

Hasító táblázatok

11. előadás

Hasító táblázatok

- A gyakorlatban sokszor csak a BESZÚR, KERES, TÖRÖL műveleteket megvalósító dinamikus szerkezetre van szükségünk
 - Hogyan lehetne ezt hatékonyan tenni?
- Az eddigi kereső szerkezeteink
 - A kulcsok összehasonlításán alapultak
 - Végrehajtási idő $\mathcal{O}(n)$ vagy $\mathcal{O}(\log n)$ volt
- Szeretnénk az előzőnél jobbat elérni....
- Ha vektorban indexelünk, az csak $\mathcal{O}(1)$...

Hasító táblázatok

- Vizsgáljuk meg, hogy a személyi szám alkalmas-e kulcsnak?

Mezőnév	nem	év	hó	nap	azonosító	
Példaérték	2	72	08	15	2806	
Értékek száma	2	100	12	31	999	~74M

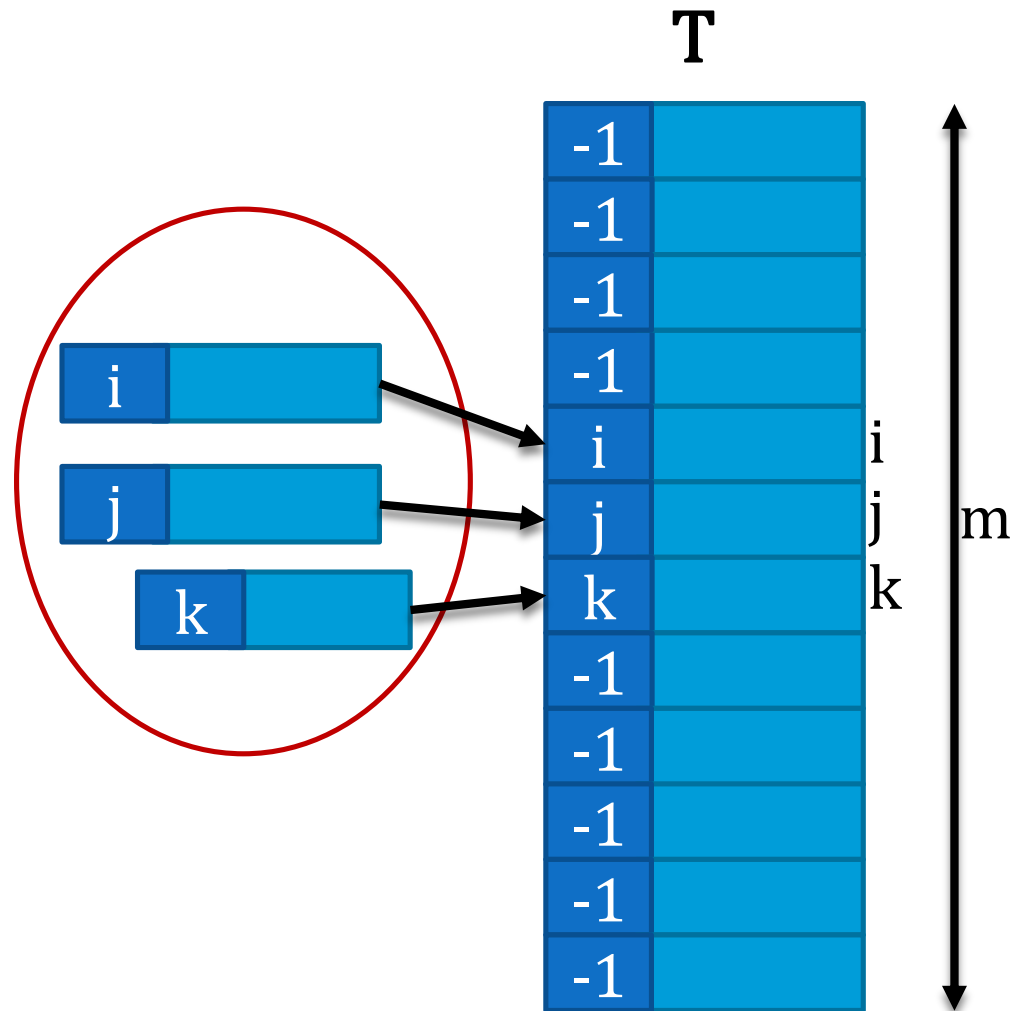
- (Az utolsó 4 jegyből csak 3 értékes jegy, 1 a többiből képződik.)
- Ha összeszámoljuk az összes kitöltési lehetőséget, akkor ~74 milliót kapunk ...
- A ~10 millió lakosra pedig elég egy ~12 millió rekordot tartalmazó fájl.
 - A személyi szám tehát **nem alkalmas** kulcsnak.
- Kell egy olyan h függvény, ami minden személyi számhoz rendel egy egészet a $[0; 12 * 10^6 - 1]$ intervallumból!

Hasító táblázatok

- A hash-elés jelensége:
 - Adott a K kulcsok halmaza. A K elemeivel azonosított rekordok száma azonban várhatóan **jóval kisebb**, mint K számossága.
 - Ekkor K -t egy alkalmas h függvénnyel leképezzük az ábrázolás alapját képező kisebb tartományra. Legyen ez a $[0 \dots M - 1]$ intervallum.
 - A $h: K \rightarrow [0 \dots M - 1]$ függvényt hash-függvénynek nevezzük.
 - Mivel $M < |K|$, sőt általában $M \ll |K|$, ezért h nem lehet injektív, hanem szükségképpen fellép a kulcsütközés: van olyan $k \neq k'$, amelyre $h(k) = h(k')$.
 - A hasítós technikák feladata éppen az, hogy megoldást adjanak a kulcsütközésre.
- Fontos kérdés:
 - Milyen hatékonyságot várhatok?

Hasító táblázatok

- A közvetlen hozzáférésű (címezésű) táblák – a legegyszerűbb eset
- Tegyük fel, hogy az elemek kulcsai különböző egész értékek a $[0, m - 1]$ intervallumból, és m nem túl nagy
- Használjuk magukat a kulcsértékeket, hogy kiválasszunk egy helyet a T közvetlen hozzáférésű táblában, melyben az elemeket tároljuk
- Egy k kulcsú elem keresése: nézzük meg a k indexű elemet
 - ha van itt egy érték, megtaláltuk,
 - ha a jelző itt pl. -1 , akkor nincs benne.
- Konstans idő, $\mathcal{O}(1)$
- Beszúrás, törlés is $\mathcal{O}(1)$



Hasító táblázatok

Közvetlen_címzésű_keresés (T, k)

return $T[k]$

Közvetlen_címzésű_beszúrás (T, x)

$T[x.kulcs] \leftarrow x$

Közvetlen_címzésű_törlés (T, x)

