4.

Hash-elés és szekvenciális keresés

Hány hajó ütközött itt össze; jóllehet voltak jelzőfényeik, kürtjeik és vészharangjaik! JULES VERNE
(A *Nemo kapitány* elbeszélőjének
tűnődése Új-Fundland vizein.)

Ebben a fejezetben először egy olyan módszercsaládot ismertetünk, amely a keresés, beszúrás, törlés és módosítás gyors és egyszerű megvalósítását teszi lehetővé. A keresőfákhoz képest fontos eltérés, hogy nem tételezzük fel a lehetséges kulcsok összességének (az U univerzumnak) a rendezettségét. Ennek megfelelően itt nem lesznek rendezéshez kapcsolódó elérési igények, mint pl. a MIN. A cél egy $S \subseteq U$ kulcshalmazzal azonosított állomány megszervezése úgy, hogy a fenti műveletek hatékonyak legyenek. A megoldások, amelyekkel megismerkedünk, *átlagos értelemben* igen gyorsak. A legrosszabb esetekben viszont nagyon lassúak is lehetnek. Olyan alkalmazásoknál jönnek tehát szóba, ahol az átlagosan jó teljesítmény az igazán fontos, és a ritkán előforduló rosszabb válaszidők nem okoznak gondot.

Először egy rövid példával szeretnénk szemléltetni a hash-elés hátterét, alapötletét és a felmerülő problémákat. Tegyük fel, hogy magyar állampolgárok adatait szeretnénk tárolni. Az állomány egy rekordjában olyan adatok (mezők) szerepelhetnek, mint például név, lakcím, személyi szám, stb. A rekordokat azonosító kulcsként a személyi szám szolgálhat. A személyi szám 11 jegyű, ebből egy redundáns, és a további ismert korlátozásokat is figyelembe véve (a hónapok száma, a hónapok napjainak száma, stb.) mintegy $2 \cdot 10^2 \cdot 12 \cdot 31 \cdot 10^3 \approx 74$ millió kulcs lehetséges. Ugyanakkor még jó adag demográfiai optimizmus esetén sem kell több, mint 11 millió tényleges rekorddal számolnunk a közeljövőben. Nagyvonalúan – ami mint később látni fogjuk, hasznos ebben az összefüggésben is – gondolkodhatunk úgy, hogy 12 millió rekord számára foglalunk helyet. Legyen a rekordok

(logikai) címeinek tartománya a $[0, 12 \cdot 10^6 - 1]$ intervallum egészeinek I halmaza. Ezután a feladatot felfoghatjuk úgy is, hogy egy *olyan megfeleltetést keresünk, mely minden kulcshoz egy címet rendel.* Olyan h függvényre gondolunk, amelynek az értelmezési tartománya a lehetséges kulcsok halmaza, értékkészlete pedig I, a logikai címek tartománya. Kényelmes volna, ha $K \neq K'$ esetén $h(K) \neq h(K')$ teljesülne. Ez azonban képtelenség, hiszen az értelmezési tartomány jóval nagyobb, mint az értékkészlet, ezért h nem lehet injektív. Az ilyen *ütközések* tehát elkerülhetetlenek. Sőt, bizonyos értelemben elég gyakran előfordulnak. Erre világít rá a születésnap-paradoxon (szokás még von Mises-paradoxonnak is nevezni): egy legalább 23 tagú társaságban legalább $1/2$ valószínűséggel van két személy, akiknek megegyezik a születésnapjuk. Általában megmutatható, hogy ha a h függvény értékkészlete M elemű, akkor $\sqrt{2 \ln 2 \cdot M}$ véletlenül választott argumentum között legalább $1/2$ valószínűséggel lesz kettő, melyre h ugyanazt az értéket adja. E tény azt sugallja, hogy jó esély van ütközésre még akkor is, amikor címtartomány méretéhez képest viszonylag kevés rekordunk van a tárolt állományban.

A *hash-elés* vagy *hash-kódolás* módszerét használó tárolási technikák alapvető közös vonása, hogy egy alkalmas h függvény, az úgynevezett hash-függvény segítségével kísérlik meg kiszámítani a beillesztendő rekord (logikai) címét. A h függvény a K kulcshoz a $h(K)$ címet rendeli. A módszerek első közelítésben a K kulcsú rekordot a $h(K)$ címnek megfelelő helyre próbálják tenni. Ez nem lehetséges ütközés esetén: amikor egy másik K' rekord érkezik, melyre $h(K) = h(K')$. A teendők fontos része éppen ezért az *ütközések feloldása.* Olyan megoldásokról van itt szó, amelyek ütközéskor helyet találnak a később jött rekordoknak. Az egyes módszereket az ütközések kezelése alapján szokás osztályozni. Ezt tesszük mi is.

A másik alapvető kérdést a *megfelelő hash-függvény* találása, kiválasztása jelenti. Olyan könnyen számítható h függvényeket keresünk, amelyek az alkalmazás során felmerülő kulcshalmazokon minél kevesebb ütközést okoznak. Ennek a mikéntjeivel is foglalkozunk.

A *szekvenciális kereséssel* már találkoztunk a rendezések tárgyalásakor. Ott rendezett állományokban való keresésre használtuk a módszert. A fejezet végén ejtünk néhány szót arról az esetről, amikor az állomány nem feltétlenül rendezett.

4.1. A hash-elés alapjai

Két igen szerteágazó módszer legegyszerűbb változatait ismertetjük. Ezek a *vödrös hash-elés* és a *nyitott címzés*. Az első technikát, a vödrös hash-elést elsősorban külső táron levő nagyméretű állományok kezelésére használják. Szinte minden komoly adatbáziskezelő rendszer ad ilyen jellegű tárolási lehetőséget. A nyitott cím-

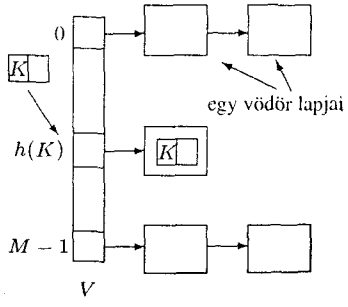
zésű módszerek ezzel szemben főleg a belső memóriában levő állományok gyors kezelését hivatottak biztosítani. Jellegzetes alkalmazásként említhetjük a programok fordításakor keletkező szimbólumtáblák kezelését vagy a később bemutatandó Lempel–Ziv–Welch-módszer szótárának a tárolását.

