Önszervező bináris keresőfák

Vágható-egyesíthető halmaz adattípus

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

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

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

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

$$H = H_1 \cup H_2$$

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

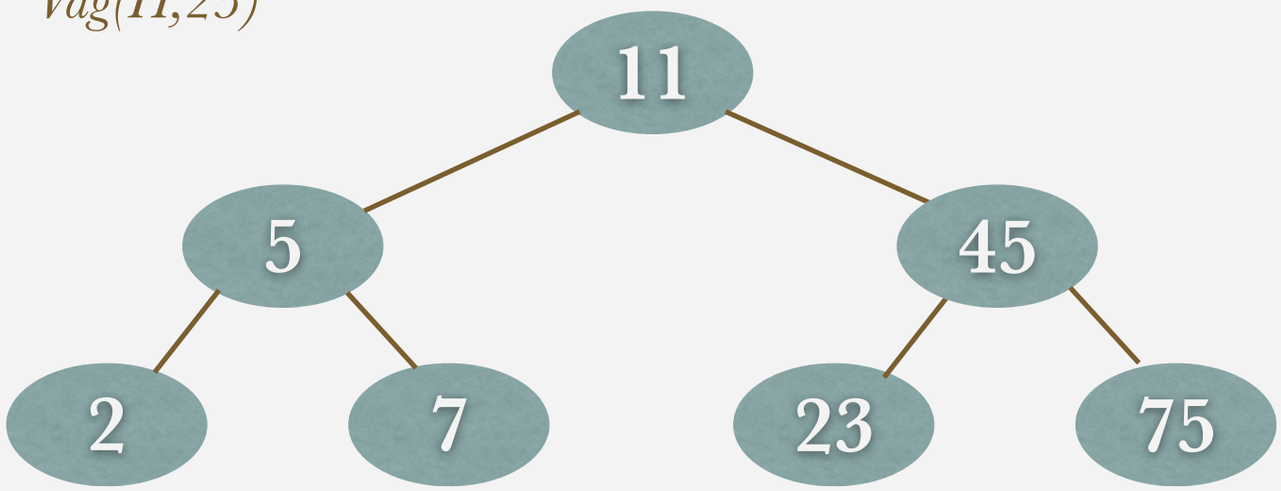
Vágás

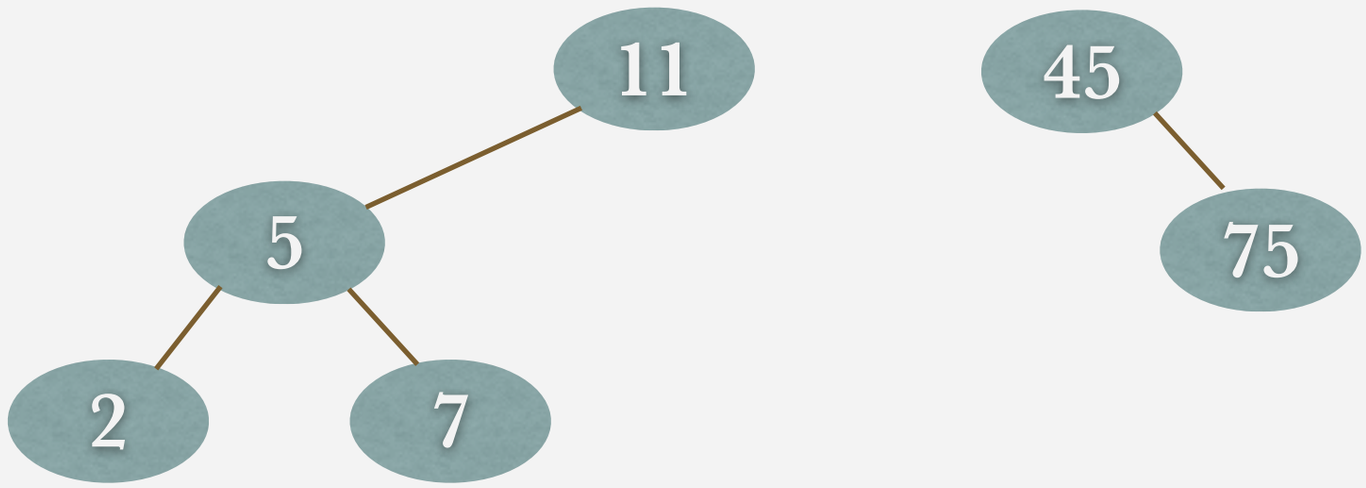
A keresési útvonal mentén feldaraboljuk a fát, majd a megfelelő részfákat összekapcsoljuk.

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

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

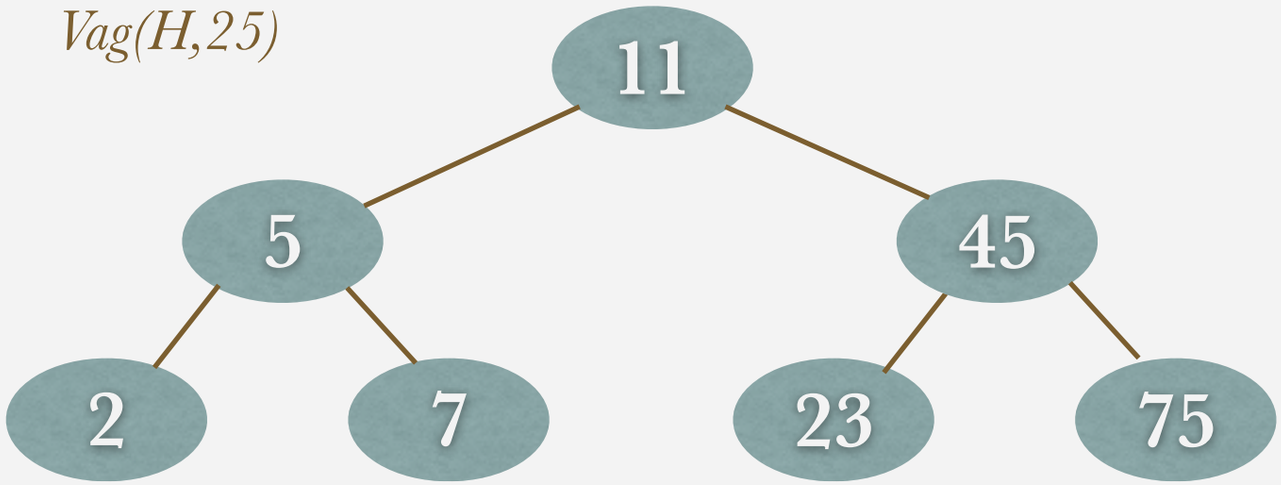
Vag(H,23)

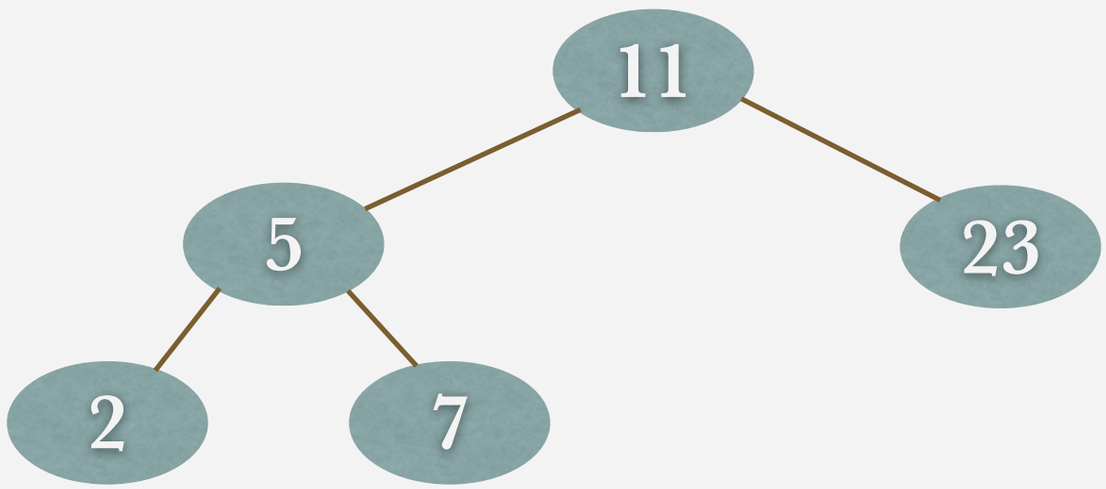


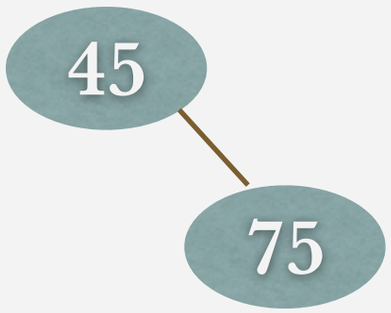


$$H=\{2,5,7,11,23,45,75\}$$

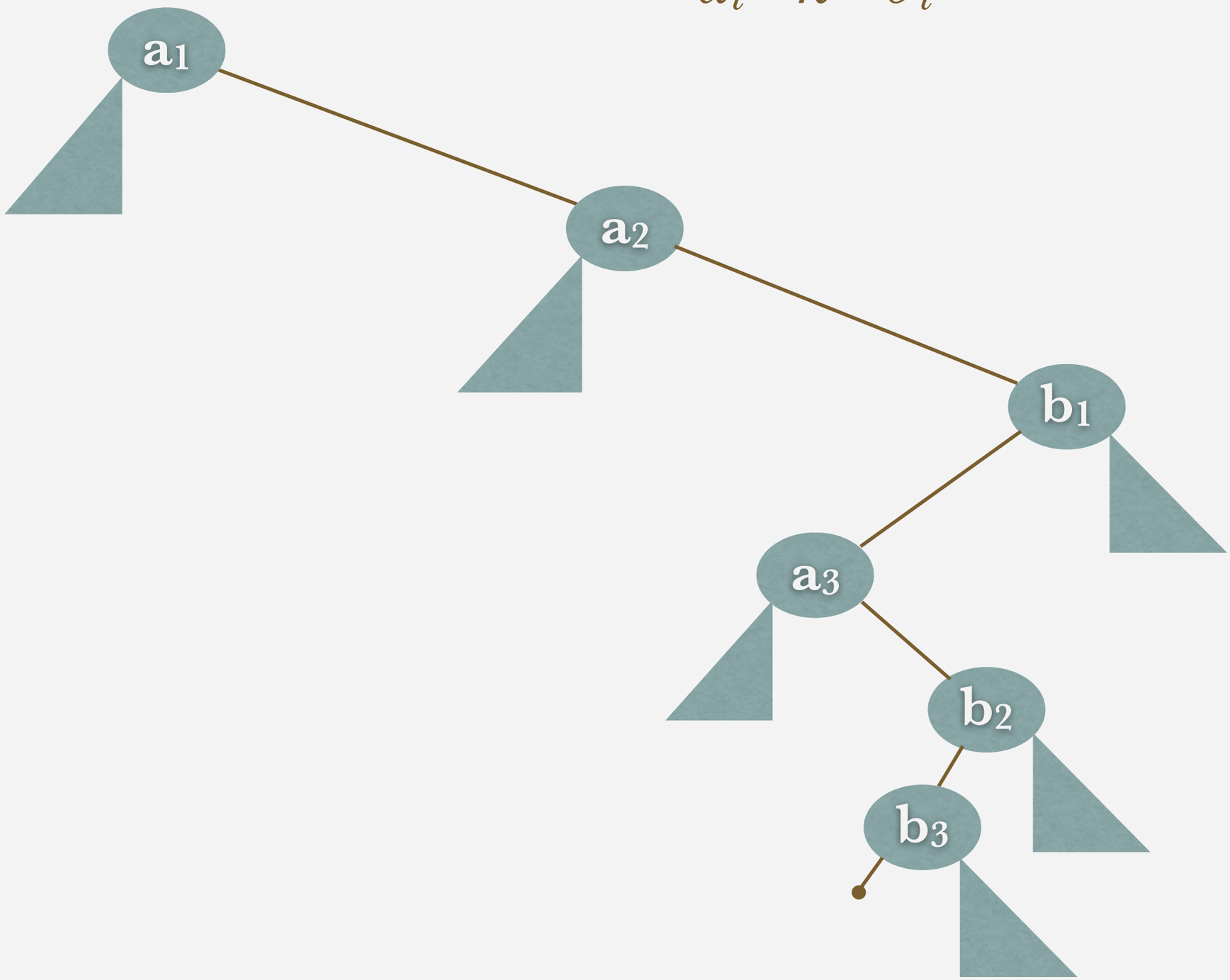
Vag(H,25)





