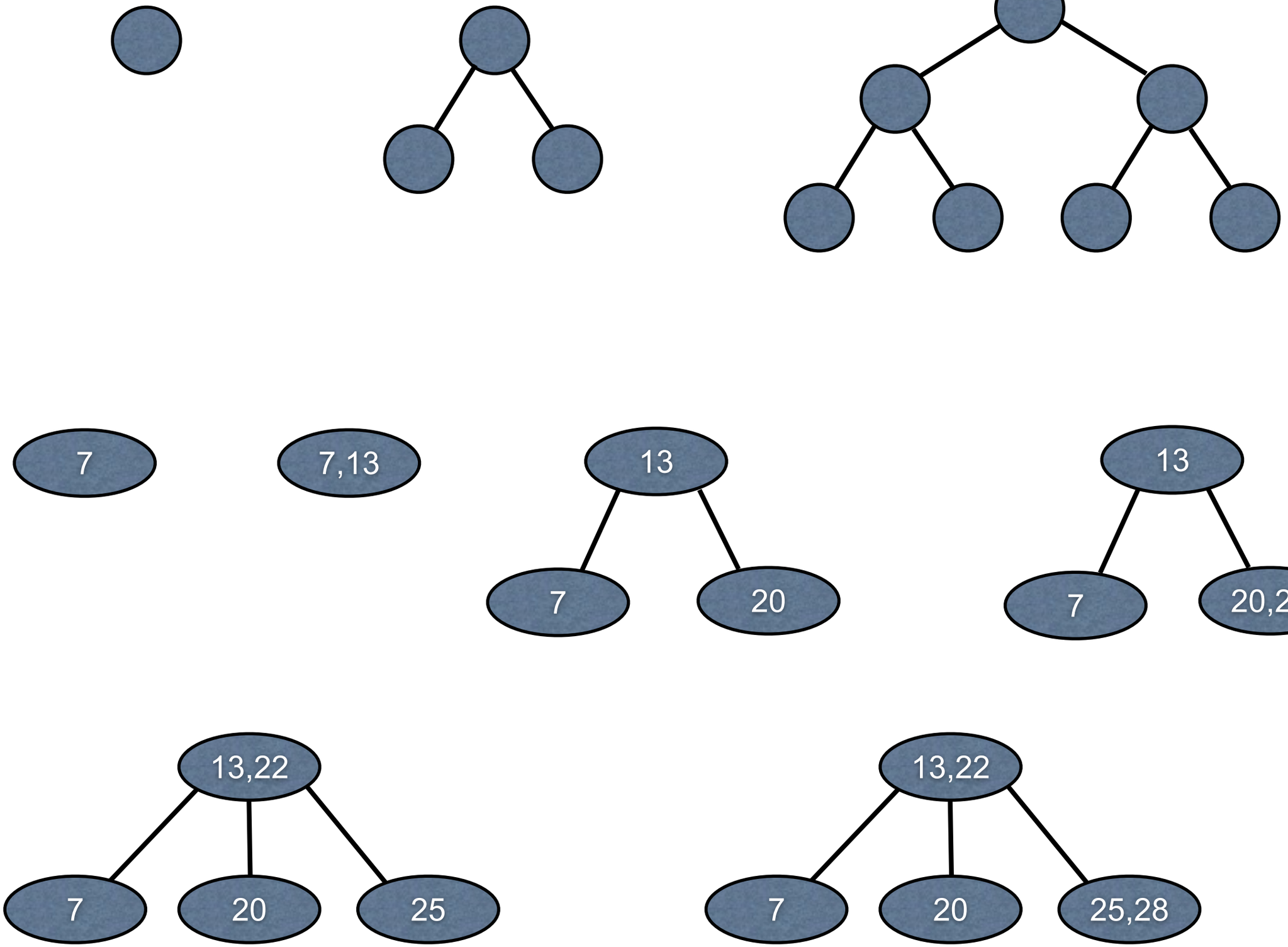
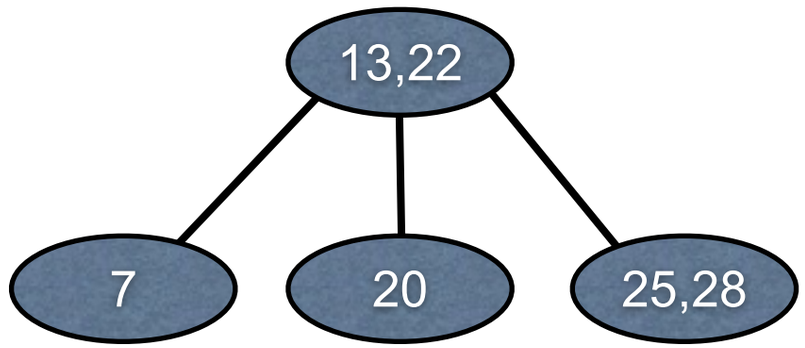
Általános keresőfák


7
7,13
13
7
20
13
7
20,22
13,22
7
20
25
13,22
7
20
25,28

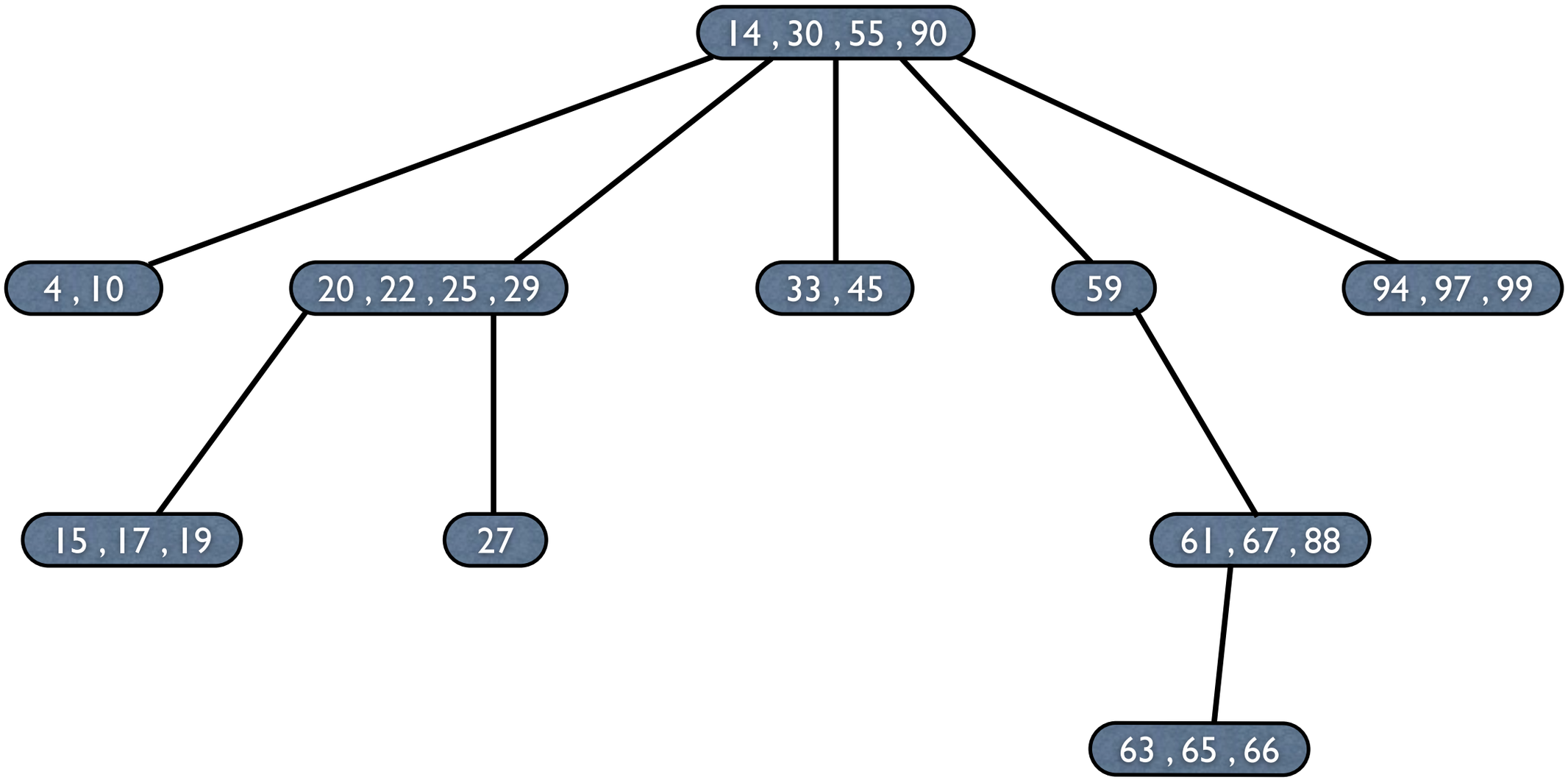
Általános keresőfa

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.