A továbbiakban mindkét esetre érvényesen feltesszük, hogy van egy h hash-függvényünk, amely a K kulcshoz a $h(K)$ logikai címet rendeli. A $h(K)$ cím egy egész a $[0, M-1]$ intervallumból. Azért beszélünk *logikai címekről*, mert a $h(K)$ értékek többnyire nem jelentenek tényleges fizikai helymegjelöléseket. Ugyanakkor feltehetjük, hogy a rendszer biztosít egy mechanizmust, amely a $[0, M-1]$ intervallum egészeit a tényleges adatmozgatást végző réteg számára értelmezhető címekké fordítja.

4.1.1. Vödrös hash-elés

A módszert szokásos *láncolásnak* is nevezni. Lényeges eleme a $V[0 : M-1]$ vödörkatalógus. Amint a jelölés is sugallja, V-t hasznos egy a h értékkészletének elemeivel indexelt tömbnek elképzelni. A kép nem teljesen hűséges, mert V nagy is lehet. Előfordulhat, hogy részben vagy egészében háttértáron kell elhelyezni. A V elemei mutatók, pontosabban lapláncok fejei. A lapláncokat szokás *vödröknek* nevezni. A vödrök lapjain helyezkednek el a tárolni kívánt rekordok. Nevezetesen a $V[i]$ mutatóval kezdődő vödörbe kerülnek azok a rekordok, amelyek K kulcsaira $h(K) = i$ teljesül.

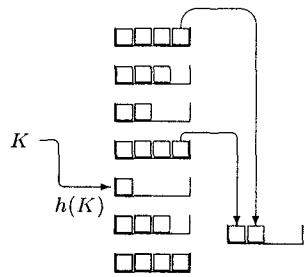
Vegyük szemügyre ezután a szerkezet algoritmusait! Tegyük fel, hogy a K kulcsú rekordot akarjuk beilleszteni. Először kiszámítjuk a $h(K)$ értéket, majd megnézzük a vödörkatalógus $V[h(K)]$ bejegyzését. Ez egy mutató arra a vödörre (lapláncra), melyben a $h(K)$ hash-értékű rekordokat kívánjuk tárolni. A vödör általában egy kicsi, legfeljebb néhány lapból álló állomány. Ebben elhelyezzük a tárolni kívánt rekordot. A következő ábrán a V vödörkatalógust függőleges helyzetben mutatjuk; a vödrök a lapokból álló vízszintes sorok.



A rekordnak a vödörben való elhelyezése több módon is történhet. Gyakran használatos és egyszerű megoldás, hogy a rekordot az első szabad helyre tesszük. Ha szükséges, új lap hozzáfűzésével bővítjük a láncot. Másik szokásos eljárás, hogy a rekordokat kulcs szerint rendezve tartjuk a vödörben. Ekkor – a beszúrásos rendezésnél tapasztaltakhoz hasonlóan – előfordulhat, hogy több rekordot is mozgatni kell a vödrön belül.

A keresés módszere ezek után nyilvánvaló. A K kulcsú rekord keresése avval kezdődik, hogy kiszámítjuk a $h(K)$ értéket. Ha a rekord a tárolt állományunkban van, akkor csak a $V[h(K)]$ mutatótól induló vödörben lehet. Ezután tehát a vödörkatalógus $V[h(K)]$ mutatóját követve a vödör első lapjára lépünk. A vödörben szekvenciális kereséssel dolgozhatunk. Addig megyünk a lapláncmentén, amíg vagy megtaláljuk a rekordot, vagy pedig kiderül, hogy az nincs a tárolt állományban. A törléssel és a módosítással kapcsolatos teendők a keresés után már kézenfekvőek, így nem részletezzük őket. Csak annyit jegyzünk meg, hogy ha a módosítás a kulcsot is érinti, akkor törlés, majd újbóli beillesztés válhat szükségessé, hiszen a kulcs módosulásával annak h-értéke is változhat.

A vödrös hash-elés kitűnően alkalmazható külső tárakon elhelyezkedő állományok kezelésére. A módszer ötlete jól illeszthető a használt fizikai tárolási médium sajátosságaihoz. A vödrök gyakran megvalósíthatók a lapláncnál alacsonyabb, hatékonyabb szinten. Egy vödör lehet pl. a mágneslemez egy sávja (track) vagy annak írás/olvasás szempontjából folytonos része. Ilyenkor az utolsó sáv szolgálhat a túlcsordulások kezelésére:



Mi mondható a módszer költségéről? Mivel külső táras adatszerkezetről van szó, a lapelérések számát kell vizsgálnunk. Ezt szem előtt tartva világos, hogy a láncolási módszer időigényét (költségét) a lapláncok hossza határozza meg. Tegyük fel, hogy M vödör van, és l-lapnyi rekordot tárolunk. Ekkor egy vödörbe átlagosan $\approx l/M$ lap kerül. Ennyi lesz tehát az átlagos lánchossz. Ha a keresés során minden vödörhöz ugyanakkora eséllyel fordulunk, akkor a keresés átlagos költsége legfeljebb $1 + l/M$ lapelérés. Itt azzal a feltételezéssel éltünk, hogy a

vödörkatalógus is háttértáron helyezkedik el, és $V[h(K)]$ olvasása egy lapelérésbe kerül. Az átlagszámításban hallgatólagosan feltettük még azt is, hogy a h függvény elég egyenletesen szórja szét az érkező kulcsokat a $[0, M-1]$ intervallumban.

