ELMÉLETI INFORMATIKA

II. rész

Algoritmus- és kiszámíthatóságelmélet

Rendezési algoritmusok, Hash táblák

9. előadás

Kulcsmanipuláción alapú rendezési módszerek

A korábban ismertetett MERGE SORT, QUICK SORT és HEAP SORT algoritmusok bonyolultsága $O(n \log n)$, átlagos esetben ennél jobb bonyolultságú összehasonlítás alapú rendezési algoritmust nem ismerünk.

Ismerhetjük azonban az U univerzum elemeinek belső szerkezetét, vagy azt, hogy az U halmaz elemszáma a rendezendő $A[1..n]$ tömb méretéhez képest mekkora. Ezekre az ismeretekre alapozva, az eddigieknél hatékonyabb rendezési módszerek is megadhatók.

Counting Sort (Leszámláló rendezés)

A rendezés alapelve: Az $A[1..n]$ tömb rendezésekor kihasználjuk, hogy annak elemei egy olyan alaphalmazból kerülnek ki, melynek k elemszáma az A tömb n méretéhez képest nem túl nagy.

Minden x tömbelemre meghatározzuk azoknak a tömbelemeknek a számát, melyek kisebbek mint x. Így a kimeneti tömbben az x elemet egyből a megfelelő helyre tudjuk mozgatni.

Szükség lesz két segédtömbre: $B[1..n]$ és $C[1..k]$

A B tömbben adjuk majd meg az A tömb elemeit sorba rendezve.

A C tömb elemeit először számlálóként használjuk, a $C[j]$ értékét az adja meg, hogy a j elem hányszor fordul elő az A tömbben. Ezután minden $C[j]$ tömbbeli elemre meghatározzuk azoknak az A tömbbeli elemeknek a számát, melyek a j elemtől kisebbek vagy vele egyenlők.

Első lépésként a $C[1..k]$ tömb minden elemét 0-ra állítjuk.

Ezután végigmegyünk az A tömbön, s ha $A[i] = s$, akkor a $C[s]$ értékét 1-gyel megnöveljük. Így a $C[s]$ elem azt fogja megadni, hogy az s elem, hányszor fordul elő az A tömbben.

A következő lépésben a $2.$ elemtől kezdve végigmegyünk a C tömbön, és minden $C[i]$ elemhez hozzárendeljük a $C[i] + C[i-1]$ értéket. Ekkor a $C[i]$ elem azoknak az A tömbbeli elemeknek a számát fogja tartalmazni, amelyek kisebb vagy egyenlők mint i.

Ha ezzel megvagyunk, akkor az A tömb elemeit már könnyen be tudjuk illeszteni a megfelelő helyre a B tömbben: az $A[i]$ elem helyét a $C\left[A[i]\right]$ mezőben tárolt érték fogja megadni.

Mivel az A tömbben lehetnek egyforma elemek is, az $A[i]$ elem B tömbbe való helyezése után a $C\left[A[i]\right]$ elem értékét 1-gyel csökkentjük. Így bebiztosítjuk, hogy a következő, az $A[i]$ elemmel megegyező értékű A tömbbeli elem a B tömbben a már korábban behelyezett $A[i]$ elem elé fog kerülni.

9.1 példa: Rendezzük **Counting Sort** algoritmussal az alábbi tömböt, melynek elemei az $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ alaphalmazból valók!

A

5	3	1	4	8	9	6

C

0	1	1	2	3	4	5	5	6

B

1	3	4	5	6	8	9

Counting Sort (pszeudokód)

Input: $A[1..n]$ tömb

Output: $B[1..n]$ nemcsökkenő sorrendben rendezett tömb

COUNTING_SORT ($A[1..n]$)

for $i = 1$ **to** k **do** $C[1] \longleftarrow 0$

for $j = 1$ **to** n **do** $C\big[A[j]\big] \longleftarrow C\big[A[j]\big] + 1$

// $C[i]$ most azoknak az A tömbbeli elemeknek a számát tartalmazza, melyek értéke $= i$

for $i = 2$ **to** k **do** $C[i] \longleftarrow C[i] + C[i-1]$

// $C[i]$ most azoknak az A tömbbeli elemeknek a számát tartalmazza, melyek értéke $\leq i$

for $j = n$ **downto** 1 **do** $B\Big[C\big[A[j]\big]\Big] \longleftarrow A[j]$

$\quad C\big[A[j]\big] \longleftarrow C\big[A[j]\big] - 1$

return $B[1..n]$

A Counting Sort bonyolultsága

A $C[1, \ldots, k]$ tömb elemeinek nullára állításakor egyszer végig kell menni ezen a tömbön, ennek a lépésnek a bonyolultsága $O(k)$.

Ezután az algoritmus végigmegy az $A[1, \ldots, k]$ tömbön, ennek a lépésnek a bonyolultsága $O(n)$.

Majd az algoritmus ismét végigmegy a $C[1, \ldots, k]$ tömbön, ennek a lépésnek a bonyolultsága $O(k)$.

Utolsó lépésben az algoritmus végigmegy az $A[1, \ldots, k]$ tömbön, ennek a lépésnek a bonyolultsága $O(n)$.

Az algoritmus bonyolultsága tehát $O(n + k)$. A módszer akkor hatékony, ha a C tömb mérete az A tömb méretéhez képest nem túl nagy, ezért feltételezhetjük, hogy $k = O(n)$ teljesül.

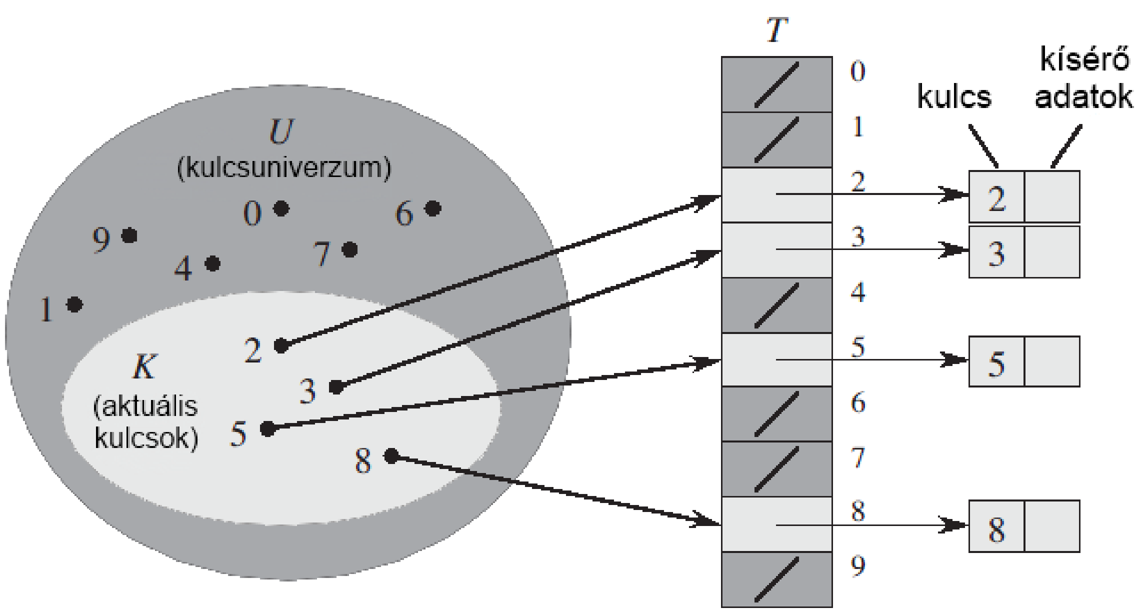
Mindezek alapján a COUNTING SORT rendezés bonyolultsága $O(n)$, s ez sokkal kedvezőbb, mint amit az összehasonlítás alapú rendezési módszerekkel el tudunk érni.

Közvetlen címzésű táblák

A **közvetlen címzés** viszonylag kis méretű kulcsuniverzumokra hatékonyan működik. Tételezzük fel, hogy egy alkalmazáshoz olyan dinamikus halmazra van szükség, amelyben az elemek kulcsai az $U = \{0, 1, 2, \ldots, m-1\}$ univerzumból valók, ahol az m értéke nem túl nagy. Tételezzük még fel, hogy nincs két egyforma kulcs (azaz minden kulcs egyedi).

