Indexelés

- **Célok:**
  - **gyors lekérdezés,**
  - **gyors adatmódosítás,**
  - **minél kisebb tárolási terület.**
- Nincs általánosan legjobb optimalizáció. Az egyik cél a másik rovására javítható (például indexek használatával csökken a keresési idő, nő a tárméret, és nő a módosítási idő).
- Az adatbázis-alkalmazások alapján az adatbázis lehet:
  - **statikus (**ritkán módosul, a lekérdezések gyorsasága a fontosabb),
  - **dinamikus** (gyakran módosul, ritkán végzünk lekérdezést).
- Hogyan mérjük a költségeket?
- Memória műveletek nagyságrenddel gyorsabbak, mint a háttértárolóról beolvasás, kiírás.
- Az író-olvasó fej nagyobb adategységeket (**blokkokat**) olvas be.
- A blokkméret függhet az operációs rendszertől, hardvertől, adatbázis-kezelőtől.
- A blokkméretet fixnek tekintjük. Oracle esetén 8K az alapértelmezés.
- **Feltételezzük, hogy a beolvasás, kiírás költsége arányos a háttértároló és memória között mozgatott blokkok számával.**

Indexelés

- Célszerű a fájlokat blokkokba szervezni.
- A fájl rekordokból áll.
- A **rekordok szerkezete** eltérő is lehet.
- A rekord tartalmaz:
  - **leíró fejlécet** (rekordstruktúra leírása, belső/külső mutatók, (hol kezdődik egy mező, melyek a kitöltetlen mezők, melyik a következő rekord, melyik az előző rekord), törlési bit, statisztikák),
  - **mezőket**, melyek üresek, vagy adatot tartalmaznak.
- A rekordhossz lehet:
  - **állandó,**
  - **változó** (változó hosszú mezők, ismétlődő mezők miatt).
- Az egyszerűség kedvéért feltesszük, hogy állandó hosszú rekordokból áll a fájl, melyek hossza az átlagos rekordméretnek felel.
- A **blokkok** tartalmaznak:
  - **leíró fejlécet** (rekordok száma, struktúrája, fájlok leírása, belső/külső mutatók (hol kezdődik a rekord, hol vannak üres helyek, melyik a következő blokk, melyik az előző blokk, statisztikák (melyik fájlból hány rekord szerepel a blokkban)),
  - **rekordokat** (egy vagy több fájlból),
  - **üres helyeket**.

Indexelés

- A költségek méréséhez paramétereket vezetünk be:
- **l** - (length) **rekordméret** (bájtokban)
- **b** - **blokkméret** (bájtokban)
- **T** - (tuple) **rekordok száma**
- **B** - a **fájl mérete blokkokban**
- **bf** – **blokkolási faktor** (mennyi rekord fér el egy blokkban: bf = ⌊b/l⌋ - alsó egészrész)
- **B=⌈T/bf⌉**
- **M** – **memória mérete blokkokban**
- Például **R×S mérete** mekkora:
  - **l(R×S) = l(R)+l(S)**
  - **T(R×S) = T(R)*T(S)**
  - **bf(R×S) = b / (l(R)+l(S))**
  - **B(R×S) = (T(R)*T(S)) * (l(R)+l(S)) / b**
    = (T(S)*T(R)*l(R)/b) + (T(R)*T(S)*l(S)/b)=
    =T(S)*B(R) + T(R)*B(S)

Indexelés

- **Milyen lekérdezéseket vizsgáljunk?**
- A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni.
- A legegyszerűbb kiválasztás:
  - **A=a** (A egy keresési mező, a egy konstans)
- Kétféle **bonyolultság**ot szokás vizsgálni:
  - **átlagos**,
  - **legrosszabb** eset.
- Az esetek vizsgálatánál az is számít, hogy az **A=a** feltételnek megfelelő rekordokból lehet-e több, vagy biztos, hogy csak egy lehet.
- Fel szoktuk tenni, hogy az **A=a** feltételnek eleget tevő rekordokból nagyjából egyforma számú rekord szerepel. (Ez az **egyenletességi feltétel.)**

Indexelés

- Az A oszlopban szereplő különböző értékek számát **képméret**nek hívjuk és **I(A)**-val jelöljük.
- **$I(A) = |\Pi_A(R)|$**
- Egyenletességi feltétel esetén:
  - **$T(\sigma_{A=a}(R)) = T(R) / I(A)$**
  - **$B(\sigma_{A=a}(R)) = B(R) / I(A)$**
- A következő fájlszervezési módszereket fogjuk megvizsgálni:
  - **kupac (heap)**
  - **hasító index (hash)**
  - **rendezett állomány**
  - **elsődleges index (ritka index)**
  - **másodlagos index (sűrű index)**
  - **többszintű index**
  - **B^+-fa, B^*-fa**
- Azt az esetet vizsgáljuk, mikor az A=a feltételű rekordok közül elég az elsőt megkeresni.
- **Módosítási műveletek:**
  - **beszúrás (insert)**
  - **frissítés (update)**
  - **törlés (delete)**
- Az egyszerűsített esetben nem foglalkozunk azzal, hogy a beolvasott rekordokat bent lehet tartani a memóriában, későbbi keresések céljára.

Indexelés

- **Kupac szervezés:**
  - a rekordokat a blokk első üres helyre tesszük a beérkezés sorrendjében.
- **Tárméret**: **B**
- A=a **keresési idő:**
  - B (a legrosszabb esetben),
  - B/2 (átlagos esetben egyenletességi feltétel esetén).
- **Beszúrás:**
  - utolsó blokkba tesszük a rekordot, 1 olvasás + 1 írás
  - módosítás: 1 keresés + 1 írás
  - törlés: 1 keresés + 1 írás (üres hely marad, vagy a törlési bitet állítják át)

Indexelés

- **Hasítóindex-szervezés** (Hashelés):
  - a rekordokat **blokkláncokba** (**bucket – kosár**) soroljuk és a blokklánc utolsó blokkjának első üres helyére tesszük a rekordot a beérkezés sorrendjében.
  - a blokkláncok száma
    - előre adott: K (**statikus hasítás**)
    - a tárolt adatok alapján változhat (**dinamikus hasítás**)
- A besorolás az indexmező értékei alapján történik.
- Egy **h(x)∈{1,...,K} hasító függvény** értéke mondja meg, hogy melyik kosárba tartozik a rekord, ha x volt az indexmező értéke a rekordban.
- A hasító függvény általában maradékos osztáson alapul. Például **mod(K).**
- Akkor **jó egy hasító függvény**, ha nagyjából egyforma hosszú blokkláncok keletkeznek, azaz egyenletesen sorolja be a rekordokat.
- Jó hasító függvény esetén **a blokklánc B/K blokkból áll**.

Indexelés

- **Keresés** (**A=a**)
  - – ha az indexmező és keresési mező eltér, akkor kupac szervezést jelent,
  - – ha az indexmező és keresési mező megegyezik, akkor csak elég a **h(a)** sorszámú kosarat végignézni, amely **B/K blokkból** álló kupacnak felel meg, azaz **B/K** legrosszabb esetben. **A keresés K-szorosára gyorsul.**
- **Módosítás**: B/K blokkból álló kupac szervezésű kosarat kell módosítani.
- **Intervallumos (a < A < b) típusú keresésre nem jó.**

Indexelés

Tegyük fel, hogy 1 blokkba 2 rekord fér el.

Szúrjuk be a következő hasító értékkel rendelkező rekordokat!

INSERT:

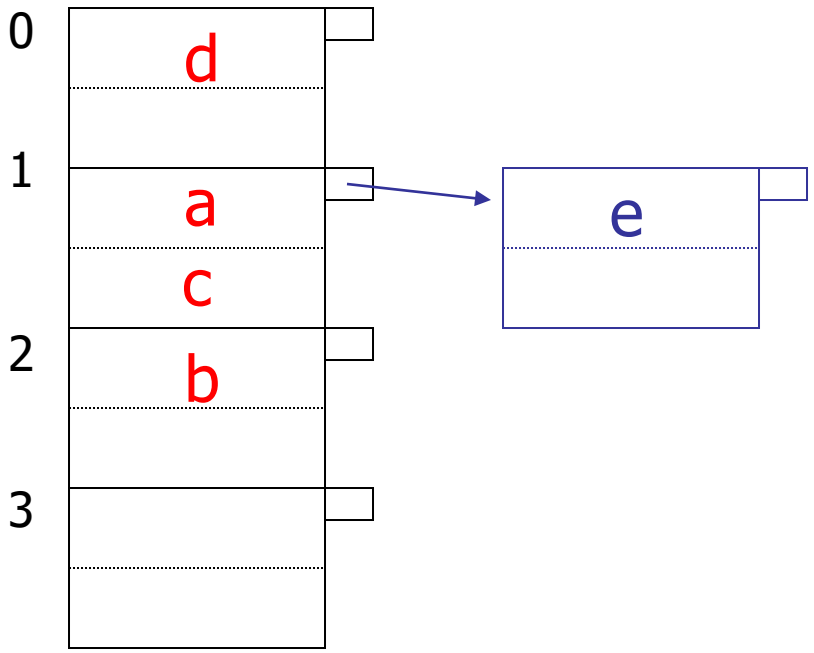
h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0

h(e) = 1



Indexelés

Töröljük a következő hasító értékkel rendelkező rekordokat!

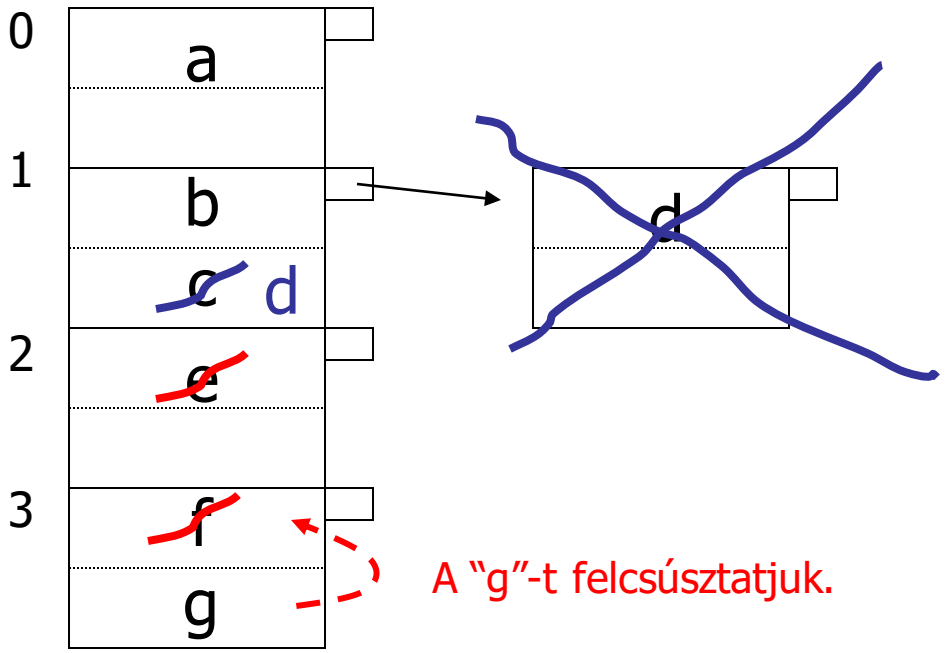
(A megüresedett túlcsordulási blokkokat megszüntetjük.)

Delete:

e

f

c



Indexelés

- **Dinamikus hasító indexek:**
  - **kiterjeszthető (expandable)**
  - **lineáris**

• Előre nem rögzítjük a kosarak számát, a kosarak száma beszúráskor, törléskor változhat.

•Itt általában majd 1 (vagy kicsivel több, mint 1) blokk tartozik egy kosárhoz.

• **Kiterjeszthető hasító index:**

•Minden kosárhoz legfeljebb 1 blokk tartozik. Keresési költség: 1.

•A kosarakat egy mutatókból álló tömb reprezentálja.

• E tömb elemszáma mindig 2-nek valamilyen hatványa.

• Bizonyos kosarak osztozhatnak egy-egy blokkon.

• Legyen **k > log (a kosarak várható számának felső korlátja).**

• **A h hasító függvény értéke egy k hosszú bináris sorozat.**

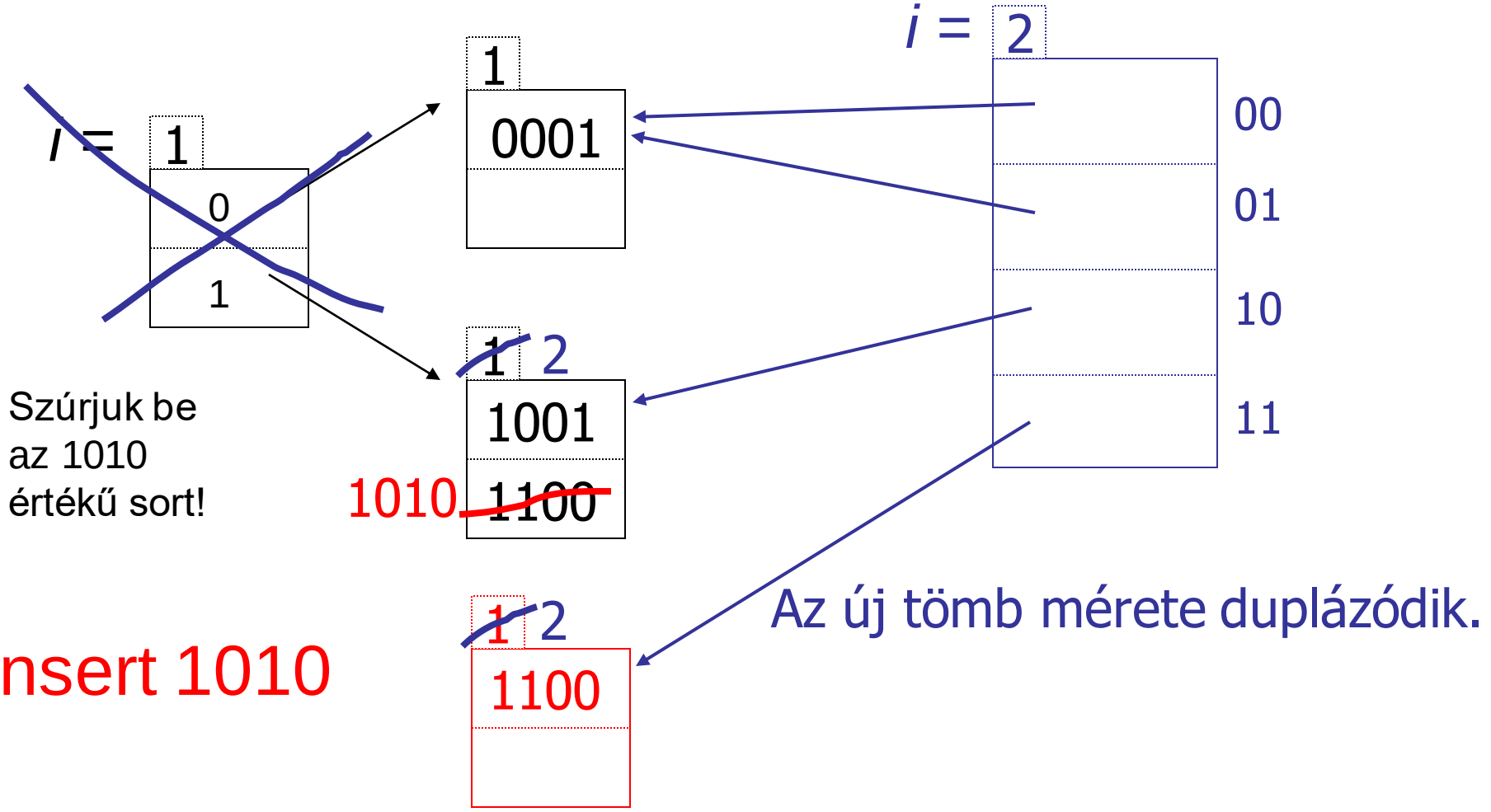
• A kosarakhoz rendelt kód **prefix kód**. A maximális prefix kód hossza legyen i. Ez azt is jelenti, hogy jelen pillanatban 2^i db kosarunk van.

Indexelés

- Beszúrás:
- A h(K) k hosszú kódnak vegyük *az **i hosszú elejét***, és **azt kosarat, amelynek kódja a h(K) kezdő szelete (*j*)**.
- Ha van hely a kosárban, tegyük bele a rekordot.
- Ha nincs, két eset lehetséges. Jelölje *j* a szóban forgó *kosár prefix* kódjának hosszát.
  - Ha *j* < *i,* a kosártömb nem változik, a szóba forgó kosárhoz tartozó blokkot kettévágjuk, a blokkba tartozó rekordokat szétosztjuk a (*j* + *1*). bit alapján. Amelyik rekordoknál itt 0 az érték bit (a kulcs (*j* + *1*)-dik bitje) maradnak, a többiek az új blokkba kerülnek. A régi és az új blokk prefix kódjának hossza *j* + *1* lesz.
  - Ha *j*= *i (*A maximális prefix kód hossza *i*), akkor megduplázzuk a kosártömb elemeinek a számát és *i* értékét 1-gyel növeljük. Ha *w* egy előző kosár prefix kódja volt a tömbben, most a *w0, w1* prefix kódú kosarak osztoznak az előbbi kosárhoz tartozó blokkon. Ezek után beszúrjuk az iménti rekordot (itt *j* már kisebb, mint az új *i*).

