# B+ fák

Az alábbiakban Dr. Carl Burch B+-trees című Internetes tananyagának kissé bővített fordítását adjuk.

**Tartalom:**



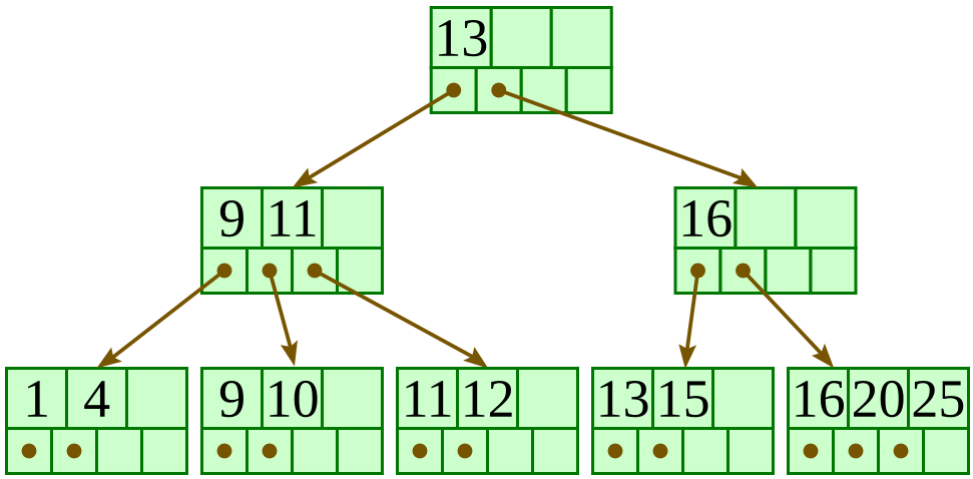
## 1. Mi a B+ fa?

Az adathalmazokon a legtöbb keresés és módosítás gyorsabban, hatékonyabban hajtható végre, ha a típusértékeket rendezve tároljuk. Nem praktikus azonban a rekordokat szekvenciálisan elhelyezni, mert így a legtöbb beszúrás és törlés kapcsán a tároló jelentős részének újraírása válna szükségessé. A rendezett vagy rendezetlen láncolt listákon pedig a keresés nem elég hatékony.

Ez arra vezet bennünket, hogy az adatokat keresőfába rendezve képzeljük el. Az első ötletünk az AVL fák vagy a piros-fekete fák alkalmazása lehetne, most azonban az adatokat egy véletlen elérésű háttértáron, pl. egy mágneslemezen kívánjuk elhelyezni. A mágneslemezek pedig úgy működnek, hogy egyszerre az adatok egy egész blokkját, tipikusan 512 byte vagy négy kilobyte mennyiségű adatot mozgatunk. Egy bináris keresőfa egy csúcsa ennek csak egy töredékét használná, ezért olyan struktúrát keresünk, ami jobban kihasználja a mágneslemez blokkjait.

Innét jön a B+ fa, amiben minden csúcs legfeljebb $d$ mutatót ($4 \le d$), és legfeljebb $d-1$ kulcsot tartalmaz. Úgy tekintjük, hogy a belső csúcsokban mindegyik referencia két kulcs "között" van, azaz egy olyan részfa gyökerére mutat, amiben minden érték a két kulcs között található (mindegyik csúcshoz hozzáképzelve balról egy "mínusz végtelen", jobbról egy "plusz végtelen" értékű kulcsot).

Íme $d=4$ értékre egy elég kicsi fa:




Az adatok a levélszinten vannak. A belső kulcsok csak *hasító kulcsok*. Egy adott kulcsú adat keresése során ezek alapján tudhatjuk, melyik ágon keressünk tovább. A levélszinten minden kulcshoz tartozik egy mutató, ami a megfelelő adatrekordra hivatkozik. (A leveleket a $d$-edik mutatókkal gyakran listába fűzik.)

A B+ fák esetében megköveteljük, hogy a gyökércsúcstól mindegyik levél azonos távolságra legyen. Így a fenti ábrán látható fában tárolt 11 kulcs bármelyikére rákeresve (amelyek mindegyikét a legalsó, a levél szinten találjuk: a belső csúcsok kulcsai csak a tájékozódást szolgálják), három csúcsot érintünk (a gyökércsúcsot, egyik középső szintű csúcsot, és az egyik levelet).

