ADATSZERKEZETEK ÉS ALGORITMUSOK

Hash tábla demonstráció

Motiváció

- Nagyon sok adatban kell nagyon gyorsan keresni
- O(log(n)) nem elég gyors
- Háttértárat/memóriát használunk a processzoridő helyett
- Konstans idejű
  - Beszúrás
  - Törlés
  - Keresés

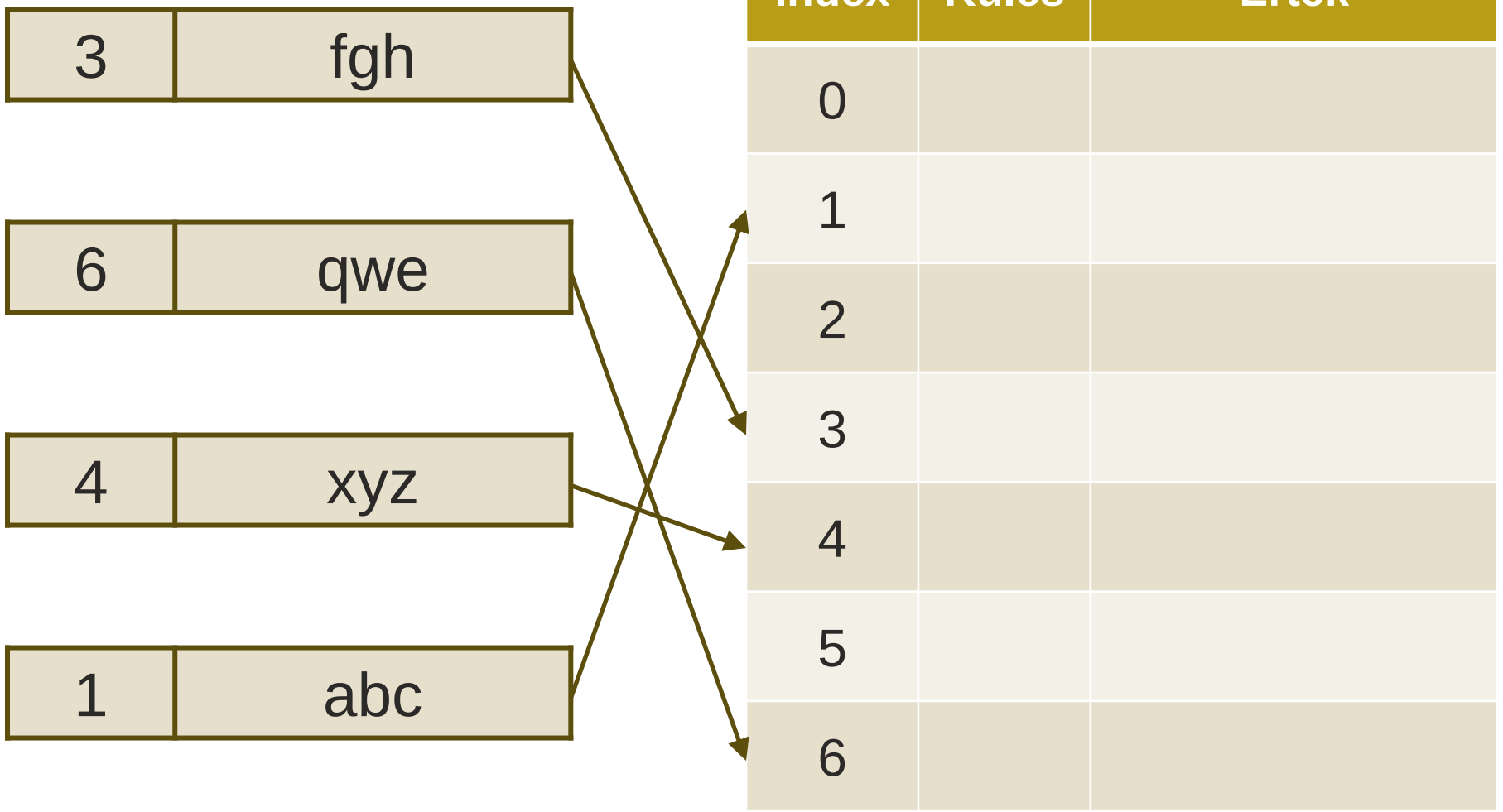
Az alapötlet

- Az elem helyét *(indexét)* a kulcsából generáljuk
- Az index független a többi értéktől, nincs összehasonlítás
- Konstans idő alatt generálható