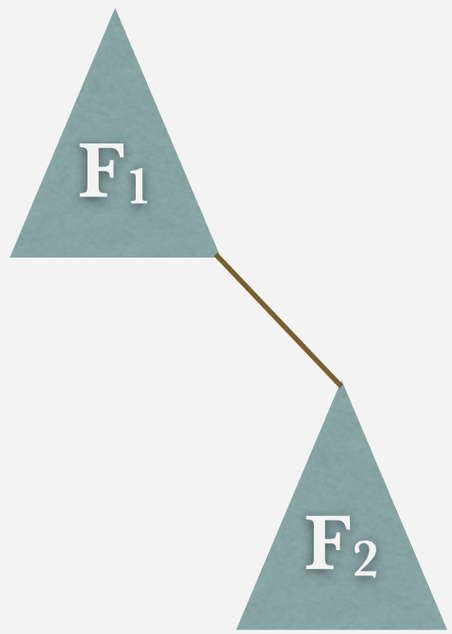


$$a_i<k<b_i$$



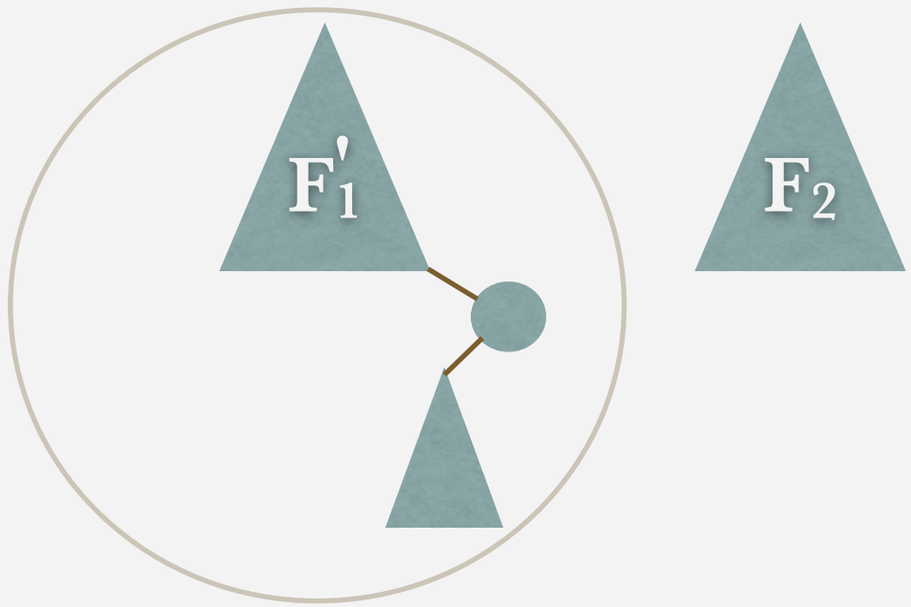
Egyesítés

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



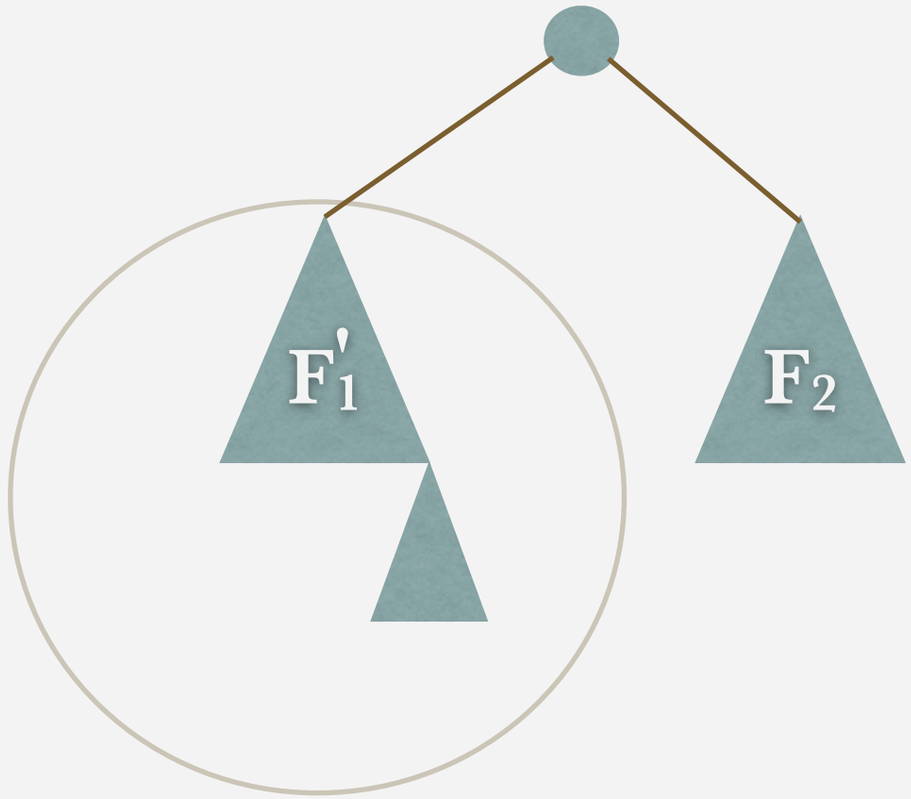
Az egyesítés futási ideje legrosszabb esetben O(1)

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



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

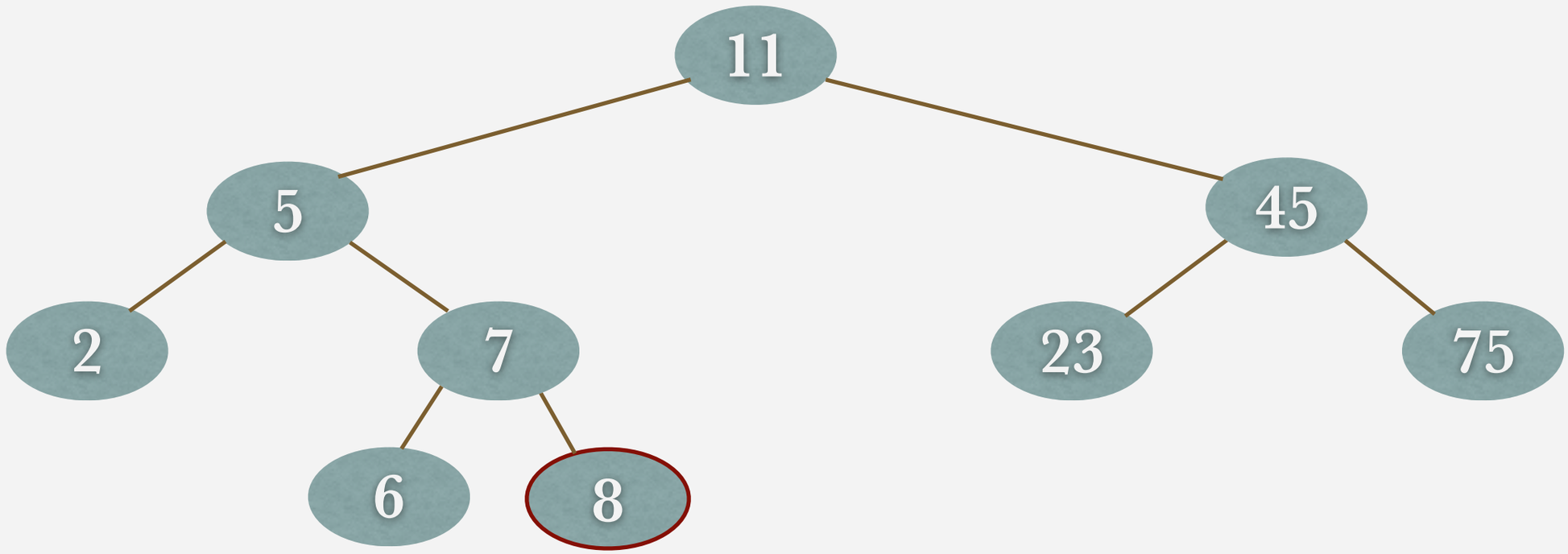
Ha a max(F_1)≤min(F_2) feltétel teljesül:



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

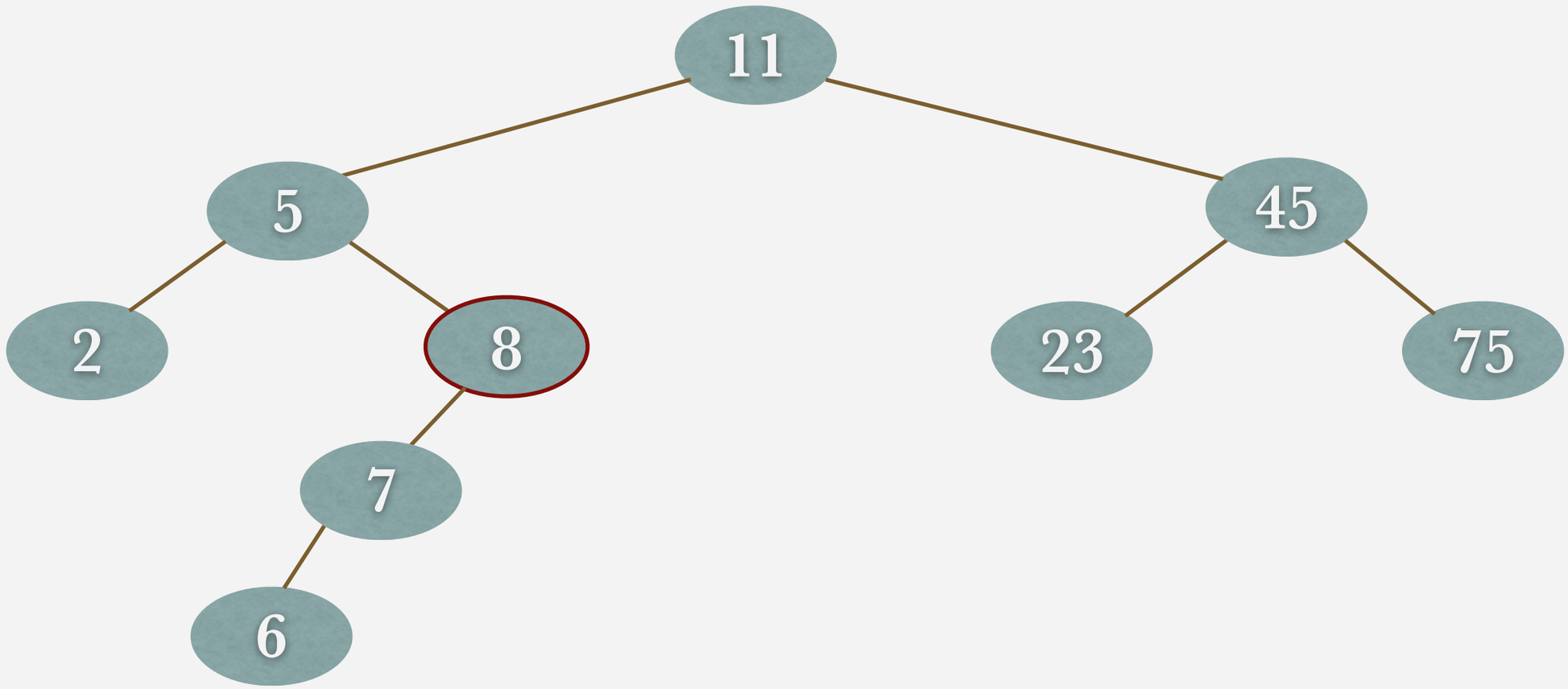
felforgat

F.ff(fapont p) ;



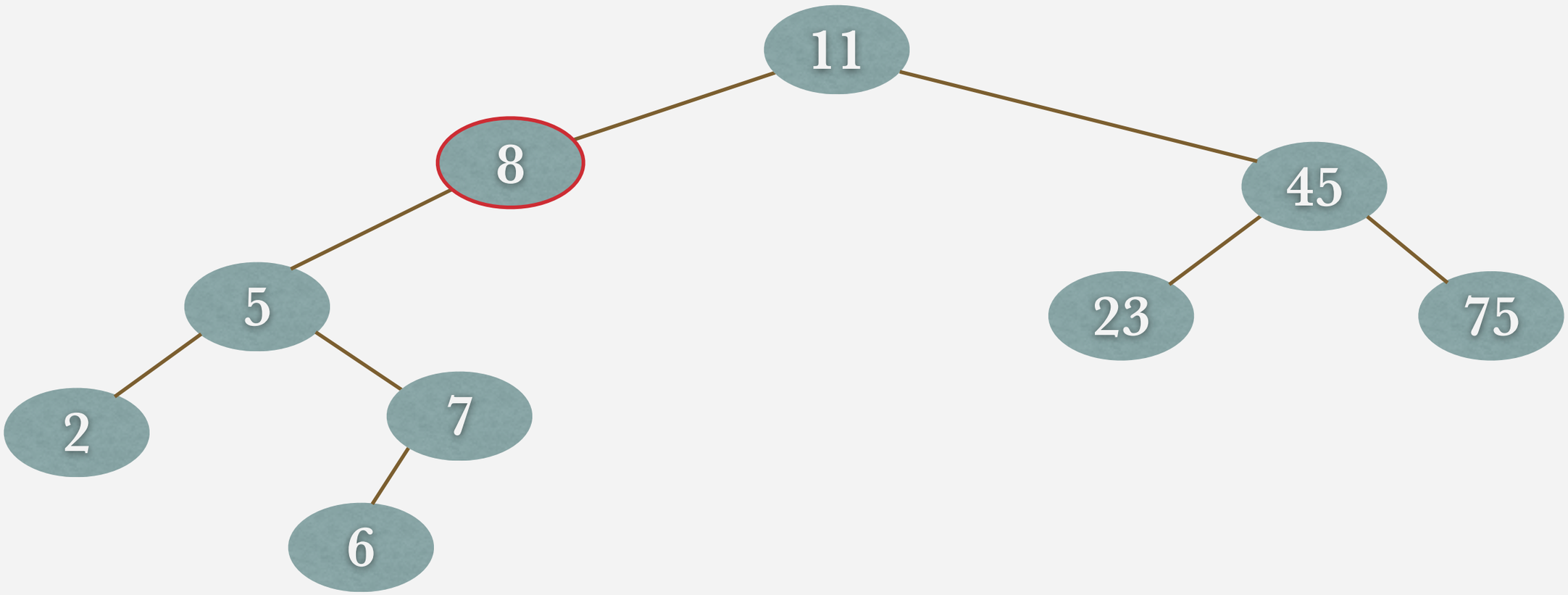
felforgat

F.ff(fapont p) ;



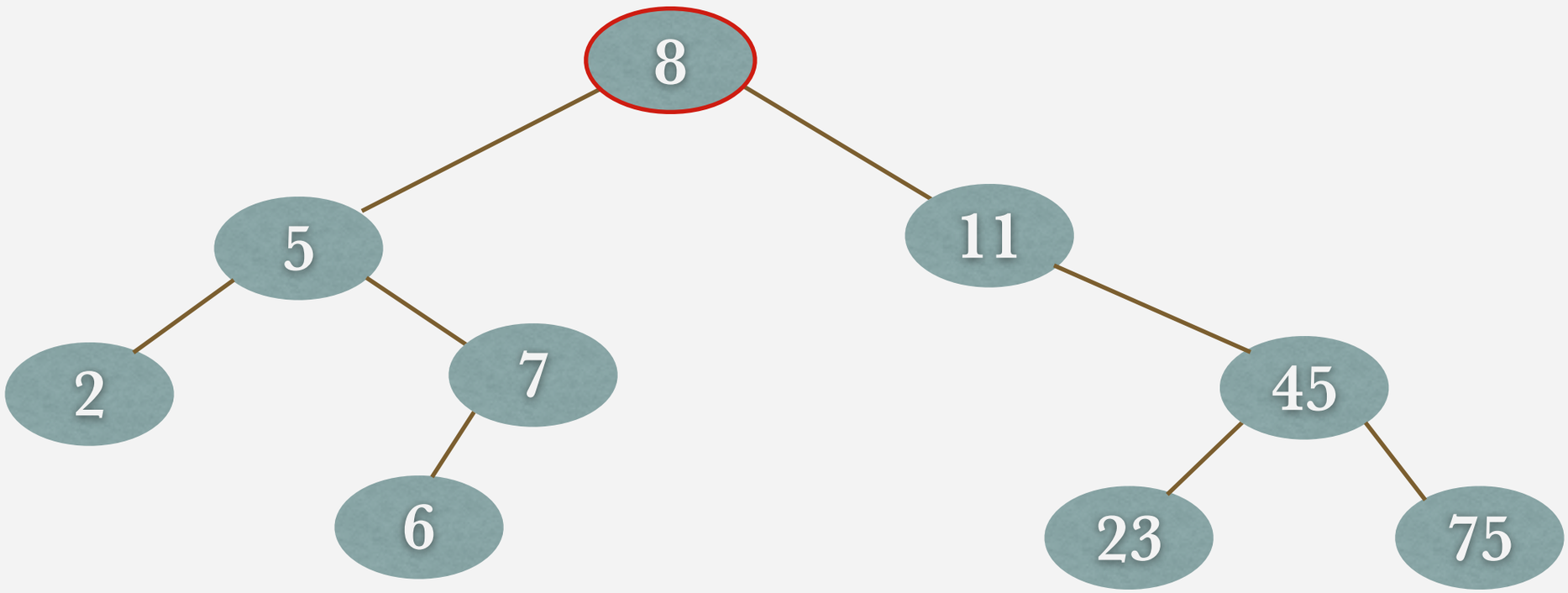
felforgat

F.ff(fapont p) ;



felforgat FF(p)

F.ff(fapont p) ;

