

ADATSZERKEZETEK ÉS ALGORITMUSOK

„Hasítási technikák”
Kulcsütközés és feloldása

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

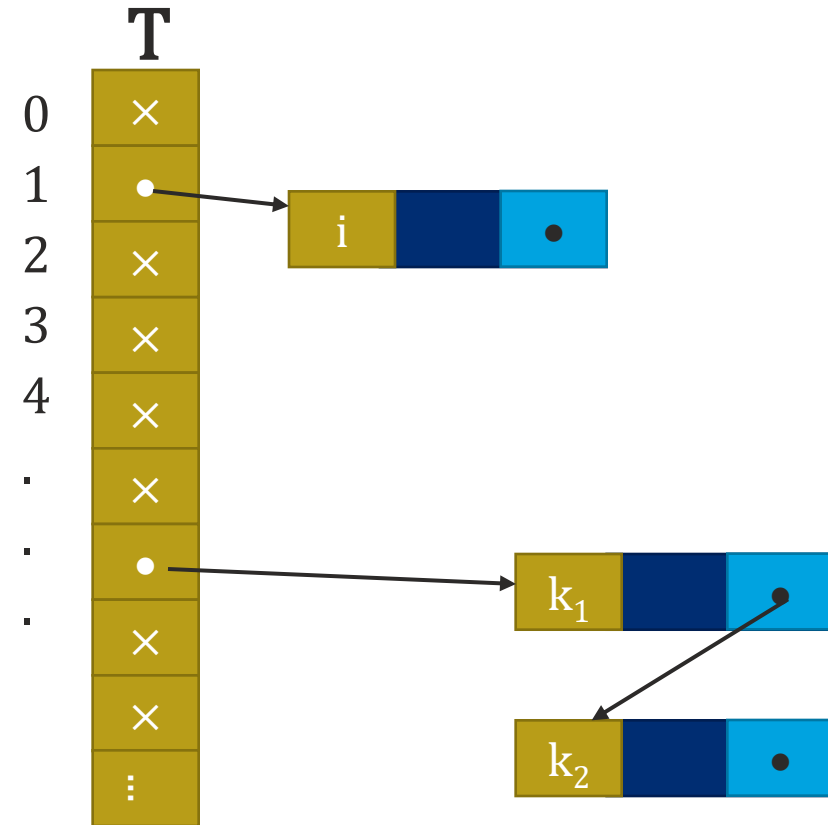
# Kulcsütközés feloldása

- Kulcsütközés

- 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

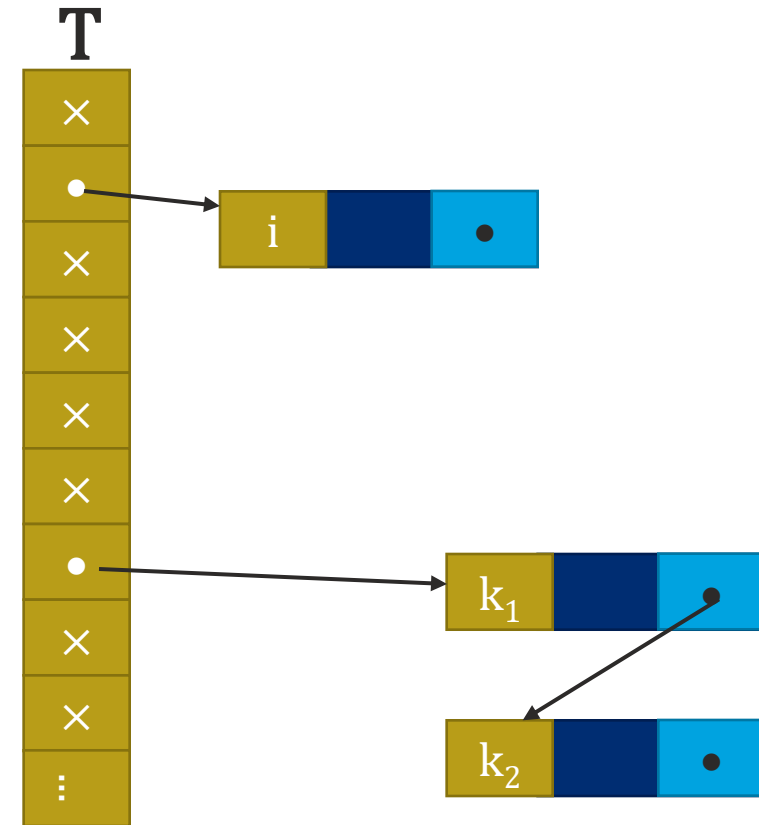
# Láncolt listák

- 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:
  - `Láncolt_hasító_beszúrás(T,x)`
    - kiszámítjuk  $h(x.kulcs)$ -t
    - Beszúrunk a  $T[h(x.kulcs)]$  lista elejére
- Törlés – hasonlóan a  $T[h(x.kulcs)]$  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
$$n = n_0 + n_1 + n_2 + \dots + n_{m-1}$$
  - $n_j$  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