A módszer alkalmazása előtt fontos méretezési feladat a katalógus elemszámának, az M paraméternek a meghatározása. Általában ismerjük (esetleg csak közelítőleg) a tárolni kívánt állomány nagyságát kifejező l értéket. Ennek birtokában az M értékét célszerű úgy választani, hogy a várható l/M hányados közel legyen 1-hez. Nem szokatlan a 20 százalékos rátartás, vagyis többlethely lefoglalása sem.

Példa: Tegyük fel, hogy egy maximum 1 000 000 rekordból álló állományt szeretnénk láncolásos módszerrel kezelni. Tegyük fel, hogy egy lapon 5 rekord fér el. Ekkor $l = 1\,000\,000/5 = 200\,000$. Az M paraméter értékét $220\,000 - 240\,000$ körülinek érdemes választani. Ha a katalógus elemei is diszken vannak, és egyenként elérhetők 1 lépésben, akkor a keresés átlagos költsége valamivel 2 lapelérés alatt marad.

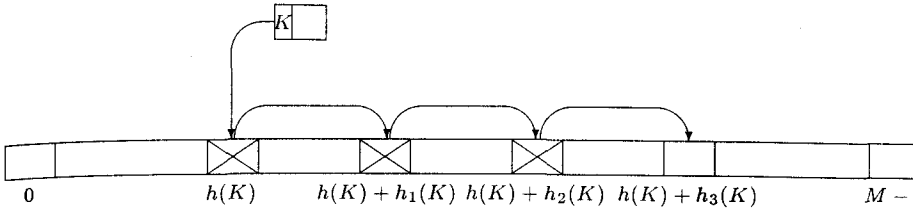
Végezetül megemlítjük, hogy a láncolás módszere belső memóriás megoldásként is versenyképes. Az előbbiekhez viszonyítva az eltérés annyi, hogy itt nem lapokat, hanem rekordokat láncolunk. Egy vödör tehát egy rekordokból álló láncot jelent.

4.1.2. Nyitott címzés

A nyitott címzést használó módszerek általában – az itt bemutatásra kerülő technikák pedig különösen is – csak belső memóriás módszerként hasznosak. A rekordokat a $T[0 : M-1]$ táblában (tömbben) kívánjuk elhelyezni. A T tömböt tehát a h függvény értékkészletének elemeivel, a $[0, M-1]$ intervallum egészeivel indexeljük. A T elemei pedig rekordok. A K kulcsú rekordnak a $T[h(K)]$ helyet szánjuk, feltéve, hogy az nem foglalt.

Ellenkező esetben, amikor a $T[h(K)]$ cellában már egy korábban beillesztett rekord van, akkor valamilyen szisztematikus módon próbálunk helyet keresni a T további celláiban. A nyitott címzésű módszerek ütközésfeloldásra a $0, 1, \ldots, M-1$ számok egy $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ *permutációját* használják. Pontosabban fogalmazva sorra végigpróbáljuk a $h(K) + h_i(K)$ sorszámú cellákat ($i = 0, 1, \ldots, M-1$) az első üres helyig, ahol a rekordot elhelyezzük. Itt az összeadás modulo M értendő, vagyis az összeg mindig 0 és $M-1$ közötti egésznek vehető. Permutációra azért van szükség, hogy a $h(K)+h_i(K)$ sorszámú cellák kiadják a tábla minden helyét. Ha nem találtunk üres helyet a sorozat mentén, akkor arra következtethetünk, hogy a tábla betelt, az új rekordot már nincs

hová tenni. A $h_i(K)$ számok jelölésében azért tüntettük fel K-t, hogy érzékeltessük: a próbasorozat függhet a kulcstól is.



Az előző bekezdésben tulajdonképpen elmondtuk a beillesztés algoritmusát. A keresés itt is egyszerű. Ha a keresett rekord kulcsa K, akkor sorra megnézzük a $h(K) + h_i(K)$ sorszámú cellákat ($i = 0, 1, \ldots$), amíg megtaláljuk a keresett rekordot, vagy üres cellához érünk, vagy pedig $i = M - 1$. Az utóbbi két esetben a keresett rekord nincs a táblában.

A törlés érdemi része is világos ezután. Megkeressük az eltávolítandó rekordot, majd töröljük a táblából. Ez utóbbi fázis a szokásosnál több körültekintést igényel. Később egy példa kapcsán még foglalkozunk ezzel a kérdéssel.

A módszerek összehasonlításához, hatékonyságuk elemzéséhez hasznosak lesznek a következő jelölések:

N – a táblában levő rekordok száma
M – a tábla celláinak száma
$\alpha = \frac{N}{M}$ – a telítettségi (betöltöttségi) tényező
C_N – a sikeres keresések (amikor a keresett rekord megtalálható T-ben) során megvizsgált cellák átlagos száma
C_N' – a sikertelen keresések (a keresett rekord nincs T-ben) során megvizsgált cellák átlagos száma.

A C_N meghatározásában az „átlagos" úgy értendő, hogy a táblában levő mindegyik elemre egyenlő eséllyel érkezik keresési igény. A C_N' esetében pedig arról van szó, hogy a $h(K)$ érték ugyanakkora eséllyel lehet a $0, 1, \ldots, M - 1$ számok bármelyike. Mindeme előkészületek után lássunk néhány fontosabb próbamódszert:

Lineáris próbálás

Itt $h_i(K) := -i$. A K kulcsú rekord beillesztése során a $h(K)$ című cellától indulva addig lépkedünk egyesével balra, amíg üres helyet nem találunk. A $T[0]$ cella után pedig, ha kell, a $T[M-1]$ cellával folytatjuk. A keresés költségére érvényesek a következők (nem bizonyítjuk):

$$C_N = \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

és

$$C'_N = \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right).$$

Néhány α értékre táblázatba foglaltuk a formulákból adódó keresési időket. Látható, hogy 80 százalékos telítettség felett a sikertelen keresések már elég sok időt igényelnek.

