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 = $\lfloor b/l \rfloor$ - alsó egészrész)
- **B=** $\lceil$ **T/bf** $\rceil$
- **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 **blokkláncok**ba (**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.**
- Miért nem érdemes nagyon nagy K-t választani?
- **Tárméret**: **B**, ha minden blokk nagyjából tele.
- Nagy K esetén sok olyan blokklánc lehet, amely egy blokkból fog állni, és a blokkban is csak 1 rekord lesz. Ekkor a keresési idő: 1 blokkbeolvasás, de B helyett T számú blokkban tároljuk az adatokat.
- **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:

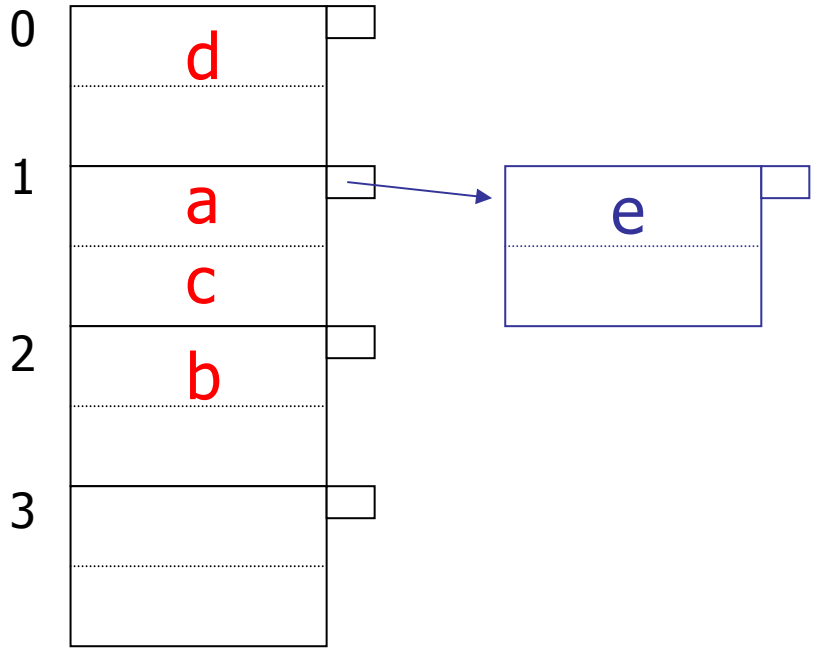
h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0

h(e) = 1



Indexelés

Töröljük a következő hasító értékkel rendelkező rekordokat!

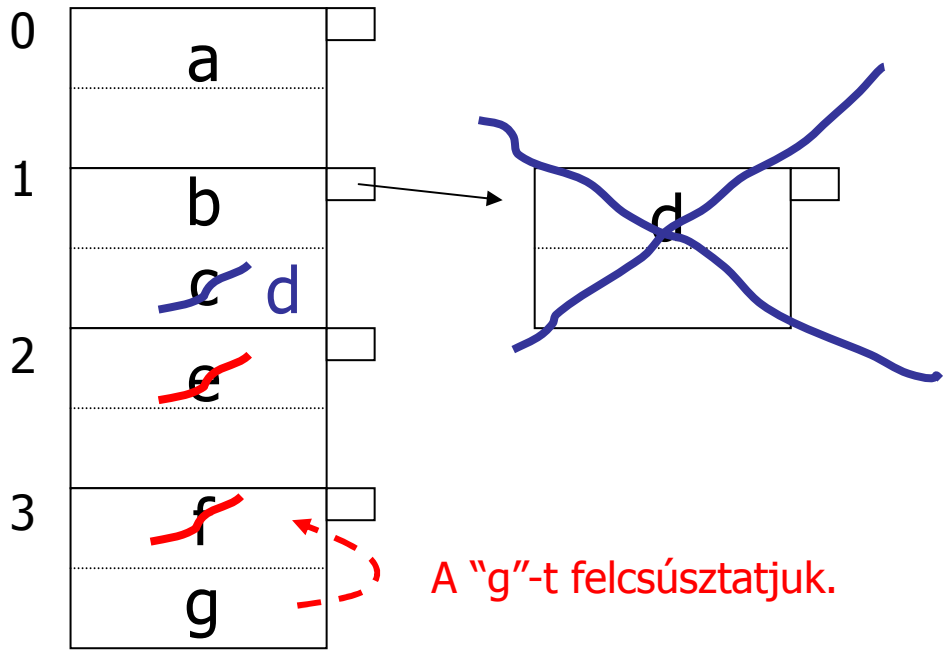
(A megüresedett túlcsordulási blokkokat megszüntetjük.)

Delete:

e

f

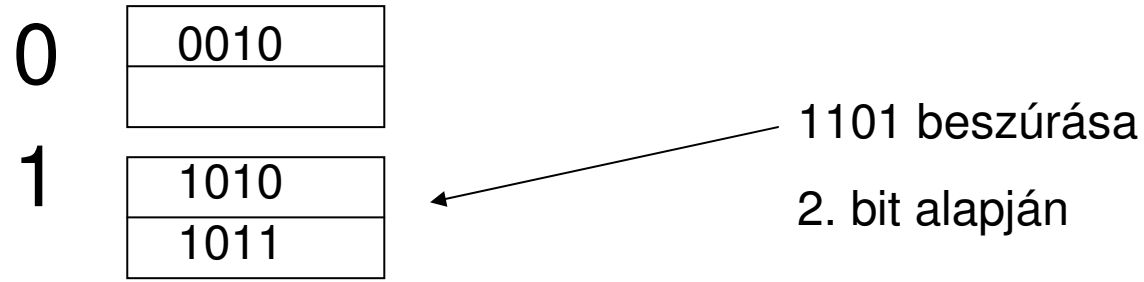
c

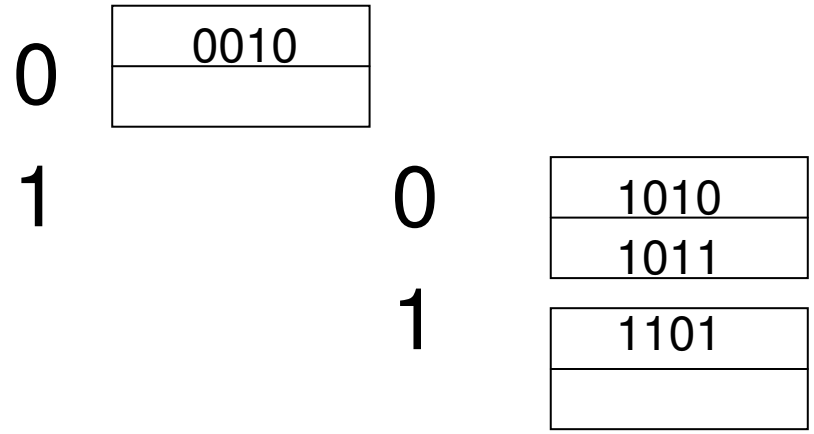


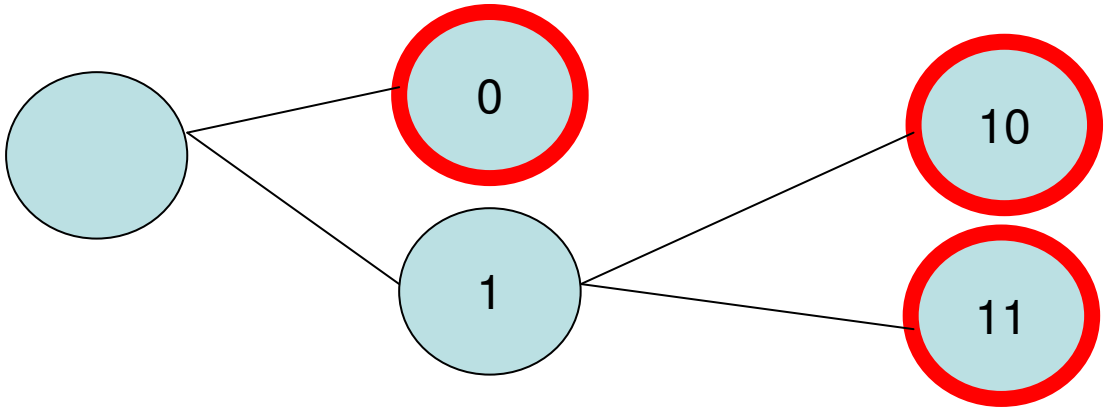
Indexelés

- **Dinamikus hasító indexek:**
  - **kiterjeszthető (expandable)**
  - **lineáris**
- Előre nem rögzítjük a kosarak számát, a kosarak száma beszúráskor, törléskor változhat.
- **Kiterjeszthető hasító index:**
- Minden kosár 1 blokkból áll. Keresési költség: 1.
- Legyen **k > log (a rekordok várható számának felső korlátja)**,
  - azaz k hosszú bináris sorozatból több van, mint ahány rekord
- **A h hasító függvény értéke egy k hosszú bináris sorozat.**
- Minden kosárhoz tartozik egy legfeljebb k hosszú bináris sorozat (kódszó).
- A kosarakhoz rendelt kód **prefix kód**. A maximális kód hossza legyen i.
- A h(K) k hosszú kódnak vegyük az **i hosszú elejét**, és **azt kosarat, amelynek kódja a h(K) kezdő szelete**. Ha van hely a kosárban, tegyük bele a rekordot, ha nincs, akkor nyissunk egy új kosarat, és a következő bit alapján osszuk ketté a telített kosár rekordjait. Ha ez a bit mindegyikre megegyezik, akkor a következő bitet vesszük a szétosztáshoz, és így tovább.

Indexelés


0
0010
1
1010
1011
1101 beszúrása
2. bit alapján


0
0010
1
0
1010
1011
1
1101


0
1
10
11

Indexelés

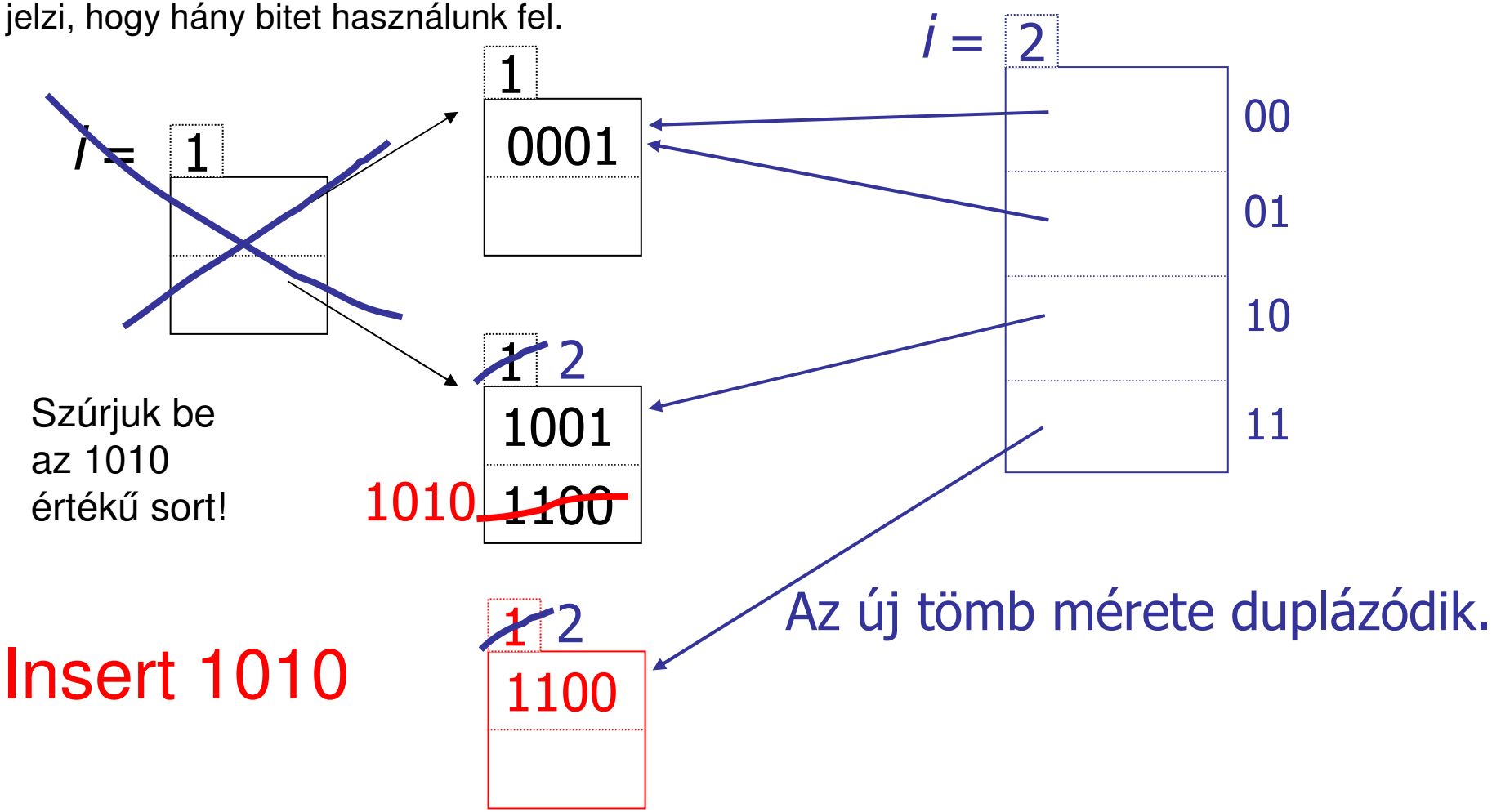
- A bináris fa levelei a kosárblokkok kódszavai. A hasító függvény értékéből annyi bitet használunk, ahányadik szinten szerepel a levél.
- A gráfot a memóriában tárolhatjuk.

Probléma: Ha az új sorok hasító értékének eleje sok bitben megegyezik, akkor **hosszú ágak** keletkezhetnek.

(Nincs kiegyensúlyozva a fa.)

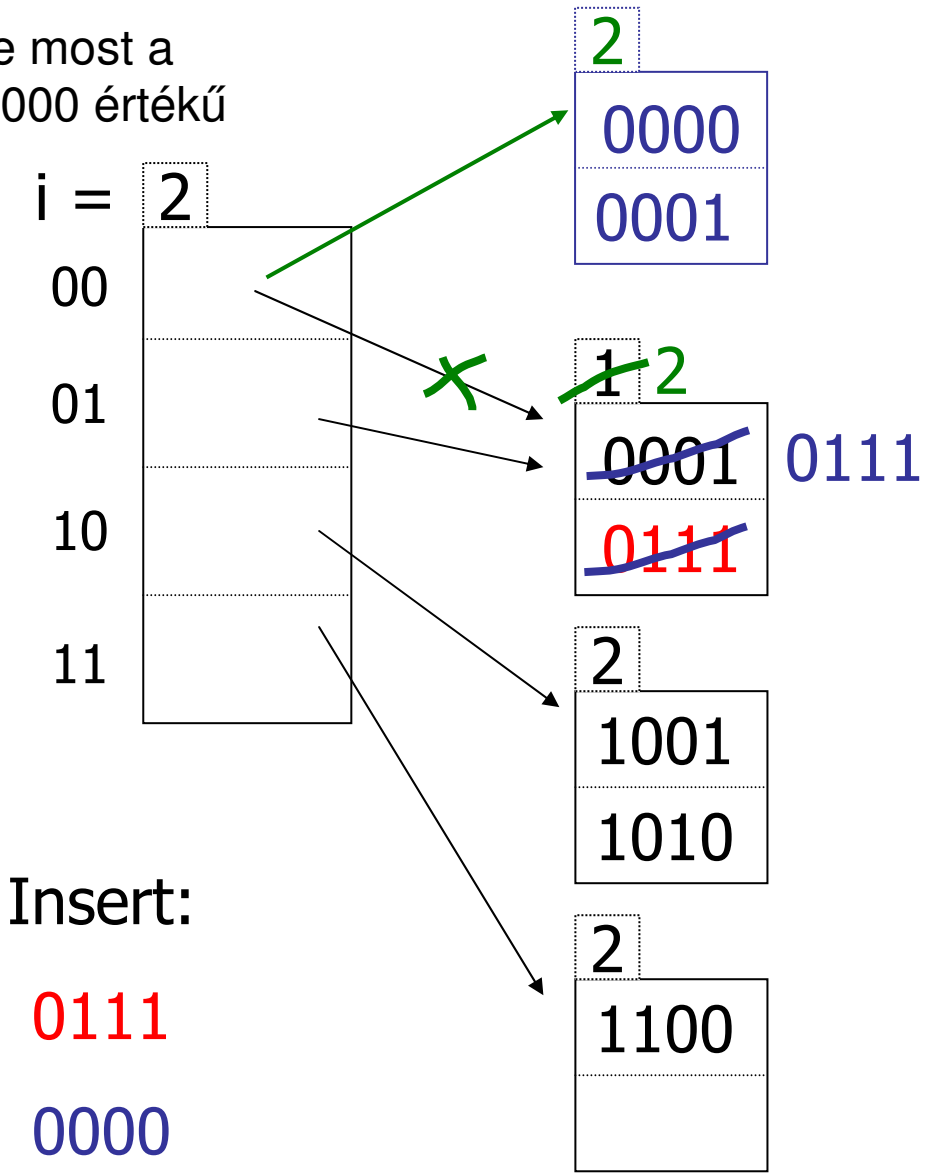
Indexelés

A bináris gráfot teljessé is tehetjük. A gráfot egy tömbbel ábrázolhatjuk. Ekkor minden kosár azonos szinten lesz, de közös blokkjai is lehetnek a kosaraknak. Túlcsordulás esetén **a kosarak száma duplázódik.** Legyen például h(k) 4 bites és 2 rekord férjen egy blokkba. Az i jelzi, hogy hány bitet használunk fel.