$T[x.kulcs] \leftarrow -1$

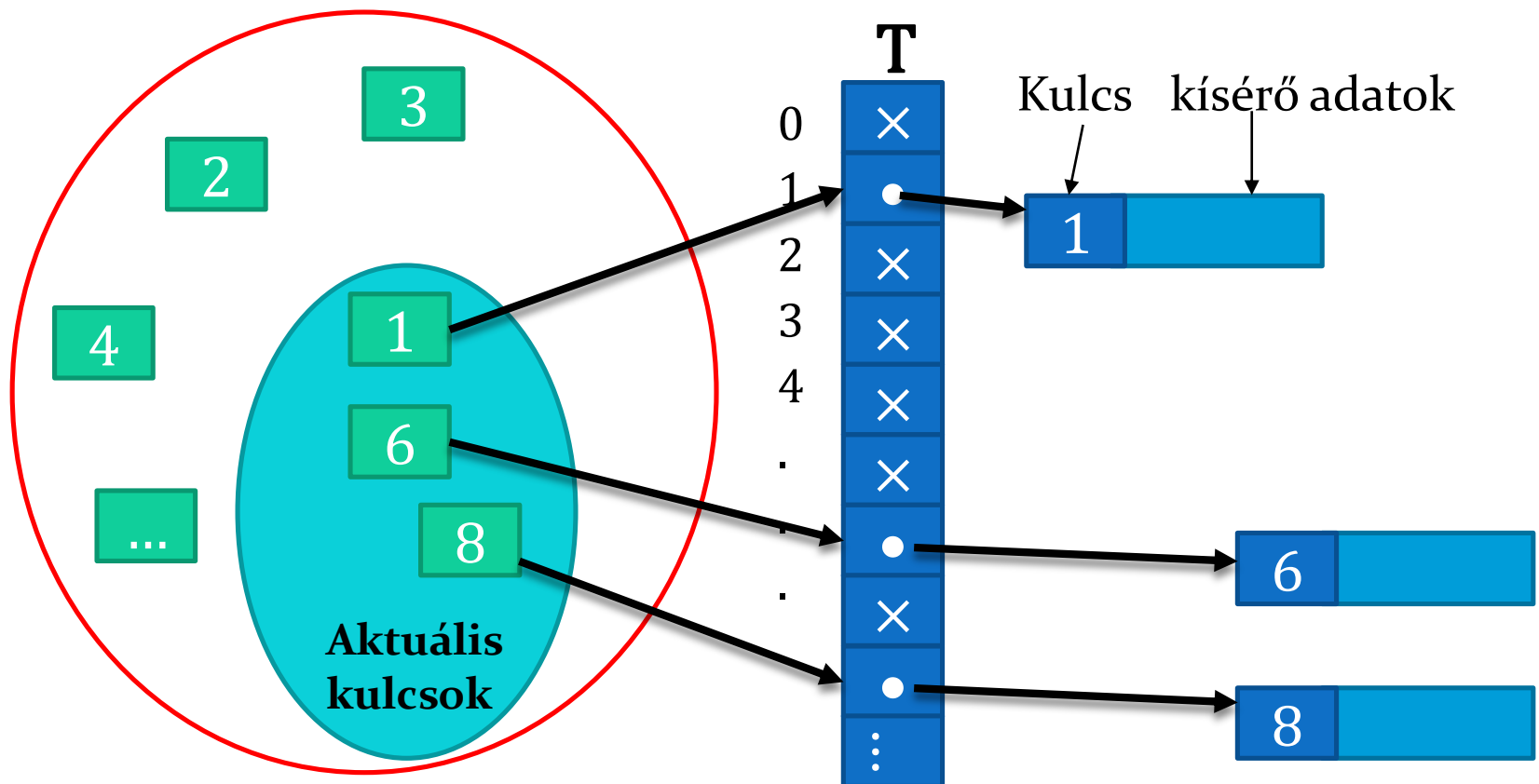
- Mindegyik $\mathcal{O}(1)$ idejű

Hasító táblázatok – megszorítások

- Megszorítások
 - Kulcsok *egészek* kell legyenek
 - Kulcsok *egyediek* kell legyenek
 - Kulcsok *kis intervallumból* kerülhetnek ki
 - A tárolás hatékonysága miatt a kulcsok sűrűn kell legyenek az intervallumban
 - Ha ritkásan vannak – sok üres hely van az értékek között – túl sok helyet használunk el, hogy a sebességet megnyerjük
 - „Hely a sebességért kereskedelem” ☺

Hasító táblázatok

- A közvetlen hozzáférésű (címezésű) táblák
 - másik lehetőség: T-indexei a kulcsok, T-ben mutatók, csak akkor tárolom az egészet, ha kell:



Hasító táblázatok

T inicializálása

minden elemét null-ra állítjuk

Közvetlen_címzésű_keresés (T, k)

return „a T[k] által mutatott objektum”

Közvetlen_címzésű_beszúrás (T, x)

helyet foglalunk x-nek,

T[x.kulcs] mutatóját erre állítjuk

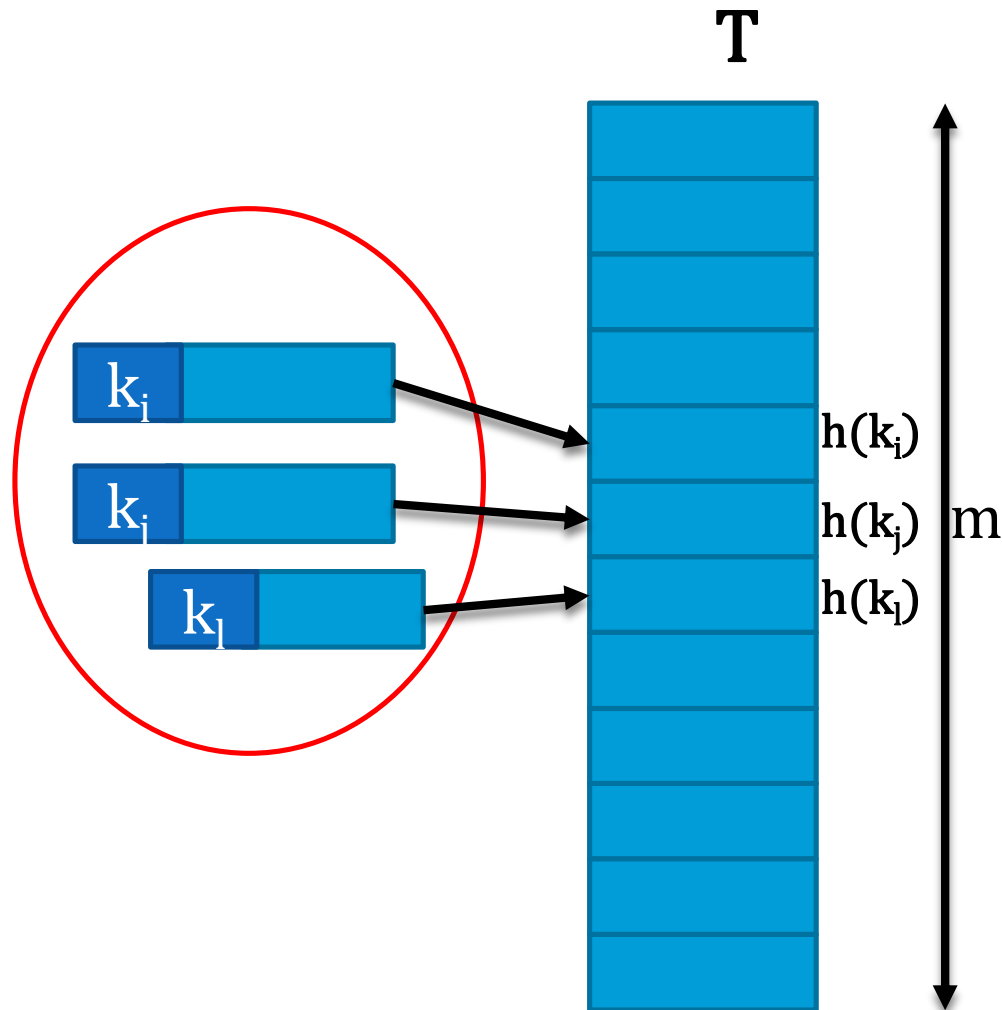
Közvetlen_címzésű_törlés (T, x)

felszabadítjuk a T[x.kulcs] által mutatott objektumot
(T[x.kulcs] is null lesz!)