A dinamikus halmaz megvalósítására egy $T[0..m-1]$ tömböt használunk, melynek minden helye – **rés** – megfelel az U egy kulcsának.

Ez az ún. **közvetlen címzésű** tábla.



Az ún. szótár műveletek (**kulcs beszúrása**, **kulcs szerinti keresés**, **kulcs törlése**) egyszerűen megvalósíthatók.

Direct-Address-Insert(T, $kulcs$, $érték$): $T[kulcs] \longleftarrow érték$

Direct-Address-Search(T, $kulcs$): **return** $T[kulcs]$

Direct-Address-Delete(T, $kulcs$): $T[kulcs] \longleftarrow \mathrm{NIL}$

Mindhárom művelet lépésszáma $O(1)$.

Megjegyzés: A közvetlen címzés általában nem valósítható meg.

- Az U kulcsuniverzum nagy mérete kezelhetetlen méretű tárkapacitást igényelne.

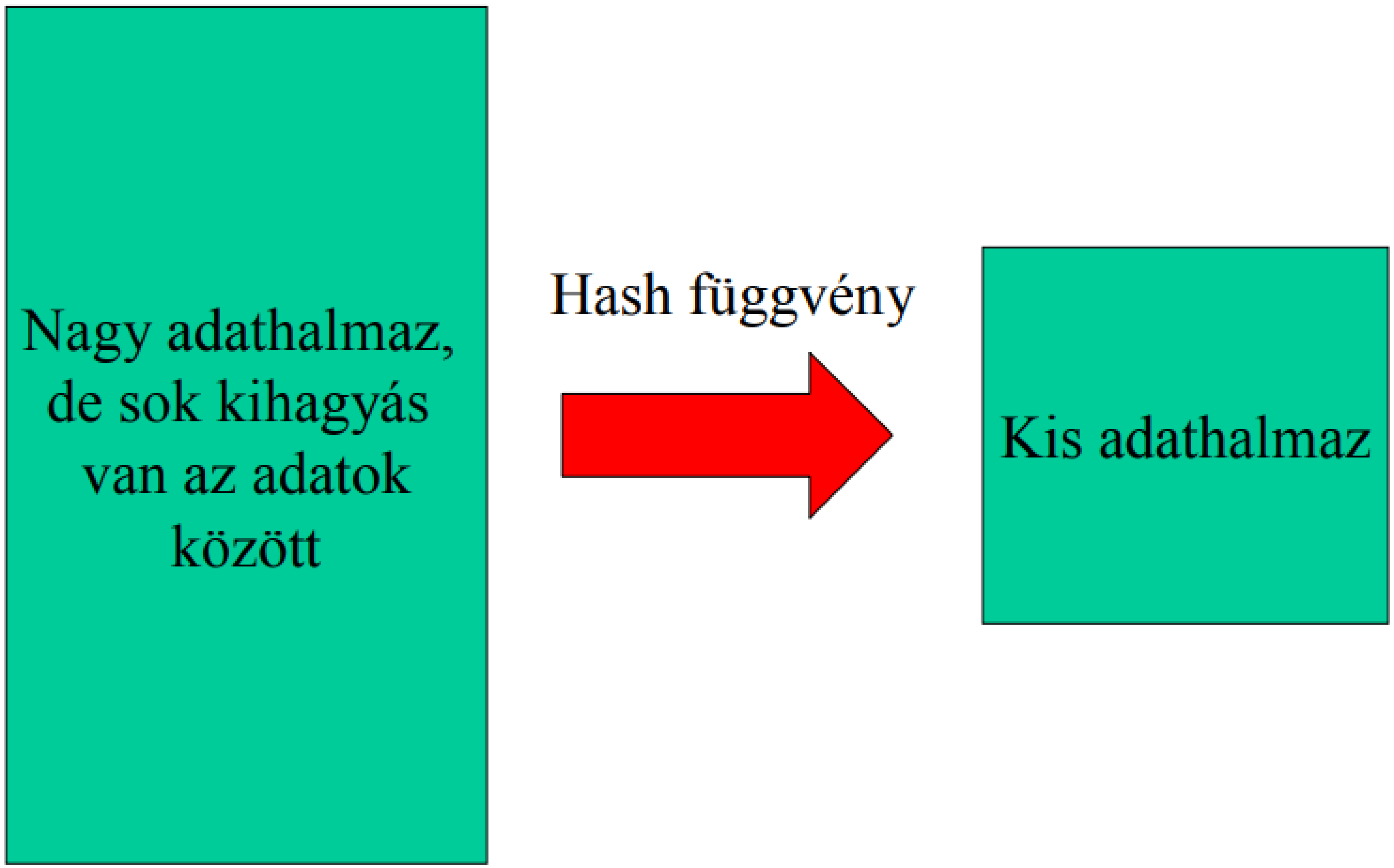
  Pl. ha a kulcs a 10-jegyű születési szám: _ _ _ _ _ _ _ _ _ _

  Durva becslés a kulcsuniverzum méretére:

  $$10 \times 10 \times 4 \times 10 \times 4 \times 10 \times 10 \times 10 \times 10 \times 10 \approx 1{,}6\ \mathrm{MLD}$$

- Az aktuálisan tárolt elemek száma (kb. $5{,}5$ millió) a kulcsuniverzum méretéhez képest kicsi lenne: $|K| \ll |U|$

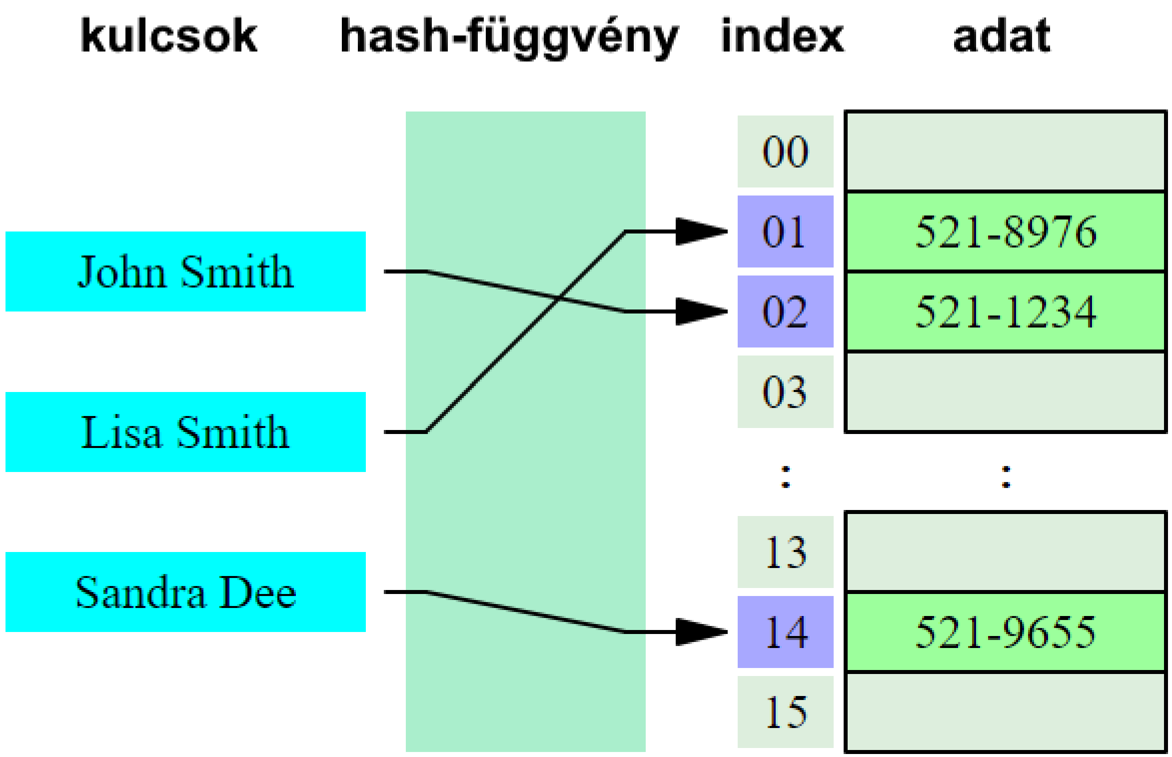
Hash táblák


Nagy adathalmaz, de sok kihagyás van az adatok között
Hash függvény
Kis adathalmaz

Hash táblák

A **hash tábla** olyan adatszerkezet, amely egy *hash függvény* segítségével állapítja meg, hogy melyik kulcshoz milyen érték tartozik – így implementál egy asszociatív tömböt.