Indexelés

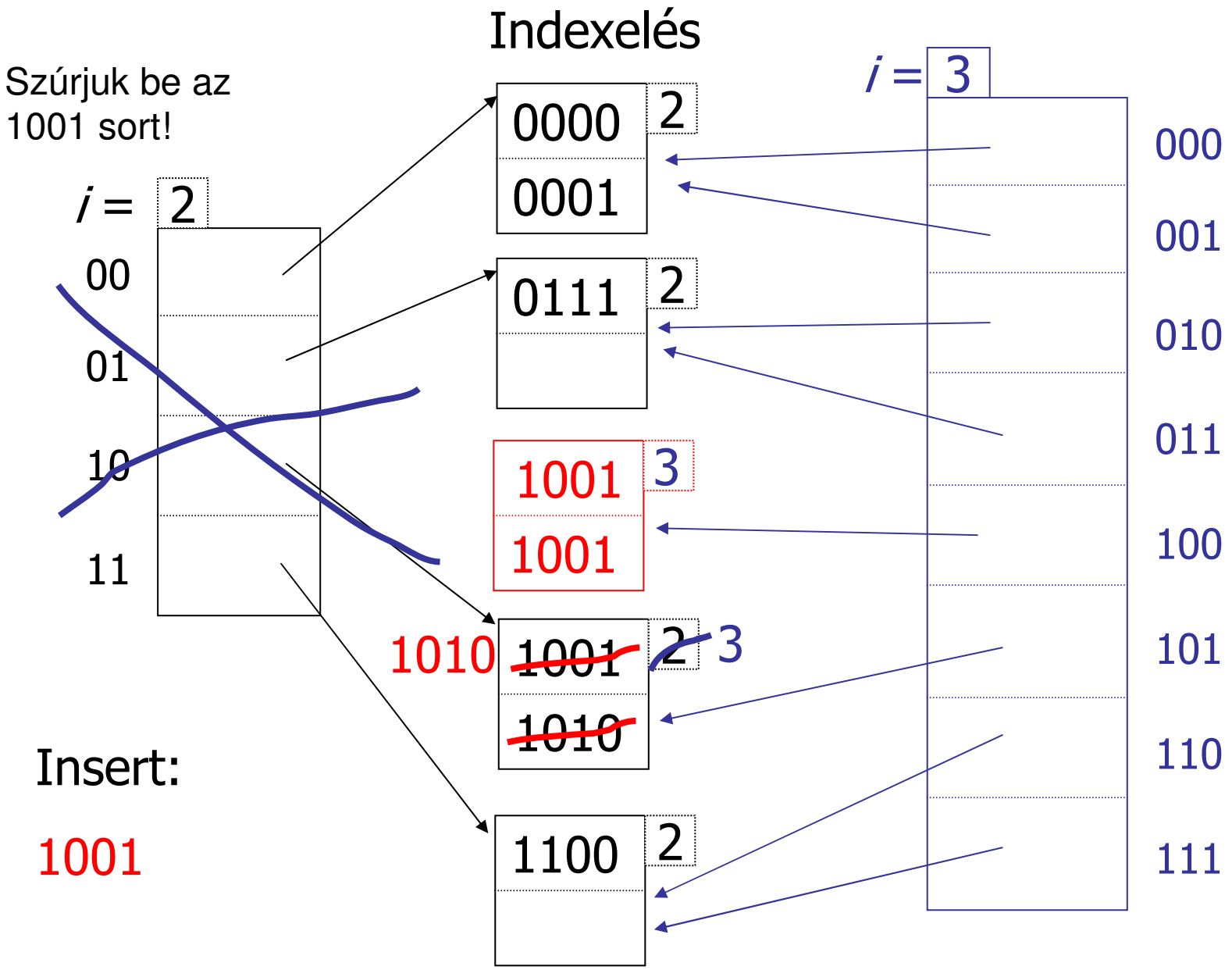
Szúrjuk be most a 0111 és 0000 értékű sorokat!



Insert:

0111

0000


Indexelés
Szúrjuk be az
1001 sort!
i = 2
00
01
10
11
0000
0001
2
0111
2
1001
1001
3
1010
1001
1010
2
3
1100
2
Insert:
1001
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.

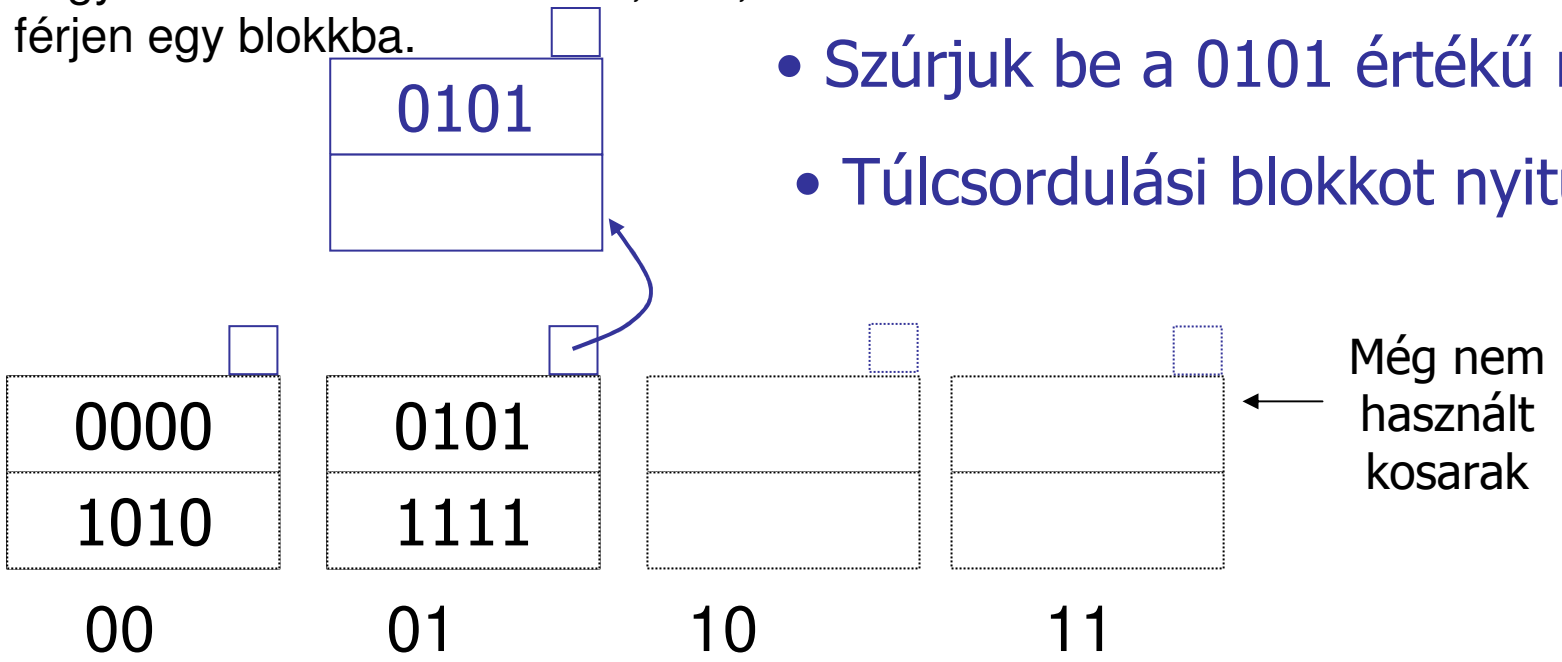
(rekordok száma/kosarak száma > küszöb)

- **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.

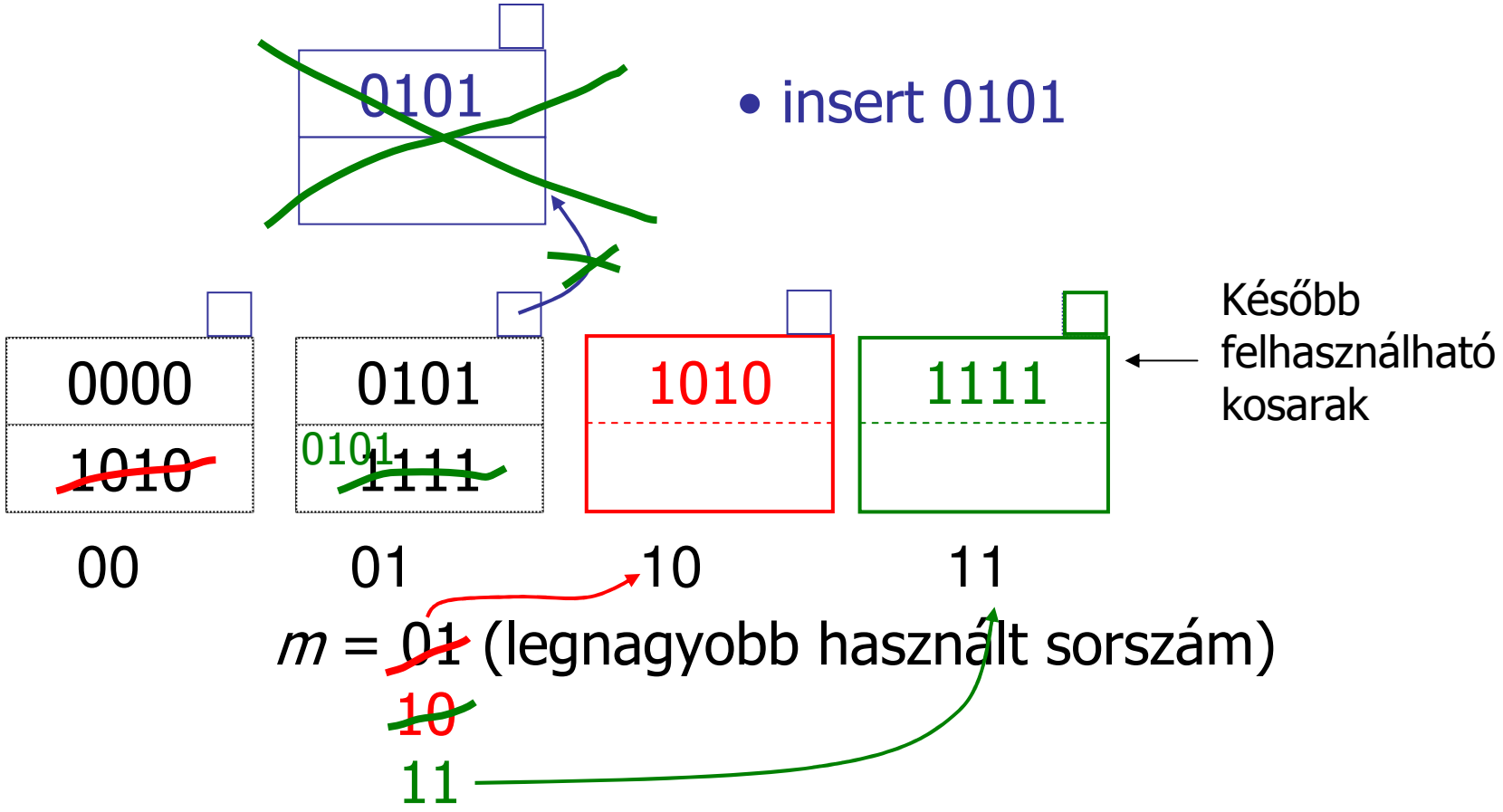


m = 01 (a legnagyobb sorszámú blokk sorszáma)

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

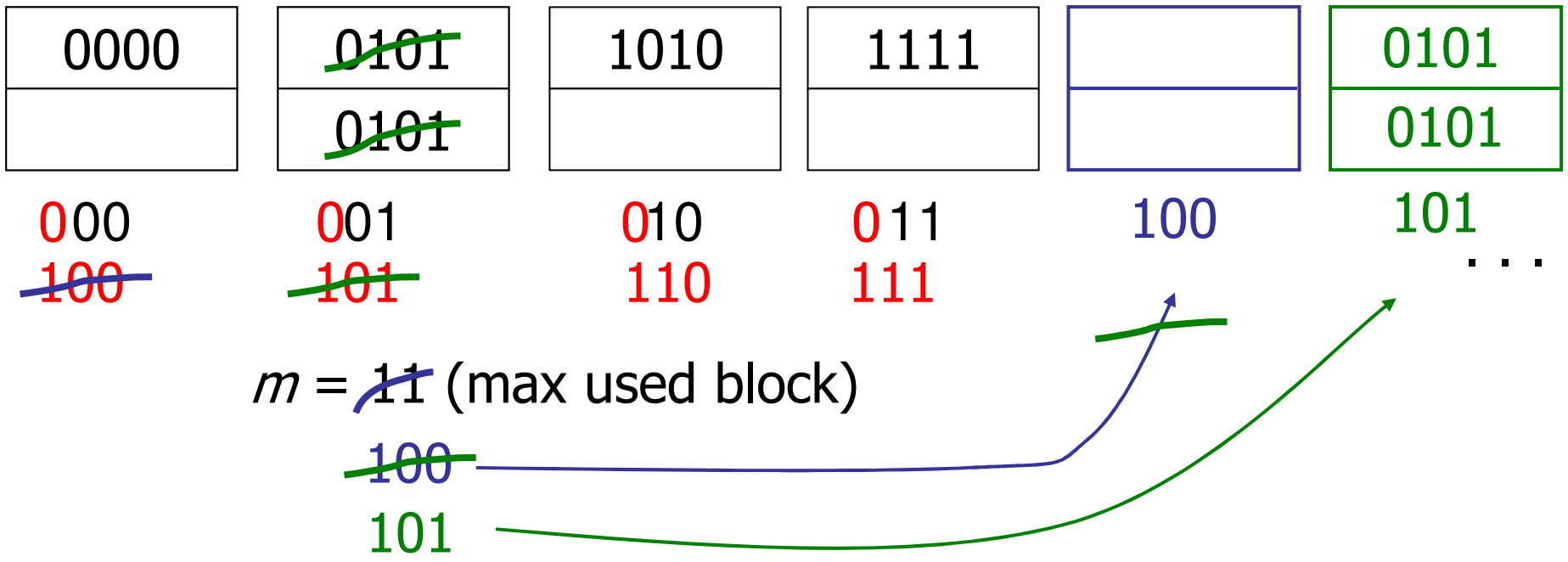


Tegyük fel, hogy átléptük a küszöbszámot, és ezért új kosarat kell nyitni, majd az első bitben különböző sorszámú kosárból át kell tenni ebbe az egyező végződésű rekordokat.

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.

$i = 2$ 3



Indexelés

- **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 **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 --

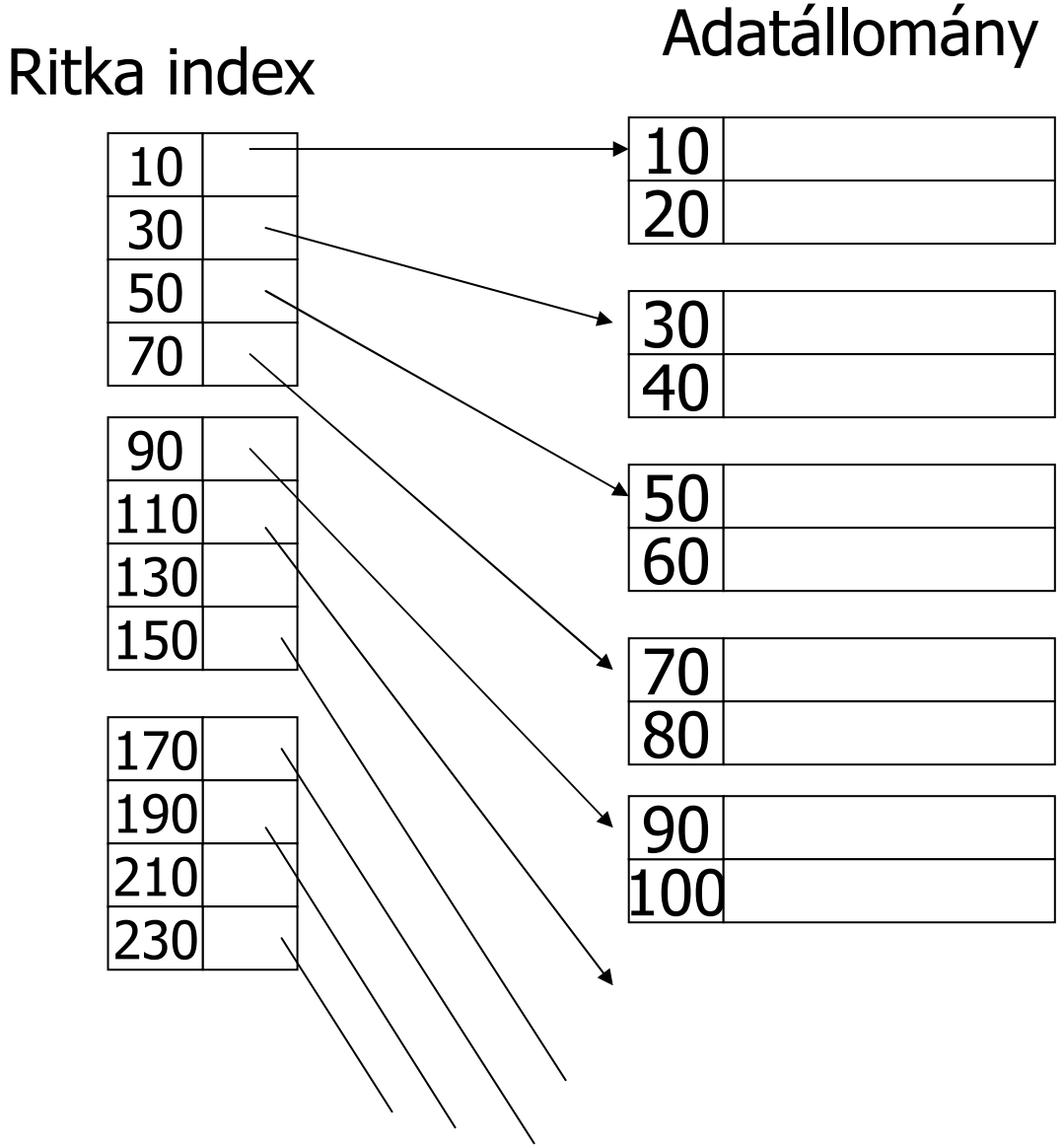
Indexelés

- **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

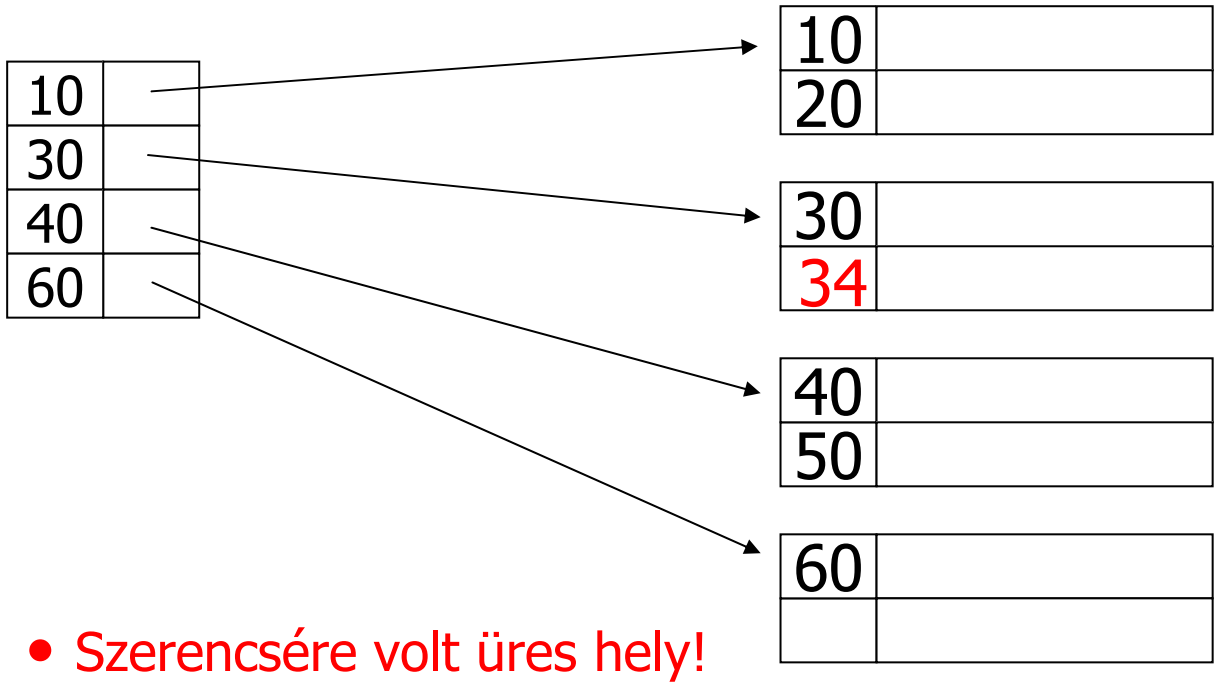
Az adatfájl rendezett, ezért elég a blokkok első rekordjaihoz indexrekordokat tárolni.



