e=(b%2==1) ? a%n:1;
   long d=a; //2-hatványok mod n: a^(2^i) mod n
   b>>=1;
   while (b>0){
      d=(d*d)%n;
      if (b%2==1) e=(e*d)%n;
      b>>=1;
   }
   return e;
}
```

$C=27^7 \bmod 35$

a=27, b=7, n=35

e=27, d=27, b=3

d=29, e=13, b=1

d=1, e=13, b=0

e=27*27=729 35*20=700

e=29*27=783 35*22=770

e=29*29=841 35*24=840

$e \cdot d \equiv 1 \pmod{\phi(n)}$

$a \cdot x \equiv b \pmod{n}$

Jelölje G(a) az a elem által generált additív részcsoportját Z_n-nek!

$G(a) = \{(a \cdot x) \bmod n : x > 0\}$.

A kongruenciának akkor és csak akkor van megoldása, ha $b \in G(a)$.

Legyen $d = \text{lnko}(a, n)$.

Ekkor $G(a) = G(d) = \{0, d, 2d, \ldots, ((n/d)-1)d\}$, így $|G(a)| = n / d$.

pl.: $18 \cdot x \equiv 2 \pmod{10}$

x	0	1	2	3	4	5	6	7	8	9
b	0	8	6	4	2	0	8	6	4	2

$G(18) = G(2) = \{0, 2, 4, 6, 8\}$, így $|G(a)| = 5 = 10 / 2$.

KULCSPÁROK GENERÁLÁSA

$a \cdot x \equiv b \pmod{n}$

$a \cdot x + n \cdot y = b$

```
lnko(a,b) {
   if (b == 0) return(a) ;
   else return(lnko(b,a%b)) ;
}
```

```
lnko(40,25) = lnko(25,15) = lnko(15,10)
                                  ||
                            lnko(10,5)    10*0+1*5=5
                                  ||
                            lnko(5,0)  =  5
                                          1*5+0*0=5
```

Az $a \cdot x = b \bmod n$ kongruencia akkor és csak akkor oldható meg, ha d=lnko(a,n) osztható b-vel.

$e \cdot d \equiv 1 \pmod{\phi(n)}$

Ha van megoldás, akkor pontosan d darab van.

KULCSPÁROK GENERÁLÁSA

$a \cdot x \equiv b \pmod{n}$

$a \cdot x + n \cdot y = b$

$40 \cdot x \equiv 10 \pmod{25}$

$40 \cdot x + 25 \cdot y = 10$

$40 \cdot 4 + 25 \cdot (-6) = 10$

```
blnko(a,b) {
   if (b == 0) return(a,1,0) ;
   else {
      (d, x, y)=blnko(b,a%b) ;
      return(d ,y, x-(a\b)·y) ;
   }
}
```

```
(d,x,y)=blnko(a,n)
```

$40 \cdot 2 + 25 \cdot (-3) = 5$	`blnko(40,25) = (5,2,-3)`
$\parallel$	
$25 \cdot (-1) + 15 \cdot 2 = 5$	`blnko(25,15) = (5,-1,2)`
$\parallel$	
$15 \cdot 1 + 5 \cdot (-1) = 5$	`blnko(15,10) = (5,1,-1)`
$\parallel$	
$10 \cdot 0 + 5 \cdot 1 = 5$	`blnko(10,5) = (5,0,1)`
$\parallel$	
$5 \cdot 1 + 0 \cdot 0 = 5$	`blnko(5,0) = (5,1,0)`

KULCSPÁROK GENERÁLÁSA

$40 \cdot x \equiv 10 \pmod{25}$ $\quad a \cdot x \equiv b \pmod{n}$

a b n

x d

$40 \cdot 2 + 25 \cdot (-3) = 5 \quad \Longrightarrow \quad 40 \cdot 4 + 25 \cdot (-6) = 10$