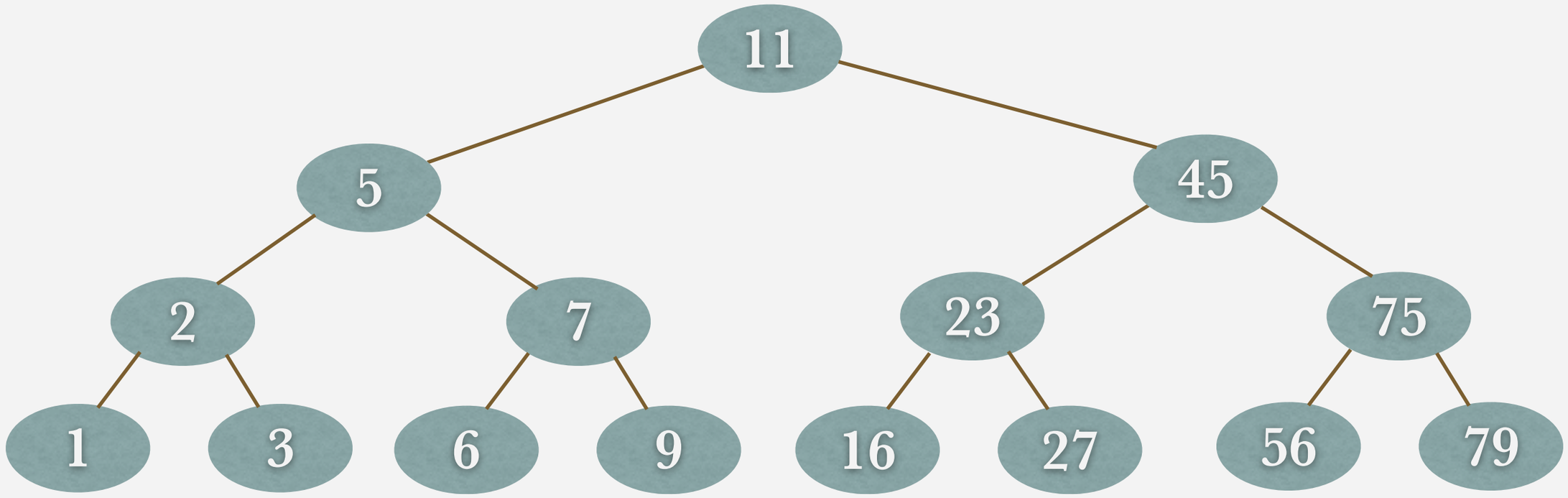


A felforgat futási ideje legrosszabb esetben O(h(F))

FF(k)

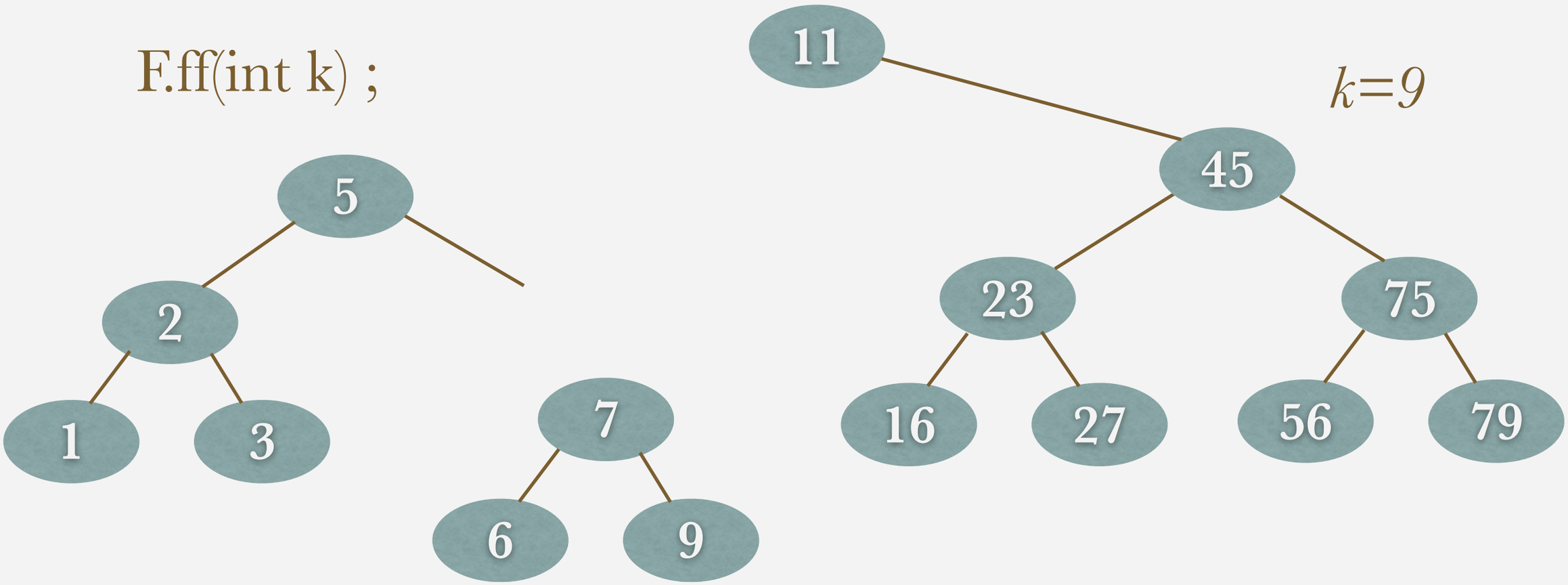
F.ff(int k) ;

k=9



FF(k)

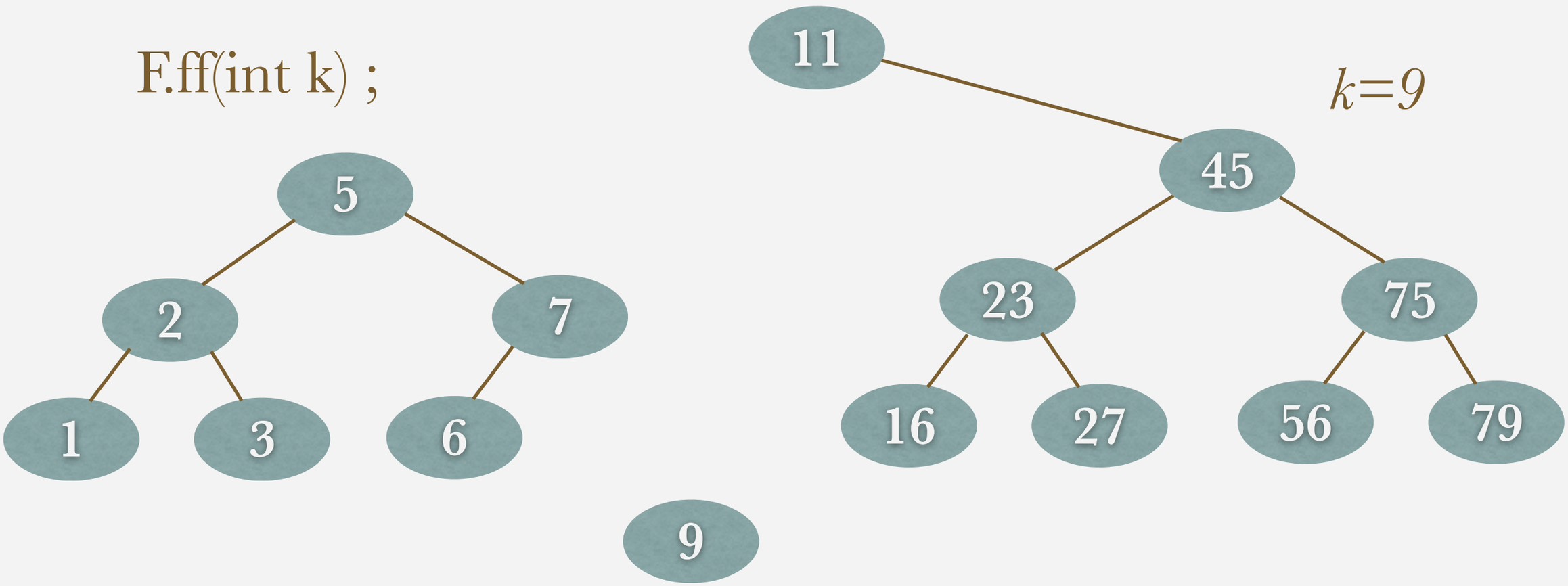
F.ff(int k) ;



k=9

FF(k)

F.ff(int k) ;

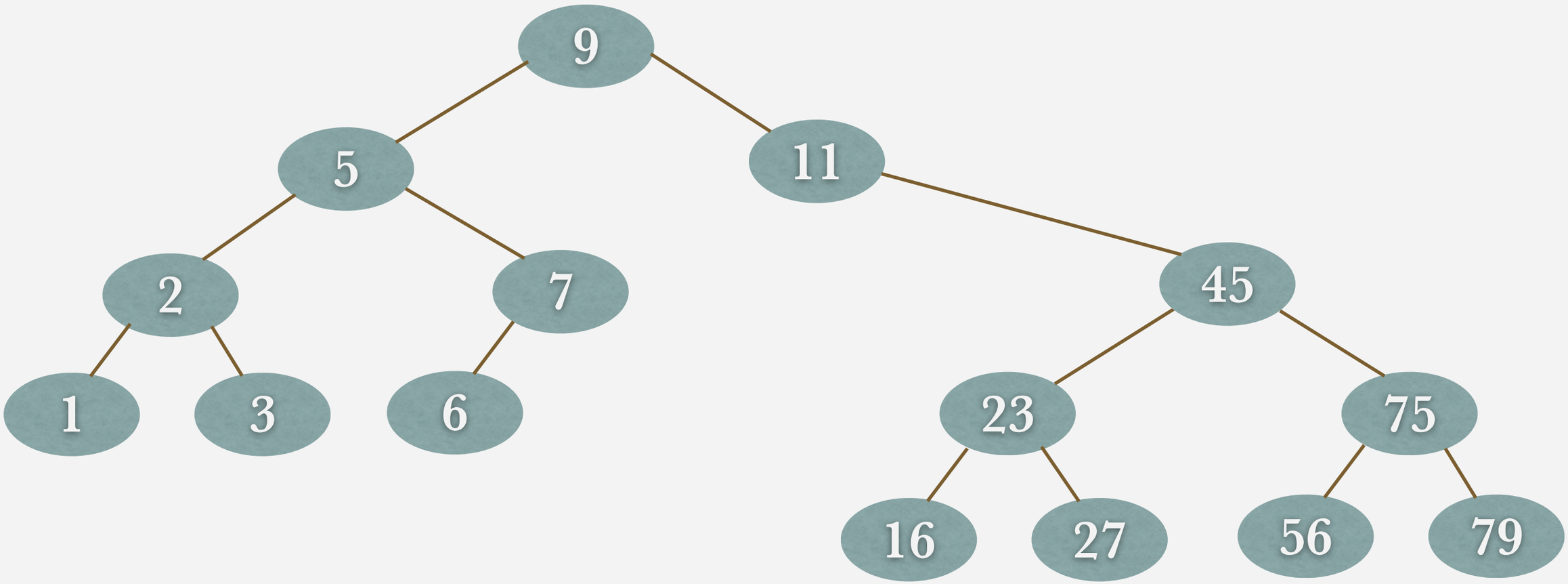


k=9

FF(k)

F.ff(int k) ;

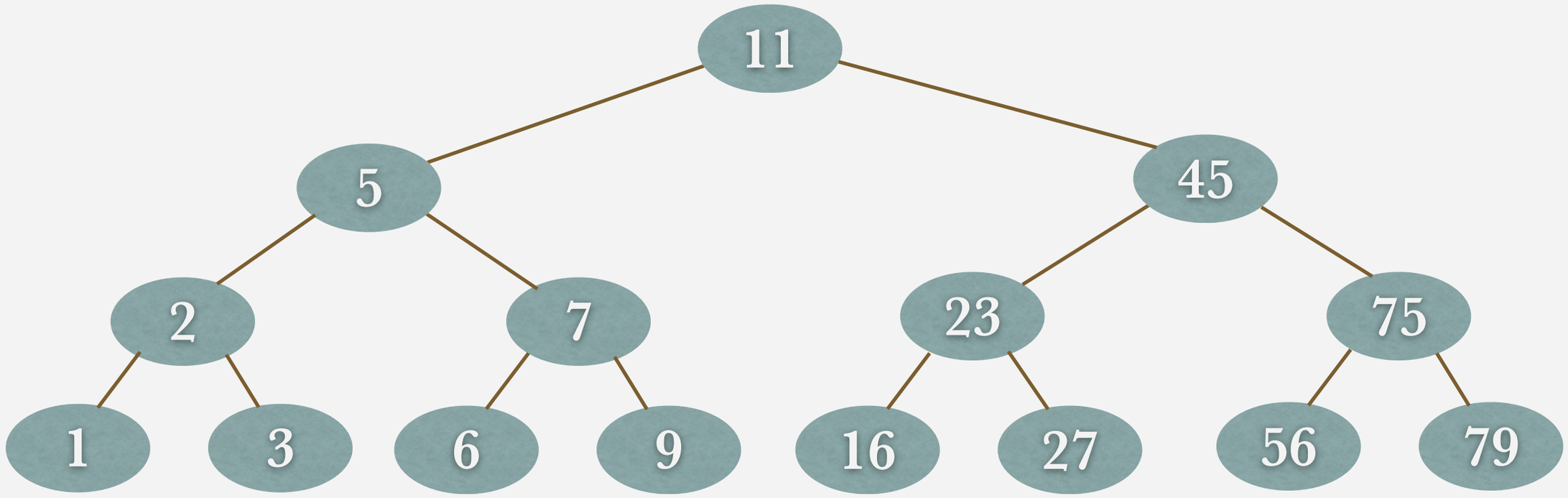
$k=9$



FF(k)

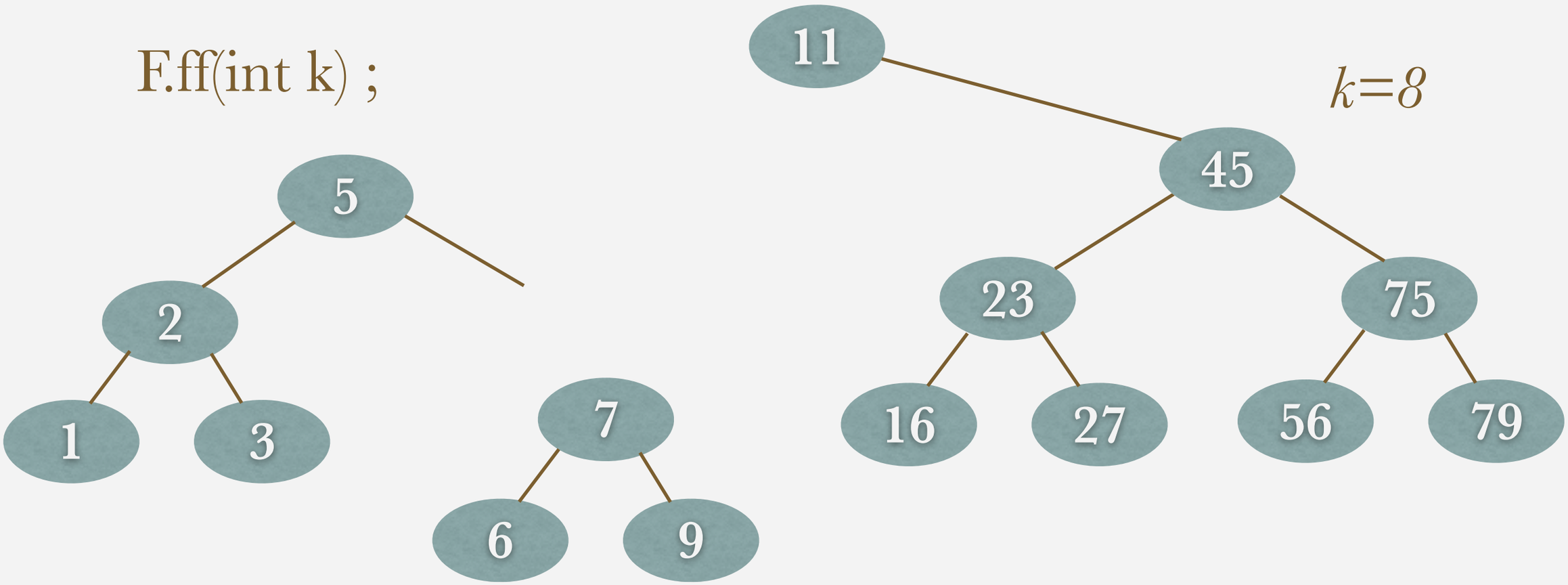
F.ff(int k) ;

k=8



FF(k)