- Mindegyik $\mathcal{O}(1)$ idejű

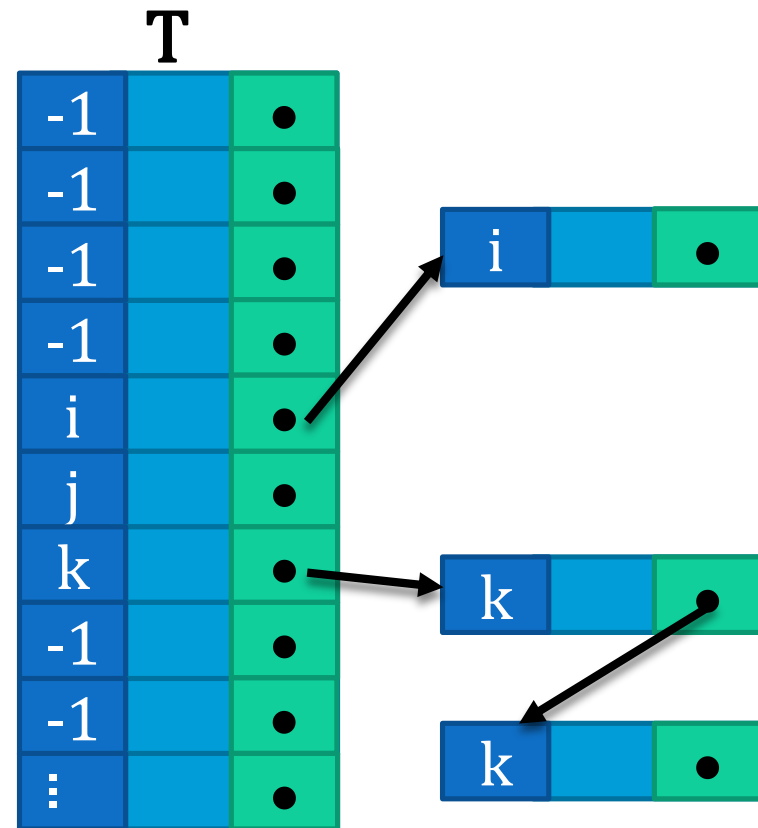
A megszorítások gyengítése

- A kulcsok egészek legyenek
 - Kell egy hash függvény $h(\text{kulcs}) \rightarrow \text{egész}$
- Ezt a függvényt a kulcsra alkalmazva egy indexet kapunk
- Ha h minden kulcsot egy egyedi egész értékre képez le a $0 \dots m - 1$ intervallumban, akkor a keresés $\mathcal{O}(1)$

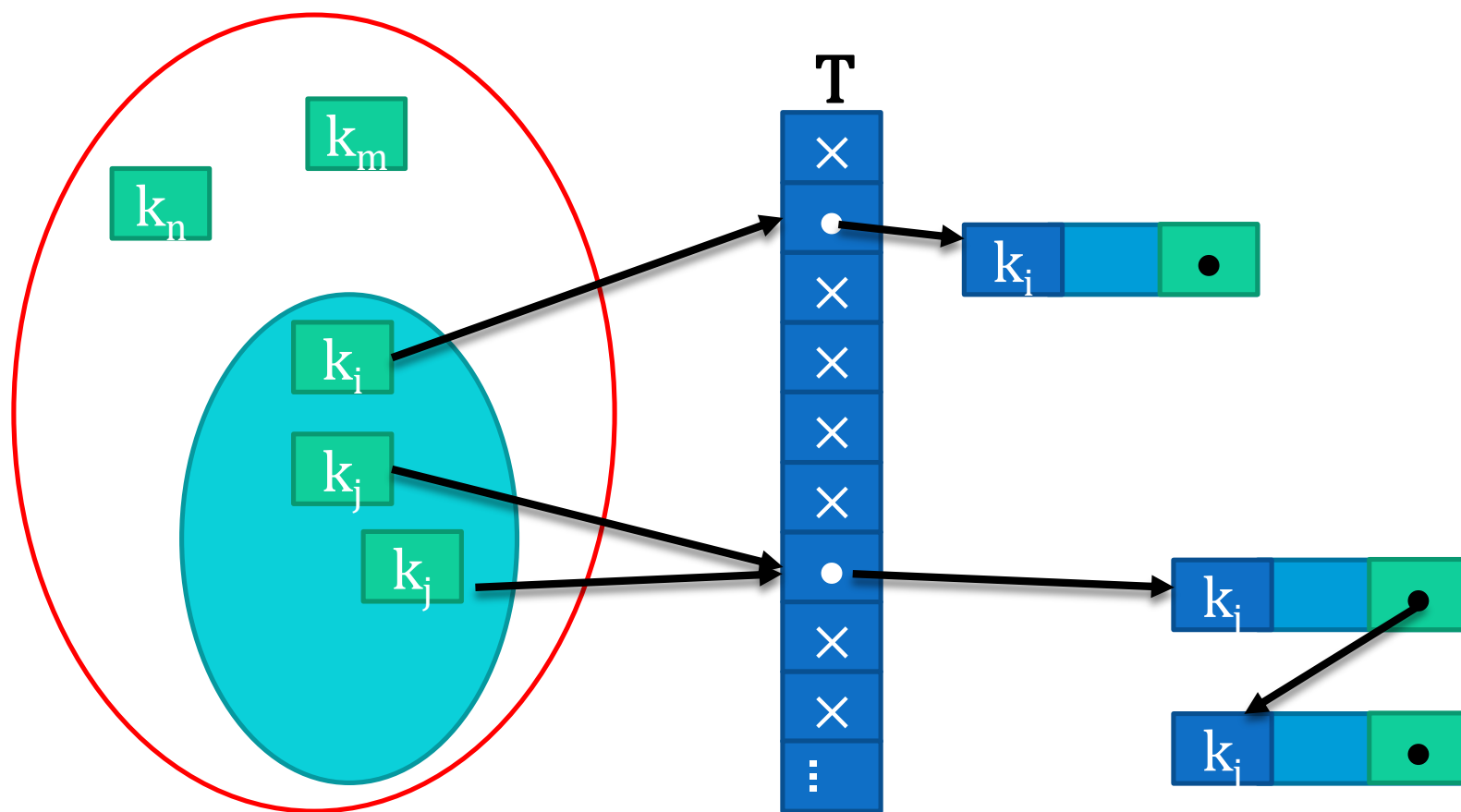


A megszorítások gyengítése

- A kulcsoknak egyedieknek kell lenniük
 - hozzuk létre a duplikátumok láncolt listáját, és ezt kapcsoljuk a táblához
 - ha egy keresésnek elég **akármelyik** k kulcsú elem, a végrehajtás még mindig $\mathcal{O}(1)$
- De ha az elemnek még van valami más megkülönböztető jegye is, ami meg kell egyezzen, akkor $\mathcal{O}(n_{dup_max})$ -t kapunk, ahol n_{dup_max} a duplikátumok legnagyobb száma – vagyis a leghosszabb lista hossza