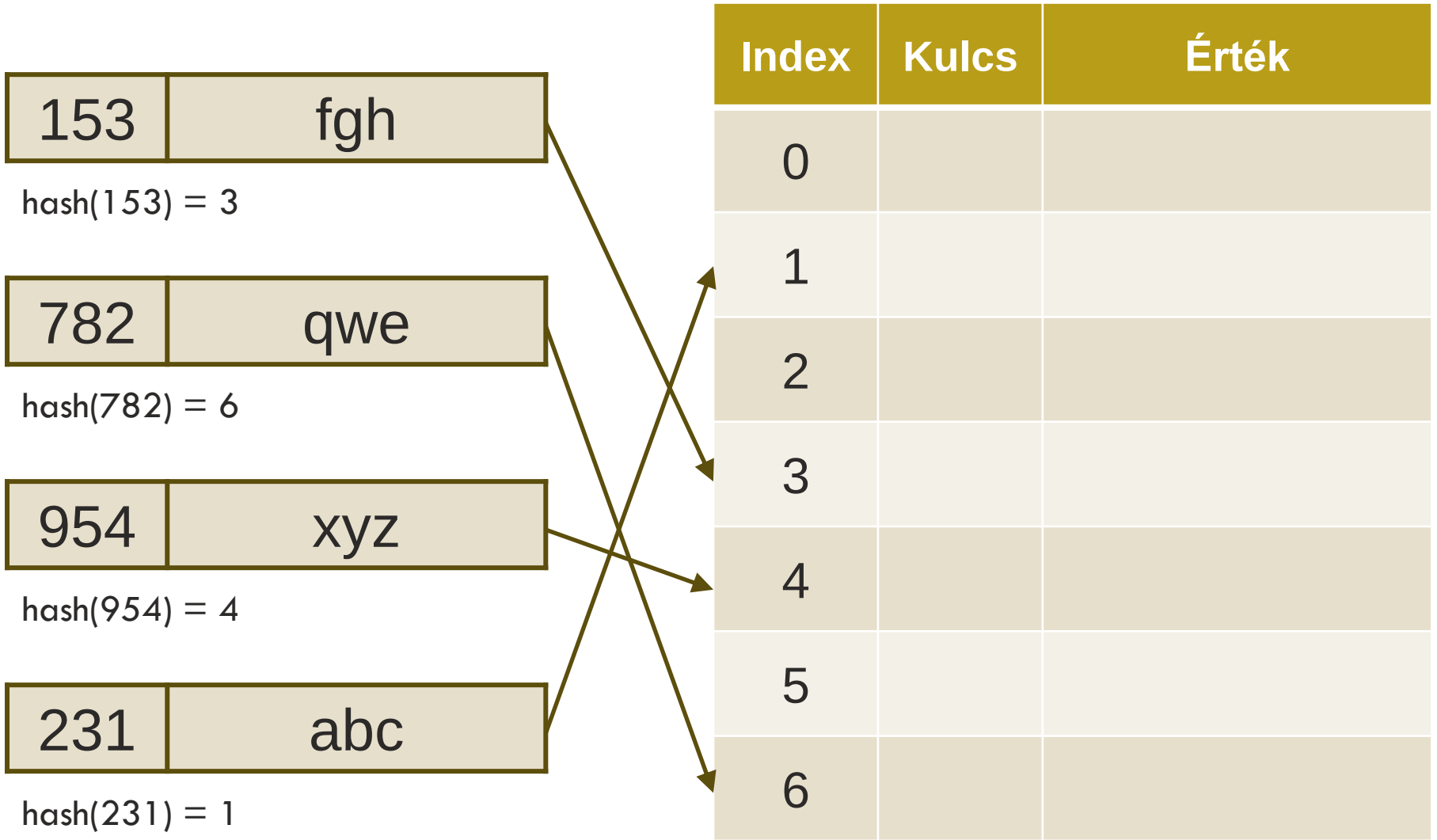
Hogyan válasszunk indexet?

- Ha a kulcs rövid intervallumon vesz fel értékeket (mindegyiknek tudunk helyet foglalni), akkor azt választjuk indexnek.
- Ha az intervallum széles, akkor ez túl nagy méretű tömböt indexelne. Ekkor a kulcsot egy ún. *hashfüggvénnyel* szűkebb tartományra vetítjük.

Kulcs mint index



Szűkített (hashelt) index



A hashfüggvény

- Kritériumok:
  - Determinisztikus – ugyanazt a kulcsot mindig ugyanarra az indexre képezi le
  - Egyenletes eloszlású kimenet – nem képez le sok elemet ugyanoda
  - Konstans idejű

Egy egyszerű hashfüggvény

- Kulcsok halmaza: 32 bites előjeles számok (-2 147 483 648 … 2 147 483 647)
- Indexek halmaza: 0…n-1, ahol n a hashtáblánk mérete
- Modulo n

