Adatszerkezetek és algoritmusok

Horváth Géza

kilencedik előadás

Előadások témái

1. Az algoritmusokkal kapcsolatos alapfogalmak bevezetése egyszerű példákon keresztül.
2. Az algoritmusok futási idejének aszimptotikus korlátai.
3. Az adatszerkezetekkel kapcsolatos alapfogalmak. A halmaz, a multihalmaz és a tömb adatszerkezet bemutatása.
4. Az adatszerkezetek folytonos és szétszórt reprezentációja. A verem, a sor és a lista.
5. Táblázatok, önátrendező táblázatok, hash függvények és hash táblák, ütközéskezelés.
6. Fák, bináris fák, bináris keresőfák, bejárás, keresés, beszúrás, törlés.
7. Kiegyensúlyozott bináris keresőfák: AVL fák.
8. Piros-fekete fák.
9. B-fák.
10. Gráfok, bejárás, legrövidebb út megkeresése.
11. Párhuzamos algoritmusok.
12. Eldönthetőség és bonyolultság, a P és az NP problémaosztályok.
13. Lineáris idejű rendezés. Összefoglalás.

Háttértárak vs. RAM

- Alapvető különbség a háttértárak és a számítógép belső memóriája között, hogy a háttértárak jóval nagyobb kapacitással rendelkeznek, sokkal olcsóbbak, és kikapcsolás vagy áramszünet esetén is megőrzik az adatokat. Ugyanakkor nagyságrendekkel lassabbak, mint a RAM.
- A winchesterek esetén, annak érdekében, hogy a mechanikai mozgásokra történő várakozást minél inkább lerövidítsék, úgy tervezték meg az adatokhoz történő hozzáférést, hogy egy időppontban a lemezeket több helyen is olvassák. Egy-egy ilyen egyszerre beolvasott adatot lapnak hívunk, és az eljárást, amivel a lapokat a memóriába töltjük, lapozásnak.
- Sok esetben tovább tart az adatok betöltése a háttértárból a memóriába, mint az adatok feldolgozása, ezért külön kell kezelni a futási idő két fő összetevőjét:
  - a háttértárhoz történő hozzáférések számát, és
  - a CPU számítási idejét.

A B-fa

- A B-fák olyan keresőfák, melyeket alapvetően a merevlemezes háttértárakon történő adatok kezelésére alakítottak ki. Általában jól használhatóak egyéb - nem merevlemezeket használó - háttértárak esetén is.
- A B-fák nem bináris fák. Alapvető céljuk a lapozások számának minimalizálása.
- Figyelembe véve, hogy a háttértárakban történő olvasás, írás és keresés végrehajtása sokkal több ideig tart, mint ugyanezen műveletek memóriában történő megvalósítása, ezért egy művelet esetében a számítási idő mellett a háttértárban végrehajtott műveletek számát is nyilvántartjuk.
- A B-fa magasságát az határozza meg, hogy hányszor fordulunk a háttértárhoz.

B-fa – példa

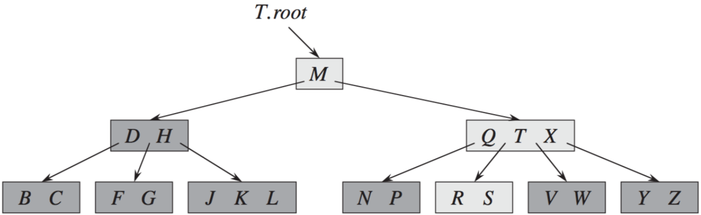
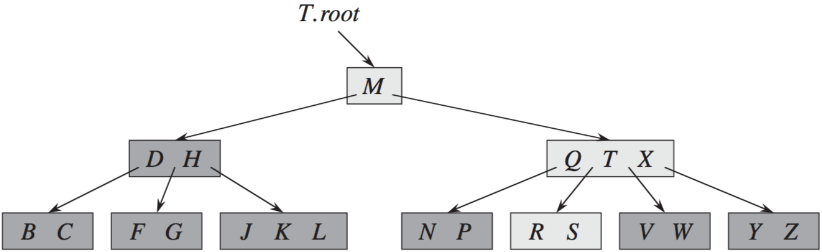