A megszorítások gyengítése



Hash függvények

- A hash függvény formája

- Példa – egy n -karakteres kulcsot használva

```
int hash(char *s, int n)
{
    int sum = 0;
    while(n-->0) sum=sum+ *s++;
    return sum % 256;
}
```

ez a függvény a 0 ... 255 egy értékét adja vissza

- a xor függvényt is gyakran használják

```
sum=sum^ *s++;
```

- De tetszőleges függvény, ami a $0..m-1$ -ben generál értékeket egy megfelelő (nem túl nagy) m -re jó lesz!
- Amíg a hash függvény maga $\mathcal{O}(1)$!

Kulcsütközések

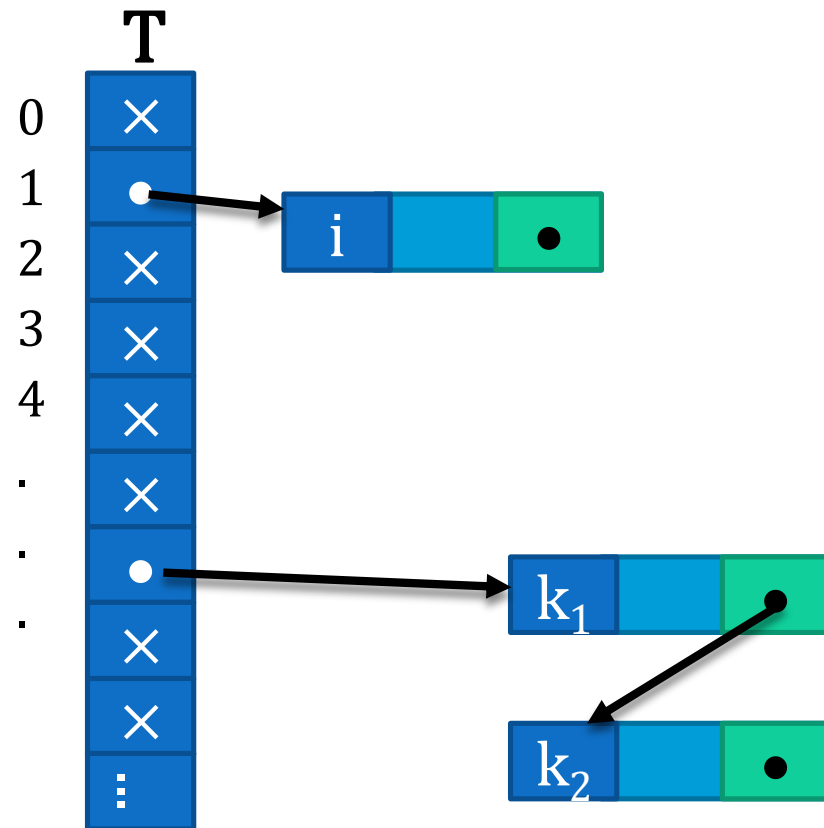
- Hash függvény
 - Ezzel a hash függvénnnyel:
 - ```
int hash(char *s, int n)
{
 int sum = 0;
 while(n--) sum=sum+*s++;
 return sum % 256;
}
```
  - `hash("AB", 2)` és `hash("BA", 2)` ugyanazt az értéket adják!
  - Ezt hívjuk **kulcsütközésnek**
    - Számos technikát használnak a **kulcsütközés** feloldására

# Kulcsützközés feloldása

- Kulcsützközés
  - Akkor fordul elő, ha a hash függvény két különböző kulcshoz rendeli ugyanazt a címet
    - A táblázat fel kell ismerje és fel kell oldja ezt
    - Felismerés
      - Tároljuk az aktuális kulcsot az elemmel a hash táblában:
      - Számítsuk ki a címét  $k = h(\text{kulcs})$
      - Ellenőrizzük a találatot:
        - `if (table[k].key == kulcs) then találat`  
`else próbáld a következőt`
    - Javaslat
      - Számos technika – néhányat megnézünk

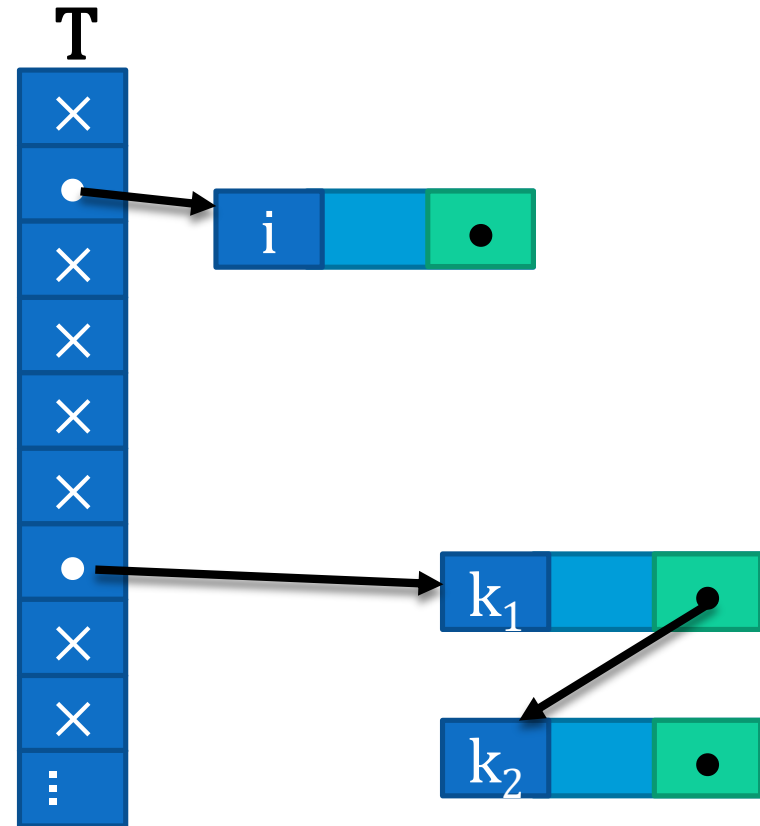
# Láncolt listák

- Kulcsütközés - Javaslat
  - minden táblaelemhez egy láncolt listát rendelünk
  - Keressünk  $i$  kulcsú elemet:
  - Láncolt\_hasító\_keresés( $T, i$ )
    - kiszámítjuk  $h(i)$ -t
    - keressük az  $i$  kulcsú elemet a  $T[h(i)]$  listában
    - ha NULL-t találunk, a kulcs nincs a táblában