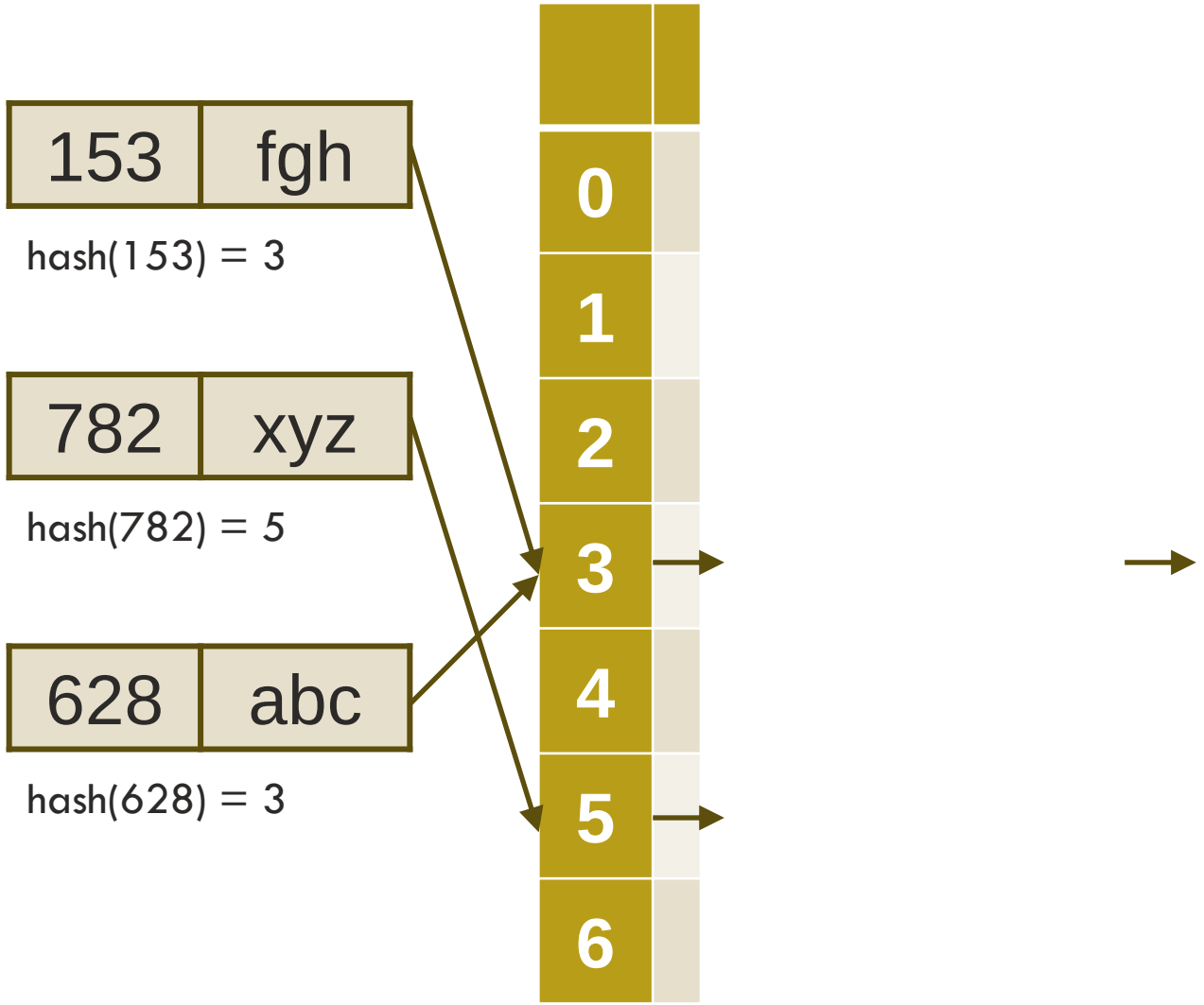
Kulcsütközések

- Még a legjobb hashfüggvénynél is előfordulhat, hogy két különböző kulcsot azonos indexre képez le.
- Ez elkerülhetetlen, mivel egy szélesebb intervallumról egy szűkebbre vetítünk.

Kulcsütközések feloldása

- Többféle módszer létezik. Néhány példa:
  - **Láncolt lista:** A tábla mezői nem egyetlen elemet, hanem egy listát tartalmaznak.
  - **Túlcsordulási terület:** A lista egy külön speciális területen található.
  - **Újrahashelés (nyílt címzés):** Speciális hashfüggvényt használunk, ütközés esetén újrageneráljuk az indexet.

