Hasító táblák

A hasító táblák (hash tables) célja egy olyan adattárolási eljárás megvalósítása, amelyben a keresés, beszúrás és törlés (szótár)-műveletek várhatóan nagyon hatékonyak. (Akár elsőre megtalálható minden keresett kulcs.) Feltesszük, hogy a tárolandó rekordoknak van egy egyedi kulcs adattagja. A kulcsok egy U halmaz elemei (U a kulcsok univerzuma), amiről ezúttal nem szükséges feltételezni, hogy rendezhető.

Fontos jelölések a jegyzetből:

Jelölések:
m : a hasító tábla mérete
$T[0..m)$: a hasító tábla
$T[0], T[1], \ldots, T[m-1]$: a hasító tábla rései (slot-jai)
$\oslash$: üres rés a hasító táblában (direkt címzésnél és a kulcsütközések láncolással való feloldása esetén)
E : üres rés kulcsa a hasító táblában (nyílt címzésnél)
D : törölt rés kulcsa a hasító táblában (nyílt címzésnél)
n : a hasító táblában tárolt adatok száma
$\alpha = n/m$: a hasító tábla kitöltöttségi aránya (load factor)
U : a kulcsok univerzuma; $k, k', k_i \in U$
$h : U \to 0..(m-1)$: hasító függvény

Feltesszük, hogy a hasító tábla nem tartalmazhat két vagy több azonos kulcsú elemet, és hogy $h(k)$, $\Theta(1)$ időben számolható.

1. Direkt címzés

A legegyszerűbb megvalósítás az ún. direkt címzés. Ezt akkor használhatjuk, ha a kulcshalmaz nem túl nagy. Ha $U = [0..m)$, ahol $m \geq n$, de m nem túl nagy, akkor egy m méretű tömbben, hasítótáblában tárolhatjuk a rekordokra mutató pointereket úgy, hogy a tömb k indexű tagja a k kulcsú rekordra mutat. Amennyiben nincs k kulcsú rekord, a megfelelő pointer értéke $\varnothing$.

T:D*[m], ahol a D típus a jegyzetből:

D
+ $k : U$ // k is the key
+ ... // satellite data

Amennyiben a kulcshalmaz nagy, a direkt címzés nem alkalmazható. Ilyenkor felveszünk nullától indexelt, m résből álló T[m] hasítótáblát, és bevezetünk egy $h : U \to [0..m)$ hasító függvényt (tipikusan $|U| >> m$). A k kulcsú rekordot a hasítótábla T[$h(k)$] részében próbáljuk meg eltárolni. Ha két adat $k1$ és $k2$ kulcsára $h(k1) = h(k2)$, akkor kulcsütközésről beszélünk. Ezt a direkt címzés nem tudja kezelni.

A most ismertetett különféle hasítási eljárások a kulcsütközést is képesek kezelni, a h függvény megválasztásában és a kulcsütközések kezelésében különböznek egymástól. A hasító táblában tárolt rekordok kulcsai páronként különbözők kell legyenek, hogy a kulcsok a rekordokat egyértelműen azonosítsák. Beszúráskor tehát ellenőrizni kell, hogy a hasító tábla tartalmaz-e a beszúrandó rekorddal azonos kulcsú bejegyzést. Ha igen, akkor a beszúrás meghiúsul. Ezért minden beszúrás kereséssel kezdődik.