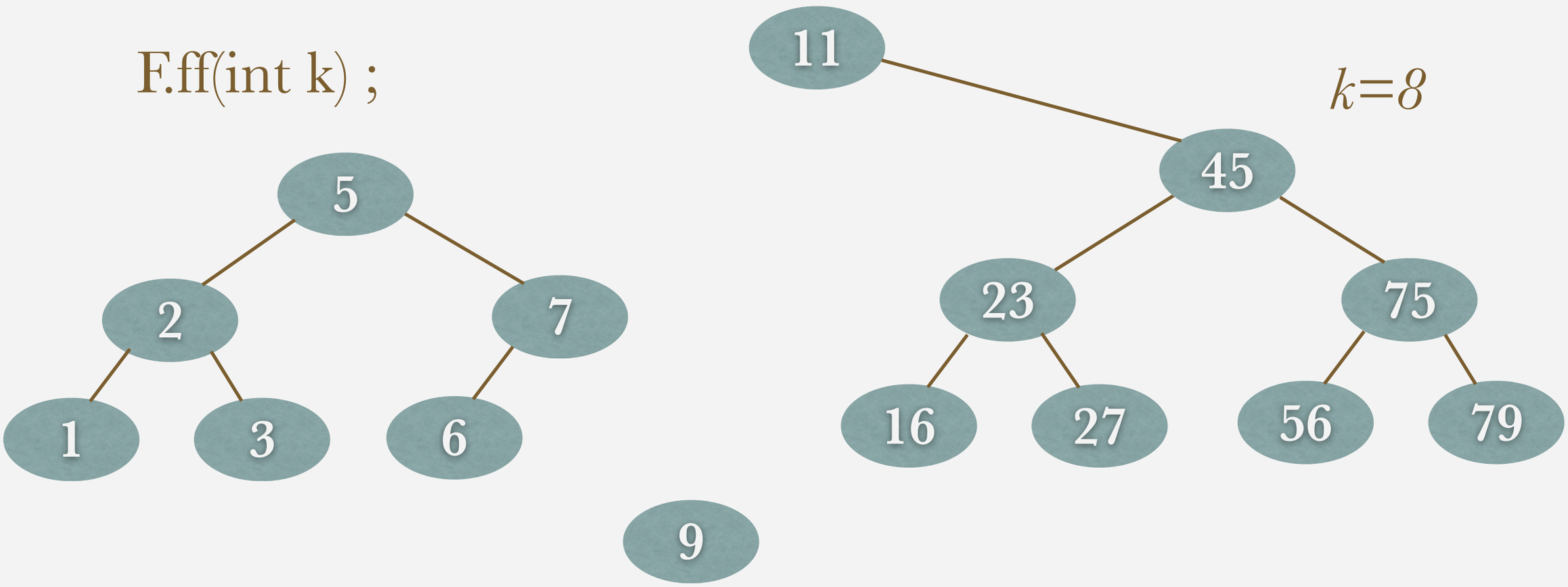
F.ff(int k) ;



k=8

FF(k)

F.ff(int k) ;

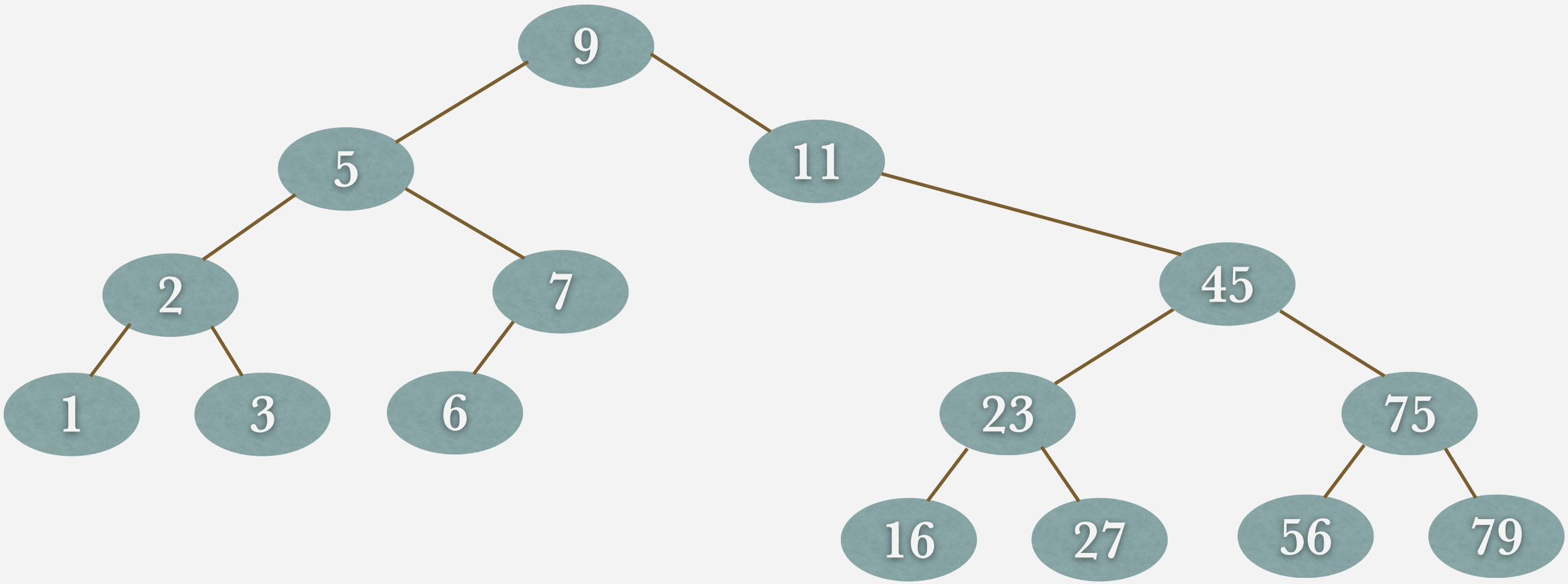


k=8

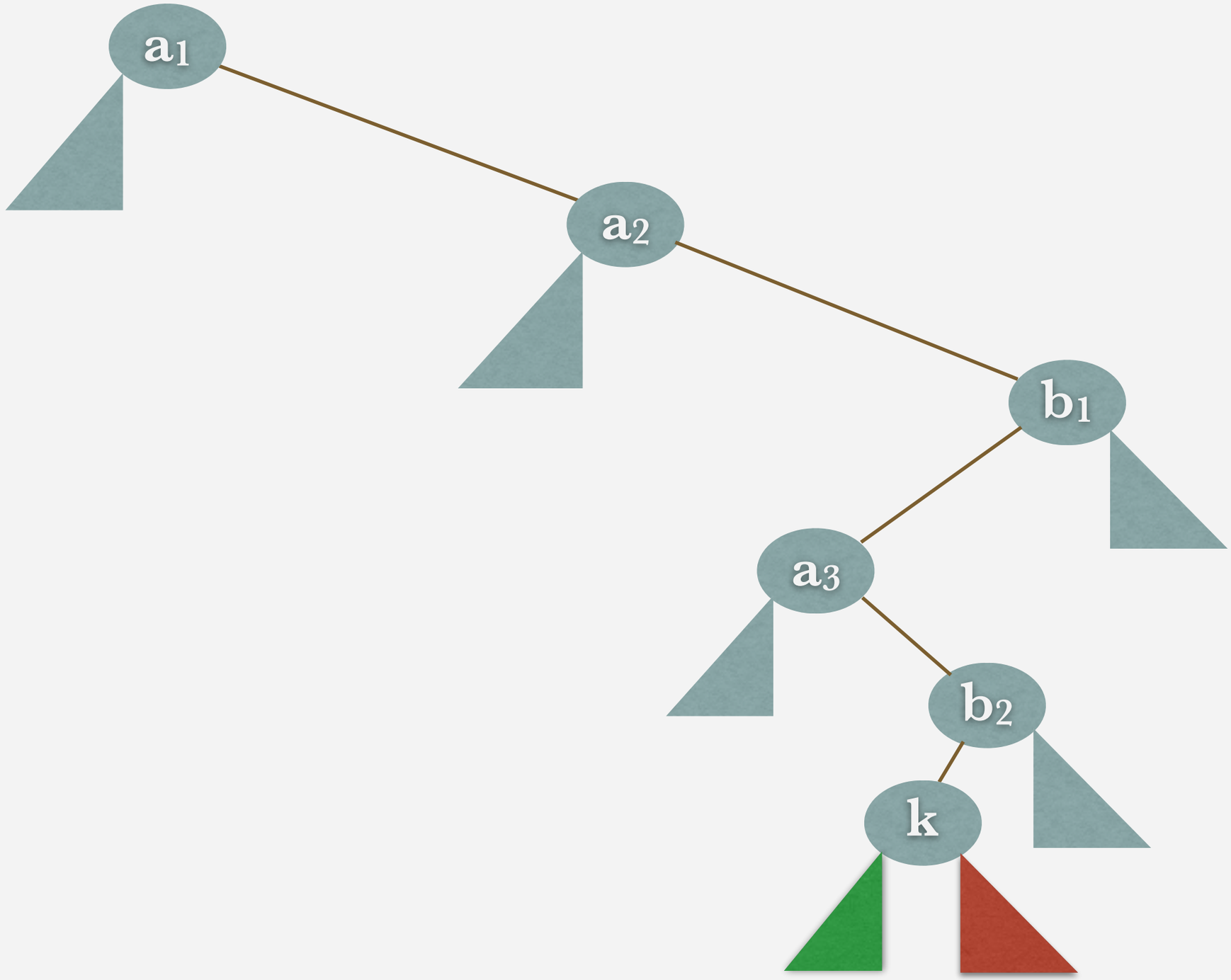
FF(k)

F.ff(int k) ;