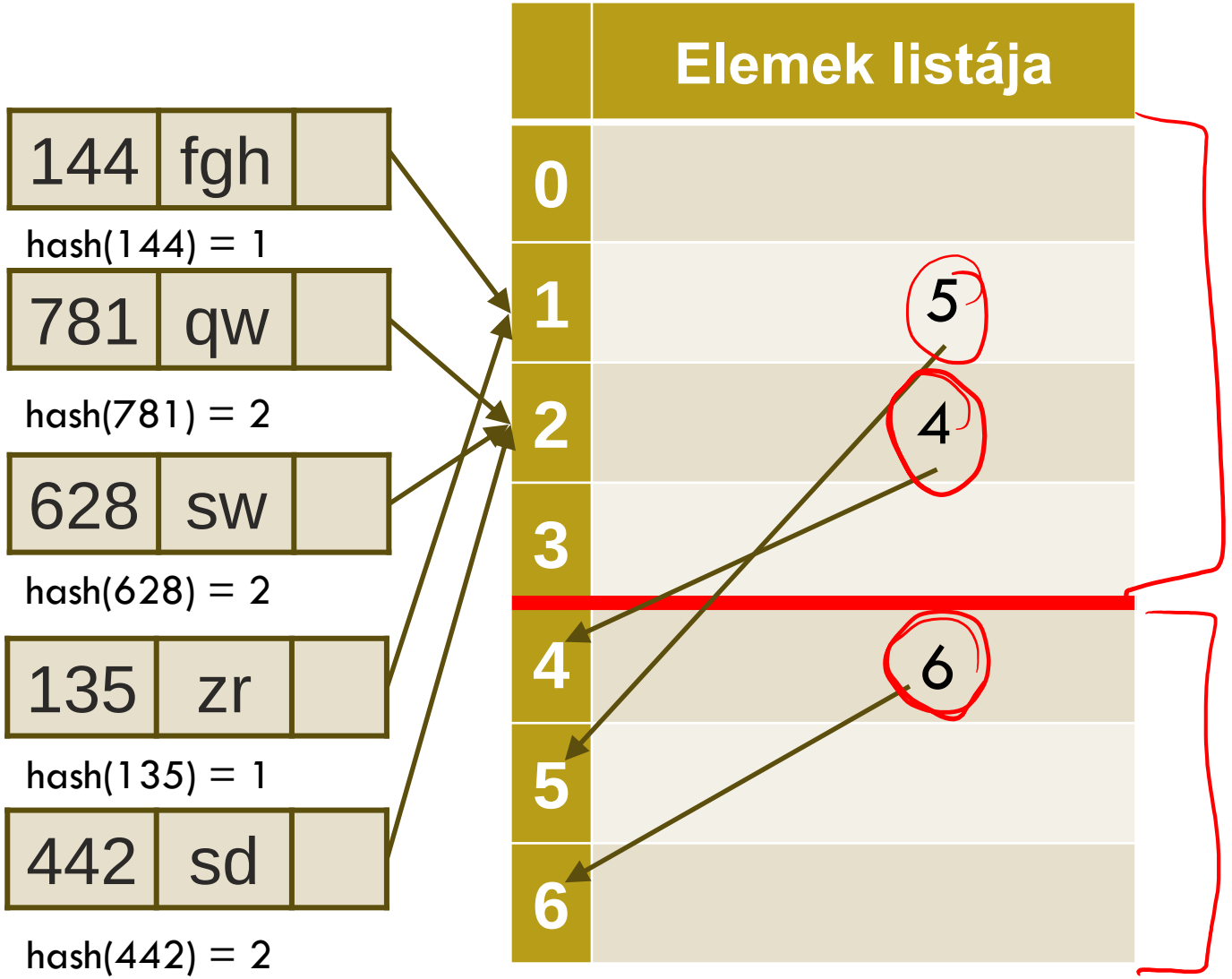
Listás ütközésfeloldás



Túlcsordulási terület

- A láncolt listát egy speciális területen hozzuk létre
- A táblában nem elférő elemeket ide tároljuk le
- Rámutatunk az előző listaelemből, ezzel létrehozva a láncolást