# Láncolt listák

- Kulcsütközés - Javaslat
  - minden táblaelemhez egy láncolt listát rendelünk
- Beszúrunk **x**-t:
  - $\text{Láncolt\_hasító\_beszúrás}(T, x)$ 
    - kiszámítjuk  $h(x.\text{kulcs})$ -t
  - Beszúrunk a  $T[h(x.\text{kulcs})]$  lista elejére
- Törlés – hasonlóan a  $T[h(x.\text{kulcs})]$  listából



# Hasító táblázatok – betöltési tényező

- Az ütközések nagyon valószínűek
- A tábla betöltési tényezőjét alacsonyan kell tartani:

$$\alpha = \frac{n}{m}$$

- $n$  az elemek száma,  $m$  a helyek száma
- Az átlagos lánc hosszúságok kiterjedt analízise ismert
- Külön láncolás
  - Minden helyhez külön adjuk a láncolt listákat
  - Jobb végrehajtást ad
  - De több helyet foglal

# Láncolásos hasítás műveletigénye

- Milyen hatékonysággal működik a láncolásos hasítás?
  - Legyen  $T$  egy  $m$  rést tartalmazó hasító táblázat, amelyben  $n$  elem van.
  - Az  $\alpha$  kitöltési tényező:  $\frac{n}{m}$  = az egy láncba fűzött elemek átlagos száma.
    - $\alpha$  lehet kisebb, egyenlő és nagyobb is mint 1.
  - A legrosszabb esetben: minden elem egy helyre képeződik le, így  $n$  hosszú lista keletkezik
    - Ez pedig  $\Theta(n)$  a keresés végrehajtási ideje (plusz a  $h$  kiszámítási ideje)

# Láncolásos hasítás műveletigénye

- Feltételezzük először, hogy a  $h$  hasító függvény egyenletesen osztja szét az elemeket, minden elem egyforma valószínűséggel képződik le bármelyik értékre, (ez az egyszerű egyenletes hasítási feltétel)

- Jelöljük a  $T[j]$  lista hosszát  $n_j$ -vel, ekkor

$$n = n_0 + n_1 + n_2 + \cdots + n_{m-1}$$

- $n_j$  várható értéke:  $E[n_j] = \frac{n}{m} = \alpha$
- Feltéve, hogy a  $h(k)$  érték  $\mathcal{O}(1)$  idő alatt számítható ki, akkor a  $k$  kulcsú elem keresése lineárisan függ a  $T[h(k)]$  lista hosszától.

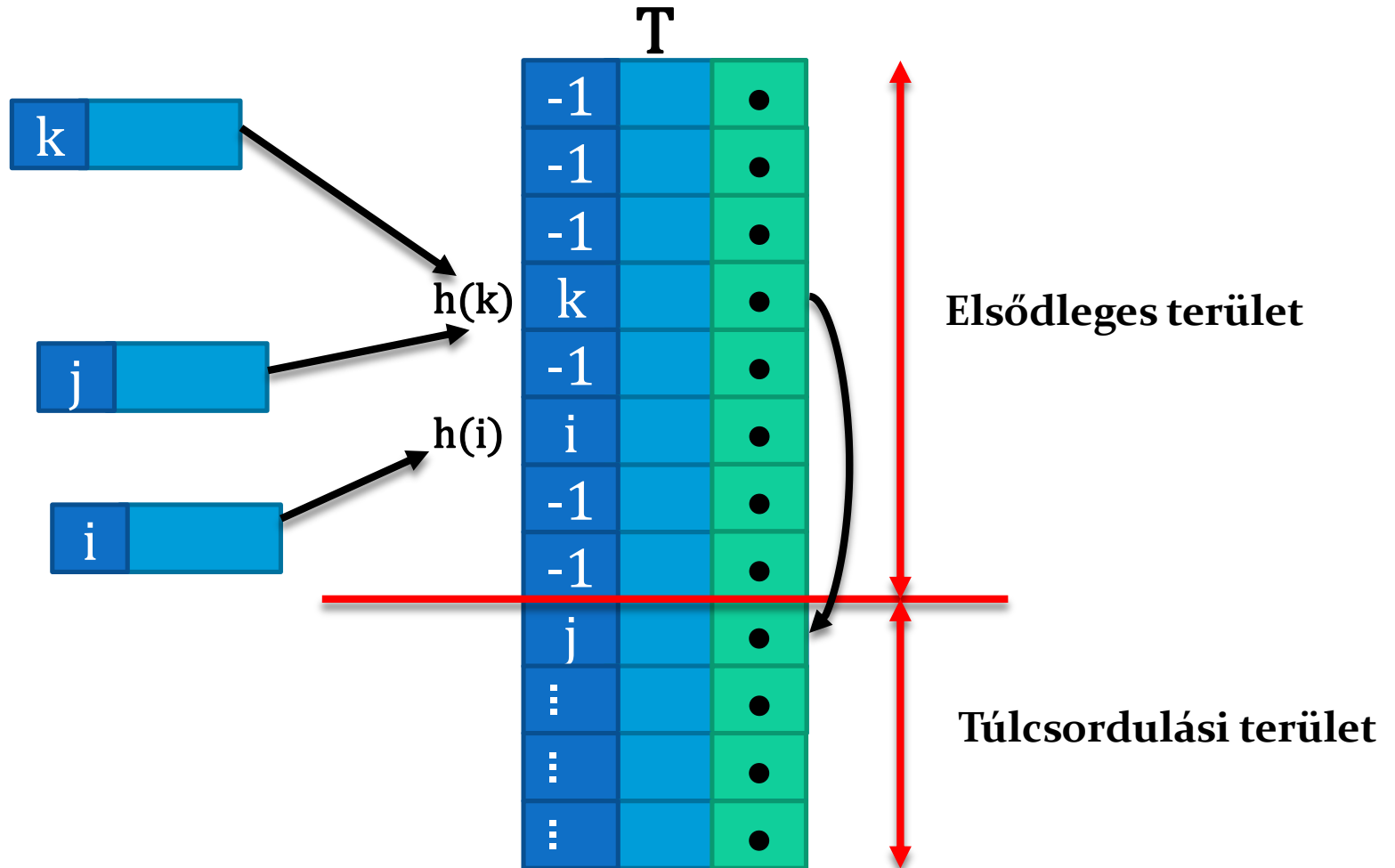
# Láncolásos hasítás műveletigénye

- Ha egy hasító táblázatban az ütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikeres és a sikertelen keresésekre is igaz, hogy az átlagos idejük  $\Theta(1 + \alpha)$ .
  - Bizonyítás: Cormen könyvében
- Mi következik ebből? Ha a hasító táblázat réseinek száma arányos a táblázatbeli elemek számával, akkor  $n = \mathcal{O}(m)$ , így  $\alpha = \frac{n}{m} = \frac{\mathcal{O}(m)}{m} = \mathcal{O}(1)$ .
- Belátható, hogy a beszúrás és törlés legrosszabb ideje is  $\mathcal{O}(1) \Rightarrow$  az összes művelet megvalósítható  $\mathcal{O}(1)$  idő alatt.

