ELTE | IK

INFORMATIKAI KAR

Adatbázisok 2

Indexek: hasító táblák

Szempontok

- Három fő cél van:
  - Gyors lekérdezés
  - Gyors adatmódosítás
  - Minél kisebb tárolási terület
- Nincs „legjobb” megoldás, egy cél a többi rovására javítható.
- Milyen feladataink vannak?
  - Lekérdezések
  - Módosítások: beszúrás (insert), törlés (delete), frissítés (update)

Költségek

- A memória műveletek nagyságrendekkel gyorsabbak, mint a háttértárról olvasás és kiírás.
- Az ABKR blokkokat olvas és ír.
  - A blokkméret egyes rendszerekben eltér (számunkra ez most mindegy).
  - Az egyszerűsítés kedvéért most ne foglalkozzunk a pufferkezelő munkájával (így felső korlátot adunk a költségekre).
  - A blokkok tartalmaznak fejlécet, rekordokat, üres helyeket – ezeket se vegyük most figyelembe.
- Jelöljük B-vel egy tábla rekordjainak a tárolásához szükséges blokkok számát.
- Két féle bonyolultságot szokás vizsgálni:
  - Átlagos eset
  - Legrosszabb eset

Milyen lekérdezéseket vizsgáljunk?

- A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni.
- A legegyszerűbb kiválasztás:

```
SELECT *
FROM R
WHERE A=a;
```

  - Ahol „A” az „R” reláció egy oszlopa, „a” pedig egy konstans.
- A feltételnek megfelelő rekordból **lehet-e több** vagy sem?
  - Vizsgáljuk azt az esetet, amikor elég az első előfordulást megkeresni.
- **Egyenletességi feltétel:** az *A=a* feltételnek eleget tévő rekordok száma nagyjából egyforma az „a”-tól függetlenül.

Fájlszervezési módszerek

- Segédstruktúra nélkül:
  - Kupac szervezés (heap)
  - Rendezett állomány
- Segédstruktúrák:
  - Hasító index (hash)
  - Elsődleges index (ritka index)
  - Másodlagos index (sűrű index)
  - Többszintű index
  - B+ fa index
  - Bittérkép index

Kupac szervezés

- A rekordokat a blokk első üres helyére tesszük a beérkezés sorrendjében.
- Tárméret: **B**
- Keresési idő:
    - A legrosszabb esetben: **B**
    - Átlagos esetben (egyenletességi feltétellel): **B/2**
- Beszúrás:
    - Az utolsó blokkba tesszük a rekordot: 1 olvasás + 1 írás
- Módosítás:
    - 1 keresés + 1 írás
- Törlés:
    - 1 keresés + 1 írás
    - Ilyenkor üres hely marad vagy törlési bitet állítjuk át (tombstone)

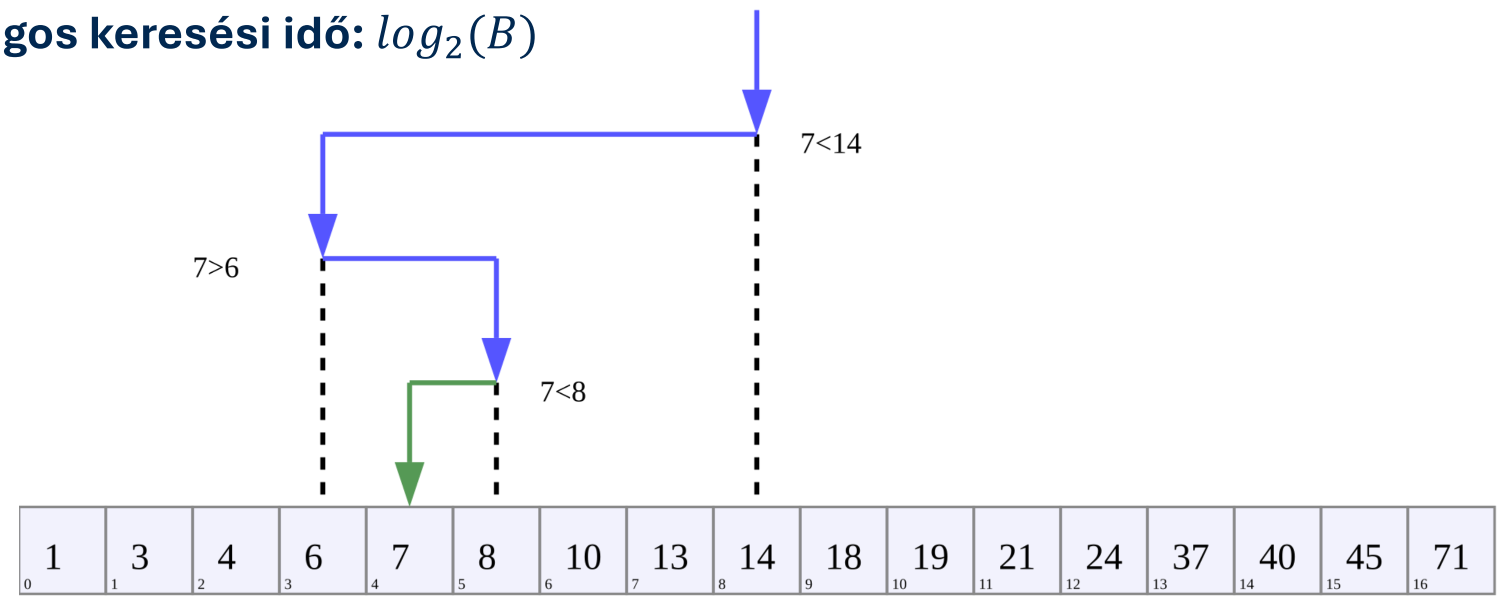
ELTE | IK

Rendezett állomány

- Egy **rendező mező** alapján rendezett, azaz a blokkok láncolva vannak és a következő blokkban nagyobb értékű rekordok szerepelnek, mint az előzőben.
- Ha a **rendező mező** és a **kereső mező** nem esik egyben, akkor igazából olyan mintha kupac szervezést használnánk :/
- Ha a rendező mező és a kereső mező egybeesik, akkor **bináris** (logaritmikus) **keresést** lehet alkalmazni.

Keresés rendezett állományban

- Beolvassuk a középső blokkot. Ha nincs benne a keresett rekord, akkor eldöntjük, hogy a blokklánc első vagy második felében szerepelhet-e.
- Beolvassuk a felezett blokklánc középső blokkját és addig folytatjuk, amíg megtaláljuk a keresett rekordot vagy a vizsgálandó maradék blokklánc már csak 1 blokkból áll.
- **Átlagos keresési idő:** $log_2(B)$



Beszúrás rendezett állományba

- Keresés + üres hely készítése miatt a rekordok eltolása az összes blokkban, az adott találati blokktól kezdve.
- Azaz átlagosan B/2 blokkot kell beolvasni és visszaírni, azaz összesen B darab művelet.
- A keresés gyors, a beszúrás lassú :/
- Lehetséges megoldások:
  - Gyűjtő (túlcsordulási) blokk használata.
  - Üres helyeket hagyunk a blokkokban.