Indexelés

Beszúrás ritka index esetén:

Vigyük be a 34-es rekordot!

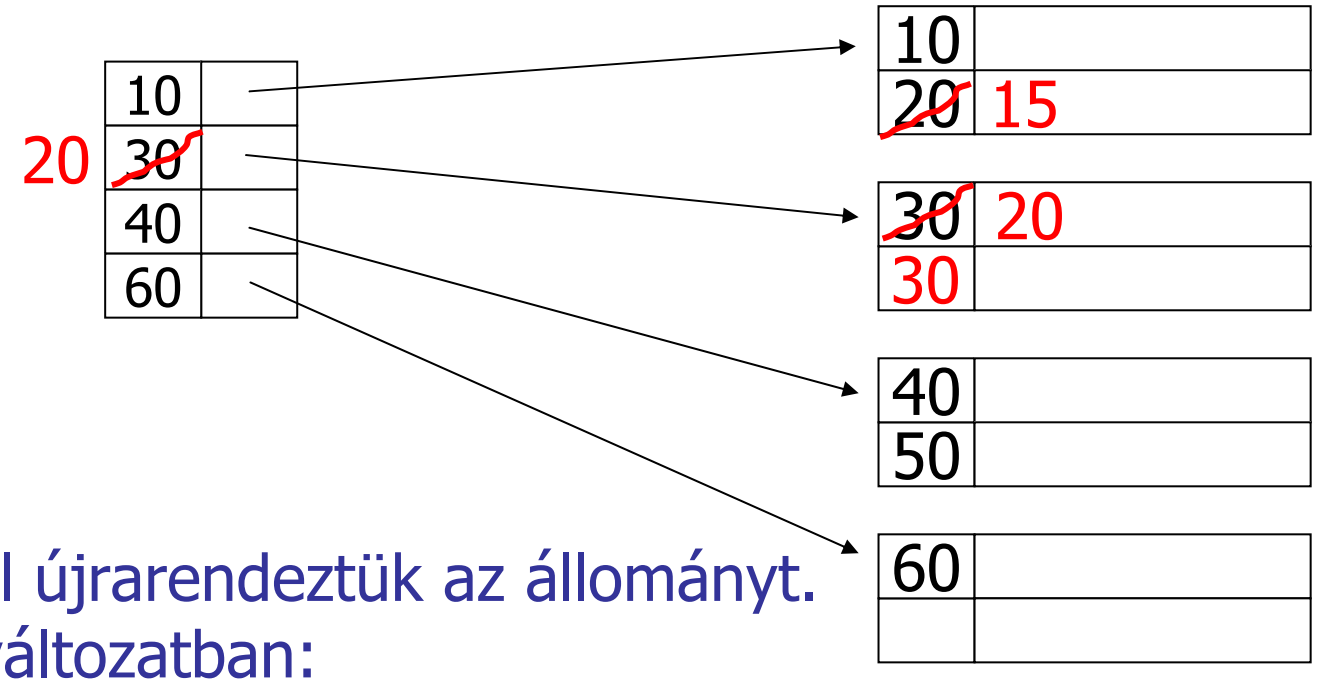


- Szerencsére volt üres hely!

Indexelés

Beszúrás ritka index esetén:

Vigyük be a 15-ös rekordot!

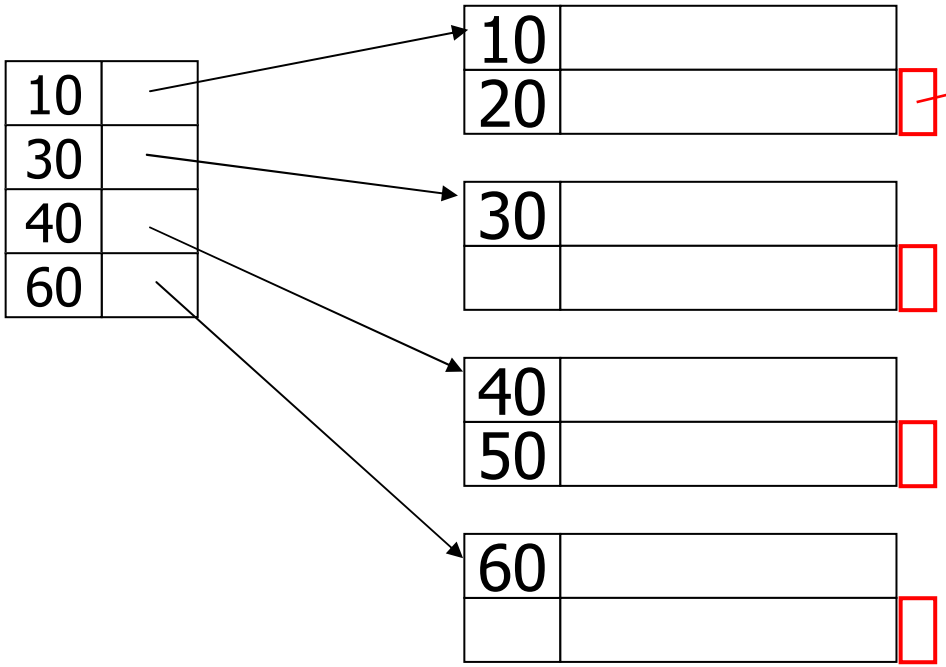


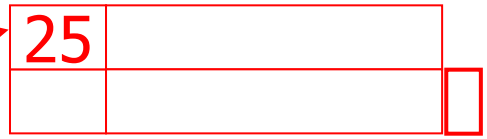
- Azonnal újrarendeztük az állományt.
- Másik változatban:
  - túlcsordulási blokkot láncolunk a blokkhoz

Indexelés

Beszúrás ritka index esetén:

Vigyük be a 25-ös rekordot!



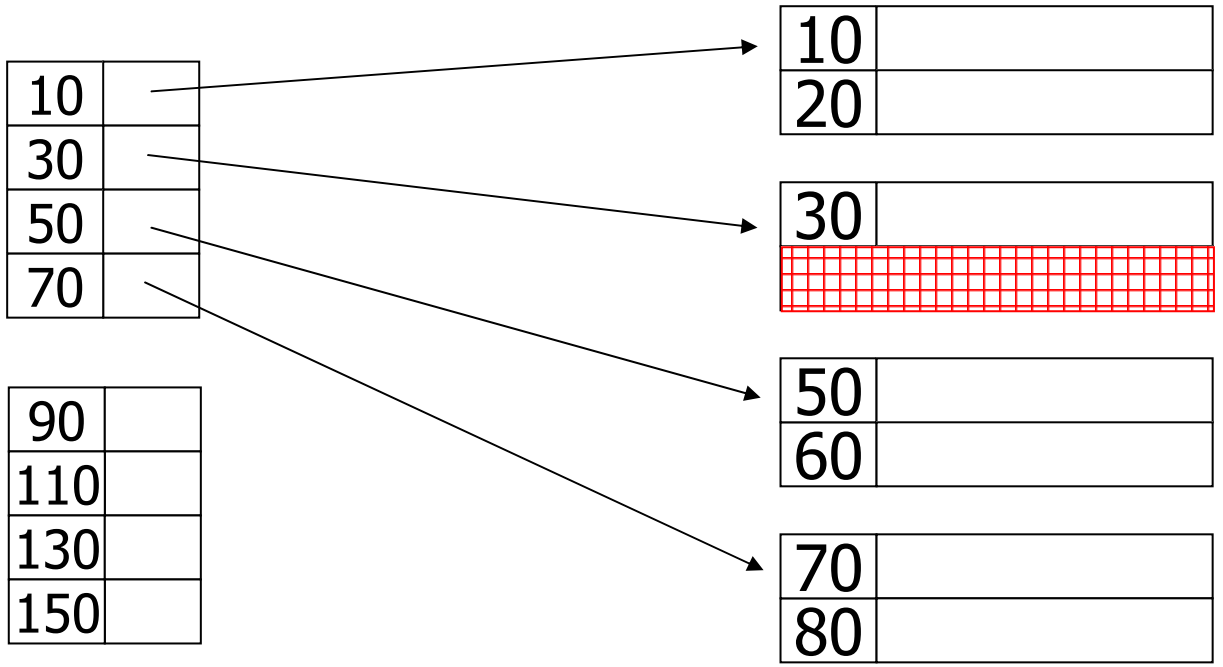


Túlcsordulási blokkot nyitunk, és későbbre halasztjuk az újrarendezést.

Indexelés

Törlés ritka indexből:

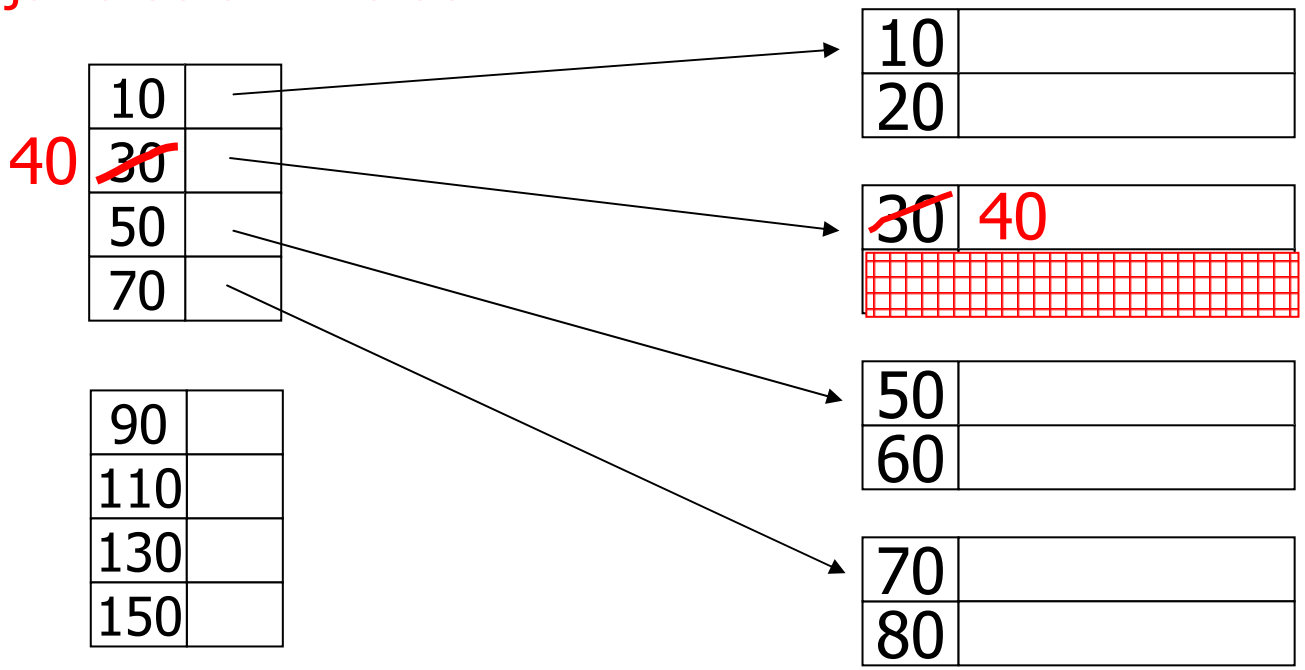
Töröljük a 40-es rekordot!



Indexelés

Törlés ritka indexből:

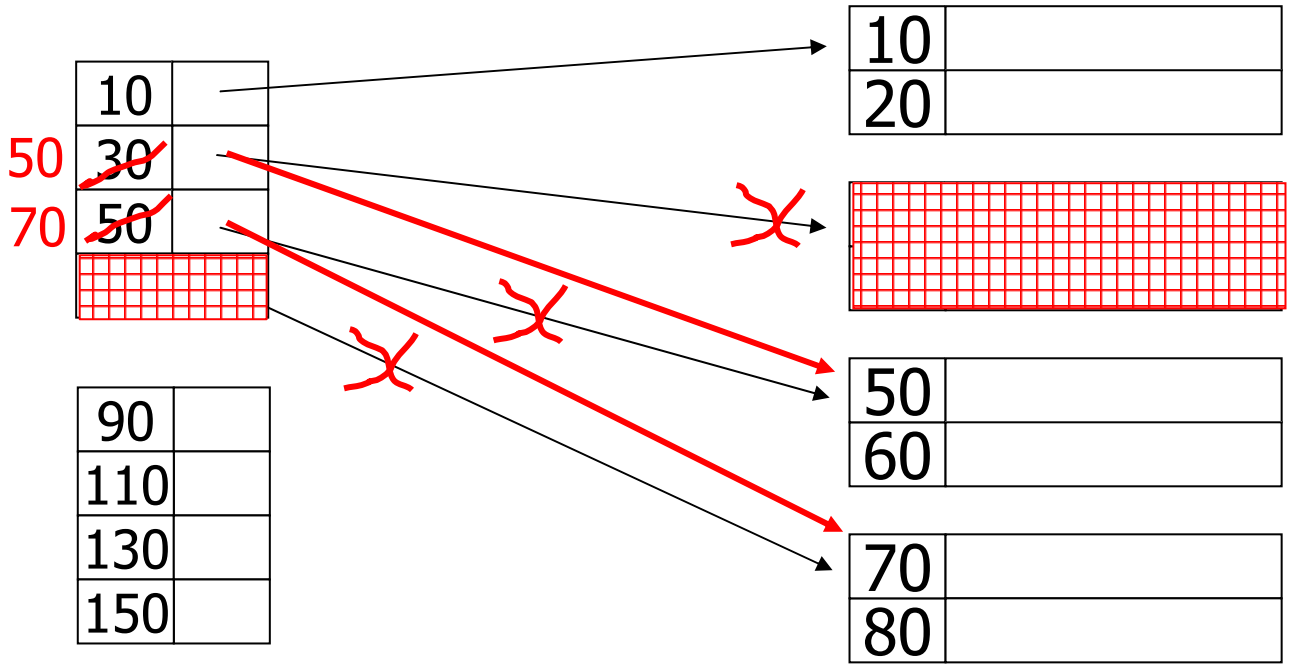
Töröljük a 30-as rekordot!



Indexelés

Törlés ritka indexből:

Töröljük a 30-as és 40-es rekordot!



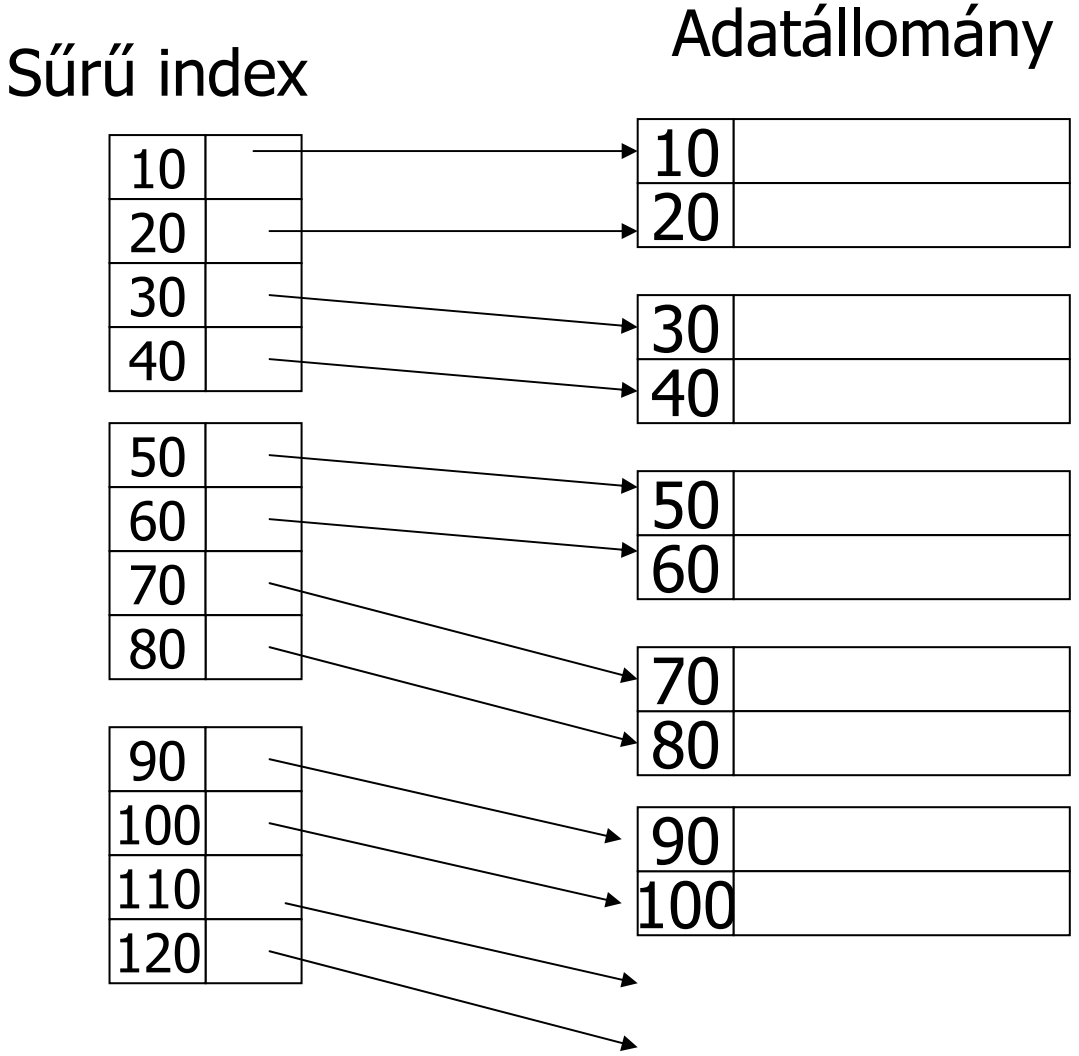
Indexelés

- **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

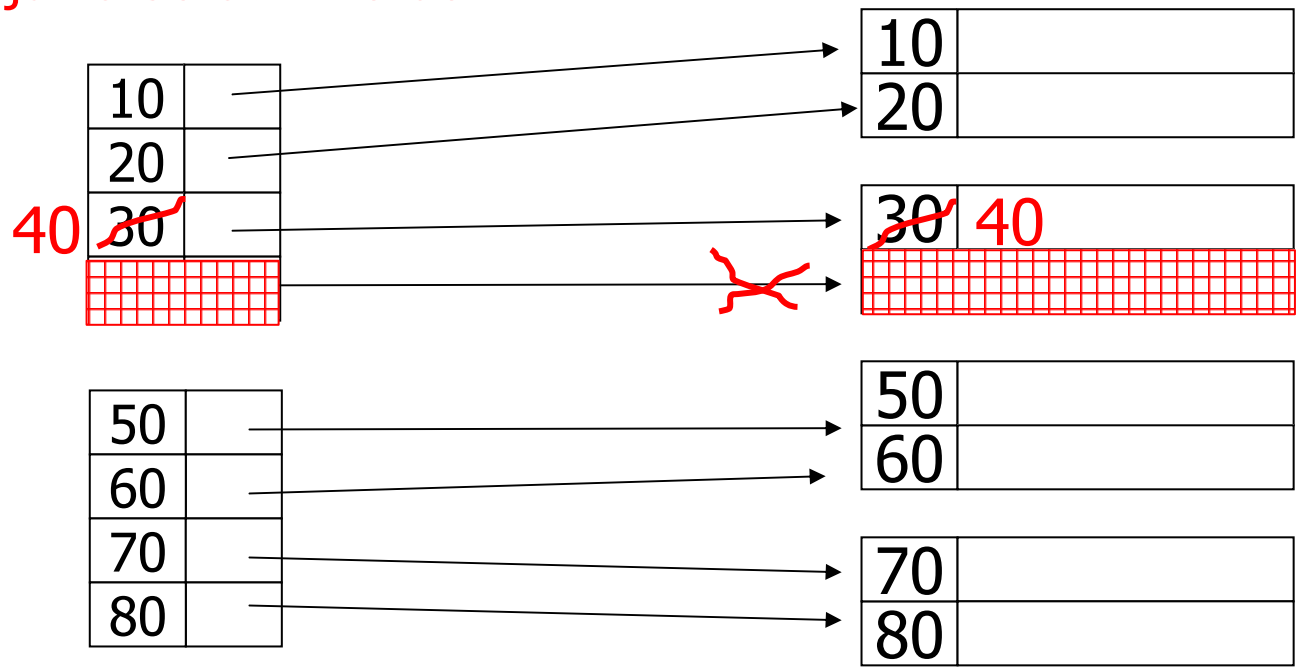
Minden rekordhoz tartozik indexrekord.