α	2/3	0,8	0,9
C_N	2	3	5,5
C'_N	5	13	50,5

Példa: Tegyük fel, hogy a kulcsok természetes számok, és a rekordjaink csak ebből az egyetlen mezőből állnak. Legyen a táblaméret $M = 7$, és legyen a hashfüggvény $h(K) := K \pmod 7$. Tegyük fel még azt is, hogy az ütközések feloldására lineáris próbálást használunk. Így néz ki a tábla a 3, 11 és 9 kulcsok beillesztése után:

0	1	2	3	4	5	6
		9	3	11		

Ha most a $K = 4$ kulcsot szeretnénk beilleszteni, akkor mivel a $T[4]$, $T[3]$ és $T[2]$ helyek is foglaltak, hármat kell lépnünk a próbasorozat mentén. A négyes a $T[1]$-be kerül:

0	1	2	3	4	5	6
	4	9	3	11		

Töröljük most a 9 kulcsot. Első gondolatunk az lenne, hogy a sikeres keresés után felszabadítjuk a $T[2]$ helyet ugyanolyan NULL értéket adva neki, mint ami a még üres $T[0]$, $T[5]$ és $T[6]$ cellákban van. Ez azonban bajt okozhat, hiszen ezután a $K = 4$ kulcs keresése sikertelen lenne. A próbasorozat mentén NULL értékű cellához jutnánk, amiből azt hihetnénk, hogy a négyes nincs a táblában. A megoldás egy a NULL-tól különböző TÖRÖLT jel használata (ez lehet pl. $*$). A keresések során ezek a cellák átléphetők; úgy tekinthetjük őket, mintha foglaltak volnának. Új rekord beillesztésekor természetesen egy ilyen hely üresnek tekinthető, és ismét használható. A törlés után a helyzet tehát így alakul:

0	1	2	3	4	5	6
	4	*	3	11		

Ezután már nem lesz baj, amikor a négyest keressük. A $T[2]$-ben levő * jelből tudni fogjuk, hogy tovább kell lépnünk a próbasorozat mentén. A törléssel kapcsolatban itt vázolt probléma általános a nyitott címzést használó módszereknél. A javasolt megoldás, a TÖRÖLT jel használata, működik más próbasorozatok esetén is.

A példa egyébként mutatja a lineáris próbálás fő hátrányát is. Ha sok rekord van a táblában, akkor hosszú, egybefüggő „csomók" alakulnak ki, megnövelve a keresési utak hosszát. Mondjuk a 18 keresésekor elég nagy utat kell megtenni, mire kiderül az igazság. Ezt a kellemetlen jelenséget *elsődleges csomósodásnak* (klasztereződésnek) szokás nevezni. A csomósodás oka abban van, hogy ha egy R' rekord beillesztése során elérjük a már tárolt R rekord keresési útját, akkor azt rendszerint végig is kell járnunk.

A lineáris próbálással ismerkedők gyakran kérdezik, hogy miért visszafelé lépkedünk a táblában, és nem előre. A magyarázat programozástechnikai természetű. Az eljárásokban (sokszor) ellenőriznünk kell, hogy a lépkedés során nem értük-e el a tábla végét. Ha visszafelé, a tábla eleje felé megyünk, akkor a nullával való egyenlőséget kell vizsgálni; ha a másik irányt választanánk, akkor egy M körüli egész szerepelne a tesztekben. Az előbbi típusú tesztek időigénye jóval kisebb, amitől a visszafelé lépkedő kód számottevően gyorsabb az előre menőnél.

Álvéletlen próbálás

Ezeknél a módszereknél a $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ próbasorozat a $0, 1, \ldots, M-1$ számoknak egy a K kulcstól független álvéletlen permutációja. A sorozatnak természetesen gyorsan és hatékonyan reprodukálhatónak kell lennie, hiszen minden adatelérési műveletnél használni akarjuk. Ez a kikötés éppenséggel ellentétes a véletlenszerűséggel. Másfelől viszont ha a sorozat eléggé „véletlenszerű", elég szeszélyesen ugrabugrál, akkor a táblában nem tapasztalunk elsődleges csomósodást. A két törekvés egymással ellentétesnek tűnik, de mint példák mutatják, összebékíthetők.

Az álvéletlen próbálás sem mentes teljesen a csomósodástól, bár ez kisebb mértékben rontja a hatékonyságot, mint a lineáris próbálásra jellemző elsődleges csomósodás. Arról van szó, hogy ha a K és L kulcsokra $h(K) = h(L)$, akkor a K és L kulcsok teljes próbasorozata is megegyezik. Ha tehát sok azonos címre kerülő kulcs van, akkor a közös próbasorozatuk mentén alakul ki csomósodás. Ezt a jelenséget *másodlagos csomósodásnak* nevezzük.

Az álvéletlen próbálásra egy hasznos és mutatós példa a *kvadratikus maradék próba*. Legyen M egy $4k+3$ alakú prímszám, ahol k egy egész. Ekkor a próbaso-

rozat legyen

$$0, 1^2, -(1^2), 2^2, -(2^2), \ldots, \left(\frac{M-1}{2}\right)^2, -\left(\frac{M-1}{2}\right)^2.$$

Egy kis számolgatás (az $M = 19$-et javasoljuk az olvasónak) érzékelteti, hogy a sorozat tényleg elég szeszélyesen lépked. Elméleti érvek is szólnak a „véletlenszerűsége" mellett; ezeket most mellőzzük. Megmutatjuk viszont, hogy a sorozat a modulo M maradékosztályok egy permutációját adja. Ez könnyen következik az alábbi tényből.

Állítás: *Ha M egy $4k + 3$ alakú prímszám, akkor nincs olyan n egész, melyre* $n^2 \equiv -1 \pmod{M}$.