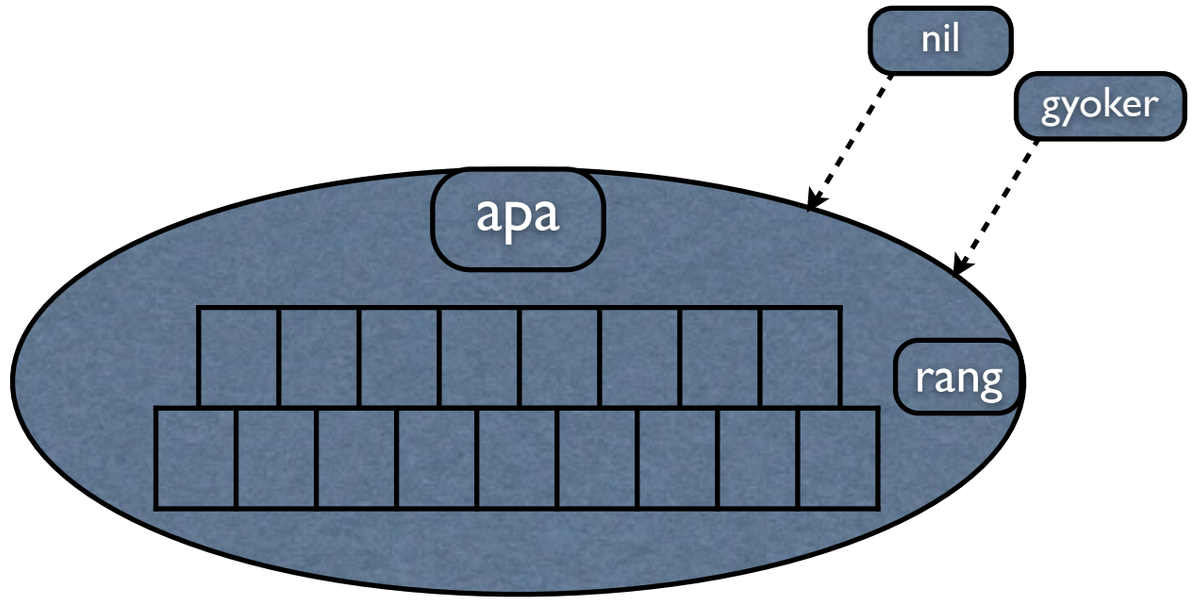


Példa

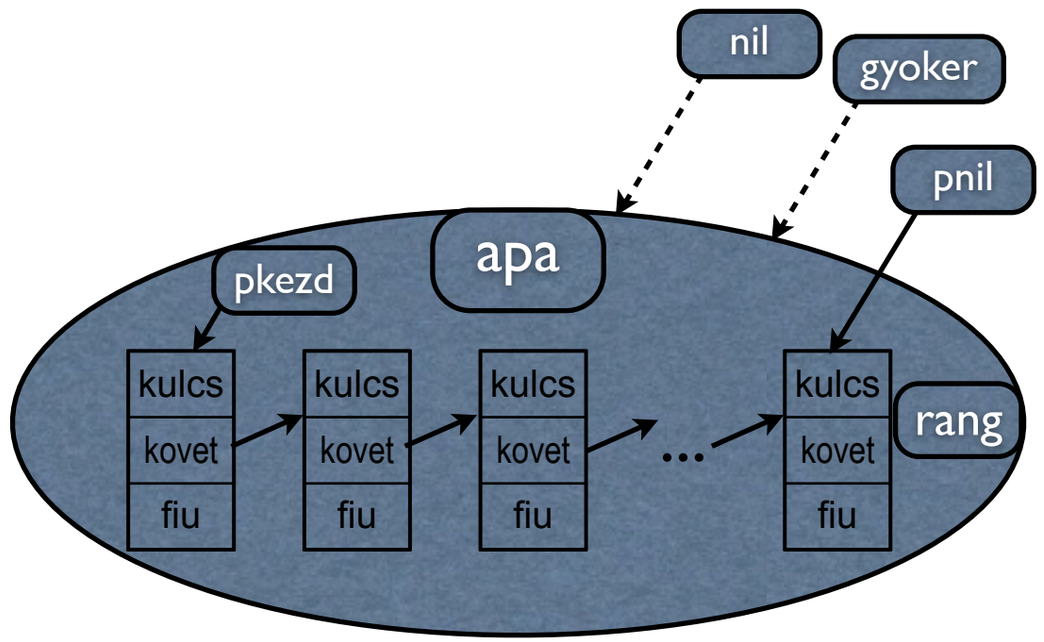

14 , 30 , 55 , 90
4 , 10
20 , 22 , 25 , 29
33 , 45
59
94 , 97 , 99
15 , 17 , 19
27
61 , 67 , 88
63 , 65 , 66

Fapont tárolása

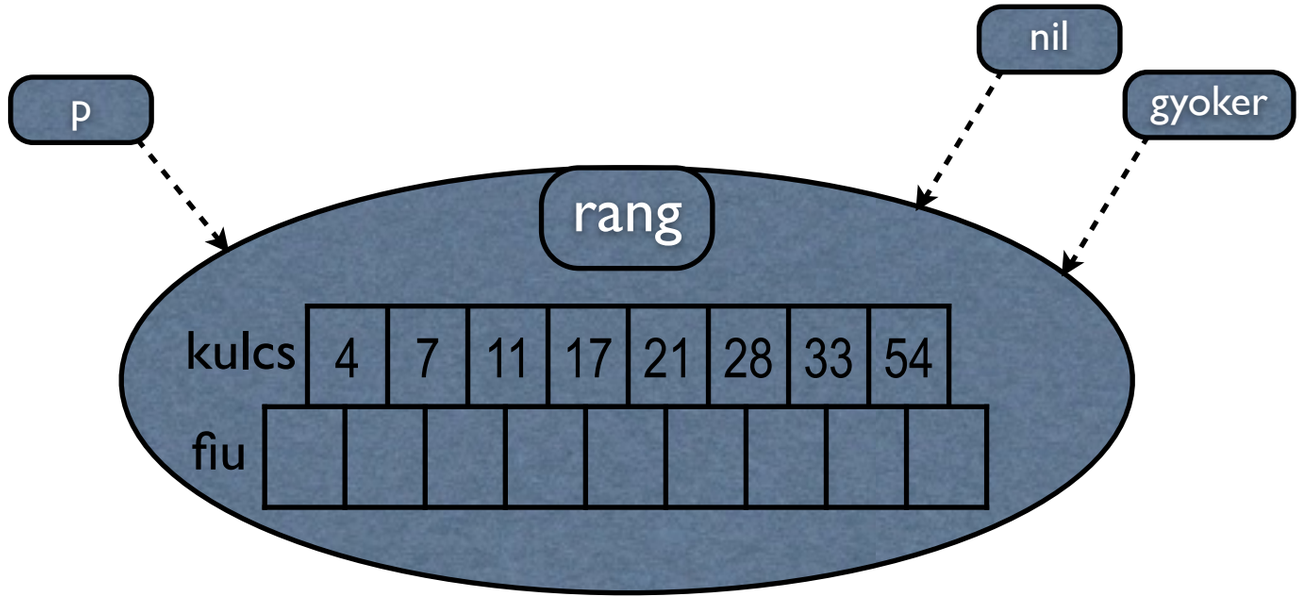
```
fapont class<T> {
   int rang ;
   fapont apa ;
   <T>[] kulcs;
   fapont[] fiu ;
}
```




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



Keresés általános keresőfában

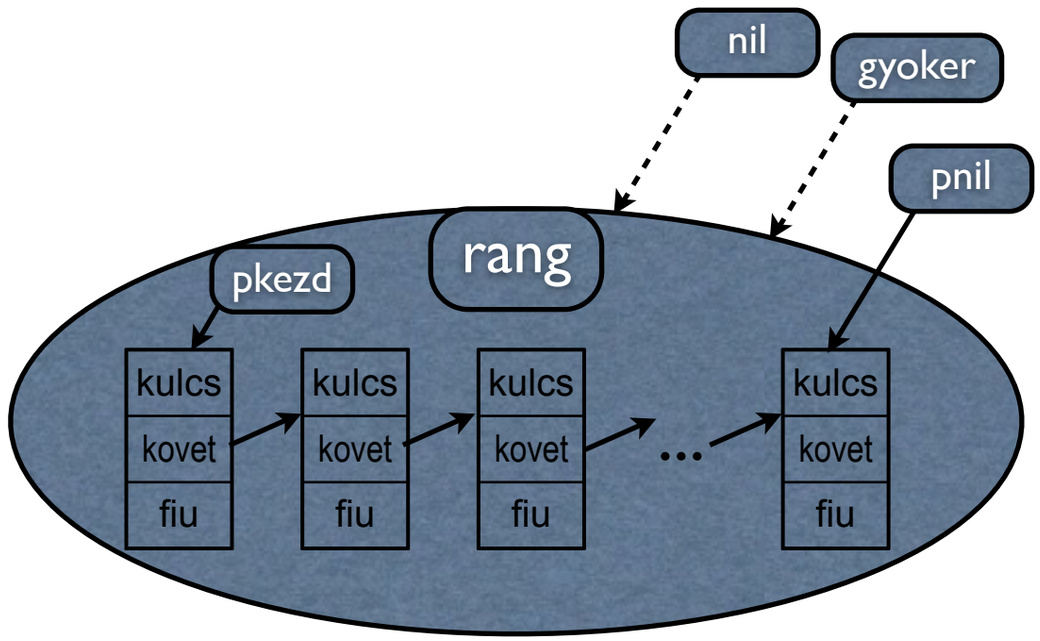



```
keres<T>(<T> x,fapont p) {
   if (p==nil) return nil ;
   int i=1 ;
   while (i<=p.rang && x>=p.kulcs[i]) i++ ;
   if (x==p.kulcs[i]) return (p,i) ;
   return(keres(x, p.fiu[i])) ;
}
```