Indexelés

Törlés sűrű indexből:

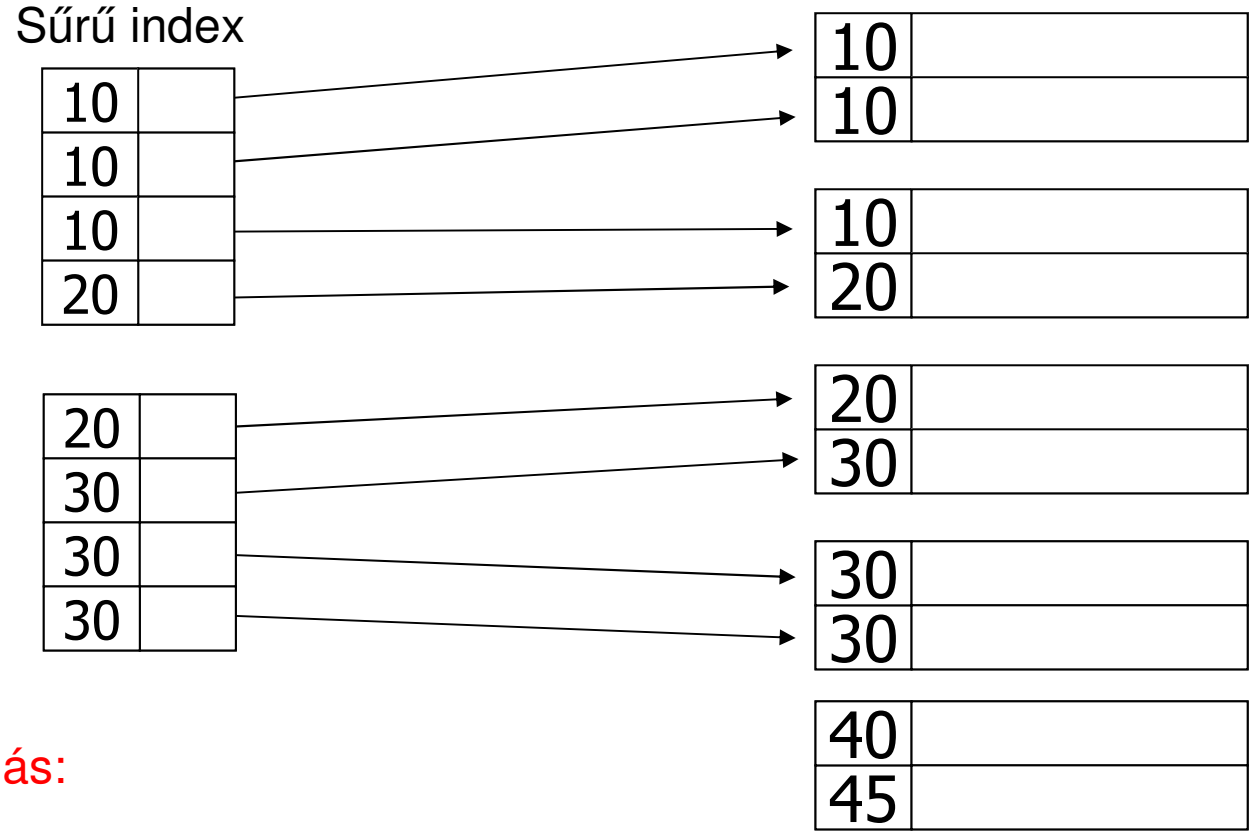
Töröljük a 30-as rekordot!



Indexelés

Mi történik, ha egy érték többször is előfordulhat?

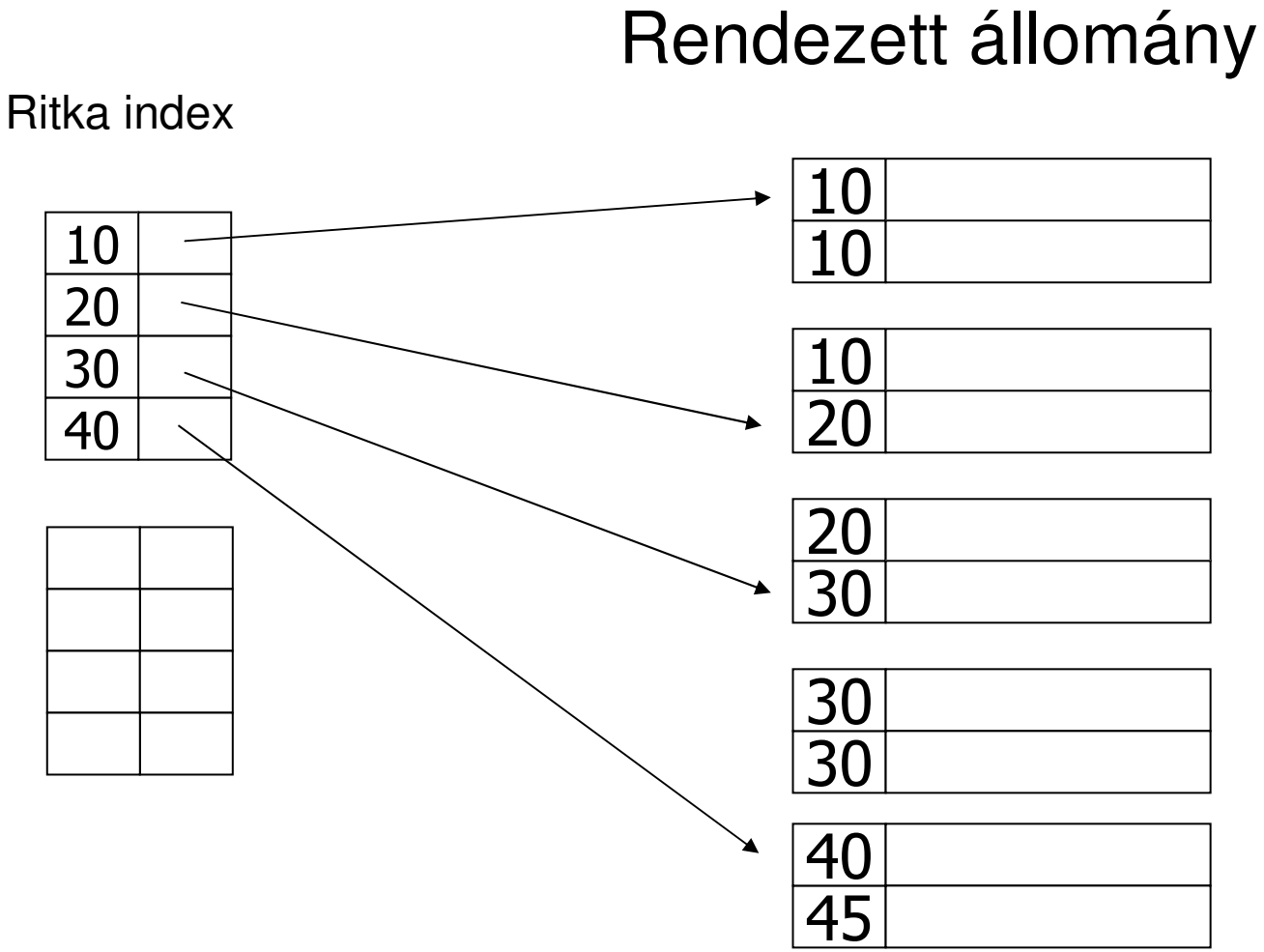
Több megoldás is lehetséges. Először tegyük fel, hogy rendezett az állomány.



1. megoldás:

Minden rekordhoz tárolunk egy indexrekordot.

Indexelés

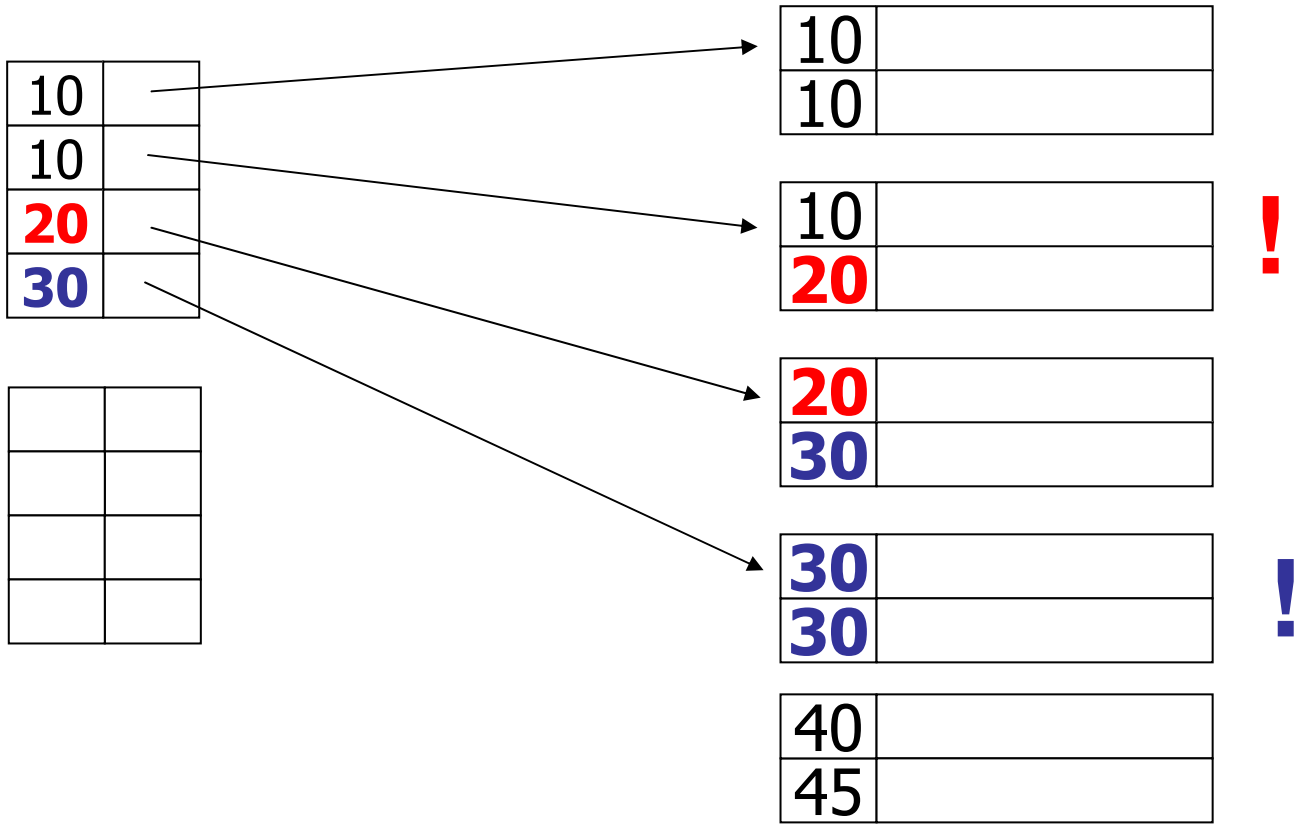


2. megoldás:

Rendezett állomány esetén csak az első előforduláshoz tárolunk egy indexrekordot.

Indexelés

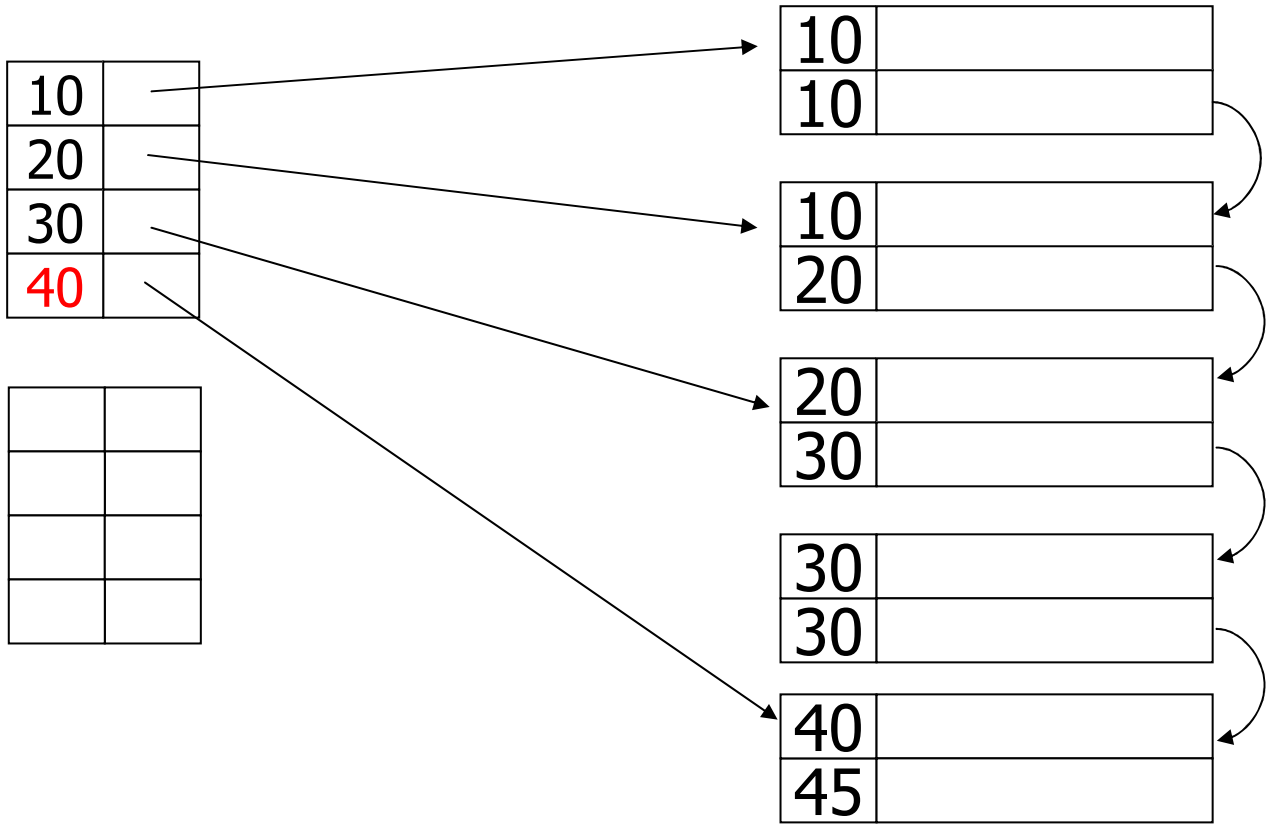
Vigyázat! A fedőértéknek megfelelő blokk előtti és utáni blokkokban is lehetnek találatok. Például, ha a 20-ast vagy a 30-ast keressük.



Indexelés

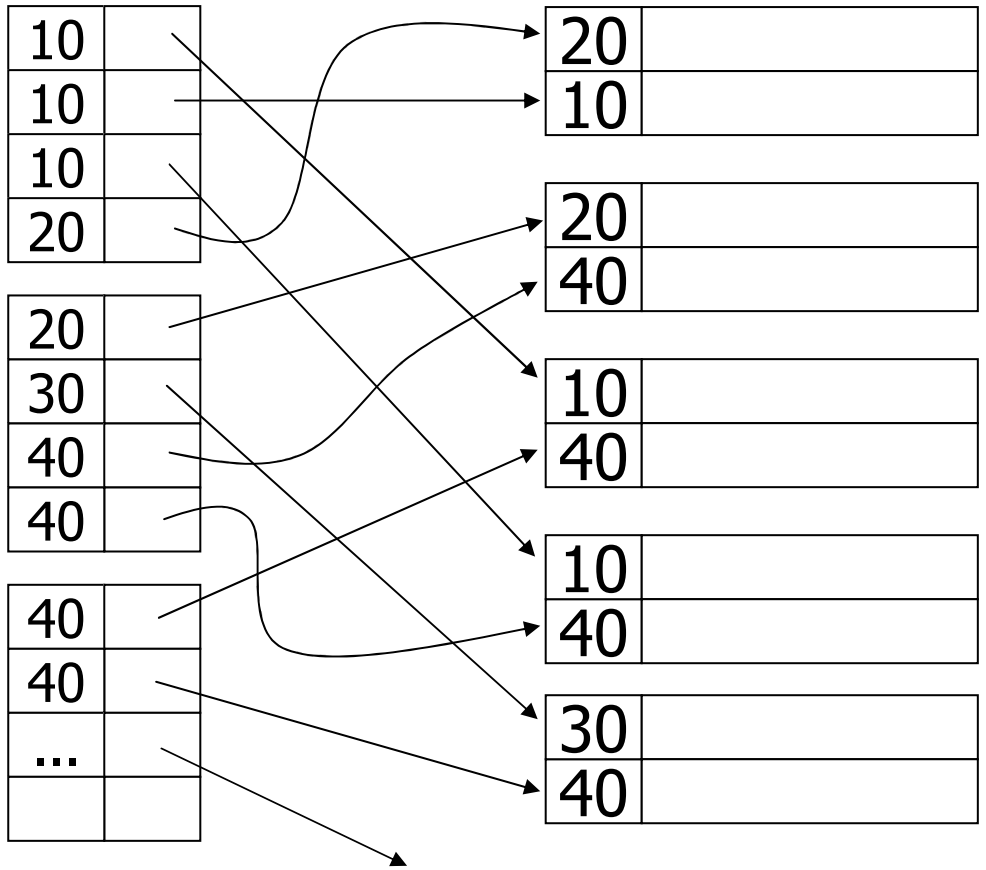
3. megoldás:

Rendezett állomány esetén nem az első rekordot, 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:

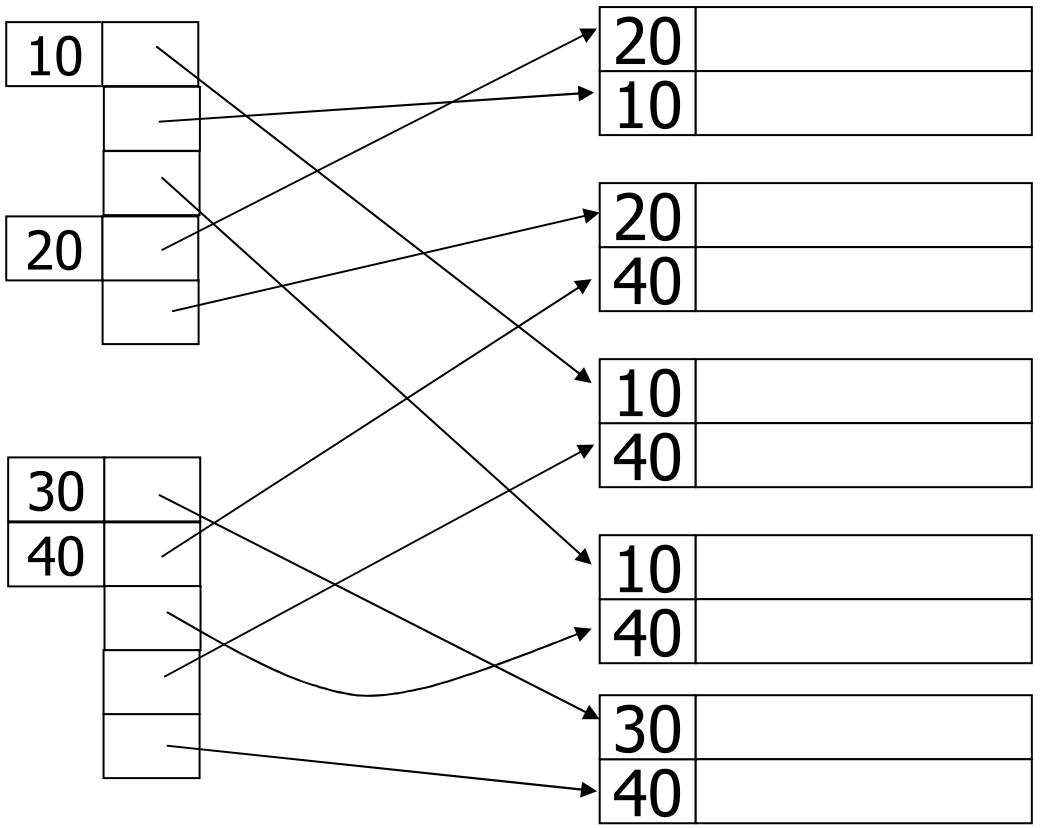


Indexelés

Egy lehetséges megoldás, hogy az indexrekordok szerkezetét módosítjuk:

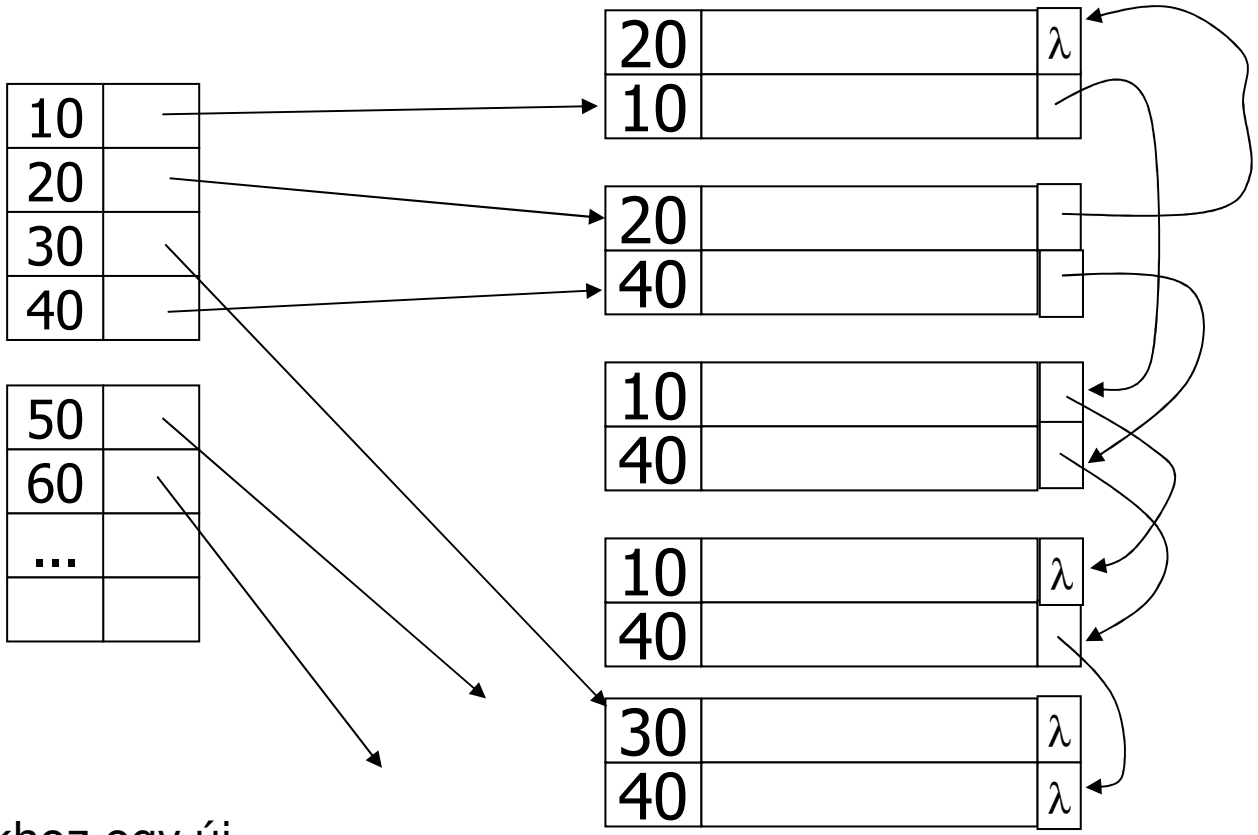
(indexérték, mutatóhalmaz)

Probléma: változó hosszú indexrekordok keletkeznek



Indexelés

Összeláncolhatjuk az egyforma értékű rekordokat.

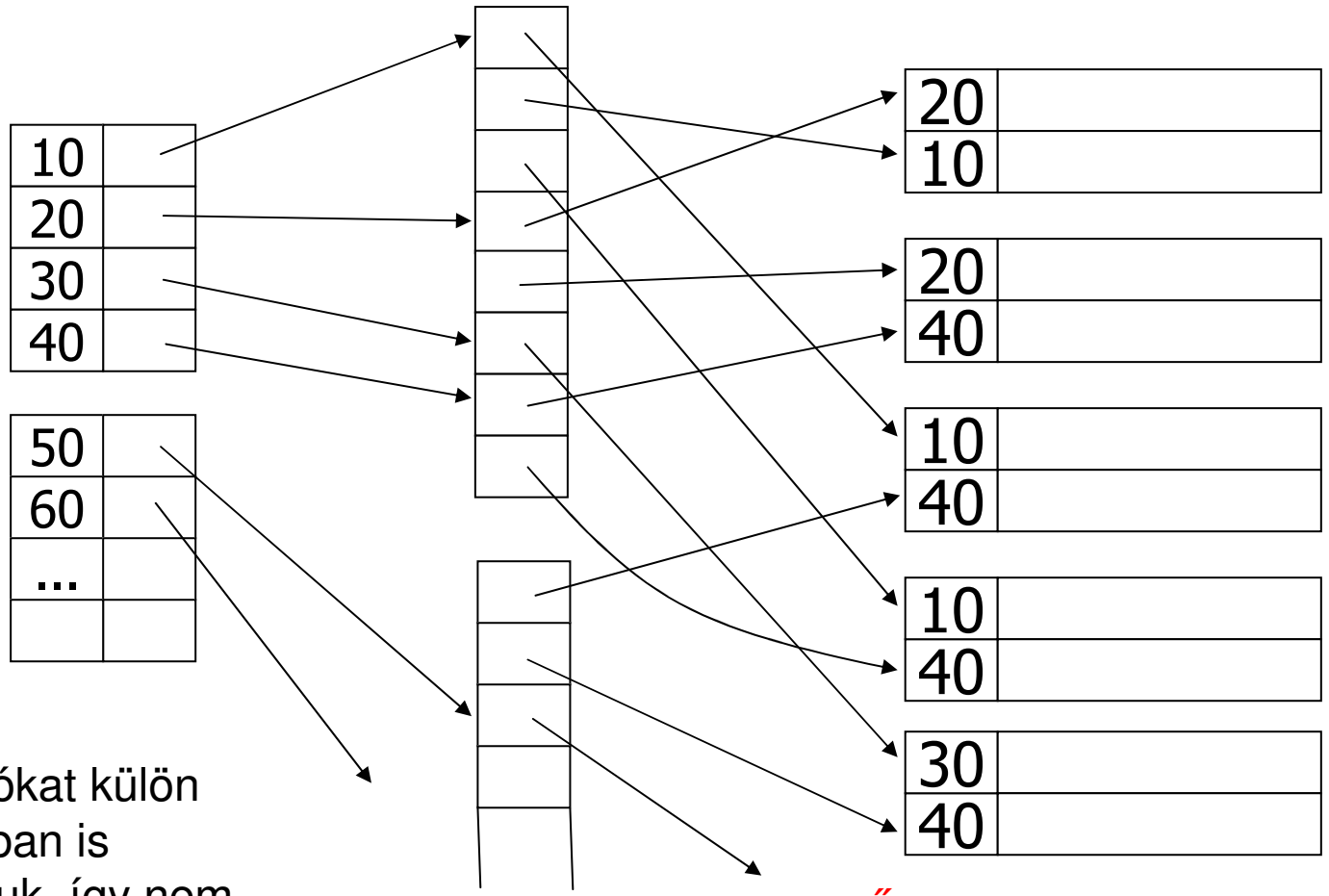


Probléma:

- a rekordokhoz egy új, mutató típusú mezőt kell adnunk

- követni kell a láncot

Indexelés



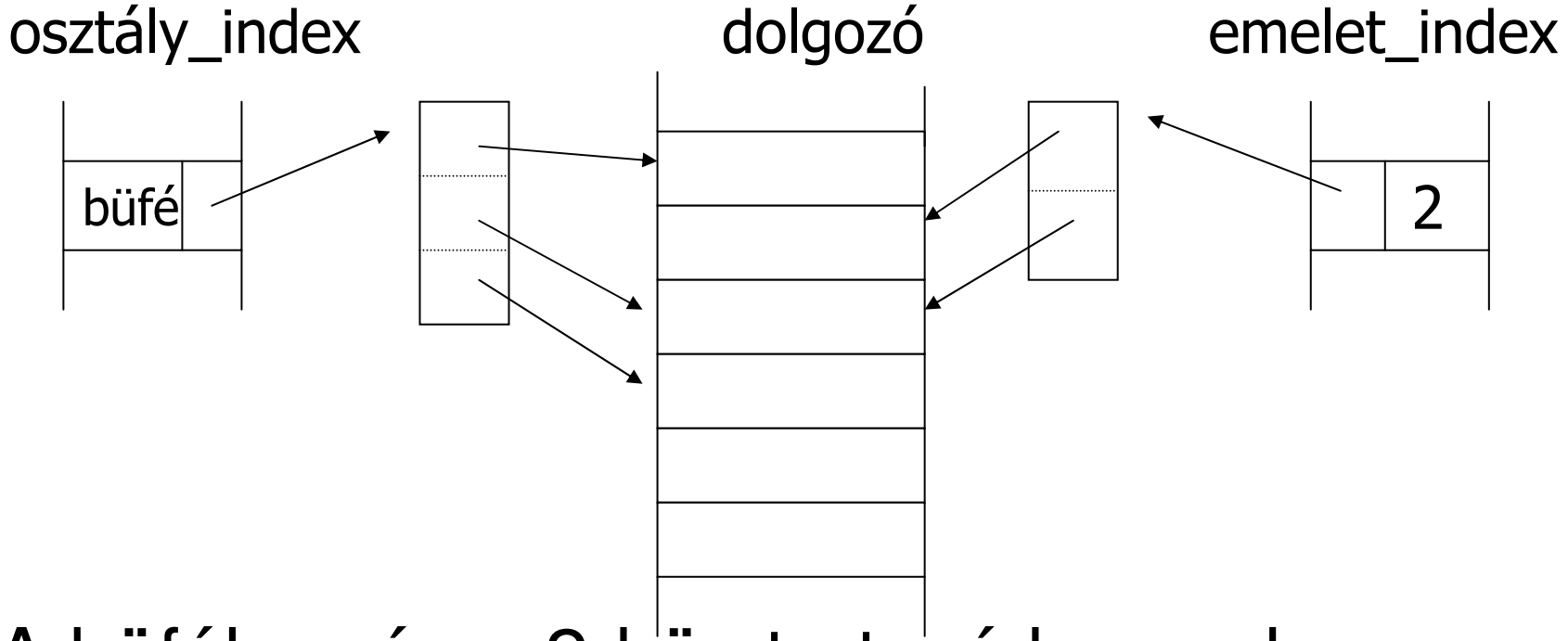
Kosarak

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

ELŐNY: több index esetén a logikai feltételek halmazműveletekkel kiszámolhatók.

Indexelés

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



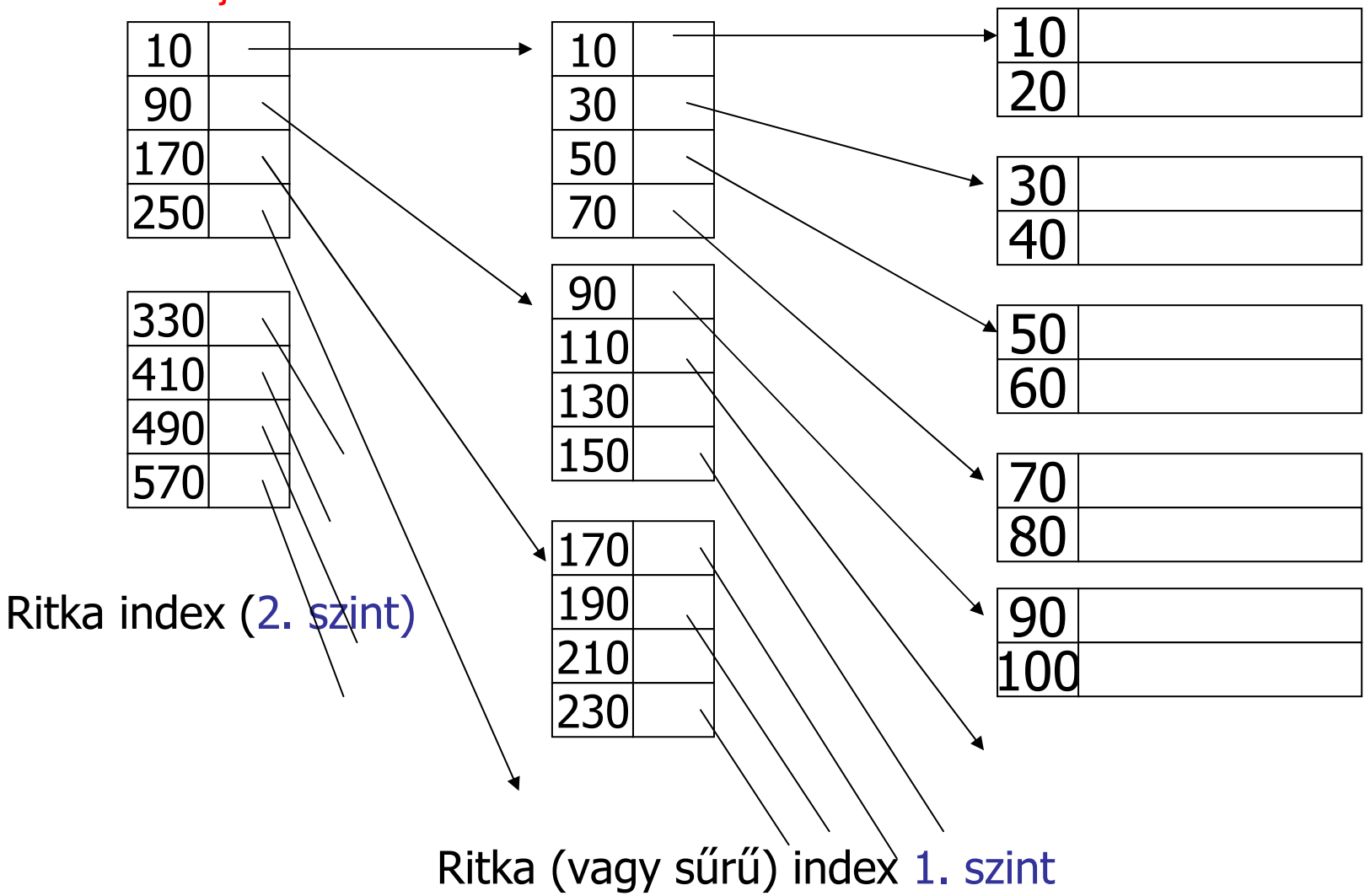
A büféhez és a 2-höz tartozó kosarak metszetét kell képezni, hogy megkapjuk a keresett mutatókat.

Indexelés

- **Klaszter (nyaláb, fürt)**
- Klaszterszervezés egy tábla esetén egy A oszlopra:
  - az azonos A-értékű sorok fizikailag egymás után blokkokban helyezkednek el.
  - CÉL: az első találat után az összes találatot megkapjuk soros beolvasással.
- **Klaszterindex:**
  - klaszterszervezésű fájl esetén index az A oszlopra
- Klaszterszervezés két tábla esetén az összes közös oszlopra:
  - a közös oszlopokon egyező sorok egy blokkban, vagy fizikailag egymás utáni blokkokban helyezkednek el.
  - CÉL: összekapcsolás esetén az összetartozó sorokat soros beolvasással megkaphatjuk.

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)**

- t-ik szinten 1 blokk: **1=B/bf(I)^t**

azaz **t=$\log_{bf(I)}$B < $\log_2$(B)** azaz jobb a rendezett fájlszervezésnél.