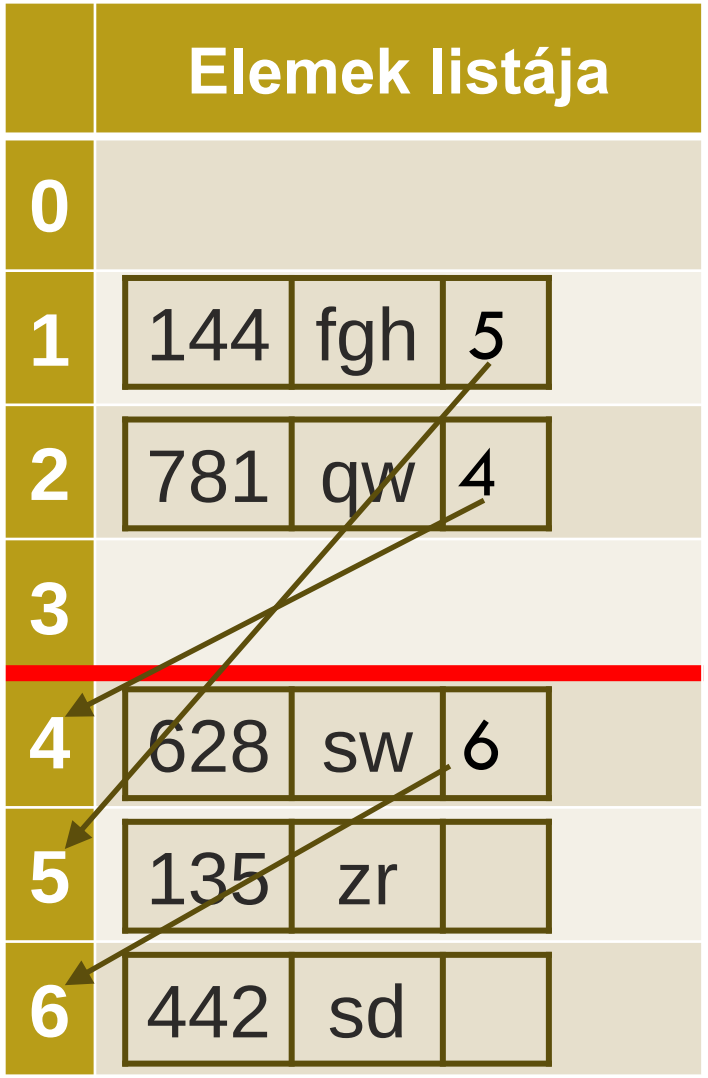
Túlcsordulási terület – beszúrás



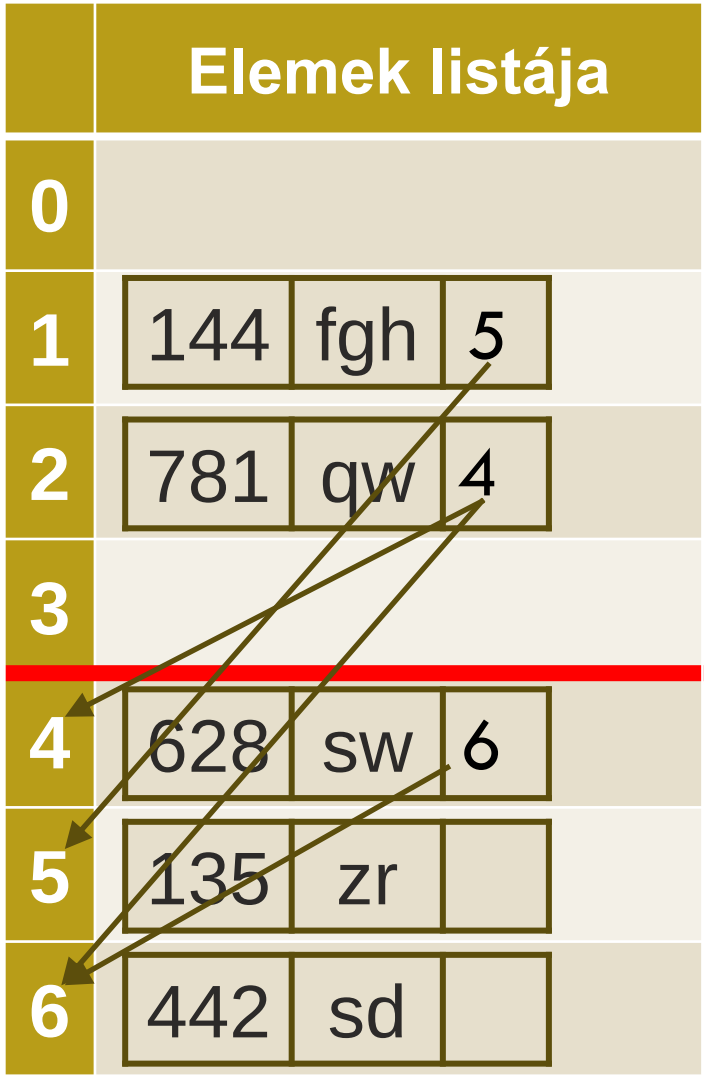
Túlcsordulási terület – törlés

- Ha a táblában található az elem
  - Kitöröljük, és ha van ahhoz a hashértékhez elem a túlcsordulási területen, akkor onnan a lista első elemét feltesszük a táblába
- Ha a túlcsordulási területen található az elem
  - Kitöröljük és átláncoljuk
- Ez nem teljesen ugyanaz, mint ami az előadáson szerepelt!