Bizonyítás: Indirekt érvelve tegyük fel, hogy n egy egész szám és $n^2 \equiv -1 \pmod{M}$. Ekkor

$$-1 \equiv (-1)^{\frac{M-1}{2}} \equiv n^{2\frac{M-1}{2}} \equiv n^{M-1} \equiv 1 \pmod{M}.$$

Az utolsó lépésnél a kis Fermat-tételt használtuk. Az elejét a végével összevetve $-1 \equiv 1 \pmod{M}$ adódik, ami képtelenség. □

Nyilvánvaló mármost, hogy ha $0 \leq i < j \leq \frac{M-1}{2}$, akkor $i^2 \not\equiv j^2 \pmod{M}$. Ugyanis a $j^2 - i^2 = (j - i)(j + i)$ felbontás egyik tényezője sem lehet osztható M-mel, tehát a szorzatuk sem. Ugyanígy kapjuk, hogy $-i^2 \not\equiv -j^2 \pmod{M}$. Az $i^2 \equiv -j^2 \pmod{M}$ kongruenciából pedig $(ij^{-1})^2 \equiv -1 \pmod{M}$ következne, ahol j^{-1} jelöli a j multiplikatív inverzét modulo M. Ez a kongruencia pedig az előző állítás szerint nem lehetséges. A próbasorozatban tényleg nincs ismétlődés.

Ami a keresés költségét illeti, a legjobb változatok időigénye így alakul (nem bizonyítjuk):

$$C_N \approx 1 - \log(1 - \alpha) - \frac{\alpha}{2}$$

a sikeres keresésre, és

$$C'_N \approx \frac{1}{(1 - \alpha)} - \alpha - \log(1 - \alpha)$$

a sikertelen keresés során megvizsgált cellák átlagos számára. Ezek az összefüggések valamivel általánosabban érvényesek az olyan módszerekre, amelyekre $h_i(K) = f_i(h(K))$; vagyis ahol a $h(K)$ érték már az egész próbasorozatot meghatározza.

Kettős hash-elés (G. de Balbine, J. R. Bell, C. H. Kaman, 1970 körül.)
A recept lényege, hogy a h mellett egy másik h' hash-függvényt is használunk. Utóbbival szemben követelmény, hogy a $h'(K)$ értékek relatív prímek legyenek az M táblamérethez. A K kulcs próbasorozata ekkor $h_i(K) := -ih'(K)$. Ha M és $h'(K)$ relatív prímek, akkor a $0, -h'(K), -2h'(K), \ldots, -(M-1)h'(K)$ sorozat elemei mind különbözők modulo M, így alkalmas lesz próbasorozatnak. E módszerek fontos sajátossága, hogy a különböző K és K' kulcsok próbasorozatai jó eséllyel akkor is különbözők lesznek, ha $h(K) = h(K')$. A legjobb ismert implementációk időigénye (empirikus adatok alapján)

$$C_N \approx \frac{1}{\alpha} \log \frac{1}{(1-\alpha)} \quad \text{és} \quad C'_N \approx \frac{1}{1-\alpha}.$$

A kettős hash-elés hatékony, a gyakorlatban is jó viselkedést mutató módszer. Ezzel kapcsolatos további tapasztalati észrevételek a következők:
1. A kettős hash-elés kiküszöböli mindkétféle csomósodást.
2. Sikertelen keresés esetén minden érdekes α-ra gyorsabb, mint a lineáris próbálás.
3. Sikeres kereséskor csak az $\alpha \geq 0,8$ tartományban lesz gyorsabb a lineáris próbálásnál.

Végezetül bizonyítás nélkül megemlítünk néhány tényt a vödrös hash-elés belső memóriás változatáról. Ekkor rekordokat (cellákat) láncolunk. Egy vödör tehát cellákból álló lánc lesz. Jelölje M a vödörkatalógus méretét. Tegyük fel, hogy a rekordok a kulcs szerint nem csökkenő sorrendben vannak a listájukra fűzve. Ekkor igazolható, hogy ha $0 < \alpha < 1$, akkor

$$C_N \approx 1 + \frac{1}{2}\alpha \quad \text{és} \quad C'_N \approx 1 + \frac{1}{2}\alpha^2.$$

A láncolás általában jóval helyigényesebb a nyitott címzésű módszereknél. A megvizsgált cellák számát tekintve – ezt mérik a C_N és C'_N mennyiségek – viszont hatékonyabb azoknál mind a sikeres, mind a sikertelen kereséseket illetően. Valódi időben mérve az összevetés már nem ennyire egyértelmű, mert a listák kezelése nehézkesebb, mint a tömbszerű tábláké.

Feladat: Igazoljuk a belső memóriás láncolás sikeres keresésének idejét megadó formulát.

4.2. Hash-függvények

A hash-elést használó módszerek hatékonysága nagymértékben függ a h hash-függvény minőségétől. A h függvénnyel szemben két alapvető követelmény van:

legyen könnyen (gyorsan) számítható, és minél kevesebb ütközést okozzon.

Az első követelmény magától értetődő: a módszer alkalmazása során minden rekordelérés egy $h(K)$ érték kiszámítása alapján történik, ahol K egy kulcs.

A második követelmény elég nehezen megfogható, mert a gyakorlatban előforduló kulcshalmazok egyáltalán nem véletlenszerűek (a fejezet elejének példájához kapcsolódva mondjuk a Kovács név sokkal gyakoribb, mint az Eördögh). Kevés elméleti eredmény van, viszont jelentős tapasztalati hátterű ismeret halmozódott fel. Ilyen hasznos empirikus tanácsok például, hogy $h(K)$ értéke lehetőleg a K kulcs minden bitjétől függjön, és a h értékkészlete a teljes $[0, M-1]$ címtartomány legyen. Fontosságát tekintve két módszer emelkedik ki a mezőnyből. Ezeket ismertetjük a következőkben.