Keresés általános keresőfában

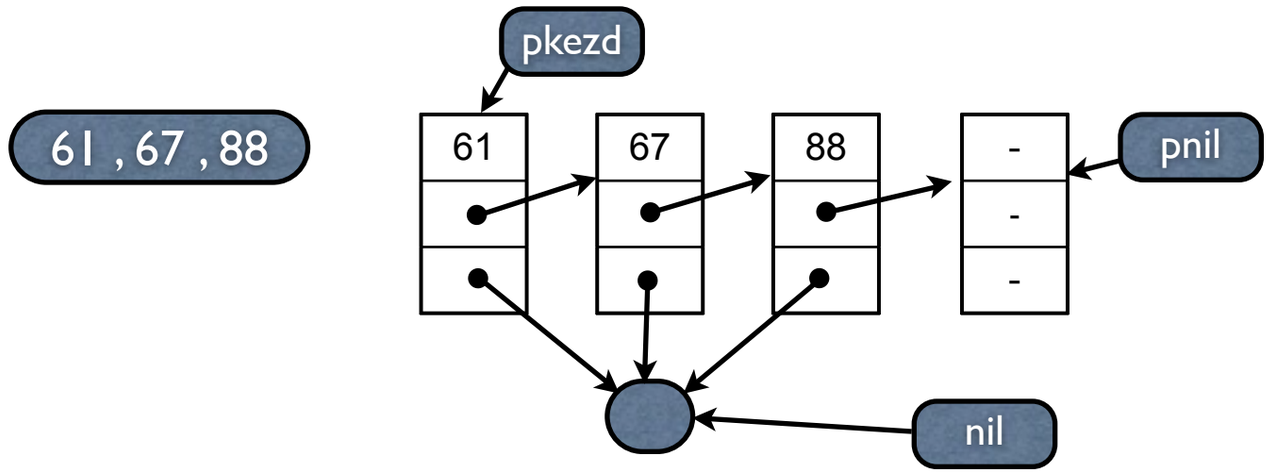
```
class par {
   fapont p;
   pontelem e;
}
par(q,i) {
   p=q ;
   e=i ;
}

par 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 adatszerkezet

még mindig fázunk...

B-fa

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

- Általános keresőfa.
- A gyökér kivételével minden p ∈ F pontra t ≤ Rang(p) ≤ 2t,

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

  azaz minden levél pont mélysége azonos.

B fa példa

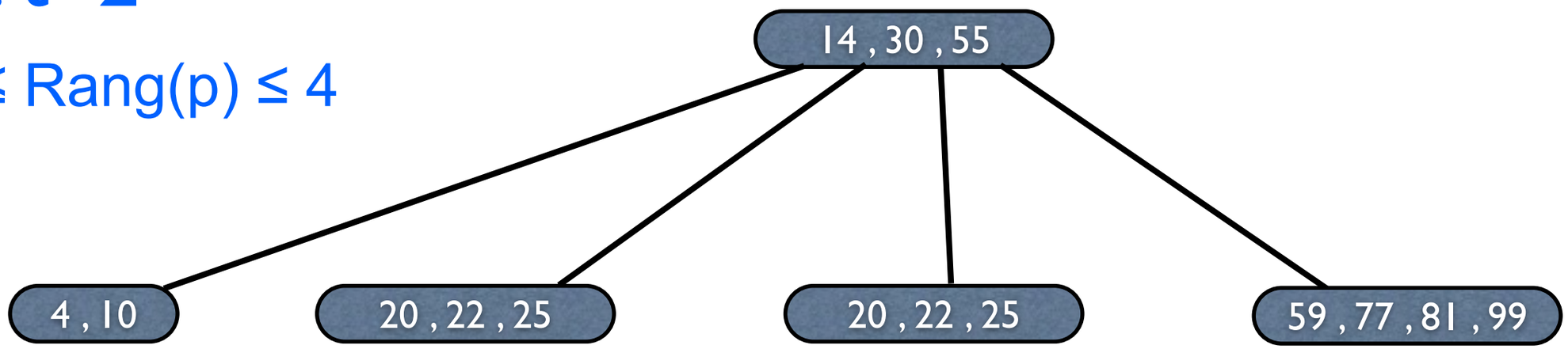
A gyökér kivételével minden $p \in F$ pontra $t \leq Rang(p) \leq 2t$,

a gyökér rangja legfeljebb 2t.

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

pl. t=2

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



B-fák

Egy p pontot telítettnek nevezünk, ha Rang(p) = 2t,

illetve minimálisnak, ha Rang(p) = t.

Továbbá feltesszük, hogy minden ponthoz tartozik egy Levél

logikai változó, ami akkor igaz, ha a pont levél.

B-fák felhasználása: B fákat olyan adatszerkezetek esetén használják, amelyek nem férnek el a memóriában.

Az aktuális pont van a memóriában, és amikor másik lapra érünk azt a háttértárból olvassuk be. Következésképpen a műveletek hatékonysága elsősorban a pontokhoz történő hozzáférések számától függ.

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)	2t(t+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).

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

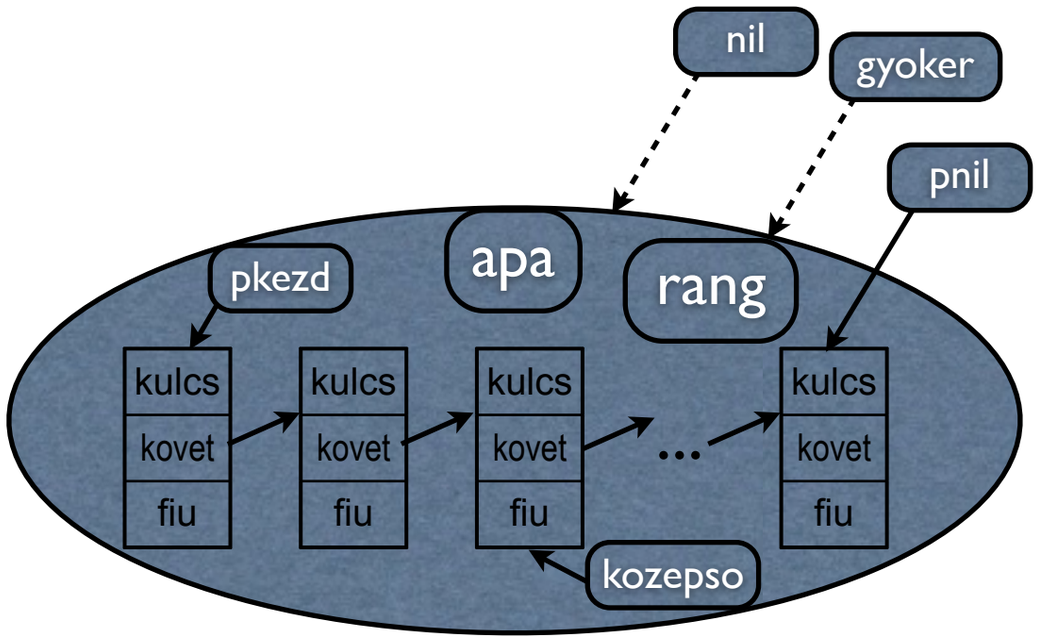
Pont szétvágása

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:

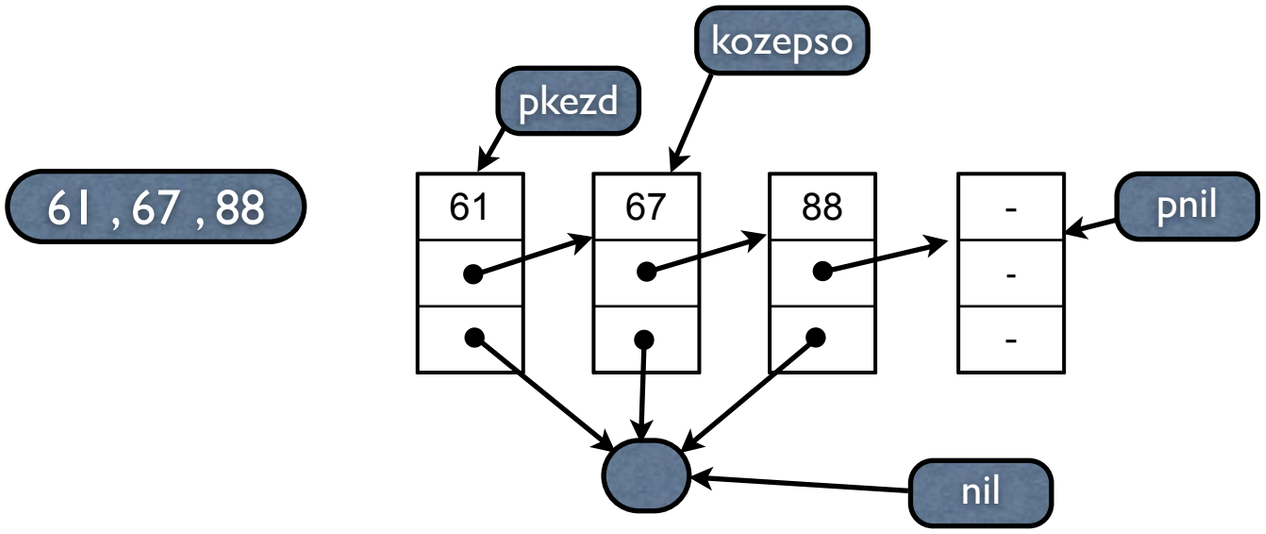
Az algoritmus az y pontot, *amelyre rang(y) = 2(t−1) és amely az p pont i-edik fia* vágja ketté a közepén. A középső adat felmegy az p-be.

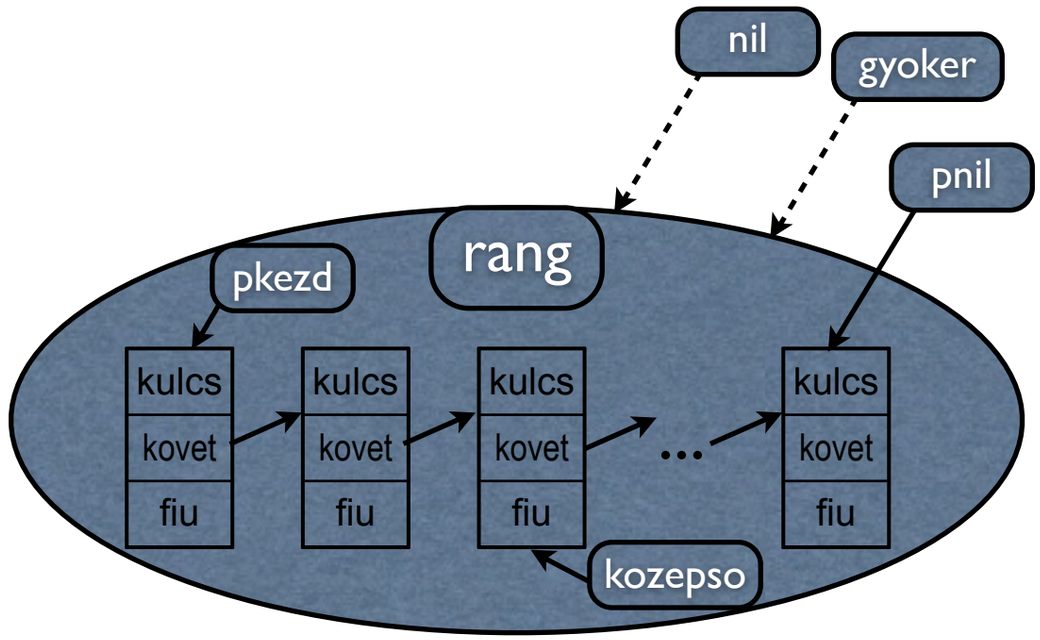
B-fa pont tárolása