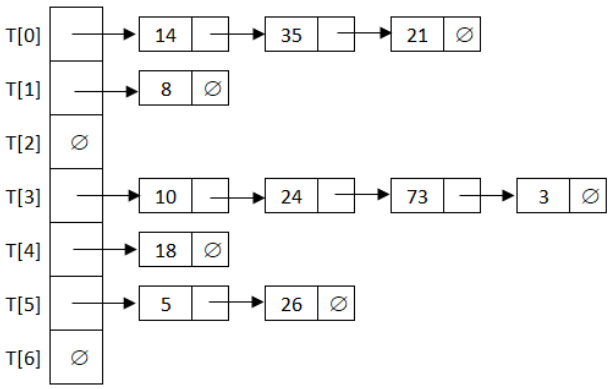
2. Láncolt (vödrös) hashelés

Az adatrekordokat *m* darab S1L listában tároljuk. A hasítótábla T[*i*] eleme az *i*. láncolt lista első elemére mutató pointer (vagy ∅). Ez a lista azokat a *k* kulcsú elemeket tartalmazza, amikre *h*(*k*) = *i*. Beszúráskor a megfelelő lista elejére szúrjuk be az új elemet azért, hogy a beszúrás minél egyszerűbb legyen (továbbá, mert a megfigyelések szerint az újabb elemekhez nagyobb valószínűséggel szeretnénk a közeljövőben hozzáférni). A beszúrás előtt ellenőrizzük a megfelelő listát, a duplikált kulcsok elkerülése céljából. T:E1*[*m*]

Példa: *m* = 7, *h* : N→0..6, *h*(*k*) = *k* mod 7. A táblába beszúrt elemek sorban:
3, 21, 73, 8, 24, 35, 26, 14, 5, 10, 18.

A kapott hasító tábla a listákkal:

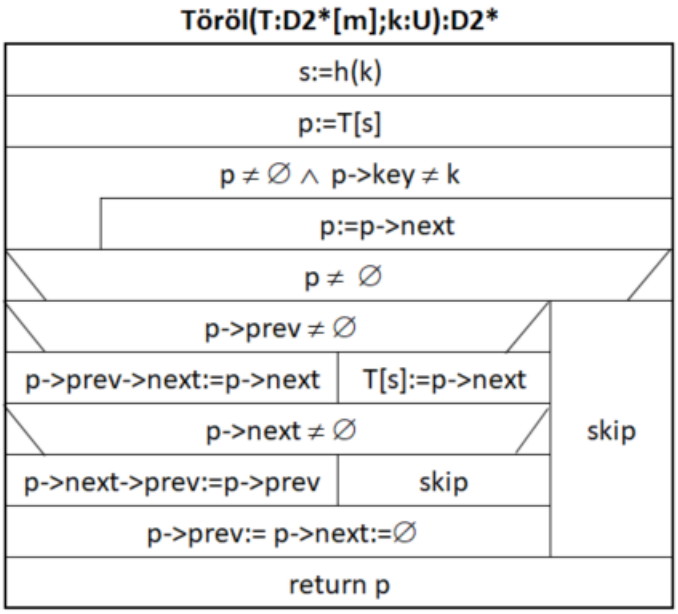
k	h(k)
3	3
21	0
73	3
8	1
24	3
35	0
26	5
14	0
5	5
10	3
18	4



Feladat: Készítsük el a törlés művelet struktogramját! A listák legyenek kétirányú, fejelem nélküli, nem ciklikus listák (S2L). A listaelemek típusára bevezetjük a D2 elemtípust:

D2
+ *prev,next:* D2* + *key*: U …. *egyéb adatok*
+D2() = {*prev:=next:=*∅}

Vigyázni kell az első és utolsó elem törlésénél (első elemnek nincs előzője, utolsó elemnek nincs rákövetkezője).



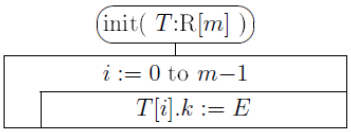
3. Nyílt címzéses hashelés

Ebben az esetben az adatrekordokat közvetlenül a hasítótábla réseiben tároljuk.

A hasító táblában R típusú rekordokat tárolunk. Az R típusú elemeknek van egy kulcsmezőjük, és lehetnek járulékos mezőik. A kulcshalmazhoz hozzáveszünk egy E és egy D extremális konstanst, amelyekkel azt fogjuk jelölni, hogy a tábla adott rése még üres, vagy már volt benne adat, de töröltük.

T:R[m], T[i].k ∈ $U \cup \{E,D\}$

R
$+\ k : U \cup \{E, D\}$ // k is a key or it is Empty or Deleted
$+\ \ldots$ // satellite data



Próbafüggvény, potenciális próbasorozat fogalma (jegyzet):

Jelölések a nyílt címzéshez:
$h : U \times 0..(m-1) \to 0..(m-1)$: próbafüggvény
$\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$: potenciális próbasorozat

Feltesszük, hogy a hasító táblában nincsenek duplikált kulcsok[24].