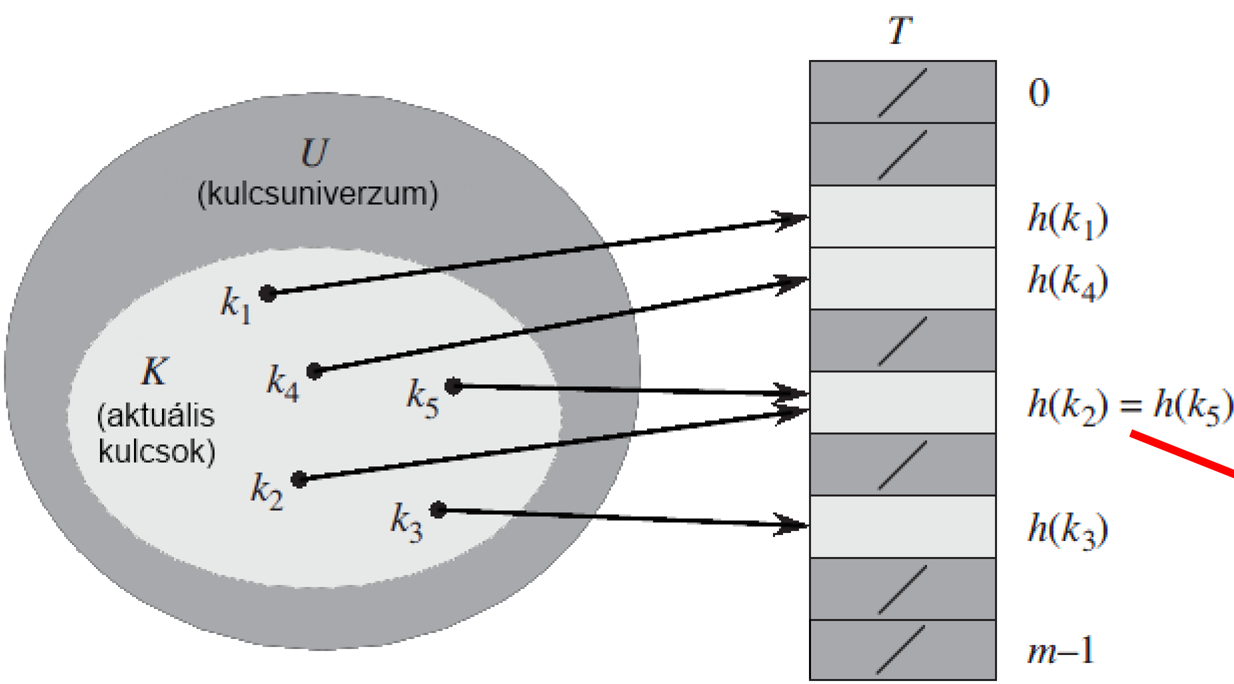
A hash függvény segítségével tehát a kulcsot leképezzük az adatokat tároló tömb egy indexére, ahol a keresett adat megtalálható.



Közvetlen címzés: a k kulcs a tábla $T[k]$ résében tárolódik.

Hash technika: a k kulcs a tábla $T[h(k)]$ résében tárolódik.

A hash függvény tehát a kulcsok U univerzumát képezi le a $T[0..m-1]$ hash tábla réseire, azaz $h: U \longrightarrow \{0, 1, \dots, m-1\}$. Használatával jelentősen csökkenteni tudjuk a szükséges tömbindexek tartományát.



Mivel $|U| > m$ ezért létezik legalább két kulcs, melyek ugyanarra a résre képződnek le. Ezt nevezzük **kulcsütközés**nek.

a k_2 és k_5 kulcsok ütköznek

A **kulcsütközés** (nem hiba!)

- előfordulásának esélye minimalizálható a hash függvény megfelelő megválasztásával,
- feloldható ütközésfeloldó technikák alkalmazásával (pl. *láncolt lista*, *nyílt címzés*).

Hash függvények

A megfelelő hash függvény megtalálása nem egyszerű feladat. *A választott hash függvénynek mindig az adott feladatnak megfelelőnek kell lennie*, általános megoldásunk nincs.

Azonban megadható néhány általános szabály:

- nagyjából elégítse ki az **egyszerű egyenletesség** feltételét, vagyis minden kulcs egyforma valószínűséggel képződjön le a T hash tábla m darab résének bármelyikébe,
- a függvényértéket úgy állítsa elő, hogy az ne függjön a kulcsokban meglévő szabályszerűségektől (pl. a kulcs számjegyeitől),
- a hasonló kulcsokat **véletlenszerűen** képezze le a résekbe,
- **kevés** ütközést idézzen elő,
- legyen **gyorsan** kiszámítható.

A legtöbb hash függvény feltételezi, hogy a kulcsuniverzum az $\mathbb{N}$ halmaz.

Amennyiben a kulcsok nem természetes számok, akkor keresni kell valamilyen módot arra, hogy természetes számként legyenek megjelenítve.

A **karakterlánc**ként megadott kulcsokat tekinthetjük valamilyen számrendszerben felírt pozitív egész számoknak. Ekkor pl. a `pt` karakterlánc felírható $(112, 116)$ számpárként, ahol a 112 a `p`, a 116 pedig a `t` karakter ASCII kódja. A $(112, 116)$ számpár a 128-as számrendszerben a $112.128 + 116 = 14452$ számmal azonosítható.

A továbbiakban feltételezzük, hogy a kulcsok természetes számok. Három hash függvényt előállító módszert mutatunk be: az **osztásos**, a **szorzásos** és az **univerzális módszert**.

1) Osztásos módszer

Ez a módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy vesz a k szám m-mel való osztásának maradékát. A hash függvény ebben az esetben tehát a következő:

$$h(k) = k \mod m$$

A módszer nagyon gyors, mivel csak egyetlen osztást igényel.

Az osztásos módszer alkalmazásakor ajánlott elkerülni bizonyos m értékeket:

- Az m ne legyen a 2 hatványa, mert ha $m = 2^p$, akkor a $h(k)$ függvényérték éppen a k kulcs p darab legalacsonyabb helyiértékű bitje (LSB – Least Significant Bit).

  Pl. ha $m = 2^5$, $k = (567)_{10} = (10\,0011\,0111)_2$,

  akkor $h(k) = 567 \bmod 32 = (23)_{10} = (1\,0111)_2$

1) Osztásos módszer

Ez a módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy vesz a k szám m-mel való osztásának maradékát. A hash függvény ebben az esetben tehát a következő:

$$h(k) = k \mod m$$

A módszer nagyon gyors, mivel csak egyetlen osztást igényel.

Az osztásos módszer alkalmazásakor ajánlott elkerülni bizonyos m értékeket:

- Ha a kulcsok 10-es számrendszerben megadott számok, akkor az m ne legyen a 10 hatványa, mert ekkor a $h(k)$ függvényérték függne a k kulcs számjegyeitől.

  Pl. ha $m = 10^2$, $k = 567$, akkor $h(k) = 567 \mod 100 = 67$

1) Osztásos módszer

Ez a módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy vesz a k szám m-mel való osztásának maradékát. A hash függvény ebben az esetben tehát a következő:

$$h(k) = k \mod m$$

A módszer nagyon gyors, mivel csak egyetlen osztást igényel.