```
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 ;
   }
   ...
}
```

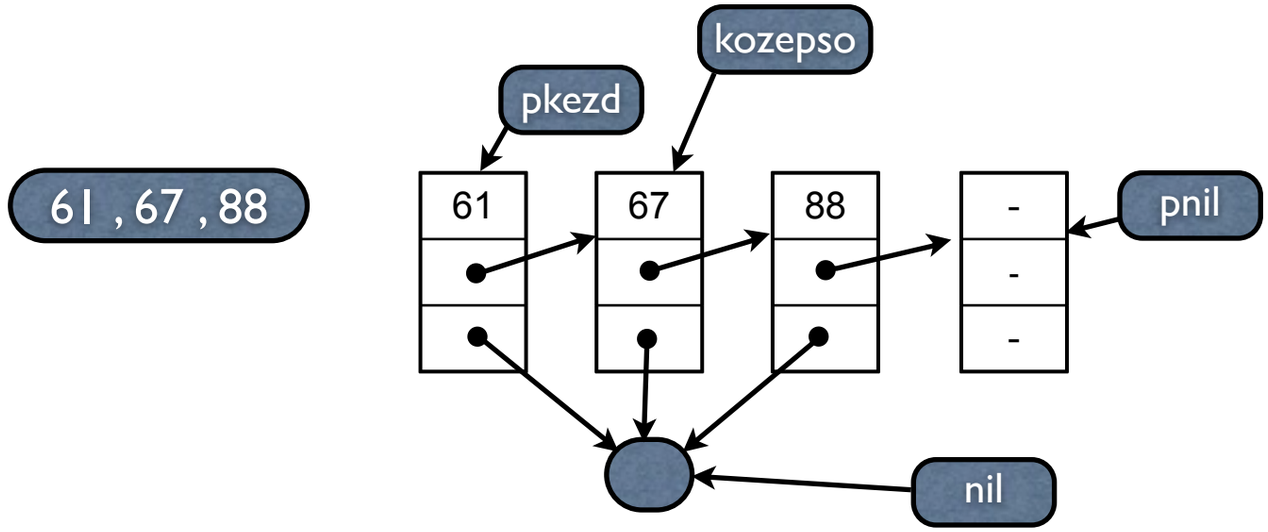


B-fa beszúr



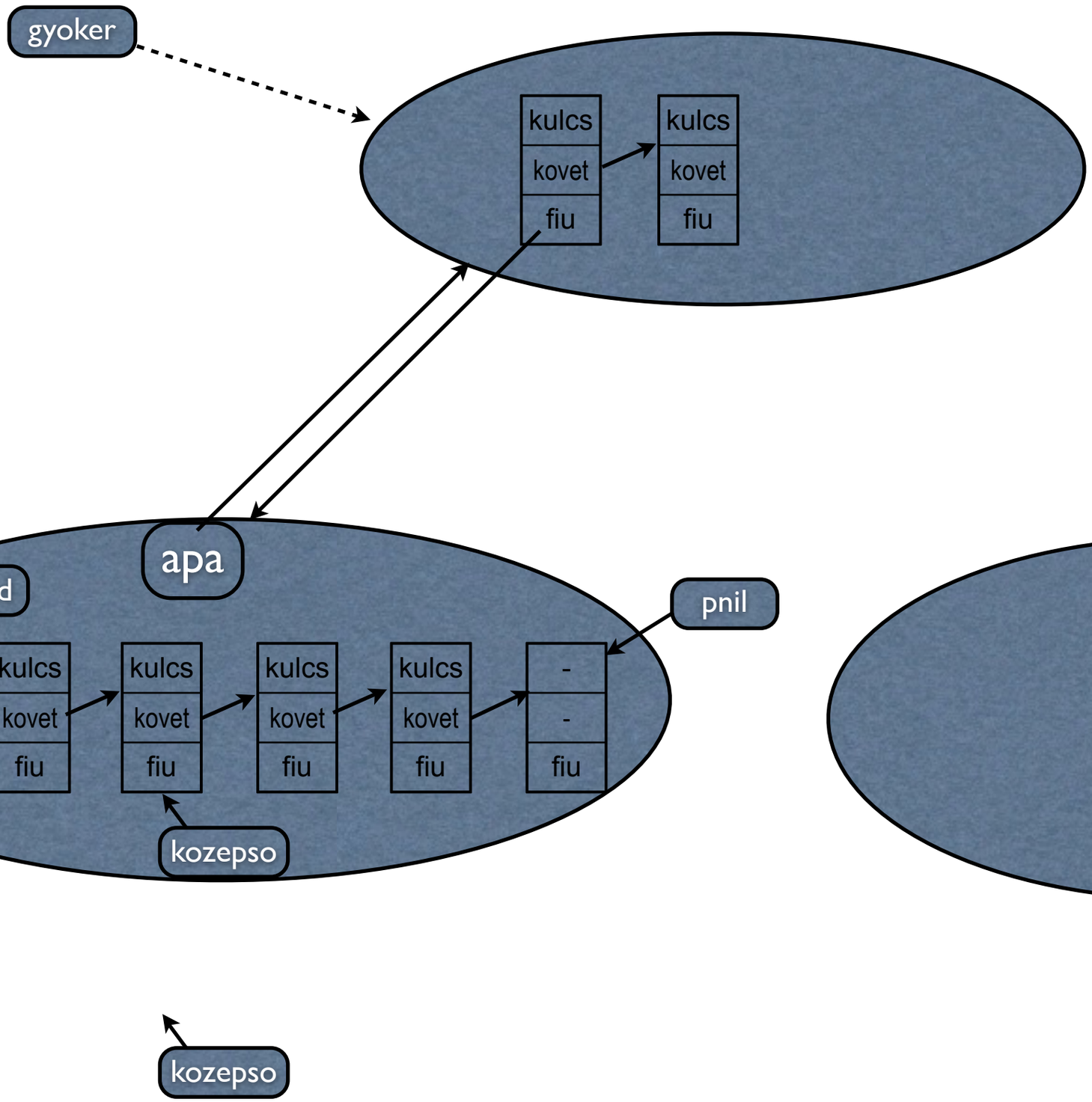


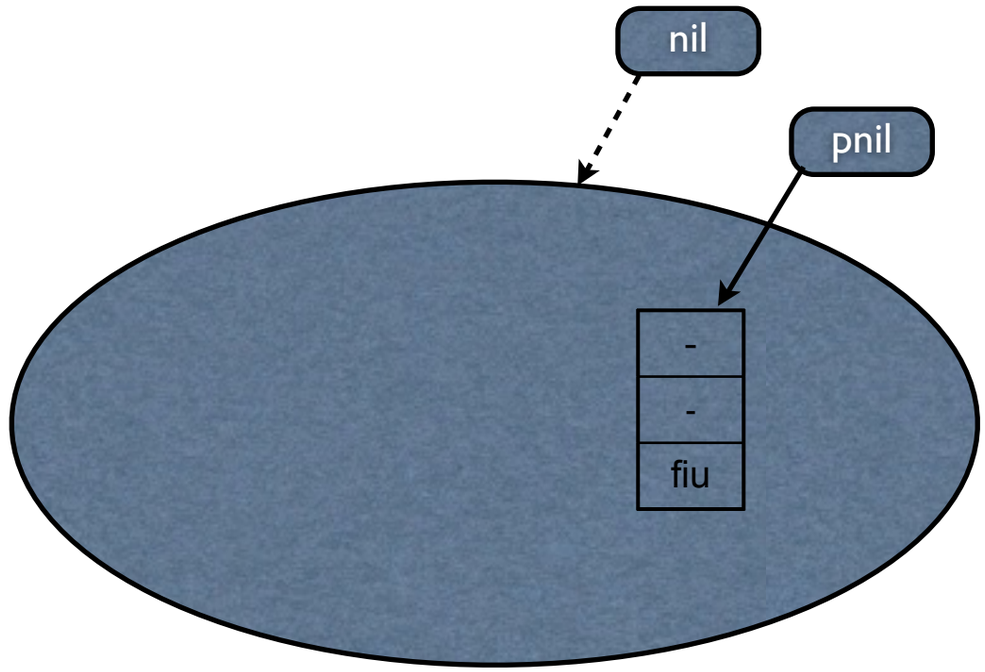

```
int beszur<T>(<T> x) {
   int poz;
   fapont q,p=gyoker;
   pontelem i;
   do {
      i=p.pkezd; poz=0;
      while (i!=pnil && x>i.kulcs) { i=i.kovet; poz++; }
      if (i!=pnil && x==i.kulcs) return -1;
      p=i.fiu ;
   } while (i.fiu!=nil);
   listabaszur(p.apa,i,poz,x) ;
   return 0 ;
}
```



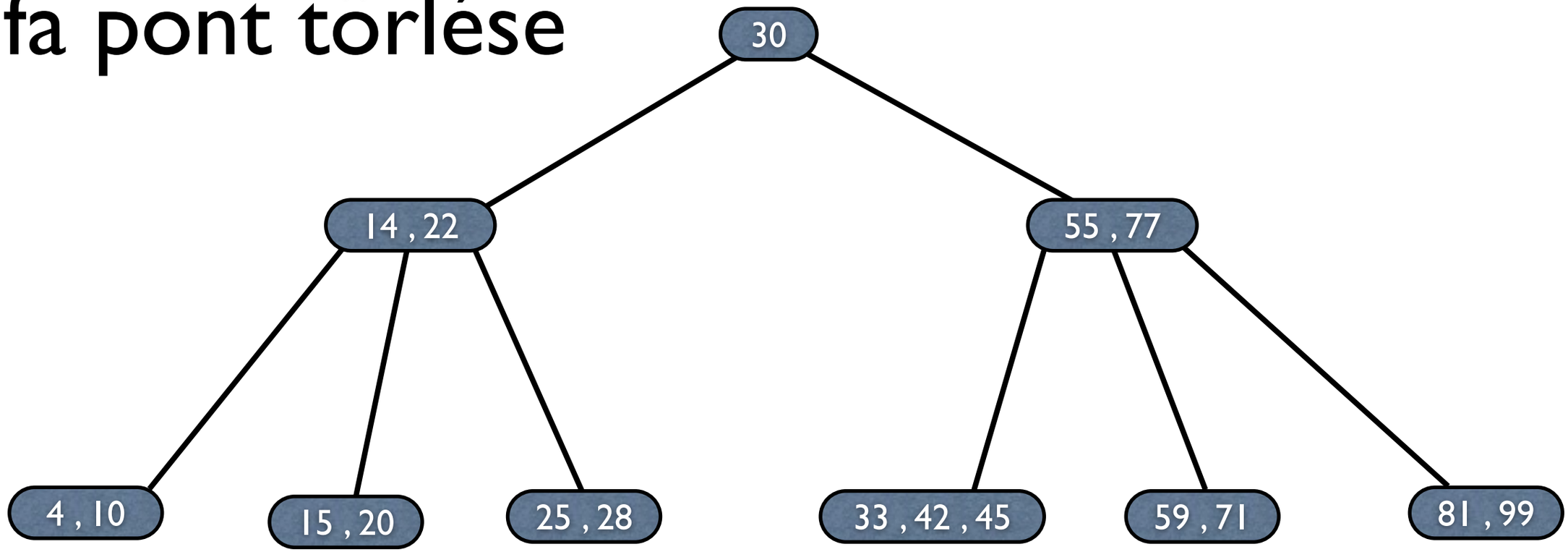