# Túlcsoordulási terület

- Túlcsoordulási terület
  - a „láncolt” listát a tábla egy speciális területén hozzuk létre – ez a **túlcsoordulási terület**
  - Tfh.  $h(k) = h(j)$
  - és a  $k$  lett először tárolva
  - Hozzáadjuk  $j$ -t:
    - Kiszámítjuk  $h(j)$ -t
    - Megtaláljuk  $k$ -t :-(
    - Megkeressük az első szabad helyet a túlcsoordulási területen
    - betesszük  $j$ -t
    - $k$  „pointere” erre mutat (a mutató itt a táblabeli index)
  - Többféle lehetőség a – lista elejére, végére is szúrhatunk be
  - Keresés – ugyanaz, mint a láncolt lista
  - Törlésnél – lehet a lista utolsó elemét idetenni, de lehet új állapotot bevezetni a törlőtre, ez majd jön

# Túlszordulási terület



# Nyílt címzés

- Az elemeket a táblában tároljuk.
  - Egy elem keresésénél végigmegyünk a táblázaton, amíg meg nem találjuk, vagy el tudjuk dönteni, hogy nincs benne a táblában.
  - Elem beszúrásánál **kipróbáljuk** az összes helyet, amíg üreset nem találunk – valamilyen stratégia szerint – ez a beszúrandó kulcs függvénye, nem a  $0 \dots m - 1$  felsorolás!
  - Kiterjesztjük a hash függvény értelmezési tartományát az ún. kipróbálási számmal:
    - $h: K \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$
    - Megköveteljük, hogy minden  $k$  kulcsra a  $(h(k, 0), h(k, 1), h(k, 2), \dots, h(k, m - 1))$  kipróbálási sorozat a  $\{0, 1, \dots, m - 1\}$  egy permutációja legyen
      - Így előbb-utóbb minden hely szóba jön

# Nyílt címzés

- Tegyük fel, hogy a T hasító táblázatban csak kulcsok vannak vagy NIL
- Beszúrás:

Hasító\_beszúr(T, k)

$i \leftarrow 0$

repeat  $j \leftarrow h(k, i)$

    if  $T[j] = \text{NIL}$

        then  $T[j] \leftarrow k$

        return

    else  $i \leftarrow i + 1$

until  $i = m$

error „hasító táblázat túlcsordulás”

# Nyílt címzés

- Keresésnél aszerint keressük, ahogy a beszúrás betehette

**Hasító\_keres**(T, k)

$i \leftarrow 0$

repeat  $j \leftarrow h(k, i)$

if  $T[j] = k$

then return j

$i \leftarrow i + 1$

until  $T[j] = \text{NIL}$  vagy  $i = m$

return NIL

# Nyílt címzés

- Törlés: bonyolultabb, ugyanis nem elég csak NIL-t írni, hiszen akkor egy következő keresés nem találná meg azokat, amelyek később vannak a táblába
  - vezessünk be a NIL-en kívül még egy szimbólumot, a TÖRÖLT-et

Hasító\_töröl( $T$ ,  $k$ )

$i \leftarrow 0$

repeat  $j \leftarrow h(k, i)$

if  $T[j] = k$

then  $T[j] \leftarrow \text{TÖRÖLT}$

return

$i \leftarrow i + 1$

until  $T[j] = \text{NIL}$  vagy  $i = m$

# Nyílt címzés

- A beszúrást is módosítani kell, hiszen most már a TÖRÖLT-be is be lehet szúrni:

**Hasító\_beszúr( $T, k$ )**

$i \leftarrow 0$

repeat  $j \leftarrow h(k, i)$

    if  $T[j] = \text{NIL}$  vagy  $T[j] = \text{TÖRÖLT}$

        then  $T[j] \leftarrow k$

        return

    else  $i \leftarrow i + 1$

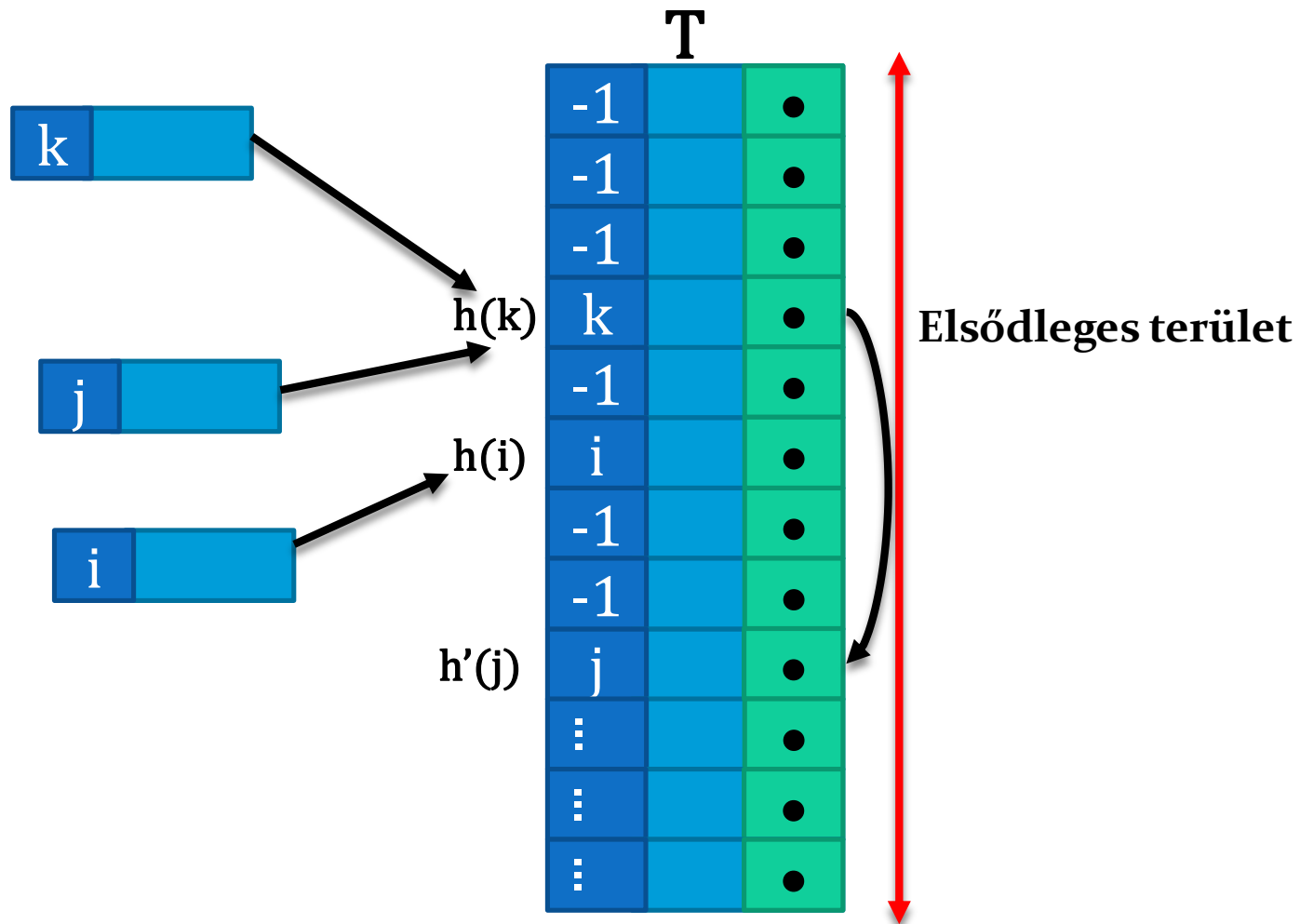
until  $i = m$

error „hasító táblázat túlcsordulás”