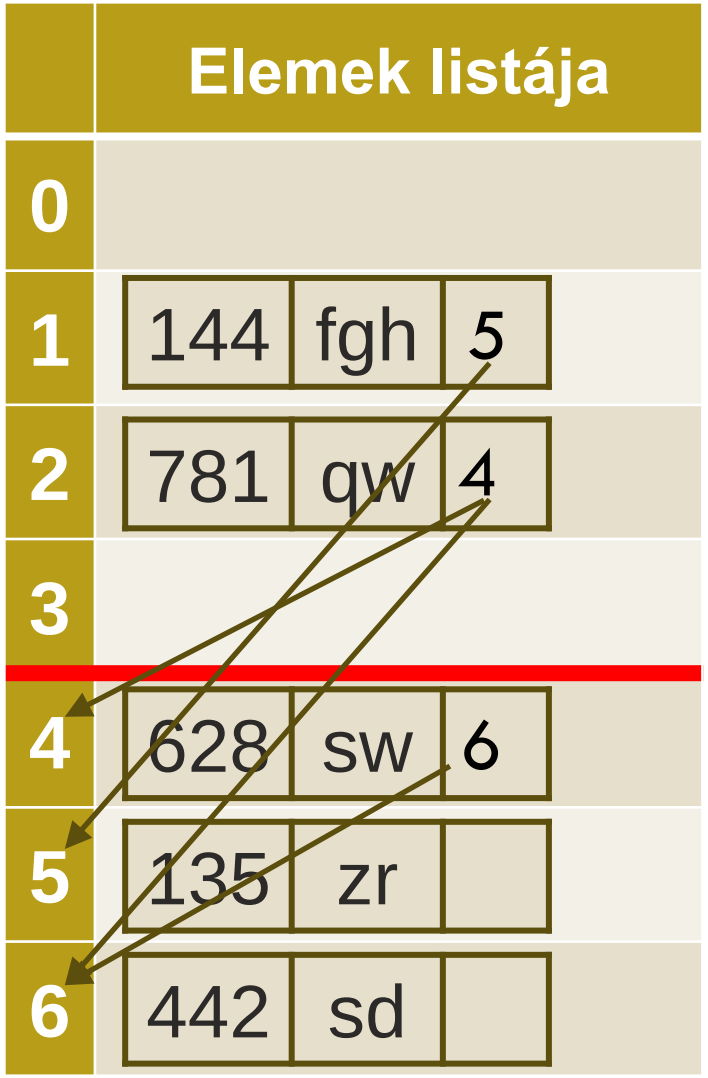
Túlcsordulási terület – törlés



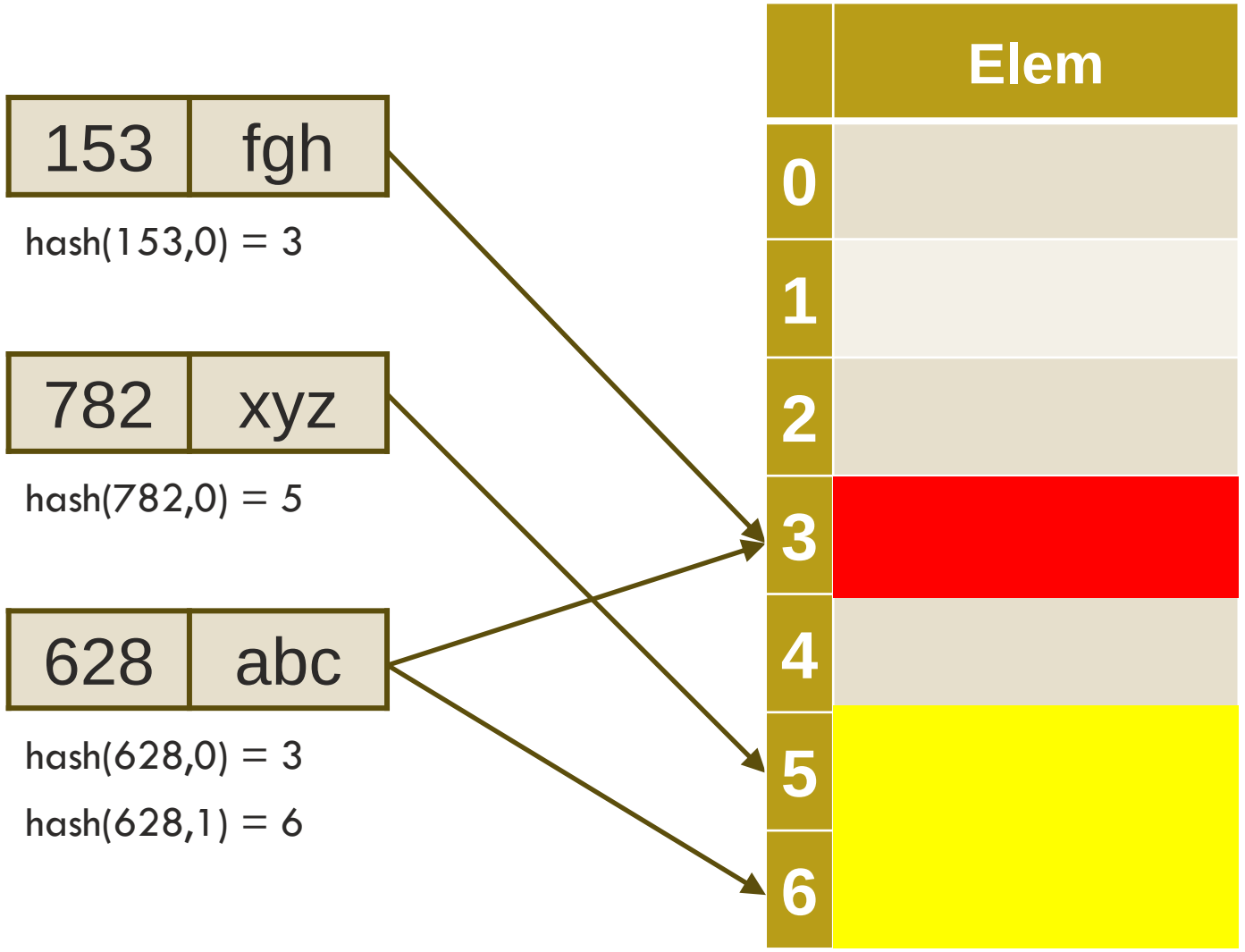
Túlcsordulási terület – törlés



Túlcsordulási terület – törlés



Újrahasheléses ütközésfeloldás



Újrahashelés – beszúrás

- Az ötlet az, hogy a hashfüggvény egy kulcshoz ne csak egy indexet rendeljen, hanem egy indexsorozatot.
- Ehhez kiegészítjük a függvényt még egy paraméterrel, amely kiválaszt a sorozatból egy konkrét indexet.
- Például ha k egy kulcs, és a hash(k,0) egy foglalt hely indexét adja vissza, akkor a hash(k,1)-gyel próbálkozunk. Ha ez is foglalt, akkor hash(k,2) jön, és így tovább.

Újrahashelés – keresés

- Korábban, az elem beszúrásakor lehet, hogy több helyet is kipróbáltunk, mielőtt betettük volna a táblába.
- Ezért kereséskor is addig kell próbálkoznunk az indexekkel, amíg vagy meg nem találjuk a keresett elemet, vagy üres helyre nem érünk.
- Tehát ha a hash(k,0) által kijelölt helyen nem egy k kulcsú elem van, akkor meg kell nézni a hash(k,1)-edik helyet is, ami esetleg szintén hamis találat, és így tovább.
- Az is előfordulhat, hogy egyszer csak egy üres táblaelemet találunk, ami azt jelenti, hogy a keresett elem nincs a táblában.

Újrahashelés – törlés

- Mi történik, ha a „hagyományos” módszerrel törlünk?
- Keressük a 458-as indexű elemet, de előtte töröltük a 785-öset.
- A kereső algoritmus megáll a 785-ös helyénél!