A gyakorlatban persze $d$ sokkal nagyobb. Tegyük fel például, hogy egy-egy blokk 4KB, a kulcsaink 4 byte-os egész számok, és mindegyik mutató egy 6 byte-os relatív cím (a fájl kezdőcíméhez képest). Akkor a $d$ értékét úgy választjuk, hogy a lehető legnagyobb egész szám legyen, amire $4\ (d - 1) + 6\ d \leq 4096$. Ezt az egyenlőtlenséget $d$-re megoldva $d \leq 410$ adódik, tehát a *d=410* értéket választjuk. Mint láthatjuk, $d$ egészen nagy lehet.

Egy B+ fa a következő invariánsokat teljesíti:

- Minden levélben legfeljebb *d-1* kulcs, és ugyanennyi, a megfelelő (azaz ilyen kulcsú) adatrekordra hivatkozó mutató található.
- A gyökértől mindegyik levél ugyanolyan távol található. (Más szavakkal, minden levél azonos mélységben, a legalsó szinten van.)
- Minden belső csúcsban eggyel több mutató van, mint kulcs, ahol $d$ a felső határ a mutatók számára.
- Minden *Cs* belső csúcsra, ahol $k$ a *Cs* csúcsban a kulcsok száma: az első gyerekhez tartozó részfában minden kulcs kisebb, mint a *Cs* első kulcsa; az utolsó gyerekhez tartozó részfában minden kulcs nagyobb-egyenlő, mint a *Cs* utolsó kulcsa; és az $i$-edik gyerekhez tartozó részfában ($2 \leq i \leq k$) lévő tetszőleges $r$ kulcsra $Cs.kulcs[i-1] \leq r < Cs.kulcs[i]$.
- A gyökércsúcsnak legalább két gyereke van (kivéve, ha ez a fa egyetlen csúcsa, következésképpen az egyetlen levele is).
- Minden, a gyökértől különböző belső csúcsnak legalább $floor(d / 2)$ gyereke van. ($floor(d / 2) = d/2$ alsó egész-rész.)
- Minden levél legalább $floor(d / 2)$ kulcsot tartalmaz (kivéve, ha a fának egyetlen csúcsa van).
- A B+ fa által reprezentált adathalmaz minden kulcsa megjelenik valamelyik levélben, balról jobbra szigorúan monoton növekvő sorrendben.

A belső csúcsokban található *hasító kulcsok* segítségével tetszőleges levélcsúcsbeli kulcsot gyorsan megtalálhatunk (illetve megtudhatjuk, ha nincs a levelekben), a fenti invariánsok alapján: a csúcsokban is logaritmikusan keresve, és mindig a megfelelő ágon továbbhaladva, $O(lg\ n)$ lépésben (ahol $n$ a B+ fával ábrázolt adathalmaz mérete). A hasító kulcsok nem okvetlenül szerepelnek a levelekben. (Mint látni fogjuk, a törlő algoritmus ténylegesen is létrehoz ilyen fákat.)

Ugyan a fa magassága $O(lg\ n)$, a gyakorlatban a fa $h$ magassága az $lg\ n$ érték töredéke:
HF: Bizonyítsuk be, hogy a $log[d](n/(d-1)) \leq h \leq log[floor(d/2)](n/2)$ egyenlőtlenség teljesül!
($log[d](n)$ = az $n$ szám $d$ alapú logaritmusa.)

Ez azt jelenti, hogy a fenti *d=410* értékkel $log[410](n/409) \leq h \leq log[205](n/2)$.
Pl. ha *n=1.000.000.000 (n=egymilliárd),* akkor $2{,}4 < h < 3{,}8$, vagyis *h=3*, azaz a fának pontosan négy szintje van. Egy ilyen fánál a felső két szintet az adatbázis megnyitásakor betöltjük a központi tárba. A levélszintről viszont még egyet lépünk a tényleges adatrekord eléréséhez. Ez azt jelenti, hogy egy-egy adat eléréséhez összesen háromszor olvasunk a lemezről, *egymilliárd* rekordot tartalmazó adatbázis esetén. Ha pedig a keresett kulcsú rekord nincs az adatbázisban, csak kétszer olvasunk a lemezről.