# Re-hashing – kettős hashelés

- Használjunk egy második hash függvényt –  $h'$ 
  - Sok variáció
  - általános név: **re-hashing – kettős hashelés**
- $h(k) = h(j)$
- $k$  lett először tárolva
- Hozzáadjuk  $j$ -t:
  - Kiszámítjuk  $h(j)$ -t
  - Így megtaláljuk  $k$ -t
  - Ismételjük, míg találunk üres helyet:
    - Kiszámítjuk  $h'(j)$ -t
    - Betesszük  $j$ -t
- Keresés – használd  $h(x)$ -t, aztán  $h'(x)$ -t
- Egy mezőnek 3-féle státusza lehet
  - üres – foglalt – törölt (hogya a keresés folytatódhasson)

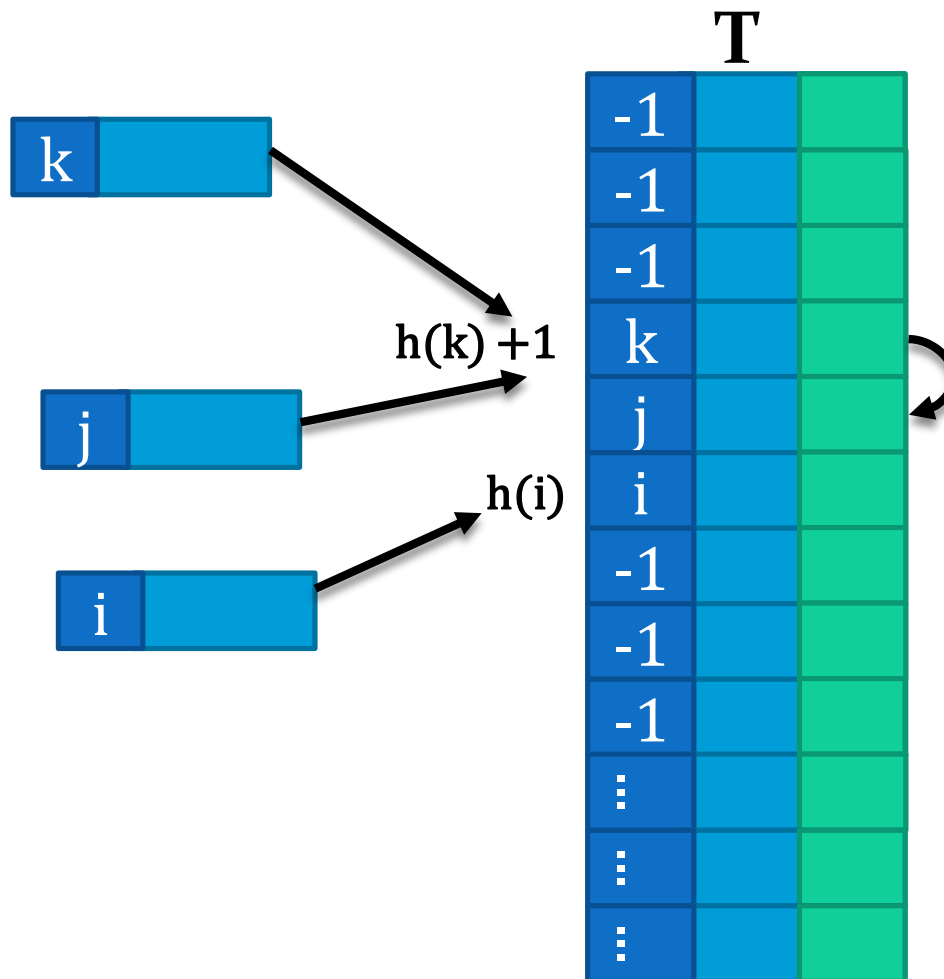
# Re-hashing – kettős hashelés



# Re-hash függvények

- A második hash függvény
  - Sok variáció
    - **Lineáris kipróbálás**
      - $h: K \rightarrow \{0, 1, \dots, m - 1\}$  hash függvény,
      - $h'(k, i) = (h(k) + i) \bmod m$
    - Próbáld először a  $T[h(k)]$ -t, aztán
    - Vegyük a következőt ciklikusan, amíg nem találunk egy üreset
    - Rossz csomósodások lehetnek
      - A Re-hash kulcsok kitöltik az üres helyet az egyéb kulcsok között és súlyosbítják a kulcsütközés problémáit – **elsődleges** csomósodás

# Re-hash függvények – lineáris kipróbálás



# Re-hash függvények

- A második hash függvény
  - Sok variáció
    - **Négyzetes próba**
      - $h'(k, i) = (h(k) + c_1 i + c_2 i^2) \bmod m$  az  $i$ . próbánál, ahol  $c_1, c_2 \neq 0, i = 0, 1, \dots, m-1$
      - Elkerüli az elsődleges csomósodást
      - Ahhoz, hogy az egész táblázatot lefedje, megkötések kellenek  $c_1, c_2$  és  $m$ -re
    - **Másodlagos csomósodás** lehet:
      - Minden kulcs, amelyik ütközik  $h$ -ban ugyanazon sorozat mentén
      - Először
        - $a = h(j) = h(k)$
      - Ez után pl.  $a + c, a + 4c, a + 9c, \dots$
      - Ez általában kisebb probléma

# Re-hash függvények

- A második hash függvény
  - Sok variáció
    - **Dupla hasítás**
      - $h'(k, i) = (h_1(k) + i h_2(k)) \bmod m$  az  $i$ . próbánál  $i \in \{0, 1, \dots, m - 1\}$ -re
      - A  $h_2(k)$  relatív prím kell legyen a táblázat  $m$  méretéhez képest.
        - Lehet például az  $m$ -t 2 hatványnak választani és  $h_2$ -t úgy választani, hogy mindig páratlan számot állítson elő
        - Lehet úgy is, hogy  $m$  prím és  $h_2$  mindig  $m$ -nél kisebb pozitív egész ad vissza

# Hasító táblázatok

- Potenciális  $O(1)$  keresési idő
  - Ha egy megfelelő  $h(\text{kulcs}) \rightarrow$  integer függvényt találunk
- „Hely a sebességért” kereskedelem
  - “Teljes” hasító táblázatok nem működnek
  - Az ütközések elkerülhetetlenek
  - A hash függvény csökkenti a kulcs információtartalmát
  - Különböző feloldási stratégiák
    - láncolt listák
    - túlcsordulási területek
    - Re-hash függvények
      - Lineáris próbálás  $h' = a + 1$
      - Négyzetes próbálás  $h' = a + ci^2$
      - Bármilyen más hash függvény!
        - vagy akár függvények sorozata!