Iteráció	Index	Elem	
0	5	128	abc
1	10	854	xyz
2	3	~~785~~	~~mno~~
3	2	544	bcd
4	9	276	fgh
5	13	**458**	**qwe**

Újrahashelés – törlés

- A megoldás:
  törléskor egy „törölt” bejegyzést helyezünk el a táblában.
- Ez megváltoztatja a beszúrást is: nem csak üres helyre, hanem „törölt” jelzésűre is tehetünk új elemet.
- A kereső algoritmus tudni fogja, hogy a „törölt” bejegyzésen nem kell megállnia.

Iteráció	Index	Elem	
0	5	128	abc
1	10	854	xyz
2	3	[törölt elem]	
3	2	544	bcd
4	9	276	fgh
5	13	**458**	**qwe**

Újrahashelés – hashfüggvény

- Újrahashelésnél a hashfüggvény tulajdonságainak a 2. paraméter növelésekor is teljesülniük kell!
- Azaz a (hash(k,0),hash(k,1),...) sorozat is:
  - pszeudorandom
  - korlátos intervallumon vesz fel értékeket
    - ugyanazon, amin (hash(a,0),hash(b,0),...)
  - az intervallumon belül minden értéket felvesz
  - minden értéket egyforma valószínűséggel vesz fel
- Ez azért fontos, hogy az újrapróbálkozásoknál legvégső esetben minden mezőt kipróbáljon.

Újrahashelés – hashfüggvény

- Szokásos módszerek újrahashelésre:
  - Lineáris kipróbálás

$$h'(k,i) = \left(h(k) + i\right) \bmod m$$

  - Négyzetes kipróbálás

$$h'(k,i) = \left(h(k) + c_1 \cdot i + c_1 \cdot i^2\right) \bmod m$$

  - Kettős hashelés

$$h'(k,i) = \left(h_1(k) + i \cdot h_2(k)\right) \bmod m$$

- Részletek az előadáson voltak.