Az üres és a törölt réseket együtt *szabad* réseknek nevezzük. (A többi rés *foglalt*.) Egyetlen hasító függvény helyett most m darab hasító függvényünk van:

$$h(\cdot, i) : U \to 0..(m-1) \qquad (i \in 0..(m-1))$$

Mivel a résekben közvetlenül helyezzük el az adatrekordokat, kezelnünk kell a kulcsütközéseket: amikor két különböző kulcs ugyanarra a résre képződik le. Emiatt vezetjük be a próbafüggvény fogalmát, amely minden *k* kulcsra megadja a tábla réseinek egy permutációját.

Ezen potenciális próbasorozat mentén hajtjuk végre a táblák alapműveleteit: a keresést, beszúrást, törlést. A műveletekhez általában nem használjuk fel a teljes potenciális próbasorozatot, hanem csak annak egy prefixét, ezt nevezzük aktuális próbasorozatnak.

Olyan próbasorozatot választunk, amely az első eleméből könnyen rekonstruálható. A próbasorozat első elemét egy *elsődleges* hasító függvény adja meg: $h(k,0) = h_1(k)$. Az elsődleges hasító függvényt h_1 jelöli.

Három nevezetes próbasorozat típussal fogunk foglalkozni: a lineáris-, a négyzetes-, és a kettős hasítás próbasorozattal.

Beszúrás (amennyiben nincs törlés művelet): Ha a *k* kulcsú elemet akarjuk beszúrni a táblába, akkor először megnézzük, hogy szabad-e a h(k,0) indexű rés. Amennyiben igen, beszúrjuk, ha nem, akkor megnézzük szabad-e a h(k,1) indexű rekesz. És így tovább, egészen addig, amíg nem találunk neki egy szabad helyet, vagy végig nem próbáljuk a teljes potenciális próbasorozatot. Ha közben a beszúrandó kulccsal találkozunk, vagy a potenciális próbasorozat végéig sem találunk szabad rést, a beszúrás meghiúsul.

Ha törlést is megengedünk a táblában, megváltozik a beszúrás algoritmusa: az első üres helyig tartó keresés közben megjegyezzük az első „törölt" rekesz indexét (ha van ilyen), mert érdemes lesz oda beszúrni a rekordot (ha az még nincs a táblában), hogy a keresésnél minél hamarabb megtaláljuk majd. Viszont azt, hogy nincs még a táblában a megadott kulcsú rekord, csak akkor tudhatjuk meg biztosan, ha az első üres rekeszig folytatjuk a keresést. Előfordulhat, hogy már minden rekeszben van vagy volt valamikor adatrekord, így arra is gondolni kell, hogy esetleg nem fogunk üres rekeszt találni, ezért menet közben érdemes az elvégzett próbák számát is nyilvántartani, ellenőrizni.

Keresés: Amennyiben meg szeretnénk keresni a *k* kulcsú elemet, úgy elkezdjük sorban végig nézni a h(k,0), h(k,1),..., h(k, m-1) indexű réseket egészen addig, amíg meg nem találjuk a keresett elemet, vagy üres helyre nem akadunk. Ha üres helyet találtunk, akkor nincs értelme tovább folytatni a keresést.

Ha törlést is megengedünk a táblában, a „törölt" réseket kereséskor úgy kezeljük, mintha foglaltak lennének, folytatjuk a keresést az első üres résig. Az előbbi megjegyzés itt is fontos lehet, hogy a tábla már nem tartalmaz üres rekeszt a sok törlés, beszúrás miatt.