Az osztásos módszer alkalmazásakor ajánlott elkerülni bizonyos m értékeket:

- Ha a kulcsok 2^p alapú számrendszerben megjelenített karakterláncok, akkor az m ne legyen $2^p - 1$, mert ekkor azok a kulcsok, amelyek egymástól csak két szomszédos karakter sorrendjében különböznek, ugyanarra a résre fognak leképződni.

1) Osztásos módszer

Ez a módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy vesz a k szám m-mel való osztásának maradékát. A hash függvény ebben az esetben tehát a következő:

$$h(k) = k \mod m$$

A módszer nagyon gyors, mivel csak egyetlen osztást igényel.

Akkor mi legyen a hash tábla mérete? Az m értékének megválasztására van egy széles körben elfogadott recept, D. E. KNUTH javaslata:

legyen m olyan prímszám, amely nem osztója az $r^a \pm b$ számnak, ahol r a karakterkészlet elemszáma (pl. 128, vagy 256), a és b pedig „kicsi” egész számok.

9.2 példa: Legyen a hash függvény $h(k) = k \mod 13$. Hány rést tartalmaz a T hash tábla és mely résekre képződnek le a 345 és az 1547 kulcsok?

A hash függvény felépítéséből láthatjuk, hogy a T hash tábla mérete $m = 13$.

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	1547	NIL	NIL	NIL	NIL	NIL	NIL	345	NIL	NIL	NIL	NIL	NIL

A 345 kulcs beszúrásának helye: $h(345) = 345 \mod 13 = 7$

Az 1547 kulcs beszúrásának helye: $h(1547) = 1547 \mod 13 = 0$

2) Szorzásos módszer

Ez a módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy vesz a k kulcsot beszorozza egy 0 és 1 közé eső β konstanssal, majd veszi a szorzat törtrészét. Ezután a kapott értéket beszorozza m-mel, és meghatározza az eredmény alsó egészrészét. A hash függvény ebben az esetben tehát a következő:

$$h(k) = \lfloor m \{\beta k\} \rfloor$$

ahol $0 < \beta < 1$, $\{x\}$ pedig az $x \in \mathbb{R}$ szám törtrészét jelöli.

Szemléletesen elmondva: az $\{\beta k\}$ érték kiszámításával a kulcsot „véletlenszerűen” belőjük a $[0,1)$ intervallumba, majd az eredményt felskálázzuk a hash tábla címtartományába.

Egy jól számíthatóan működő választás: legyen a hash tábla mérete $m = 2^t$, legyen $w = 2^{32}$, és legyen N egy w-vel relatív prím egész szám. Ekkor a $\beta = \frac{N}{w}$ választás mellett a $h(k)$ függvényérték igen jól számítható.

Bár a módszer a β konstans valamennyi megengedett értékére működik, bizonyos értékekre jobban, mint másokra. D. E. KNUTH javaslata a β értékének megválasztására:

$$\beta = \phi^{-1} = \frac{\sqrt{5} - 1}{2} = 0{,}61803988\ldots$$

ahol ϕ az aranymetszés arányszáma.

A szorzásos módszernél tehát érdemes az N értékét úgy megválasztani, hogy az $\frac{N}{w}$ hányados értéke közel legyen ϕ^{-1}-hez.

9.3 példa: Legyen a hash függvény $h(k) = \lfloor 13\,\{0{,}6180\,k\} \rfloor$. Hány rést tartalmaz a T hash tábla és mely résekre képződnek le a 345 és az 1547 kulcsok?

A hash függvény felépítéséből láthatjuk, hogy a T hash tábla mérete $m = 13$.

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	1547	NIL	345	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

A 345 kulcs beszúrásának helye:

$$h(345) = \lfloor 13\,\{0{,}6180\,.\,345\} \rfloor = \lfloor 13\,\{213{,}21\} \rfloor = \lfloor 13\,.\,0{,}21 \rfloor = \lfloor 2{,}73 \rfloor = 2$$

Az 1547 kulcs beszúrásának helye:

$$h(1547) = \lfloor 13\,\{0{,}6180\,.\,1547\} \rfloor = \lfloor 13\,\{956{,}046\} \rfloor = \lfloor 13\,.\,0{,}046 \rfloor =$$
$$= \lfloor 0{,}598 \rfloor = 0$$

3) Univerzális módszer

Az előzőleg megismert hash függvény előállító módszerek esetén, ha a paraméter előre rögzített, akkor könnyen találhatók olyan kulcsok, amelyek a hash tábla ugyanazon résére képződnek le.

Hatásos módszer, ha a hash függvényt a kulcsoktól független módon, véletlenül választjuk ki.

Itt tehát nem egy hash függvénnyel dolgozunk, hanem több hash függvényt tartalmazó **függvényosztály**al. A módszer a k kulcsot úgy képezi le a $T[0..m-1]$ hash tábla valamelyik résébe, hogy véletlenszerűen választ egy hash függvényt a függvényosztályból, majd meghatározza a $h(k)$ függvényértéket.

Így az algoritmus különböző lefutásai során ugyanaz a k kulcs a hash tábla más-más résére képződhet le.

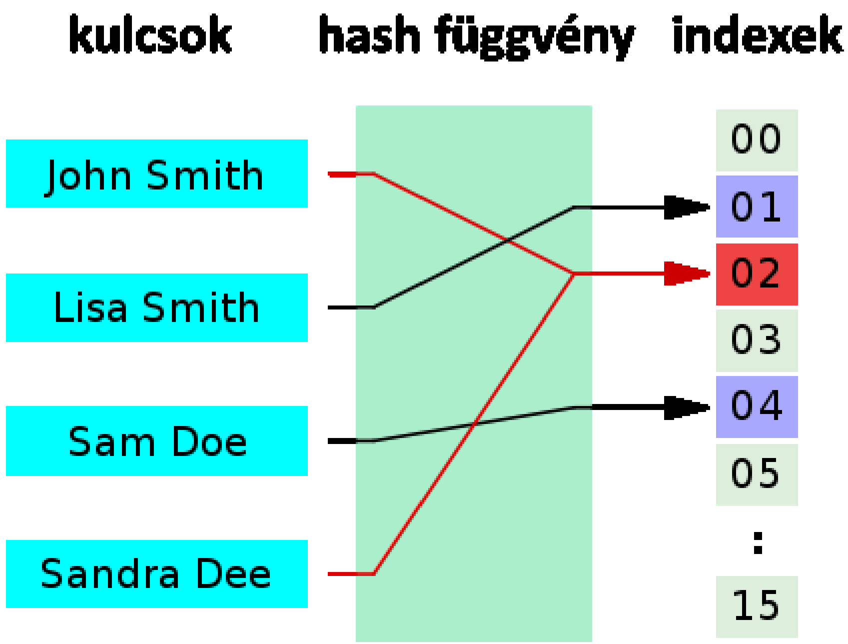
Egy $\mathcal{H}$ hash függvényosztály **univerzális**, ha tetszőleges $x \neq y$ kulcsokra azoknak a $h \in \mathcal{H}$ hash függvényeknek a száma, melyekre $h(x) = h(y)$, pontosan $|\mathcal{H}|/m$.

Mindez azt jelenti, hogy egy véletlenszerűen választott $h \in \mathcal{H}$ hash függvény esetén az x és y $(x \neq y)$ kulcsok ütközésének valószínűsége $1/m$, ami egyenlő a $\{0, 1, \dots, m-1\}$ halmazból véletlenszerűen választott $h(x)$ és $h(y)$ értékek egyenlőségének valószínűségével.

Megjegyzés: Gyakorlatban jól használatható, ha a hash függvény paramétereit választjuk meg véletlenszerűen (pl. a súlyozási paramétereket tartalmazó függvények esetén a különböző súly értékeket).

Kulcsütközések kezelése

Kulcsütközésről beszélünk, ha a kulcstranszformációt megvalósító hash függvény két különböző kulcsú elemet a hash tábla ugyanazon indexű résére képez le.



Bár a hash függvényt próbáljuk úgy megadni, hogy minél kevesebb kulcsütközést idézzen elő, azonban nyilvánvaló, hogy ha a kulcsuniverzum elemszáma nagyobb, mint a hash tábla réseinek száma, akkor ütközés előbb-utóbb be fog következni.