Osztómódszer

Legyen $h(K) := K \pmod M$, ahol M a tábla vagy a vödörkatalógus mérete. Itt és a továbbiakban is feltesszük, hogy a kulcsok egész számok. Ez nem jelent lényegi korlátozást. A gyakorlatban előforduló kulcsok általában könnyen és gyorsan konvertálhatók egészekké. Legtöbbször arról van szó, hogy valamilyen bitsorozatot számként értelmezünk. A $h(K)$ számítása gyors és egyszerű: mindössze egy maradékos osztást igényel. A lineáris próbálásnál szereplő példában az osztómódszert alkalmaztuk.

Az osztómódszer használatakor M, a tábla mérete sem teljesen közömbös. Például ha M a 2 egy hatványa, akkor $h(K)$ csak a kulcs utolsó néhány bitjétől függ. A jó M értékeket illetően van egy széles körben elfogadott recept, amit a szakma *D. E. Knuth javaslata*ként tart számon. Eszerint megfelelő, ha M-et prímnek választjuk, úgy, hogy M nem osztja $r^k \pm a$-t, ahol r a karakterkészlet elemszáma (pl. 128, vagy 256) és a, k „kicsi" egészek.

Az, hogy M prím, lényeges feltétel a kvadratikus maradék próbánál. Ugyancsak előnye az ilyen számoknak, hogy könnyű hozzájuk relatív prím számot találni. Az utóbbi tulajdonság a kettős hash-elésnél jut szerephez.

Szorzómódszer

Ennél az eljárásnál egy rögzített β paraméter használatos, ami egy valós szám. Ennek birtokában legyen

$$h(K) := \lfloor M \cdot \{\beta K\} \rfloor.$$

A formulában $\{x\}$ jelöli az x valós szám törtrészét. Szemléletesen az történik, hogy $\{\beta K\}$ kiszámításával a K kulcsot „véletlenszerűen" belőjük a $[0, 1)$ intervallumba, majd az eredményt felskálázzuk a címtartományba.

Hatékonyan számítható speciális eset a következő: legyen a táblaméret $M = 2^t$, jelölje w a gépünk szókapacitását (pl. $w = 2^{32}$), és legyen A egy a w-hez relatív prím egész. Ekkor $\beta = \frac{A}{w}$ választás mellett $h(K)$ igen jól számolható. A számok bináris ábrázolásával dolgozva lényegében egy szorzást és egy eltolást kell elvégezni.

A szorzómódszer (és az osztómódszer is) jól viselkedik számtani sorozatokon, azaz $\{K, K + d, K + 2d, \ldots\}$ alakú kulcshalmazokon. Ilyenek könnyen adódnak gyakorlati adatkezelési feladatokból, amint azt a *termék1, termék2, termék3, ...* és a hasonló szerkezetű kulcshalmazok mutatják. Megmutatható, hogy a $h(K), h(K+d), h(K+2d) \ldots$ sorozat közelítőleg számtani sorozat lesz, azaz h jól „szétdobja" a kulcsok számtani sorozatait. Ennek a ténynek szép elméleti háttere van. Tegyük fel, hogy β egy irracionális szám, és nézzük a $0, \{\beta\}, \{2\beta\}, \ldots, \{n\beta\}$ sorozat eloszlását $[0, 1)$-ben. Nagy n értékre ez egyenletes: egy z hosszúságú $(0 < z < 1)$ részintervallumba körülbelül zn elem esik, ha $n \to \infty$. Ez adódik az alábbi csodaszép eredményből.

Tétel (T. Sós Vera, 1957): *Legyen β irracionális szám, és nézzük a 0, $\{\beta\}$, $\{2\beta\}$,..., $\{n\beta\}$ pontok által meghatározott $n + 1$ részintervallumot $[0, 1)$-ben. Ezek hosszai legfeljebb 3 különböző értéket vehetnek fel, és $\{(n + 1)\beta\}$ a leghosszabbak egyikét fogja két részre vágni.* □

Ismert még, hogy a $[0, 1)$-beli számok közül a legegyenletesebb eloszlást a $\beta = \phi^{-1} = \frac{\sqrt{5}-1}{2} = 0.618033988\ldots$ és a $\beta = \phi^{-2} = 1 - \phi^{-1}$ értékek adják. Ekkor ha $\{(n + 1)\beta\}$ a korábbi osztópontok által meghatározott I részintervallumba esik, és azt a, valamint b hosszúságú részekre osztja, akkor $\frac{1}{2} < \frac{a}{b} < 2$ mindig teljesül. Eszerint érdemes a szorzómódszernél az A egészet úgy választani, hogy $\frac{A}{w}$ közel legyen ϕ^{-1}-hez. A β ilyen választásával kapott módszereket *Fibonacci-hash-elésnek* nevezzük.

A kettős hash-elés második függvénye

A kettős hash-elésnél olyan h' függvényre van szükség, melynek értékei a $[0, M - 1]$ intervallumba esnek, és relatív prímek az M táblamérethez. Ha M prím, akkor a következő függvény megteszi:

$$h'(K) := K \pmod{M - 1} + 1.$$

Az első tag 0 és $M - 2$ közé eső egész, ezért a h' értékei 1 és $M - 1$ között lesznek, sőt az értékkészlete éppen az $[1, M - 1]$ intervallum egészeiből áll. Ebből két hasznos dolog következik. Egyrészt az, hogy $h'(K)$ és M relatív prímek. Másrészt az is teljesül, hogy elég sok – szám szerint $M - 1$ – különböző próbasorozatot ad.