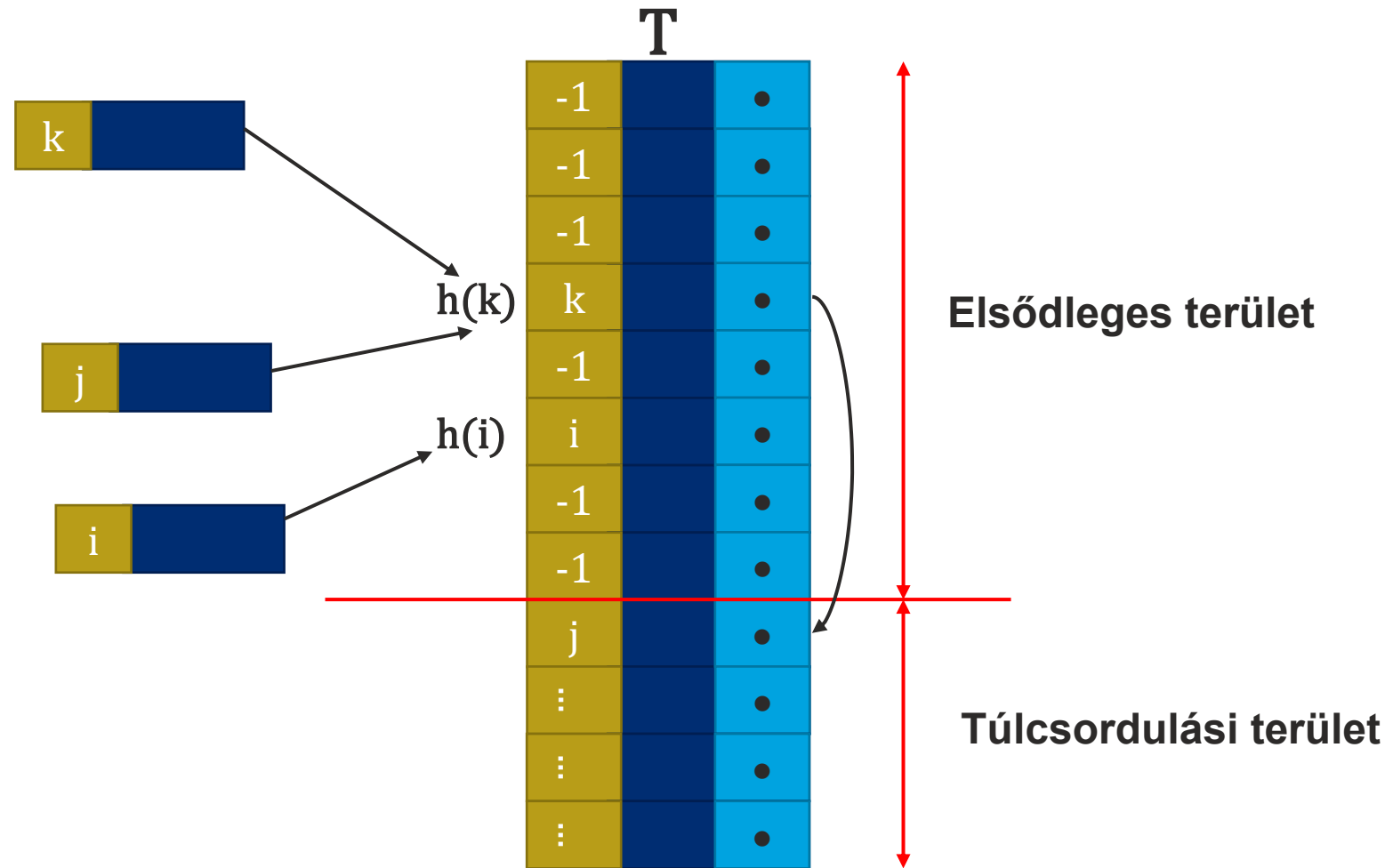
- Ha egy hasító táblázatban az ütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikeres és a sikertelen keresésekre is igaz, hogy az átlagos idejük  $\Theta(1 + \alpha)$ .
  - Bizonyítás: Cormen könyvében
- Mi következik ebből? Ha a hasító táblázat réseinek száma arányos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