Ha a kulcsütközés bekövetkezik, akkor azt kezelni kell!

A kulcsütközés feloldására különböző technikák használatosak:

- **túlcsordulási területtel**,
- **láncolt listával**,
- **nyílt címzéssel**.

1) Ütközésfeloldás túlcsordulási területtel

A hash tábla adatait tartalmazó tömb mellett lefoglalunk egy másik memóriaterületet is. A kulcsütközés miatt a hash táblába nem férő elemeket ezen a **túlcsordulási terület**en tároljuk.

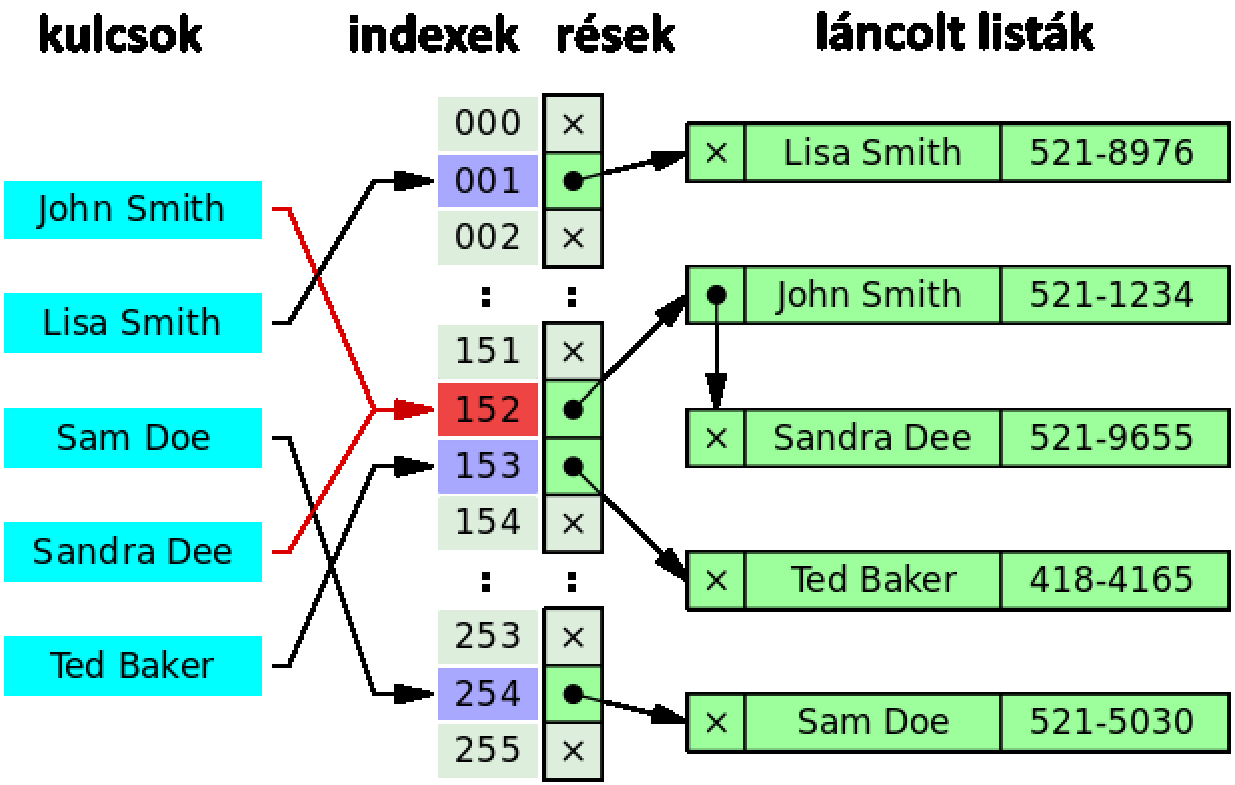
A k kulcs **beszúrás**ának művelete: Ha a T tömb $h(k)$ rése szabad, akkor ide helyezzük a k kulcsot. Ha a $h(k)$ rés nem szabad, akkor pedig a túlcsordulási területre.

A k kulcs **keresés**ének művelete: Megnézzük, hogy a T tömbben a $h(k)$ helyen van-e a k kulcs. Ha nincs ott, akkor megvizsgáljuk a túlcsordulási területet. Ha egyik helyen sincs, akkor hibát jelzünk: "*Nincs ilyen kulcs*".

A k kulcs **törlés**ének művelete: Ha a k kulcs a T tömbben van, akkor töröljük. Ha nincs ott, akkor ellenőrizzük a túlcsordulási területet is. Ha ott van, akkor töröljük onnan. Ha egyik helyen sincs, akkor hibát jelzünk: "*Nincs ilyen kulcs*".

2) Ütközésfeloldás láncolt listával

Az egymással ütköző kulcsokat összefogjuk egy **láncolt listá**ba. A hash tábla i-edik rése egy mutatót tartalmaz, amely az i címre leképződő kulcsok listájának fejére mutat. Amennyiben a lista üres, akkor az i-edik rés tartalma NIL. A listák mérete általában kicsi.

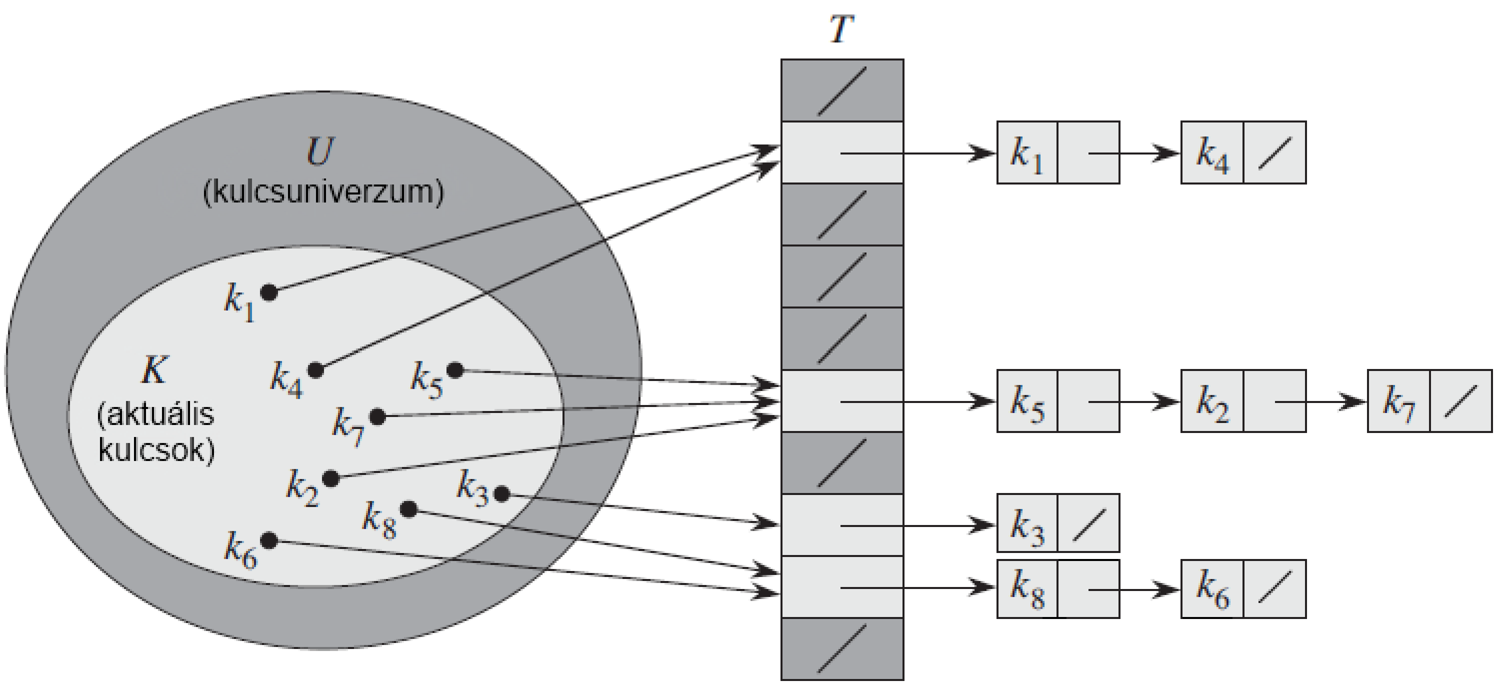


Célszerű lehet a rendezni a láncolt listákat!