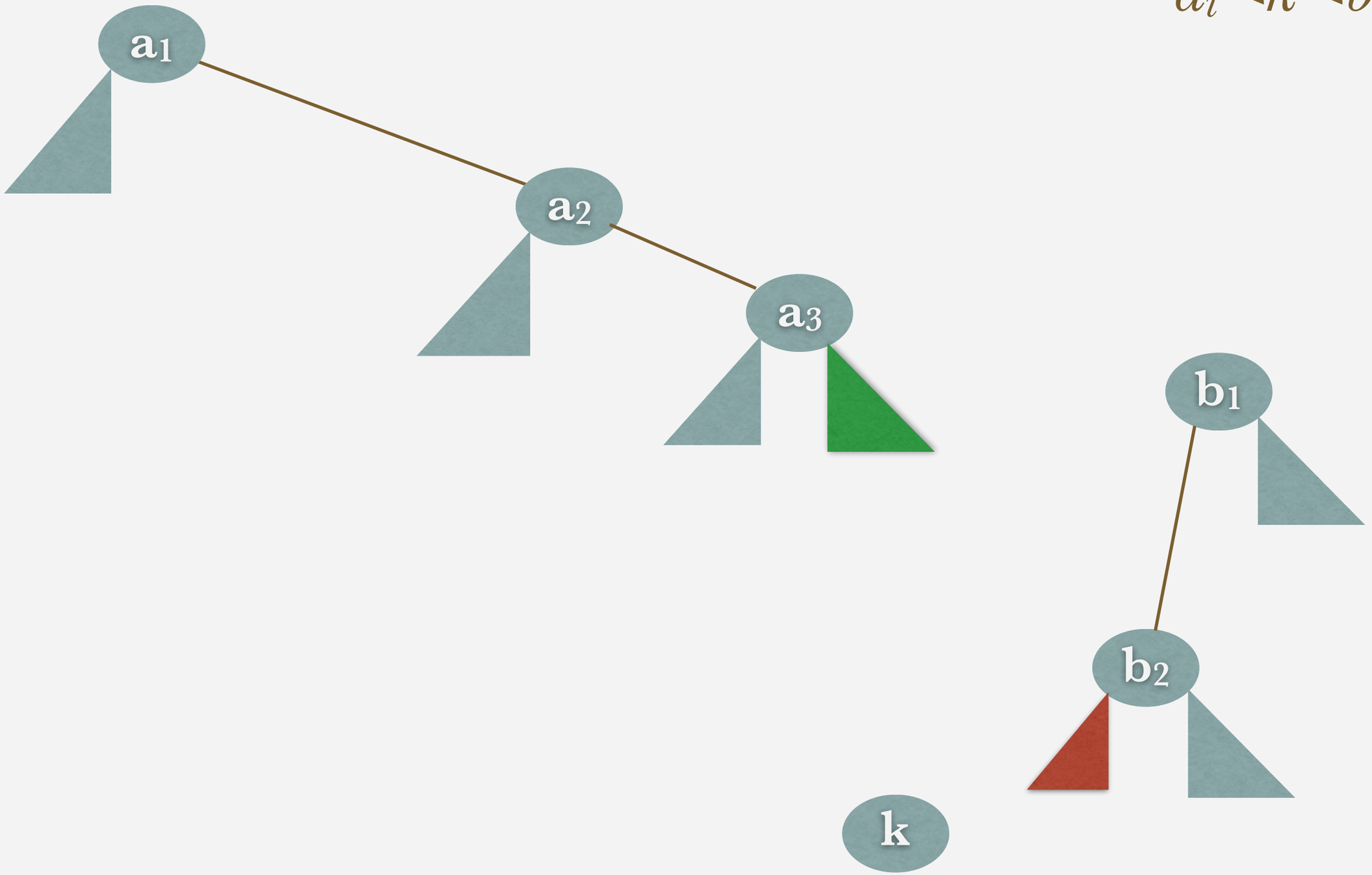
k=8



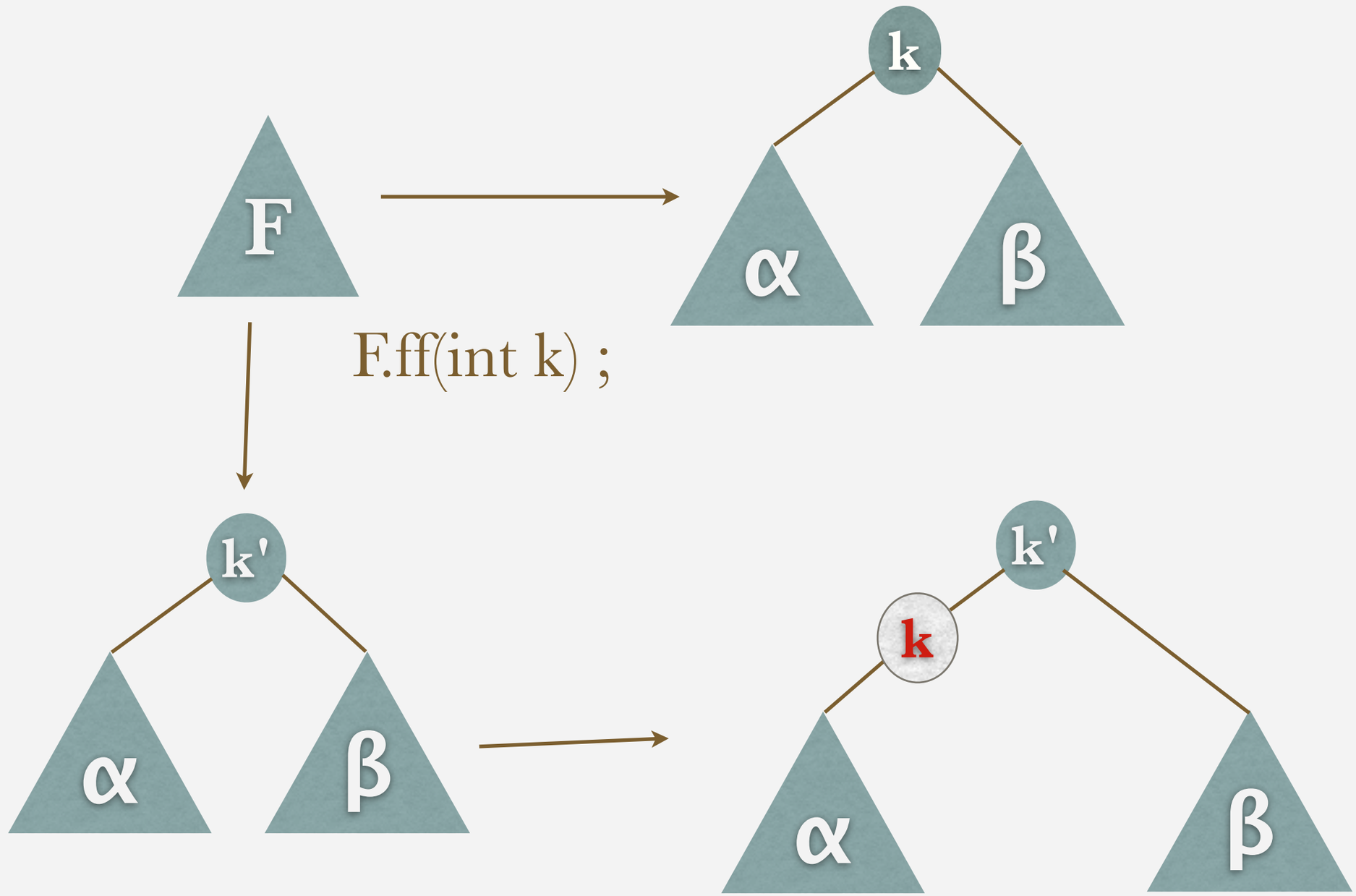
$a_i<k<b_i$



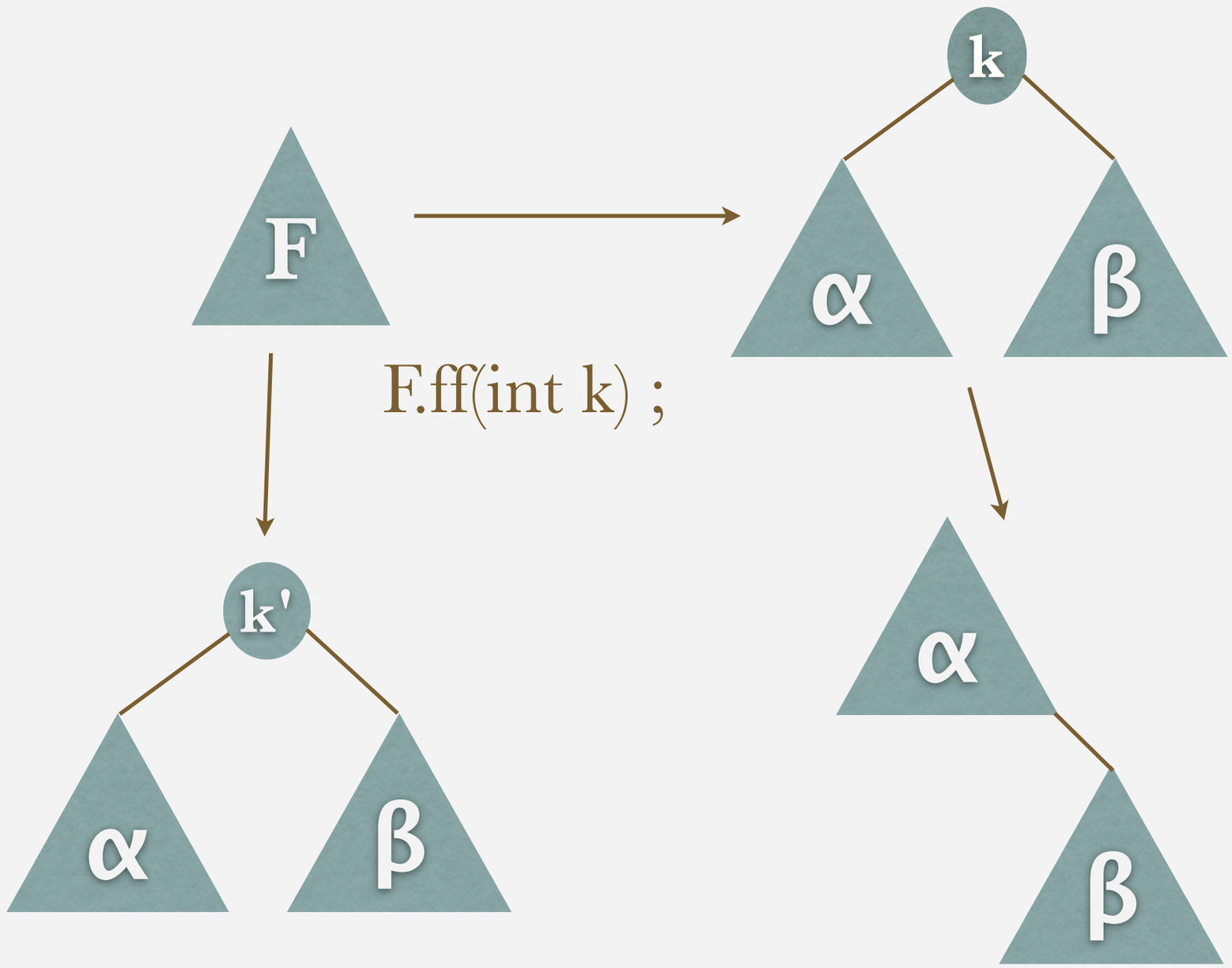
$a_i<k<b_i$



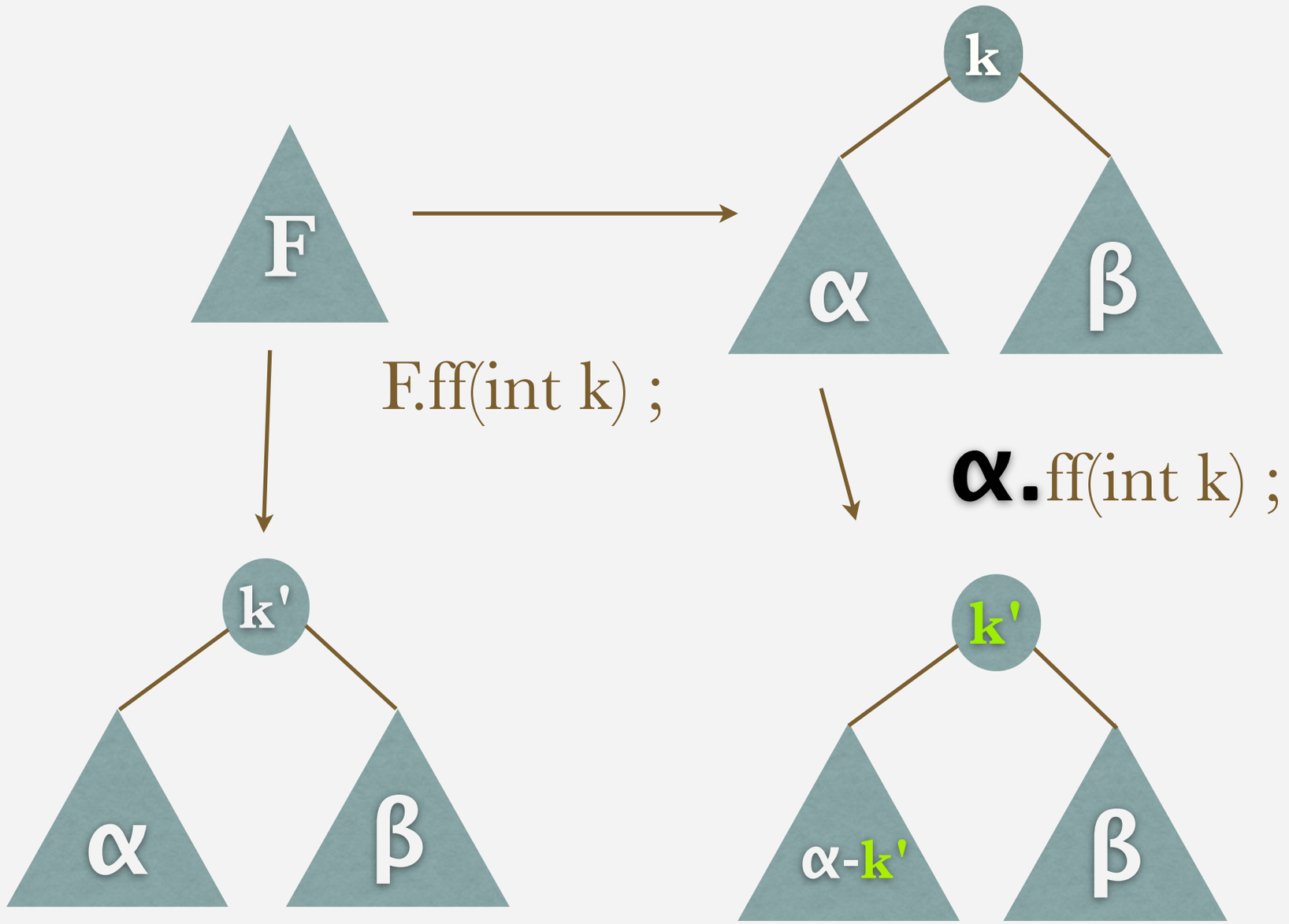
F.bovit(k) ;



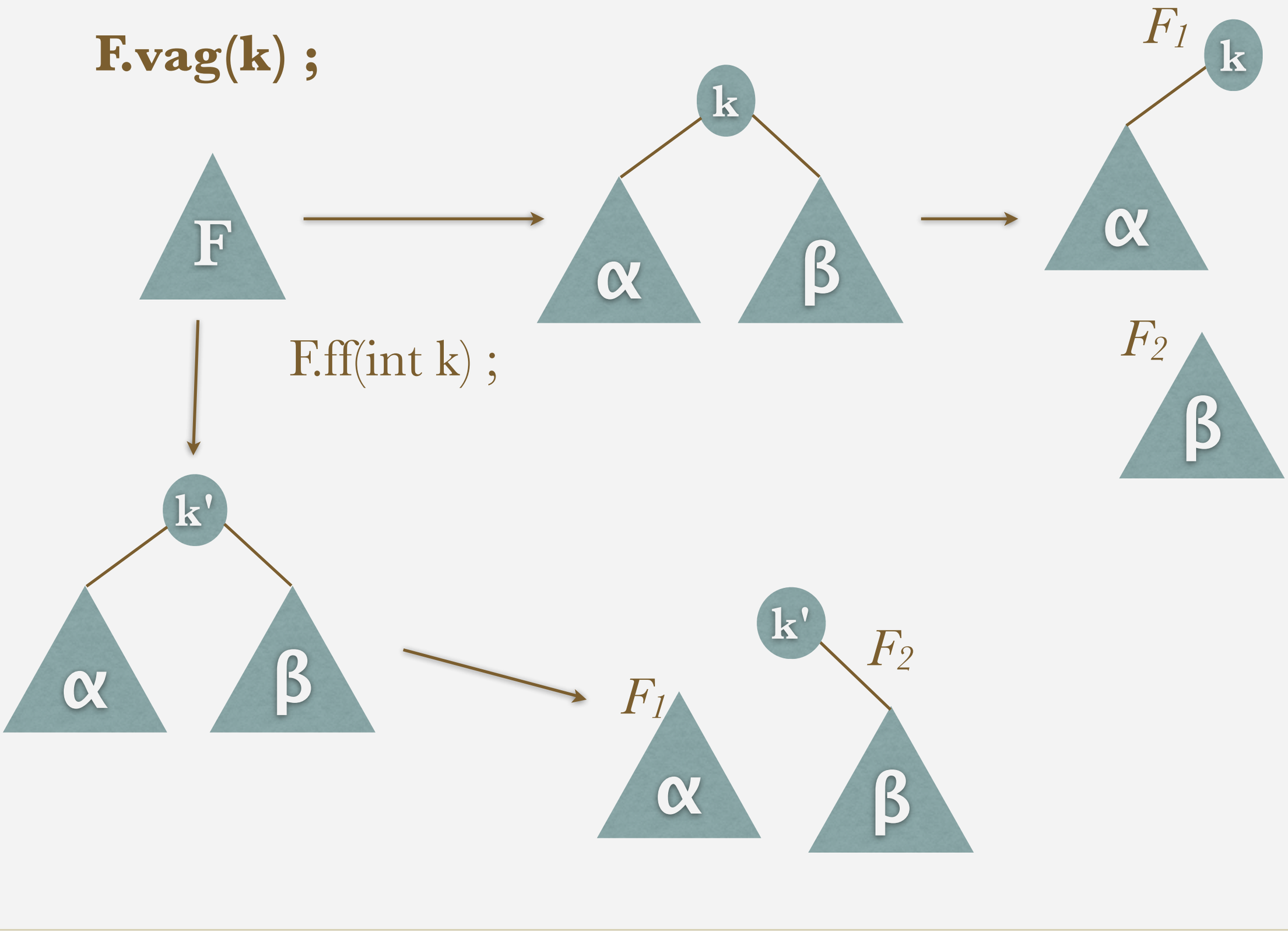
F.torol(k) ;