4.3. Hash-elés kontra keresőfák

Két erőteljes és gazdag módszercsaládot ismertünk meg a keresés, beszúrás, törlés és az ezekre épülő műveletek megvalósítására; divatos kifejezéssel élve: dinamikus halmazok kezelésére. Az egyik megközelítés a hash-elt szervezés, a másik pedig a keresőfa adatszerkezet. Természetesen vetődik fel a kérdés, hogy milyen szempontok szerint válasszunk a kettő közül. Ez felmerülhet úgy, hogy nekünk kell egy dinamikus halmazokat kezelő programot írni, de úgy is, hogy egy készen kapható rendszer (pl. adatbáziskezelő) által nyújtott lehetőségek közül kell választani. Három szempontot említünk, amelyek segítenek az eligazodásban.

Átlagos viselkedés – legrosszabb eset
A hash-elt szervezésnél az alapműveletek (keresés, beszúrás, törlés) átlagos értelemben gyorsabbak, mint a keresőfáknál. Ezt tükrözi az a tény, hogy a hash-elésnél rögzített $\alpha < 1$ telítettségi tényezőnél a megvizsgált lapok, illetve cellák száma átlagosan konstans korlát alatt marad. Számos alkalmazásnál éppen az átlagos sebesség, ami igazán számít. Például az ügyviteli alkalmazások egy jelentős részénél csak az a fontos, hogy nagy mennyiségű teendőt összességében kedvező idő alatt elvégezzünk, és nem érdekes, ha ezek közül néhány hosszabb ideig tart, mint a többi. Az ilyen esetekben a hash-elés felé billen a mérleg nyelve.

Vannak ezzel szemben olyan helyzetek, ahol a legrosszabb, legkedvezőtlenebb esetekben is elég gyorsnak kell lenni. A legdrámaibb példák ebből a szempontból az ún. *valós idejű* feladatok. Ezek jellemzője, hogy a programnak meg kell felelnie valamilyen külső folyamatok feszített időbeli kényszereinek. Mondjuk egy erőmű kazánjának szelepeit vezérlő rendszertől elvárjuk, hogy a megnövekedett hőmérsékletre még azelőtt reagáljon, nyissa ki a szelepeket, mielőtt szétrobban a kazán. Groteszk lenne a katasztrófa után azzal takarózni, hogy a program éppen egy hosszú vödörben kotorászott valami adat után. Az ilyen körülmények között adódó dinamikus halmazokat keresőfákban kell tárolnunk, mert azok a legrosszabb esetek idejére is garanciát nyújtanak.

Rendezés a felhasználói igényekben
Előfordul, hogy az adatok rendezésére vonatkozó feltételek komoly szerepet játszanak a felhasználói igényekben, például sok MIN, MAX, és/vagy TÓLIG-típusú kérdésre kell válaszolni. Ekkor a keresőfák ajánlhatók, hiszen jó megoldásokat nyújtanak az ilyen feladatokra. A hash-elt szervezések nem, vagy csak meglehetősen bonyolult kiegészítő megoldásokkal tudják figyelembe venni a rendezettséget.

Dinamika
A hash-elt szervezés nem támogatja az állomány nagyon dinamikus növekedését. Ha a tábla betelik, vagy ha α már túl nagy, akkor új, nagyobb táblával (vödörkatalógussal) újra kell szervezni az állományt. A keresőfák ezzel szemben rendkívül rugalmasan, tág határok között tudják változtatni a méretüket. Ha tehát úgy látjuk, hogy a tárolandó állomány mérete erőteljesen változni fog, vagy nem tudunk a méretre realisztikus felső korlátot adni, akkor inkább a keresőfák ajánlhatók.

4.4. Szekvenciális keresés

Gyakran előfordul, hogy egy állomány (tömb, lista, stb.) szegényes szerkezetű. Ez alatt azt értjük, hogy nincs benne elég rendszer, és ezért a keresésre nincs jobb módszerünk, mint a szerkezetet „elejétől a végéig" bejárni, vagy legalábbis addig, amíg a keresett adatot meg nem találjuk. Az ilyen esetekben *szekvenciális keresésről* beszélünk. A modell, amit használunk, a *rekordokból álló tábla.*

R_1	R_2	R_3	...	R_{N-1}	R_N

A feladat az R rekord megtalálása a táblában, ha egyáltalán benne van. A keresés úgy történik, hogy a tábla elejétől kezdve sorra összehasonlítjuk az R rekordot az R_i rekordokkal ($i = 1, 2, \ldots$). A keresés befejeződik, ha $R = R_i$, vagy $i = N$. A keresés költsége a tábla megvizsgált celláinak száma.

Ha semmiféle további feltevéssel nem élünk, akkor a sikeres keresés átlagos költsége

$$\frac{1 + 2 + \cdots + N}{N} = \frac{N+1}{2}.$$

Az átlag úgy értendő, hogy az $R = R_i$ események egyenlő esélyűek. Sikertelen kereséskor, amikor az R rekord nincs a táblában, a költség mindig N.

Ha a tábla a rekordokat egyértelműen meghatározó kulcs szerint rendezett, akkor sikeres keresésnél a helyzet változatlan, $\frac{N+1}{2}$ az átlagos költség. A sikertelen keresés időigénye csökkenhet, hiszen a táblát csak akkor kell végignézni, ha R kulcsa nagyobb az R_N kulcsánál. Beszélhetünk átlagos költségről is: a beszúrásos rendezés elemzésében használt feltételhez hasonlóan tegyük fel, hogy az R egyenlő eséllyel kerül az R_i rekordok által meghatározott $N + 1$ rendezési intervallum bármelyikébe. Ekkor a megvizsgált cellák átlagos száma

$$\frac{1 + 2 + \cdots + N - 1 + N + N}{N+1} = \frac{N}{2} + \frac{N}{N+1} < \frac{N}{2} + 1.$$

Tegyük fel a továbbiakban, hogy csak sikeres keresésekkel van dolgunk, és legyen p_i annak a valószínűsége, hogy az R_i rekordot keressük. Ekkor $p_1 + p_2 + \cdots + p_N = 1$, és a keresés várható költsége

$$C_N = p_1 + 2p_2 + 3p_3 + \cdots + Np_N.$$

Feladat: Mutassuk meg, hogy adott valószínűségek mellett C_N akkor minimális, ha $p_1 \geq p_2 \geq \ldots \geq p_N$. (Ha $i < j$ és $p_i < p_j$, akkor az $R_i \leftrightarrow R_j$ csere után C_N kisebb lesz.)