Ez a módszer főleg külső táron tárolt nagyméretű állományok kezelésére használatos.

2) Ütközésfeloldás láncolt listával

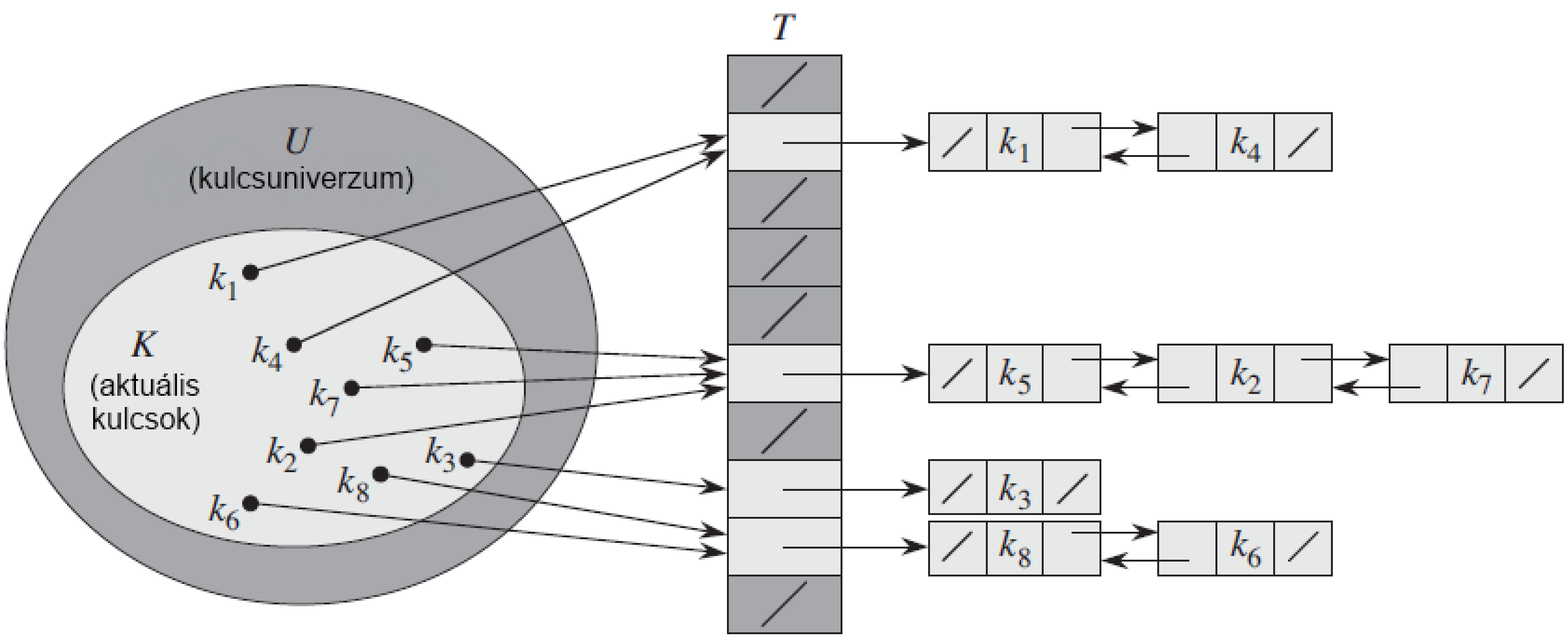
Az egymással ütköző kulcsokat összefogjuk egy **láncolt listá**ba. A hash tábla i-edik rése egy mutatót tartalmaz, amely az i címre leképződő kulcsok listájának fejére mutat. Amennyiben a lista üres, akkor az i-edik rés tartalma NIL. A listák mérete általában kicsi.



Selye János Egyetem
GIK, Informatika Tanszék

2) Ütközésfeloldás láncolt listával

Az egymással ütköző kulcsokat összefogjuk egy **láncolt listá**ba. A hash tábla i-edik rése egy mutatót tartalmaz, amely az i címre leképződő kulcsok listájának fejére mutat. Amennyiben a lista üres, akkor az i-edik rés tartalma NIL. A listák mérete általában kicsi.



- A k kulcs **beszúrás**ának művelete:

A hash tábla $T[h(k)]$ rése által mutatott listát az elején új listaelemmel bővítjük. Ha szükséges, akkor a listát rendezzük.

A **beszúrás** bonyolultsága $O(1)$.

- A k kulcs **keresés**ének művelete:

Kiszámítjuk a $h(k)$ függvényértéket. A hash tábla $T[h(k)]$ rése által mutatott listában keresünk. Ha nincs ott, akkor hibát jelzünk: "*Nincs ilyen kulcs*".

A **keresés** bonyolultsága <u>átlagos esetben</u> $\Theta(1+\alpha)$, ahol α az n elemet tartalmazó $T[0..m-1]$ hash tábla **kitöltöttségi aránya**, $\alpha = n/m$ (megadja az egy listába fűzött elemek átlagos számát).

A **keresés** bonyolultsága <u>legkedvezőtlenebb esetben</u> $O(n)$, ekkor az n darab elem mindegyike ugyanarra a résre képződik le, s egy n hosszúságú listát alkot.

- A k kulcs **törlés**ének művelete:

Kiszámítjuk a $h(k)$ függvény-értéket. A hash tábla $T[h(k)]$ rése által mutatott listában keresünk. Ha ott van, akkor töröljük onnan. Ha nincs ott, akkor hibát jelzünk: "*Nincs ilyen kulcs*".

A **törlés** bonyolultsága $O(1)$, amennyiben a listák láncolása két-irányú.

Ha a listák láncolása egyirányú, akkor a hash tábla $T[h(k)]$ rése által mutatott listában először meg kell keresni a k kulcsot megelőző elemet, hogy annak a mutatóját a k kulcs törlése utáni helyzetnek megfelelően a rákövetkező elemre tudjuk állítani. Ekkor a végrehajtási idő átlagos esetben $\Theta(1+\alpha)$, legkedvezőtlenebb esetben pedig $O(n)$.

3) Ütközésfeloldás nyílt címzéssel

Ennél a módszernél a kulcsokat a $T[0..m-1]$ hash táblában tároljuk. A tábla réseinek tartalma tehát vagy egy kulcs, vagy pedig NIL. A hash tábla természetesen egy idő után betelhet, s ilyenkor további elemek nem szúrhatók bele.

Nyílt címzés alkalmazásakor a hash táblában egy előre definiált szisztematikus üres hely keresést végzünk. Tehát ha egy k kulcs beszúrásakor hash tábla $h(k)$ rése nem szabad, akkor valamilyen stratégia szerint egy másik helyet keresünk neki (ugyanezt követjük keresés és törlés során). Ezek az ún. **kipróbálási stratégiák**.

Ehhez bevezetünk egy kétparaméteres hash függvényt, melynek első paramétere a *kulcs*, második pedig a *próbálkozás száma*:

$$h\colon\ U \times \{0, 1, \dots, m-1\} \longrightarrow \{0, 1, \dots, m-1\}$$

Megköveteljük, hogy minden k kulcsra a $h(k,0)$, $h(k,1)$, $h(k,2)$, …, $h(k,m-1)$ **kipróbálási sorozat** a $0,1,\dots,m-1$ indexek egy permutációja legyen. Ebből következik, hogy a kulcsok hash táblába történő beszúrásakor előbb-utóbb minden rés számításba jön mint lehetséges beszúrási hely.

A hash tábla réseinek esetében az **üres** (NIL) állapot mellett célszerű bevezetni egy **törölt** (DEL, DELETED) állapotot is!