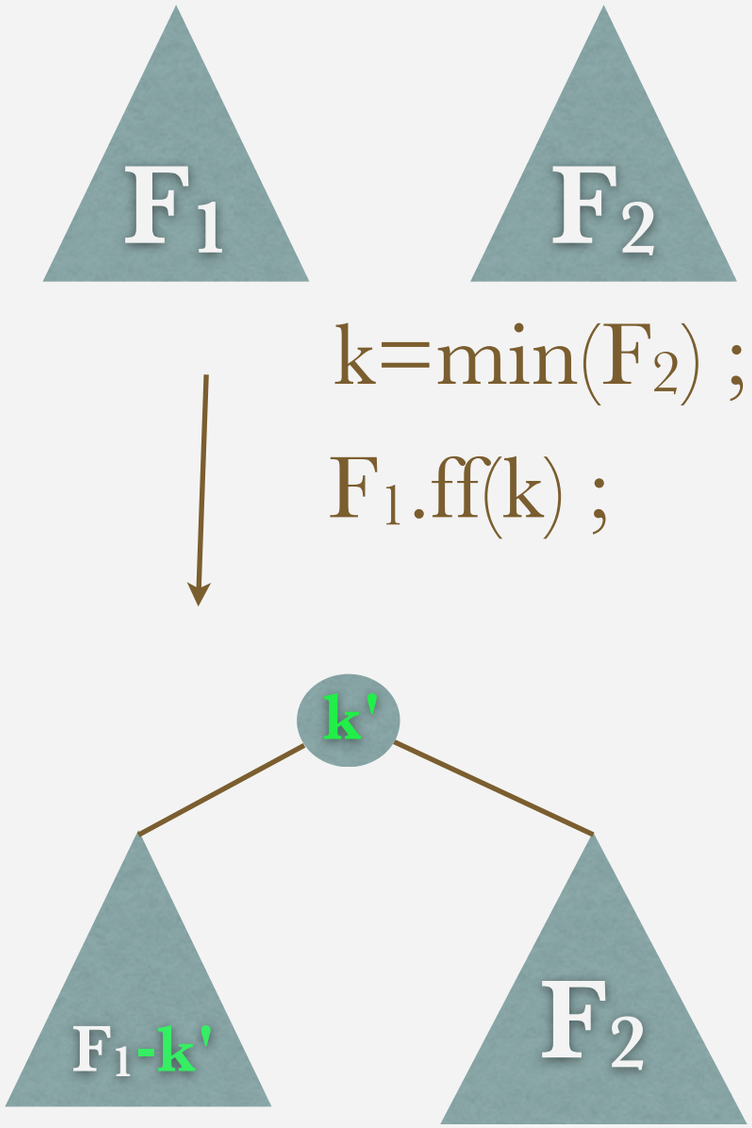
F.torol(k) ;



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

Adott egy $M_1,\ldots,M_n$ műveletsor.

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 **n** darab, és kezdetben üres verem!*

*Mivel **torol(k)** futási ideje legrosszabb esetben **n** (k=n esetén) és a műveletekből **n** darab van, így a futási idő a legrosszabb esetben **n²** lenne…*

Legrosszabb esetre ez a módszer nem mindig éles!

Összesítéses módszer

A teljes műveletsor összesített költségét számítjuk, nem az egyes műveletekét! - T(n)

Minden elem a verembe egyszer kerülhet be és egyszer ki.

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

*így a teljes futási idő T(n)=**O(n)!***

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

Az amortizált költség bármely műveletre

T(n)/n=n/n=1!

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.

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

Példa:

betesz(x) művelet tényleges költsége *1*.
kivesz() művelet tényleges költsége *1*.
töröl(k) művelet tényleges költsége *k*.

betesz(x) művelet amortizált költsége *2*.
kivesz() művelet amortizált költsége *0*.
töröl(k) művelet amortizált költsége *0*.

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

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 m műveletre:

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

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

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

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

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:

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

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

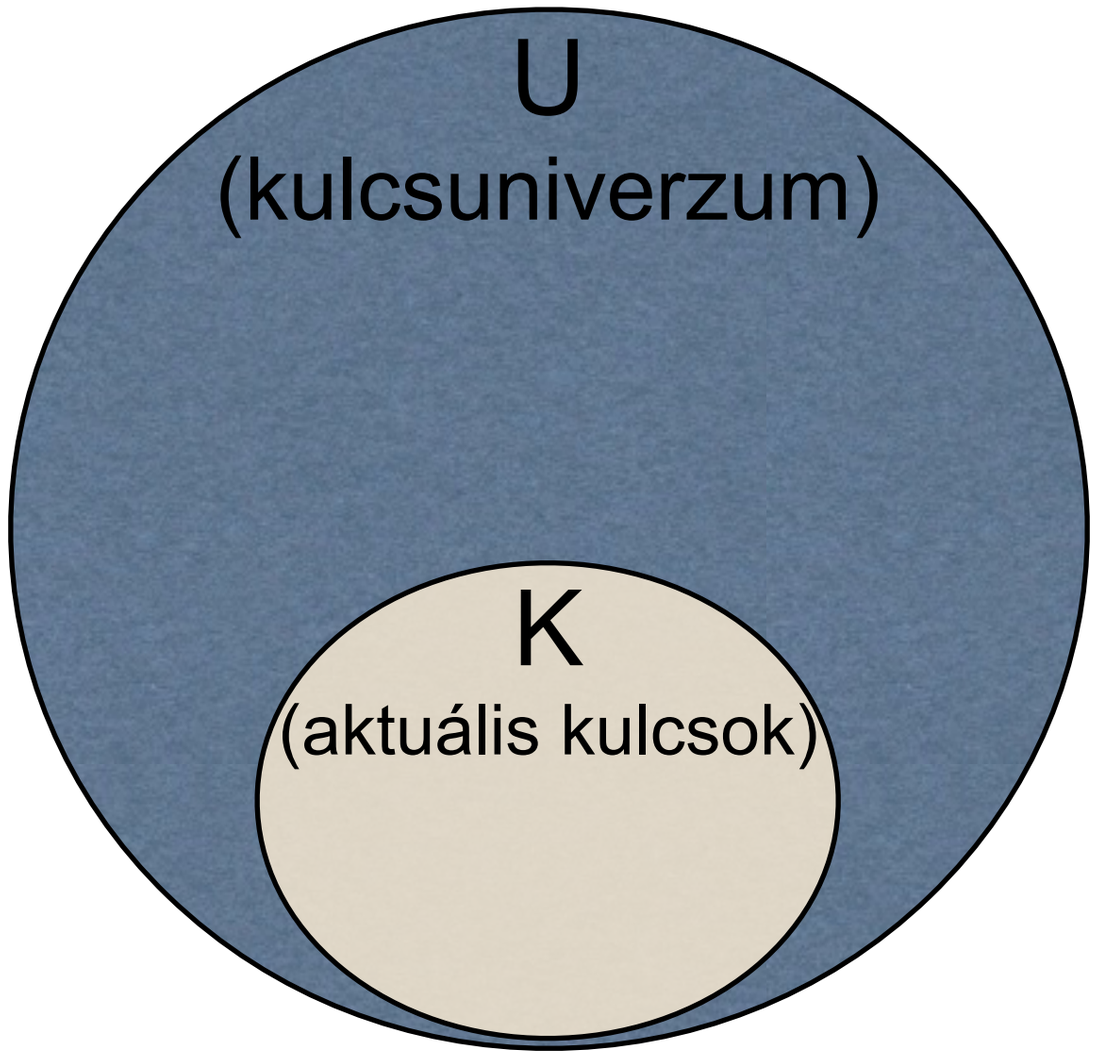
$$\hat{c}_i = c_i + \overbrace{\varphi(D_i)-\varphi(D_{i-1})}^{-k'} = 0$$

O(1)

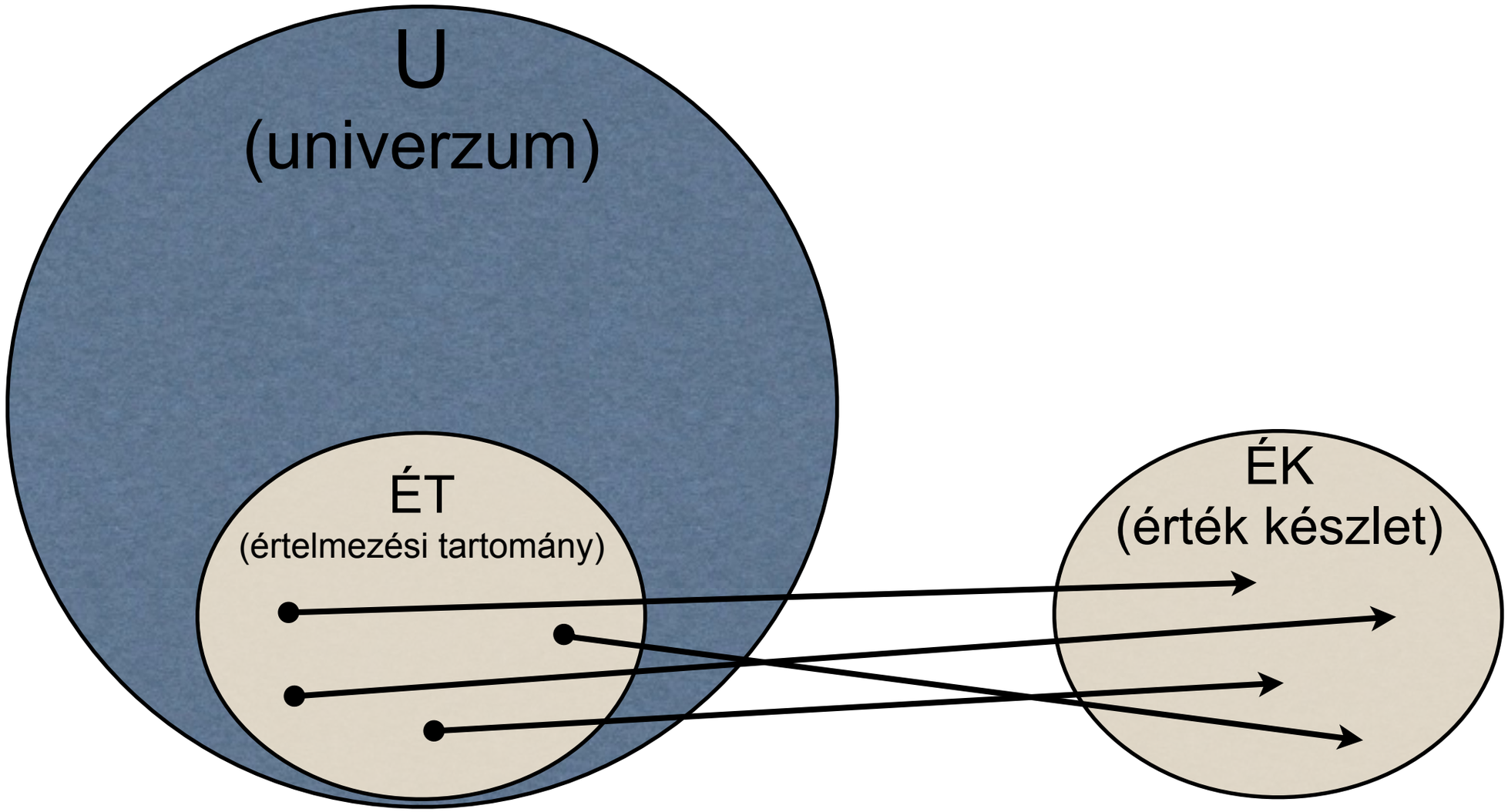
Mindhárom művelet amortizált költsége O(1) ezért tetszőleges m hosszú BVEREM műveletsor költsége O(m)

Hasító táblázatok

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)

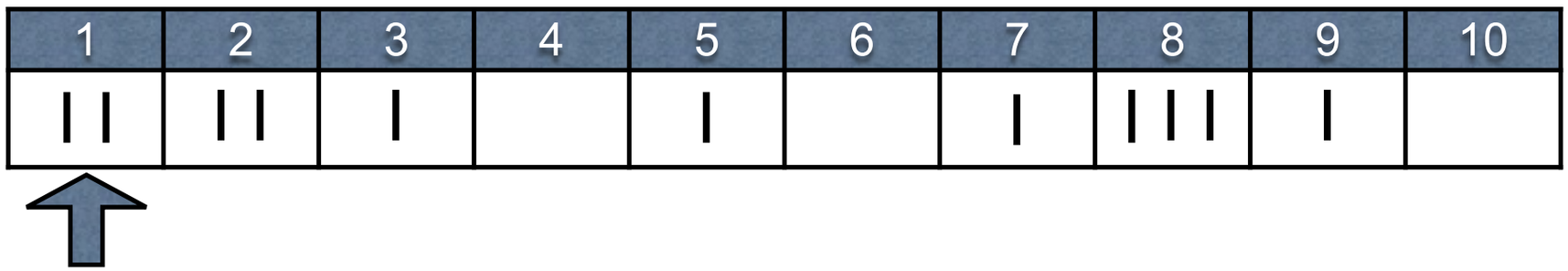
Leszámláló rendezés

felelevenítés

Csak speciális esetre!

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

T



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)

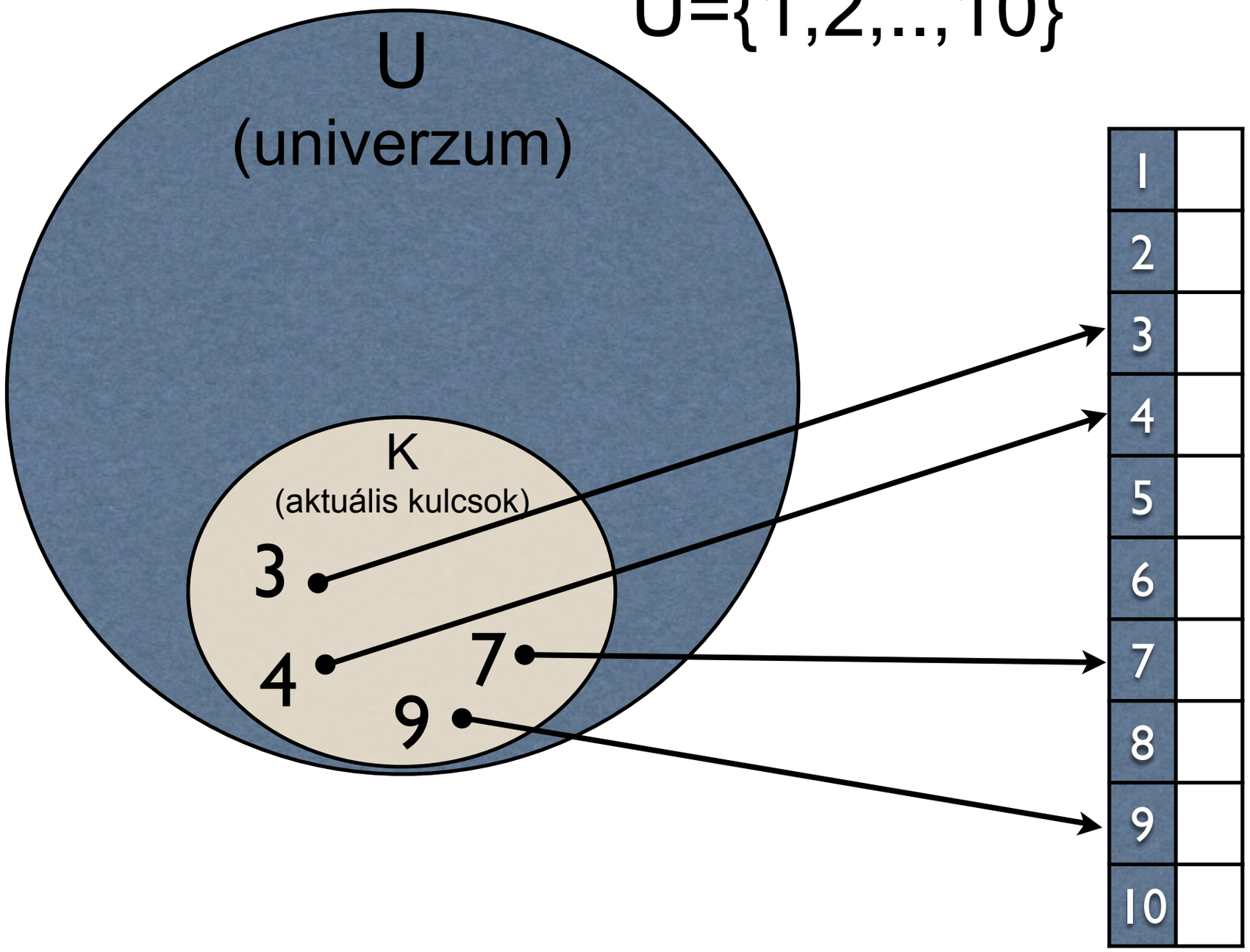
n elem tömbbe tétele O(n)