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

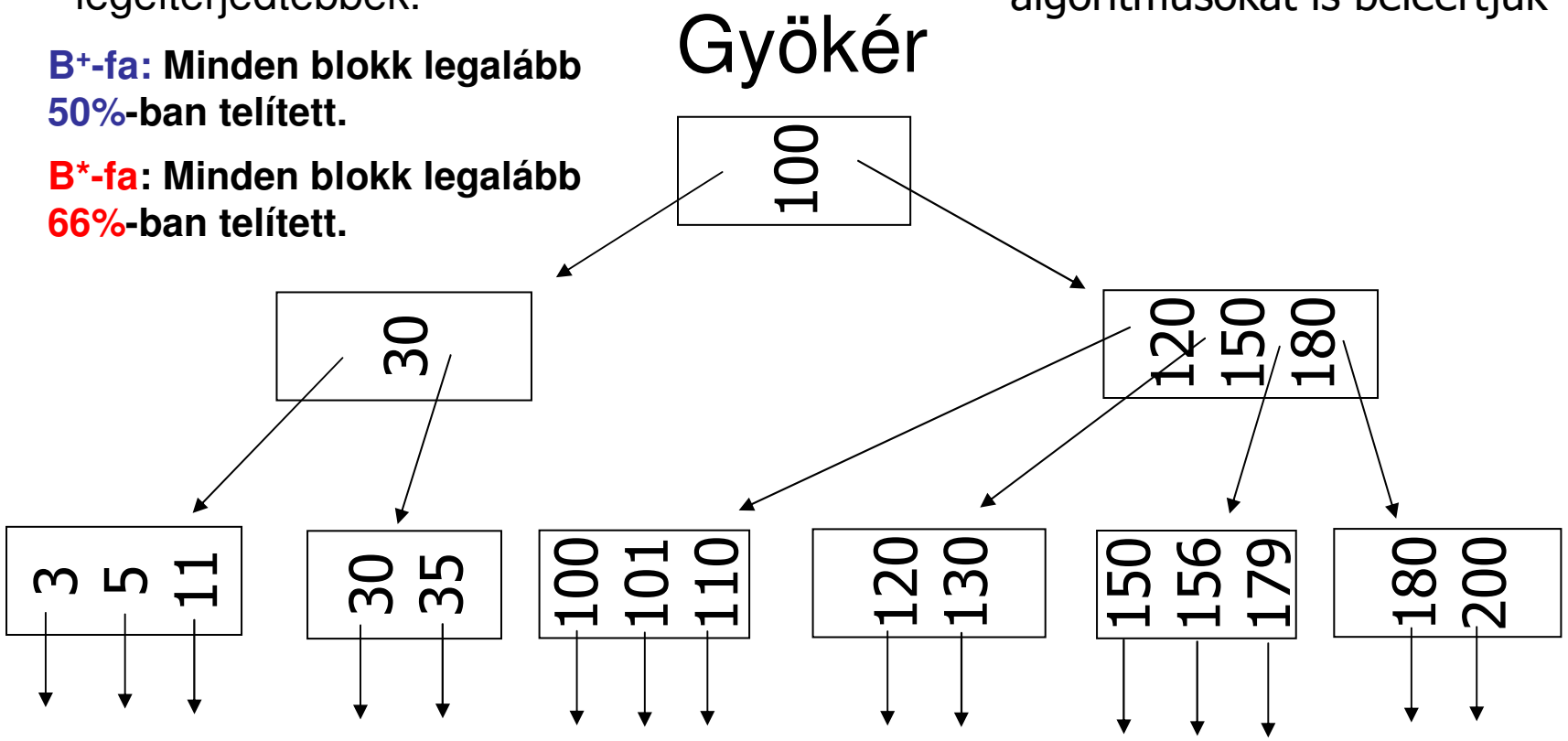
Indexelés

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

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

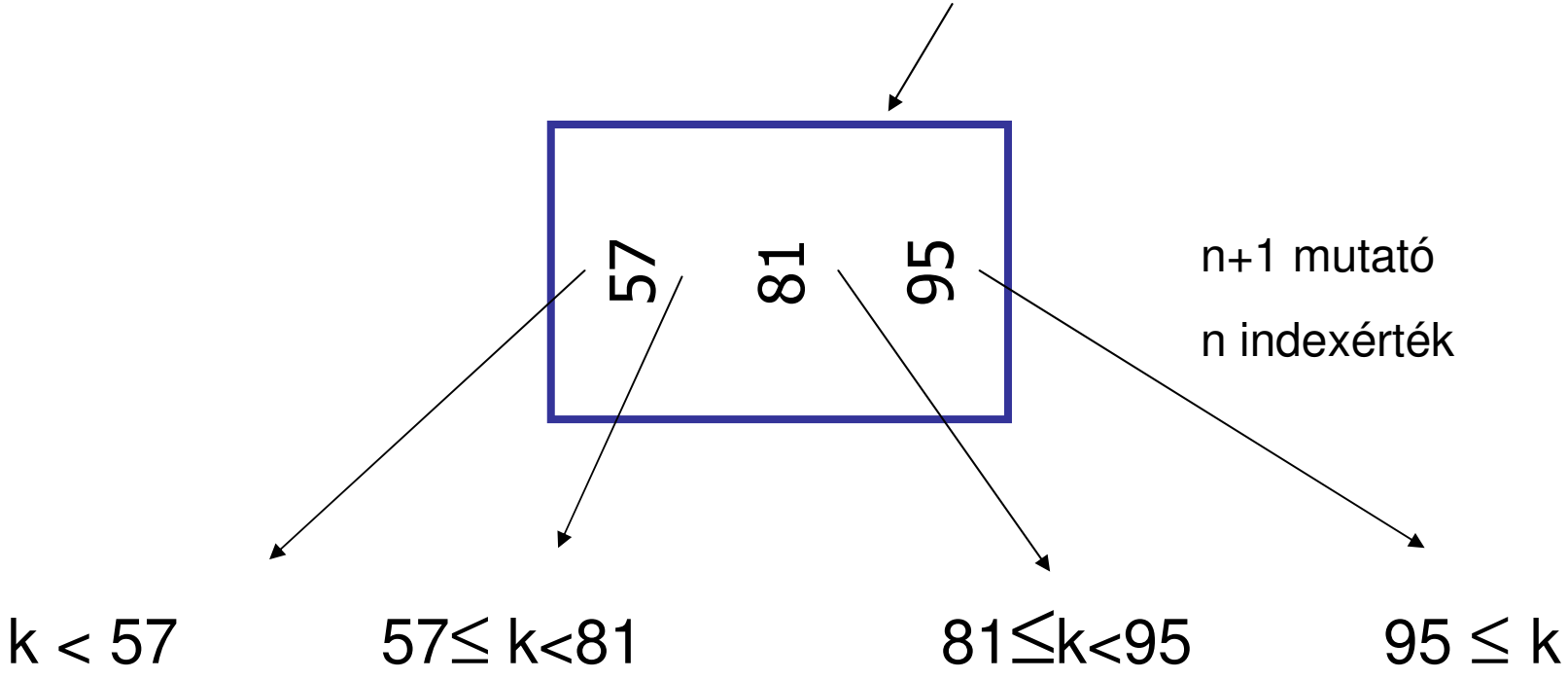
B*-fa: Minden blokk legalább 66%-ban telített.

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



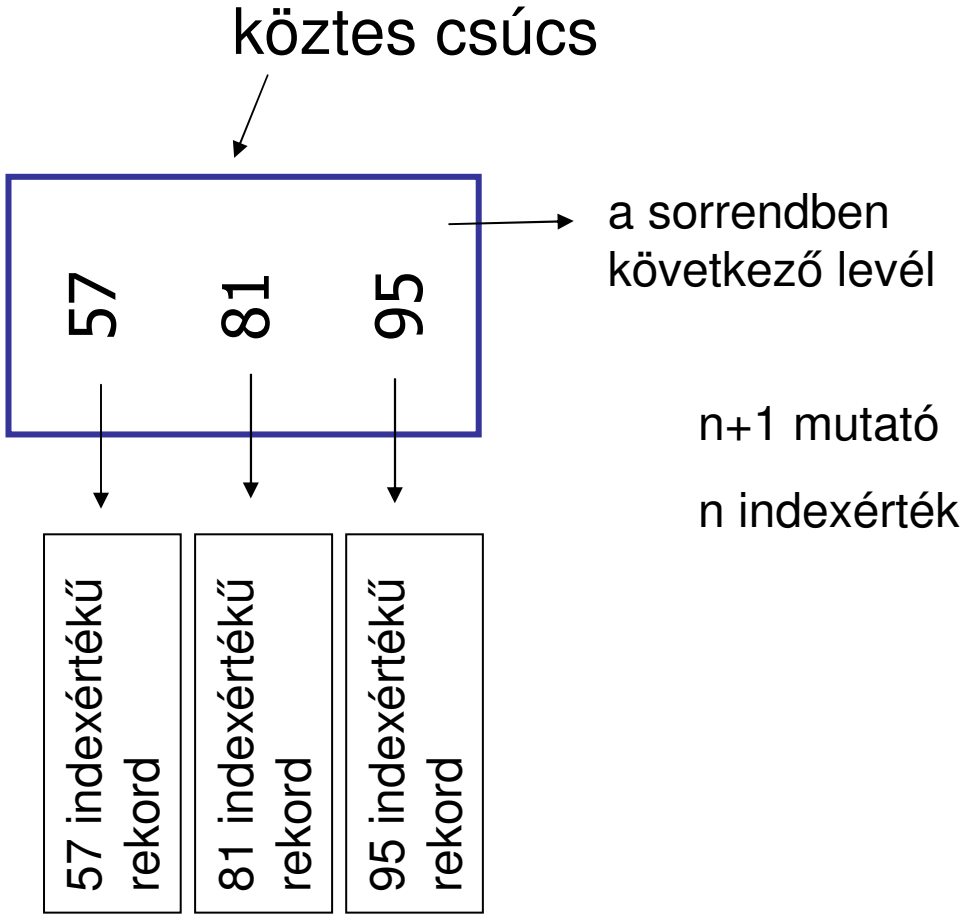
Indexelés

Köztes (nem-levél) csúcs szerkezete



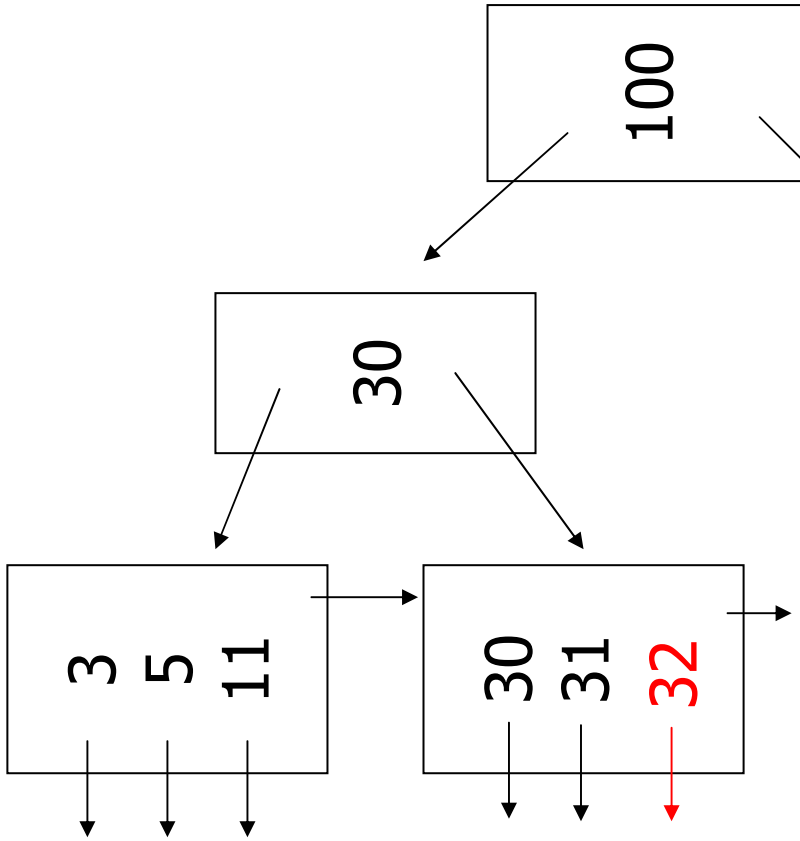
Ahol k a mutató által meghatározott részben (részgráfban) szereplő tetszőleges indexérték

Indexelés
Levél csúcs szerkezete



Indexelés

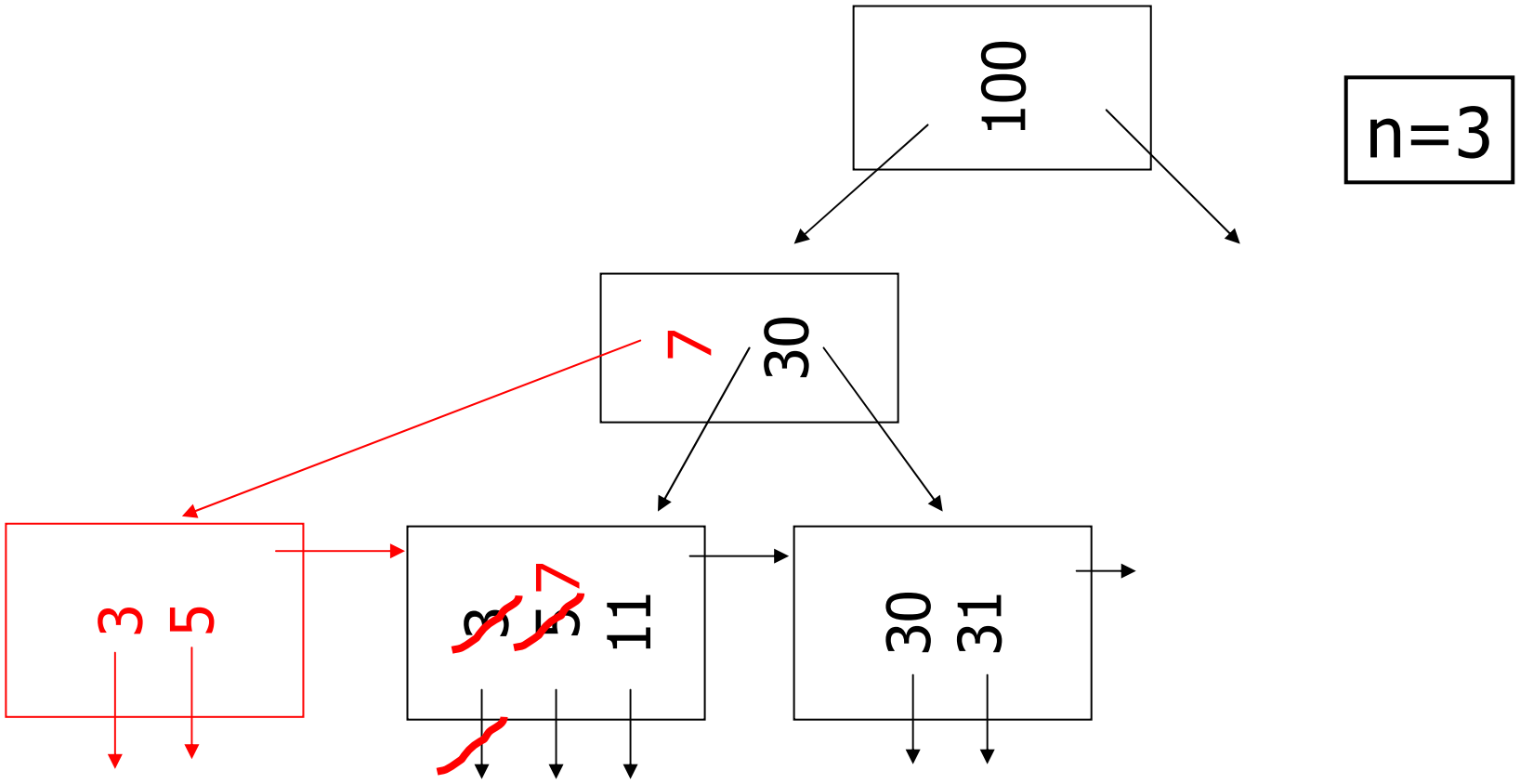
Szúrjuk be a 32-es indexértékű rekordot!



n=3

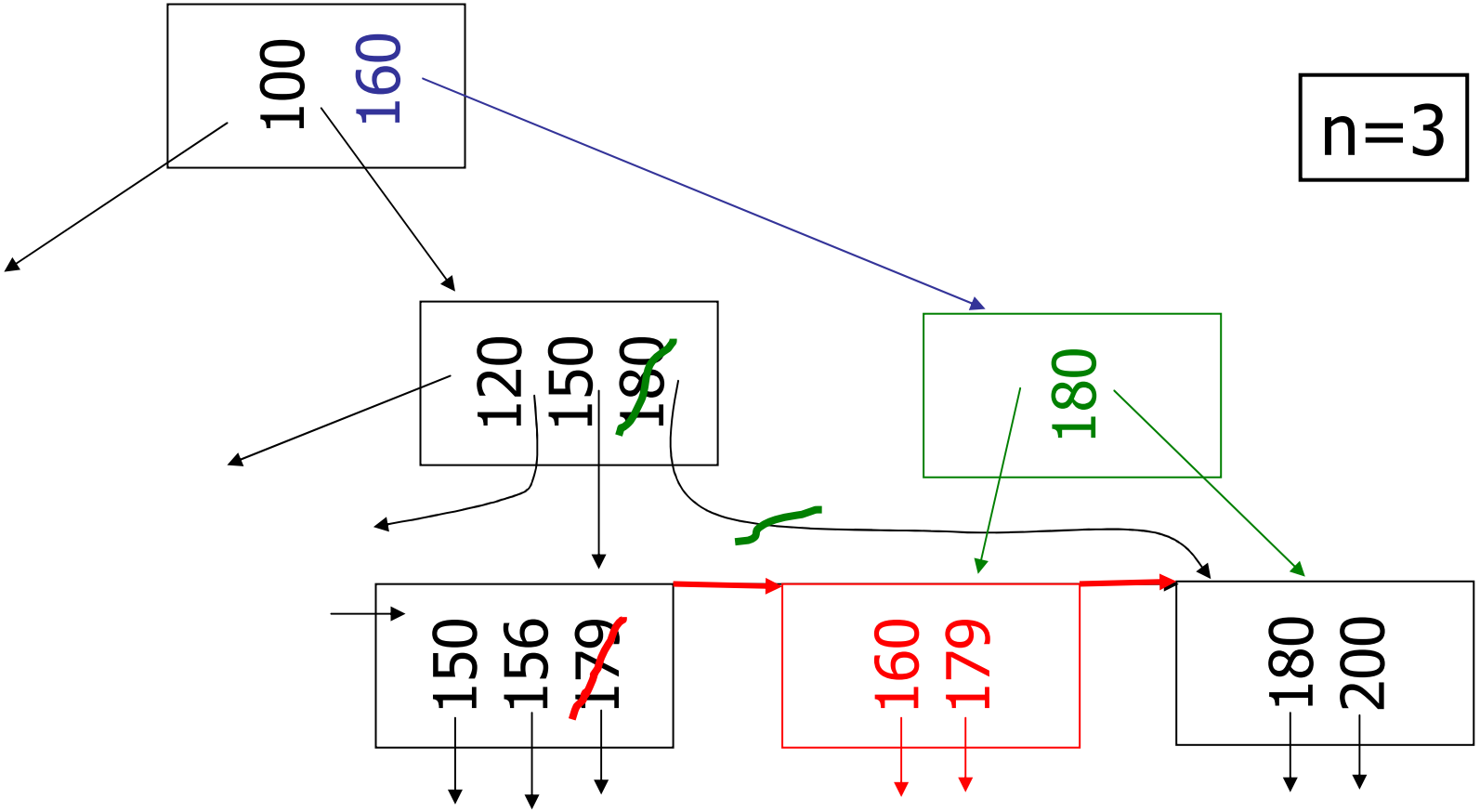
Indexelés

Szúrjuk be a 7-es indexértékű rekordot!