- A k kulcs **beszúrás**ának művelete:

Első lépésben a hash tábla $h(k,0)$ indexű rését nézzük meg, hogy szabad-e (NIL vagy DEL). Ha nem, akkor a próbálkozás számát növeljük, s a $h(k,1)$ indexű rést vizsgáljuk meg, stb. Ha találunk szabad rést, akkor a k kulcsot beillesztjük. Ha nem találunk m próbálkozás után sem, akkor hibát jelzünk: "*Nincs szabad hely*".

Megköveteljük, hogy minden k kulcsra a $h(k,0)$, $h(k,1)$, $h(k,2)$, …, $h(k,m-1)$ **kipróbálási sorozat** a $0,1,\ldots,m-1$ indexek egy permutációja legyen. Ebből következik, hogy a kulcsok hash táblába történő beszúrásakor előbb-utóbb minden rés számításba jön mint lehetséges beszúrási hely.

A hash tábla réseinek esetében az **üres** (NIL) állapot mellett célszerű bevezetni egy **törölt** $(\mathrm{DEL}$, DELETED) állapotot is!

- A k kulcs **keresés**ének művelete:

Kereséskor a ugyanolyan sorrendben nézzük végig a hash tábla réseit, amilyen sorrendben azt a k kulcs beszúrásakor tettük:

- ha üres (NIL) résre lépünk, akkor hibát jelzünk: "*Nincs ilyen kulcs*",
- ha törölt (DEL) résre lépünk, akkor keresünk tovább, hiszen utána még lehet a k kulcs.

Megköveteljük, hogy minden k kulcsra a $h(k,0)$, $h(k,1)$, $h(k,2)$, …, $h(k,m-1)$ **kipróbálási sorozat** a $0,1,\dots,m-1$ indexek egy permutációja legyen. Ebből következik, hogy a kulcsok hash táblába történő beszúrásakor előbb-utóbb minden rés számításba jön mint lehetséges beszúrási hely.

A hash tábla réseinek esetében az **üres** (NIL) állapot mellett célszerű bevezetni egy **törölt** (DEL, DELETED) állapotot is!

- A k kulcs **törlés**ének művelete:

Az első lépés ugyanaz, mint a keresésnél, hiszen előbb meg kell találni a törölni kívánt kulcsot:

- ha a keresés sikeres, akkor a k kulcsot töröljük, a rés tartalmát DEL–re állítjuk (ha NIL-re állítanánk, akkor lehetetlen lenne minden olyan kulcs visszakeresése, amelynek beszúrásakor ezt a rést kipróbáltuk és foglaltnak találtuk),
- ha a keresés sikertelen, akkor hibát jelzünk: "*Nincs ilyen kulcs*".

A nyílt címzés módszere csak az operatív memóriában tárolt állományok kezelésére használatos.

Nyílt címzéskor feltételezzük, hogy a hash függvény **egyenletes**, azaz minden kulcsra a $\{0, 1, \ldots, m-1\}$ indexek lehetséges $m!$ darab permutációjának mindegyike azonos valószínűséggel fordul elő, mint kipróbálási sorozat.

A továbbiakban három olyan módszert ismertetünk, amelyek kipróbálási sorozat előállítására használatosak: a **lineáris próbát**, a **négyzetes próbát** és a **dupla hashelést**.

1) Lineáris próba

Legyen adott a $h\colon\ U \longrightarrow \{0, 1, \dots, m-1\}$ hash függvény. Ekkor a **lineáris próba** módszere az alábbi hash függvényt használja:

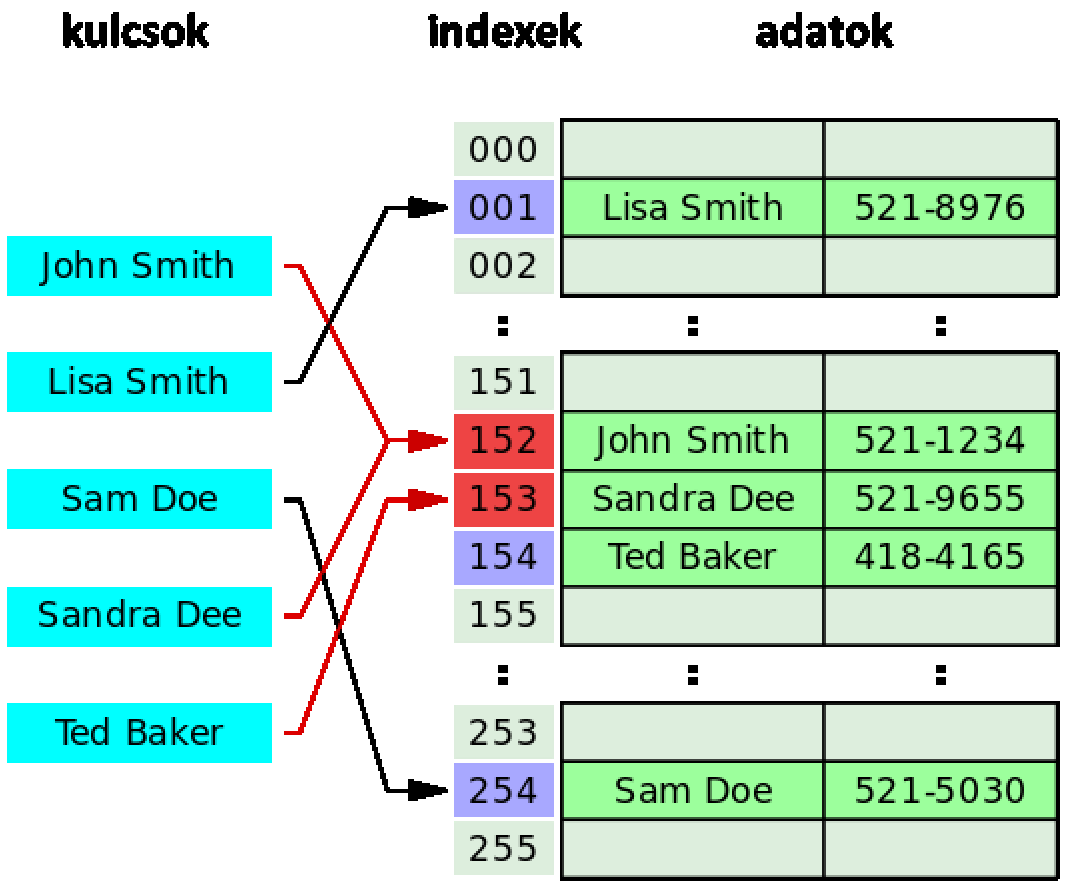
$$h(k, j) = (h(k) + zj) \mod m$$

ahol k a beszúrni kívánt kulcs,
j az aktuális próba száma,
m a T hash tábla réseinek száma,
z tetszőleges egész konstans, legegyszerűbb esetben $z = 1$.

Ha tehát $z = 1$, akkor a k kulcs beszúrásakor az elsőként kipróbált rés a $T[h(k)]$ lesz, majd következnek a $T[h(k)+1]$, $T[h(k)+2]$, …, $T[m-1]$, $T[0]$, $T[1]$, …, $T[h(k)-1]$ rések.

Az összes lehetséges kipróbálási sorozat száma tehát m.

1) Lineáris próba



A lineáris próba hátránya, hogy a hash táblában hosszú, kulcsok által elfoglalt rés-sorozatok, ún. *elsődleges klaszterek* jönnek létre.

9.4 példa: Legyen a hash függvény $h(k) = k \mod 13$. Lineáris próbát alkalmazva ($z = 1$) mely résekre képződnek le a 345, 1555, 8772, 463, 5302, 2217 és a 6209 kulcsok?

A hash függvény felépítéséből láthatjuk, hogy a T tábla mérete $m = 13$.