# A hash függvény választása

- “Majdnem minden függvény jó lesz”
  - De bizonyos függvények egyértelműen jobbak, mint mások!
- Kulcs kritérium
  - ütközések minimális száma
    - röviden tartja a láncokat
    - karbantartja az  $\mathcal{O}(1)$  átlagot

# Hash függvény választása

- Egyszerű egyenletes hasítás
  - Ideális hash függvény
    - $P(k)$  = annak valószínűsége, hogy a  $k$  kulcs előfordul
    - ha van  $m$  hely a hasító táblánkban,
    - egy **egyenletes hash függvény**,  $h(k)$ , biztosítani fogja

$$\sum_{k|h(k)=0} P(k) = \sum_{k|h(k)=1} P(k) = \dots = \sum_{k|h(k)=m-1} P(k) = \frac{1}{m}$$

- Olyan  $k$  értékekre történik az összegzés, ahol a  $h(k) = 0$
- a kulcsok száma minden helyre azonos

# Egyenletes hash függvény

- Ha a kulcsok a  $[0, r)$ -on egyenletesen szétszórt egészek, akkor

$$h(k) = \left\lfloor \frac{mk}{r} \right\rfloor$$

egy **egyenletes hash függvény** (megmutatható)

- A legtöbb hash függvény megadható oly módon, hogy a kulcsokat valamely  $r$ -re a  $[0, r)$ -ra képezze le.

# Csökkentsünk a $[0, m)$ -ra

- A kulcsokat egészek egy intervallumára képeztük le:  
 $0 < k < r$
- Most csökkentsük ezt az intervallumot  $[0, m)$ -ra, ahol  $m$  a hash tábla egy elfogadható mérete
- Stratégiák:
  - **Osztás** – használjuk a mod függvényt
  - **Szorzás**
  - **Univerzális hashelés**

# Csökkentsünk a $[0, m)$ -ra

- **Osztás** – használjuk a mod függvényt  
$$h(k) = k \bmod m$$

- $m$  választása?

- A 2-hatványok általában nem jók!  
 $h(k) = k \bmod 2^n$  a  $k$  **utolsó**  $n$  **bit**jét választja

$k \bmod 2^8$  ezeket választja

0110010111000011010

- Általában nem egyformán valószínű minden kombináció
- Jobb olyan hasító függvényt választani, ami a kulcs összes bitjétől függ
- A  $2^n$ -hez közeli **prímek** jó választásnak tűnnek
- Például  $\sim 4000$  méretű tábla kellene, válasszunk  $m = 4093$ -t

# Csökkentsünk a $[0, m)$ -re

- **Szorzó** módszer

- Szorozzuk meg a kulcsot egy  $A$  konstanssal,  $0 < A < 1$
- Vegyük ki belőle a tört részt ( $kA - \lfloor kA \rfloor$ )
- Szorozzuk meg  $m$ -mel  $h(k) = \lfloor m * (kA - \lfloor kA \rfloor) \rfloor$
- Most  $m$  nem kritikus, és 2 hatvány választható
- Így ez a módszer gyors egy tipikus digitális számítógépen
  - Legyen  $m = 2^p$
  - Szorozzuk meg  $k$ -t ( $w$  bit)  $\lfloor A \cdot 2^w \rfloor$ -val  $\leftarrow 2^w$  bit szorzat
  - vedd ki a  $p$  alsó felének a legszignifikánsabb bitjeit
  - $A = \frac{1}{2}(\sqrt{5} - 1)$  jó választásnak tűnik (lásd Knuth)

# Univerzális hashelés

- Ha egy rosszakarónk válogatja ki a hasító-táblába kerülő kulcsokat, tud adni olyan sorozatot esetleg, hogy mind az  $n$  elemre ugyanaz legyen a  $h(i)$  érték, s így a keresés átlagos ideje  $\mathcal{O}(n)$  legyen.
- Univerzális hasító technika: a hash-függvényt véletlenül, az aktuálisan tárolandó kulcsoktól függetlenül választjuk meg – ez jó átlagos teljesítményhez vezet.
- Alapgondolat: a hasító függvényt egy gondosan megtervezett függvényosztályból futás közben véletlenül választjuk ki
  - Így nem lehet olyan bemenet, ami biztosan a legrosszabb viselkedést váltja ki.

# Univerzális hashelés

- Legyen  $H$  hasító függvények egy véges halmaza, melyek egy adott  $K$  kulcsuniverzumot a  $[0, m)$  tartományba képeznek le.
- A  $H$ -t univerzálisnak hívjuk, ha  $\forall x, y \in K, x \neq y$  kulcspárra azoknak a  $h \in H$  hasító függvényeknek a száma, amelyre  $h(x) = h(y)$  pontosan  $\frac{|H|}{m}$
- Ez azt is jelenti, hogy egy véletlenül választott  $h \in H$  hasító függvényre  $\forall x, y \in K, x \neq y$  kulcsok közötti kulcsütközés valószínűsége pontosan  $\frac{1}{m}$ , ami ugyanaz, mint a  $\{0, 1, \dots, m - 1\}$  halmazból véletlenül kiválasztott  $h(x)$  és  $h(y)$  egyenlőségének valószínűsége

# Univerzális hashelés

- Meg tudjuk tervezni univerzális hash függvények egy halmazát?
- Elég könnyen:
  - Válasszunk egy olyan  $p$  prímszámot, amely elég nagy, hogy minden kulcs benne legyen a  $[0 \dots p - 1]$ -ben ( $p > m$ )
  - Jelölés:  $Z_p = \{0, 1, \dots, p - 1\}$ ,  $Z_p^* = \{1, 2, \dots, p - 1\}$
  - Definiáljuk:  $\forall a \in Z_p^*, \forall b \in Z_p$ ,
    - $h_{a,b}(k) = ((a * k + b) \bmod p) \bmod m$
  - Az ilyen függvények osztálya:
    - $H_{p,m} = \{ h_{a,b} : a \in Z_p^*, b \in Z_p \}$
- Tétel:
  - A hasító függvények fenti egyenlőségekkel definiált  $H_{p,m}$  osztálya univerzális.

# Hasító táblázatok – Általános tervezés

- Válasszuk meg a tábla méretét
  - A nagy tábla csökkenti az ütközések valószínűségét!
  - tábla méret:  $m$
  - $n$  elem
  - ütközések valószínűsége  $\alpha = \frac{n}{m}$
- Válasszuk meg a tábla szervezését
  - növekedni fog a gyűjtemény?
    - Láncolt listák – Gondoljunk a fákra is!
    - A méret relatíve statikus?
    - Túlcsordulási terület vagy
    - Re-hash
- Válasszunk egy hash függvényt



# Hasító táblázatok - Általános tervezés

- Válasszunk egy hash függvényt
  - Egy egyszerű (és gyors) jó lenne ...
  - Olvassunk irodalmat jó ötletekért!
- Vizsgáljuk a hash függvényt az adatainkkal
  - Fix adatokkal
    - Próbáljunk különböző  $h$ ,  $m$  értékeket amíg a maximális ütközési lánc elfogadható lesz
    - Ismert hatékonyság
  - Változó adatokkal
    - Válasszunk jellemző adatokat
    - Próbáljunk különböző  $h$ ,  $m$  értékeket amíg a maximális ütközési lánc elfogadható lesz
    - Általában megjósolható hatékonyság