Törlés: Egy elem törlése során először megkeressük az előző eljárással a törlendő adatrekord helyét. Ha a tábla tartalmazza a törlendő adatrekordot, fontos, hogy a művelet után nem változtathatjuk a felszabadult rést üresre (E), hiszen ez gondot okozhatna a későbbi kereséseknél, beszúrásoknál. Épp ezért a törölt elemet tároló rekeszben az addig ott tárolt kulcsot lecseréljük az U halmazban nem szereplő D (deleted) értékre, ami jelzi, hogy a rekeszbe elhelyezhetünk új adatrekordot, de korábban ez már volt foglalt státuszú.

Észrevehető, hogy ha egy táblában a törlések folytán már nincsenek üres rekeszek, a sikertelen keresés -és így az összes többi művelet, mely Empty, azaz üres résig keres- hatékonysága nagyon elromlik. Ezen a tábla újraépítésével szoktak segíteni.

3.1 Lineáris próba

Lineáris próba esetén a próbasorozat egy számtani sorozatot alkot, ennek differenciája a leggyakrabban +1, vagy -1, esetünkben +1 lesz.

Tehát $h(k,i) = (h_1(k) + i) \bmod m$, $i=0,..,m-1$

Példa: $m = 11$, $h_1(k) = k \bmod 11$, $h(k,i) = (h_1(k) + i) \bmod 11$, $i=0,1,..,10$

Művelet	kulcs	$h_1(k)$	próbasorozat	siker?	0	1	2	3	4	5	6	7	8	9	10
i	24	2	2	✓			24								
i	16	5	5	✓			24			16					
i	57	2	2, 3	✓			24	57		16					
i	32	10	10	✓			24	57		16					32
i	15	4	4	✓			24	57	15	16					32
d	57	2	2, 3	✓			24	D	15	16					32
i	21	10	10, 0	✓	21		24	D	15	16					32
s	2	2	2, 3, 4, 5, 6	×	21		24	D	15	16					32
i	2	2	2, 3, 4, 5, 6	✓	21		24	2	15	16					32
i	2	2	2, 3	×	21		24	2	15	16					32
s	21	10	10, 0	✓	21		24	2	15	16					32
i	35	2	2, 3, 4, 5, 6	✓	21		24	2	15	16	35				32
d	15	4	4	✓	21		24	2	D	16	35				32
i	35	2	2, 3, 4, 5, 6	X	21		24	2	D	16	35				32

Elsődleges csomósodás jelensége:
Az utolsó beszúró műveletnél már jól megfigyelhető a probléma: kulcsütközés esetén az azonos próbasorozatok miatt nagyon besűrűsödik a táblának egy-egy szakasza, így megnő a keresés (beszúrás, törlés) lépésszáma. (Nem egyenletes a lineáris próba.)

A lépések lejátszása után határozzuk meg a hasítótábla kitöltöttségi arányszámát: α =6/11=0,54545455

Érdekes a keresés várható maximális lépésszáma. Az erre vonatkozó képleteket a jegyzetben arra az esetre találjuk meg, amikor nincs a táblában törlés művelet, és a hasítás egyenletes. Ez a fenti táblára és próbasorozatra ugyan nem teljesül, de érdekességképpen megnézhetjük, mit adna a képlet ideális esetben erre a kitöltöttségre.

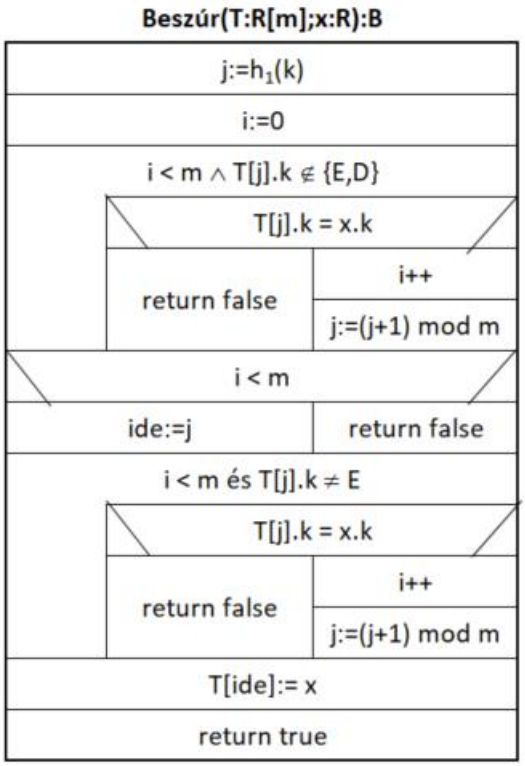
> Amennyiben a táblában nincsenek törölt rések – egyenletes hasítást és a hasító tábla $0 < \alpha < 1$ kitöltöttségét feltételezve –, egy sikertelen keresés illetve egy sikeres beszúrás várható hossza legfeljebb
>
> $$\frac{1}{1-\alpha}$$
>
> míg egy sikeres keresés illetve sikertelen beszúrás várható hossza legfeljebb
>
> $$\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