# Túlcsordulási terület

- Túlcsordulási terület

- a „láncolt” listát a tábla egy speciális területén hozzuk létre – ez a **túlcsordulási terület**
  - Tfh.  $h(k) = h(j)$
  - és a  $k$  lett először tárolva
  - Beszúrjuk a  $j$ -t a hash táblába
    - Kiszámítjuk  $h(j)$  értéket
    - Megtaláljuk  $k$ -t, ami a táblában már benne van
    - Megkeressük az első szabad helyet a túlcsordulási területen
    - Oda betesszük  $j$  értéket
    - $k$  „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 töröltre, ez majd jön

# Túlcsondulási 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  $i = m$

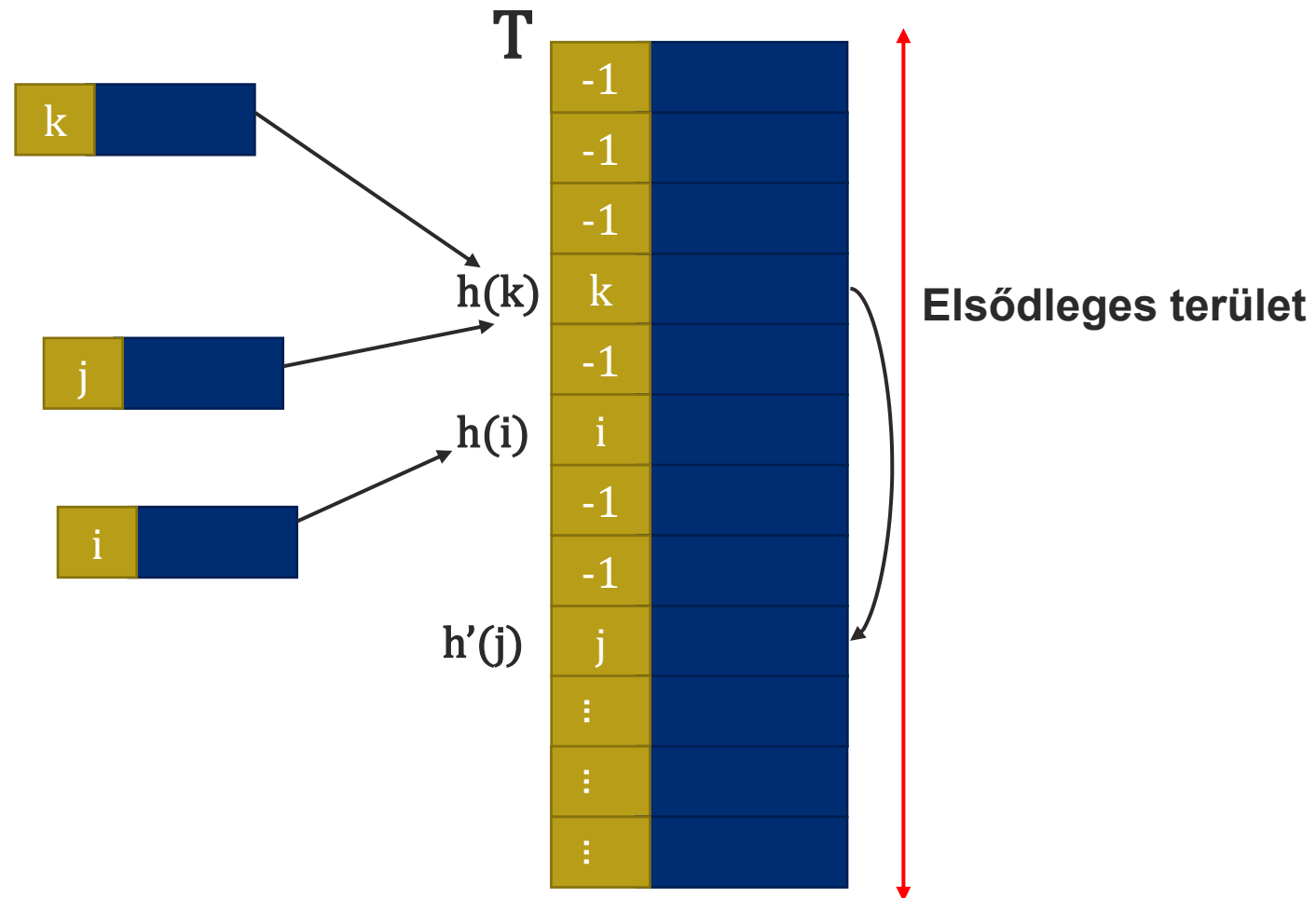
error „hasító táblázat túlcsordulás”