```
linearis-konguencia-megoldo(a,b,n) {
   (d,x,y)=blnko(a,n) ;
   if (!d%b) {
      x *= (b/d)%n ; 4
      for (i=0;i<d;i++)
         print x+i*(n/d)%n ;
                          5
   }
   else print "nincs megoldás" ;
}
```

4 9 14 19 24

KULCSPÁROK GENERÁLÁSA

Vegyünk véletlenszerűen két különböző nagy prímszámot, p-t és q-t.

Ismert, hogy végtelen sok prím van, így tetszőlegesen nagy prím található.
A prímek nem helyezkednek el nagyon ritkán.
Ha π(n) jelöli az n-nél nem nagyobb prímek számát, akkor

$$\lim_{n\to\infty} \frac{\pi(n)}{n / \ln n} = 1$$

1. VEGYÜNK EGY VÉLETLENÜL VÁLASZTOTT SZÁMOT
2. NÉZZÜK MEG PRÍM-E
3. HA NEM PRÍM VISSZA AZ 1. PONTBA

A Fermat tétel alapján, minden p prímre és minden
a = 1, ..., p–1 -re $a^{p-1} = 1 \bmod p$.

Az első prímtesztelő algoritmus kiszámolja a mhatv(2,n-1,n) értéket,
és ha nem 1-et kapunk, akkor tudjuk, hogy n összetett szám.

Az algoritmus, akkor hibázik, ha n olyan összetett szám,
amelyre $2^{n-1} = 1 \bmod n$.

Általában azokat az összetett számokat,
amelyekre $a^{n-1} = 1 \bmod n$, *a* alapú álprímnek nevezzük.

Természetesen adódik a kérdés, hogy milyen gyakran téved az
algoritmus, azaz milyen gyakran fordulnak elő 2 alapú álprímek.
1000-nél kisebb 2 alapú álprím 22 van, az álprímek aránya tart a 0-hoz.

Kézenfekvő ötlet, hogy használjunk a teszteléshez a 2 számon kívül más alapokat is.

Ez nem oldja meg a problémát, mert vannak olyan számok, amelyek minden a-ra a alapú álprímek.

Ezeket a számokat nevezzük Carmichael számoknak.

Ezekből még kevesebb van, 10^9-nél kisebb Carmichael szám 255 van, a legkisebb az 561.

MILLER RABIN VALÓSZÍNŰSÉGI PRÍMTESZT

A prímteszt két lépésből áll.
Egyrészt véletlenül választott *a* értékekre ellenőrzi, hogy a vizsgált szám *a* alapú álprím-e, továbbá megvizsgálja, hogy van-e nemtriviális négyzetgyöke 1-nek modulo n.
A második részbeli elutasítás helyessége az alábbi tételen alpul:

Tétel: Ha p páratlan prím, akkor az $x^2 = 1 \bmod p$ kongruenciának, csak 2 megoldása van, az $x = 1$ és az $x = -1$.

Def.: Az x szám 1 nem triviális négyzetgyöke modulo n, ha megoldása az $x^2 = 1 \bmod n$ kongruenciának, és nem egyezik meg a triviális –1,1 négyzetgyökökkel.

Következmény Ha az 1-nek létezik nemtriviális négyzetgyöke modulo n, akkor az n összetett szám.

KULCSPÁROK GENERÁLÁSA

A prímteszt használja a következő TANÚ(a,n) algoritmust.
Ehhez legyen $n-1 = 2^t u$, ahol $t \geq 1$ és u páratlan.

```
tanu(a,n) {
   x[0]=mhatv(a,u,n) ;
   for (i=1;i<=t;i++) {
      x[i]=(x[i-1]*x[i-1])%n ;
      if (x[i]==1 and x[i-1]!=1 and x[i-1]!=n-1)
         return True;
   }
   if (x[t]!=1)return True;
   return False;
}
```

Lemma.
Ha a tanu(a,n) eljárás Igaz értéket ad vissza, akkor az n szám összetett.

Bizonyítás.
Két esetben kaphatunk Igaz értéket, ha x(i–1) nem triviális négyzetgyöke 1-nek modulo n,
vagy ha nem teljesül n-re a Fermat tétel.
Mindkét esetben adódik, hogy n nem lehet prím.

KULCSPÁROK GENERÁLÁSA

prímtesztelés*