Gyűjtő blokk használata

- Az új rekordok számára nyitunk egy blokkot, ha pedig betelik, akkor hozzáláncolunk újabb blokkokat.
- A keresést két helyen végezzük:
    - Először keresés a rendezett részben: $log_2(B - G)$
    - Ha nem találjuk, akkor a gyűjtőben is megnézzük: G blokk olvasása, ahol G a gyűjtő mérete.
    - Összesen: $log_2(B - G) + G$
- Ha G túl nagy a $log_2(B)$-hez képest, akkor újrarendezzük a teljes fájlt.
- A rendezés költsége, ha befér az adat a memóriába (később részletesebben kifejtjük): $B * log_2(B)$

Üres helyek a blokkokban

- Amikor feltöltjük a blokkokat üres helyeket is hagyunk, pl. félig töltjük fel.
- Előny: a keresés után egyszerűen visszaírjuk a blokkot, amelyikbe beírtuk az új rekordot (azaz nem kell semmit átszervezni/átrendezni).
- A tárméret (ha félig üresek): **2*B**
- Keresési idő: $log_2(2 * B) = 1 + log_2(B)$
- Ha betelik egy blokk, vagy elér egy határt a telítettsége, akkor 2 blokkba osztjuk szét a rekordjait, a rendezettség fenntartásával.

Törlés rendezett állományból

- A keresés ideje + a törlés elvégzése (vagy a törlési bit beállítása) után visszaírás (1 blokkírás).
- Túl sok törlés után újraszervezhetjük (pl. egymás utáni blokkok összevonása).

Kupac szervezés vagy rendezett állomány?

- Alapértelmezetten a kupac szervezés a **legelterjedtebb** a relációs DBMS-ek körében (pl. Oracle, PostgreSQL)
- **Kivétel**: MySQL InnoDB (segédstruktúrával együtt használja)
- A rendszerek általában **lehetőséget adnak** rendezett tárolásra is segédstruktúrák használata mellett (pl. Oracle: Index Organized Table; SQL Server: clustered index)

Hasító táblák (tördelőtáblázatok)

- A hasító tábla egy olyan adatszerkezet, amely egy hasító függvény segítségével határozza meg, hogy egy kulcs melyik kosárba tartozik.
- Az adatbázisok kontextusában a kosár általában egy blokkláncot jelöl.
  - Ha a kosarak száma előre adott és nem változik, akkor **statikus hasításról** beszélünk.
  - Ha a kosarak száma változhat, akkor **dinamikus hasításról**.

Statikus hasító tábla példda

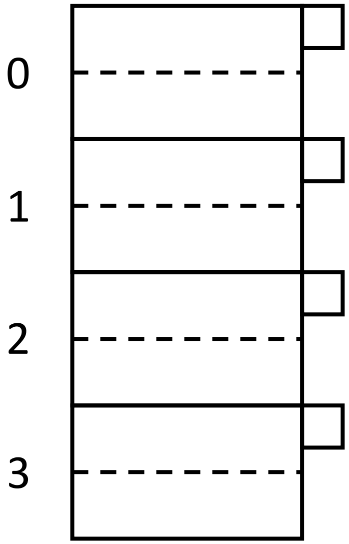
- Adott egy *h tördelőfüggvény*, amely argumentumként megkap egy keresési kulcsot és eredményül ad egy 0 és K-1 közötti egész számot, ahol K a kosarak száma, pl: $h(x) = hash(x)\ \%\ K$
- Tegyük fel, hogy egy blokkban csak 2 rekord fér el és K=4, a keresési kulcsok pedig betűk.
- Szúrjuk be a következő értékeket:

h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0



ELTE | IK

Statikus hasító tábla: beszúrás

- Ha van üres hely a megfelelő kosárhoz tartozó blokkban, akkor beszúrjuk a rekordot.