```
listabaszur<T>(fapont p, pontelem i, int poz, <T>x) {
   pontelem e=new pontelem();
   e.kulcs=x ;
   e.kovet=i.kovet ;
   i.kovet=e ;
   p.rang++ ;
   if (p.rang % 2) {
      if (poz>maxrang/2) kozepso=kozepso.kovet ;
      else //előrréb állítani?
   }
   if (p.rang>maxrang) szetszed(p) ;
}
```

Szétszed

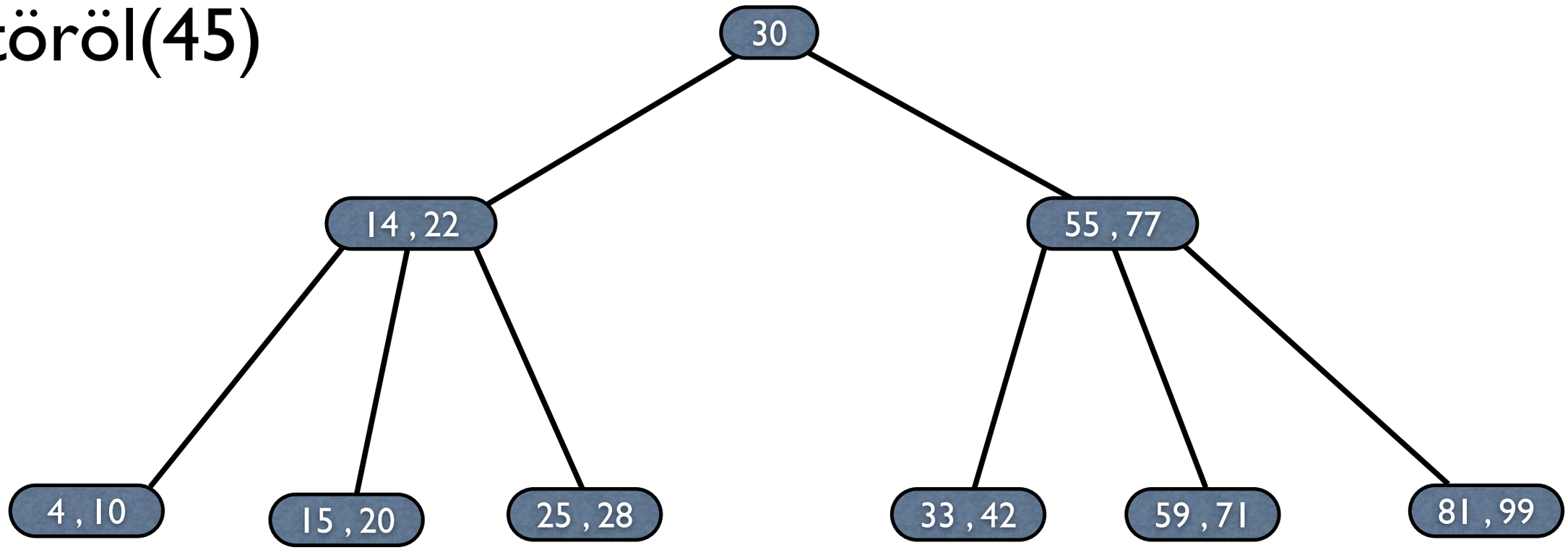

gyoker
kulcs
kovet
fiu
kulcs
kovet
fiu
apa
pkezd
pnil
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
kulcs
kovet
fiu
-
-
fiu
kozepso
kozepso


nil
pnil
-
-
fiu

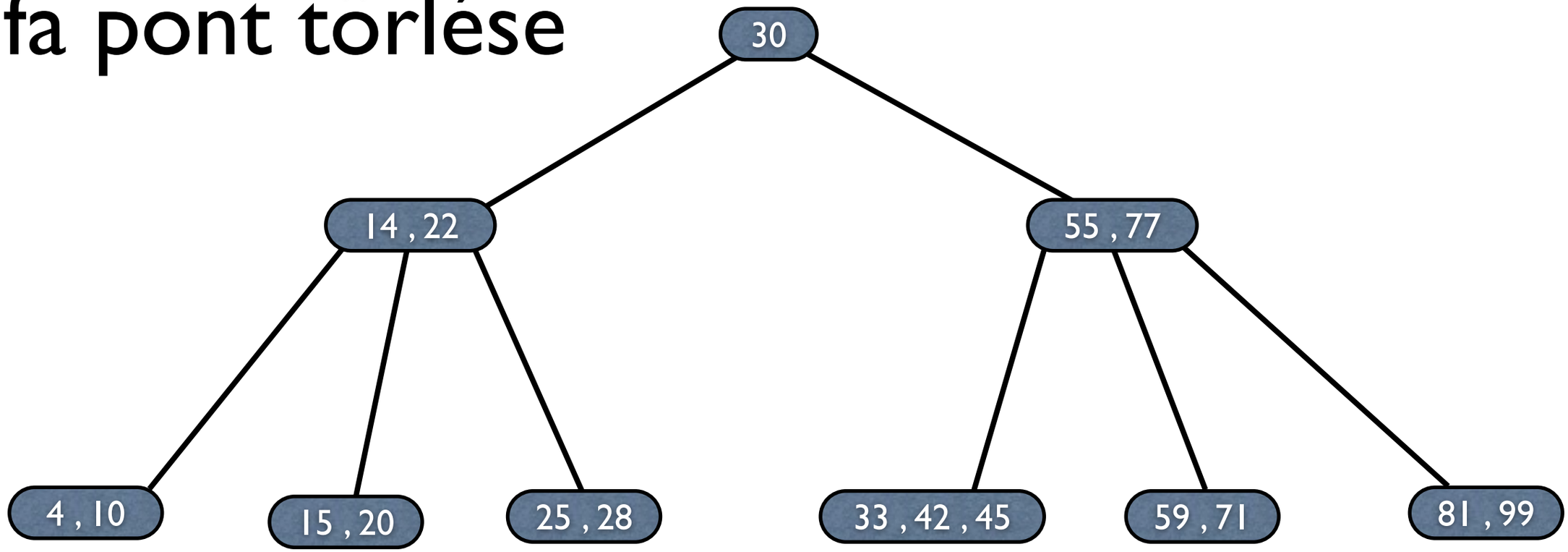
B-fa pont törlése