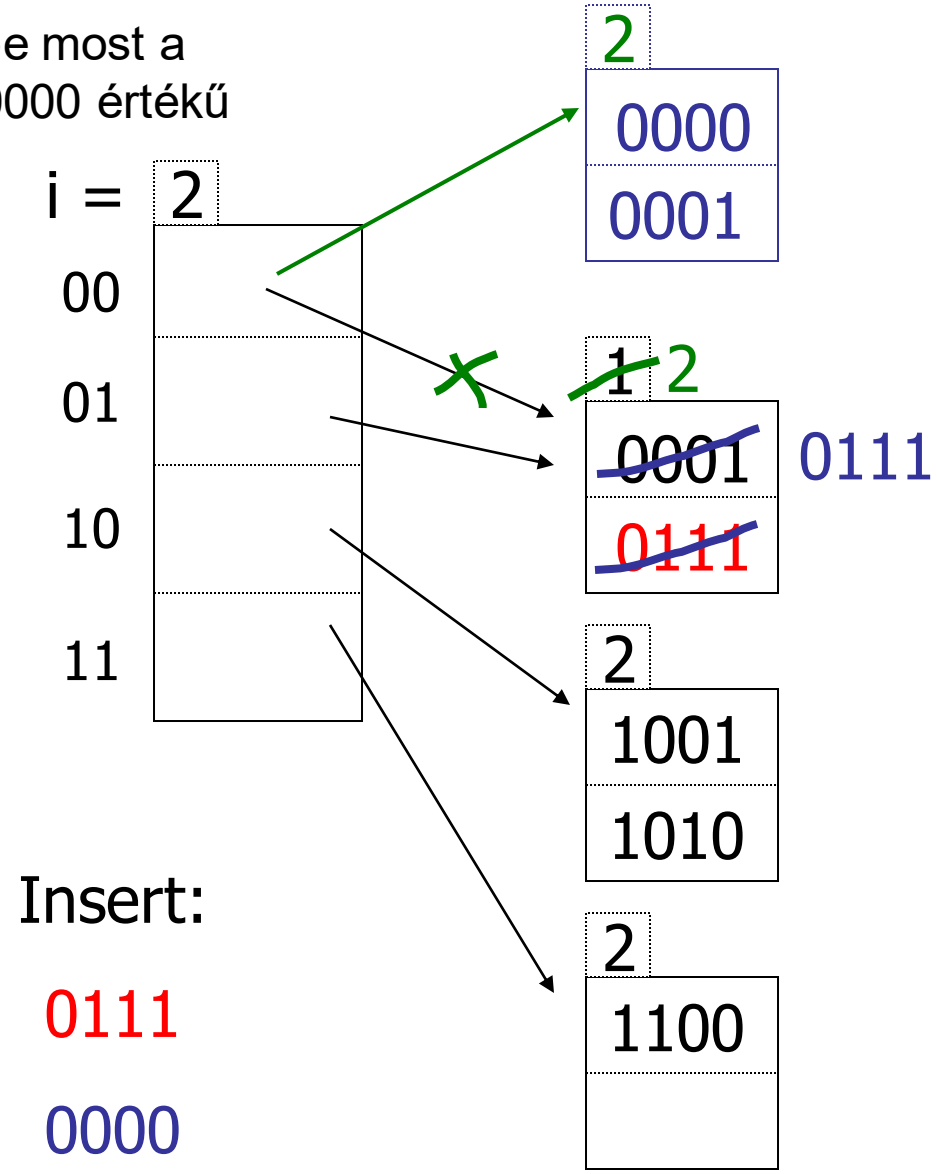
Indexelés

Legyen például *h(k)* 4 bites és 2 rekord férjen egy blokkba. Az *i* jelzi, hogy hány bitet használunk fel.



Indexelés

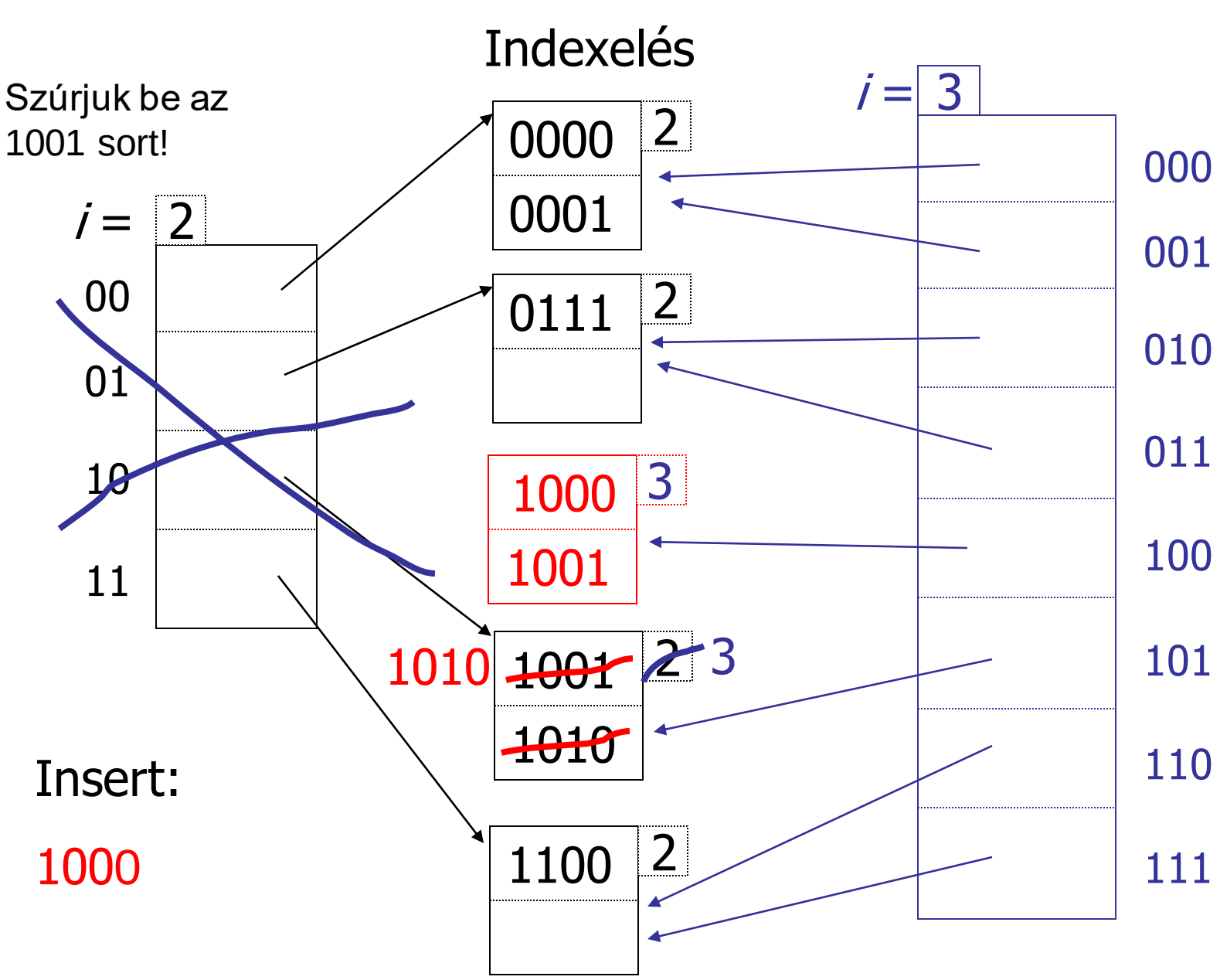
Szúrjuk be most a 0111 és 0000 értékű sorokat!



Insert:

0111

0000


Indexelés
Szúrjuk be az
1001 sort!
i = 2
00
01
10
11
0000
0001
2
0111
2
1000
1001
3
1010
1001
1010
2
3
1100
2
Insert:
1000
i = 3
000
001
010
011
100
101
110
111

Indexelés

- **Lineáris hasító index:**
- A kosarak 1 vagy több blokkból is állhatnak.
- Új kosarat akkor nyitunk meg, ha egy előre megadott értéket elér a kosarakra jutó átlagos rekordszám.

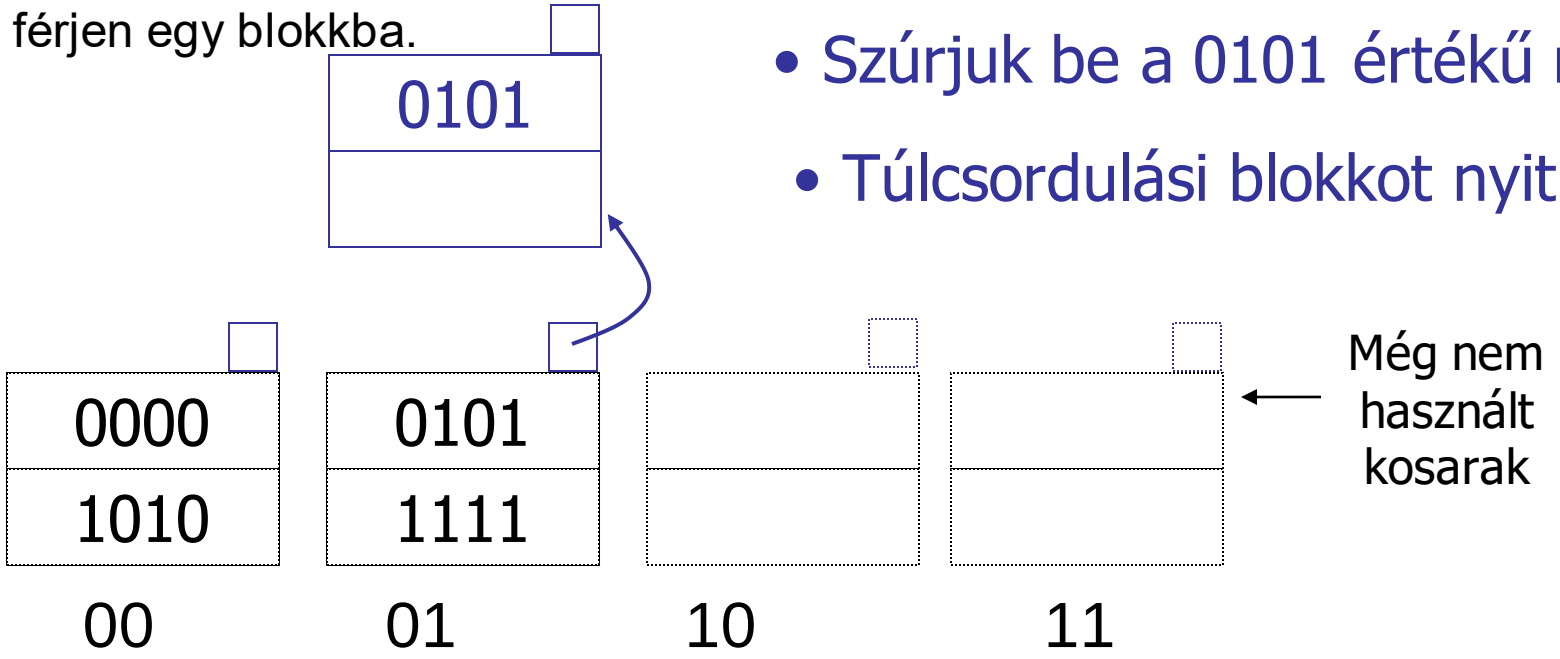
  (rekordok száma/kosarak száma > küszöb pl. 80%)

  (rekordok száma/blokkok száma > küszöb pl. 80%)
- **A kosarakat 0-tól kezdve sorszámozzuk, és a sorszámot binárisan ábrázoljuk.**
- Ha n kosarunk van, akkor a hasító függvény értékének **utolsó ⌈ log(n) ⌉ bit**jével megegyező sorszámú kosárba tesszük, ha van benn hely. Ha nincs, akkor hozzáláncolunk egy új blokkot és abba tesszük.
- Ha nincs megfelelő sorszámú kosár, akkor abba a sorszámú kosárba tesszük, amely csak az első bitjében különbözik a keresett sorszámtól.

Indexelés

Legyen a hasító érték 4 bites, i =2, és 2 rekord férjen egy blokkba.

- Szúrjuk be a 0101 értékű rekordot!
- Túlcsordulási blokkot nyitunk.



m = 01 (a legnagyobb sorszámú blokk sorszáma)

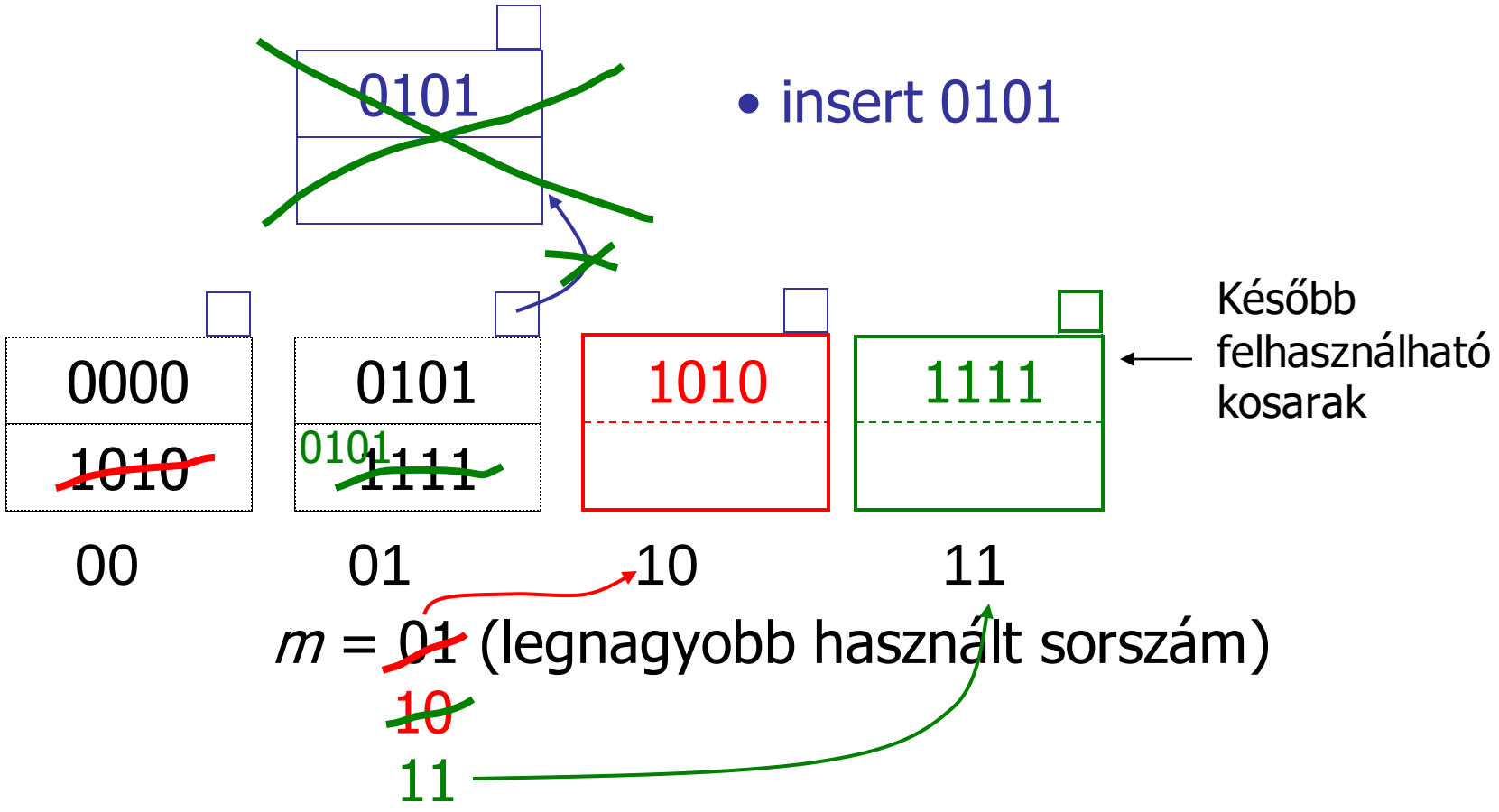
Szabály: Ha **h(K)[*i*] ≤ *m***, akkor a rekordot tegyük a **h(K)[*i*]** kosárba,

ha ***m*** **≤ h(K)[*i*] ≤ 2^i**

pedig tegyük a **h(K)[*i*] - 2^{i-1}** kosárba!

Megjegyzés: **h(K)[i]** és **h(K)[i] - 2^{i-1}** csak az első bitben különbözik!

Indexelés

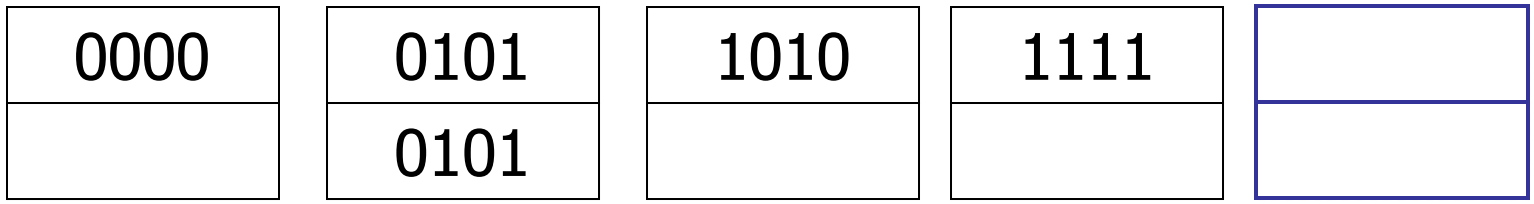


Tegyük fel, hogy átléptük a küszöbszámot, és ezért új kosarat kell nyitni, majd az első bitben különböző sorszámú kosárból át kell tenni ebbe az egyező végződésű rekordokat.

Indexelés

Ha i bitet használunk és 2^i kosarunk van, akkor a következő kosárnyitás előtt i-t megnöveljük 1-gyel, és az első bitben különböző sorszámú kosárból áttöltjük a szükséges rekordokat és így tovább.

$i = \cancel{2}\ 3$



000 001 010 011 100

~~100~~

. . .

$m = \cancel{11}$ (max used block)

100

Indexelés

- **Rendezett állomány**
- A tárolás 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 **kereső mező** nem esik egybe, akkor kupac szervezést jelent.
- Ha a rendező mező és kereső mező egybeesik, akkor **bináris (logaritmikus) keresés**t lehet alkalmazni:
  - **beolvassuk a középső blokkot,**
  - ha nincs benne az A=a értékű rekord, akkor eldöntjük, hogy a blokklánc második felében, vagy az első felében szerepelhet-e egyáltalán,
  - beolvassuk a felezett blokklánc középső blokkját,
  - addig folytatjuk, amíg megtaláljuk a rekordot, vagy a vizsgálandó maradék blokklánc már csak 1 blokkból áll.
- **Keresési idő**: **$\log_2(B)$**

Indexelés

- **Beszúrás:**
  - keresés + üres hely készítés miatt a rekordok eltolása az összes blokkban, az adott találati blokktól kezdve (B/2 blokkot be kell olvasni, majd az eltolások után visszaírni=B művelet)
- **Szokásos megoldások:**
- **Gyűjtő (túlcsordulási) blokk** használata:
  - az új rekordok számára nyitunk egy blokkot, ha betelik hozzáláncolunk egy újabb blokkokat,
  - keresést 2 helyen végezzük: **$\log_2(B-G)$** költséggel keresünk a rendezett részben, és ha nem találjuk, akkor a gyűjtőben is megnézzük (**G** blokkművelet, ahol **G a gyűjtő mérete**), azaz az összköltség: **$\log_2(B-G)+G$**
  - ha a G túl nagy a $\log_2(B)$ – hez képest, akkor újrarendezzük a teljes fájlt (a rendezés költsége **$B*\log_2(B)$**).