h(e) = 1

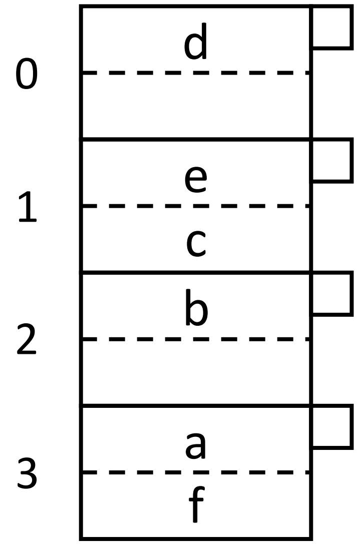
h(d) = 0

h(a) = 3

h(c) = 1

h(b) = 2

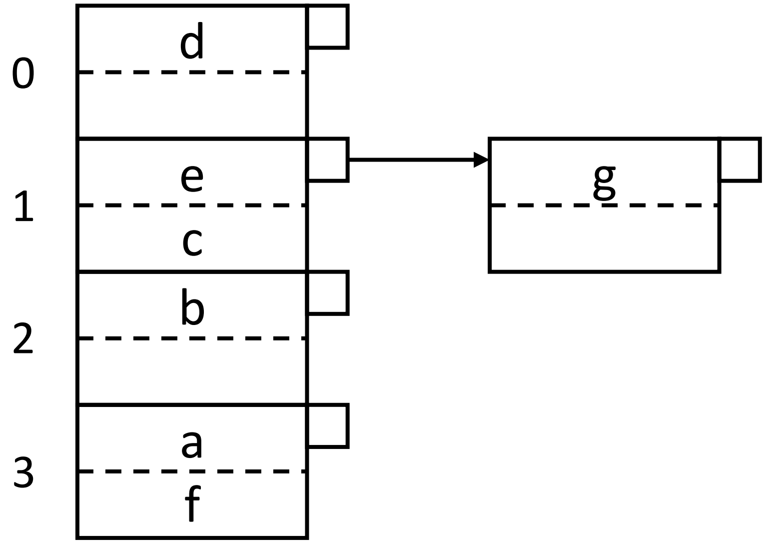
h(f) = 3



Statikus hasító tábla: beszúrás

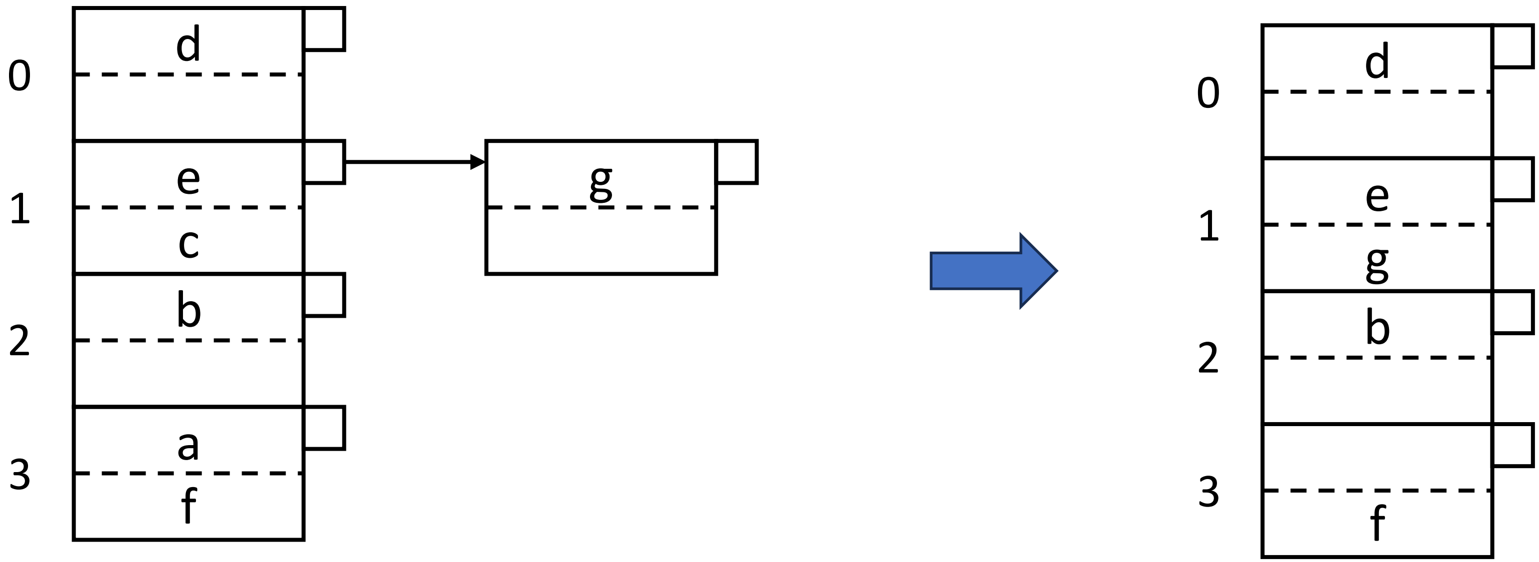
- Ha nincs hely, akkor a korábbi blokkhoz hozzá láncolunk egy újat és oda szúrjuk be a rekordot.

h(g) = 1



Statikus hasító tábla: törlés

- Töröljük az „a” és „c” rekordokat.
- Mozgassuk a blokkon belül a rekordokat vagy sem?
- A felszabaduló üres túlcsordulás blokkok törölhetjük.



A hasítófüggvényről

- A hasítófüggvény tetszőleges kulcshoz visszaad egy (általában) fix hosszúságú kódot.
- Olyan hasítófüggvényre van szükségünk, amely **gyors** és **alacsony „collision rate”-**el rendelkezik (ütközések aránya) – általában ez egy „trade-off”.
- Nincs szükségünk titkosításra (kriptográfia hasítófüggvény).
- Legnépszerűbbek:
    - xxHash: https://xxhash.com/
    - Google FarmHash: https://github.com/google/farmhash
    - SMhasher (összehasonlító): https://github.com/rurban/smhasher
- Ha egy hasítófüggvény egyenletesen sorolja be a rekordokat, akkor nagyjából egyforma blokkláncok keletkeznek.
- Ilyenkor az egyes blokkláncok **B/K** blokkból állnak.

Keresés hasító táblában

- Egyenlőségi keresésnél (*A*=*a*) elég csak a *h(a)* által meghatározott kosárhoz tartozó blokkokat végignézni.
    - **B/K** darab blokk legrosszabb esetben, ideális hasító függvénnyel
- **Kérdés**: miért nem érdemes nagy K-t választani?
- Tárméret: B (ha minden blokk nagyjából tele van)
- **Fontos**: intervallumos keresésre (*a* < *A* < *b*) a hasító tábla nem jó.
    - Ha diszkrét értékek szerepelnek az intervallumban, akkor használatható (de nem biztos, hogy érdemes).
    - Folytonos értékeknél nem tudjuk használni.

Mi a baj a bemutatott módszerrel?

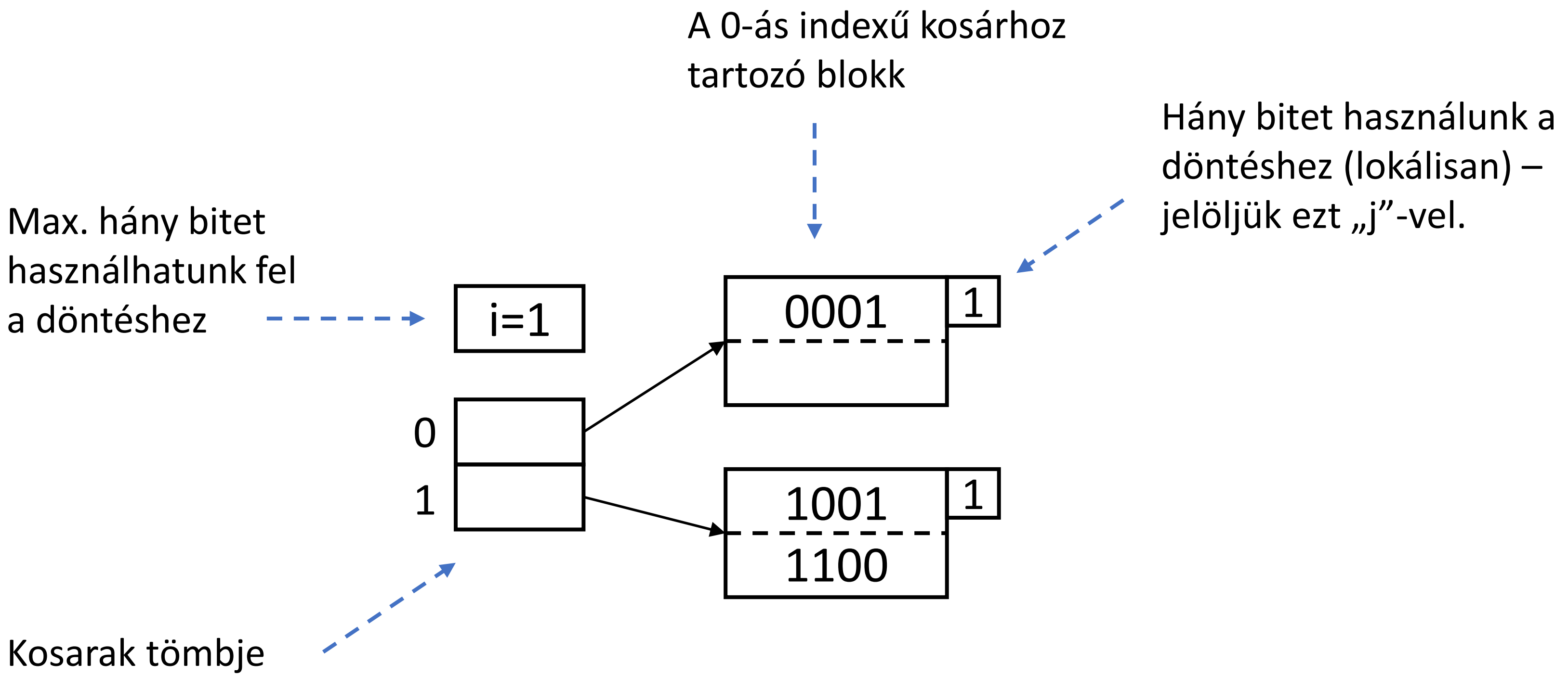
- A bemutatott hasító tábla **statikus** (a kosarak száma nem változik).
  - **Életszerűtlen** azt feltételezni, hogy előre tudjuk a rekordok számát.
  - Ha a rekordok száma nő, akkor egy idő után egy kosárhoz tartozó lánc **sok blokkot** fog tartalmazni és ezeket mind át kell nézni kereséskor.
- Amit megnéztünk: láncolt statikus hashelés (chained hashing)
- Vannak más statikus módszerek is (open addressing):
  - Linear probe hashing
  - Cuckoo hashing
  - Robin hood hasing
- Az „open addressing” hasító tábláknál nincsenek blokk láncok. Ha nincs hely egy kosárhoz tartozó blokkban, akkor máshová tesszük a rekordot.
- A statikus módszerek is használatban vannak (pl. hash join).