Egyenletes hasítás: Ideális esetben egy tetszőleges potenciális próbasorozat a (0,1,..,m-1) sorozatnak mind az m! permutációját azonos valószínűséggel állítja elő. Ilyenkor egyenletes hasításról beszélünk.

Ideális esetet feltételezve, az alulról a harmadik sorban (ahol nincs törölt rés) a kitöltöttség 7/11≈64%.
Ehhez:
a sikertelen keresés, vagy sikeres beszúrás várható értékben legfeljebb 2,75 lépésből,
a sikeres keresés, vagy sikertelen beszúrás várható értékben legfeljebb 1,59 lépésből állna.

A lineáris próba sajnos nem ideális, ui. *m*! helyett csak *m* féle próbasorozatot fog előállítani! Gyakorlatban akkor használható, ha a kulcsütközés valószínűsége nagyon kicsi.

Feladat: Készítsük el a nyílt címzéses hasítás beszúró műveletének struktogramját lineáris próbasorozat esetére! Az algoritmus visszatérési értéke egy logikai érték, ami megmondja, hogy sikeres volt-e a beszúrás.



R - rekord típus

elsődleges hasító függvény (a nulladik próba helye)

próbák száma

Első üresig megyünk

Ha esetleg a táblában már szerepel az adott

kulcsú rekord, nem szúrjuk be újra.

Számoljuk a próbák számát, és kiszámítjuk a következő próba index

Ha i=m, nem volt sem üres, sem törölt, a tábla megtelt.

Egyébként megjegyezzük a rekesz címét

és üresig még vizsgáljuk, hogy esetleg az adott kulcs szerepel-e

a táblában.

Ciklus feltétele figyeli a próbaszámot, és a rekesz státuszát (üres-e).

Sikeres a beszúrás, "ide" vagy az első törölt rés indexe, vagy ha törölt ner

akkor az első üres rés indexe.

3.2 Négyzetes próba

Négyzetes próba esetén a próbasorozat egy másodfokú függvény segítségével írható le, vagyis $h(k, i) = (h_1(k) + c_1*i + c_2*i^2) \bmod m$ (i=0,1,..m-1) valamilyen c_1 és c_2 valós konstansokra ($c_2 \neq 0$). Ezeket a konstansokat úgy kell megválasztani, hogy a próbasorozat a teljes táblát kiadja. Például, ha *m* egy kettőhatvány, akkor $c_1 = c_2 = 1/2$ egy jó választás.

A próbasorozat ilyenkor a következő rekurzív képlettel számítható ki:

$h(k,0) = h_1(k)$ $h(k,i) = (h(k,i-1)+i) \bmod m$, i=1,..,m-1

Példa: m = 8, $h_1(k) = k \bmod 8$, próbasorozatunk a fenti képlet szerinti.