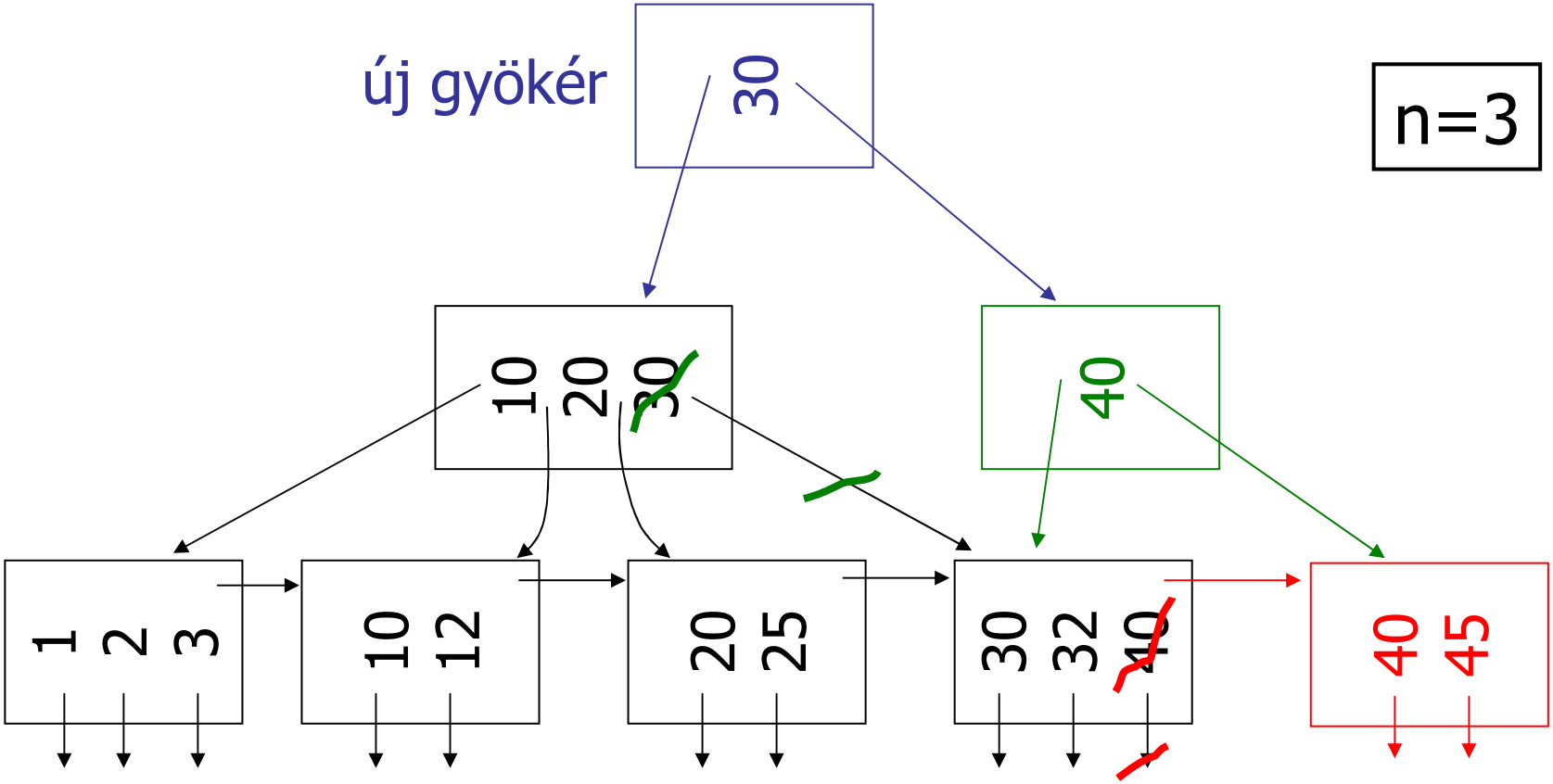
Indexelés

Szúrjuk be a 160-as indexértékű rekordot!



Indexelés

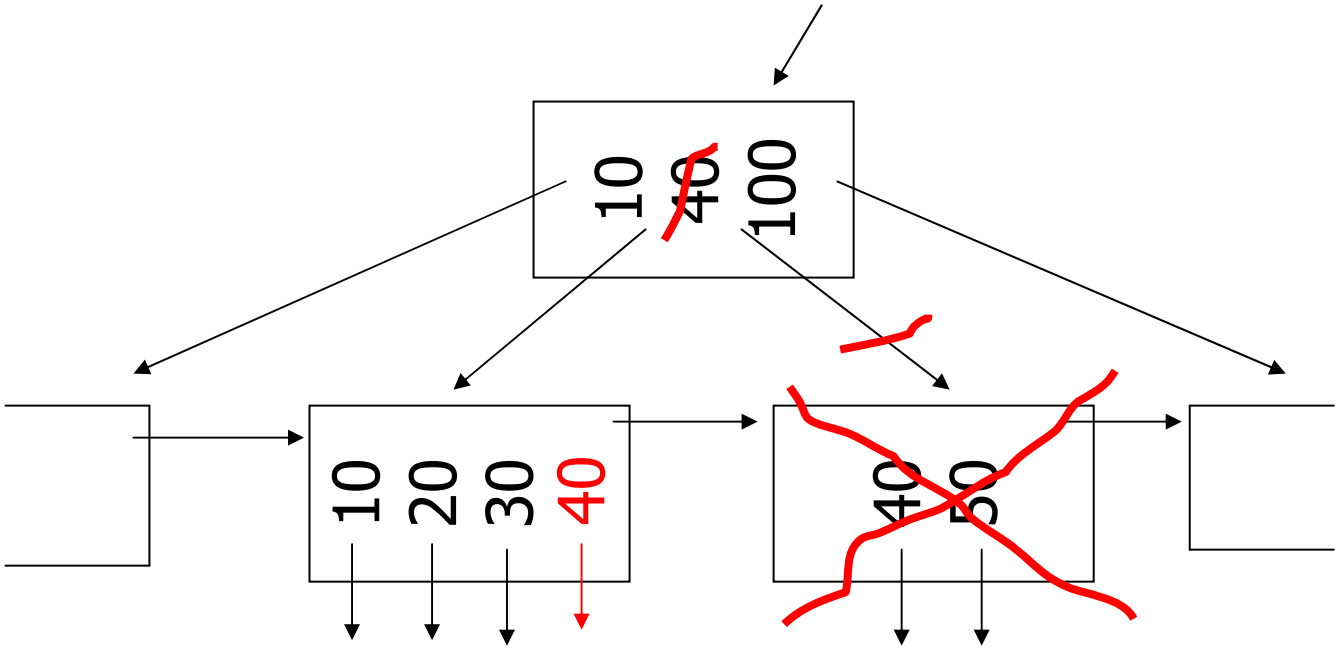
Szúrjuk be a 45-ös indexértékű rekordot!



Indexelés

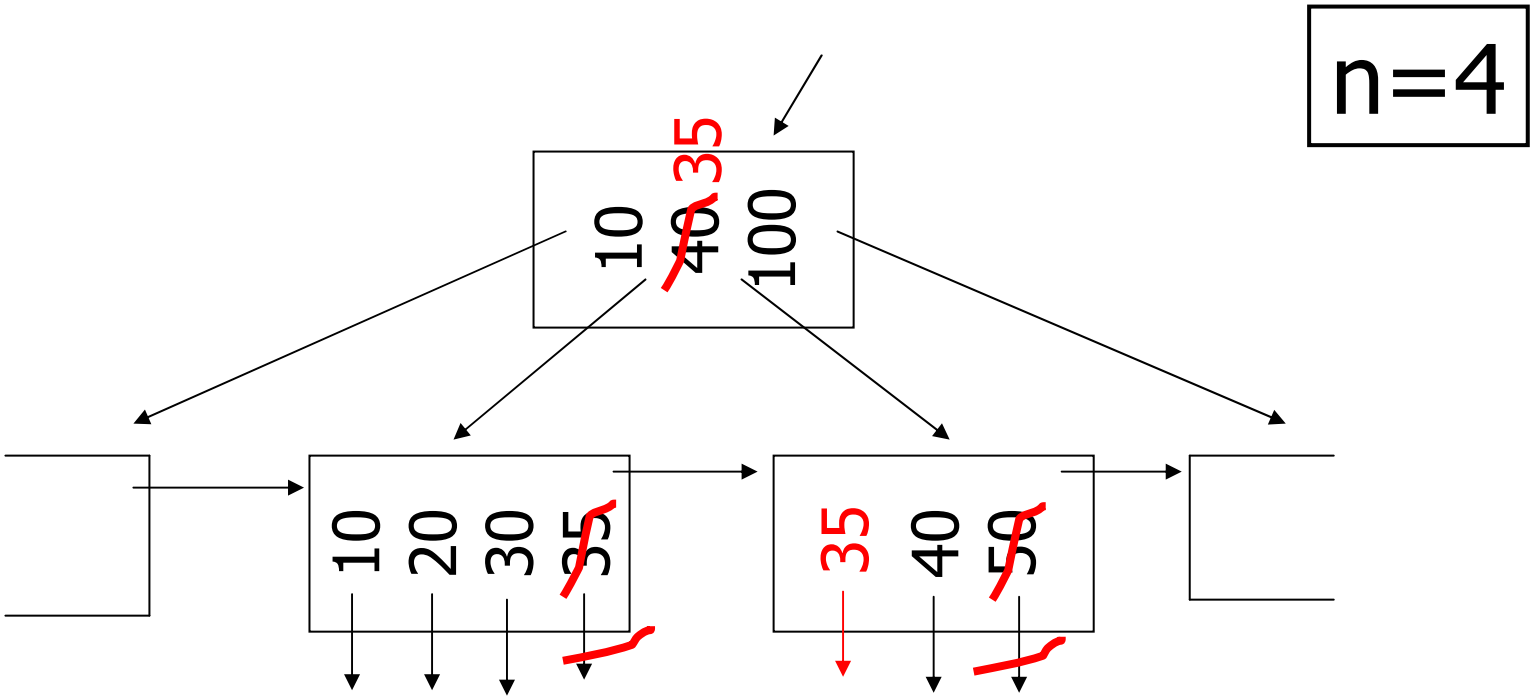
Töröljük az 50-es indexértékű rekordot!

n=4



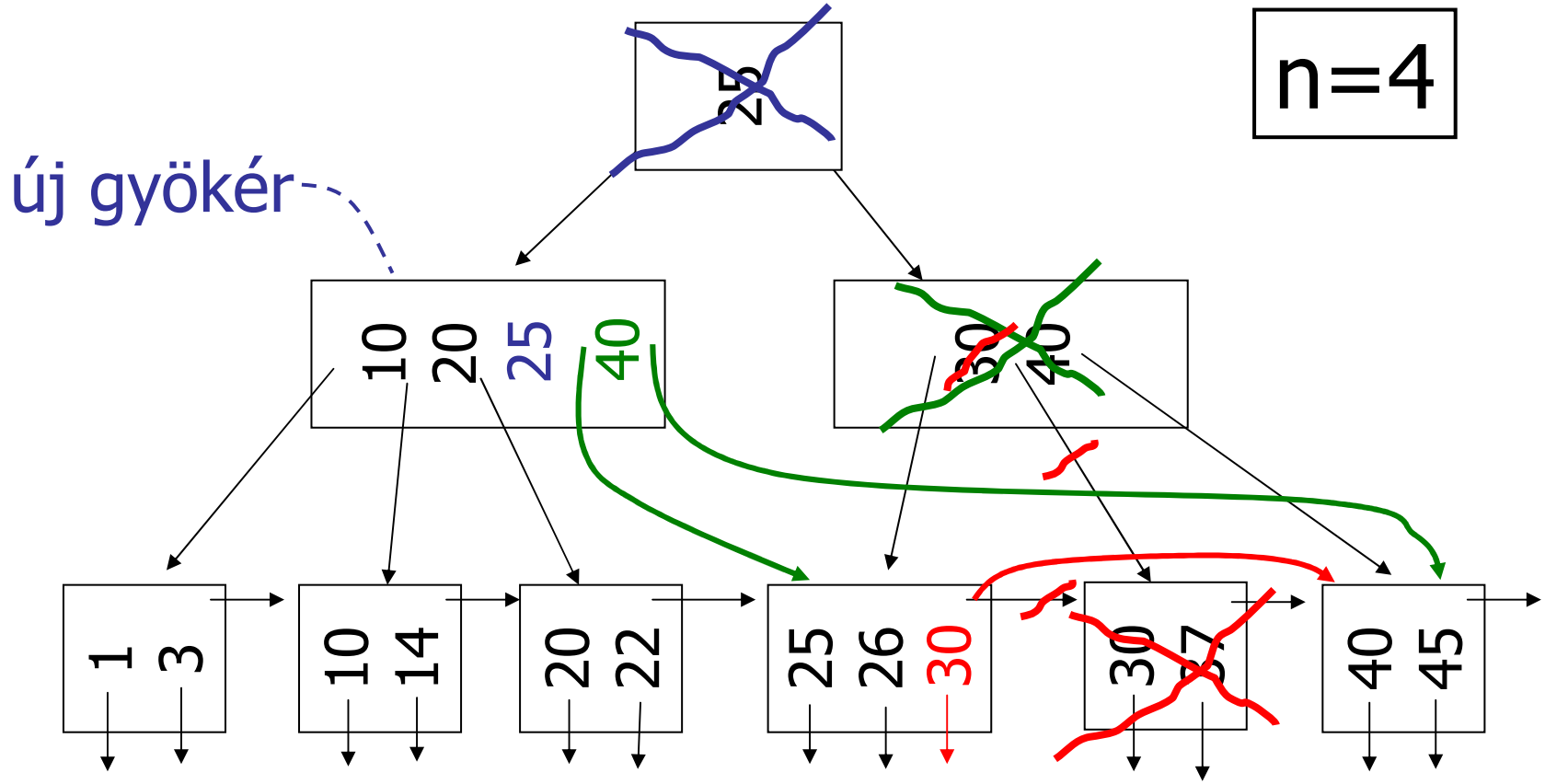
Indexelés

Töröljük az 50-es indexértékű rekordot!



Indexelés

Töröljük a 37-es indexértékű rekordot!



Feladatok és megoldások

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

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, ha az első szinten sűrű az index, és ha az első szinten ritka 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.

Szintek száma:

- az index blokkolási faktora, bf(I)=10

$1=\text{indexméret}/bf(I)^{t-1}$

$t-1=\log_{10}(\text{indexméret})$, azaz

sűrű index esetén $t=1+\log_{10}(n/10)$, illetve

ritka index esetén $t=1+\log_{10}(n/30)$.

Feladatok és megoldások

3. Megoldás (folytatás):

Az összes indexblokkok száma t indexszint esetén (szintenként összegezve):

$1+10+100+\ldots 10^{t-1}=(10^t-1)/(10-1)=(10^t-1)/9.$

Azaz ha legalsó indexszint ritka, akkor

$(10^{(1+\log_{10}(n/10))}-1)/9=10*(n/10)/9=n/9,$

ha a legalsó szint sűrű, akkor

$(10^{(1+\log_{10}(n/30))}-1)/9=10*(n/30)/9=n/27.$

Feladatok és megoldások

4. Minden blokkba 3 rekord, vagy 10 indexrekord (érték-mutató pár) fér. Hány adatblokkot kell átlagosan beolvasni az összes adott értékű rekord eléréséhez, ha rendezett állományunk van, egy érték 1,2 vagy 3-szor szerepelhet 1/3, 1/3, 1/3 valószínűségekkel, és olyan sűrű indexünk van, amelyben csak az első előforduláshoz tartozik indexrekord? Feltesszük, hogy tudjuk előre, hogy mennyi előfordulásunk van az adott értékből.

Megoldás:

Az index alapján megtaláljuk az első előforduláshoz tartozó adatblokkot. Ha csak 1-szer szerepel az érték, akkor csak ezt kell beolvasni. 2 ismétlődés esetén, ha a keresett érték első előfordulása a 3. rekord volt a blokkban, akkor a következő blokkot is be kell olvasni. 3 ismétlődés esetén csak akkor nem kell beolvasni a következő blokkot, ha a keresett érték első előfordulása az 1. volt a blokkban.

$1/3*1+1/3*(2/3*1+1/3*2)+1/3*(1/3*1+2/3*2)=$

$1/3+4/9+5/9=1\,{}^{1}/_{3}$.

Feladatok és megoldások

5. Minden blokkba 3 rekord, vagy 10 indexrekord (érték-mutató pár) fér. Hány blokkot kell átlagosan beolvasni az összes adott értékű rekord eléréséhez, ha egy érték 1,2 vagy 3-szor szerepelhet 1/3, 1/3, 1/3 valószínűségekkel, és olyan sűrű indexünk van, amelyben minden előforduláshoz tartozik indexrekord? Feltesszük, hogy már kiszámoltuk, hogy melyik indexblokkban van az első keresett indexérték, és tudjuk, hogy hány van belőle.

Megoldás:

Az index alapján megtaláljuk az első előforduláshoz tartozó adatblokkot. Ha csak 1-szer szerepel az érték, akkor 1 indexblokkot, és 1 adatblokkot kell beolvasni:

1/3*(1+1) = (1/3)*2

2 ismétlődés esetén, ha a keresett érték első előfordulása a 10. indexrekord volt az indexblokkban, akkor a következő indexblokkot is be kell olvasni, plusz a 2 megfelelő adatblokkot:
1/3*(9/10*(1+1)+1/10*(2+2)) = (1/3)*(22/10)

3 ismétlődés esetén, ha a keresett érték első előfordulása a 9. vagy a 10. volt a blokkban, akkor a következő indexblokkot is be kell olvasni, plusz a 2 megfelelő adatblokkot:

1/3*(8/10*(1+1)+2/10*(2+2)) = (1/3)*(24/10)

Összesen: (1/3)*(20+22+24)/10 = 66/30 = 2.2

Feladatok és megoldások

6. 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?

Megoldás:

Az adatblokkok száma: 3000/3 = 1000.

Ha nem használnánk kosarakat a mutatóknak, akkor a másodlagos indexrekordok száma 3000, az index mérete 3000/10 = 300 lenne.

A különböző értékű indexrekordok száma: 3000/10 = 300, az index mérete 300/10 = 30.

Minden rekordhoz kell egy mutató, így mutatók száma 3000, a mutatók összmérete 3000/50 = 60.

A teljes méret: 1000+30+60 = 1090.

Feladatok és megoldások

7. 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. Mekkora az adatfájl és az index együttes mérete? Mennyi a keresés blokkolvasási költsége?

Megoldás:

Az adatblokkok száma: 1 000 000/10 = 100 000.

A rekordokra mutató mutatók száma a sűrű indexben 1 000 000. A sűrű index mérete 1 000 000/70 ≈ 14286.

A következő szinten már ritka indexet használhatunk, azaz minden blokkhoz elég 1 mutató, és a mutatókból legalább 70 fér 1 blokkba:

14286/70 ≈ 205.

A következő szint blokkjainak száma: 205/70 ≈ 3

A következő szinten már csak 1 blokkra van szükség, ez a gyökér.

Összesen: 100000+14286+205+3+1 = 114495.

A kereséshez a B-fa mind a 4 szintjéről 1 blokkot olvasunk be, és végül 1 adatblokkot, ami összesen 5 blokkolvasás.

Lekérdezések optimalizálása

CÉL: A lekérdezéseket gyorsabbá akarjuk tenni a táblákra vonatkozó paraméterek, statisztikák, indexek ismeretében
és általános érvényű tulajdonságok, heurisztikák segítségével.