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	6209	NIL	NIL	NIL	NIL	NIL	NIL	345	1555	463	8772	5302	2217

A 345 kulcs helye: $h(345,0) = \big((345 \mod 13) + 1.0\big) \mod 13 = 7$

Az 1555 kulcs helye: $h(1555,0) = \big((1555 \mod 13) + 1.0\big) \mod 13 = 8$

A 8772 kulcs helye: $h(8772,0) = \big((8772 \mod 13) + 1.0\big) \mod 13 = 10$

A 463 kulcs helye: $h(463,1) = \big((463 \mod 13) + 1.1\big) \mod 13 = 9$

Az 5302 kulcs helye: $h(5302,0) = \big((5302 \mod 13) + 1.0\big) \mod 13 = 11$

A 2217 kulcs helye: $h(2217,5) = \big((2217 \mod 13) + 1.5\big) \mod 13 = 12$

A 6209 kulcs helye: $h(6209,5) = \big((6209 \mod 13) + 1.5\big) \mod 13 = 0$

9.5 példa: Tekintsük az alábbi hash táblát és legyen a hash függvény $h(k) = k \mod 13$. Lineáris próbát alkalmazva $(z = 1)$ töröljük a 463 és 6209 kulcsokat!

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	DEL	NIL	NIL	NIL	NIL	NIL	NIL	345	1555	DEL	8772	5302	2217

A 463 kulcs törlése: $h(463,1) = \big((463 \mod 13) + 1.1\big) \mod 13 = 9$

A 6209 kulcs törlése: $h(6209,5) = \big((6209 \mod 13) + 1.5\big) \mod 13 = 0$

2) Négyzetes próba

Legyen adott a $h\colon\ U \longrightarrow \{0, 1, \dots, m-1\}$ hash függvény. Ekkor a **négyzetes próba** módszere az alábbi hash függvényt használja:

$$h(k, j) = (h(k) + z_1 j + z_2 j^2) \mod m$$

ahol k a beszúrni kívánt kulcs,

j az aktuális próba száma,

m a T hash tábla réseinek száma,

z_1, z_2 tetszőleges egész konstansok ($z_1 \neq 0$, $z_2 \neq 0$).

A k kulcs beszúrásakor az elsőként kipróbált rés a $T[h(k)]$ lesz, majd a következő pozíció az őt megelőző próbaszámtól négyzetesen függő értékkel történő eltolással kapható meg.

Az összes lehetséges kipróbálási sorozat száma m.

Ha k_1, k_2 kulcsokra $h(k_1, 0) = h(k_2, 0)$, akkor mindkét kipróbálási sorozat meg fog egyezni, így *másodlagos klaszterek* jönnek létre.

9.6 példa: Tekintsük az alábbi hash táblát és legyen a hash függvény $h(k) = k \mod 13$. Négyzetes próbát alkalmazva $(z_1 = 1,\ z_2 = 2)$ mely résekre képződnek le a 345 és a 6211 kulcsok?

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	NIL	NIL	12	NIL	345	6211	NIL	4378	98	NIL	777	NIL	NIL

345 kulcs: $h(345, 2) = \left((345 \mod 13) + 1.2 + 2.2^2\right) \mod 13 = 4$

6211 kulcs: $h(6211, 3) = \left((6211 \mod 13) + 1.3 + 2.3^2\right) \mod 13 = 5$

3) Dupla hashelés

Legyenek adottak a $h_1, h_2: U \longrightarrow \{0, 1, \dots, m-1\}$ hash függvények. Ekkor a **dupla hashelés** módszere az alábbi hash függvényt használja:

$$h(k,j) = \left(h_1(k) + jh_2(k)\right) \mod m$$

ahol k a beszúrni kívánt kulcs,
j az aktuális próba száma,
m a T hash tábla réseinek száma.

A k kulcs beszúrásakor az elsőként kipróbált rés a $T[h_1(k)]$ lesz, majd a következő pozíció az őt megelőző pozíció $h_2(k)$ értékkel történő eltolásával kapható meg.

Az összes lehetséges kipróbálási sorozat száma m^2: minden $\left(h_1(k), h_2(k)\right)$ értékpár különböző kipróbálási sorozathoz vezet, mivel a $h_1(k)$ kezdő próbapozíció és a $h_2(k)$ eltolás egymástól függetlenül változik.

9.7 példa: Tekintsük az alábbi hash táblát és legyen a két hash függvény $h_1(k) = k \mod 13$, $h_2(k) = 1 + (k \mod 11)$. Dupla hashelést alkalmazva mely résekre képződnek le a 14 és a 95 kulcsok?

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	NIL	79	NIL	NIL	69	98	NIL	72	NIL	14	NIL	50	NIL

A 14 kulcs beszúrásának helye:

$$h(14,0) = \big(h_1(14) + 0\,h_2(14)\big) \mod 13 = \big((14 \mod 13) + 0\big) \mod 13 = 1$$

a $T[1]$ rés **foglalt**

$$h(14,1) = \big(h_1(14) + 1\,h_2(14)\big) \mod 13 =$$

$$= \Big((14 \mod 13) + 1\,\big(1 + (14 \mod 11)\big)\Big) \mod 13 = 5$$

a $T[5]$ rés **foglalt**

$$h(14,2) = \big(h_1(14) + 2\,h_2(14)\big) \mod 13 =$$

$$= \Big((14 \mod 13) + 2\,\big(1 + (14 \mod 11)\big)\Big) \mod 13 = 9$$

a $T[9]$ rés **szabad**

9.7 példa: Tekintsük az alábbi hash táblát és legyen a két hash függvény $h_1(k) = k \mod 13$, $h_2(k) = 1 + (k \mod 11)$. Dupla hashelést alkalmazva mely résekre képződnek le a 14 és a 95 kulcsok?

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	NIL	79	NIL	NIL	69	98	NIL	72	NIL	14	NIL	50	95

A 95 kulcs beszúrásának helye:

$$h(95,0) = \left(h_1(95) + 0\, h_2(95)\right) \mod 13 = \left((95 \mod 13) + 0\right) \mod 13 = 4$$

a $T[4]$ rés **foglalt**

$$h(95,1) = \left(h_1(95) + 1\, h_2(95)\right) \mod 13 =$$

$$= \left((95 \mod 13) + 1\left(1 + (95 \mod 11)\right)\right) \mod 13 = 12$$

a $T[12]$ rés **szabad**

Megjegyzés:

A $h_2(k)$ eltolási értéknek relatív prímnek kell lennie a T hash tábla m méretéhez képest. Ha ugyanis valamely k kulcsra m-nek és $h_2(k)$-nak volna valamilyen $d > 1$ közös osztója, akkor a kulcs beszúrási helyének keresésekor csak a hash tábla $1/d$ részét tudnánk megvizsgálni.

Ez a feltétel egyébként könnyen bebiztosítható. Pl.

1) legyen $m = 2^t$, a $h_2(k)$ hash függvényt pedig adjuk meg úgy, hogy mindig páratlan számot állítson elő,
2) legyen m prímszám, a $h_2(k)$ hash függvényt pedig adjuk meg úgy, hogy mindig m-nél kisebb pozitív egész számot állítson elő.

Legyen $T[0..m-1]$ nyílt címzéses hash tábla $\alpha = n/m$ kitöltött-ségi aránnyal, h pedig egy egyenletes hash függvény.

Ekkor a **sikertelen keresés** várható próbaszáma legfeljebb $\frac{1}{1-\alpha}$ ami azt jelenti, hogy a keresés $O(1)$ idő alatt lefut. Pl.

- ha a hash tábla félig van kitöltve, akkor a sikertelen keresés várható próbaszáma legfeljebb $\frac{1}{1-0,5} = 2$
- ha a hash tábla 90%-ban van kitöltve, akkor a sikertelen keresés várható próbaszáma legfeljebb $\frac{1}{1-0,9} = 10$

Legyen $T[0..m-1]$ nyílt címzéses hash tábla $\alpha = n/m$ kitöltöttségi aránnyal, h pedig egy egyenletes hash függvény. Tételezzük fel, hogy a tábla minden elemét egyforma valószínűséggel keressük.

Ekkor a **sikeres keresés** várható próbaszáma legfeljebb $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ ami azt jelenti, hogy a keresés $O(1)$ idő alatt lefut. Pl.

- ha a hash tábla félig van kitöltve, akkor a sikeres keresés várható próbaszáma legfeljebb $1{,}387$
- ha a hash tábla 90%-ban van kitöltve, akkor a sikeres keresés várható próbaszáma legfeljebb $2{,}559$