# ADATSZERKEZETEK ÉS ALGORITMUSOK

Általános hasítás

# Hasító táblázatok

- Potenciális $\mathcal{O}(1)$ keresési idő
  - Ha egy megfelelő $h(\text{kulcs}) \rightarrow \text{integer}$ 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

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

      - 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!
    $h(k) = k\ mod\ 2^n$ a $k$ utolsó $n$ bitjét választja

**$k$ mod $2^8$ ezeket választja**

`01100101110`$\boxed{\texttt{00011010}}$

    - 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 $[0, m)$-ra

- 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) $\lfloor A \cdot 2w \rfloor$–val ← $2w$ 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 $\frac{1}{m}$
  - Ez 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 $[0 \ldots p-1]$-ben ($p > m$)
  - Jelölés: $Z_p = \{0,1,\ldots,p-1\}$, $Z_p^* = \{1,2,\ldots,p-1\}$
  - Definiáljuk: $\forall a \in Z_p^*, \forall b \in Z_p$,
    - $h_{a,b}(k) = \big((a * k + b) \bmod p\big) \bmod m$
  - 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

# Hash táblák megvalósítása

Következő téma