```
Miller-Rabin(a,b,s) {
for (j=1;j<=s;j++) {
   q:=VeletlenGeneralt(a,b) ;
   if tanu(q,b) return tuti nem prím
}
return valószínűleg prím
```

Tétel.
Az n páratlan összetett számnak legalább (n–1)/2 darab összetettséget igazoló tanúja van.

Következmény:
Legyen n > 2 páratlan egész, s pedig pozitív egész.
A `Miller-Rabin(n,s)` teszt tévedési valószínűsége legfeljebb 2^{-s}.

Bizonyítás:
A fenti tétel alapján ha n nem prím, akkor a Miller-Rabin(n,s) teszt minden iterációja 1/2 valószínűséggel felfedez egy tanút. Így annak a valószínűsége, hogy s lépésben nem találunk egyetlen tanút sem, kisebb, mint 2^{-s}.

Mersenne prímek

Def: Mersenne-prímnek nevezzük a kettő-hatványnál eggyel kisebb, azaz a 2^n-1 alakban felírható prímszámokat, ahol n szintén prímszám.

2008-ban fedezték fel a 45-ödik Mersenne-prímet, ez a $2^{43112609}-1$ szám, amely 12 978 189 számjegyű.
Ez a jelenleg ismert legnagyobb prímszám.

RSA támadások

Kulcskereséses támadás

Számítási idő mérése

Hibás alkalmazás

A modulus faktorizációján alapuló támadások

Martin Gardner, a Scientific American világhírű rovatvezetőjének 1977-es gondolatai: *„Ha a ma ismert legjobb algoritmust és a leggyorsabb számítógépeket használjuk, egy ilyen 125 jegyű RSA kulcs megfejtésére, Rivest becslése szerint a szükséges mefejtési idő körülbelül 40 quadrillió év! Ez azt jelenti, hogy praktikusan a belátható jövőben reménytelen az RSA kulcsok faktorizáció útján történő megfejtése. Ugyanakkor maga Rivest és kollégái is elismerik, hogy semmilyen elméleti bizonyítékuk nincs arra, hogy az RSA titkosítási eljárás megfejthetetlen.”*

Hibás alkalmazás

A. A hálózat egy részénél az **e** nyilvános kulcs azonos és kicsi. A támadás bemutatásához legyen **e**=3. Ha egy adó három helyre is elküldi ugyanazt a nyílt blokkot. akkor

$$y_1=x^3 \text{ modulo } m_1$$
$$y_2=x^3 \text{ modulo } m_2$$
$$y_3=x^3 \text{ modulo } m_3$$

Ekkor a kínai maradéktétel alapján az x üzenet megkapható. A támadás kivédésének két útja van: ne válasszunk a hálózat csomópontszámával összemérhető nyilvános kulcsot, és/vagy gondoskodjunk arról, hogy ne legyen két azonos üzenet. Ez történhet például egy néhány byte-nyi pszeudovéletlen blokk beszúrásával.

Hibás alkalmazás

B. Átgondolandó veszélyforrást jelent az RSA-leképezés művelettartása is. Mind a rejtjelzés, mind a digitális aláírás művelettartó a moduláris szorzásra nézve. Vagyis

$$f(x_1)*f(x_2)=f(x_1*x_2)$$

Ez azt jelenti, hogy ha ismerjük az x_1 x_2 üzenetekhez tartozó digitális aláírást, akkor egyszerű szorzással (a titkos kulcs ismerete nélkül is) képezhetjük a nyílt üzenetek szorzatához tartozó digitális aláírást. Ha azonban az eredeti nyílt szövegek strukturáltak, szerkezetük van, akkor gyakorlatilag nincs valószínűsége, hogy a szorzat üzenet is ilyen lesz, vagyis a szorzat nem lesz egy lehetséges nyílt, s ekkor ennek aláírása is haszontalan a támadó szempontjából.

C. Létezik két olyan felhasználó, hogy az általuk használt modulus azonos. Ekkor mindkét felhasználó ki tudja számítani a másik titkos **d** kulcsát.[5] Tehát nem szabad, hogy a hálózatban azonos modulusokat használjanak

Prímszámok véletlen hibája

Ha az RSA modulus egy Carhmichael szám valamely osztója, akkor a Fermat-féle ciklus hossza éppen 1, így az üzenek könnyen visszafejthető.

n	Carmichael számok száma
<10	-
$<10^1$	-
$<10^2$	1
$<10^3$	7
$<10^4$	16
...	..
$<10^9$	255

A Carmichael számok aránya ismeretlen.