ADATSZERKEZETEK ÉS ALGORITMUSOK

„Hasításos technikák”
Hasításos technikák

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	státusz	év	hó	nap	azonosító	
Példaérték	2	72	08	15	2806	
Értékek száma	8	100	12	31	999	~300M

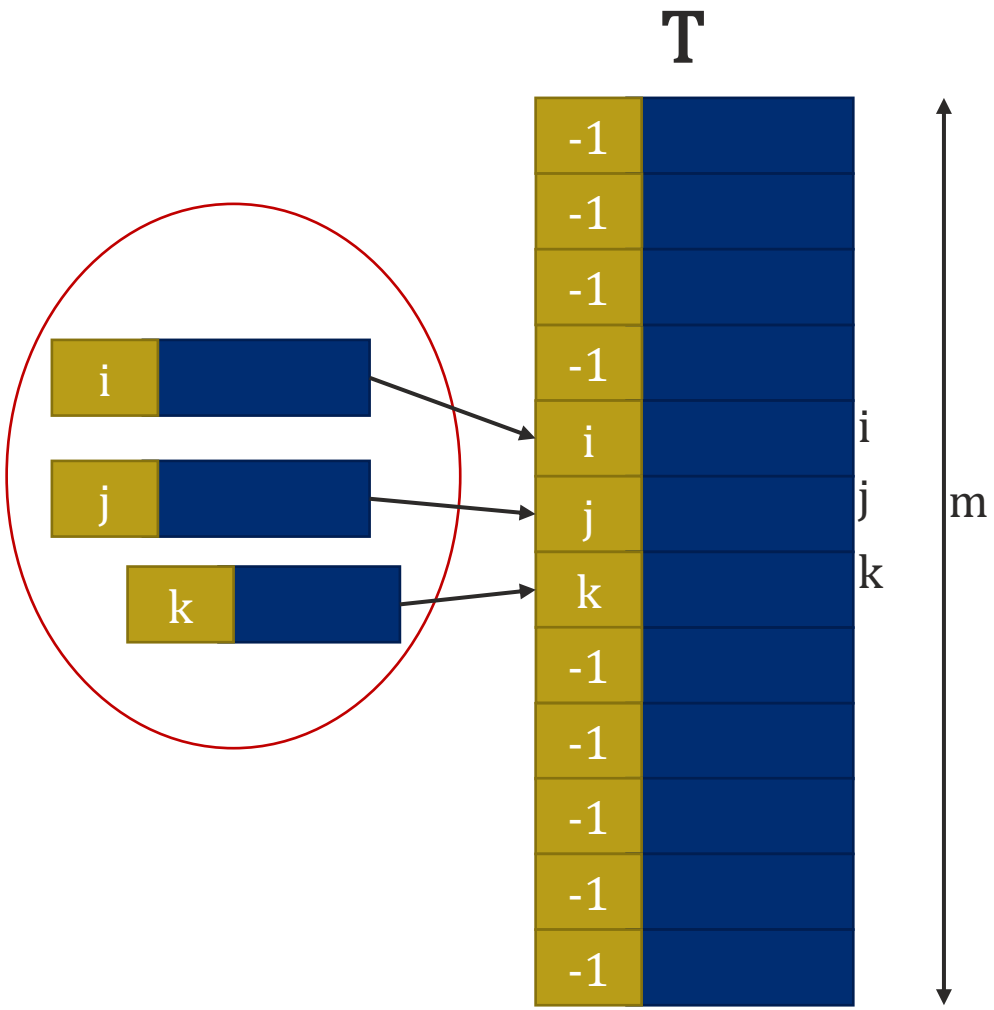
  - (Az utolsó 4 jegyből csak 3 értékes jegy, 1 a többiből képződik.)
- Ha összeszámoljuk az összes kitöltési lehetőséget, akkor ~ 300 milliót kapunk …
- A ~ 10 millió lakosra pedig elég egy ~ 12 millió rekordot tartalmazó fájl.
  - A személyi szám tehát nem alkalmas kulcsnak.
- Kell egy olyan h függvény, 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 $[0\,..\,M-1]$ intervallum.
  - A $h: K \to [0\,...\,M-1]$ 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 hasításos technikák feladata éppen az, hogy megoldást adjanak a kulcsütközésre.
- Fontos kérdés
  - Milyen hatékonyságot várhatok?