Kiterjeszthető hasító tábla (extendible hasing)

- Dinamikus, azaz a kosarak száma időben változhat (törléskor, beszúráskor).
- Minden kosár 1 blokkból fog állni – így a keresési költség mindig 1 lesz.
- Bevezetünk egy közvetett szintet, amely egy mutatókból álló tömb.
  - Ezek mutatnak a blokkokra.
  - A tömb mérete 2 valamilyen hatványa lesz (azaz duplázódva növekszik).
  - Nem minden kosárhoz tartozik külön blokk, bizonyos kosarak osztozhatnak egy blokkon.
- A *h* tördelőfüggvény egy *k* bitből álló kódot ad vissza.
  - Elvárjuk: $k > log_2(R_{max})$, ahol R_{max} a rekordok várható számának felső korlátja.
- A k hosszú kód elejéről maximum i bitet használunk annak az eldöntésére, hogy hová kerüljön egy rekord.

Kiterjeszthető hasító tábla példa

- Legyen *k=4*.
- Kiinduló állapot:

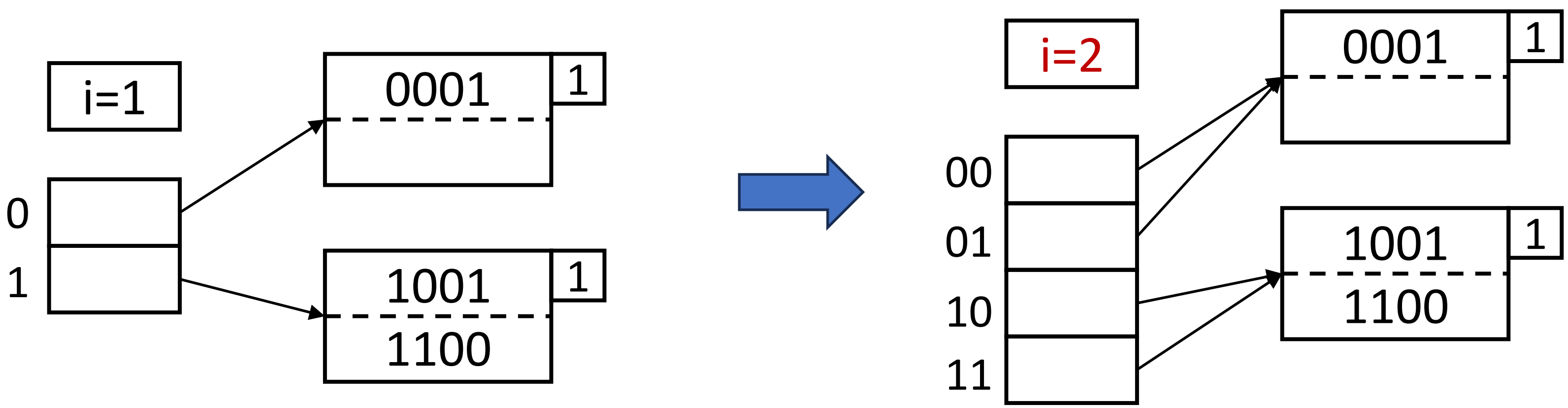


Beszúrás kiterjeszthető hasító táblába

- A hasítófüggvény által adott kód első i bitjével megegyező kosár mutatóját követjük. Ha van hely a blokkban, akkor beszúrjuk a rekordot.
- Ha nincs hely és j $< i$, akkor:
  - A blokkot ketté vágjuk, a rekordokat szétosztjuk a két blokk között a $(j + 1)$-edik bit értéke alapján.
  - Az új blokkok lokális bit számlálója $j + 1$ lesz.
  - A kosártömb mutatóit a megfelelő blokkokra állítjuk.
- Ha $j = i$, akkor először növeljük az i értékét egyel. Ez megduplázza a kosártömböt. Ha w egy i számú bitből álló kosár index volt, akkor az új kosártömbben a $w0$ és $w1$ kosárindexek ugyanarra a blokkra mutatnak (amelyikre a w mutatott). Ezután úgy járunk el, mint a j $< i$ esetben.