Indexelés

- **Üres helyeket** hagyunk a blokkokban:
  - például félig üresek a blokkok:
  - a keresés után 1 blokkművelettel visszaírjuk a blokkot, amibe beírtuk az új rekordot,
  - tárméret **2*B** lesz
  - 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:**
  - keresés + a törlés elvégzése, vagy a törlési bit beállítása után visszaírás (1 blokkírás)
  - túl sok törlés után újraszervezés
- **Frissítés: törlés + beszúrás**

Indexelés

- **Indexek használata:**
  - keresést gyorsító segédstruktúra
  - több mezőre is lehet indexet készíteni
  - az index tárolása növeli a tárméretet
  - **nem csak a főfájlt, hanem az indexet is karban kell tartani, ami plusz költséget jelent**
  - ha a keresési mező egyik indexmezővel sem esik egybe, akkor kupac szervezést jelent
- **Az indexrekordok szerkezete:**
  - **(a,p)**, ahol a egy érték az indexelt oszlopban, p egy **blokkmutató**, arra a blokkra mutat, amelyben az **A=a** értékű rekordot tároljuk.
  - az **index mindig rendezett** az indexértékek szerint
  - Oracle SQL-ben:
  - **egyszerű index:**
    - **CREATE INDEX supplier_idx**
      ON supplier (supplier_name);
  - **összetett index:**
    - **CREATE INDEX supplier_idx**
      ON supplier (supplier_name, city)
      COMPUTE STATISTICS;
    - -- az optimalizáláshoz szükséges statisztikák elkészítésével --

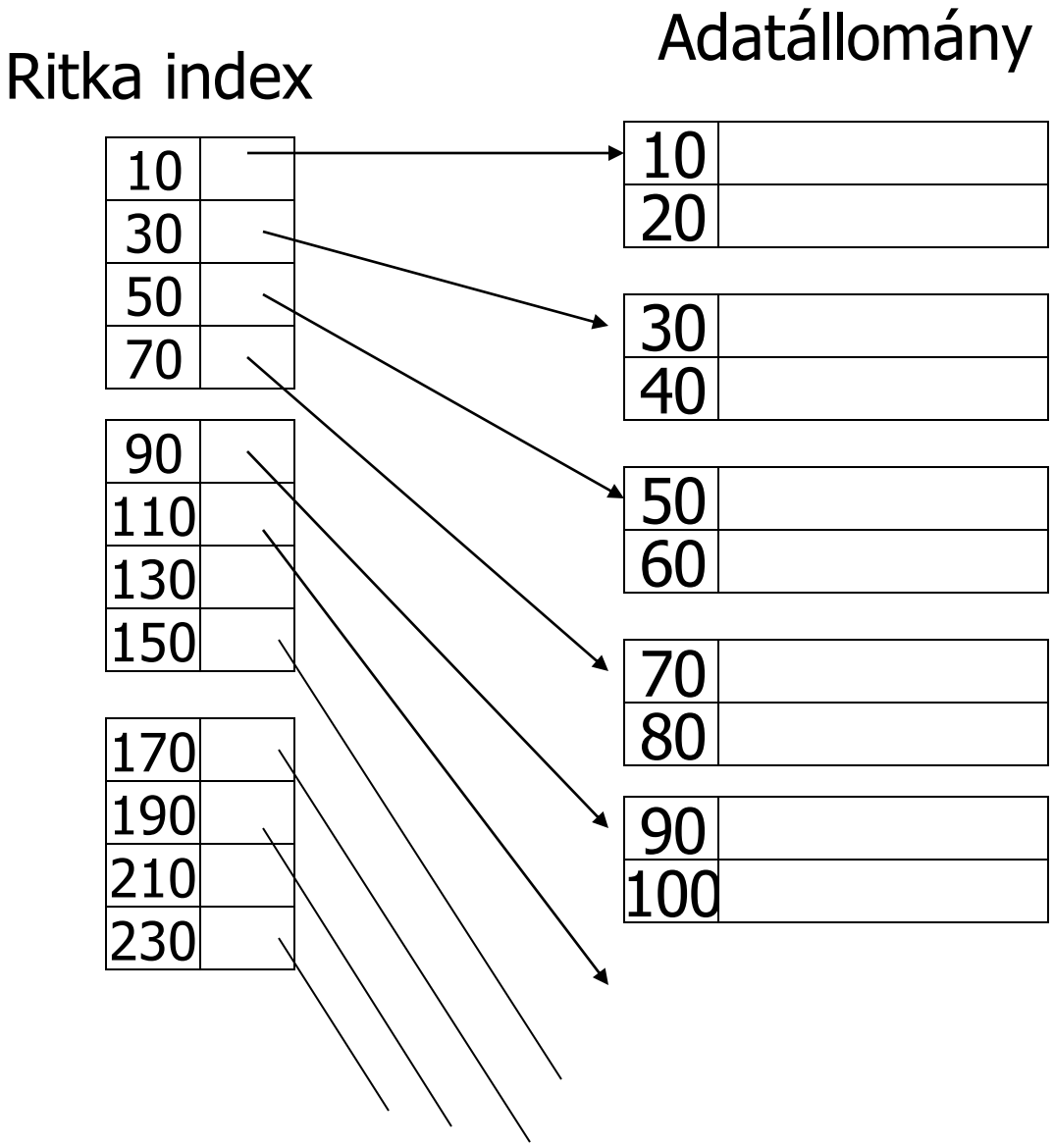
Indexelés

- **Elsődleges index:**
  - **főfájl is rendezett**
  - csak 1 elsődleges indexet lehet megadni (mert csak egyik mező szerint lehet rendezett a főfájl.
  - elég a főfájl minden blokkjának legkisebb rekordjához készíteni indexrekordot
  - indexrekordok száma: $T(I) = B$ **(ritka index)**
  - indexrekordból sokkal több fér egy blokkba, mint a főfájl rekordjaiból: **bf(I)>>bf**, azaz az indexfájl sokkal kisebb rendezett fájl, mint a főfájl: **B(I)** = B / bf(I) **<<** **B**=T / bf
- **Keresési idő:**
  - az indexfájlban nem szerepel minden érték, ezért csak **fedő értéket kereshetünk**, **a legnagyobb olyan indexértéket, amely a keresett értéknél kisebb vagy egyenlő**
  - fedő érték keresése az index rendezettsége miatt bináris kereséssel történik: **$\log_2(B(I))$**
  - a fedő indexrekordban szereplő blokkmutatónak megfelelő blokkot még be kell olvasni
  - **$1+\log_2(B(I)) << \log_2(B)$ (rendezett eset)**
- **Módosítás:**
  - rendezett fájlba kell beszúrni
  - ha az első rekord változik a blokkban, akkor az indexfájlba is be kell szúrni, ami szintén rendezett
  - megoldás: **üres helyeket hagyunk a főfájl, és az indexfájl blokkjaiban is**. Ezzel a tárméret duplázódhat, de a beszúrás legfeljebb egy főrekord és egy indexrekord visszaírását jelenti.

Indexelés

Elsődleges index

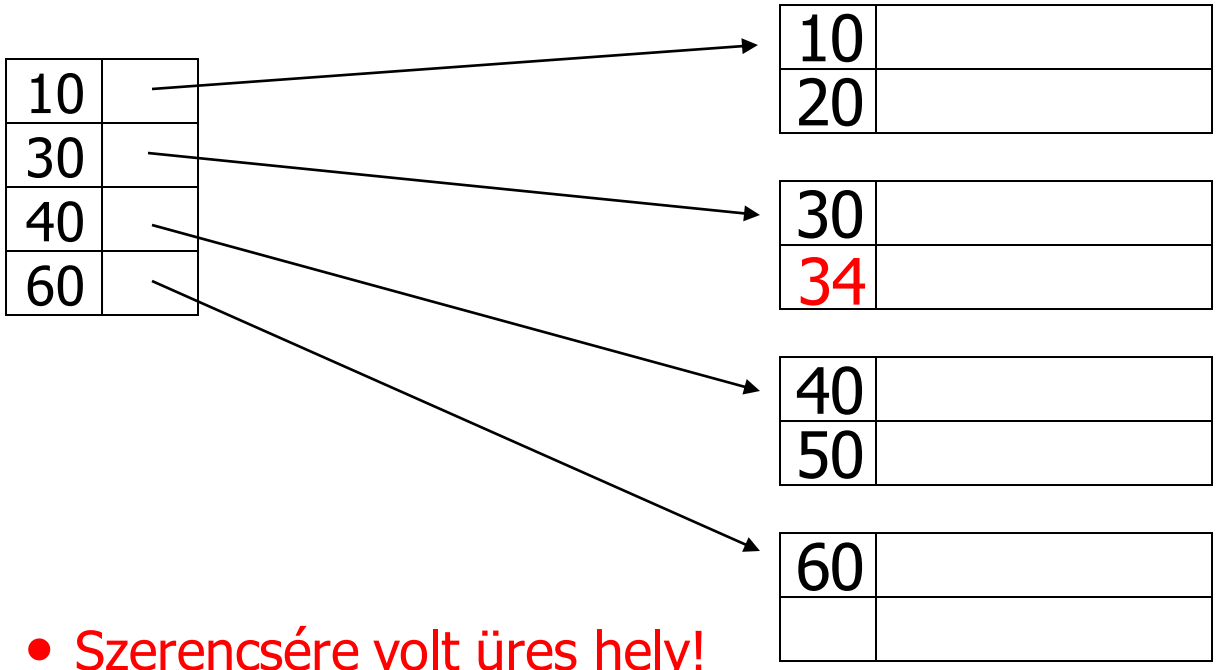
Az adatfájl rendezett, ezért elég a blokkok első rekordjaihoz indexrekordokat tárolni.



Indexelés

Beszúrás ritka index esetén:

Vigyük be a 34-es rekordot!

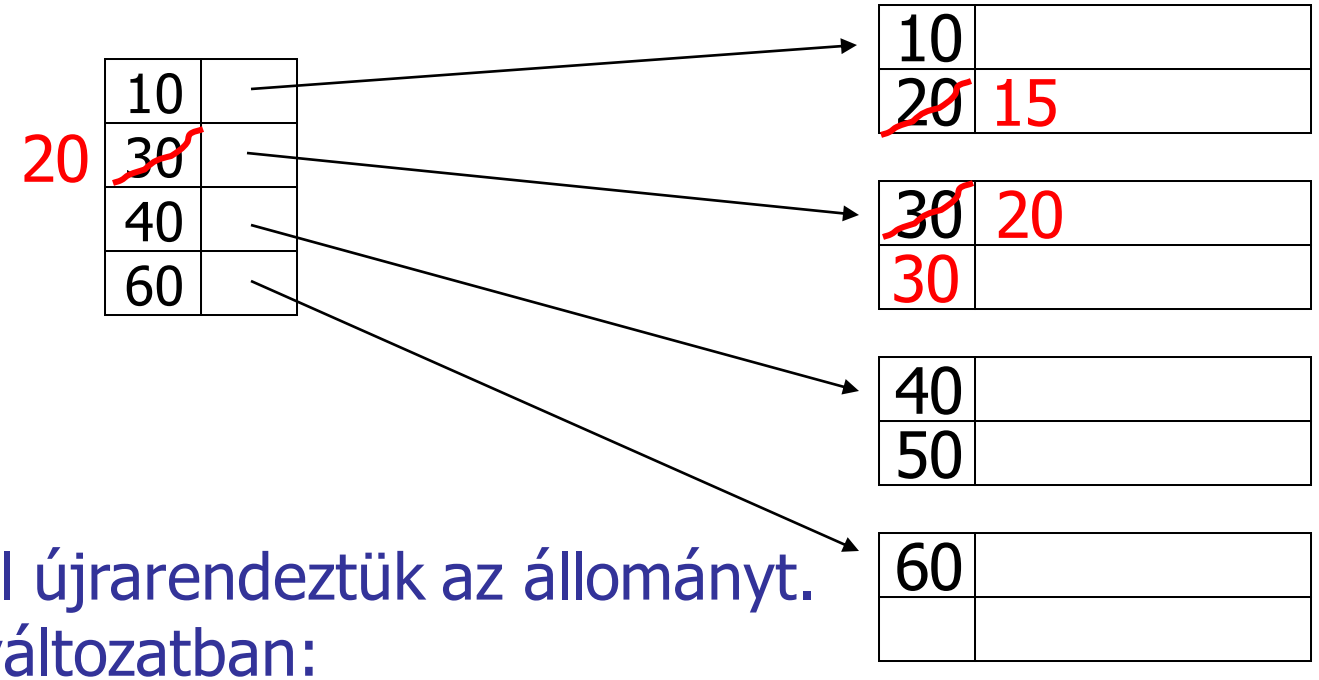


- Szerencsére volt üres hely!

Indexelés

Beszúrás ritka index esetén:

Vigyük be a 15-ös rekordot!

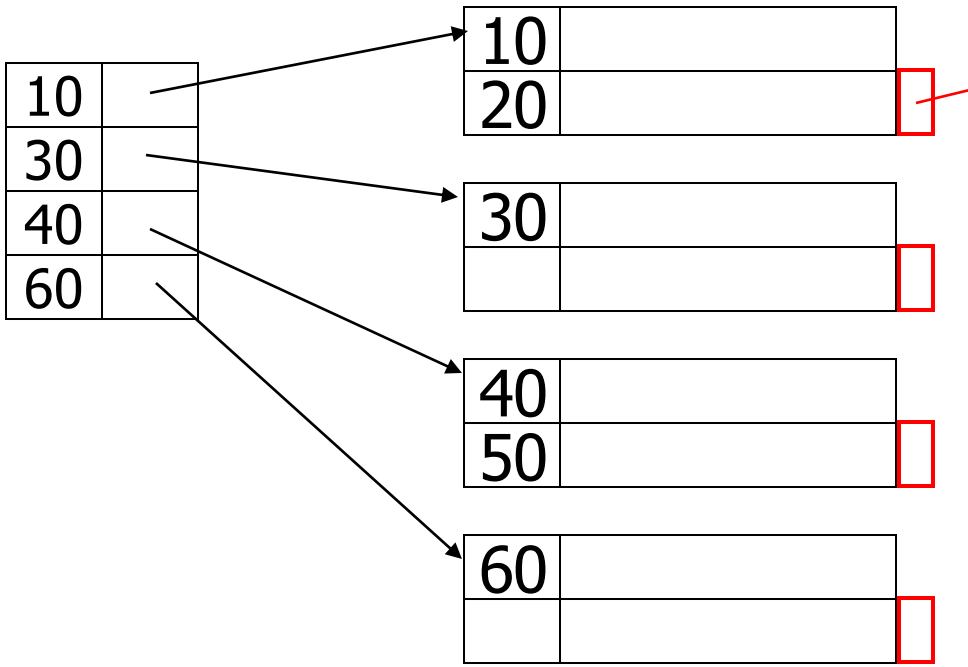


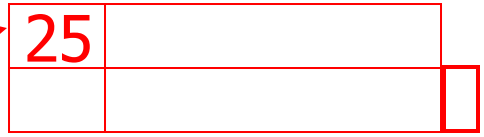
- Azonnal újrarendeztük az állományt.
- Másik változatban:
  - túlcsordulási blokkot láncolunk a blokkhoz

Indexelés

Beszúrás ritka index esetén:

Vigyük be a 25-ös rekordot!



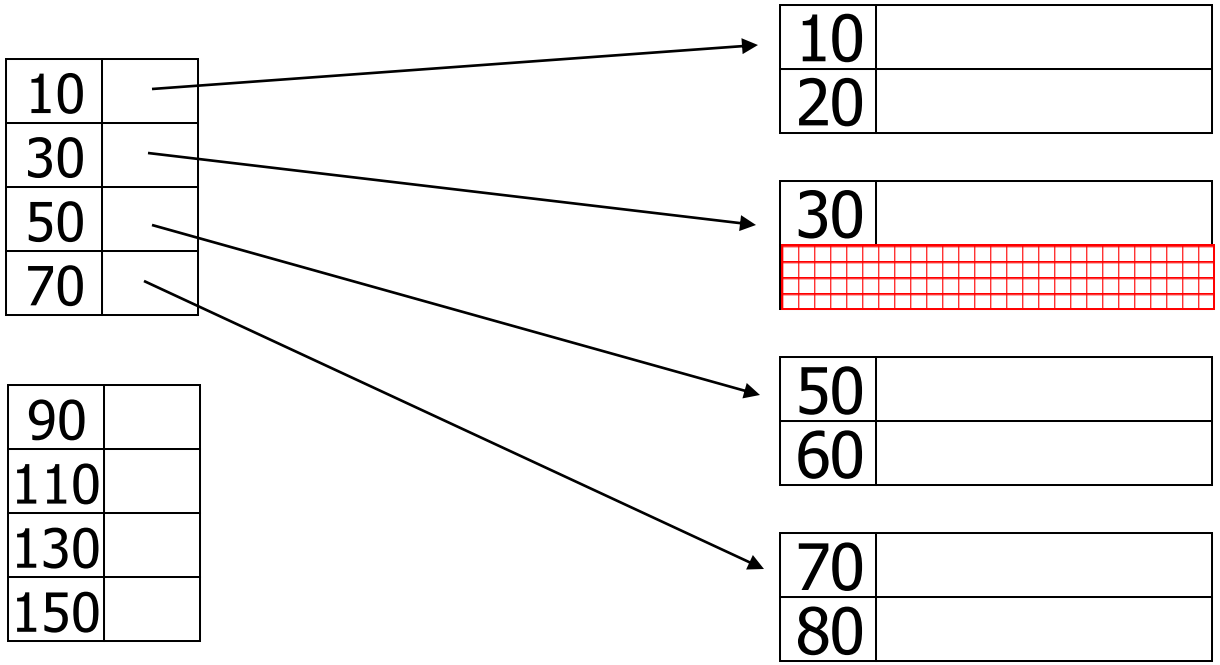


Túlcsordulási blokkot nyitunk, és későbbre halasztjuk az újrarendezést.

Indexelés

Törlés ritka indexből:

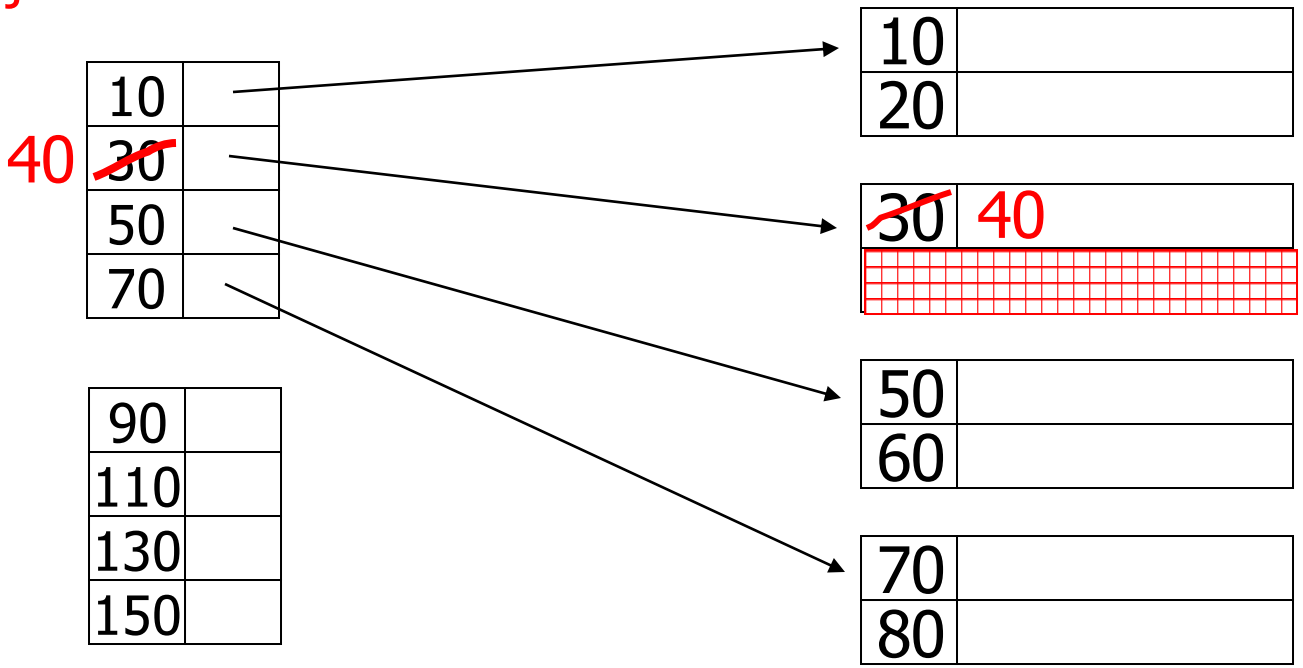
Töröljük a 40-es rekordot!