Ujjlenyomat-készítés

- Mi van, ha nem csak int típusú kulcsaink vannak?
- Más-más hashfüggvény minden típushoz?
- Hogy ne kelljen minden típusra külön hasító függvényt írni, a hasító függvényt két függvényre bontjuk:
  - Ujjlenyomat-képező függvény
  - Hasító függvény

Ujjlenyomat-készítés

- Minden típusnak saját ujjlenyomat-készítő függvény, ez ugyanolyan ujjlenyomatot állít elő minden típusra
- A hashfüggvény az immár egységes ujjlenyomatból fog indexet készíteni
- Az ujjlenyomat méretét általában érdemes 32 bites egésznek (int) választani
  - 32 bit a legtöbb esetben elég, ezt könnyű kezelni
  - Ha esetleg ez nem elég, akkor 64 bites (long long)
    - Nagyon nagy tárhelyigény (32 bittel 16 GB-os méret még kezelhető, 64 bitnél 32 GB-ról indulunk)

Ujjlenyomat, `sizeof(T) <= sizeof(int)`

- sizeof(T): a T típus ábrázolásához szükséges memóriaterület
- Egész típusok (char, short, int, enum, …)
  – (érték szerinti) típuskonverzió
    - **`x`** → **`int(x)`**
    - ‘k’ (char) → 107 (int)
    - 1337 (short) → 1337 (int)
- Egyéb típusok – bitminta újraértelmezése
    - float val_f = 3.14 (float) → **`4048F5C3`** → 1078523331 (int)
    - Általánosan char tömbbé érdemes: char *val_c = (char*)(&val_f);

Ujjlenyomat, `sizeof(T) > sizeof(int)`

- String
  - Egyszerű megoldások
    - Karakterek összege mod 2^{32}
    - Utolsó 4 karakter intként értelmezve

Ujjlenyomat, `sizeof(T) > sizeof(int)`

- Vegyük észre, hogy a stringeket értelmezhetjük úgy, mint egy nagy egész számot.
  - Pl. egy 12 bájt hosszú string tekinthető egy 96 bites egész számnak.
- Másik értelmezés: egy string tekinthető egy 256-os számrendszerben felírt számnak.

$$'abc' = 'a' \cdot 256^2 + 'b' \cdot 256 + 'c'$$

$$= 97 \cdot 256^2 + 98 \cdot 256 + 99 = 6382179$$

Ujjlenyomat, `sizeof(T) > sizeof(int)`

- Általánosítás: egy string tekinthető egy *q*-s számrendszerben felírt számnak.

$$'abc' = 'a' \cdot q^2 + 'b' \cdot q + 'c'$$

- Hasító függvény: tekintsük a stringet számnak, és határozzuk meg a *p* prímszám szerinti maradékát.
- Polinom forma: hatékonyan kiértékelhető Horner sémával. Pszeudokód:

```
h = 0
a string minden c karakterére
  h = ( q * h + c ) modulo p
```

Ujjlenyomat, `sizeof(T) > sizeof(int)`

- Paraméterválasztás: a pszeudokód akkor működik jól, ha kiértékelés során nem történik túlcsordulás.
- Ezért a p és q értékét okosan kell megválasztani.
- String esetén q = 256 = 2^8
- Ahhoz, hogy q-val szorozva és c-t hozzáadva ne legyen túlcsordulás, h max 2^{24}-1 lehet
- Ennél kisebb prímszám a 2^{24}-3=16 777 213

```
h = 0
a string minden c karakterére
  h = ( q * h + c ) modulo p
```

Egy kicsit bonyolultabb ujjlenyomat

- SHA-256
  - 64 karakter (32 byte, 256 bit)
  - Pl. „PPKE-ITK” = d5c00aaf7aa1a5cc0a60121a680e4ae59b70035b950bafb245c0e052a7dfe40d
  - Mennyire sok ez?
    - Jelenleg nem ismert két azonos hasht elérő bemenet
    - De ez nem jó nekünk. Miért?
    - Kicsit lassabb számolni – itt nem a teljesítmény a lényeg, hanem hogy nehéz legyen kulcsütközést elérni
    - Ha ekkora hashtáblát akarunk előállítani, akkor 2^256 sorra lenne szükségünk
    - Ez mennyi?
      - Atomok száma a világegyetemben: ~2^80
    - De vehetjük mondjuk az utolsó 8 karakterét számként (32 bit), már ekkor is nagyon ritka lesz az ütközés
    - Kriptográfiára használt hasheknél a lényeg, hogy az inverzió költsége nagyon magas legyen, és hogy nagyon nehéz legyen két azonos hasht generáló üzenetet készíteni

Hasító függvények

- Most konkrétan azokra a függvényekre gondolunk, amelyek az ujjlenyomatot képezik le a tábla méretének megfelelő intervallumra.
- Osztó módszer: **modulo** *m*
  - Jól működik, ha m prímszám.
  - Tehát a módszer használható, ha garantálni tudjuk, hogy a tábla mérete prímszám.