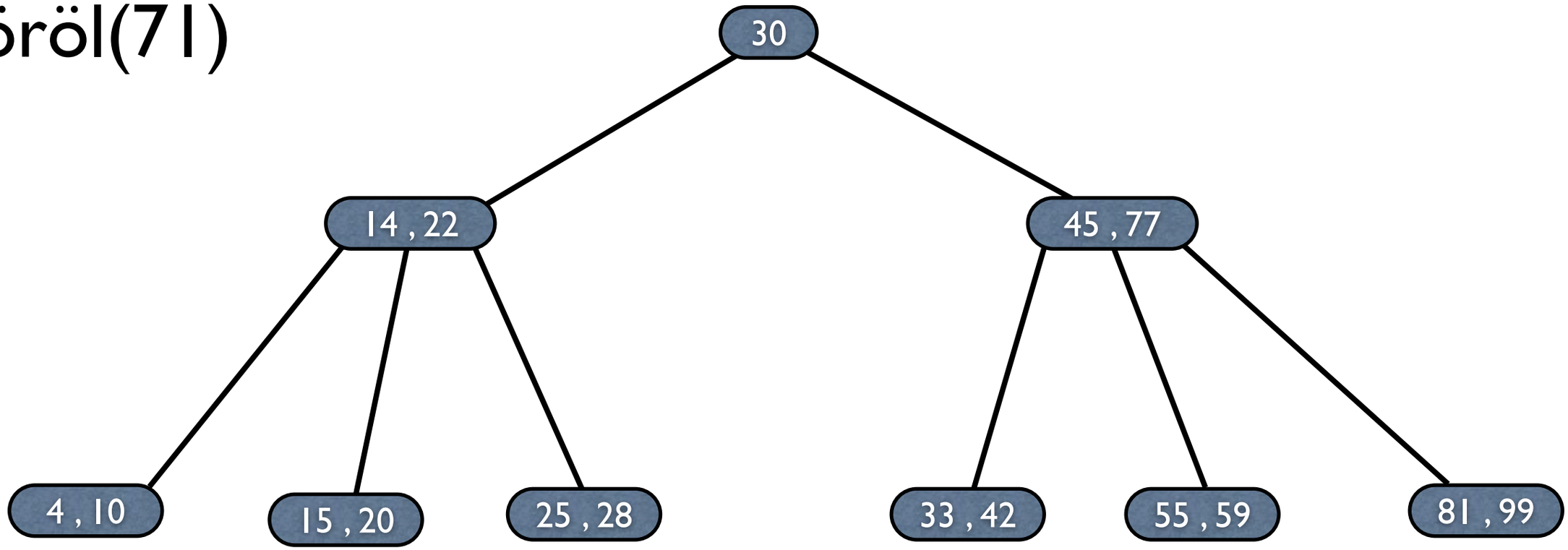
töröl(45)



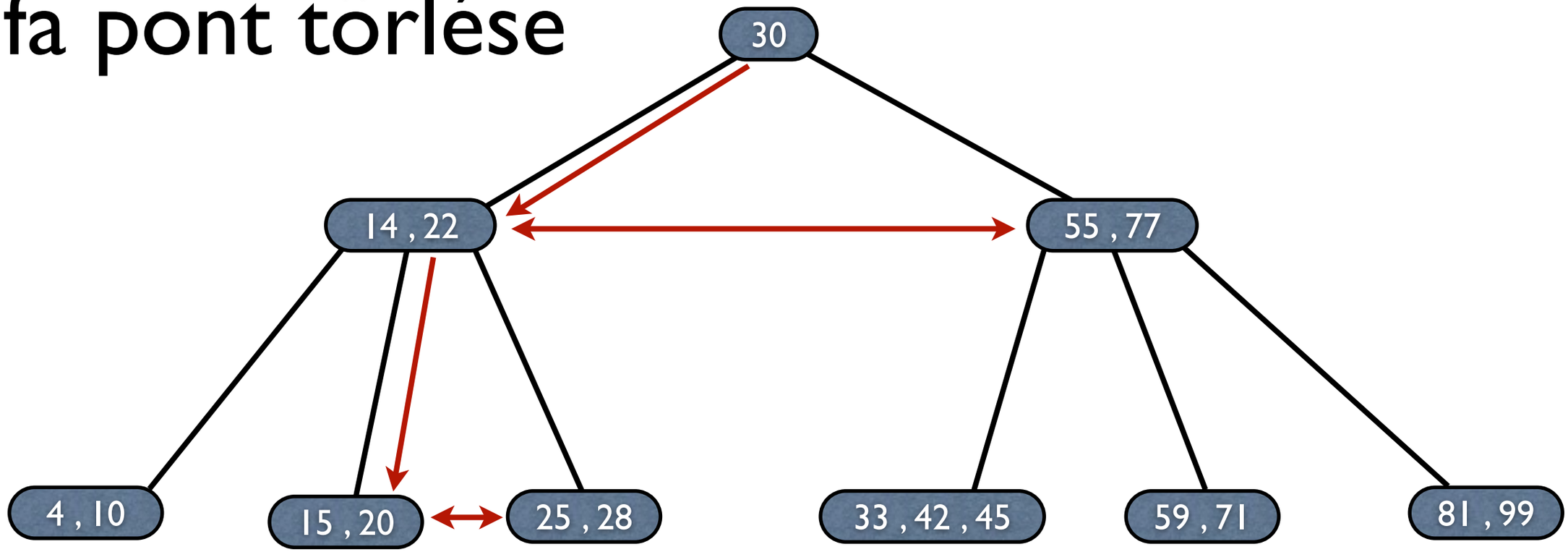
B-fa pont törlése



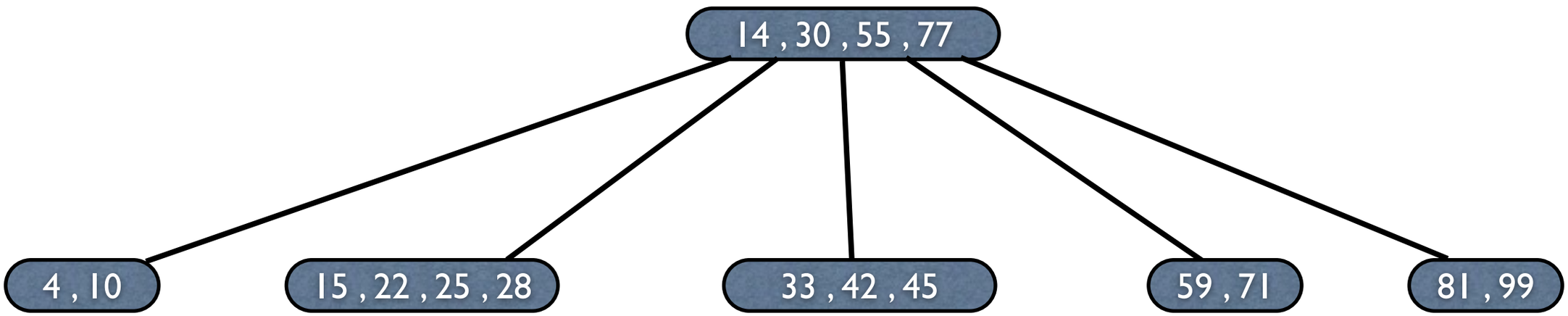
töröl(71)



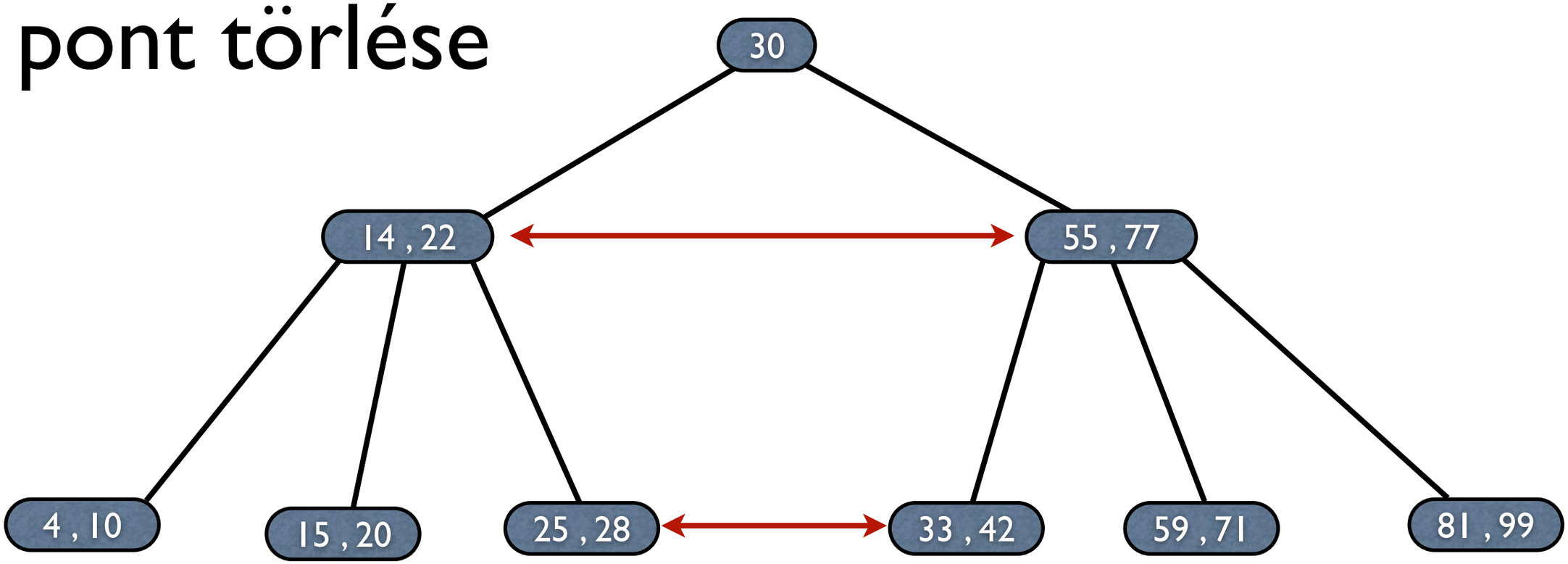
B-fa pont törlése



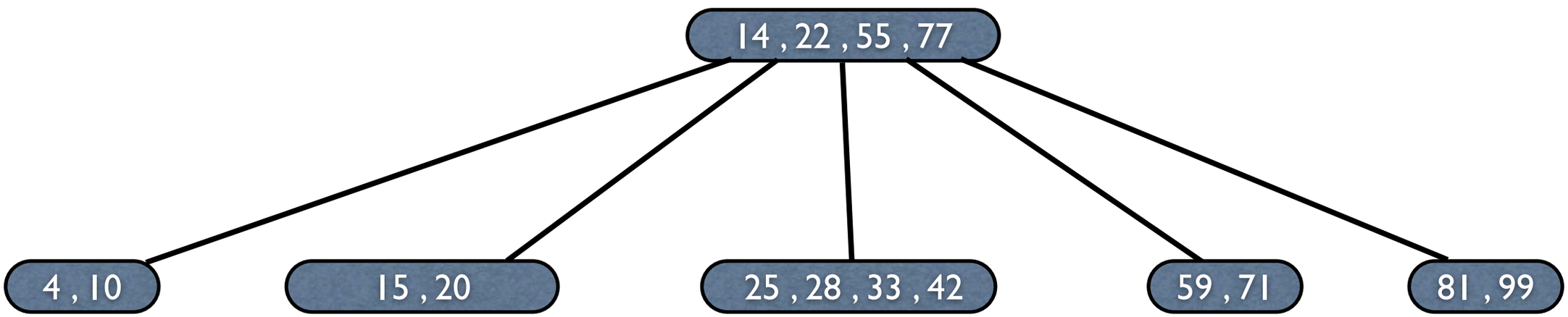
töröl(20)



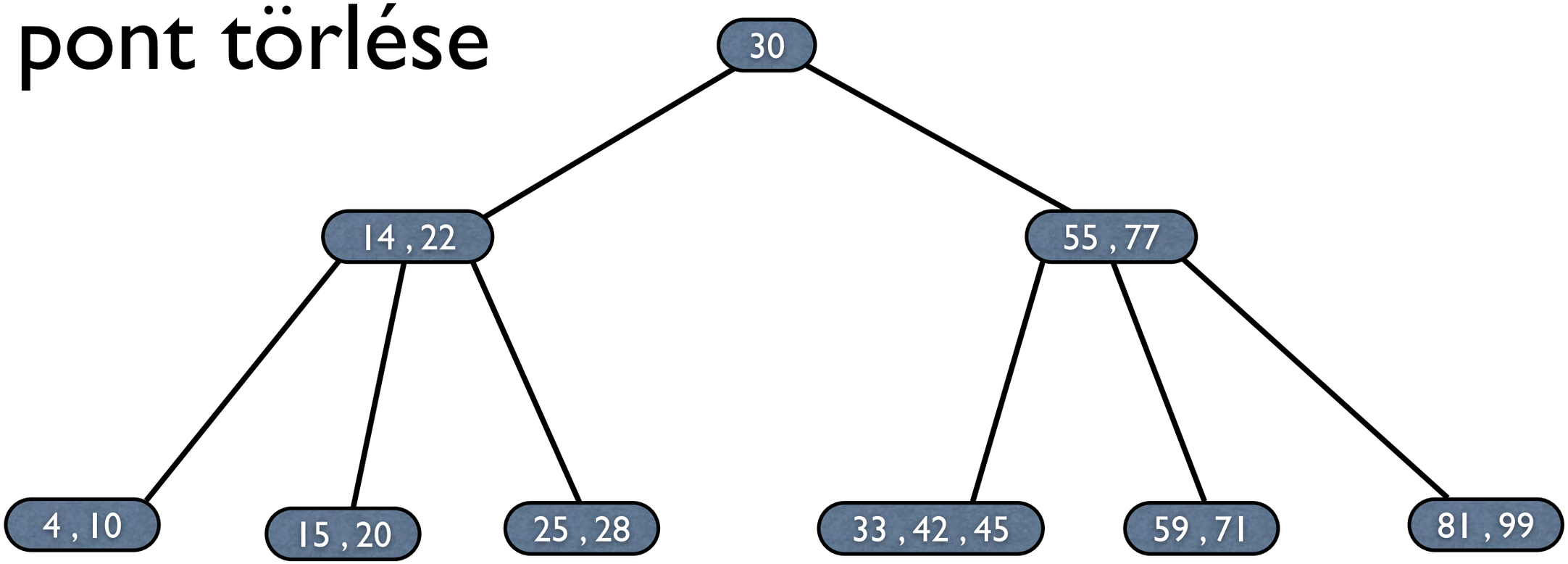
B-fa pont törlése



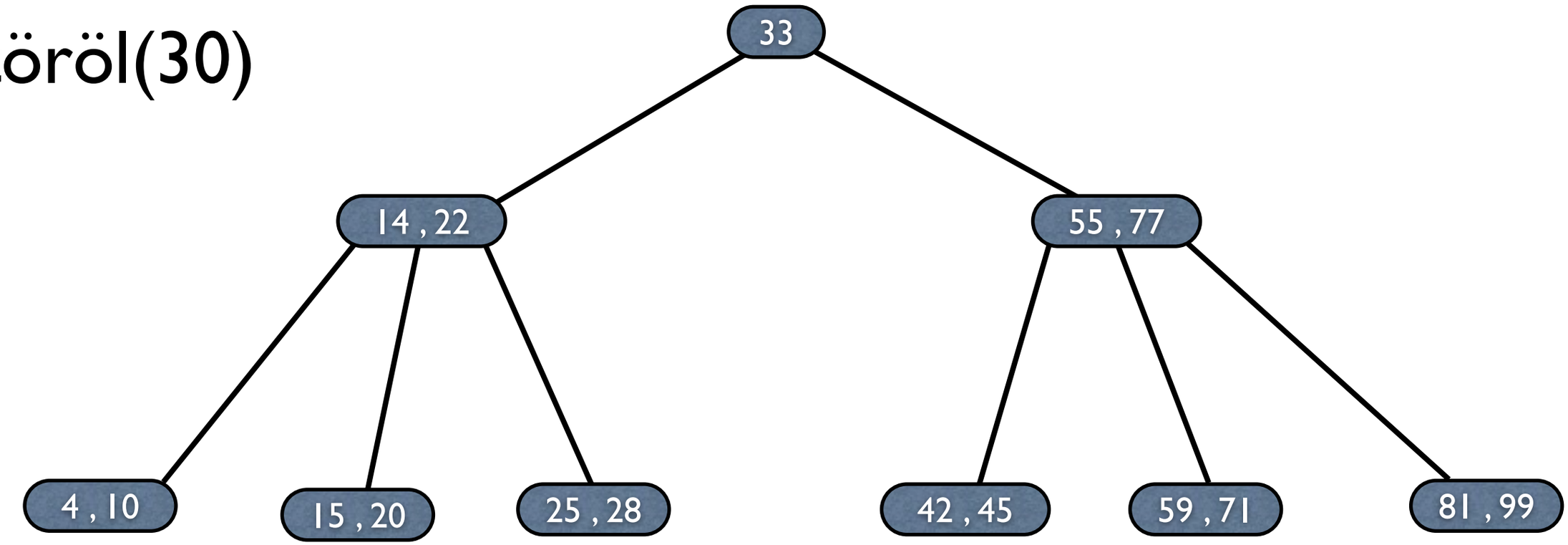
töröl(30)



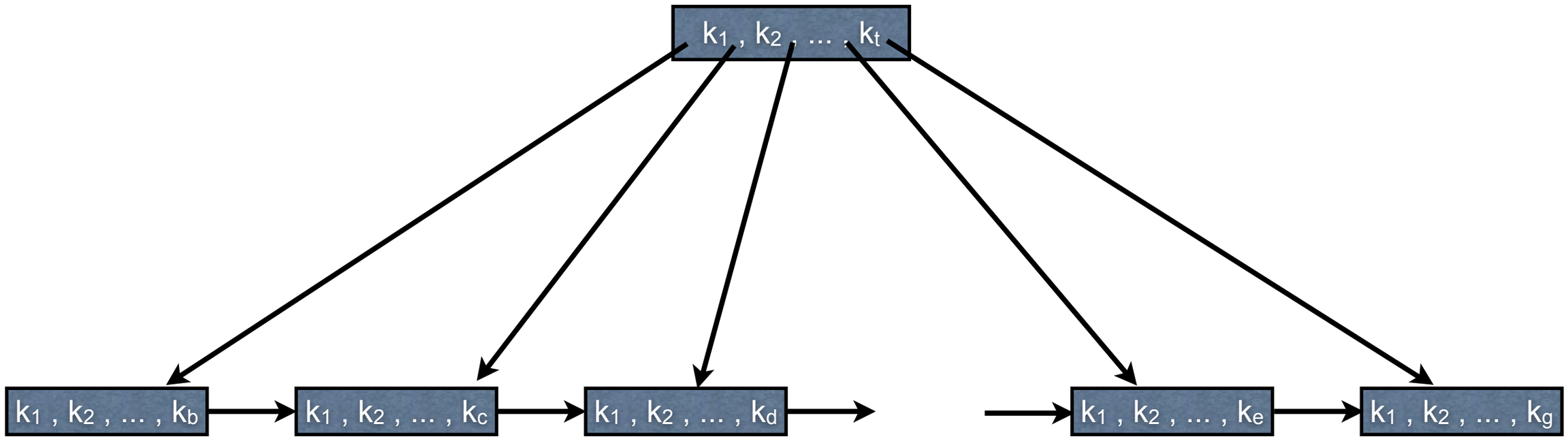
B-fa pont törlése



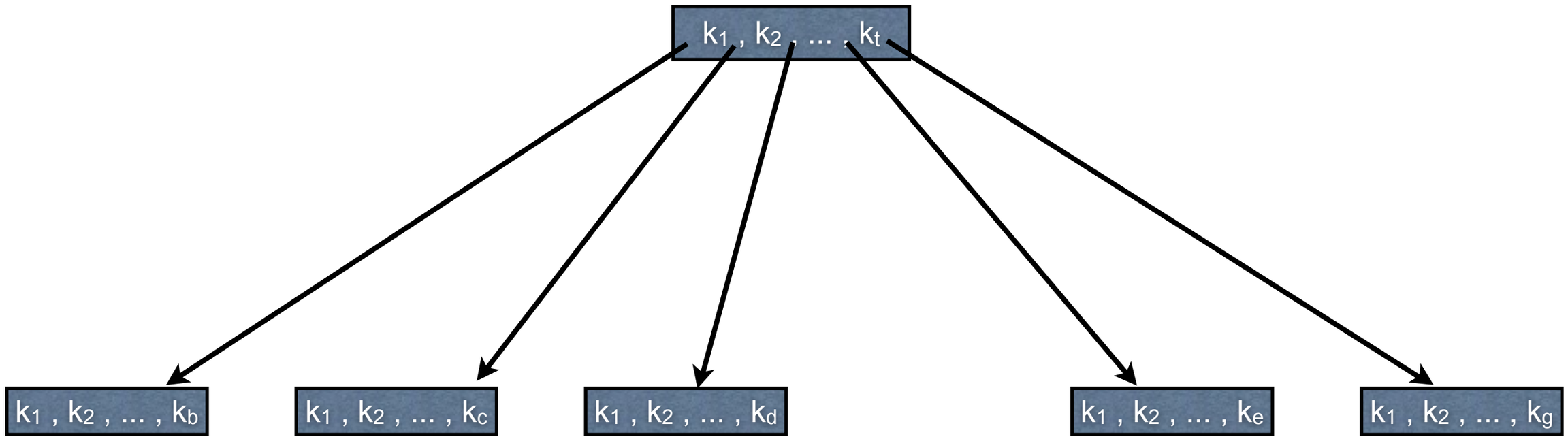
töröl(30)



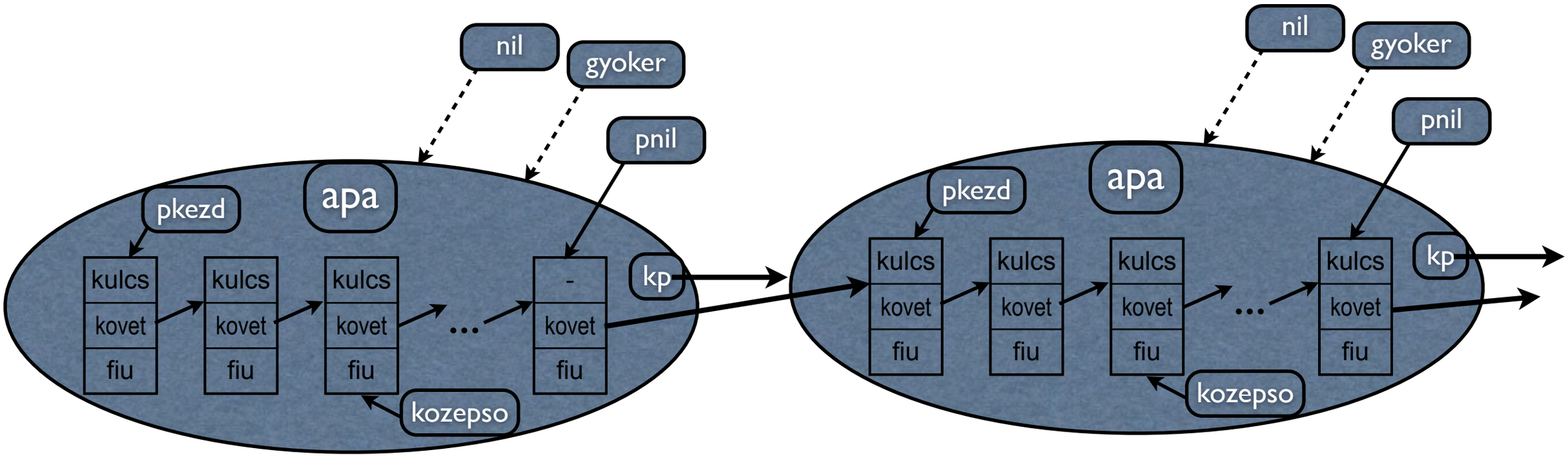
B+ fa



B+ fa



B+ fa