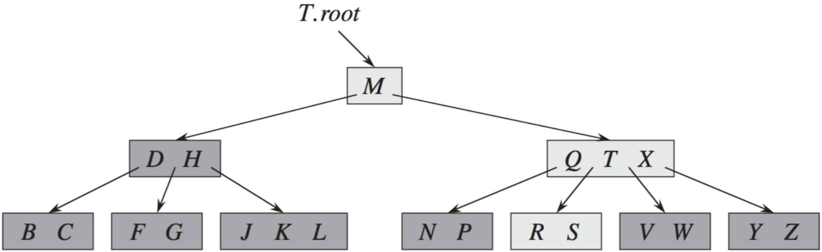
Figure 18.1 A B-tree whose keys are the consonants of English. An internal node x containing $x.n$ keys has $x.n + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

A B-fa



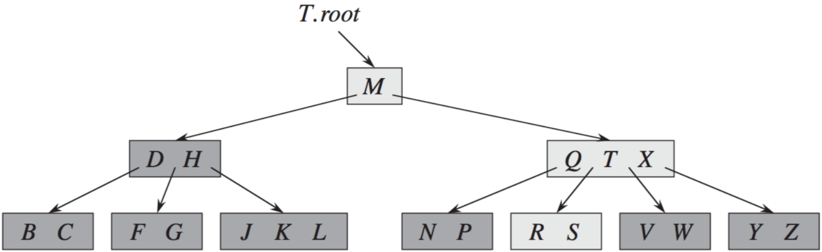
A B-fa használatának tipikus esete, amikor olyan nagy mennyiségű adattal dolgozunk, ami nem fér el egyszerre a számítógép belső memóriájában. Ekkor a B-fa algoritmusa bemásolja a háttértárról a belső memóriába azokat az adatrészeket, amelyek szükségesek az aktuális feladathoz, majd visszaírja azokat a háttértárba. A belső memóriában mindig konstans számú lap található, így a belső memória mérete nem korlátozza a teljes B-fa méretét.

A B-fa



Mivel a B-fát használó algoritmusok futási idejét alapvetően meghatározza a végrehajtott DISK-READ és DISK-WRITE műveletek száma, ezért ezen műveletek esetén szeretnénk a lehető legtöbb információt betölteni és kiírni, így a B-fa csúcsainak méretét egy teljes merevlemez lap méretére érdemes beállítani. Ez a méret egyben meghatározza a B-fa közbenső csúcsainak gyerekszámát is. (Ezáltal a fa magasságát is, a teljes fájl méretének függvényében.)

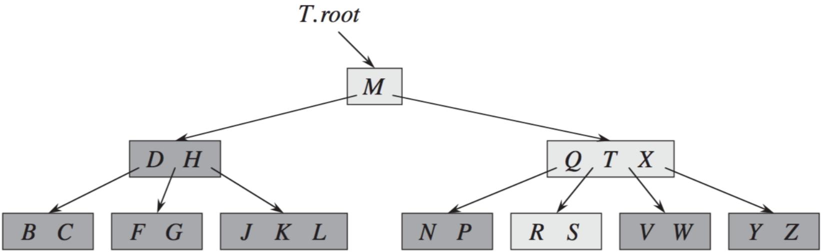
"Elágazási tényező" (branching factor)



Definíció

Fa adatszerkezet esetén elágazási tényezőnek nevezzük a csúcsok gyermekeinek számát. Amennyiben ez változó, átlagos elágazási tényezőről beszélünk.

Elágazási tényező



Nagyobb B-fák esetén sokszor előfordul, hogy 50 és 2000 közötti elágazási tényezővel rendelkeznek. A nagy elágazási tényező drasztikusan csökkenti a fa magasságát, ezáltal a háttértárban történő olvasások számát egy-egy kulcs megkeresésekor.

Elágazási tényező

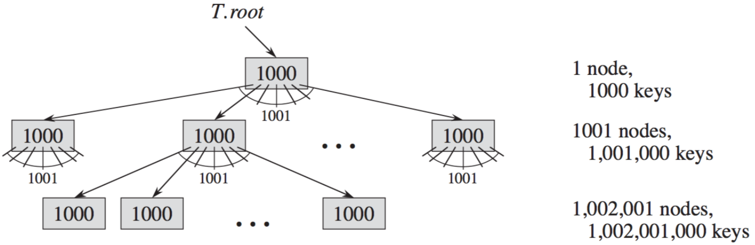
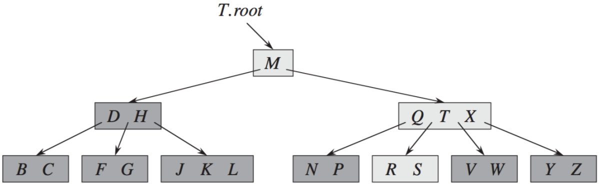


Figure 18.3 A B-tree of height 2 containing over one billion keys. Shown inside each node x is $x.n$, the number of keys in x. Each internal node and leaf contains 1000 keys. This B-tree has 1001 nodes at depth 1 and over one million leaves at depth 2.