Például, hogyan, milyen procedúrával értékeljük ki az alábbi SQL (deklaratív) lekérdezést?

```
Select B,D
From R,S
Where R.A = 'c' and S.E = 2 and R.C=S.C;
```

Lekérdezések optimalizálása

R

A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

S

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

A lekérdezés eredménye:

B	D
2	x

Lekérdezések optimalizálása

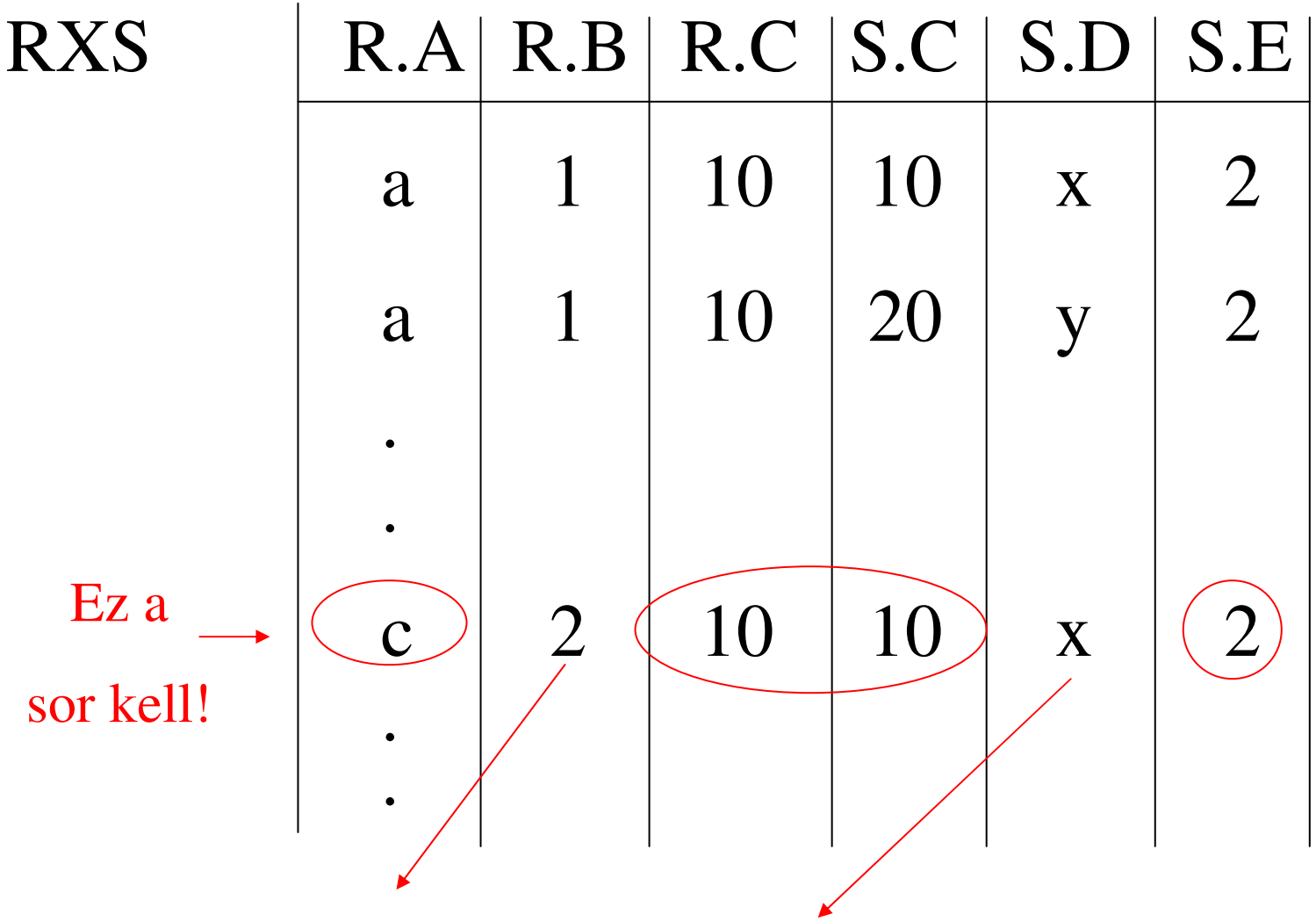
Hogy számoljuk ki tetszőleges tábla esetén az eredményt?

Egy lehetséges terv

- Vegyük a két tábla szorzatát!
- Válasszuk ki a megfelelő sorokat!
- Hajtsuk végre a vetítést!

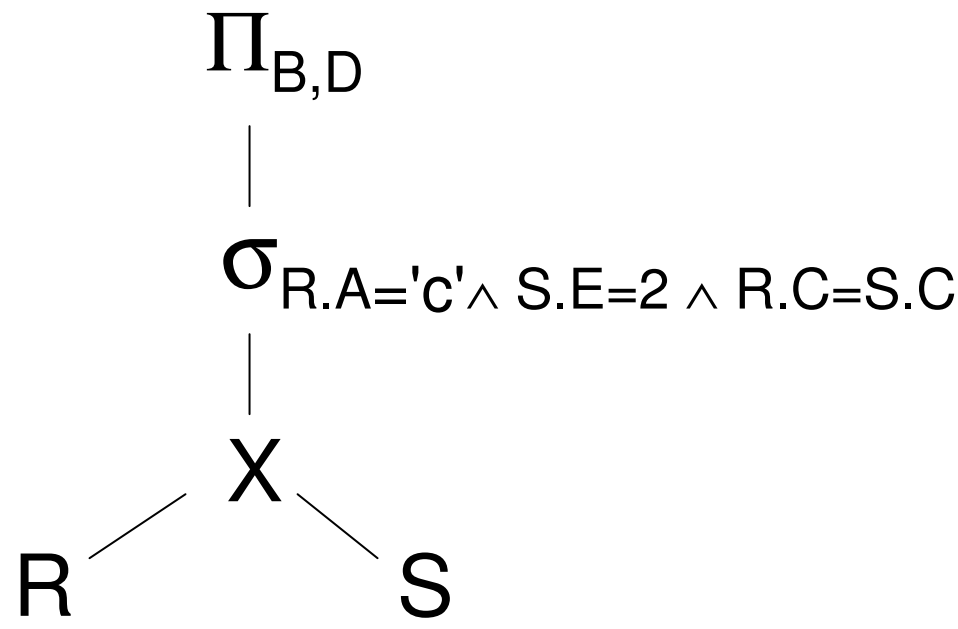
- Ez a direktszorzaton alapuló összekapcsolás.
- Oracle-ben: NESTED LOOP.
- Nagyon költséges!

Lekérdezések optimalizálása



Lekérdezések optimalizálása

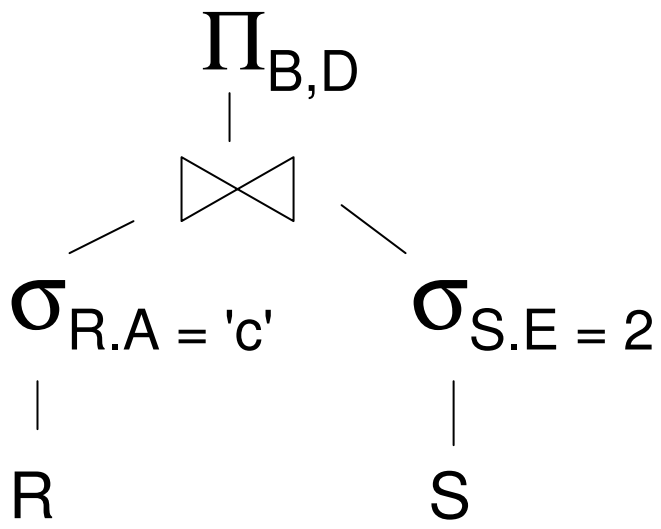
Ugyanez a terv relációs algebrában:



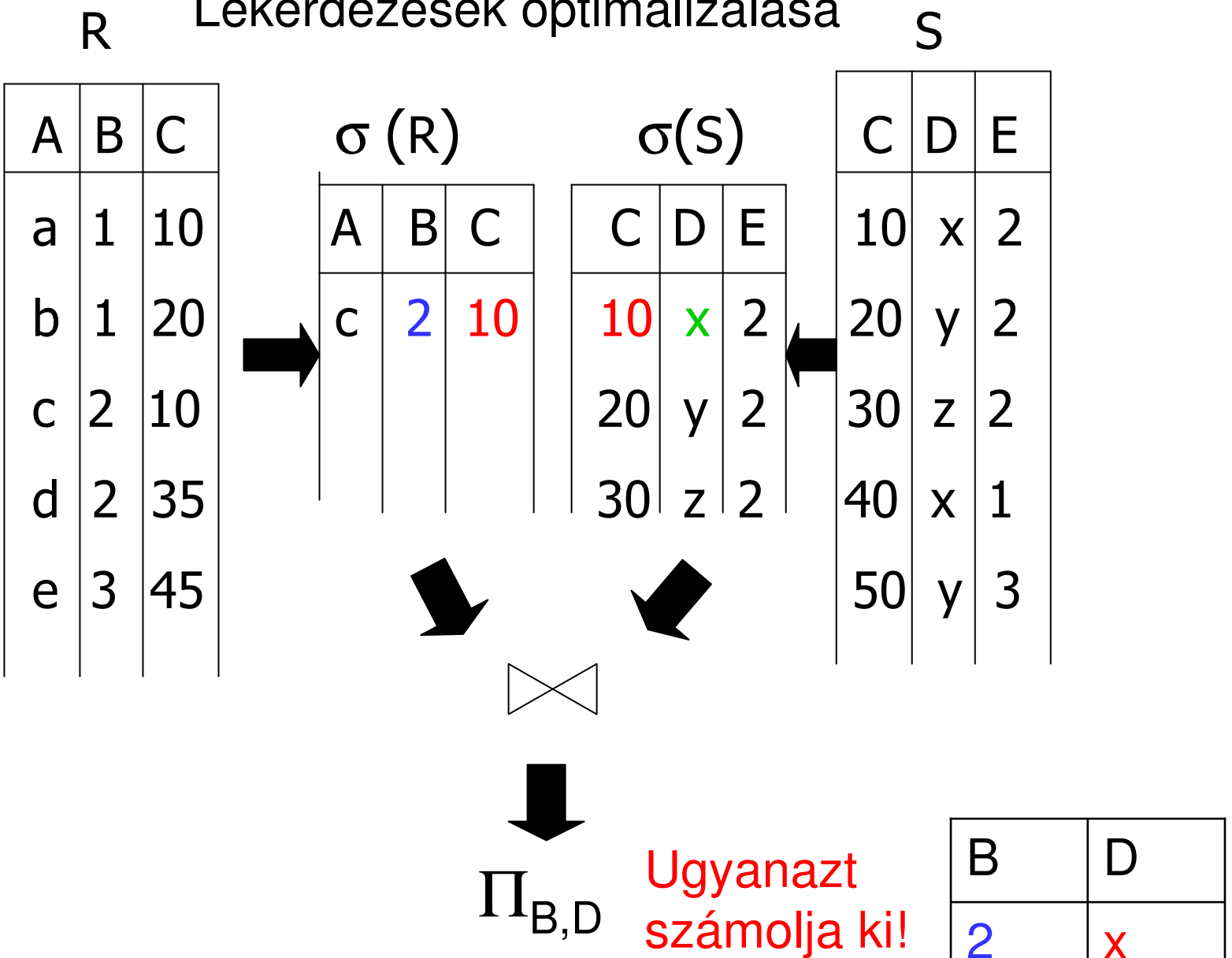
$$\Pi_{B,D}\,[\,\sigma_{R.A='c' \wedge S.E=2 \wedge R.C = S.C}\,(RXS)]$$

Lekérdezések optimalizálása

Egy másik lehetséges kiszámítási javaslat:



Lekérdezések optimalizálása



Lekérdezések optimalizálása

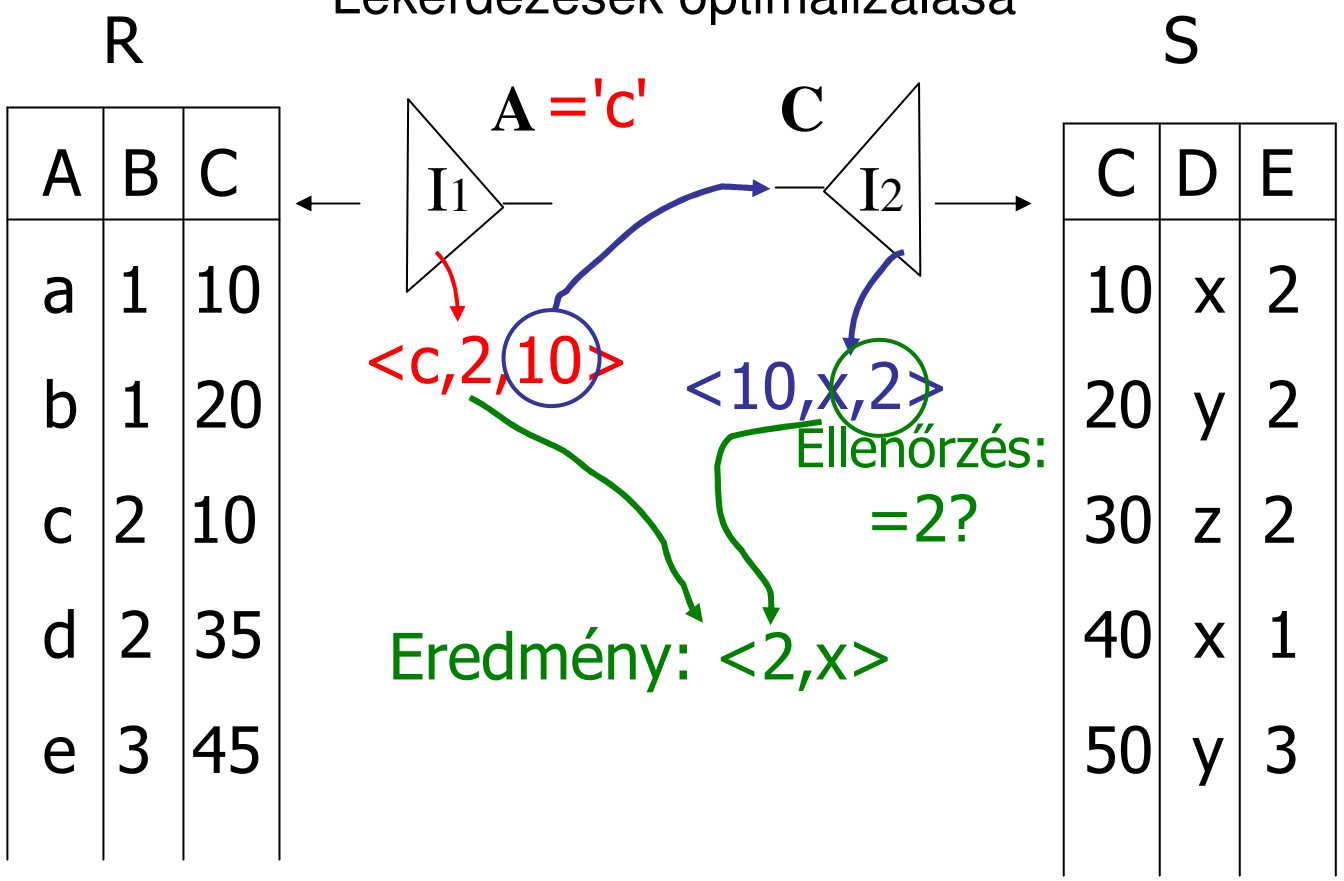
Használjuk ki az R.A és S.C oszlopokra készített indexeket:

(1) Az R.A index alapján keressük meg az R azon sorait, amelyekre R.A = 'c'!

(2) Minden megtalált R.C értékhez az S.C index alapján keressük meg az S-ből az ilyen értékű sorokat!

(3) Válasszuk ki a kapott S-beli sorok közül azokat, amelyekre S.E = 2!

(4) Kapcsoljuk össze az R és S így kapott sorait, és végül vetítsünk a B és D oszlopokra.

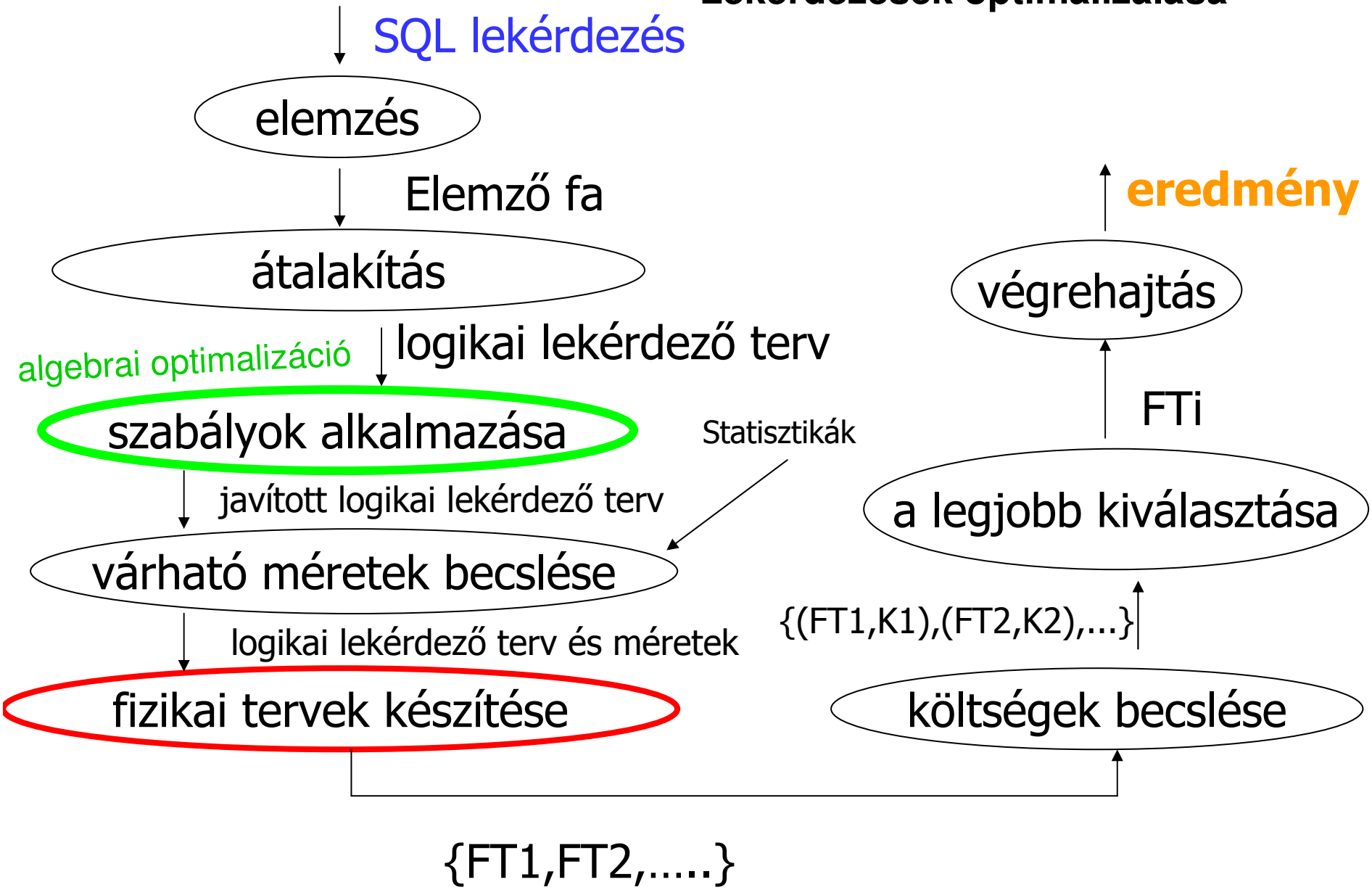


A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

2-INDEXES ÖSSZEKAPCSOLÁS

Lekérdezések optimalizálása



Fizikai tervek

- Az algebrai műveleteket többféle módon valósíthatjuk meg. Becslések, statisztikák ismeretében minden művelet megvalósításának költséget meg tudjuk becsülni.
- A legköltségesebb műveletekkel, azaz a szorzásokkal és összekapcsolásokkal kezdjük.
- **Költség** = számítási költség (az eredmény blokkjainak memóriában előállításához mennyi blokkműveletre van szükség) +

  eredmény mérete (a memóriából kimásoljuk az eredményt tartalmazó blokkokat)