Indexelés

Törlés ritka indexből:

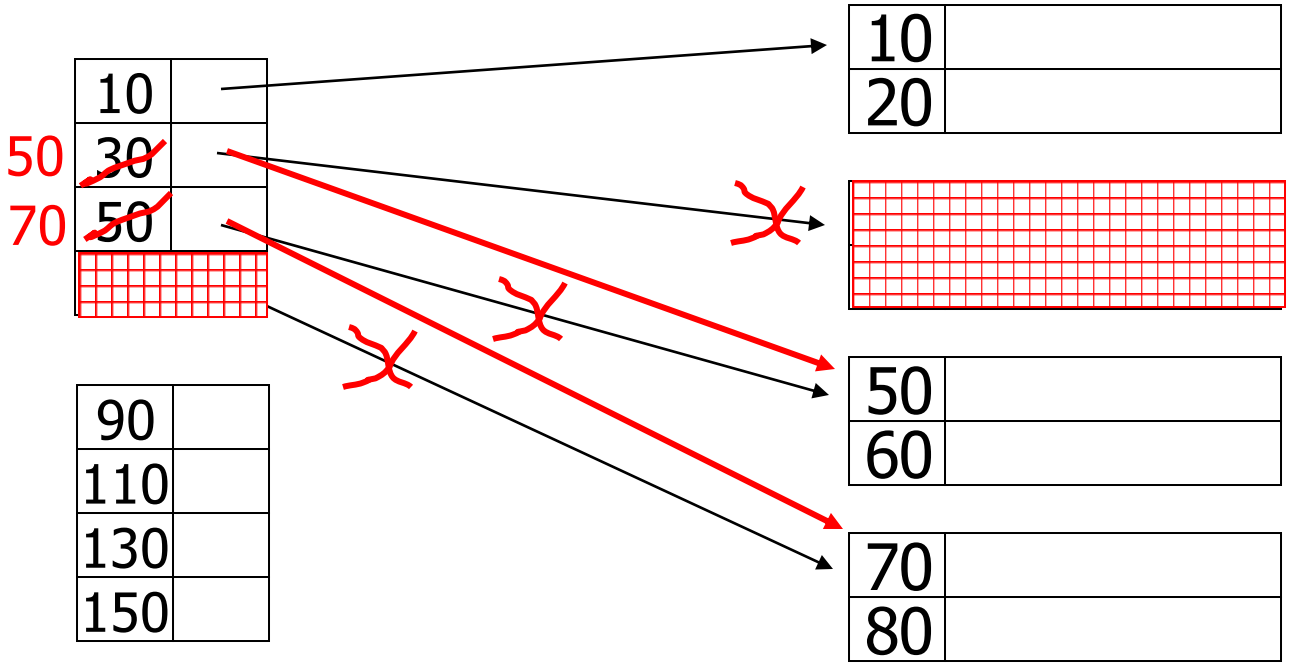
Töröljük a 30-as rekordot!



Indexelés

Törlés ritka indexből:

Töröljük a 30-as és 40-es rekordot!



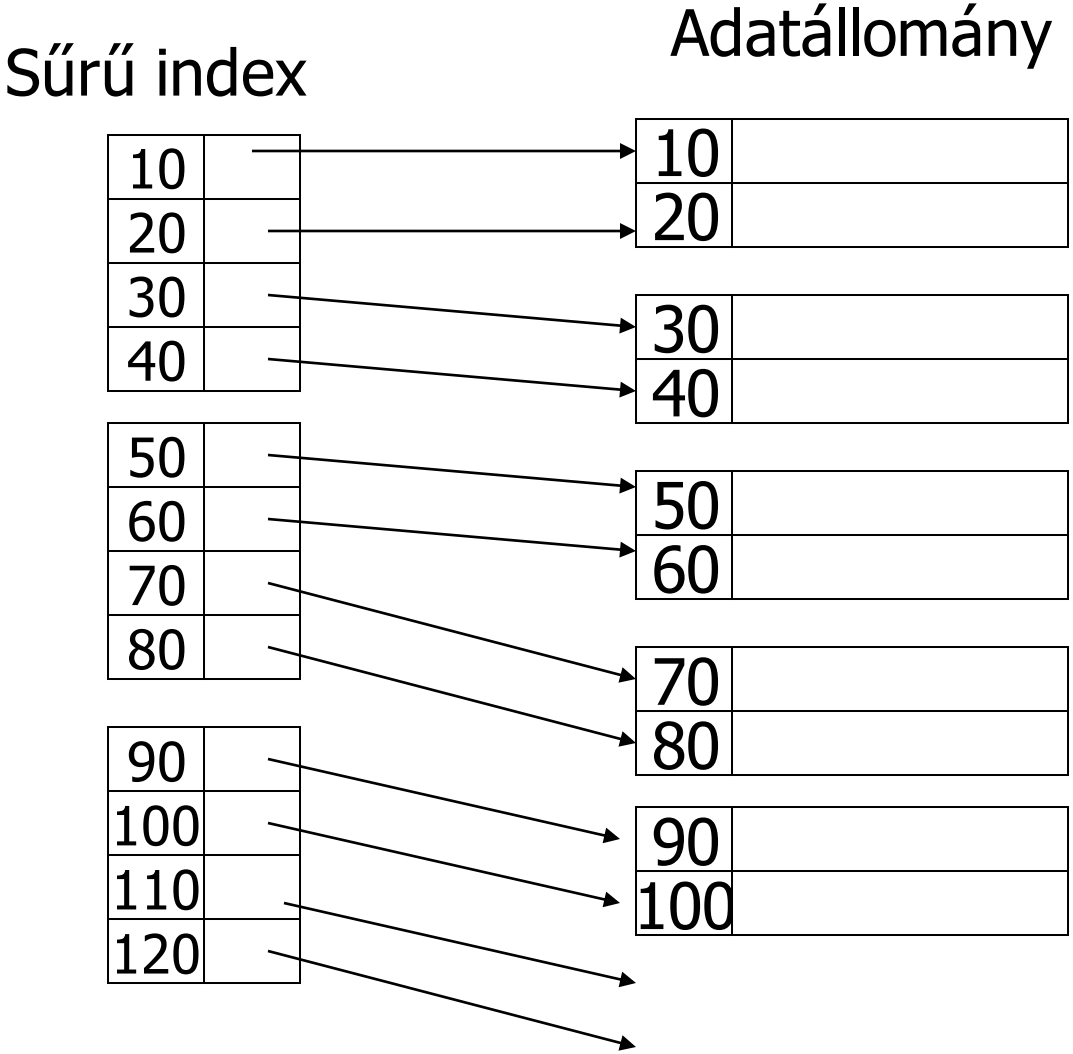
Indexelés

- **Másodlagos index:**
  - **főfájl rendezetlen (az indexfájl mindig rendezett)**
  - több másodlagos indexet is meg lehet adni
  - **a főfájl minden rekordjához kell készíteni indexrekordot**
  - indexrekordok száma: $T(I)=T$ (sűrű index)
  - indexrekordból sokkal több fér egy blokkba, mint a főfájl rekordjaiból: $bf(I)>>bf$, azaz az indexfájl sokkal kisebb rendezett fájl, mint a főfájl:
  - $B(I) = T/bf(I) << B=T/bf$
- **Keresési idő:**
  - az indexben keresés az index rendezettsége miatt bináris kereséssel történik: $\log_2(B(I))$
  - a talált indexrekordban szereplő blokkmutatónak megfelelő blokkot még be kell olvasni
  - $1+\log_2(B(I)) << \log_2(B)$ (rendezett eset)
  - az elsődleges indexnél rosszabb a keresési idő, mert több az indexrekord
- **Módosítás:**
  - a főfájl kupac szervezésű
  - rendezett fájlba kell beszúrni
  - ha az első rekord változik a blokkban, akkor az indexfájlba is be kell szúrni, ami szintén rendezett
  - megoldás: **üres helyeket hagyunk a főfájl, és az indexfájl blokkjaiban is**. Ezzel a tárméret duplázódhat, de a beszúrás legfeljebb egy főrekord és egy indexrekord visszaírását jelenti.

Indexelés

Másodlagos index

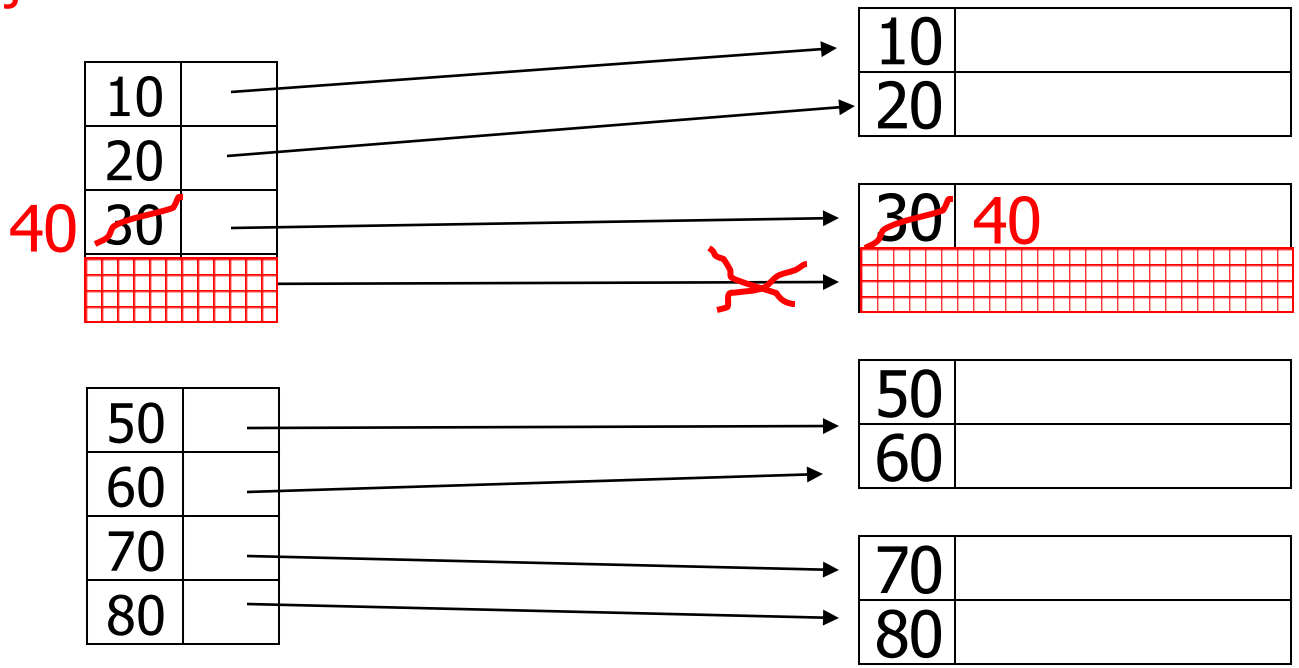
Minden rekordhoz tartozik indexrekord.



Indexelés

Törlés sűrű indexből:

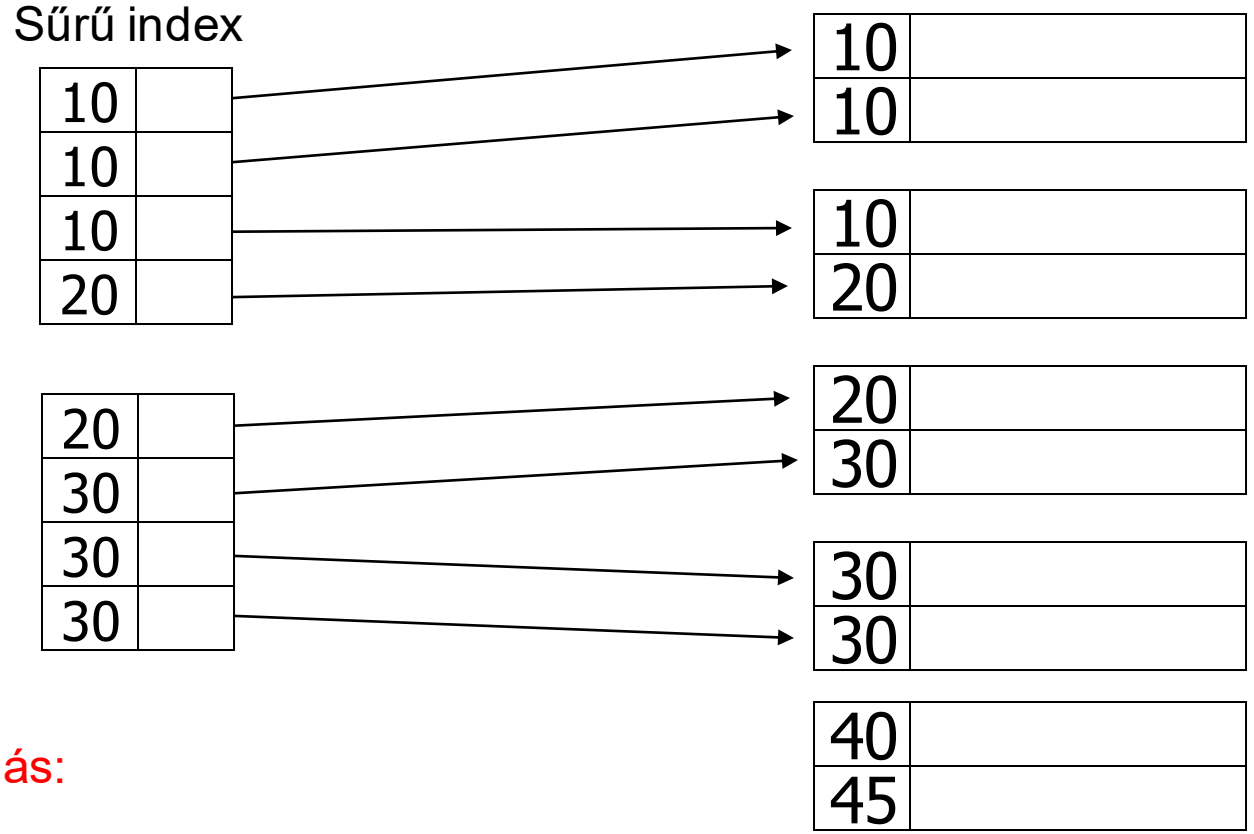
Töröljük a 30-as rekordot!



Indexelés

Mi történik, ha egy érték többször is előfordulhat?

Több megoldás is lehetséges. Először tegyük fel, hogy rendezett az állomány.

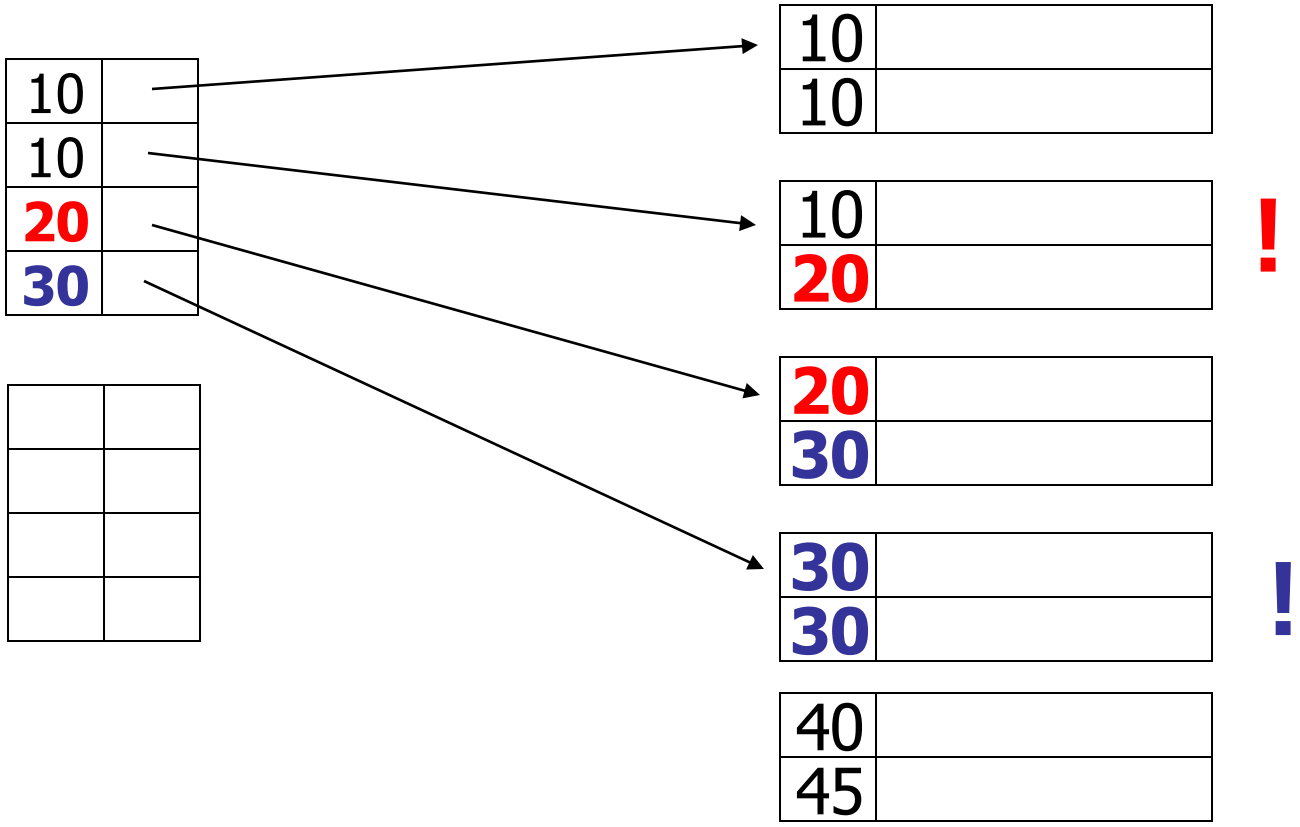


1. megoldás:

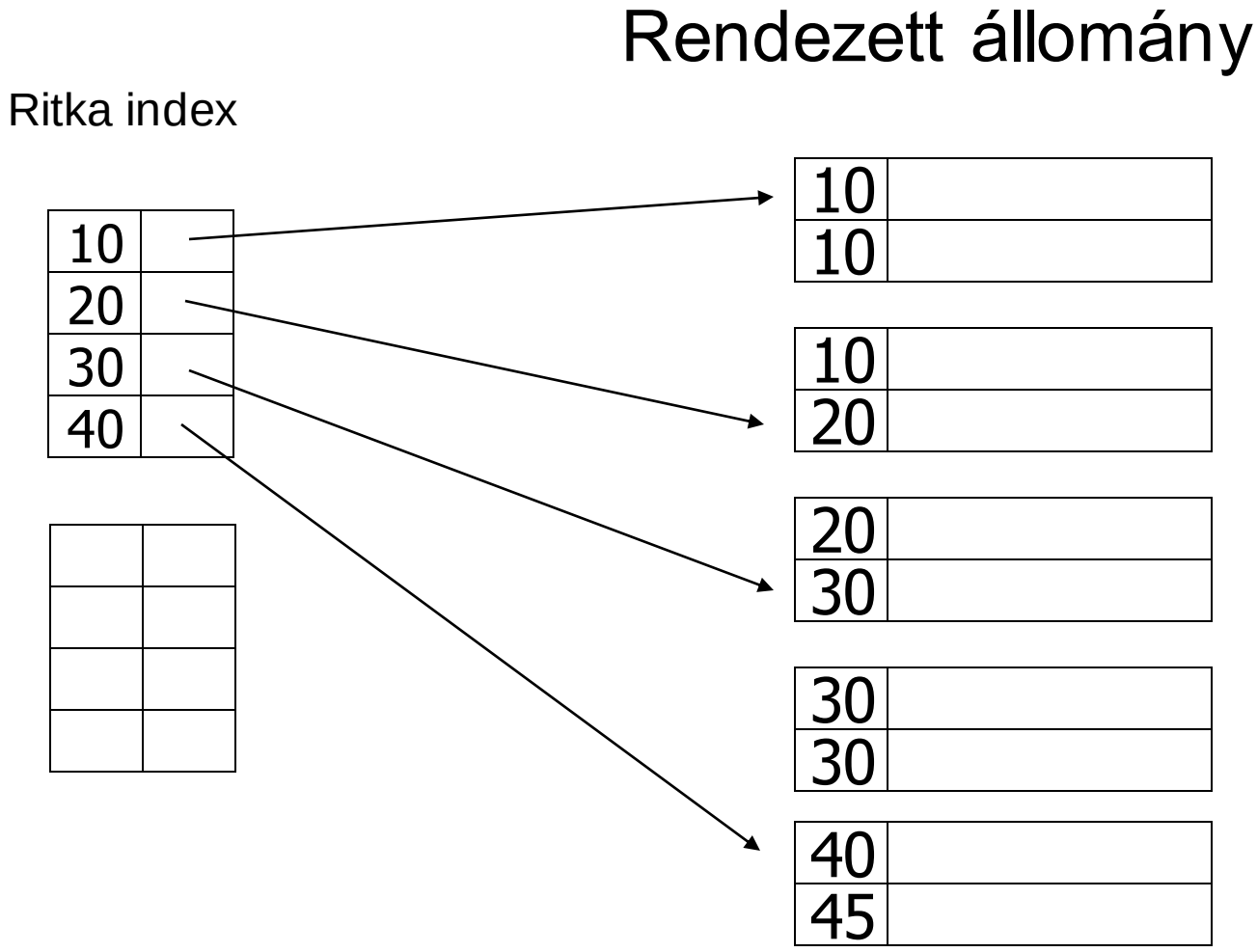
Minden rekordhoz tárolunk egy indexrekordot.

Indexelés

Vigyázat! Ritka index nem jó. A fedőértéknek megfelelő blokk előtti és utáni blokkokban is lehetnek találatok. Például, ha a 20-ast vagy a 30-ast keressük.



Indexelés

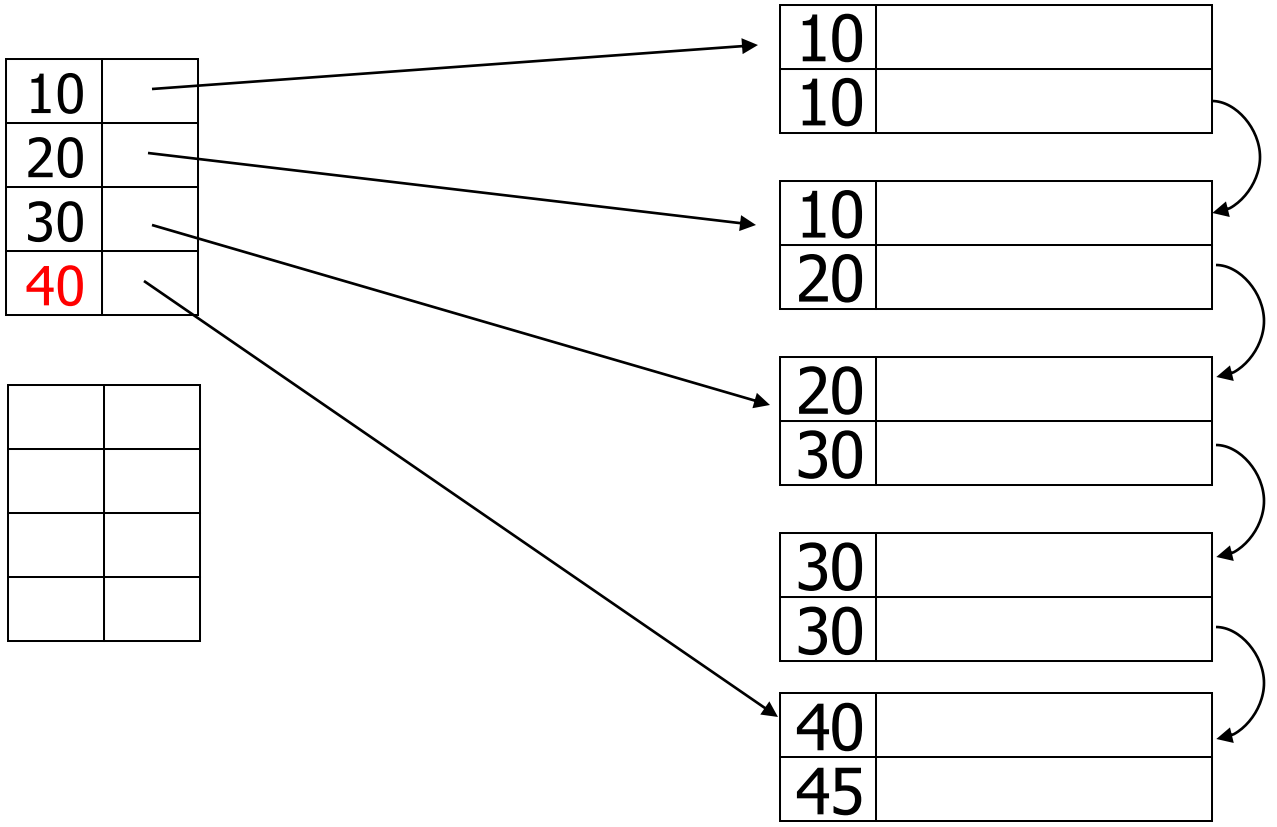


2. megoldás:

Rendezett állomány esetén csak az első előforduláshoz tárolunk egy indexrekordot.

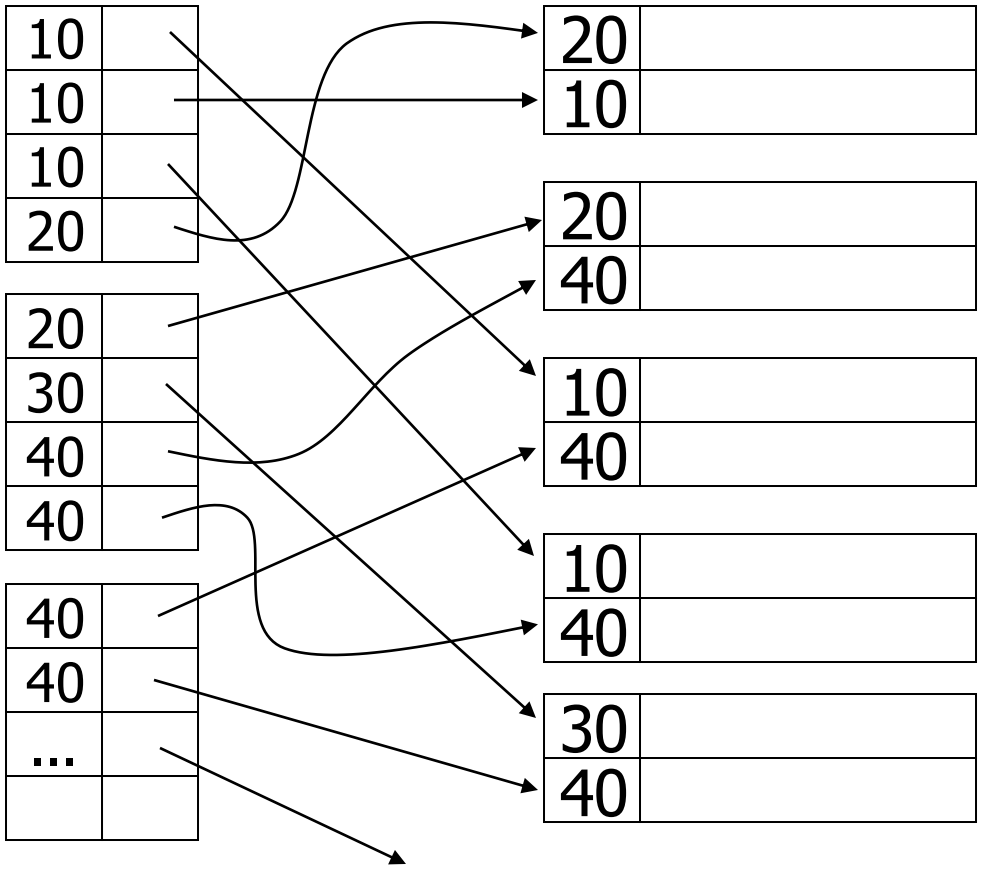
Indexelés

Rendezett állomány esetén nem az első rekordhoz, hanem az értékhez tartozó első előforduláshoz készítünk indexrekordot. Az adatállomány blokkjait láncoljuk.



Indexelés

Ha nem rendezett az állomány, akkor nagy lehet a tárolási és keresési költség is:

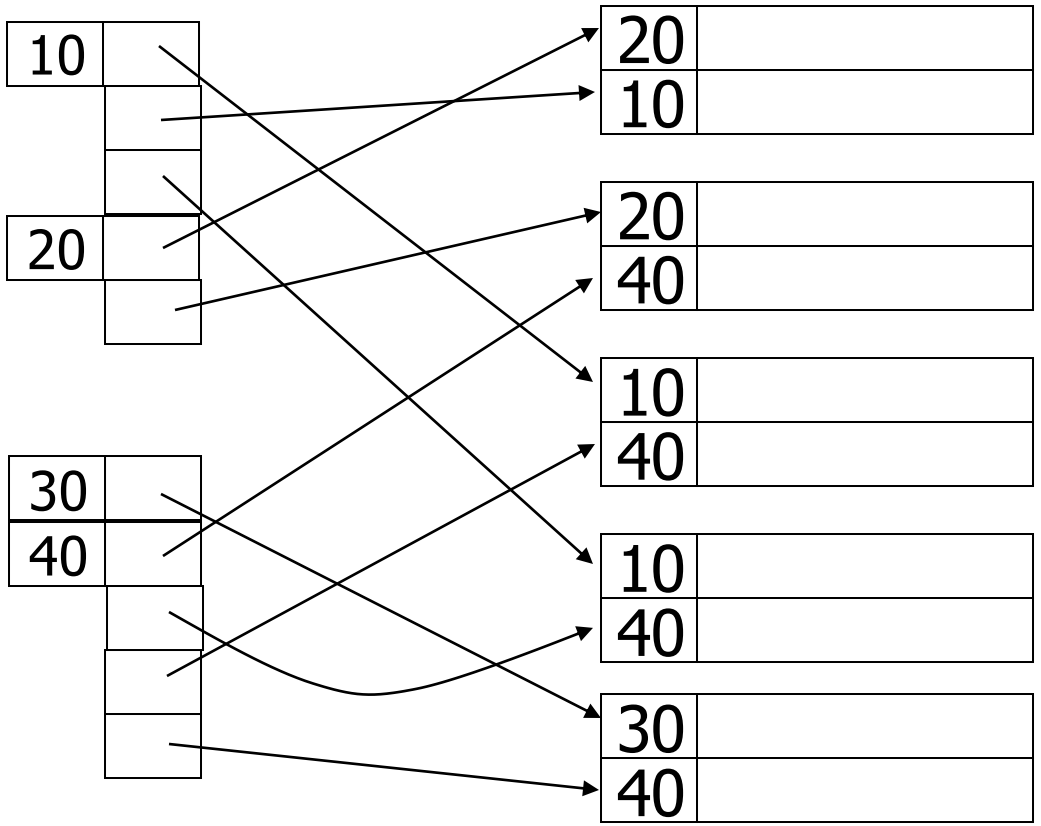


Indexelés

Egy lehetséges megoldás, hogy az indexrekordok szerkezetét módosítjuk:

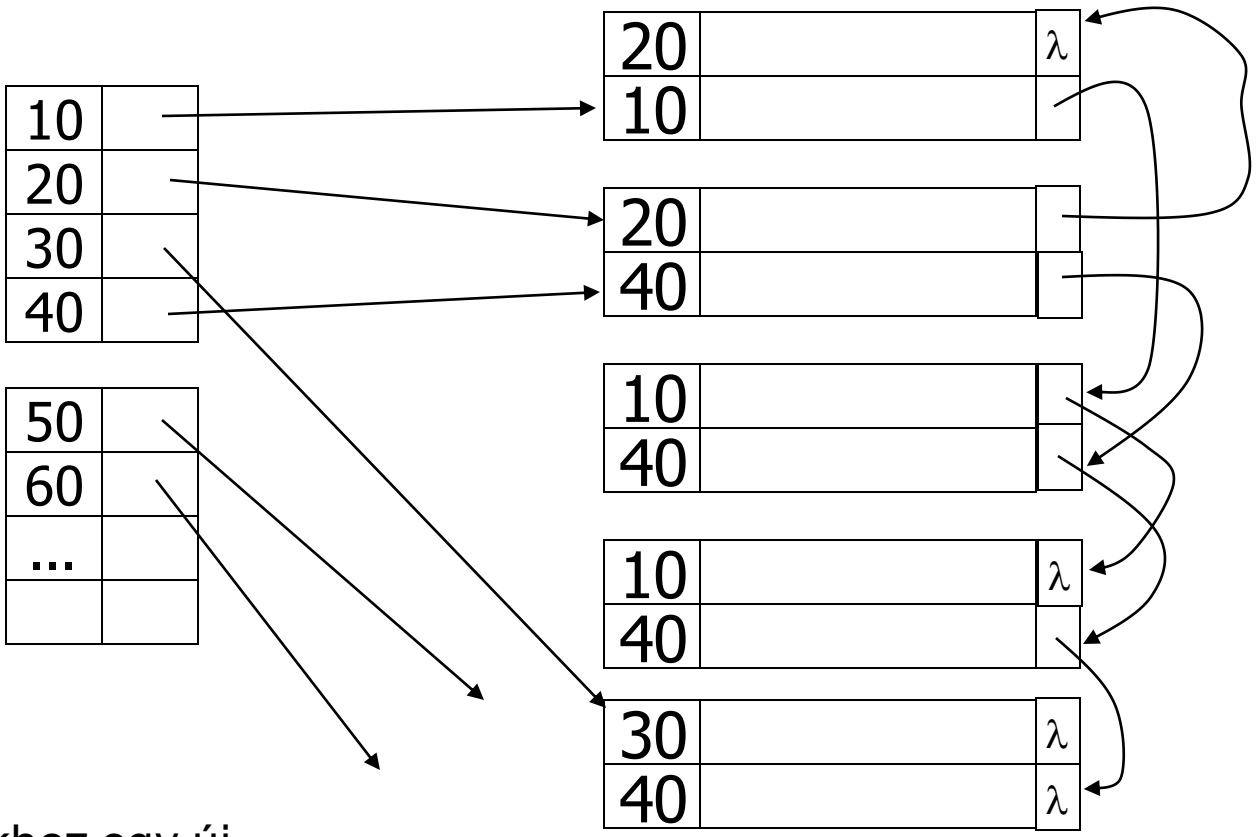
(indexérték, mutatóhalmaz)

Probléma: változó hosszú indexrekordok keletkeznek



Indexelés

Összeláncolhatjuk az egyforma értékű rekordokat.

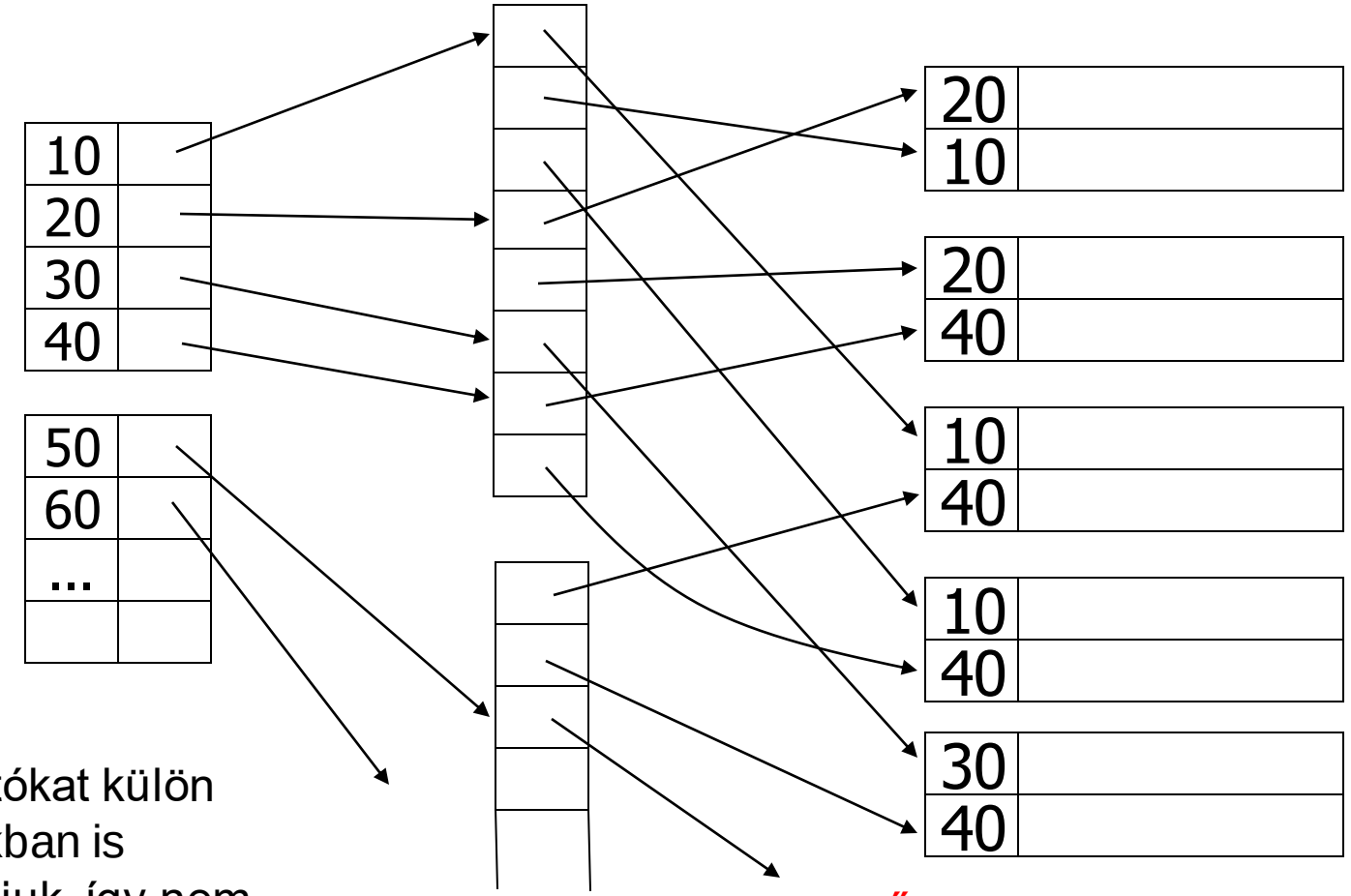


Probléma:

- a rekordokhoz egy új, mutató típusú mezőt kell adnunk

- követni kell a láncot

Indexelés



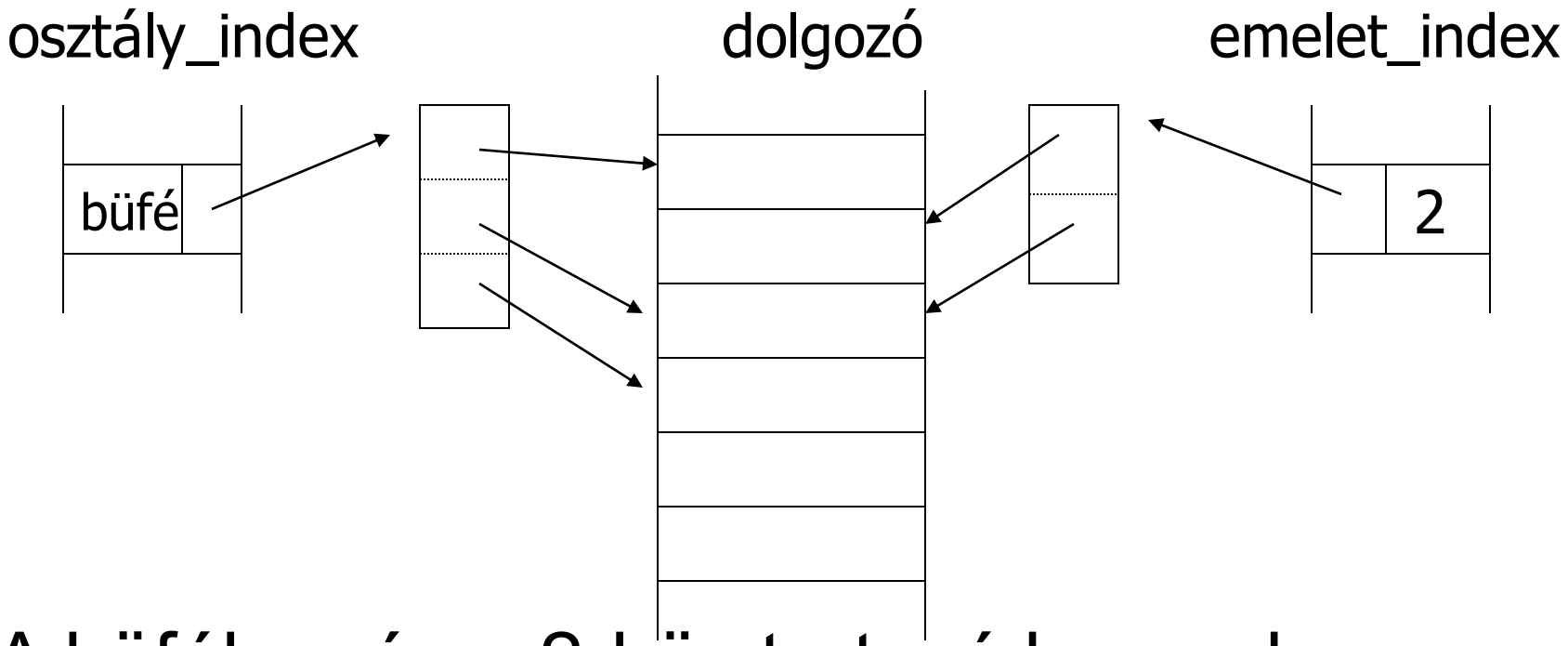
A mutatókat külön blokkokban is tárolhatjuk, így nem kell változó hosszú indexrekordokat kezelni.

Kosarak

ELŐNY: több index esetén a logikai feltételek halmazműveletekkel kiszámolhatók.

Indexelés

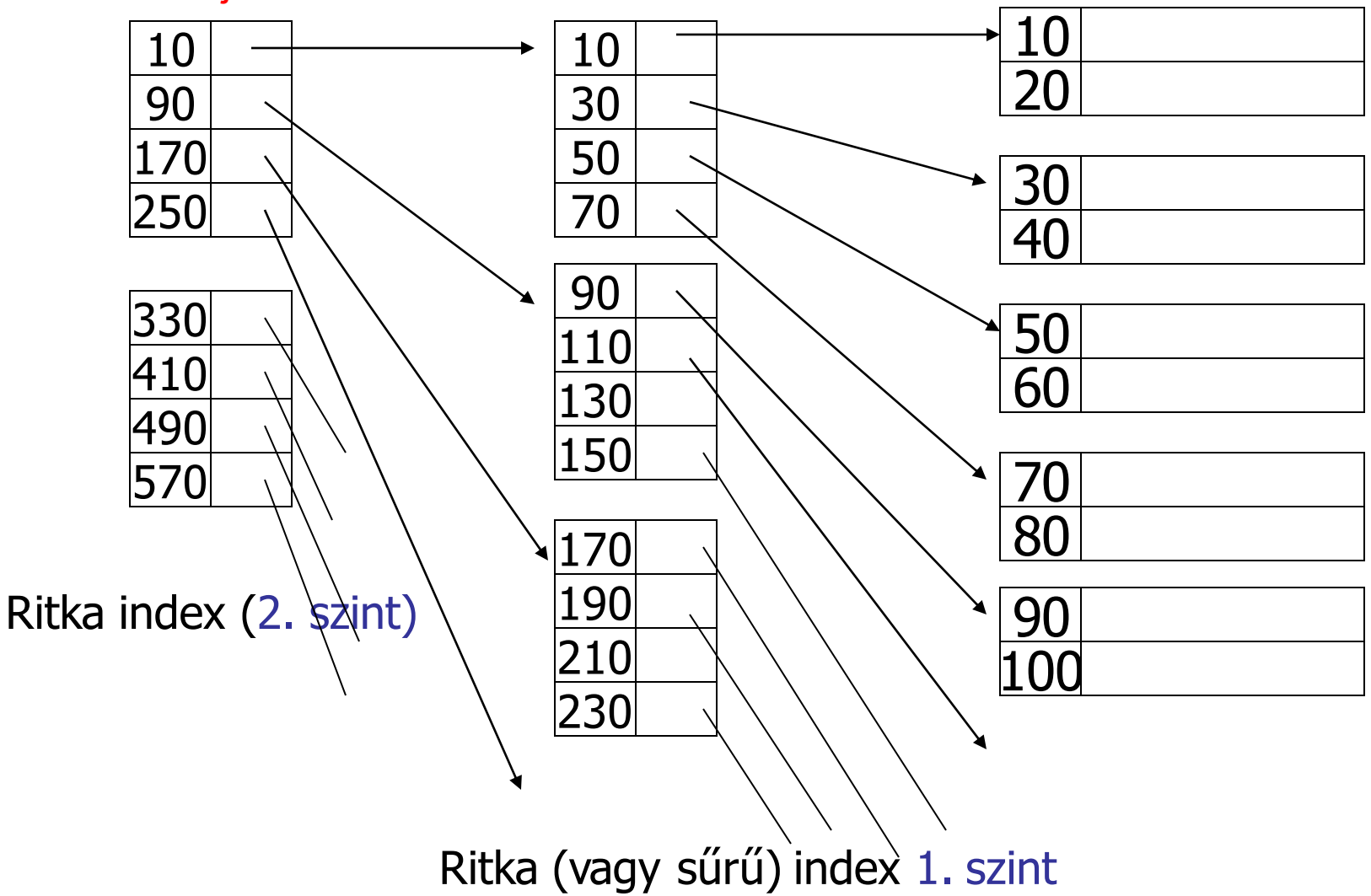
select * from dolgozó where osztály='büfé' and emelet=2;



A büféhez és a 2-höz tartozó kosarak metszetét kell képezni, hogy megkapjuk a keresett mutatókat.

Indexelés

Ha nagy az index, akkor az indexet is indexelhetjük.



Indexelés

- **Többszintű index:**
  - az indexfájl (1. indexszint) is fájl, ráadásul rendezett, így ezt is meg lehet indexelni, elsődleges indexszel.
  - a főfájl lehet rendezett vagy rendezetlen (az indexfájl mindig rendezett)
  - **t-szintű index**: az indexszinteket is indexeljük, összesen t szintig

Keresési idő:

- a t-ik szinten ($I^{(t)}$) bináris kereséssel **keressük meg a fedő indexrekordot**
- követjük a mutatót, minden szinten, és végül a főfájlban: $\log_2(B(I^{(t)}))+t$ **blokkolvasás**
- ha a legfelső szint 1 blokkból áll, akkor **t+1** blokkolvasást jelent. **(t=?)**
- **minden szint blokkolási faktora megegyezik**, mert egyforma hosszúak az indexrekordok.

Indexelés

	FŐFÁJL	1. szint	2. szint	...	t. szint
blokkok száma	B	B/bf(I)	B/bf(I)^2	...	B/bf(I)^t
rekordok száma	T	B	B/bf(I)	...	B/bf(I)$^{(t-1)}$
blokkolási faktor	bf	bf(I)	bf(I)	...	bf(I)

•Indexmutató minden blokkra, amely tartalmaz rekordot (*ritka index*)

• t-ik szinten 1 blokk: $1=B/bf(I)^t$

azaz $t=log_{bf(I)}B < log_2(B)$ azaz jobb a rendezett fájlszervezésnél. (Itt bf azt adja meg, hogy egy blokkba hány rekord fér. A $bf(I)$ azt adja meg, hogy egy blokkba hány index rekord fér el.)

• $log_{bf(I)}B < log_2(B(I))$ is teljesül általában, így az egyszintű indexeknél is gyorsabb

Feladatok és megoldások

1. Minden blokkba 3 rekord, vagy 10 indexrekord (érték-mutató pár) fér. Összesen *n* rekordunk van. Hány blokkos az adatfájl, mennyi kell az *index fájlra*, ha sűrű az index és ha ritka az index?

Megoldás:

Adatfájl mérete: n/3.

Sűrű index (n indexrekordot jelent) mérete: n/10.

Ritka index (minden blokkhoz 1 indexrekord) mérete: (n/3)/10=n/30.

Feladatok és megoldások

2. Minden blokkba 30 rekord, vagy 200 indexrekord (érték-mutató pár) fér. Összesen *n* rekordunk van. Semelyik blokk telítettsége nem lehet több, mint 80%. Hány blokkos az adatfájl, mennyi kell az *index fájlra*, a sűrű index és a ritka index esetében?

Megoldás:

Adatfájl mérete: (*n*/30)/0.8=n/24.

Sűrű index (*n* indexrekordot jelent) mérete: (*n*/200)/0.8=n/160.

Ritka index (minden blokkhoz 1 indexrekord) mérete: ((*n*/24)/200)/0.8=n/3840.

Feladatok és megoldások

3. Minden blokkba 3 rekord, vagy 10 indexrekord (érték-mutató pár) fér. Összesen *n* rekordunk van. Többszintű indexünk legfelső szintje csak 1 blokkból áll. Hány blokkos az indexfájl és a főfájl, ha az első szinten sűrű az index, vagy ha az első szinten ritka index?

	FŐFÁJL	1. szint	2. szint	...	t. szint
blokkok száma	**B**	**B/bf(I)**	**B/bf(I)^2**	**...**	**B/bf(I)^t**
rekordok száma	**T**	**B**	**B/bf(I)**	**...**	**B/bf(I)$^{(t-1)}$**
blokkolási faktor	**bf**	**bf(I)**	**bf(I)**	**...**	**bf(I)**

$B=T/bf$, $bf=3$, $B(I)=B/bf(I)$, $bf(I)=10$

Sűrű index $n=T=T(I)$;adat / index rekordok száma

Ritka index $T(I)=B$

Feladatok és megoldások

4. Minden blokkba 3 rekord, vagy 10 indexrekord (érték-mutató pár), vagy 50 mutató fér. Tegyük fel, hogy átlagosan 10-szer szerepel minden indexérték. Összesen 3000 rekordunk van. Másodlagos indexet készítünk, úgy, hogy az egy indexértékhez tartozó mutatókat kosarak blokkjaiban tároljuk. Mekkora az állomány mérete összesen, beleértve az adatokat, indexeket és mutatókat tartalmazó blokkokat?
   1. Mennyi blokkra van szükségünk, ha nem használunk kosarakat (*buckets*)?
   2. Ha nem ismerünk semmilyen korlátot a különböző indexértékere, „keresési-kulcs—mutató” párokra, akkor mennyi a minimálisan és maximálisan szükséges blokkok száma?

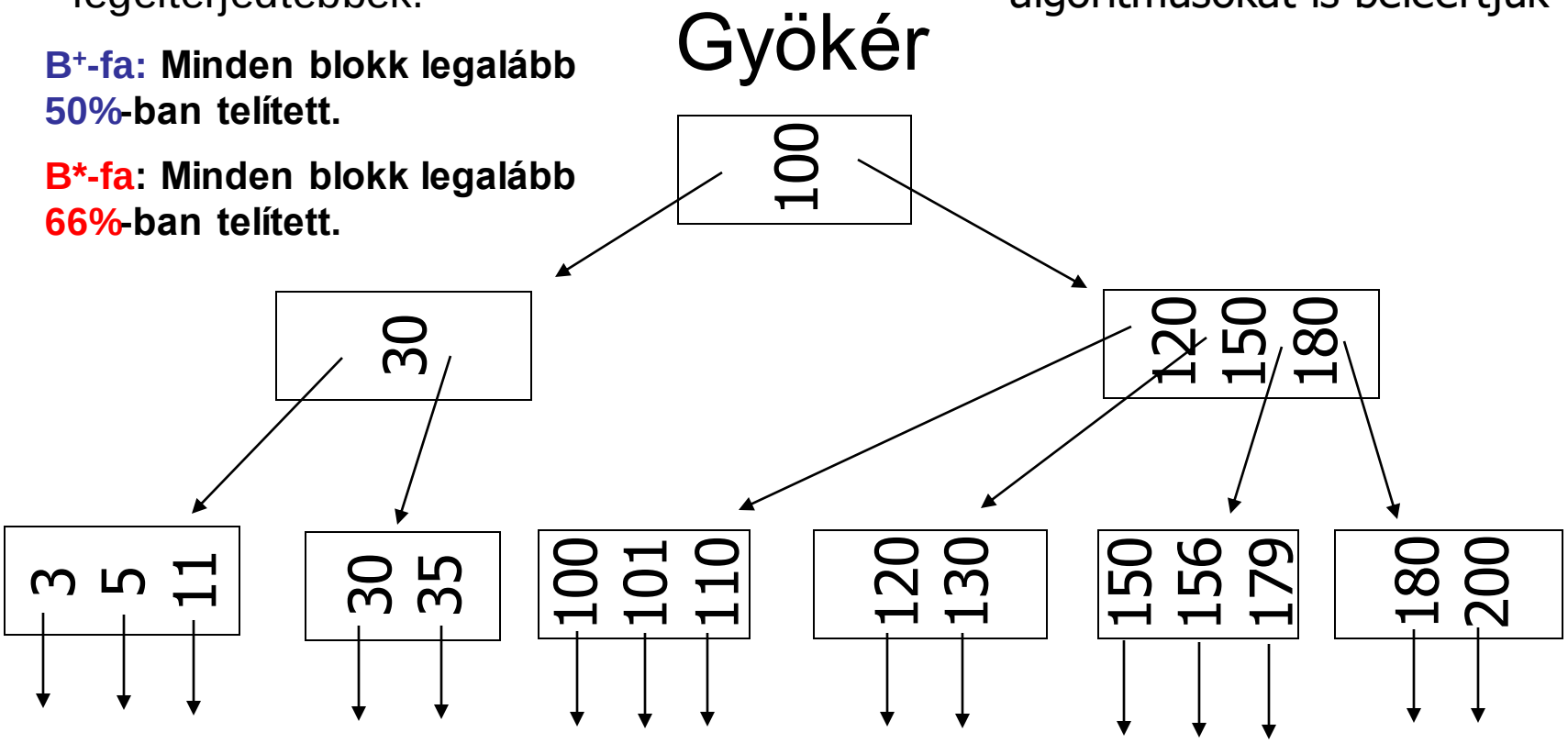
Indexelés

A többszintű indexek közül a **B⁺-fák, B*-fák** a legelterjedtebbek.

B⁺-fa: Minden blokk legalább 50%-ban telített.

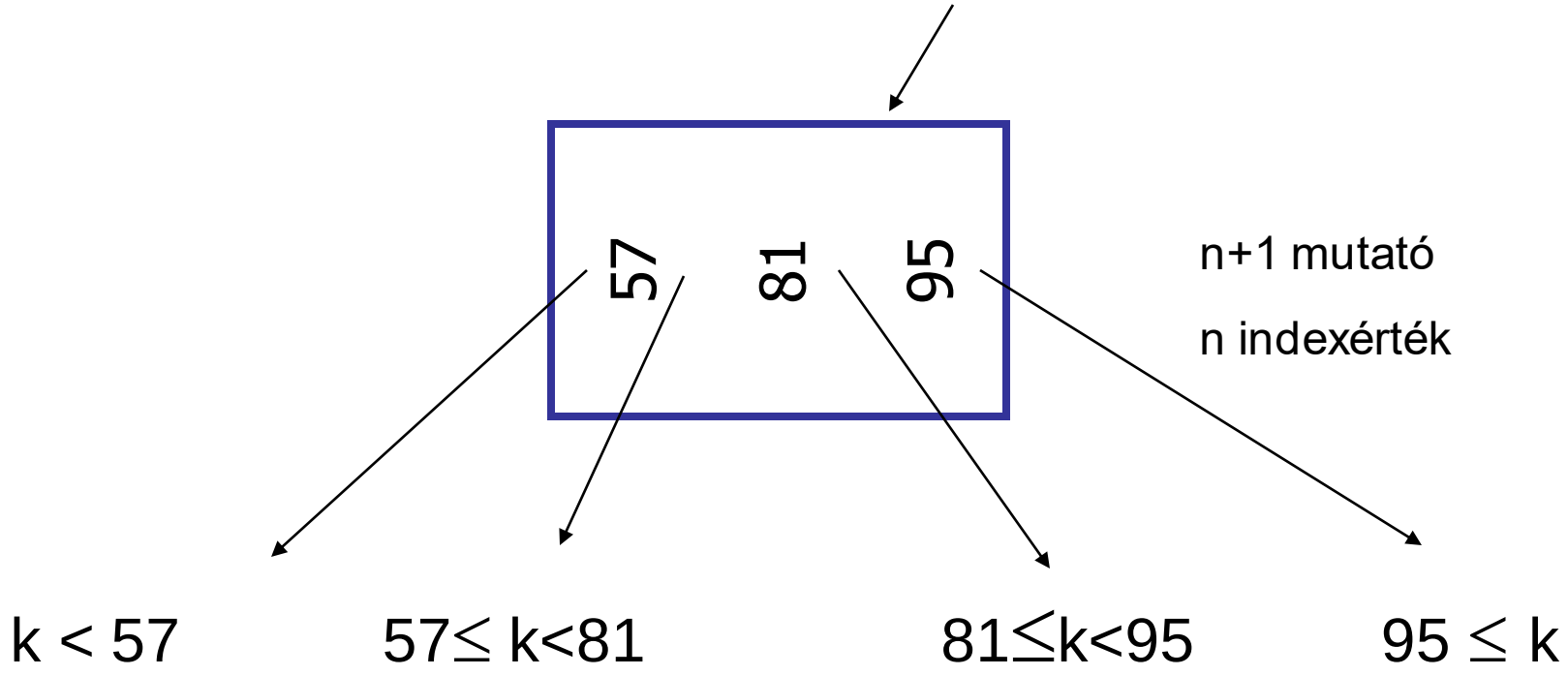
B*-fa: Minden blokk legalább 66%-ban telített.