A 18.3-as ábrán látható B-fa elágazási tényezője 1001, így minden csúcsában 1000 kulcs található. Magassága mindössze 2, a teljes fa mégis több mint egymilliárd kulcsot tartalmaz. Mivel a gyökér permanensen megtalálható a belső memórában, ezért mindössze 2 merevlemez művelettel (olvasással) bármely kulcs elérhető.

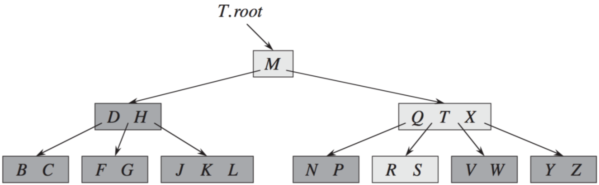
B-fa – definíció



A B-fa egy olyan fa, amelyre a következő tulajdonságok teljesülnek:

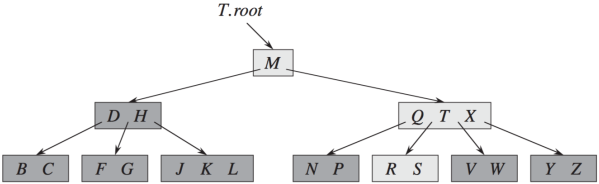
1. Minden x csúcsnak a következő attribútumai vannak:
   - x.n, az x csúcsban tárolt kulcsok száma,
   - maguk a kulcsok, növekvő sorrendben, $x.kulcs_1 < x.kulcs_2 < \ldots < x.kulcs_n$,
   - egy logikai érték, az x.levél, amelynek az értéke pontosan akkor igaz, ha x levél.

B-fa – definíció



2. Ha az x csúcs nem levél, akkor tartalmaz még x.n+1 darab mutatót is, amelyek az x csúcs gyerekeire mutatnak.
   Ezek jelölése a következő: $x.c_1, x.c_2, \ldots, x.c_{x.n+1}$.
3. Az $x.c_1$ mutató az x csúcs azon gyerekére mutat, amelyből induló részfában az $x.kulcs_1$-től kisebb kulcsok találhatóak, az $x.c_2$ arra a gyerekére, amelyből induló részfában az $x.kulcs_1$ és az $x.kulcs_2$ közötti kulcsok találhatóak, és így tovább. Végül pedig az $x.c_{x.n+1}$ kulcs x azon gyerekére mutat, amelyből induló részfában az $x.kulcs_n$-től nagyobb kulcsok vannak.
4. Minden levél azonos szinten helyezkedik el, azaz a gyökértől azonos távolságra van.

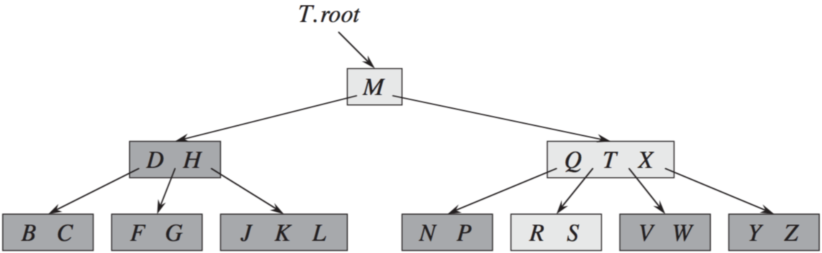
B-fa – definíció



5. A csúcsokban tárolható kulcsok számának minden B-fa esetén van egy alsó és egy felső korlátja. Ezek a korlátok egy egész számtól függenek, mely számot a B-fa minimális fokszámának hívunk, t betűvel jelölünk és értéke legalább 2.
   - A gyökérben található kulcsok száma legalább 1, ha a fa nem üres. Az összes többi csúcs minim t-1 kulcsot kell, hogy tartalmazzon.
   - Minden csúcs legfeljebb 2t-1 kulcsot tartalmazhat.

További példák a táblán!