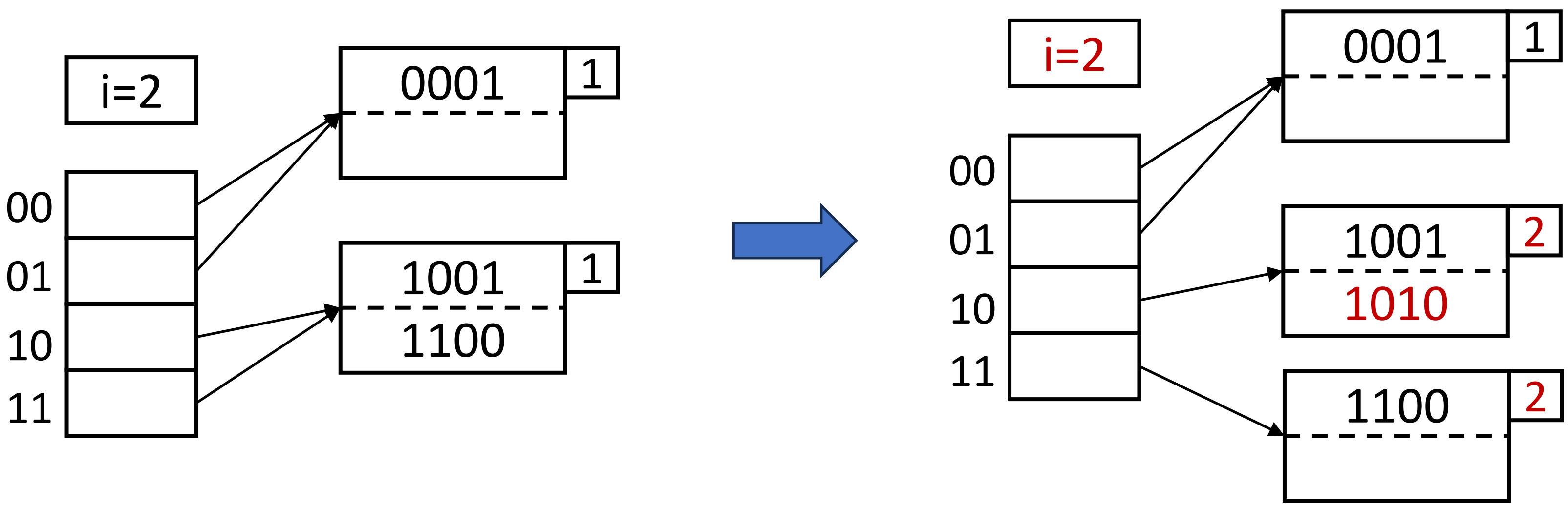
Beszúrás kiterjeszthető hasító táblába

- **Szúrjuk be az 1010 értéket** a bal oldali kiterjeszthető tördelő táblába.
- Az első i bitje az „1”, ezért a 1-es indexű kosár mutatóját követjük.
- A blokkban nincs hely. Mivel $j = i$ ezért meg kell növelnünk az i értékét egyel.



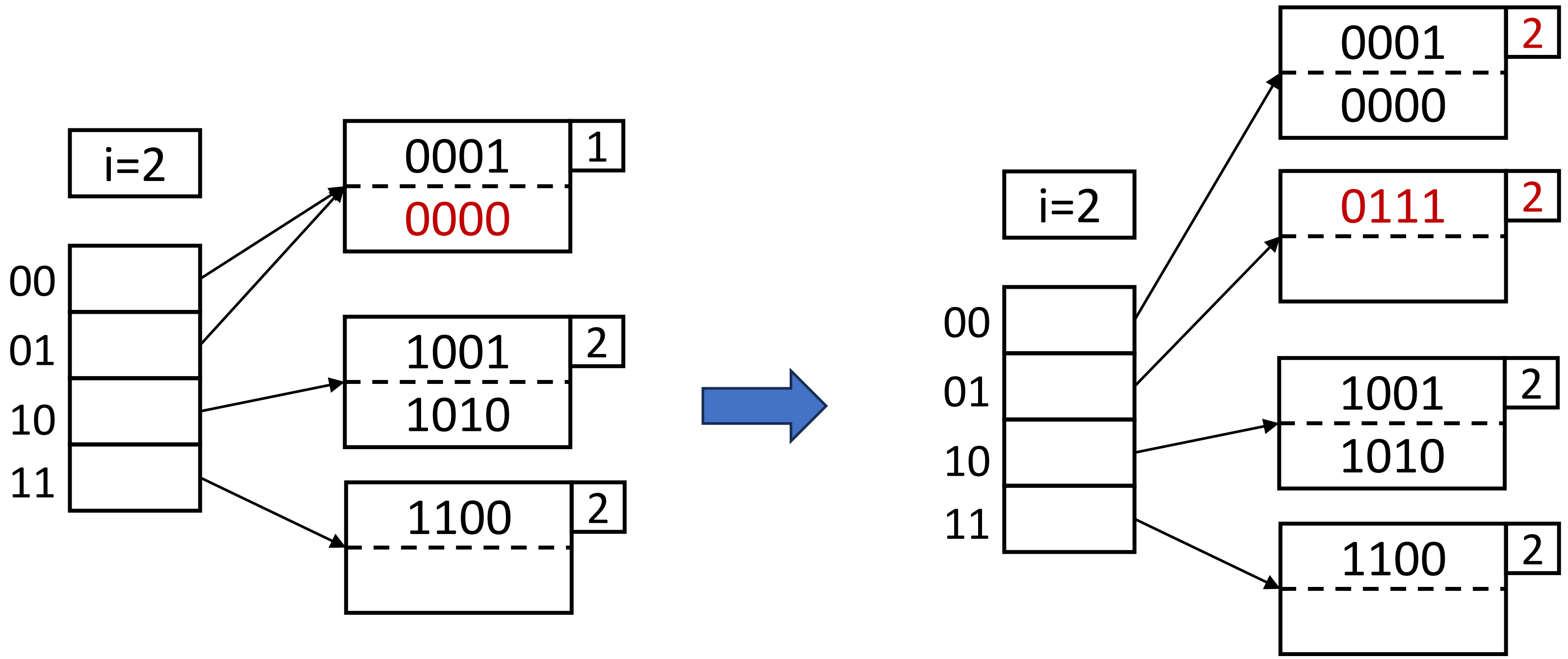
Beszúrás kiterjeszthető hasító táblába

- Ezután ketté vágjuk a megfelelő blokkot és szétosztjuk a rekordokat.
- A blokkok lokális bit számlálója (*j*) növekszik 1-el.
- Végül beszúrjuk az új rekordot is (**1010**).