A feladat szerint a rekordokat a keresési gyakoriságok szerint nem növekvő sorrendben érdemes a táblában tartani. Előfordul, hogy a $\{p_i\}$ eloszlást, vagy annak elég jó közelítését ismerjük, és ezért ki tudjuk alakítani az optimális sorrendet. Olyan alkalmazásoknál fordul ez elő, ahol a tábla statikus, a tartalma nem változik, és kizárólag keresési igények merülnek fel. Ilyen alkalmazásra elég jó példa egy archív szövegeket (mondjuk régi folyóiratok tartalmát) tároló rendszer. Nézzük meg ezután, hogy miként alakul a C_N értéke néhány érdekes eloszlás esetén, amikor a rekordok a p_i értékek szerint nem növekvő sorrendben vannak a táblában.

Egyenletes eloszlás: Itt $p_i = \frac{1}{N}$ minden i-re. Ekkor, ahogy azt az előbb már láttuk, $C_N = \frac{N+1}{2}$. Átlagosan tehát a tábla celláinak a felét kell megnézni.

Egy nagyon ferde eloszlás: Legyen $p_i = \frac{1}{2^i}$ ha $1 \leq i \leq N-1$, és $p_N = \frac{1}{2^{N-1}}$. A kupacépítés elemzésénél használt háromszög alakú táblázat összegezésével belátható, hogy C_N értéke nagyon kicsi: $C_N = 2 - \frac{1}{2^{N-1}}$. Ez nem meglepő, hiszen a p_i sorozat rohamosan csökken.

Zipf eloszlás: Jelölje H_n az n-edik harmonikus számot:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

Ahogy azt a gyorsrendezés elemzésénél már említettük, $H_n = \log n + \gamma + o(1)$, ahol $\gamma = 0.55721\ldots$ az Euler-állandó. A Zipf eloszlás valószínűségei $p_i = \frac{c}{i}$, ahol $c = \frac{1}{H_N}$. A Zipf eloszlás előfordul több, az adatkezelés szempontjából érdekes összefüggésben. Közelítőleg ilyen eloszlást követ például egy átlagos (angol nyelvű) szövegben az i-edik leggyakoribb szó gyakorisága. Egyszerűen adódik, hogy

$$C_N = \sum_{i=1}^{N} ip_i = \sum_{i=1}^{N} i\frac{c}{i} = Nc = \frac{N}{H_N}.$$

A keresés várható költsége tehát $N/\log N$ körül van.

80-20 szabály: Arról a számítógépes gyakorlatban megfigyelt, sok esetben érvényes jelenségről van szó, hogy az adatelérési igények 80%-a a rekordoknak körülbelül csak 20%-át érinti. Erre a 20%-ra ugyanez a szabály érvényes. Ebből következik, hogy az igények 64%-a az állománynak mindössze 4%-át érinti.

A 80-20 szabály szerint viselkedő eloszlásra minden $1 \leq m \leq N$ esetén

$$\frac{p_1 + p_2 + \cdots + p_{0,2m}}{p_1 + \ldots + p_m} \approx 0,8.$$

E követelményeknek megközelítően eleget tevő eloszlást kapunk a $p_i = \frac{c}{i^{1-\vartheta}}$ választással, ahol

$$\vartheta = \frac{\log 0,8}{\log 0,2} \approx \frac{1}{7}, \quad c = \frac{1}{H_N^{(1-\vartheta)}} \quad \text{és} \quad H_N^{(1-\vartheta)} = 1 + \frac{1}{2^{(1-\vartheta)}} + \cdots + \frac{1}{N^{(1-\vartheta)}}.$$

A sikeres keresés várható idejére megmutatható, hogy $C_N \approx 0,122N$. Egy ilyen eloszlást követő állományban az optimális sorrend mellett a keresés mintegy négyszer gyorsabb, mint a rekordok (egyenletes eloszlás szerinti) véletlen sorrendje esetén.

Önszervező módszerek

Mit tehetünk abban az esetben, ha a p_i keresési valószínűségeket nem ismerjük, vagy esetleg azok idővel változnak? Ilyenkor a sikeres keresés után érdemes változtatni a rekordok sorrendjén. Az S-fáknál már láttunk példát arra, hogy visszacsatoláson alapuló egyszerű változtatások jelentősen javíthatják egy adatszerkezet hatékonyságát. A jelen esetben a keresések sorozatát úgy is felfoghatjuk, hogy mintát veszünk a $\{p_i\}$ eloszlásból, vagyis nézzük az $R = R_i$ események q_i tapasztalati gyakoriságát. A q_i gyakoriság nő, ha az éppen megtalált rekord R_i, ezért ekkor R_i-t előbbre visszük a táblában, feltéve, hogy $i > 1$. Két alapvető ötletet említünk.

(a) A keresett (és megtalált) R_i elemet a tábla elejére visszük. Az eredmény:

R_i	R_1	R_2	...		R_N

(b) A keresett (és megtalált) R_i elemet felcseréljük a megelőzővel.

R_1	...	R_i	R_{i-1}	...		R_N

Ha az eloszlás időben változik, akkor az (a) megoldás ajánlatos. Ha a $\{p_i\}$ eloszlás stabil, azaz időben nem változik, akkor a (b) heurisztika eredményesebb. A (b) változatról azt sejtik, hogy optimális megoldást ad az olyan esetekben, amikor egy ismeretlen statikus eloszlással van dolgunk, és nem áll módunkban nyilvántartani a q_i elérési gyakoriságokat.