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

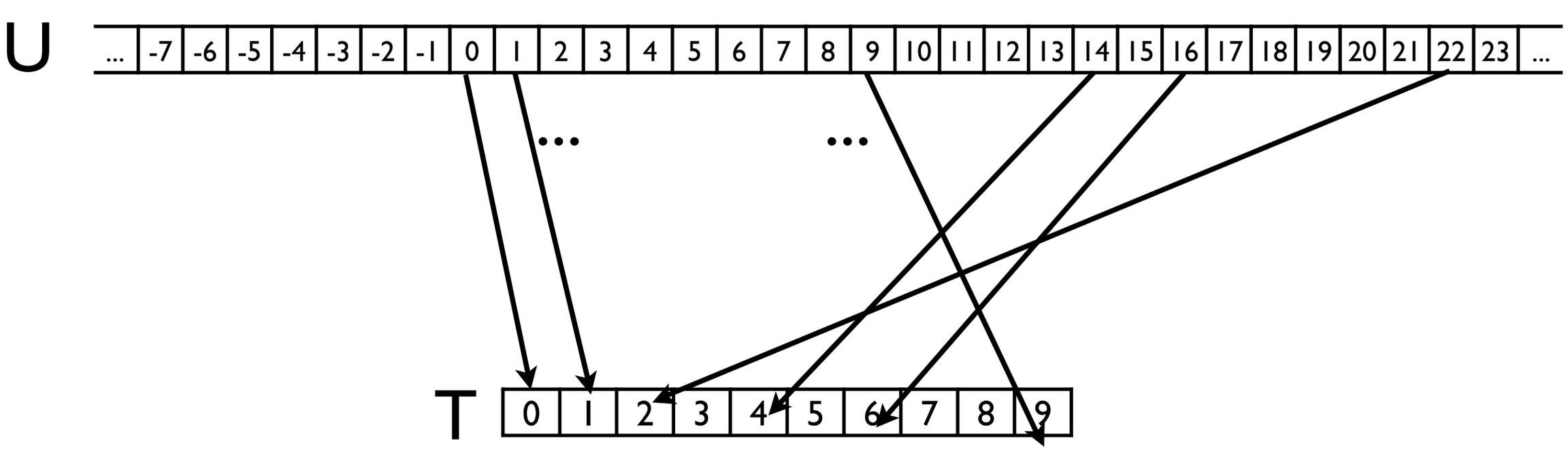
U={0,1,2,..,99}

```
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?

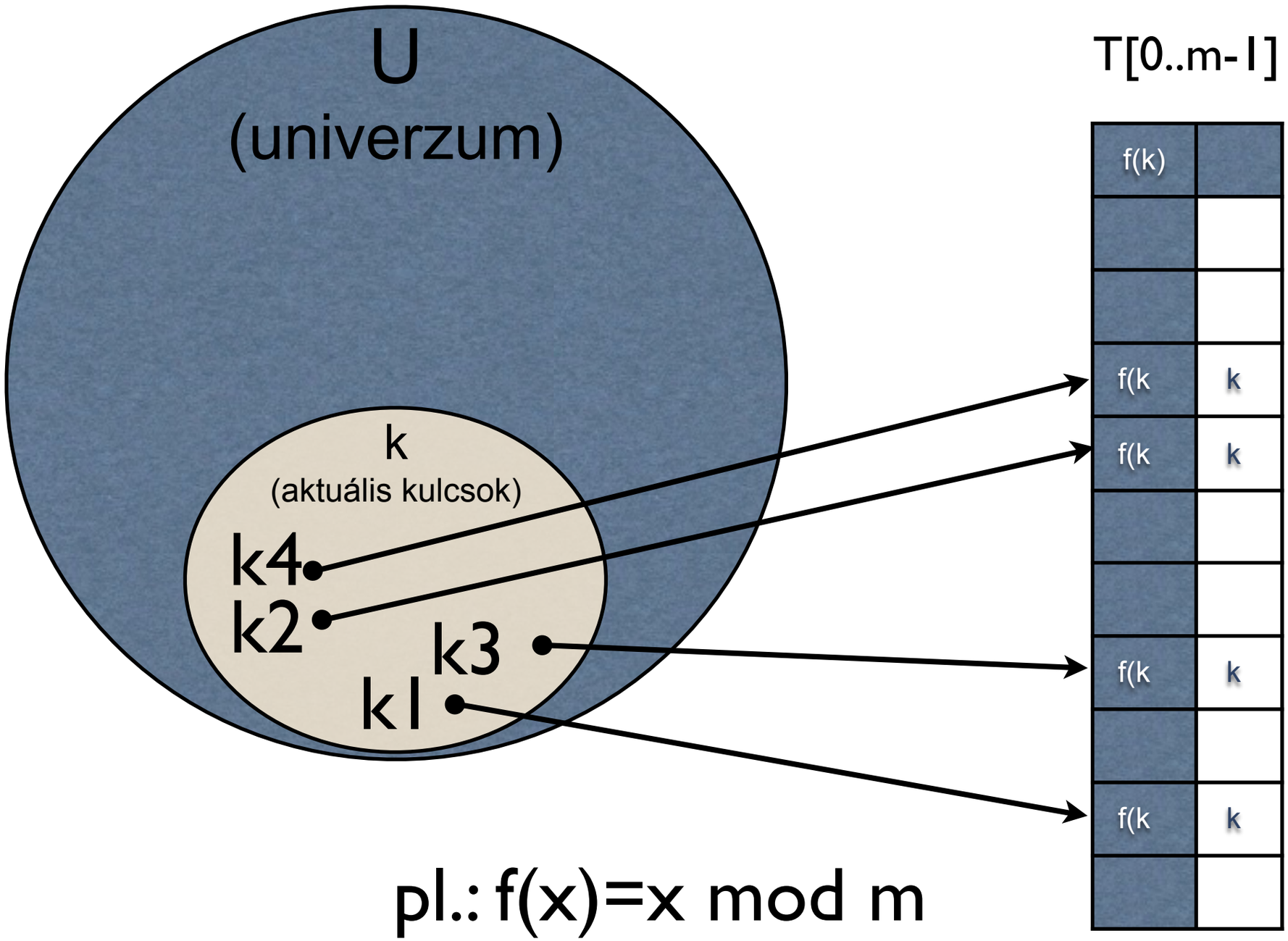
U={0,1,2,..} U={..,-1,0,1,..}

nem használhatunk végtelen tömböt!



Hasító táblázat

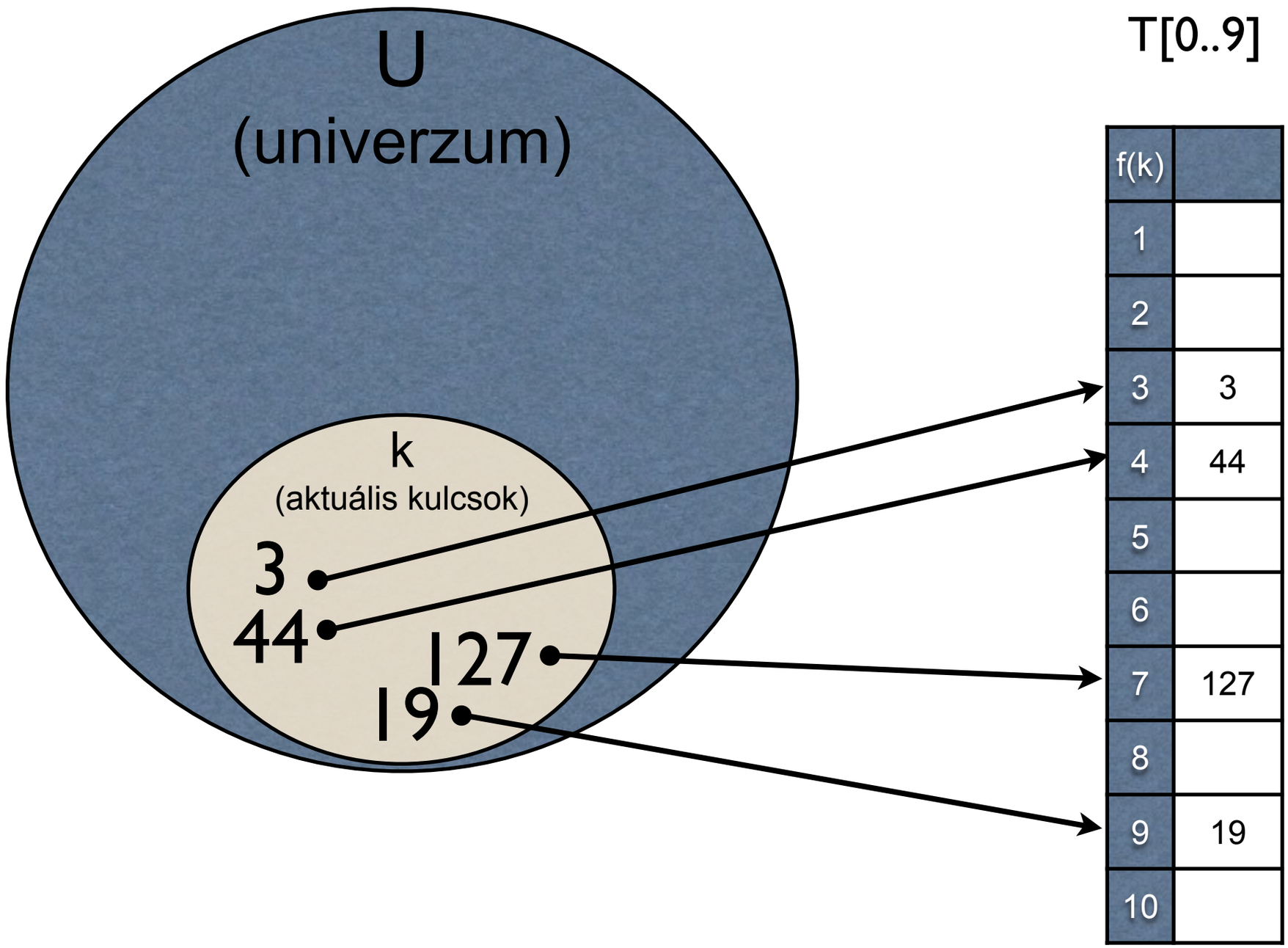
f(x) hasító függvény



pl.: f(x)=x mod m

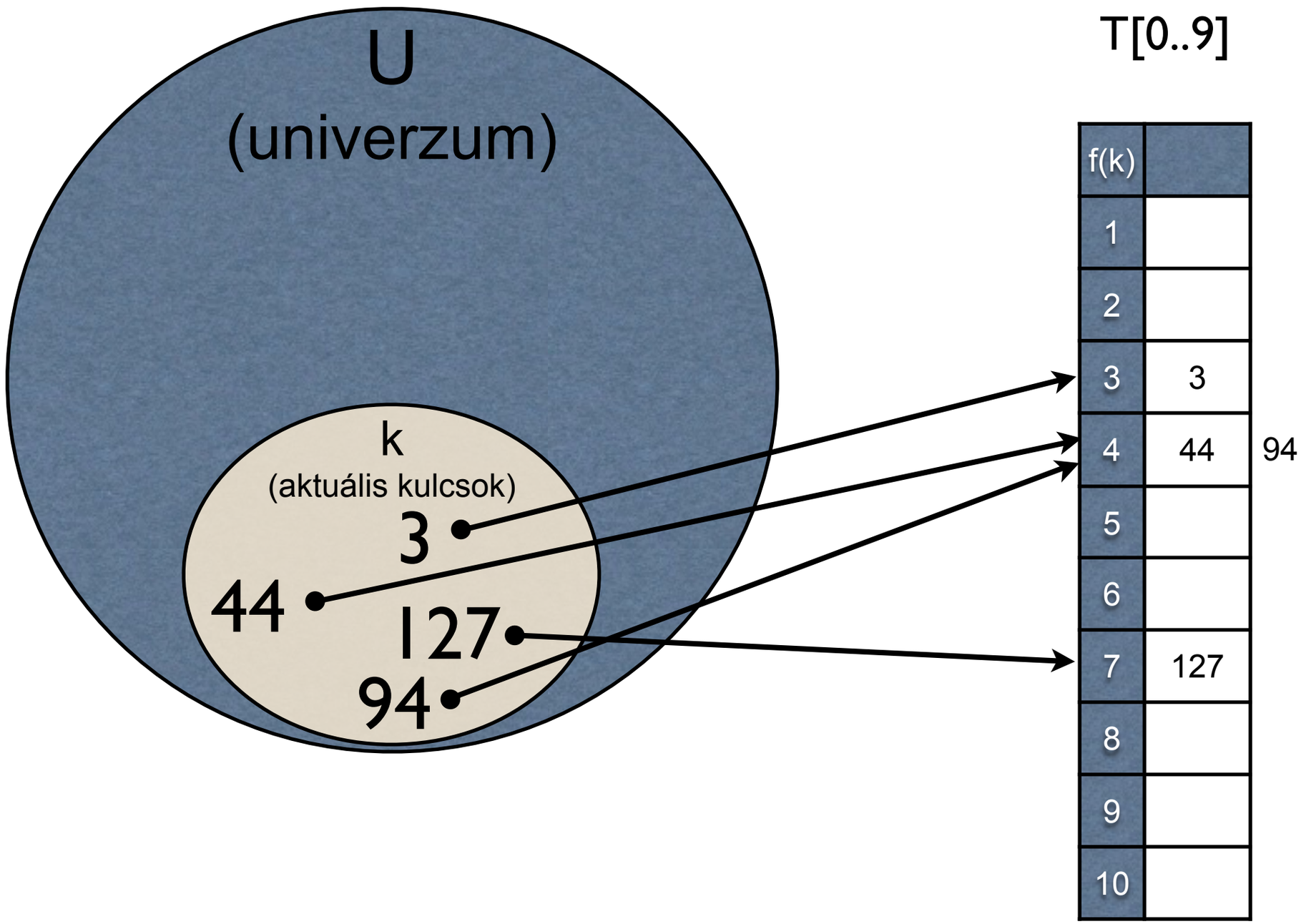
Hasító táblázat példa

pl.: f(x)=x mod 10

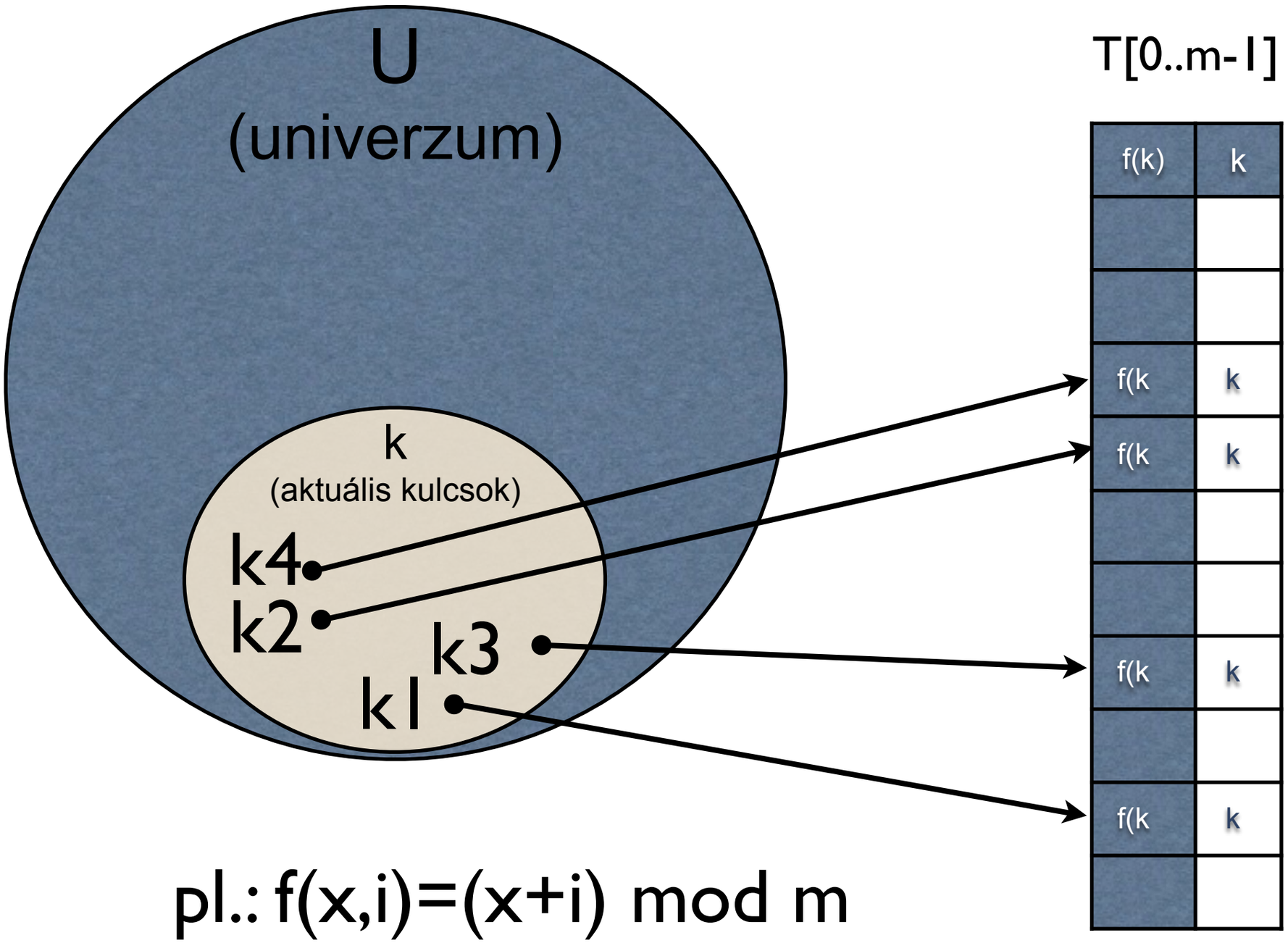


Hasító táblázat példa

pl.: f(x)=x mod 10



Ütközésfeloldás nyílt címzéssel



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

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

Ütközésfeloldás nyílt címzéssel

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

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

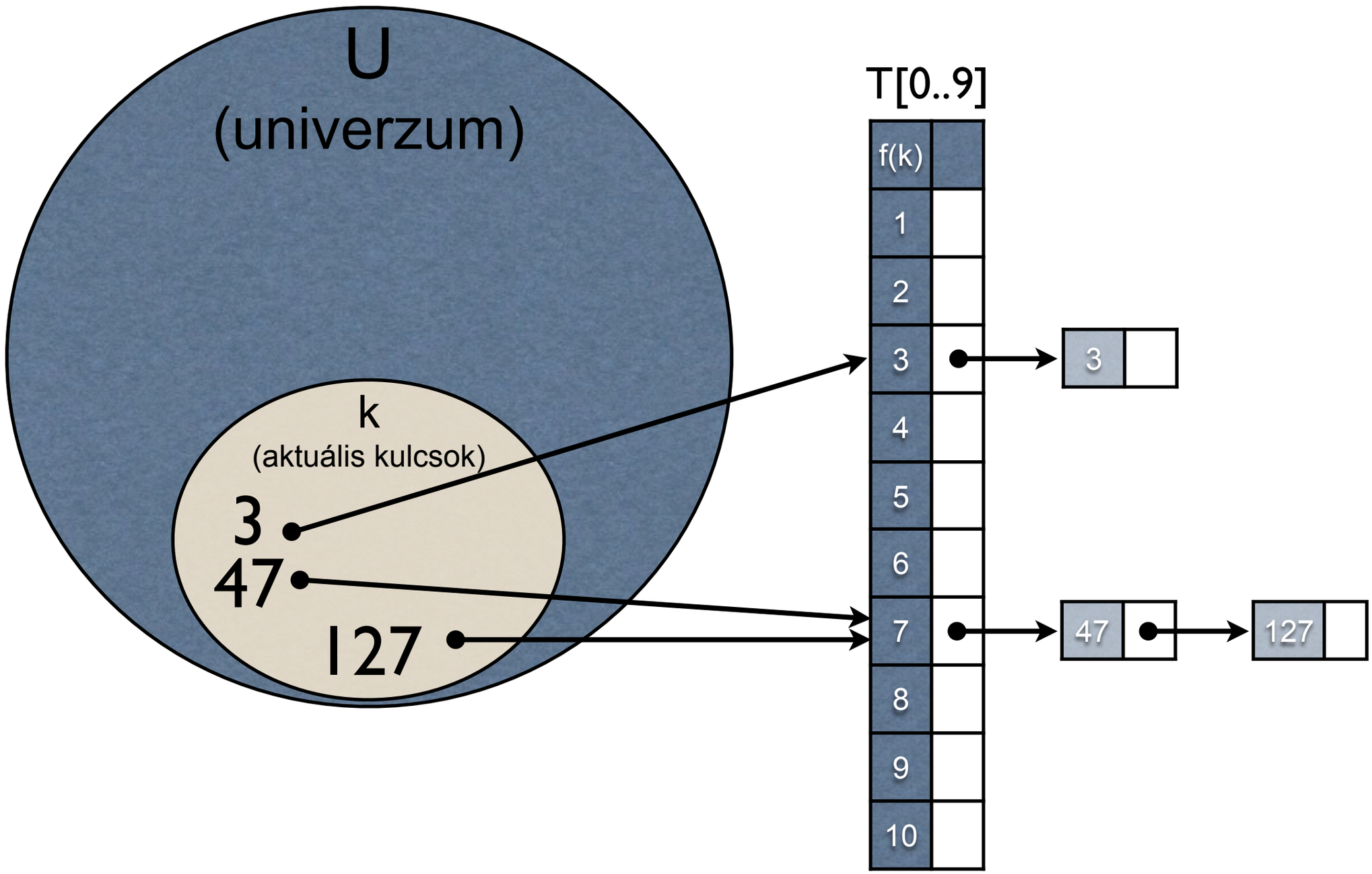
24