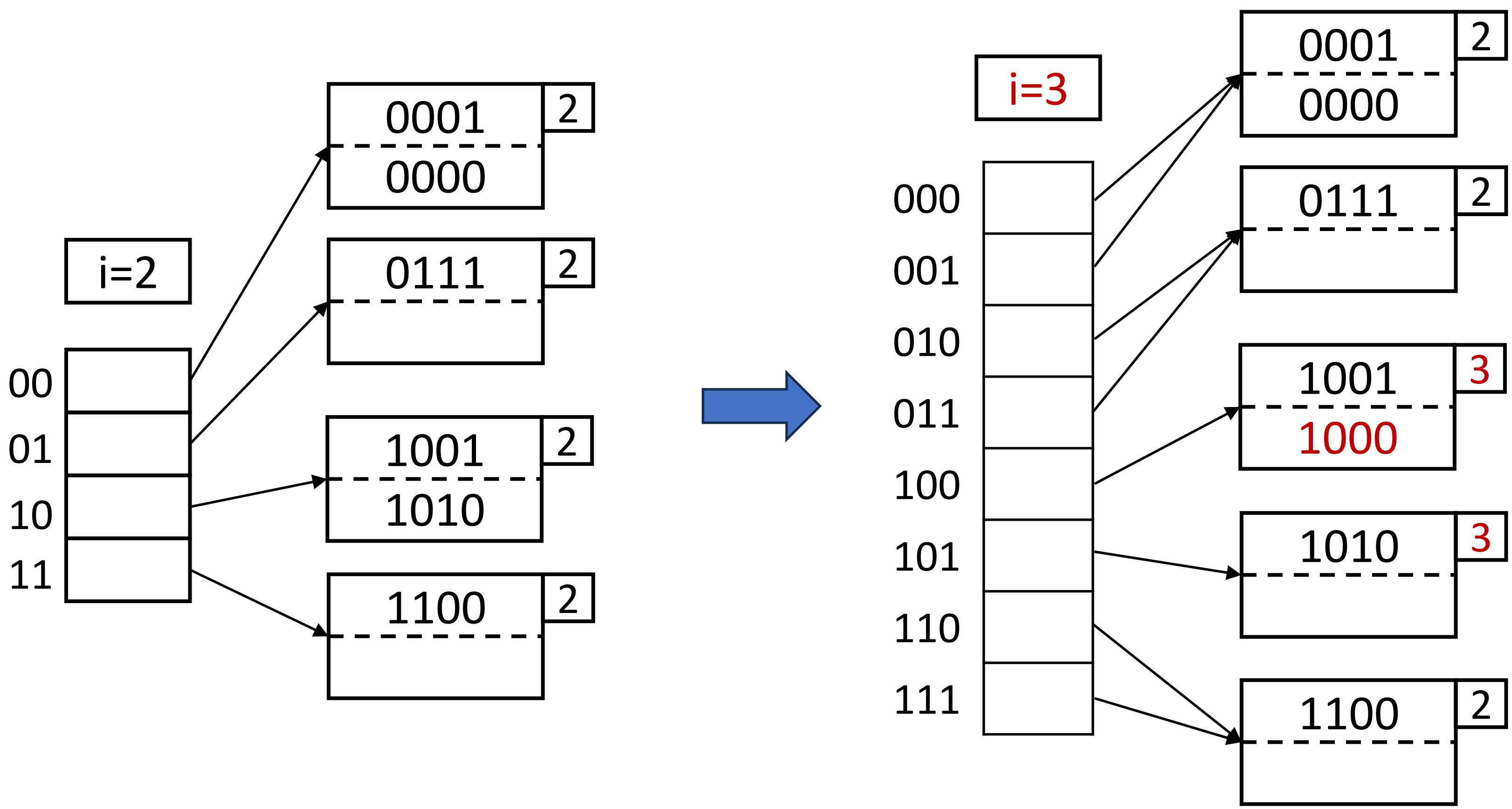
Beszúrás kiterjeszthető hasító táblába

- 0000 beszúrása:
- 0111 beszúrása:



Beszúrás kiterjeszthető hasító táblába

- Szúrjuk be az 1000 hasító értékű rekordot.
- Mivel $j = 1$, ezért növelnünk kell az i értékét is.



Törlés kiterjeszthető tördelő táblából

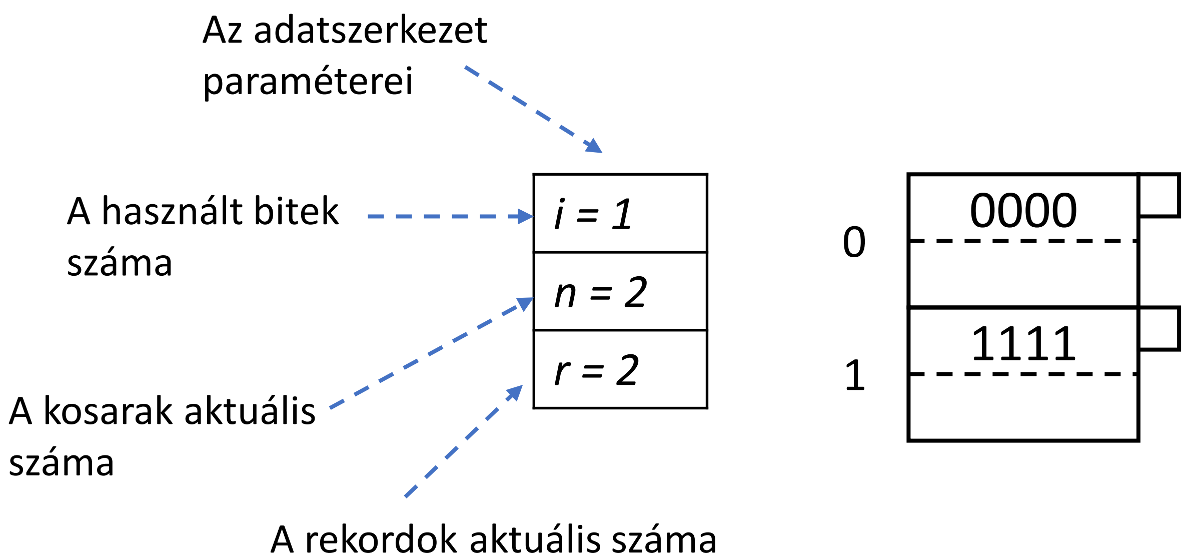
- **Blokkok összevonása**
  - Ha törlünk egy rekordot egy blokkból, akkor nézzük meg a blokk kosarának a „párját” (bucket buddy), azaz amelyiktől csak az utolsó bitben különbözik.
  - Ha a két kosárhoz tartozó blokkokban lévő rekordok elférnek egyetlen blokkban, akkor a két blokkot összevonhatjuk, a kosarakban lévő mutatókat pedig ennek megfelelően kell beállítanunk.
- **Kosártömb felezése**
  - Ha rekordokat törlünk, megvizsgálhatjuk a blokkok lokális bit számlálóit.
  - Ha minden blokk lokális bit számlálója kisebb, mint i, akkor felezhetjük a kosártömböt.
  - Ezt nem biztos, hogy megéri, hiszen ha ezután ismét meg kell növelnünk, akkor sok felesleget munkát végzünk el.

Kiterjeszthető hasító tábla hátrányai

- Ha nagy az i és meg kell dupláznunk a kosártömböt, akkor nagy munkát kell végezni (addig nem használhatjuk a hasító táblát).
- Ha a hasító függvény nem egyenletesen osztja el a rekordokat, akkor úgy is lehet nagy kosártömbünk, ha a rekordok száma viszonylag kicsi.
- A duplázás miatt a kosártömb nagyra nőhet (ki kell lapoznunk a memóriából).

Lineáris hasító tábla

- A kosarak számát egyesével növeljük (lineárisan).
- A kosarak száma úgy alakul ki, hogy a rekordok blokkonkénti átlagos száma a blokkot megtöltő rekordoknak egy állandó hányadát képezze (pl. 80%).
- A túlcsordulás blokkok megengedettek.
- A kosártömb indexelésére használt bitek száma $\lceil \log_2 n \rceil$. Ezeket a bitsorozat végéről vesszük (alacsony prioritás).