B⁺-fa: a szerkezeten kívül a telítettséget biztosító karbantartó algoritmusokat is beleértjük



Indexelés

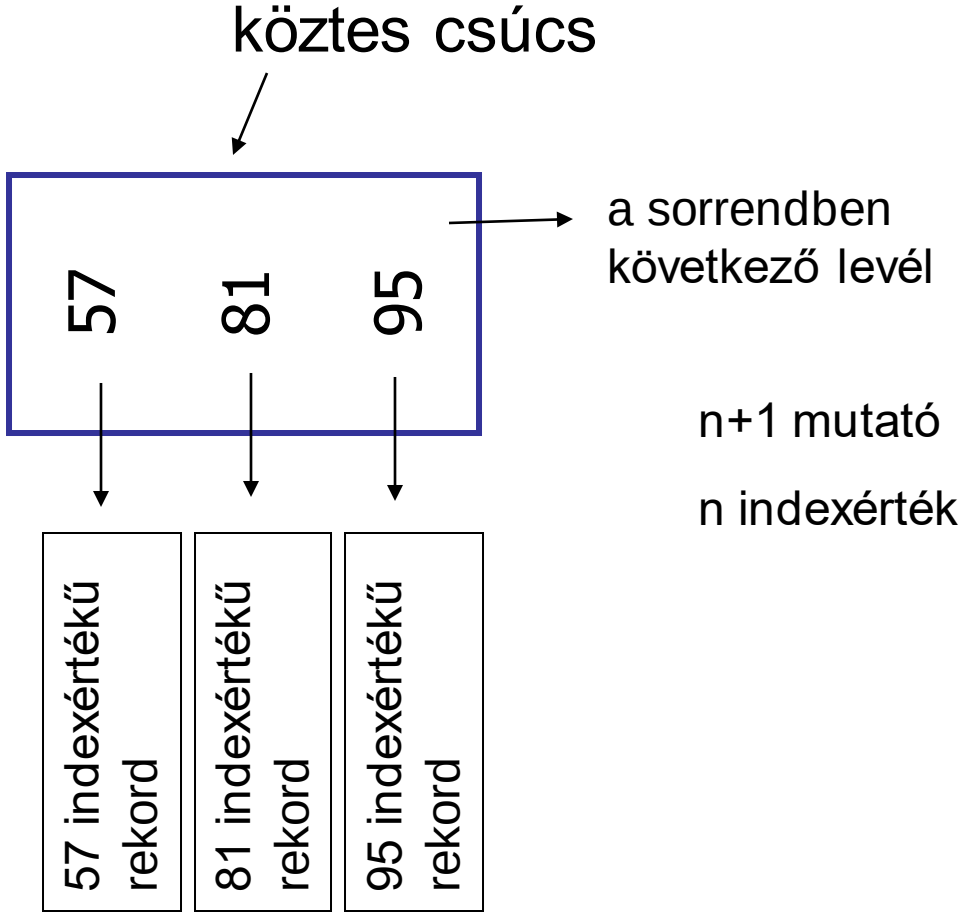
Köztes (nem-levél) csúcs szerkezete



Ahol k a mutató által meghatározott részben (részgráfban) szereplő tetszőleges indexérték

Indexelés

Levél csúcs szerkezete



Felépítésre vonatkozó szabályok

- Szabályok (feltesszük, hogy egy csúcs n értéket és n+1 mutatót tartalmazhat):
  - a gyökérben legalább két mutatónak kell lennie (kivéve, ha a B-fa (B+-fa) egyetlen bejegyzést tartalmaz),
  - a levelekben az utolsó mutató a következő (jobboldali) levélre mutat, legalább $\lfloor (n+1)/2 \rfloor$ mutatónak használatban kell lennie,
  - köztes pontokban minden mutató a B-fa következő szintjére mutat, közülük legalább $\lceil (n+1)/2 \rceil$ darabnak használatban kell lennie,
  - ha egy mutató nincs használatban úgy vesszük, mintha NILL mutató lenne.

Beszúrás I.

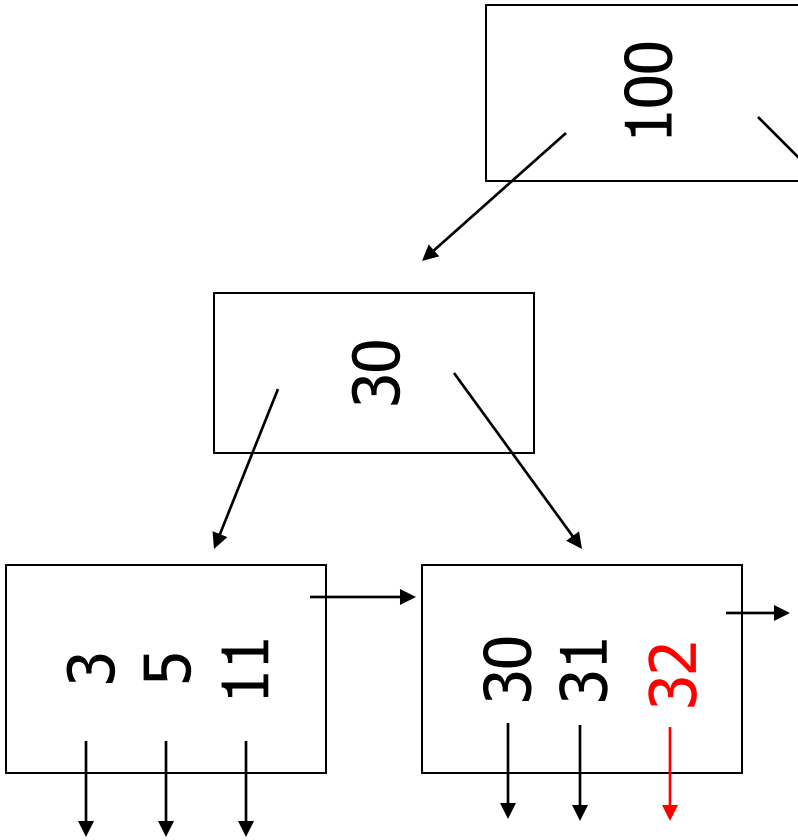
- Ha van szabad hely az új kulcs számára a megfelelő levélben, beszúrjuk.
- Ha nincs, kettévágjuk a levelet, és szétosztjuk a kulcsokat a két új levél között, így mindkettő félig lesz telítve, vagy éppen csak egy kicsit jobban.
- Az eggyel feljebb lévő szint megfelelő csúcsát ennek megfelelően kell kiigazítani.

Beszúrás II.

- Egy csúcs szétvágása tehát hatással lehet a fölötte lévő szintre is. Itt az előbbi két pontban megadott stratégiát alkalmazzuk rekurzívan.
- Itt viszont: ha N olyan belső csúcs, aminek kapacitása n kulcs, n+1 mutató és most az (n+2). mutatót illesztenénk be, akkor szintén létrehozunk egy új M pontot. N-ben most marad $\lceil n/2 \rceil$ kulcs, M-ben lesz $\lfloor n/2 \rfloor$ kulcs, a "középen lévő" kulcs pedig az eggyel fentebbi szintre kerül, hogy elválassza N-t és M-t egymástól.
- Ha a gyökérbe nem tudunk beszúrni, mert nincs hely, akkor szétvágjuk a gyökeret két új csúcsra, és „fölöttük” létrehozunk egy új gyökeret, aminek két bejegyzése lesz.

Indexelés

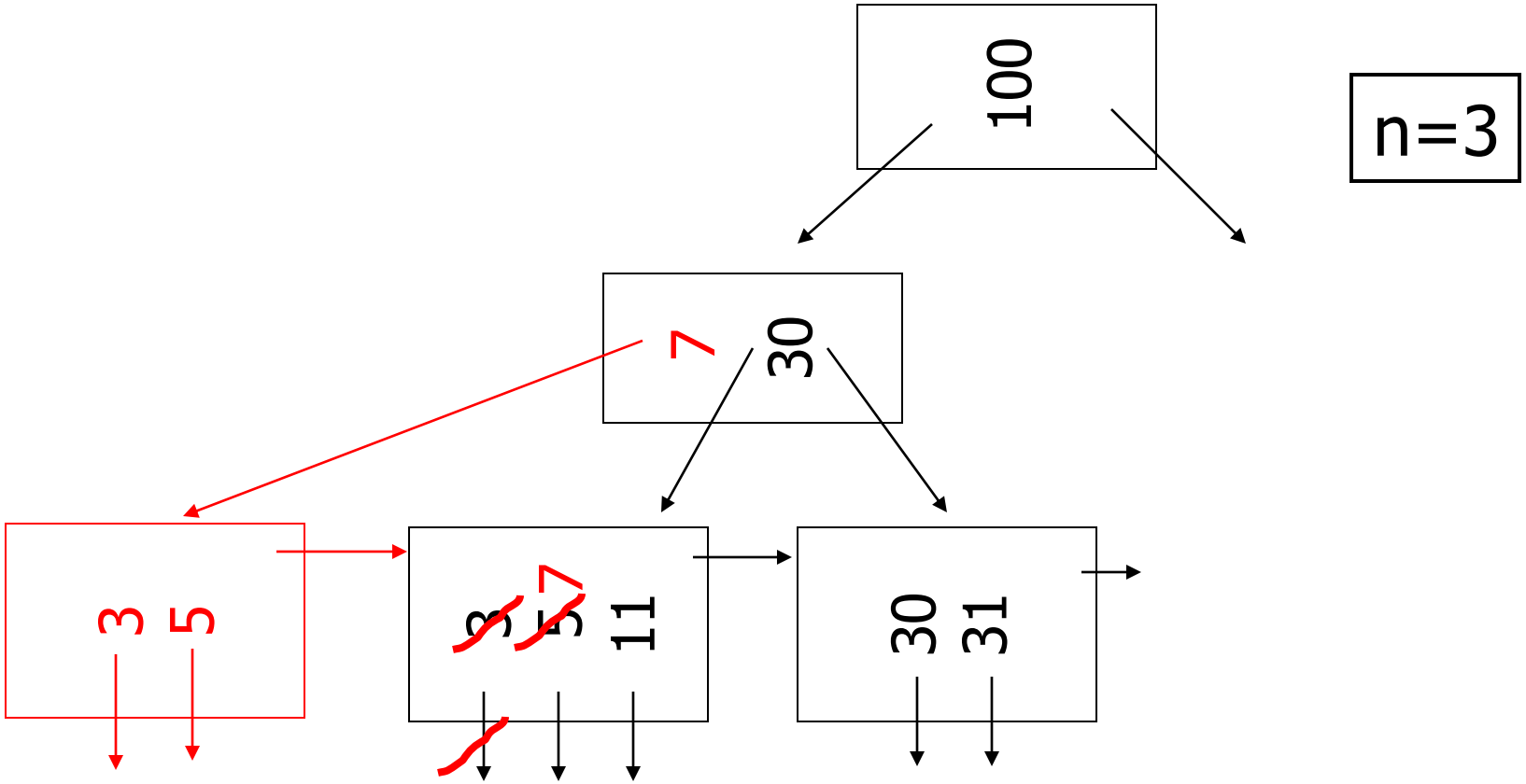
Szúrjuk be a 32-es indexértékű rekordot!



n=3

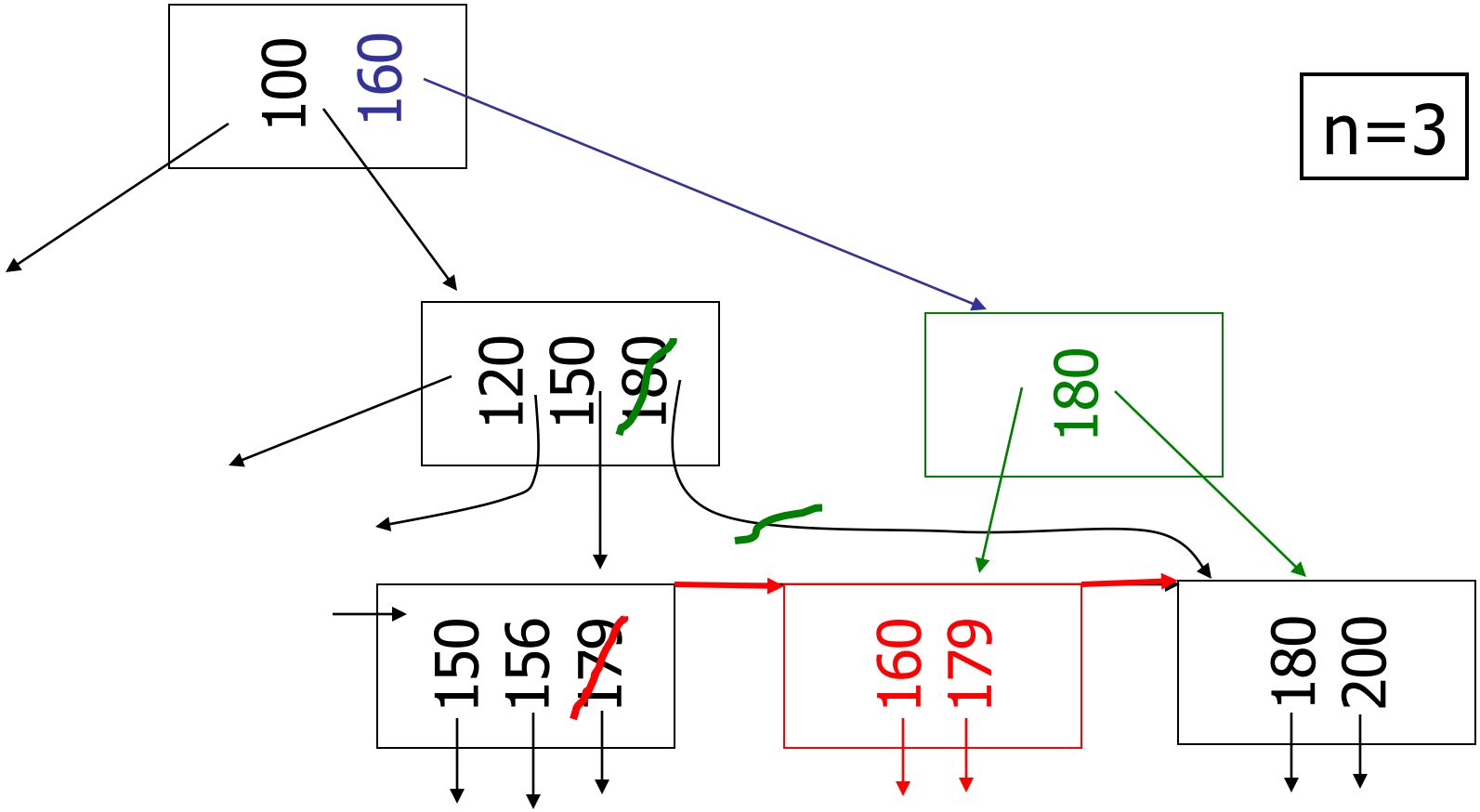
Indexelés

Szúrjuk be a 7-es indexértékű rekordot!



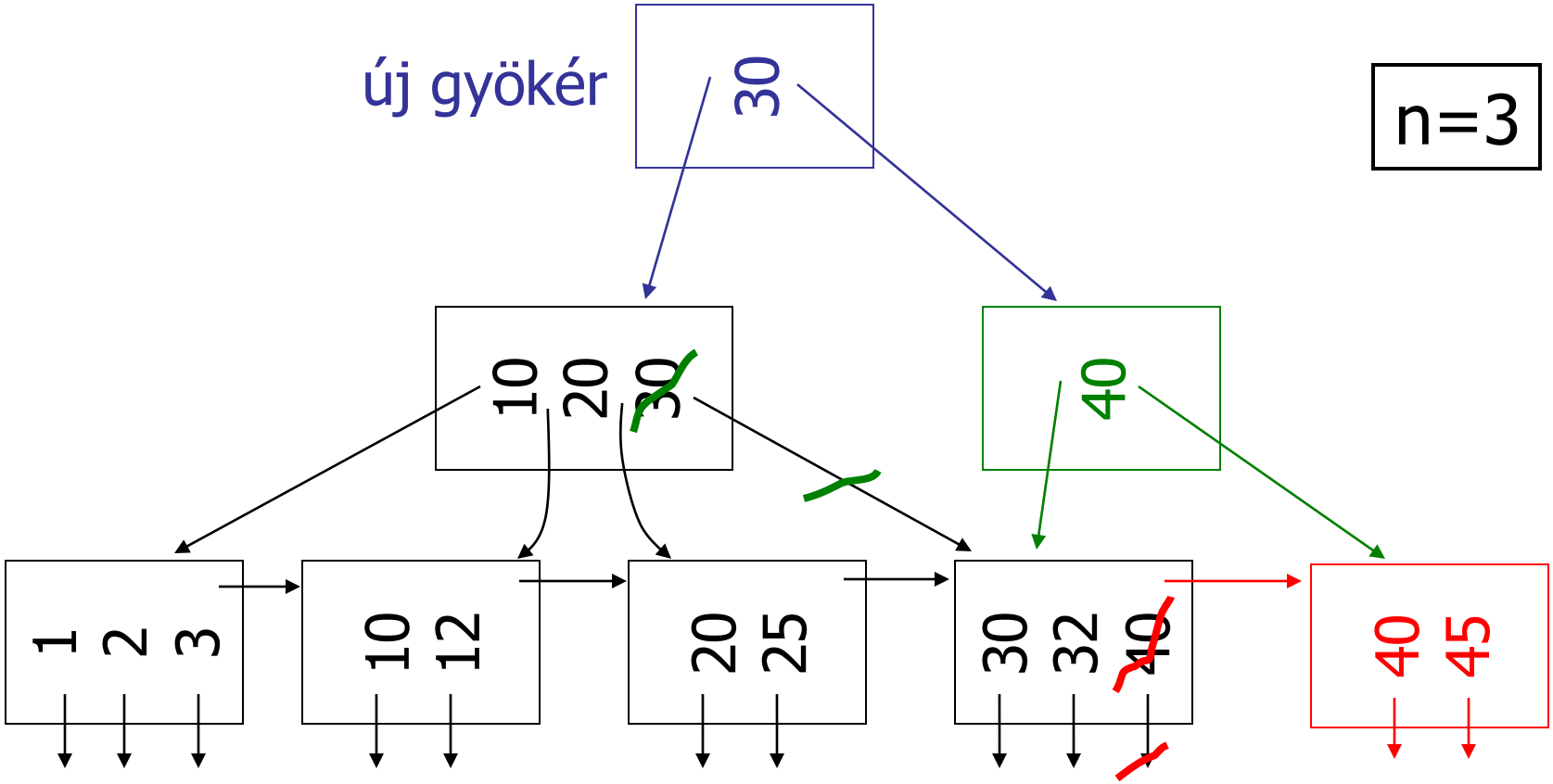
Indexelés

Szúrjuk be a 160-as indexértékű rekordot!