Művelet	kulcs	$h_1(k)$	próbasorozat	siker?	0	1	2	3	4	5	6	7
i	13	5	5	✓						13		
i	20	4	4	✓					20	13		
i	31	7	7	✓					20	13		31
i	87	7	7, 0	✓	87				20	13		31
i	12	4	4, 5, 7, 2	✓	87		12		20	13		31
d	31	7	7	✓	87		12		20	13		D
s	12	4	4, 5, 7, 2	✓	87		12		20	13		D
s	15	7	7, 0, 2, 5, 1	×	87		12		20	13		D
i	4	4	4, 5, 7, 2, 6	✓	87		12		20	13		4
d	10	2	2, 3	×	87		12		20	13		4
i	35	3	3	✓	87		12	35	20	13		4
i	10	2	2, 3, 5, 0, 4, 1	✓	87	10	12	35	20	13		4

$\alpha = 7/8 = 0{,}88$

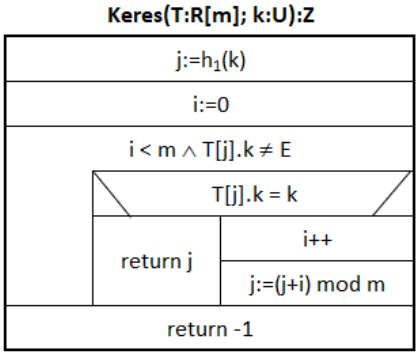
Elméleti, ideális esetben:

Ideális esetet feltételezve, a 88%-os kitöltöttséghez:
a sikertelen keresés, vagy sikeres beszúrás várható lépésszáma legfeljebb: 8
a sikeres keresés, vagy sikertelen beszúrás várható lépésszáma legfeljebb: 2,78

Például a sikeres keresés tényleges lépésszáma az elméleti, ideális eset várható lépésszámánál gyakran több. Sajnos ennél a módszernél is jelentkezik probléma, az úgynevezett másodlagos csomósodás problémája: az azonos próbasorozatok mentén keletkeznek a táblában csomók, amelyek itt is azt eredményezik, hogy megnő a keresések (beszúrások, törlések) lépésszáma.

A négyzetes próba sem tekinthető egyenletes hasításnak. (Csak m különböző próbasorozatunk van.)

Feladat: Készítsük el a keresés struktogramját erre a négyzetes próbára! Az algoritmus visszatérési értéke egy egész szám, ami megadja a paraméterül kapott kulcsú rekord indexét. Sikertelen keresés esetén a visszatérési érték -1 lesz.



Megjegyzés: persze a kettőhatvány alakú tábla méret nem kedvező az osztó módszert (mod m) használó hasító függvénynek. Így léteznek más alakú négyzetes próbasorozatok is. Egy érdekes változat: ha m egy 4d+3 alakú prím (d ∈ N), akkor a következő próbasorozat szintén lefedi a teljes táblát:

$h(k,0)=h_1(k)$
$h(k,1)=h(k,0)-1 \bmod m$, $h(k,2)= h(k,0)+1 \bmod m$
$h(k,3)=h(k,0)-2^2 \bmod m$, $h(k,4)= h(k,0)+2^2 \bmod m$
azaz h(k,0) -hoz képest $-j^2$, $+j^2$ távolságra próbálkozunk, , j=1,..,(m-1)/2 (persze mod m értve a távolságot).

De sajnos a másodlagos csomósodás jelensége ekkor is jelentkezik a próbasorozatok mentén.

3.3 Kettős hasítás

Ebben az esetben a próbasorozat egy számtani sorozat, azonban a differencia kulcsonként eltérő. Az első elemet egy elsődleges hasító függvény adja: h_1: U → 0..m− 1, a számtani sorozat differenciáját pedig egy másodlagos hasító függvény: h_2: U → 1..(m− 1) függvény adja. Innen a módszer neve, két hasító függvényt használunk, az elsőt a szokásos módon, a másodikat a próbasorozat lépéshosszának megállapításához.

A próbasorozat általános képlete: $h(k,i) = (h_1(k) + i*h_2(k)) \bmod m$, i=0,..,m-1.

Két fontos elvárás van h_2 hasító függvénnyel kapcsolatban:

1. ne adjon semmilyen *k* értékre nullát,
2. legyen tetszőleges *k* érték esetén relatív prím a tábla méretéhez képest.

Ezek könnyen teljesíthetők. Mivel általában a tábla méretét prímnek választják, a második hasító függvénynek alkalmas, és megfelel mindkét feltételnek az 1+(k mod (m-1)) függvény, így példánkban is ennek választjuk.