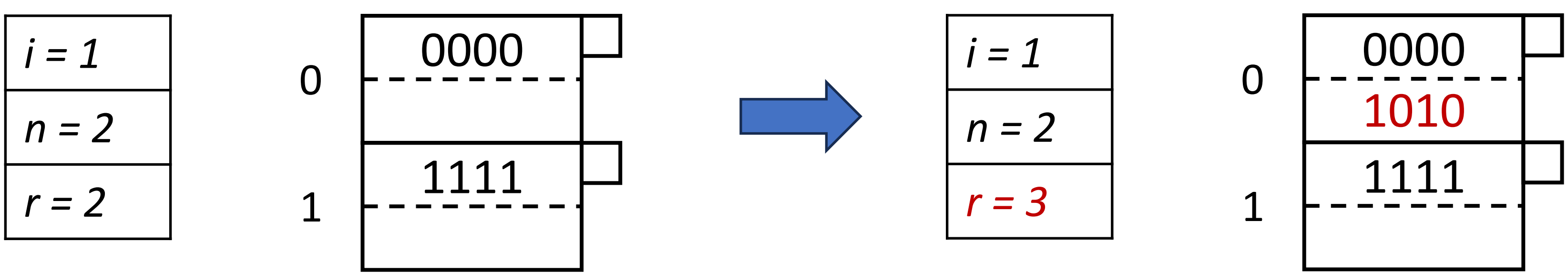


Beszúrás lineáris hasító táblába

- Minden egyes beszúrásnál összehasonlítjuk a rekordok aktuális számát az r/n hányados felső határával. Ha átlépjük a határt új kosarat kell hozzáadni.
- Jelöljük az új beszúrandó érték utolsó i bitjét $a_1 a_2 \dots a_n$-el. Tekintsük a bitsorozatot decimális számként és jelöljük m-el.
  - Ha $m < n$, akkor az m indexű kosár létezik és a rekordot elhelyezhetjük a hozzá tartozó blokkban.
  - Ha $n \leq m < 2^i$, akkor az m indexű kosár még nem létezik, így a rekordot az $m - 2^{i-1}$ indexű kosárba helyezzük el (ami igazából az a kosár lesz, mintha az a_1-et kicserélnénk 0-ra).
- Ha a kosárban nincs szabad hely, akkor készítünk egy túlcsordulás blokkot.

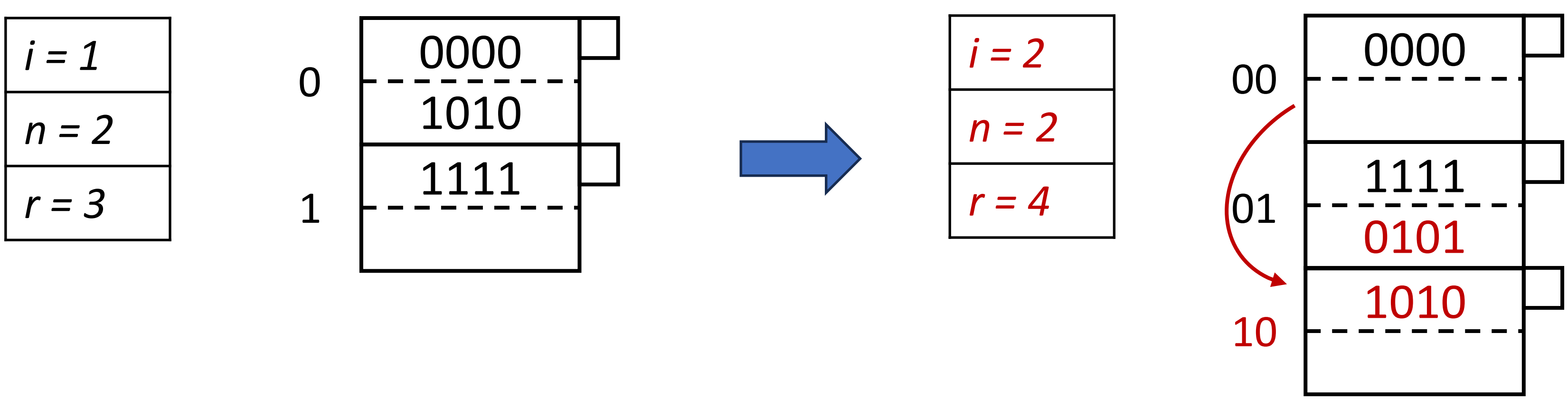
Beszúrás lineáris hasító táblába

- Legyen az r/n hányados határa most 1,7, azaz legfeljebb $1{,}7 * n$ rekordunk lehet.
- Szúrjuk be a **1010** hasító értéket -> nem lépjük át a határt.
- Az utolsó bit alapján a 0-s indexű kosárba kell betennünk a rekordot.
  - Mivel van benne hely, ezért egyszerűen beszúrjuk.



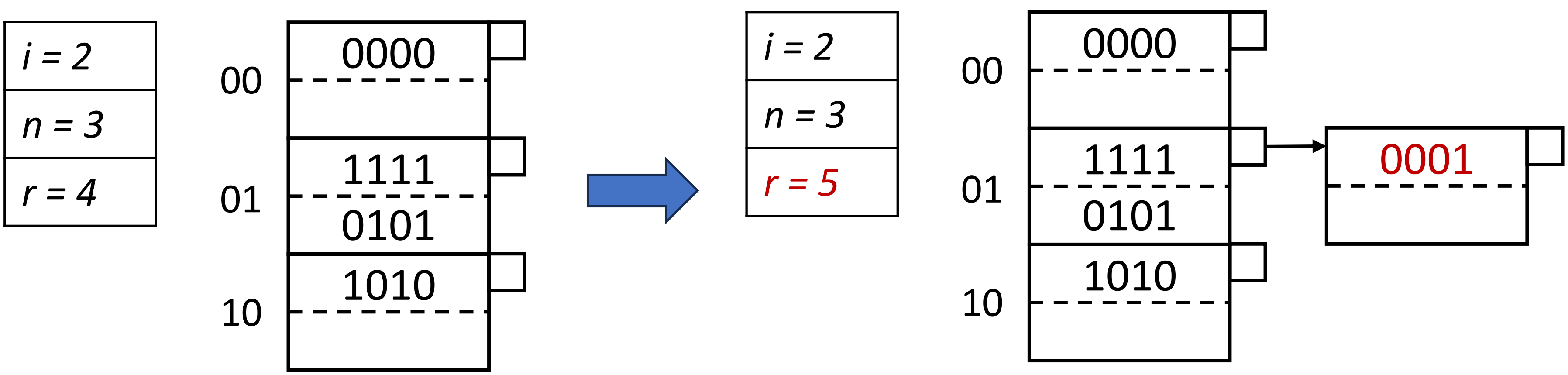
Beszúrás lineáris hasító táblába

- Szúrjuk be a **0101** hasító értéket -> 4/2 átlépi a határt ezért új kosár kell.
- 1 bittel nem lehet több kosarat indexelni, ezért az i értékét is meg kell növelnünk.
    - Ezután az utolsó $i + 1$ bitet fogjuk a döntéshez használni.
- A már létező indexek elé csak beírunk egy 0-t és létrehozzuk a soron következő indexű kosarat (10).
- Ha az új kosár indexe $1a_2 \dots a_n$, akkor a $0a_2 \dots a_n$ indexű kosárban lévő rekordokat szétosztjuk a régi és az új kosár között.