A B-fa magassága



Tétel

Legyen $n \geq 1$ természetes szám. Bármely n kulcsot tartalmazó, $t \geq 2$ minimális fokszámú B-fa esetén a fa magassága

$$h \leq log_t \frac{n+1}{2}.$$

Műveletek B-fákkal, mint adatszerkezetekkel

Műveletek:

- adatszerkezetek **létrehozása**: B-TREE-CREATE
- adatszerkezetek **módosítása**
  - elem hozáadása: B-TREE-INSERT
  - elem törlése: B-TREE-DELETE
  - elem cseréje: nincs
- elem **elérése**: B-TREE-SEARCH

B-TREE-CREATE(T)

```
x = ALLOCATE-NODE()
x.leaf = TRUE
x.n = 0
DISK-WRITE(x)
T.root = x
```

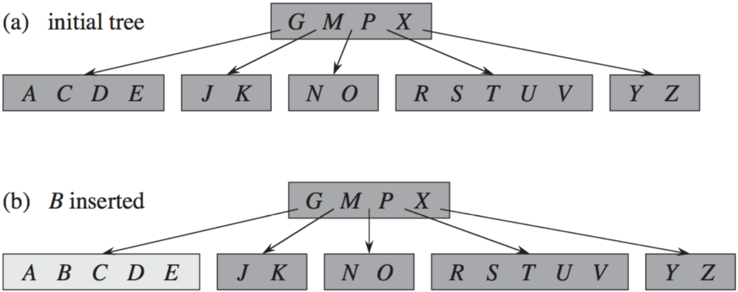
B-TREE-CREATE requires $O(1)$ disk operations and $O(1)$ CPU time.