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

2-3-fák

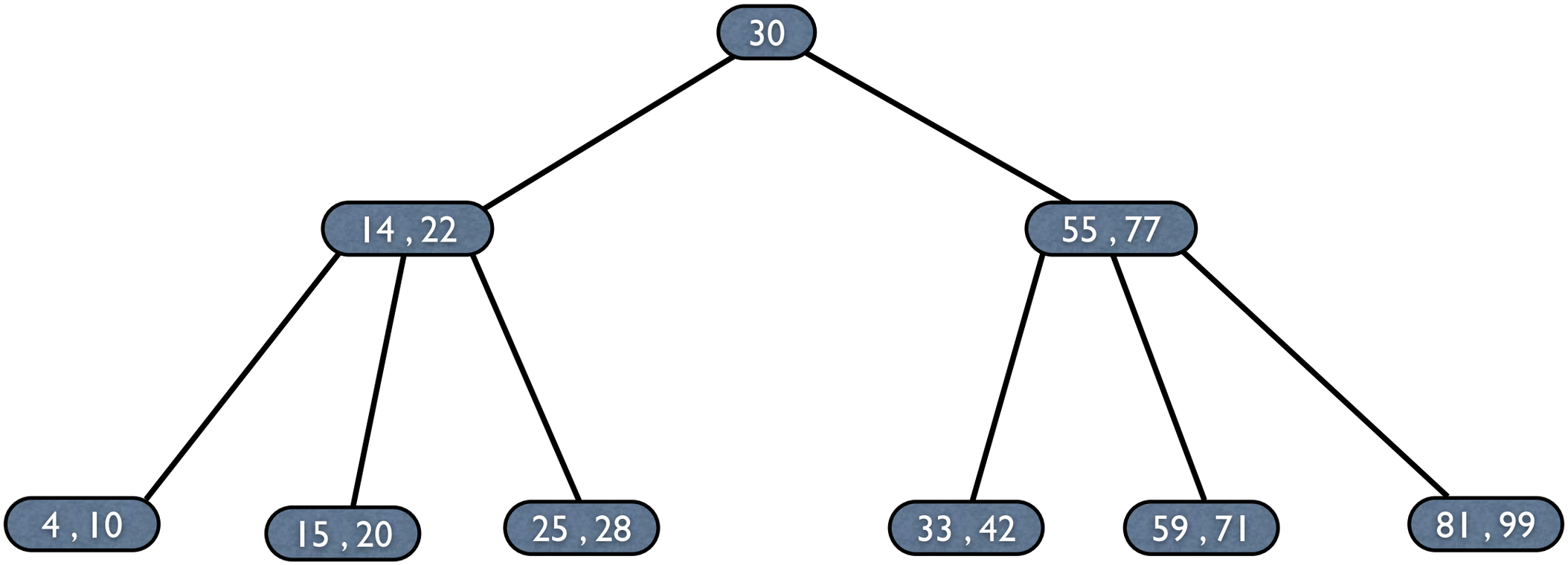
2-3 fa a (t = 1) rangú B-fa, ha teljesül rá az alábbi 4 feltétel:

- Általános keresőfa.
- A gyökér kivételével minden $p \in F$ pontra $1 \leq Rang(p) \leq 2$,

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

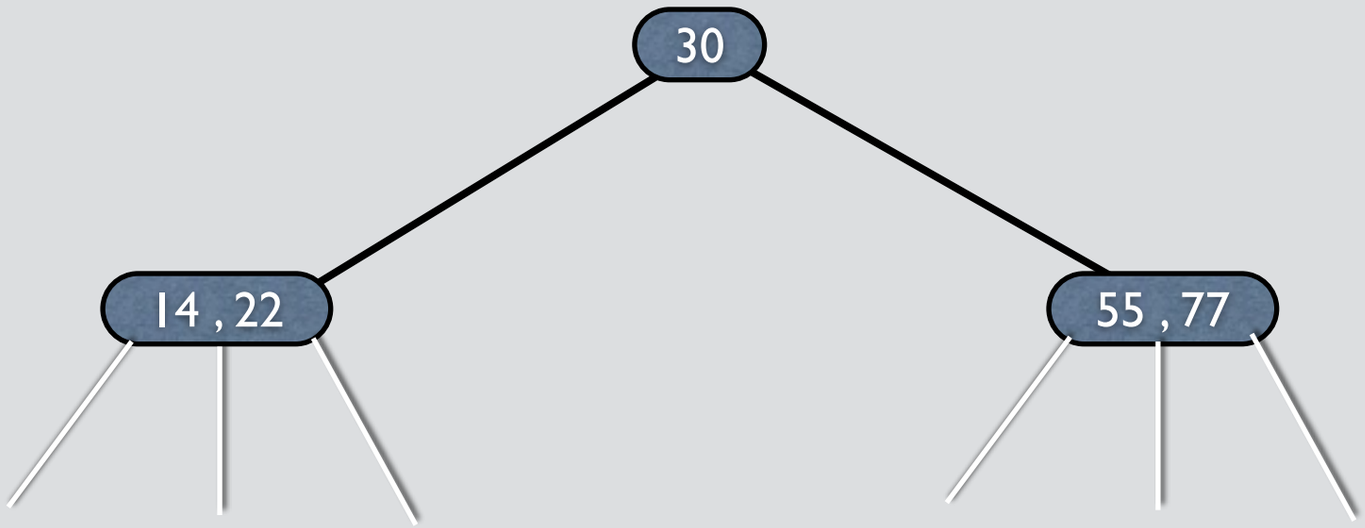
  azaz minden levél pont mélysége azonos.

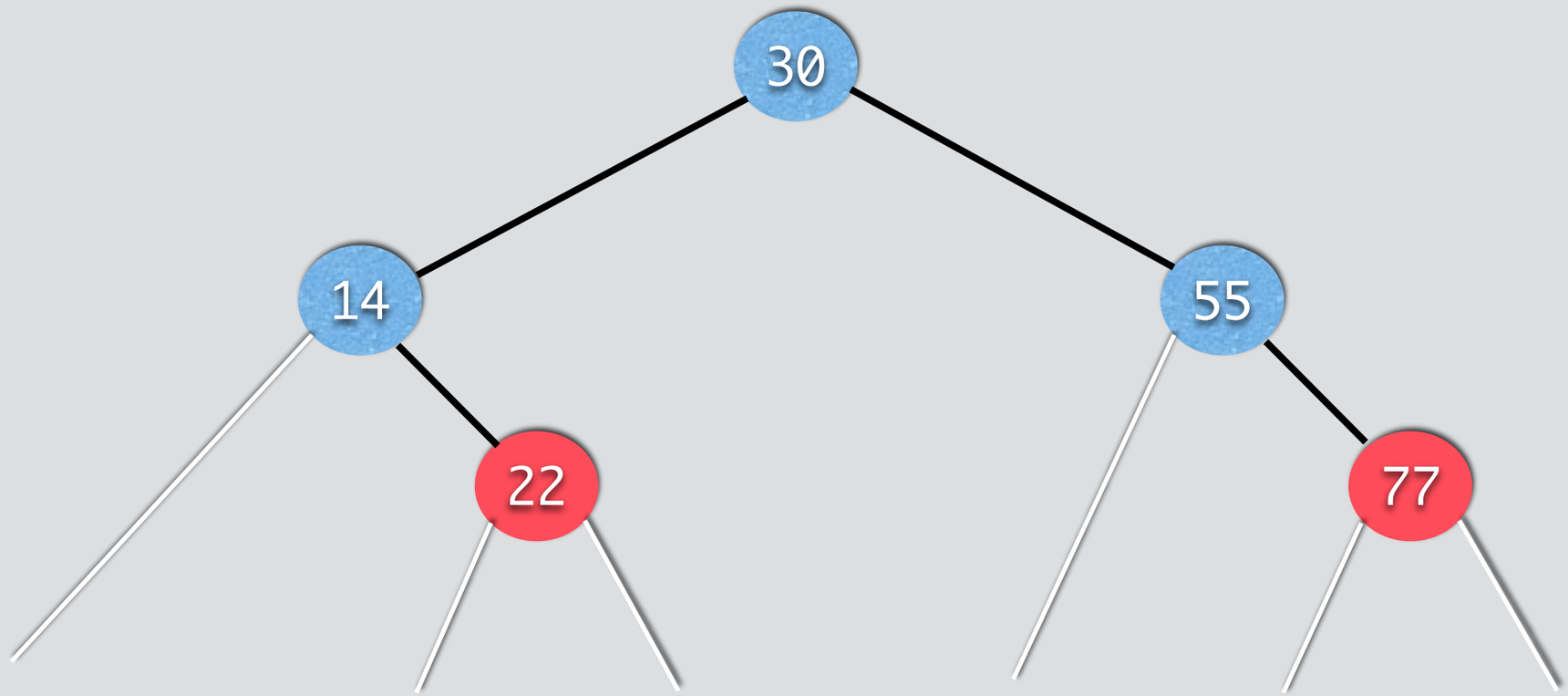
Példa



Piros-Fekete fák

Példa

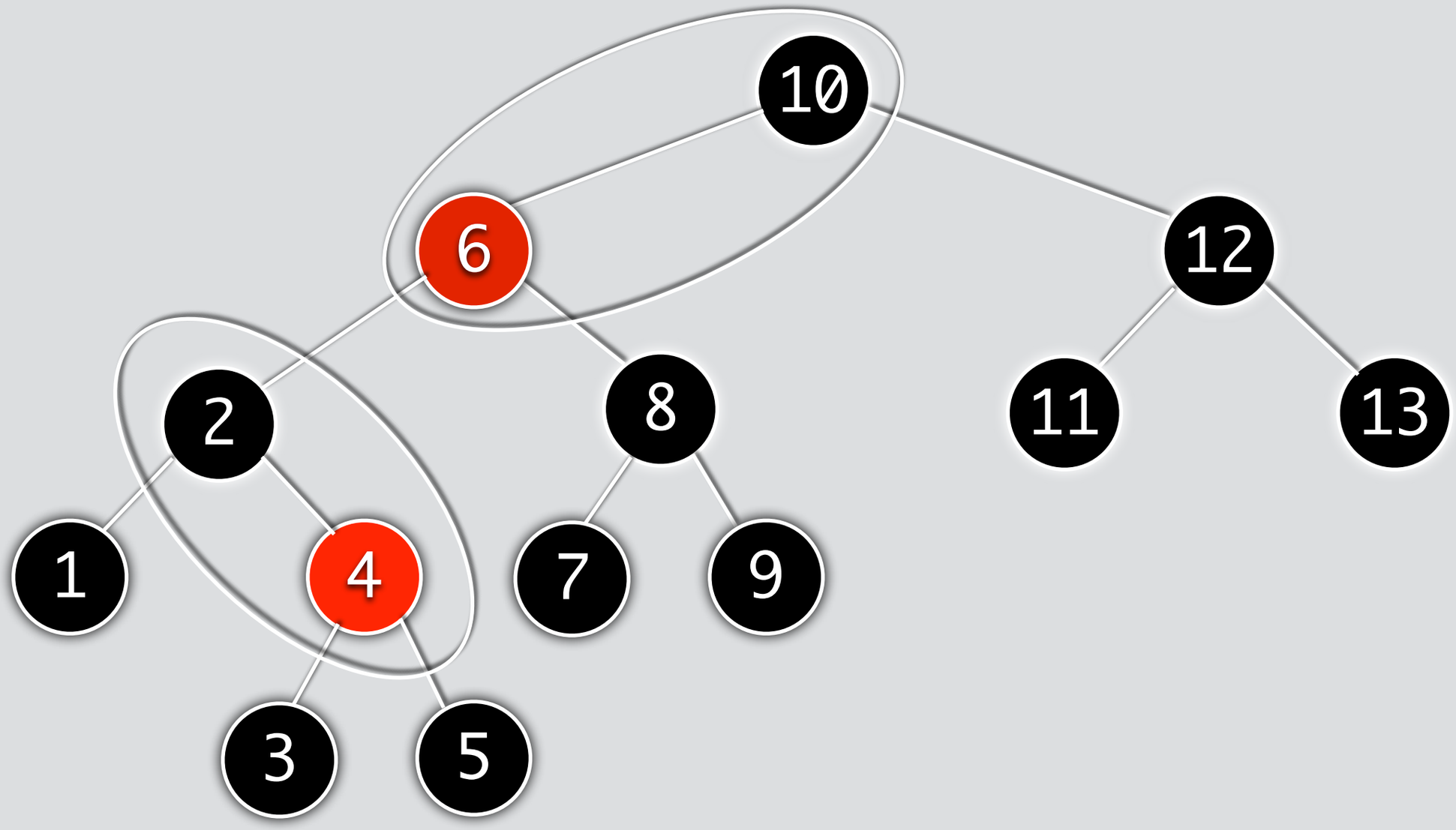




Piros-Fekete fák

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