Indexelés

Szúrjuk be a 45-ös indexértékű rekordot!



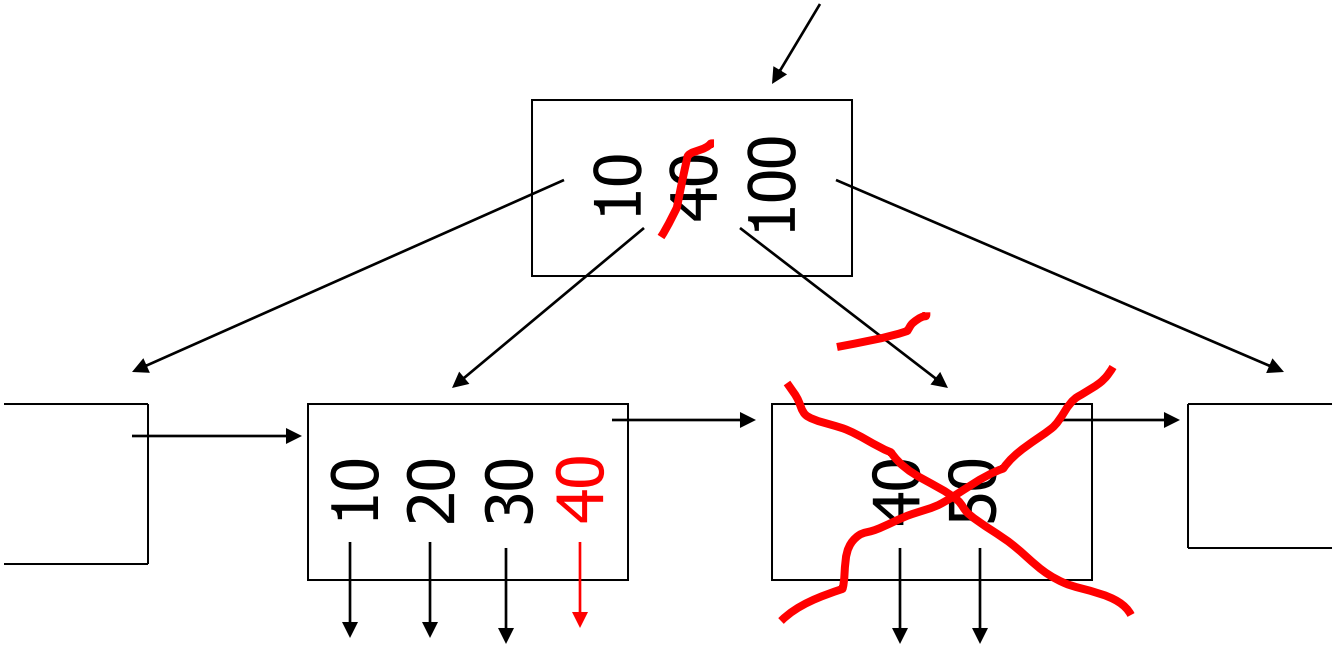
Törlés

- Ha a törlés megtörtént az N pontban, és N még mindig megfelelően telített, akkor készen vagyunk.
- Ha N már nem tartalmazza a szükséges minimum számú kulcsot és mutatót, akkor:
  - ha N valamelyik testvérével összevonható, akkor vonjuk össze. A szülőcsúcsban törlődik ekkor egy elem, s ez további változtatásokat eredményezhet.
  - Ha nem vonható össze, akkor N szomszédos testvérei több kulcsot és mutatót tartalmaznak, mint amennyi a minimumhoz szükséges, akkor egy kulcs-mutató párt áttehetünk N-be. (Baloldali testvér esetén a legutolsó, jobboldali testvér esetén a legelső kulcs-mutató párt.) A változást a szülő csúcson is „regisztrálni kell”.

Indexelés

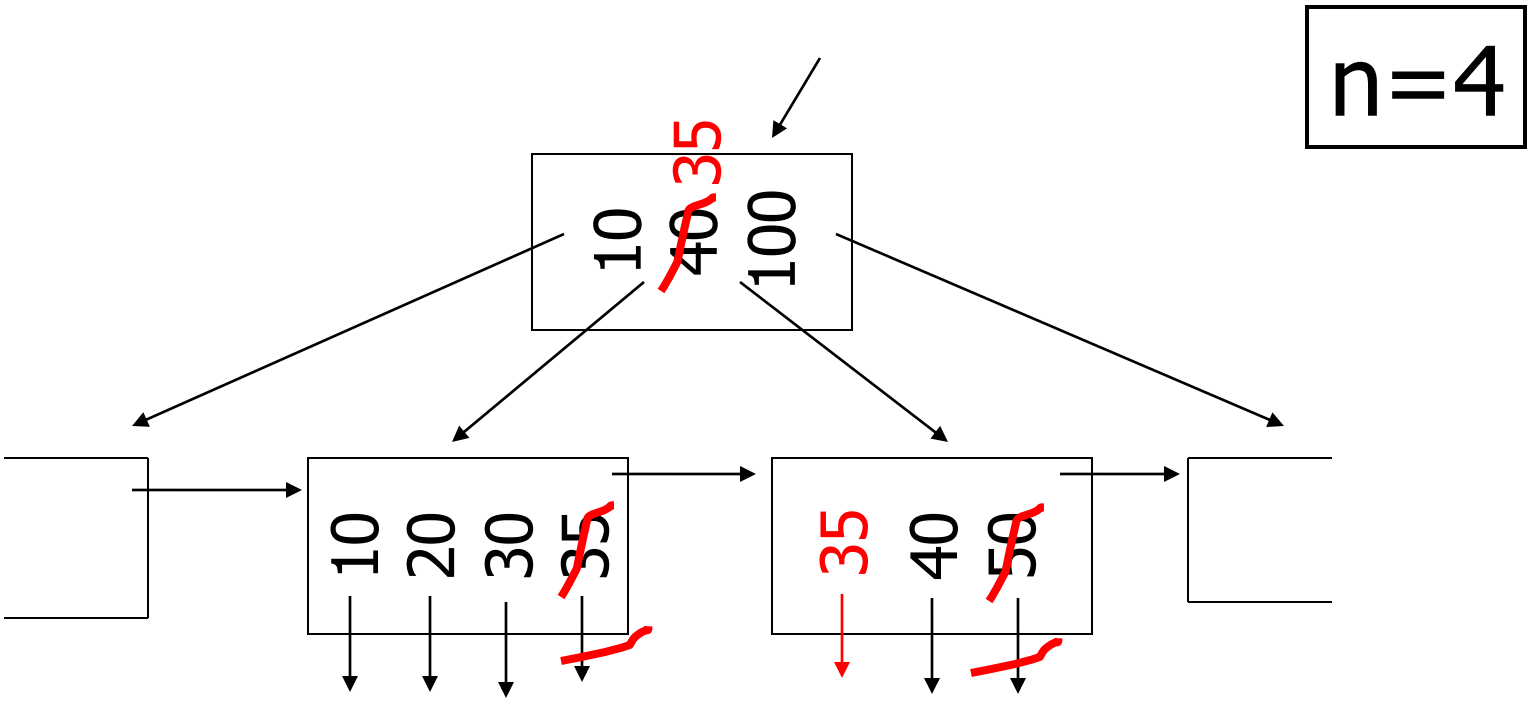
Töröljük az 50-es indexértékű rekordot!

n=4



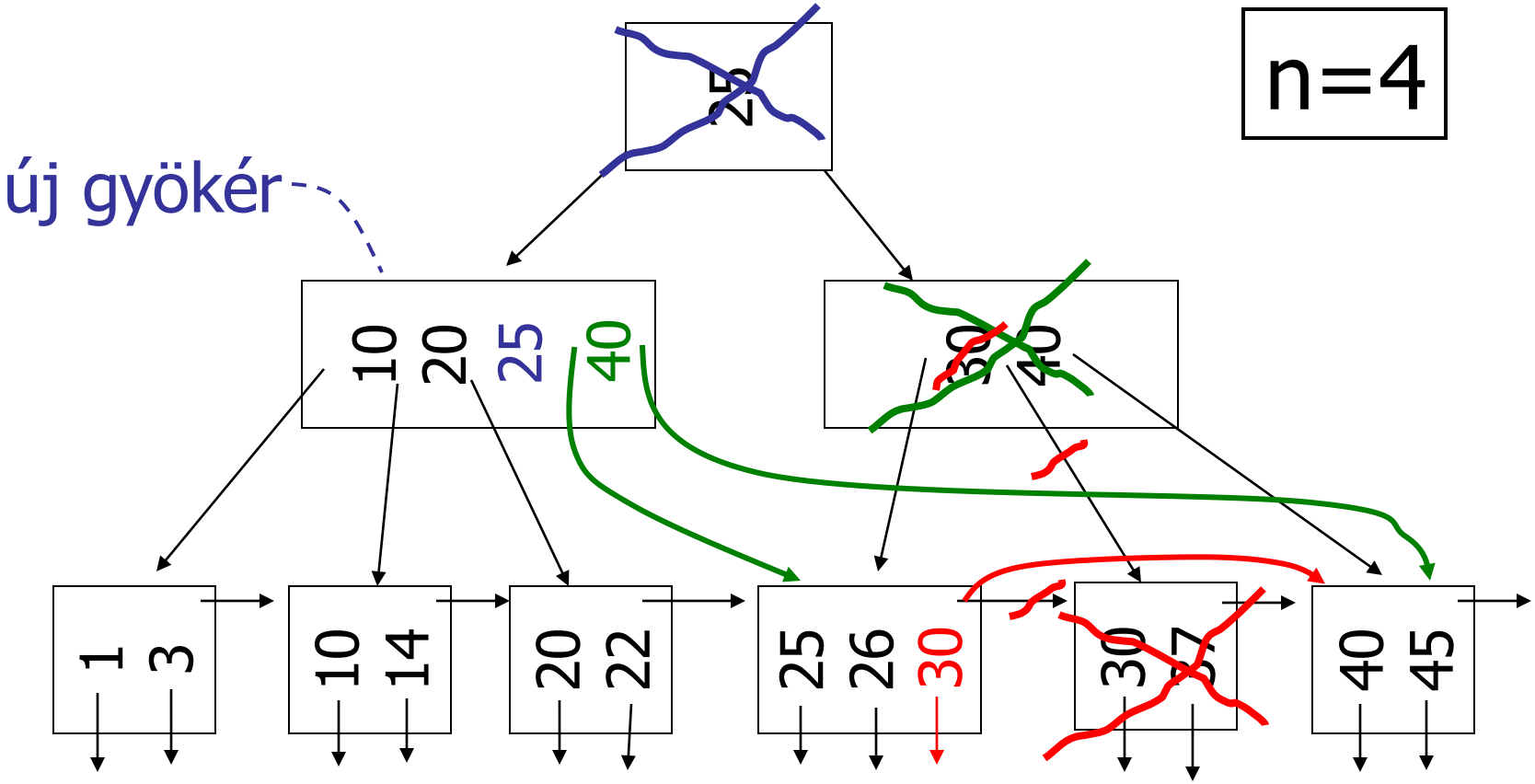
Indexelés

Töröljük az 50-es indexértékű rekordot!



Indexelés

Töröljük a 37-es indexértékű rekordot!



Feladatok és megoldások

5. Legyen a rekordok száma 1 000 000, és az indexelt oszlopban minden érték különböző. Sűrű indexre készítünk ***B-fát***. Egy blokkba 10 rekord vagy (99 kulcs és 100 mutató) fér. Legyen a telítettség 70%, azaz legalább 69 kulcs és 70 mutató szerepel az indexblokkban.

i. Mekkora az adatfájl és az index együttes mérete?

ii. Mennyi a keresés blokkolvasási költsége?

a. Szekvenciális adatállomány, a keresési kulcs alapján rendezve, 10 rekord/blokk. A *B-fa* sűrű index.

b. Ugyanaz mint *a.,* az adatállomány nincs rendezve, 10 rekord/blokk.

c. Ugyanaz mint *a.,* de *B-fa* ritka index.

d. A *B-fa* levelei nem mutatókat (*pointers*) hanem magukat az adatrekordokat, tartalmazza. Egy blokk 10 rekordot tartalmazhat, de a blokk telítettség 70%; azaz 7 rekord/blokk.

e. Az adatállomány szekvenciális fájl, a *B-fa* ritka index, de mindegyik elsődleges blokkhoz tartozik egy túlcsordulási blokk. Általában az elsődleges blokk tele van, a túlcsordulási blokk félig telített. Azonban a rekordok nincsenek rendezve sem az elsődleges sem a túlcsordulási blokkban,

Bitmap indexek

Személy

név	nem	kor	kereset
Péter	férfi	57	350000
Dóra	nő	25	30000
Salamon	férfi	36	350000
Konrád	férfi	21	30000
Erzsébet	nő	20	30000
Zsófia	nő	35	160000
Zsuzsanna	nő	35	160000

érték	vektor
férfi	1011000
nő	0100111

érték	vektor
30000	0101100
160000	0000011
350000	1010000

Bitmap indexek haszna

SELECT COUNT(*)
FROM személy
WHERE nem='nő' and kereset = 160000;

- 0100111 AND 0000011 = 0000011, az eredmény: 2.

SELECT név
FROM személy
WHERE kereset > 100000;

- 0000011 (160000) OR 1010000 (350000) = 1010011, azaz az 1., 3., 6. és 7. rekordokat tartalmazó blokko(ka)t kell beolvasni.

Tömörítés I.

- Ha a táblában n rekord van, a vizsgált attribútum pedig m különböző értéket vehet fel, ekkor, ha m nagy, a bitmap index túl naggyá is nőhet (n*m méret). Ebben az esetben viszont a bitmap indexben az egyes értékekhez tartozó rekordokban kevés az 1-es. A tömörítési technikák általában csak ezeknek az 1-eseknek a helyét határozzák meg.
- Tegyük fel, hogy *i db 0-t* követ egy 1-es. Legegyszerűbb megoldásnak tűnik, ha i-t binárisan kódoljuk. Ám ez a megoldás még nem jó: (a 000101 és 010001 vektorok kódolása is 111 lenne,; 010101).
- Tegyük fel, hogy *i* (*0-k száma*) binárisan ábrázolva j bitből áll. Ekkor először írjunk le j-1 db 1-est, majd egy 0-t, és csak ez után *i* bináris kódolását.
- Példa: a 000101 kódolása: 101101, a 010001 kódolása: 011011.

Tömörítés II.

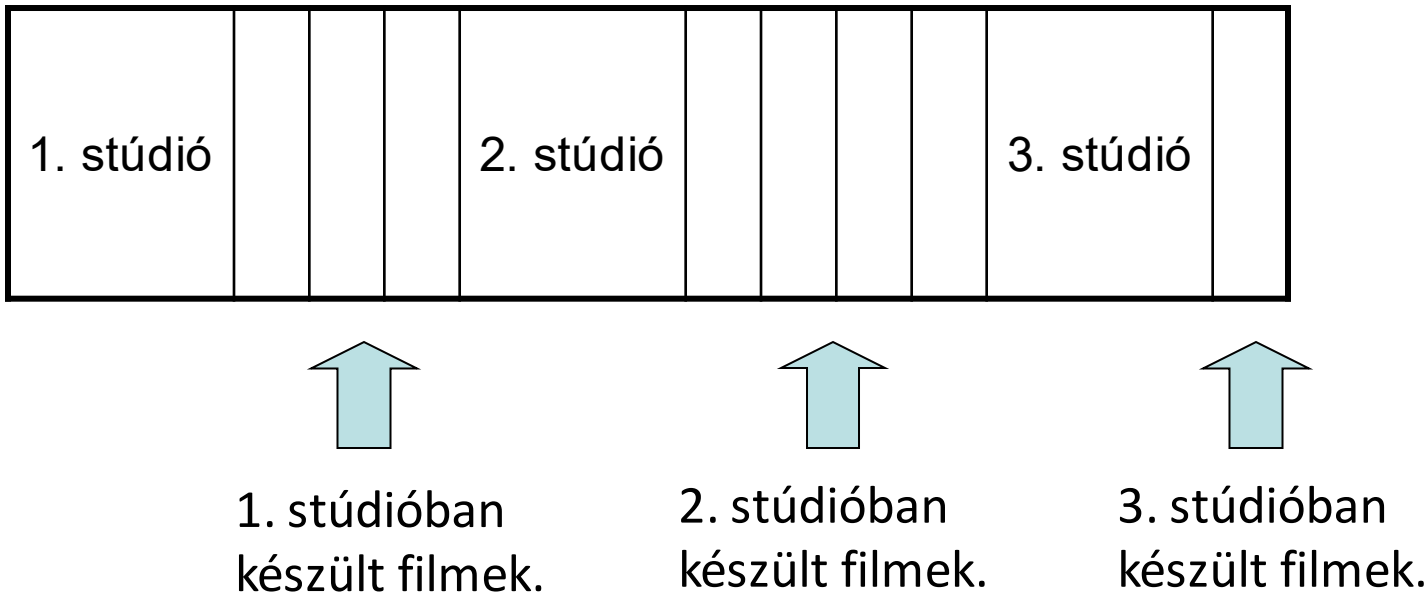
- Állítás: a kód így egyértelművé válik.
- Tegyük fel, hogy m=n, azaz minden rekordérték különböző a vizsgált attribútumban.
- Ekkor, mivel a bitmap indexekben n hosszú rekordokról van szó, egy rekord kódolása legfeljebb $2\log_2 n$. Az indexet alkotó teljes bitek száma pedig $2n\log_2 n$, n^2 helyett.

Klaszter

- Klaszterszervezés esetén a két tábla közös oszlopain megegyező sorok egy blokkban, vagy fizikailag egymás utáni blokkokban helyezkednek el.
- CÉL: összekapcsolás esetén az összetartozó sorokat soros beolvasással megkaphatjuk.

Példa klaszterre

- Film (cím, év, hossz, stúdiónév)
- Stúdió (név, cím, elnök)



Pl. gyakori a: SELECT cím, év
FROM film, stúdió
WHERE cím LIKE '%Moszkva%' AND stúdiónév = név; jellegű lekérdezés.