```
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?

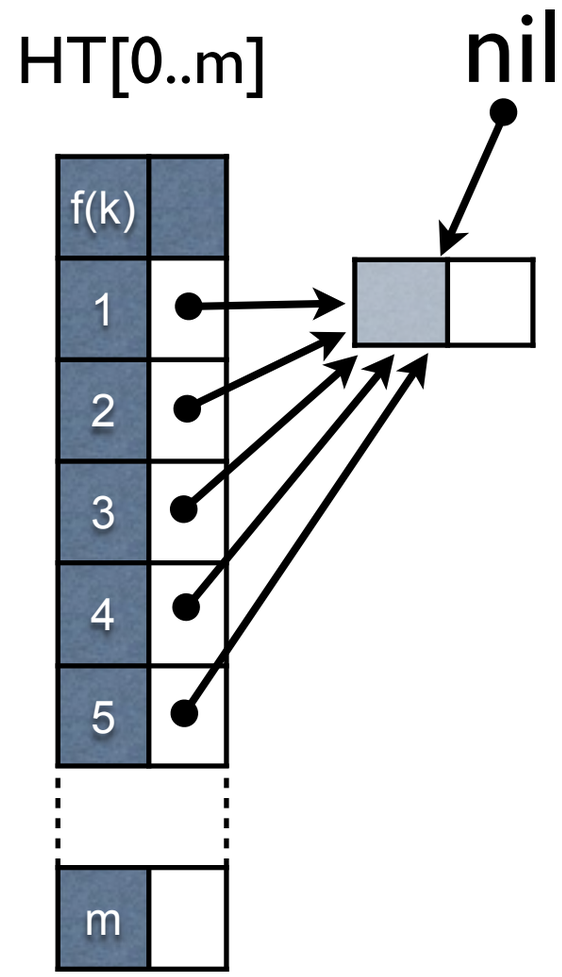
Ütközésfeloldás láncolással

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

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

Tétel: **Láncolásos ütközésfeloldás**nál, ha a hasítás **egyszerű egyenletes**, akkor a *sikertelen keresés* **átlagos** ideje: O(1+α).

Biz.: 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+α)

□

Tétel: **Láncolásos ütközésfeloldás**nál, ha a hasítás **egyszerű egyenletes**, akkor a *sikeres keresés* **átlagos** ideje: O(1+α).

Megjegyzés: 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.

□

Következtetések

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

Hogy lehet biztosítani, hogy az α konstans legyen?

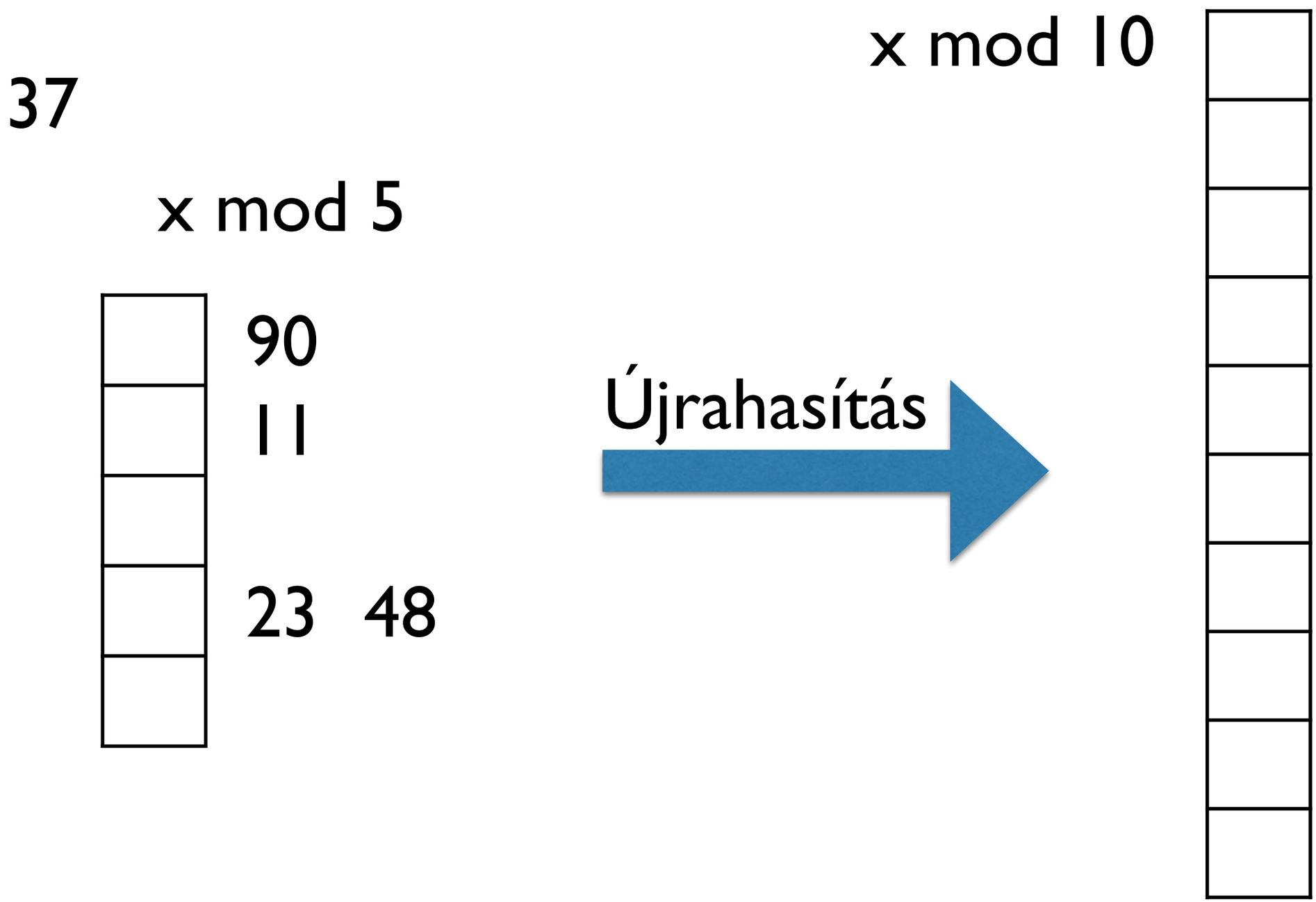
Ha m arányos n-el azaz tudjuk előre az adatszerkezetbe kerülő adatok maximális számát (vagy legalább annak nagyságrendjét)

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

Dinamikus táblák - tábla kiterjesztése



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

5 5 10 10

O(1)

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

n Beszur teljes költsége

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

Minden kiterjesztés után közvetlenül T.eszam = T.meret/2+1, vagyis Φ(T) = 2, közvetlenül előtte pedig T.eszam = T.meret, így Φ(T) = T.eszam. Φ(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

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

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

[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest: Algoritmusok
Műszaki Könyvkiadó, 2003.

[2] Imreh Csanád: Algoritmusok és Adatszerkezetek II. 2009
http://www.inf.u-szeged.hu/~cimreh/n10alg26el.pdf

[3] Horváth Gyula: Algoritmusok és Adatszerkezetek II. 2008

http://www.inf.u-szeged.hu/~tnemeth/a2anyagok/v10.pdf

http://www.inf.u-szeged.hu/~tnemeth/a2anyagok/v13.pdf