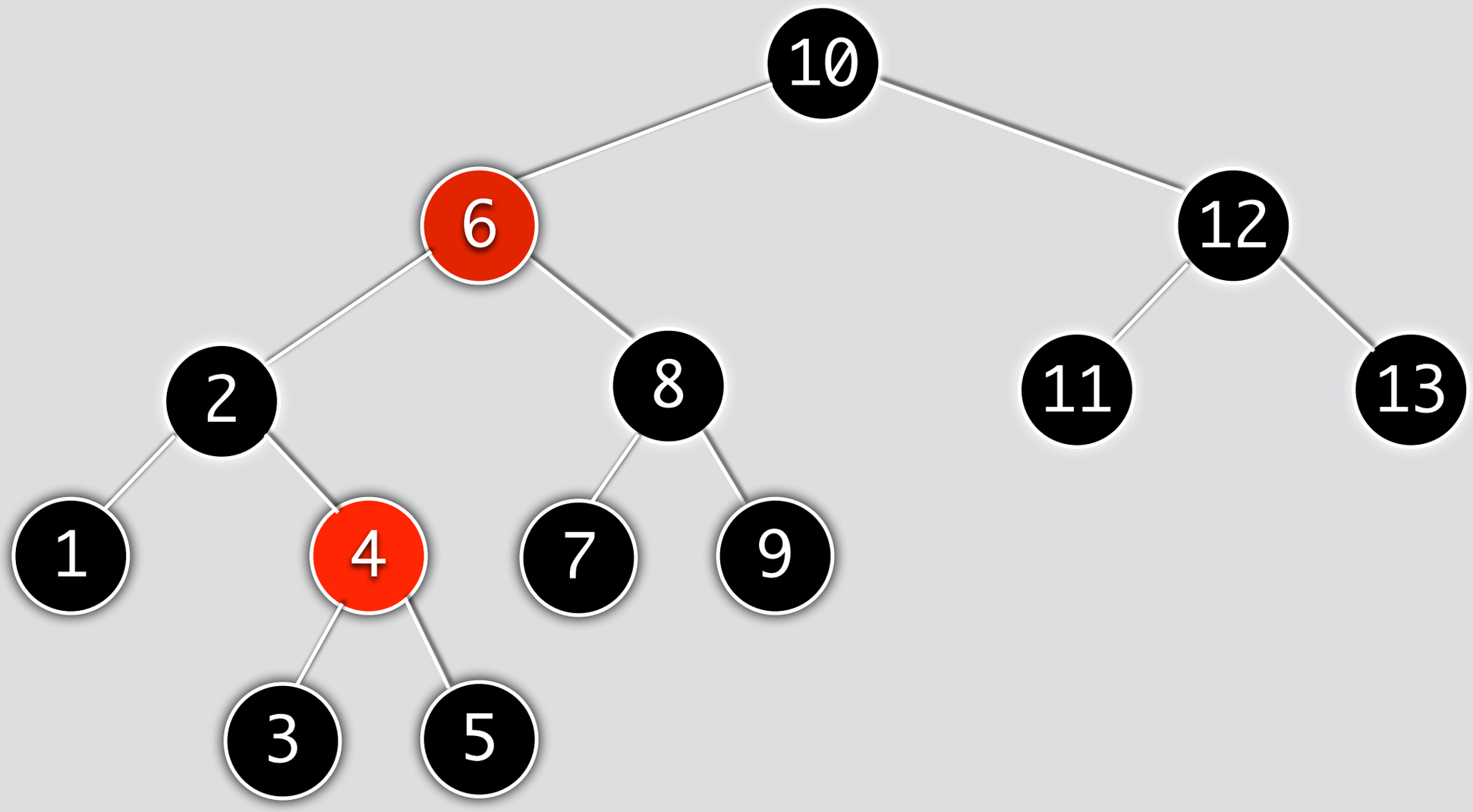
Tehát a fa minden pontja tartalmazza a szín, kulcs, bal, jobb és apa mezőket. Ha egy ponthoz tartozó fiú vagy apa nem létezik, akkor a megfelelo mező a NIL értéket tartalmazza. Úgy tekintjük, hogy az ilyen NIL mutató értékek a bináris keresőfa külső (levél) pontjaira mutatnak, míg a fa kulcsot tartalmazó 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:

Tétel: 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(logn), ahol n a pontok száma. Azt kell megvizsgálni a piros-fekete fa tulajdonságok karbantartásának mekkora az időigénye.