Hasító táblázatok

- A közvetlen hozzáférésű (címzésű) 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 táblában
  - 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.
- Komplexitás: $\mathcal{O}(1)$
  - Beszúrás, törlés is



Hasító táblázatok

```
Közvetlen_címzésű_keresés (T, k)
        return T[k]
Közvetlen_címzésű_beszúrás (T, x)
        T[x.kulcs] ← x
Közvetlen_címzésű_törlés (T, x)
        T[x.kulcs] ← -1
```

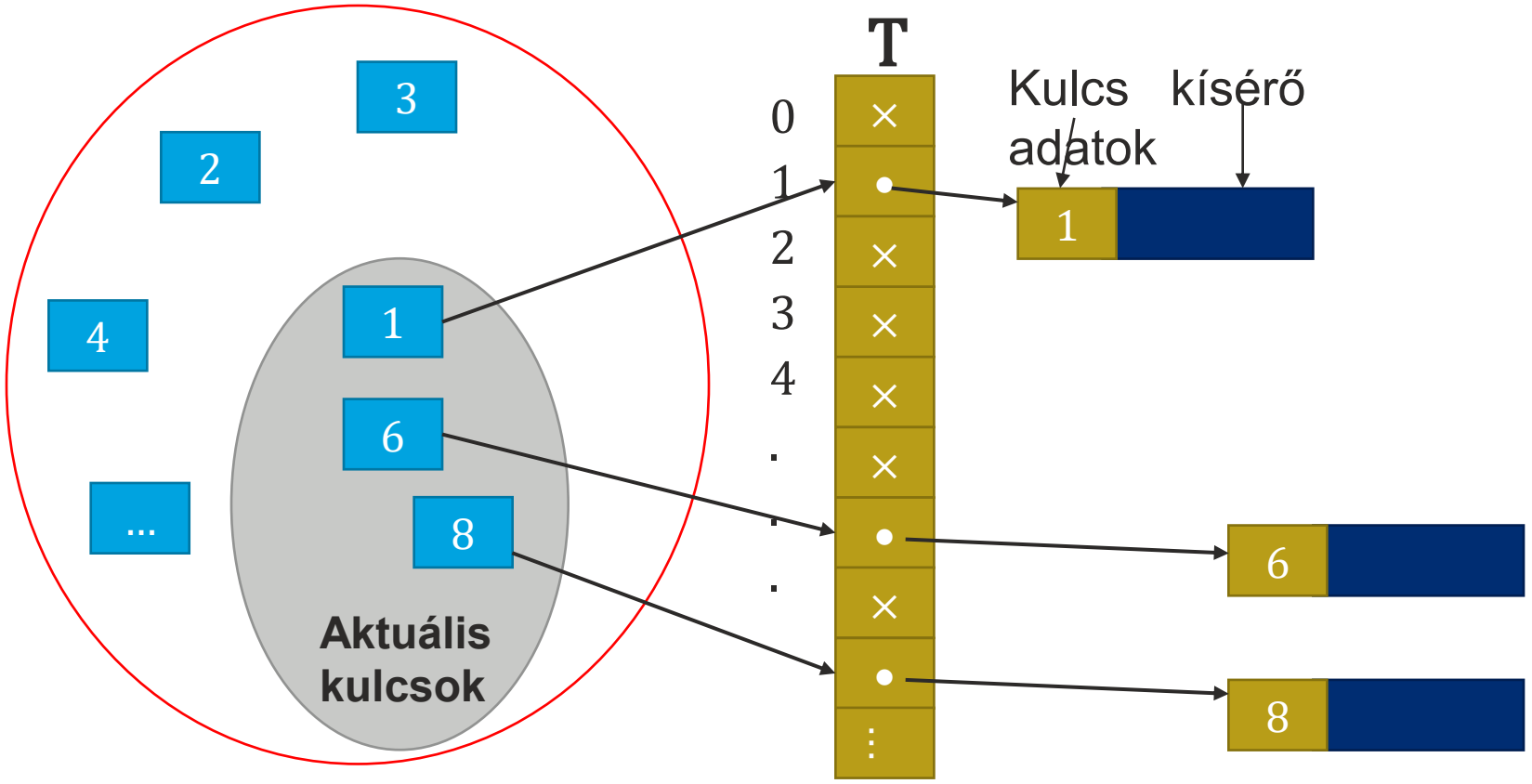
Mindegyik $\mathcal{O}(1)$ idejű

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 címzésű táblák: T-indexei a kulcsok, T-ben mutatók, csak akkor tárolom az egészet, ha kell:



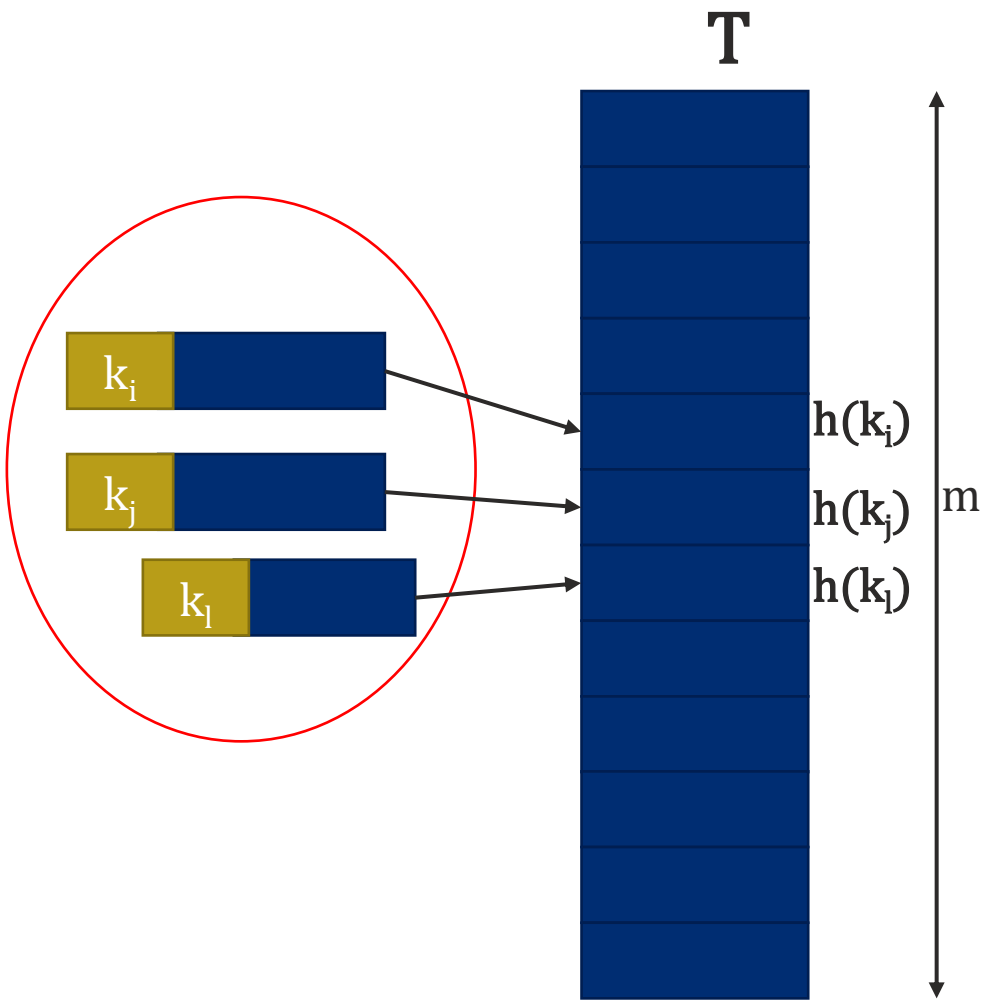
Hasító táblázatok

```
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

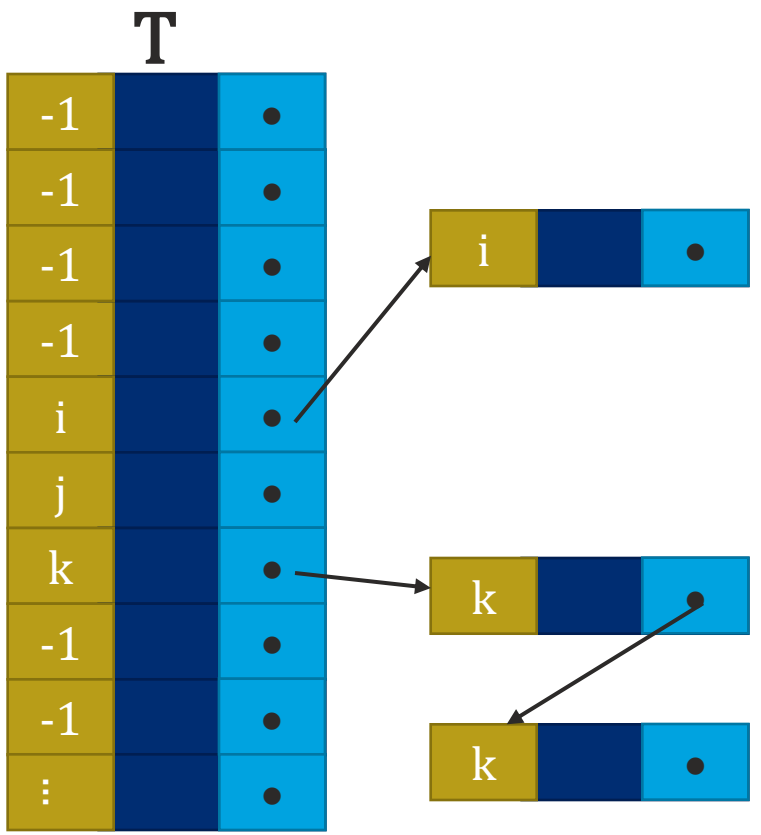
A megszorítások gyengítése

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

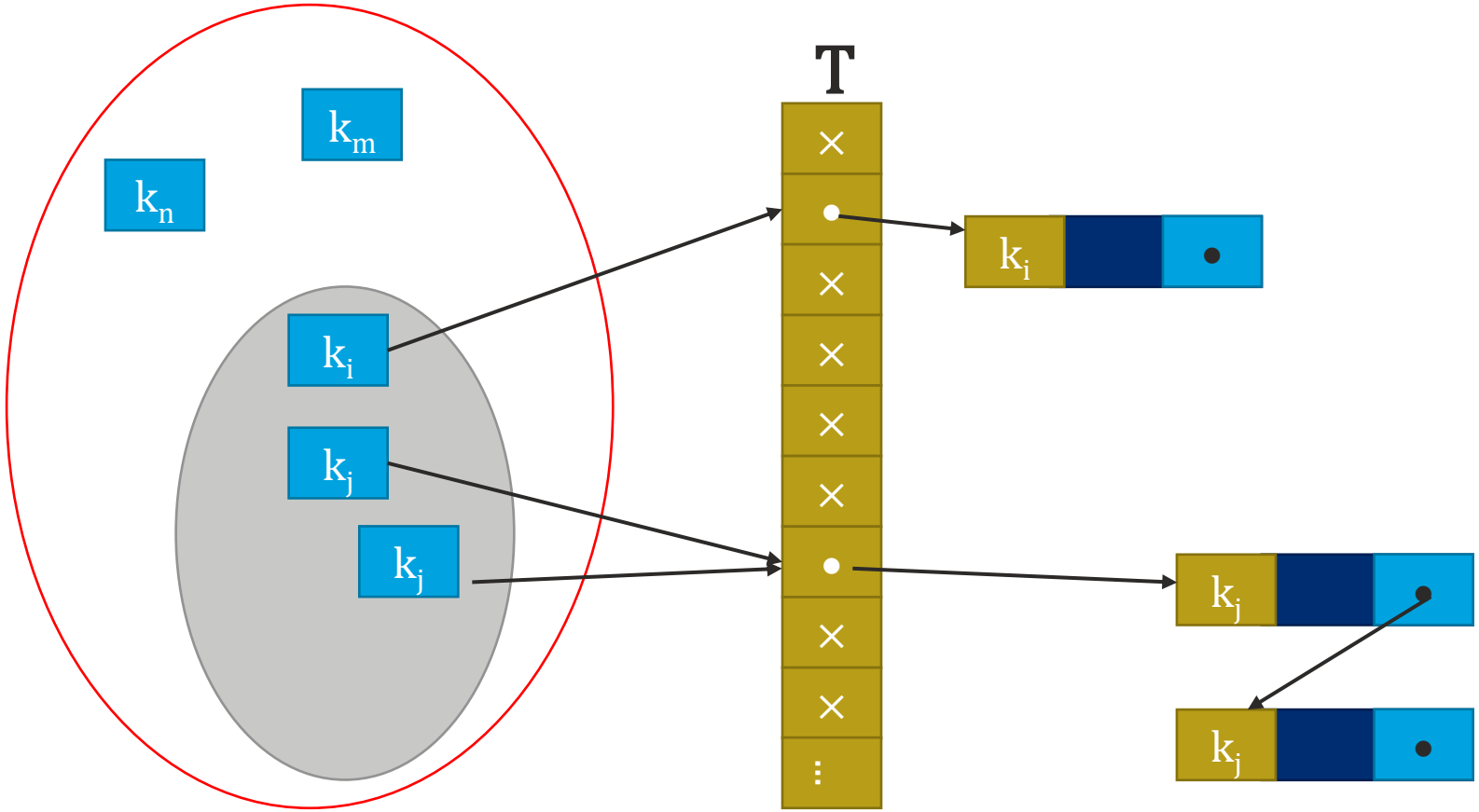


A megszorítások gyengítése

- 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



Láncolásos hasítás műveletei

T inicializálása

  minden elemét null-ra állítjuk

Láncolt_hasító_keresés (T, k)

  a k kulcsú elem keresése a T[h(k)] listában

Láncolt_hasító_beszúrás (T, x)

  helyet foglalunk x-nek,
  beszúrjuk a T[h(x.kulcs)] lista elejére

Láncolt_hasító_törlés (T, x)

  x törlése a T[h(x.kulcs)] listából

  (T[x.kulcs] is null lesz!)

Hash függvények

- A hash függvény formája
  - Példa – egy n-karakteres kulcsot használva

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

ez a függvény a $0 \ldots 255$ 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 $\mathcal{O}(1)$!