B-fa – keresés

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3      i = i + 1
4  if i ≤ x.n and k == x.key_i
5      return (x, i)
6  elseif x.leaf
7      return NIL
8  else DISK-READ(x.c_i)
9      return B-TREE-SEARCH(x.c_i, k)
```

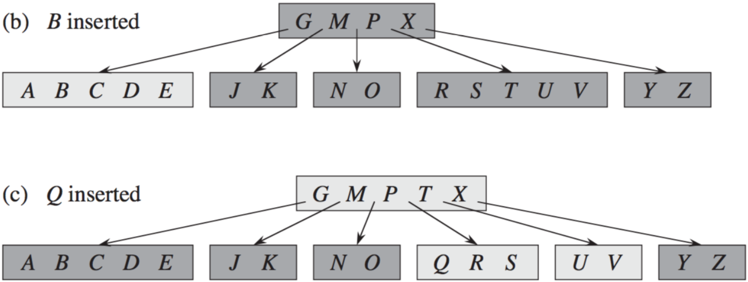
A B-TREE-SEARCH eljárás végrehajtásához $\mathcal{O}(h)$=$\mathcal{O}(log_t n)$ lapozás szükséges, ahol h a fa magasságát, n pedig a kulcsok számát jelöli. Mivel $x.n < 2 * t$, a 2-3 sorokban található while ciklus lépésszáma $\mathcal{O}(t)$ minden csúcsra, így a teljes futási idő $\mathcal{O}(t * h)$=$\mathcal{O}(t * log_t n)$.

B-fa – beszúrás



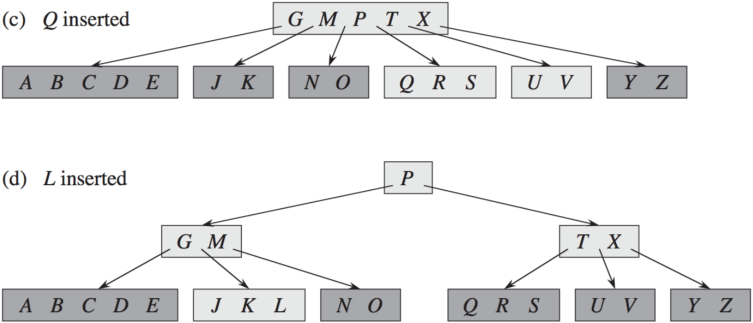
18.7. ábra. Egy B-fába kulcsokat szúrunk be. A B-fa t minimális fokszáma 3, ezért egy csúcsnak legfeljebb 5 kulcsa lehet. Azokat a csúcsokat, amelyeket a beszúrás alatt módosítottuk, világosszürke színnel jelöltük. **(a)** A példában szereplő fa kezdeti állapota. **(b)** A B beszúrásának az eredménye; ez egy egyszerű eset, egy levélbe kellett beszúrni. **(c)** Az előző fába egy Q kulcsot szúrtunk be. Az $RSTUV$ csúcs két részre bomlott, az RS és az UV csúcsokra, a T a gyökércsúcsba ment, és a Q a bal oldali új csúcsba (RS-be) került. **(d)** Az előző fába egy L kulcsot szúrtunk be. A gyökércsúcsot fel kellett bontani, mivel már telített volt, így a B-fa magassága eggyel nőtt. Ezután az L kulcsot a JK-t tartalmazó levélbe szúrtuk be. **(e)** Az előző fába az F kulcsot szúrtuk be. Az $ABCDE$ csúcsot szétvágtuk, és ezután az F kulcsot a jobb oldali új csúcsba (DE-be) szúrtuk be.

B-fa – beszúrás



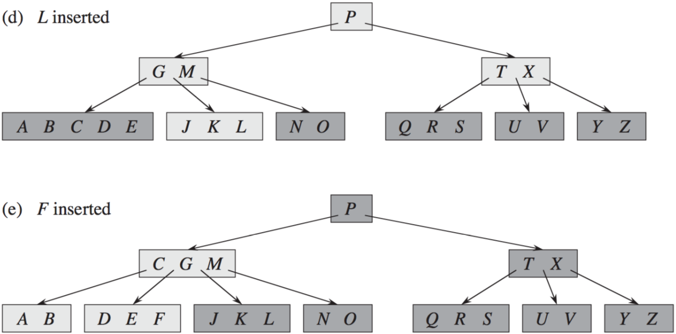
18.7. ábra. Egy B-fába kulcsokat szúrunk be. A B-fa t minimális fokszáma 3, ezért egy csúcsnak legfeljebb 5 kulcsa lehet. Azokat a csúcsokat, amelyeket a beszúrás alatt módosítottuk, világosszürke színnel jelöltük. **(a)** A példában szereplő fa kezdeti állapota. **(b)** A B beszúrásának az eredménye; ez egy egyszerű eset, egy levélbe kellett beszúrni. **(c)** Az előző fába egy Q kulcsot szúrtunk be. Az $RSTUV$ csúcs két részre bomlott, az RS és az UV csúcsokra, a T a gyökércsúcsba ment, és a Q a bal oldali új csúcsba (RS-be) került. **(d)** Az előző fába egy L kulcsot szúrtunk be. A gyökércsúcsot fel kellett bontani, mivel már telített volt, így a B-fa magassága eggyel nőtt. Ezután az L kulcsot a JK-t tartalmazó levélbe szúrtuk be. **(e)** Az előző fába az F kulcsot szúrtuk be. Az $ABCDE$ csúcsot szétvágtuk, és ezután az F kulcsot a jobb oldali új csúcsba (DE-be) szúrtuk be.

B-fa – beszúrás



18.7. ábra. Egy B-fába kulcsokat szúrunk be. A B-fa t minimális fokszáma 3, ezért egy csúcsnak legfeljebb 5 kulcsa lehet. Azokat a csúcsokat, amelyeket a beszúrás alatt módosítottuk, világosszürke színnel jelöltük. **(a)** A példában szereplő fa kezdeti állapota. **(b)** A B beszúrásának az eredménye; ez egy egyszerű eset, egy levélbe kellett beszúrni. **(c)** Az előző fába egy Q kulcsot szúrtunk be. Az $RSTUV$ csúcs két részre bomlott, az RS és az UV csúcsokra, a T a gyökércsúcsba ment, és a Q a bal oldali új csúcsba (RS-be) került. **(d)** Az előző fába egy L kulcsot szúrtunk be. A gyökércsúcsot fel kellett bontani, mivel már telített volt, így a B-fa magassága eggyel nőtt. Ezután az L kulcsot a JK-t tartalmazó levélbe szúrtuk be. **(e)** Az előző fába az F kulcsot szúrtuk be. Az $ABCDE$ csúcsot szétvágtuk, és ezután az F kulcsot a jobb oldali új csúcsba (DE-be) szúrtuk be.

B-fa – beszúrás



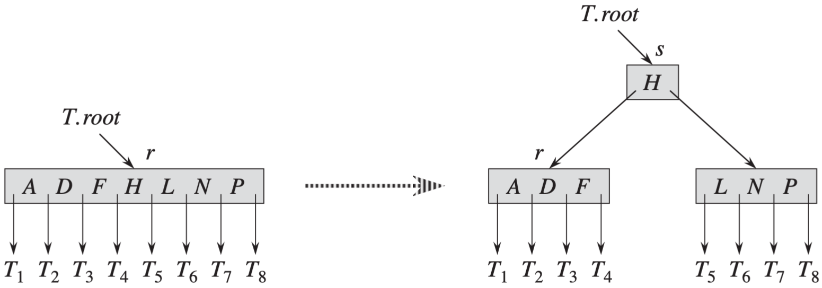
18.7. ábra. Egy B-fába kulcsokat szúrunk be. A B-fa t minimális fokszáma 3, ezért egy csúcsnak legfeljebb 5 kulcsa lehet. Azokat a csúcsokat, amelyeket a beszúrás alatt módosítottuk, világosszürke színnel jelöltük. **(a)** A példában szereplő fa kezdeti állapota. **(b)** A B beszúrásának az eredménye; ez egy egyszerű eset, egy levélbe kellett beszúrni. **(c)** Az előző fába egy Q kulcsot szúrtunk be. Az $RSTUV$ csúcs két részre bomlott, az RS és az UV csúcsokra, a T a gyökércsúcsba ment, és a Q a bal oldali új csúcsba (RS-be) került. **(d)** Az előző fába egy L kulcsot szúrtunk be. A gyökércsúcsot fel kellett bontani, mivel már telített volt, így a B-fa magassága eggyel nőtt. Ezután az L kulcsot a JK-t tartalmazó levélbe szúrtuk be. **(e)** Az előző fába az F kulcsot szúrtuk be. Az $ABCDE$ csúcsot szétvágtuk, és ezután az F kulcsot a jobb oldali új csúcsba (DE-be) szúrtuk be.

B-fa – beszúrás