# 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
- Beszúrjuk  $j$ -t a hash táblába
  - Kiszámítjuk  $h(j)$  értéket
  - Így megtaláljuk  $k$  kulcsot, ami már a táblában van
  - 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)

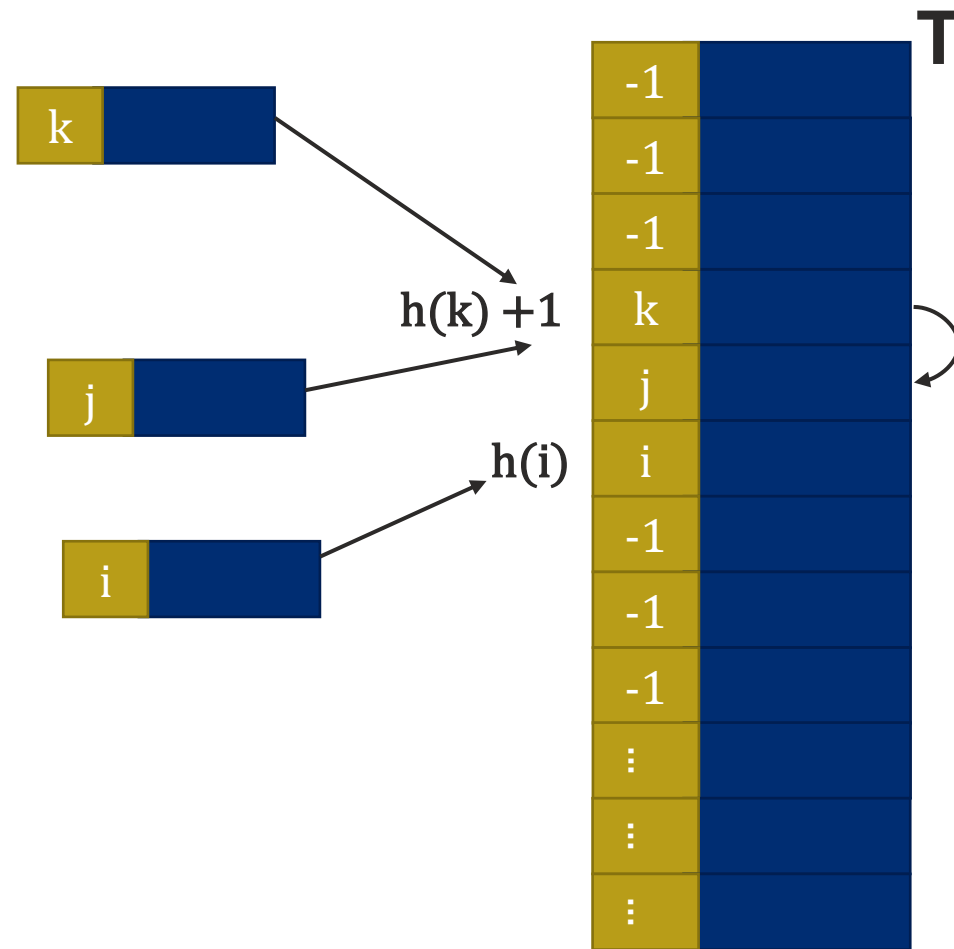
# Re-hashing – kettős hashelés



# 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 kellene  $c_1$ ,  $c_2$  és  $m$ -re
      - **Másodlagos csomósodás** lehet:
        - Minden kulcs, amelyik ütközik  $h$ -ban ugyanazon sorozat mentén
        - Először
          - $a = h(j) = h(k)$
        - Ez után pl.  $a + c$ ,  $a + 4c$ ,  $a + 9c$ , ....
        - Ez általában kisebb probléma

# Re-hash függvények

- A második hash függvény

- Sok variáció

- Dupla hasítás

- $h'(k, i) = (h_1(k) + ih_2(k)) \bmod m$  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 egészet ad vissza