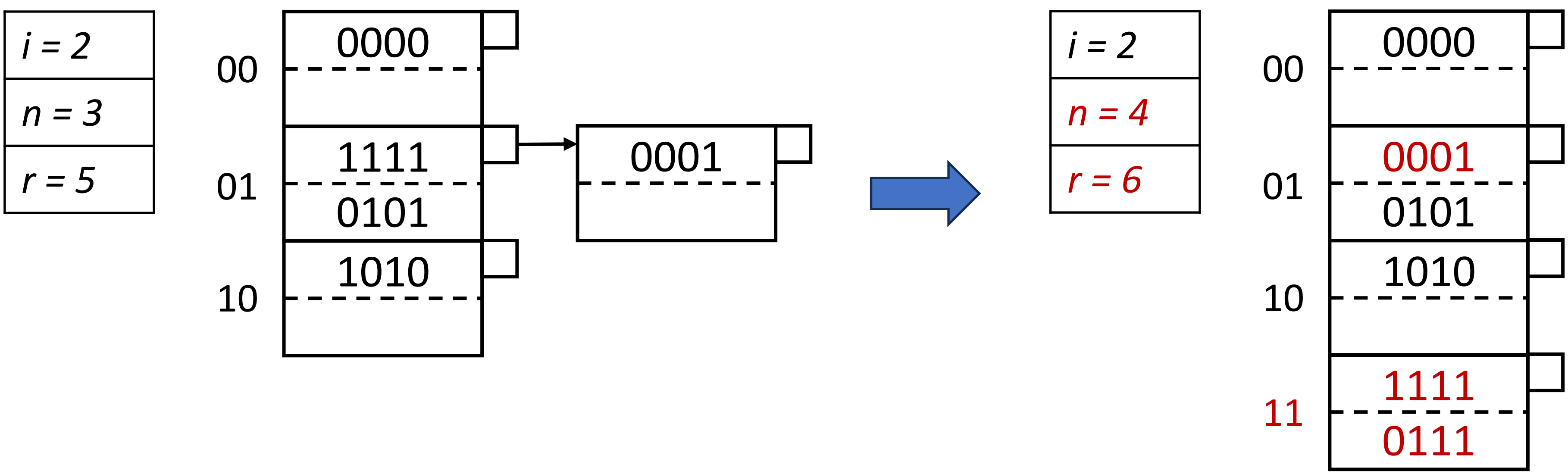
Beszúrás lineáris hasító táblába

- Szúrjuk be a **0001** hasító értéket -> 5/3 nem lépi át a határt, nem kell új kosár.
- Ehhez a kosárhoz (01) tartozó blokk megtelt, ezért új blokkot láncolunk hozzá.
  - Mivel nem létük át a határt, ezért NEM hozhatunk létre új kosarat.
- Vegyük észre, hogy nem minden érték van „jó helyen”. Az 1111-es értéknek az 11 indexű kosárban kellene lennie, azonban ilyen kosár még nincs.



Beszúrás lineáris hasító táblába

- Szúrjuk be a **0111** hasító értéket -> 6/3 nem lépi át a határt, új kosár kell (11).
- Az 1111 átkerült a „helyére”, az 11 indexű kosárba.
- A túlcsordulás blokkot megszüntethetjük, a 0001 átkerült a megfelelő blokkba.
- Az új érték bekerült az 11 indexű kosárhoz tartozó blokkba



Keresés és törlés lineáris hasító táblában

- A keresés megegyezik azzal az eljárással, amellyel kiválasztjuk azt a kosarat, amelybe a beszúrni kívánt rekord kerülni fog.
- Törlés
  - Ha törlünk egy rekordot egy olyan kosárból, ahol van túlcsordulás blokk, akkor lehetőség szerint megszüntetjük a túlcsordulás blokkot.
  - Kosárt csak akkor szüntetünk meg, ha az r/n hányados egy határ alá esik. Ilyenkor a legnagyobb indexű kosarat szüntetjük meg, a benne lévő rekordok más kosárba kerülnek. Ezt nem minden törléskor ellenőrizzük, inkább csak valamilyen időközönként.

Lineáris hasító tábla hátrányai

- Alkalmaz túlcsordulás blokkokat, így a keresés költsége nem mindig 1.
- Több rekord beszúrásakor sok egymás utáni kosár létrehozásra lehet szükség.

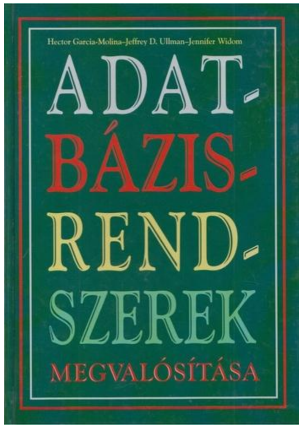
Megjegyzések

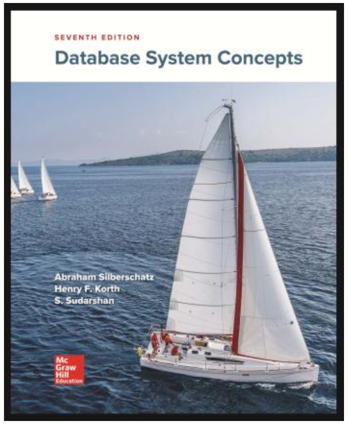
- Mire használhatjuk a hasító táblákat az adatbázisban?
  - Indexstruktúraként
  - Belső metaadatok tárolására (pl. laptábla)
  - Temporális (ideiglenes) adatstruktúraként (pl. hash join)
- **Még egyszer**: intervallumos keresésre ($a < A < b$) a hasító tábla nem jó.
- Ezért az esetek nagy részében **NEM hasító táblát** használunk indexstruktúraként.
- Oracle-ben nem is lehet hasító tábla indexet létrehozni önmagában (csak klaszteren tárolt tábla esetén) – MySQL, PostgreSQL lehetőséget ad rá.

Tankönyv fejezetek

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:

  Adatbázisrendszerek megvalósítása
  - 4.4 fejezet: Tördelőtáblázatok (200-211. oldalak)
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 14. Indexing: 14.5 Hash Indices
  - Chapter 24. Advaned Indexing Techniques: 24.5 Hash Indices





ELTE | IK

Következő előadás

- B+ fa – a leggyakrabban használt indexstruktúra a relációs rendszerekben.