```
B-TREE-INSERT(T, k)
r = T.root
if r.n == 2t − 1
    s = ALLOCATE-NODE()
    T.root = s
    s.leaf = FALSE
    s.n = 0
    s.c_1 = r
    B-TREE-SPLIT-CHILD(s, 1)
    B-TREE-INSERT-NONFULL(s, k)
else B-TREE-INSERT-NONFULL(r, k)
```

A B-TREE-INSERT eljárás végrehajtásához $\mathcal{O}(h){=}\mathcal{O}(log_t n)$ lapozás szükséges, ahol h a fa magasságát, n pedig a kulcsok számát jelöli. A teljes futási idő $\mathcal{O}(t * h){=}\mathcal{O}(t * log_t n)$.

B-fa – beszúrás



18.6. ábra. Egy olyan gyökércsúcs szétvágása, amelyre $t = 4$. Az r gyökércsúcs két részre válik, és egy új s gyökércsúcs keletkezik. Az új gyökércsúcs r középső kulcsát tartalmazza, és két gyereke van, ezek r félbevágásából származnak. A gyökércsúcs szétvágásával a B-fa magassága eggyel nő.

B-fa – beszúrás

```
B-TREE-SPLIT-CHILD(x, i)
z = ALLOCATE-NODE()
y = x.c_i
z.leaf = y.leaf
z.n = t - 1
for j = 1 to t - 1
    z.key_j = y.key_{j+t}
if not y.leaf
    for j = 1 to t
        z.c_j = y.c_{j+t}
y.n = t - 1
for j = x.n + 1 downto i + 1
    x.c_{j+1} = x.c_j
x.c_{i+1} = z
for j = x.n downto i
    x.key_{j+1} = x.key_j
x.key_i = y.key_t
x.n = x.n + 1
DISK-WRITE(y)
DISK-WRITE(z)
DISK-WRITE(x)
```

B-fa – beszúrás

```
B-TREE-INSERT-NONFULL(x, k)
i = x.n
if x.leaf
    while i ≥ 1 and k < x.key_i
        x.key_{i+1} = x.key_i
        i = i − 1
    x.key_{i+1} = k
    x.n = x.n + 1
    DISK-WRITE(x)
else while i ≥ 1 and k < x.key_i
        i = i − 1
    i = i + 1
    DISK-READ(x.c_i)
    if x.c_i.n == 2t − 1
        B-TREE-SPLIT-CHILD(x, i)
        if k > x.key_i
            i = i + 1
    B-TREE-INSERT-NONFULL(x.c_i, k)
```

Irodalomjegyzék