Példa: m = 11, $h_1(k)$ = k mod 11, $h_2(k)$ = 1 + (k mod 10).

Művelet	kulcs	$h_1(k)$	$h_2(k)$	próbasorozat	siker?	0	1	2	3	4	5	6	7	8	9	10
i	23	1	4	1	✓		23									
i	42	9	3	9	✓		23								42	
i	31	9	2	9, 0	✓	31	23								42	
i	110	0	1	0, 1, 2	✓	31	23	110							42	
i	55	0	6	0, 6	✓	31	23	110				55			42	
d	31	9	2	9, 0	✓	D	23	110				55			42	
d	55	0	6	0, 6	✓	D	23	110				D			42	
s	13	2	4	2, 6, 10	×	D	23	110				D			43	
i	6	6	7	6, 2, 9, 5	✓	D	23	110				6			44	
d	31	9	2	9, 0, 2, 4	×	D	23	110				6			44	
i	13	2	4	2, 6, 10	✓	D	23	110				6			44	13
s	11	0	2	0, 2, 4	×	D	23	110				6			44	13

A kettős hasítás közelíti legjobban az egyenletes hasítást, hiszen kulcsütközés esetén általában nem ugyanazon próbasorozat mentén próbálkozunk, a lépések nagysága szintén függ a kulcstól, így kevésbé tud a tábla csomósodni.

4. A hasítófüggvény megválasztása

A h : U → 0..(m-1) függvény egyszerű egyenletes hasítás, ha a kulcsokat a rések között egyenletesen szórja szét, azaz hozzávetőleg ugyanannyi kulcsot képez le az m rés mindegyikére. Fontos az is, hogy kiszámítása ne legyen túlságosan költséges (Θ(1)). Tetszőleges hasító függvénnyel szembeni elvárás, hogy egyszerű egyenletes hasítás legyen.

4.1 Osztó módszer

Ha a kulcsok egész számok, gyakran választják a

h(k) = k mod m

hasító függvényt, ami gyorsan és egyszerűen számolható, és ha m olyan prím, amely nincs közel a kettő hatványokhoz, általában egyenletesen szórja szét a kulcsokat a 0..(m-1) intervallumon.

A tábla méretének megválasztása: m legyen prím, és legyen „távol” a kettő hatványoktól.

A jegyzetben található példa: ha pl. a kulcsütközést láncolással szeretnénk feloldani, és kb. 2000 rekordot szeretnénk tárolni $\alpha \approx 3$ kitöltöttségi aránnyal, akkor a 701 jó választás: A 701 ui. olyan prímszám, ami közel esik a 2000/3-hoz, de a szomszédos kettőhatványoktól, az 512-től és az 1024-től is elég távol van.

4.2 Kulcsok a [0, 1) intervallumból:

Ha egyenletesen oszlanak el, a

$h(k) = \lfloor k * m \rfloor$

függvény is kielégíti az egyszerű, egyenletes hasítás feltételét.

4.3 Szorzó módszer:

Ha a kulcsok valós számok, tetszőleges $0 < A < 1$ konstanssal alkalmazható a

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

hasító függvény. ({x} az x törtrészét jelöli.) Kihívást jelent a jó *A* konstans értékének megválasztása. Nem minden lehetséges konstanssal szór egyformán jól. Donald E. Knuth[1] az $A = (\sqrt{5}-1)/2 \approx 0{,}618$ választást javasolja, mint ami a kulcsokat valószínűleg szépen egyenletesen fogja elosztani a rések között. Az osztó módszerrel szemben előnye, hogy nem érzékeny a hasító tábla méretére.

Ha a kulcsaink nem számok, hanem például sztringek, akkor valamely egyszerű transzformációval könnyen képezhetünk belőlük egészeket, amelyhez aztán az osztó vagy szorzó módszerrel már készíthetünk alkalmas hasító függvényeket.

[1] Donald E. Knuth: A számítógép-programozás művészete (The Art of Computer Programming), Addison-Wesley Professional, magyarul is megjelent a Műszaki Könyvkiadó gondozásában.