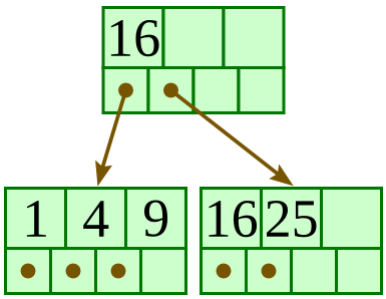
Az alábbi példákban továbbra is a *d=4* értékkel dolgozunk. A fenti invariánsok alapján ez azt jelenti, hogy minden levél legalább két kulcsot tartalmaz; minden belső csúcsnak pedig legalább két gyereke és legalább egy hasító kulcsa van.

# 2. A beszúrás algoritmusa

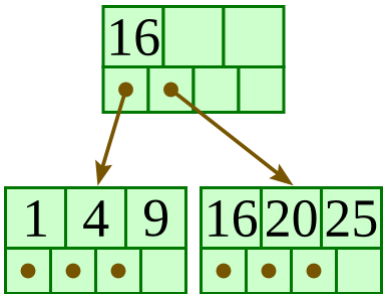
Keressük meg a kulcsnak megfelelő levelet!

1. Ha a csúcsban van üres hely, szúrjuk be a megfelelő adat/mutató párt kulcs szerint rendezetten ebbe a csúcsba!
2. Ha a csúcs már tele van, vágjuk szét két csúccsá, és osszuk el a *d* darab kulcsot egyenlően a két csúcs között! Ha a csúcs egy levél, vegyük a második csúcs legkisebb értékének másolatát, és ismételjük meg ezt a beszúró algoritmust, hogy beszúrjuk azt a szülő csúcsba! Ha a csúcs nem levél, vegyük ki a középső értéket a kulcsok elosztása során, és ismételjük meg ezt a beszúró algoritmust, hogy beszúrjuk ezt a középső értéket a szülő csúcsba! (Ha kell, a szülő csúcsot előbb létrehozzuk.)

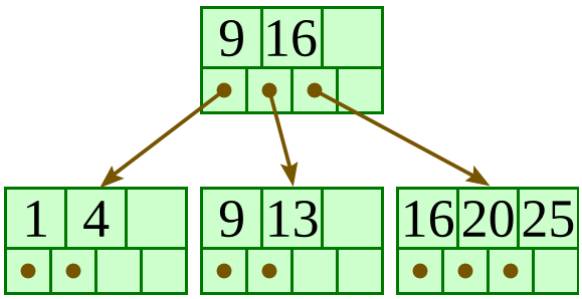
Initial:



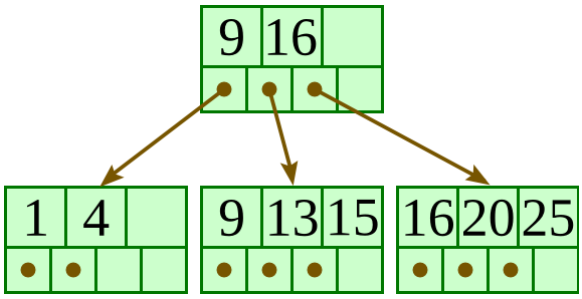

Insert 20:



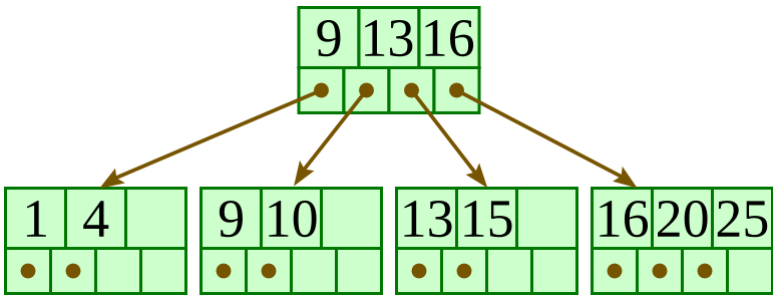

Insert 13:



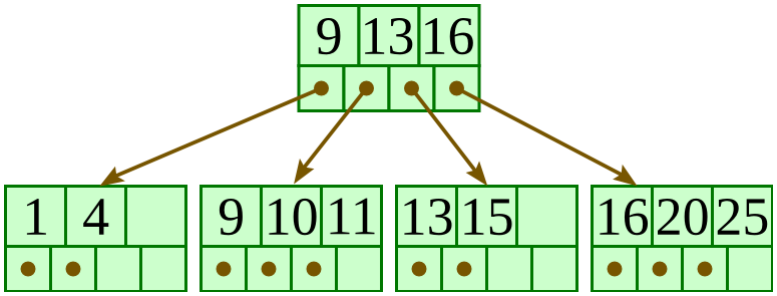

Insert 15:



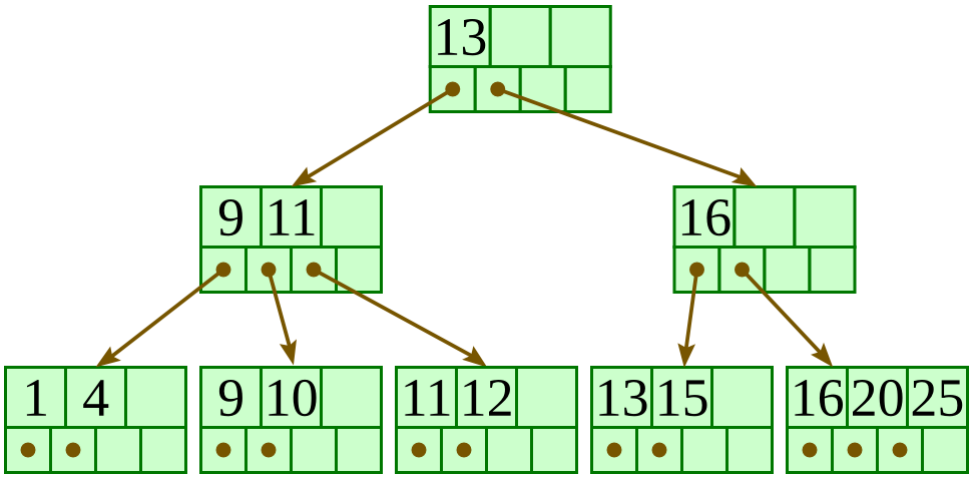

Insert 10:



Insert 11:




Insert 12:

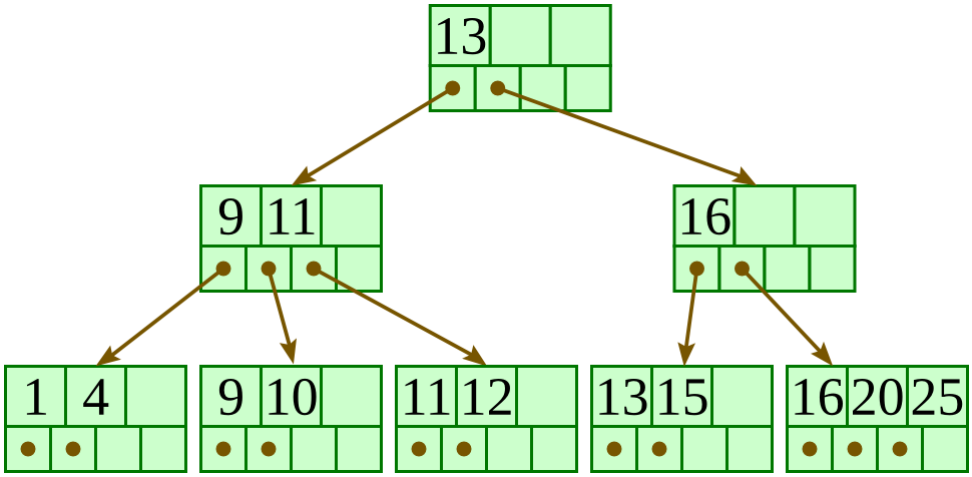



# 3. A törlés algoritmusa

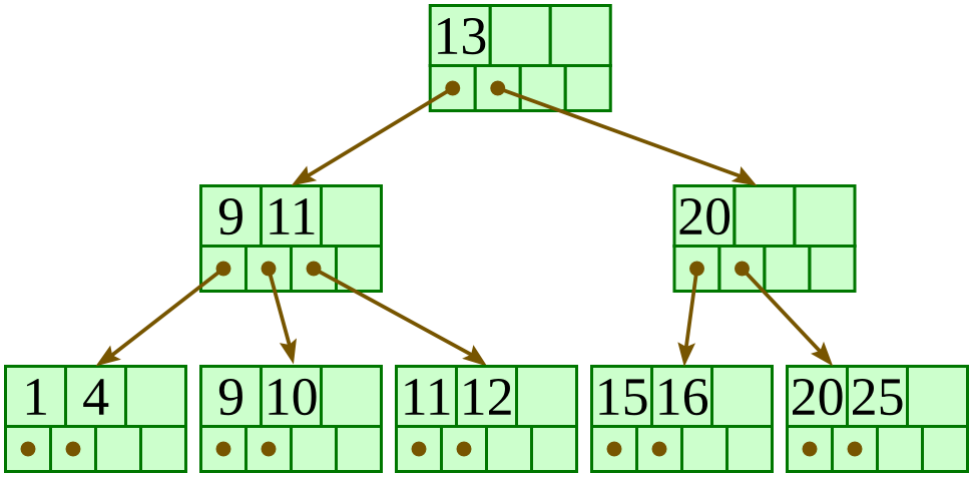
Keressük meg a törlendő kulcsot tartalmazó levelet!

1. Töröljük a megfelelő kulcsot és a hozzá tartozó mutatót a csúcsból!
2. Ha a csúcs még tartalmaz elég kulcsot és mutatót, hogy teljesítse az invariánsokat, kész vagyunk.
3. Ha a csúcsban már túl kevés kulcs van ahhoz, hogy teljesítse az invariánsokat, de a következő, vagy a megelőző testvérének több van, mint amennyi szükséges, osszuk el a kulcsokat közte és a megfelelő testvére között! Javítsuk ki a testvérek szülőjében a két testvérhez tartozó hasító kulcsot úgy, hogy megfelelően reprezentálja a köztük megváltozott vágási pontot! Ennek során csak a közös szülőjükben lévő, a testvérekhez tartozó hasító kulcsot kell átírni, törlés vagy beszúrás nélkül.
4. Ha a csúcsban már túl kevés kulcs van ahhoz, hogy teljesítse az invariánst, és a következő, valamint a megelőző testvére is a minimumon van, hogy teljesítse az invariánst, akkor egyesítsük egy vele szomszédos testvérével; ha a csúcs nem levél, akkor be kell vonnunk a szülőből a hasító kulcsot a testvérek egyesítésébe. Levél vagy nem, mindkét esetben meg kell ismételnünk a törlő algoritmust a szülőre, hogy eltávolítsuk a szülőből a hasító kulcsot, ami eddig elválasztotta a most egyesített csúcsokat — kivéve, ha a szülő a gyökércsúcs, és az utolsó kulcsa az eltávolítandó hasító kulcs, amikor is a most egyesített csúcs lesz az új gyökér (és a fa egy szinttel alacsonyabb lesz, mint a törlés előtt volt).

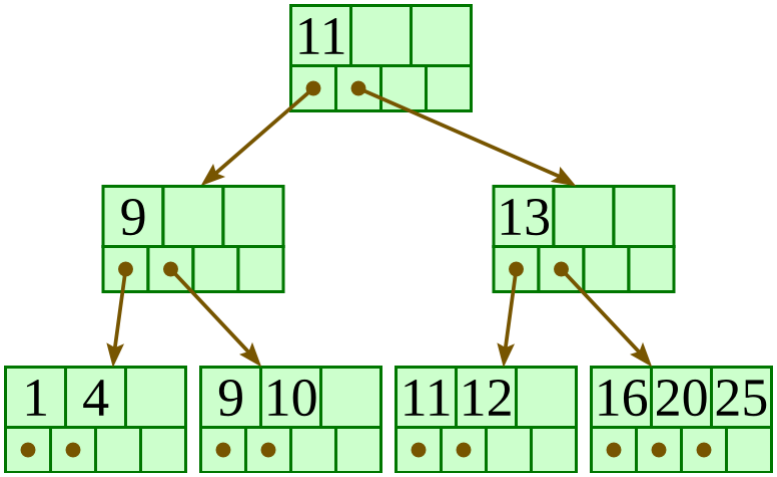
Initial:




Delete 13:




Delete 15:




Delete 1: