Eötvös Loránd Tudományegyetem
Természettudományi Kar

Univerzális hasítás

Szakdolgozat

Kézér Tamás Gábor

Matematika BSc, Alkalmazott matematikus szakirány

Témavezető: Fekete István, egyetemi docens

ELTE Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Budapest, 2011

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg. (A támogatás száma TÁMOP 4.2.1./B-09/1/KMR-2010-0003.)

Tartalomjegyzék

1. fejezet

Bevezetés

Életünk szinte minden területén jelen vannak különböző adathalmazok, még ha sokszor észre sem vesszük őket és nem gondolunk bele, hogy milyen fontos is a minél jobb adatkezelés. Pedig a számológép, a hűtő, az autóbusz, a laptop vagy az űrrakéta működéséhez egyaránt elengedhetetlen, hogy a fontos - mért, kapott vagy számolt - adatokat el tudják tárolni, illetve szükség esetén tudjanak a kapott adatszerkezetben keresni, törölni vagy új elemet beszúrni. A dinamikus adatkezelés tehát kiemelten fontos feladat, így nem meglepő, hogy megvalósítására többféle módszer is létezik. Természetesnek tűnő elvárás egy adatkezelőtől, hogy bármely adathalmaz bármely elemét gyorsan és hatékonyan szúrja be, keresse meg, vagy törölje az adatszerkezetből. Ugyanakkor még a legjobb adatkezelők sem képesek ezt a látszólag alapvető feltételt teljesíteni. Szakdolgozatomban rámutatok ennek a jelenségnek az okára, illetve arra, hogy mi az, amit reálisan elvárhatunk egy jó adatkezelőtől.

A második fejezet elején bemutatásra kerül az egyik legelterjedtebben használt adatkezelő megoldás, a hasításos technika. Megismerkedünk az alapokkal, majd a két legjellemzőbb megvalósítással, a láncolással illetve a nyílt címzéssel. Utóbbi esetében sorra veszünk három lényegesen különböző eljárást, a lineáris kipróbálást, a kvadratikus maradékpróbát illetve a kettős hasítást, majd összehasonlítjuk őket. A láncolás esetében egy fontos lemma bizonyítása révén belátjuk, hogy bizonyos feltételek mellett a hasítás segítségével mindhárom szótárművelet $O(1)$, vagyis konstans időben végrehajtható. A második fejezet végén röviden kitérünk a hasításnak a másik igen elterjedt és jól használható dinamikus adatkezelési megoldással, a bináris keresőfával való összehasonlítására. Látni fogjuk, hogy a két módszernek más-más előnyei és hátrányai vannak, így a feladat jellegétől függ, hogy melyiket érdemes használni.

A "közönséges" hasítási technikák komoly hátránya, hogy bizonyos adathalmazokra kifejezetten rosszul viselkednek. Ugyanakkor mi olyan eljárást szeretnénk,

melynek nincsenek "gyenge pontjai", "rossz adathalmazai". Egy nagyon érdekes ötleten alapszik a szakdolgozatom második felében részletezett módszer, az univerzális hasítás, mely pont azt tudja, amit elvárunk tőle: nagy valószínűséggel bármely inputra hatékonyan működik. Nagy előnye, hogy készakarva sem lehet rossz adathalmazzal lassú futásra kényszeríteni. Az elméleti áttekintés után saját programokkal illusztrálok két fontos eredményt, melyeknek a bizonyítása is szerepel jelen dolgozatban. A konkrét példák egyfelől szemléltetik majd az eredményeket, másfelől alá is támasztják majd azokat. Azt is látni fogjuk, hogy az univerzális hasítás hasítófüggvény-halmazának ügyes megtervezésével lényegében elhanyagolhatóan kevés rossz futást fogunk kapni.

2. fejezet

Hasítás

2.1. A hasításos technika

A hasítás a dinamikus adatkezelés egy igen elterjedt módszere. A tárolandó adatok az adatrekordok, melyeknek egyedi azonosítójelük van. Ezt a jelet nevezzük az adatrekord **kulcsának**, erre hivatkozva kereshetőek, törölhetőek, stb. az adatok. A kulcsok olyan azonosítói az adatrekordoknak, mint amilyen megkülönböztető jele a rendszám a motorizált járműveknek. Mivel az adatkezelés folyamatában indifferens az adatok mérete és tartalma, vagyis csak a kulcsuk számít, elég a kulcsokkal dolgoznunk a következőkben. A továbbiakban feltesszük, hogy minden rekord kulcsa nemnegatív egész szám. Ez a megkötés lényegében nem korlátozza a kezelhető esetek számát, ugyanis a legtöbb előforduló kulcsot könnyedén egész számmá transzformálhatjuk. Vegyük például a {rab, **, @!} kulcshalmazt. A kulcsokat ASCII kódjuk alapján 128-as számrendszerben ábrázolva a $\{114 \cdot 128^2 + 97 \cdot 128 + 98, 42 \cdot 128 + 42, 64 \cdot 128 + 33\}$ halmazt kapjuk, mivel az r, a, b, *, @, ! karakterek ASCII kódja rendre 114, 97, 98, 42, 64 illetve 33. Vagyis a fenti három kulcsnak az {1 880 290, 5418, 8225} kulcshalmazt feleltettük meg. Módszerünk garantálja, hogy bármely két különböző karaktersorozatnak különböző kódja lesz, ha csupa ASCII karaktert használunk. Persze előfordulhat, hogy így igen nagy számokat kapunk, melyeknek a kezelése nehézkes, de összességében ez nem okoz jelentős romlást a hatékonyságban, így ennek a kérdésnek a vizsgálatával nem foglalkozunk. Mostantól tehát csupa nemnegatív egész kulcsértékkel dolgozunk.
Mivel az adatrekordok keresése/törlése/beszúrása vagy módosítása helyett elegendő a kulcsaik megfelelő kezelése, a dinamikus adatkezelés az alábbi három szótárművelet előre meghatározott sorrendű végrehajtását jelenti:

Keresés(T, k)**:** a k kulcs keresése a T adatstruktúrában

Beszúrás(T, k)**:** a k kulcs beszúrása a T adatstruktúrába

Törlés(T, k)**:** a k kulcs törlése a T adatstruktúrából

Célunk egy olyan dinamikus adatstruktúra megtalálása, melynek segítségével a fenti három művelet hatékonyan végrehajtható. Legyen T olyan adatstruktúra (memória), melyben direkt elérésű minden memóriacella. Ez azt jelenti, hogy T bármely eleme $O(1)$ időben elérhető (le)kérés esetén. A memóriacellákat **slotoknak**, más néven réseknek, T-t **hasítótáblának** (más néven direkt (vagy közvelten) elérésű táblázatnak) szokás nevezni. A T tábla méretét $|T|$-vel jelöljük és értéke T slotjainak száma, mely legyen mostantól m. Vagyis $|T| = m$ valamely m pozitív egész számra. Az általánosság megszorítása nélkül feltehetjük, hogy

$$T = \{0, 1, \ldots, m-1\},$$

ahol $0, 1, \ldots, m-1$ rendre az egyes slotok sorszámát adja. Legyen U a lehetséges kulcsok halmaza, melyet **kulcsuniverzumnak**, vagy röviden **univerzumnak** nevezünk. Általában $|U| \gg |T|$, vagyis lehetetlen a teljes U univerzumot T-ben elhelyezni. Sőt, U sokszor szinte kezelhetetlenül nagy. Mivel azonban a gyakorlati problémákban jellemzően nem fordul elő egyszerre az összes lehetséges kulcs, csak U egy részhalmaza, elég, ha az aktuálisan előforduló adatrekordok kulcsainak S halmazával foglalkozunk. Amennyiben például egy nagy sportrendezvényre érkező nézők gépjárműveiről szeretnénk egy nyilvántartást készíteni, akkor elég az autók rengeteg jellemzője (szín, méret, típus, fogyasztás, gyártási év, tulajdonos, stb.) helyett csak az autó, mint adatrekord kulcsát, vagyis a rendszámot tárolni. (Ugyanis a rendszám alapján már kideríthető kis utánajárással a többi adat, illetve például adott autót keresve az azonosítás a rendszám alapján elvégezhető, még ha némiképp nehézkesebb is egy nagy parkolóban a PXK - 473 rendszámú jármű megtalálása csak a rendszám alapján, mint akkor, ha azt is tudjuk, hogy egy piros 20 tonnás kamionról van szó.) Ha a rendezvénynek helyt adó sportcsarnok vagy stadion befogadóképessége 20 000 fő, akkor minden bizonnyal elegendő 10 000 rendszám tárolását terveznünk, ennél több autó (és motor) szinte biztosan nem fog érkezni. Tehát a konkrét esetünkben előforduló kulcsok halmaza $|S| = 10\ 000$ elemű, ezeket kell eltárolnunk, szemben azzal, hogy az összes lehetséges kulcsok U halmaza $26^3 \cdot 10^3 = 17\ 576\ 000$ kulcsból áll.

A továbbiakban U-t a természetes számok véges részhalmazának tekintjük, vagyis általában

$$U = \{0, 1, 2, \ldots, d\}$$

valamely pozitív egész d-re, illetve néha $U = \mathbb{N}$. Ezt a fentebb ismertetett okok miatt tehetjük meg az általánosság megszorítása nélkül. A feladatunk tehát S elemeit eltárolni T-ben. Általában nincs befolyásunk S kiválasztására, sőt, legtöbbször semmit sem tudunk az eltárolandó kulcshalmazról. Az S halmaz ugyanis az adott feladattól függ, mint ezt az előbbi példában is láttuk. Nekünk, mint az adatkezelést megvalósító algoritmus (program) tervezőinek, az a célunk, hogy az n elemű S halmazt "jól szétszórva" tároljuk el T-ben. (A "jól szétszórt" helyett a továbbiakban az egyenletes kifejezést fogjuk használni.) Ez azért lényeges, mert a fentebb említett szótárműveletek várhatóan jóval gyorsabbak, ha egyenletesen tároltuk el a kulcsokat T-ben, mint ha összevissza helyezkednek el. A későbbiekben látni fogjuk, hogy az egyenletességen a két jellemző megvalósítás - a láncolt listás és a nyílt címzéses ütközésfeloldás - esetében pontosan mit is értünk. Általánosan is igaz mindenesetre, hogy szeretnénk, ha minden S-beli kulcs egyenlő valószínűséggel képeződne le bármely slotra, függetlenül attól, hogy a korábbi ill. későbbi kulcsok hova képeződtek le ill. hova fognak leképeződni. Ezt a tulajdonságot egyszerű egyenletességi feltételnek nevezzük. A h U-ból T-be képező függvényt **hasítófüggvénynek** nevezzük. A továbbiakban elvárjuk, hogy az előbb megfogalmazott feltételt egy "jó" hasítófüggvény kielégítse.

A gyakorlatban két elterjedt módszert használnak "jó" hasítófüggvények előállítására, az osztómódszert és a szorzómódszert. Az osztómódszer esetében a hasítófüggvény $h(k) := k \bmod m$, a legjobb működést pedig akkor érhetjük el, ha a hasítótábla m méretét prímnek olyan választjuk, mely nem osztja $r^k \pm a$-t, ahol r a karakterkészlet elemszáma (például 128) és a, k "kis" egészek (*D. E. Knuth* javaslata). A szorzómódszer hasítófüggvénye $h(k) := \lfloor m \cdot \{\beta k\} \rfloor$, ahol β valós szám. A módszer egyfajta álvéletlenséget használ, ugyanis a fenti függvény a βk törtrészének kiszámításával a k kulcsértéket szinte véletlenszerűen hasítja a $[0, 1)$ intervallumba. Mindkét módszer jó eredményeket ad számtani sorozatot alkotó kulcshalmazok esetén. A gyakorlatban sűrűn adódnak ilyen inputok (például a {hallgató1, hallgató2, hallgató3, ...} halmaz hallgatói adatbázisokban). A szorzómódszer használatával a $h(k)$, $h(k+d)$, $h(k+2d)$, ... sorozat közelítőleg számtani sorozat lesz, vagyis h közel egyenletesen hasítja a kulcsok számtani sorozatait. Ez az eredmény könnyen adódik a következő igen szép tételből, mely T. Sós Vera nevéhez fűződik.

1. Tétel. *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.*

Mivel U és T adott, S pedig kezdetben ismeretlen előttünk, mindössze egyféleképpen tudjuk befolyásolni a hasítás (várható) eredményét. Ez pedig a hasítófüggvény megválasztása. De akármilyen h hasítófüggvényt is választunk, sokszor nem tudjuk elkerülni azt, hogy T egyes slotjaira egynél több kulcsot képezzünk. Ennek egyrészt az az oka, hogy nem ismerjük előre a hasítandó S kulcshalmazt, csak az egész kulcsuniverzumot, mely sokkal nagyobb általában, mint $|T|$, így nem adható meg olyan h, mely tetszőleges $S \in U$ esetén minden $x \in S$ kulcsot különböző slotra hasít. Másfelől pedig sokszor már önmagában az S elhelyezendő halmaznak is több eleme van, mint amekkora a T hasítótábla mérete. Így tehát mindenképpen foglalkoznunk kell az úgynevezett ütközésekkel. Ha az $x, y \in S$ kulcsokra $h(x) = h(y) = j$, vagyis h a T hasítótábla ugyanazon j slotjára hasítja mindkettőt, akkor azt mondjuk, hogy x és y **ütköznek** j-ben. A hasítási módszereket két fő csoportba szokták sorolni attól függően, hogy az ütközések kezelését hogyan oldják meg. A továbbiakban ezt a két módszert ismertetjük.

2.2. Láncolt lista

A láncolt listás megvalósítás során is szükséges megadnunk egy megfelelő

$$h : U \to T = \{0, 1, \ldots, m-1\}$$

hozzárendelést, mely a kulcsok elosztását megvalósítja. A kulcsok eloszlását a láncolt listás esetben akkor tartjuk jónak (egyenletesnek), ha S elemei egyenletesen vannak elosztva T slotjai között, vagyis ideális esetben minden slotra körülbelül

$$\frac{|S|}{|T|} = \frac{n}{m}$$

kulcs jut S-ből. Ha T egy b slotjára $l \geq 2$ db S-beli kulcs kerül, akkor készítünk egy láncolt listát, mely az l db kulcsot (adatrekordot) tartalmazza, majd egy pointert irányítunk a b slotból a listára, hogy az elemek elérhetőek legyenek a műveletek során. Ezen megoldás következményeként a keresés maximum l lépést igényel, a várható lépések száma pedig $l/2$. Természetesen a maximálisan egy slothoz rendelt kulcsok száma kritikus a Keresés(T, k), Beszúrás(T, k) és a Törlés(T, k) szótárműveletek műveletigénye szempontjából. Értelemszerűen a célunk az lenne, hogy ezt a maximumot minél lejjebb szorítsuk, ám az előbb már sejtettük, hogy ez nem megy, így arra törekszünk, hogy várható értékben minél jobb eredményt érjünk el. A legnagyobb problémánk ezzel kapcsolatban az, hogy (mint láttuk) h-t nem definiálhatjuk S-ből T-be rendelő függvénynek, mivel S a konkrét feladat függvénye és nekünk h-t minden lehetséges S-re egységesen, előre meg kell adnunk. Ugyanis nem elég,

hogy S ismeretlen előttünk, ráadásképpen még változhat is az adatkezelés során a beszúrások és törlések eredményeképp. Mit lehet ezek után tenni, lehet-e megfelelő h függvényt találni ennyi bizonytalanság mellett is?

Elvárásokat támasztunk tehát a h hasítófüggvénnyel szemben annak érdekében, hogy a hasítást minél sikeresebbnek értékelhessük. Ezek az (ésszerű) elvárások a következők:
(i) h legyen hatékonyan programozható
(ii) h a legtöbb $S \subseteq T$ adathalmazt úgy hasítsa a T hasítótáblára, hogy minden $i \in \{0, 1, \ldots, m-1\}$ esetén a

$$T(i) = \{x \in S \mid h(x) = i\}$$

halmaz számossága $O(|S|/|T|)$-ben van.
Sajnos a (ii)-ben megfogalmazott elvárás nem szigorítható olyan módon, hogy elvárjuk h-tól, hogy minden $S \in U$ kulcshalmazt egyenletesen hasítson T-be. Ugyanis minden h hasítófüggvényhez és T minden i slotjához létezik az alábbi

$$U_{h,i} = \{x \in U \mid h(x) = i\}$$

kulcshalmaz, melynek minden elemét éppen T i-edik slotjára képezi h. Így tehát bármely $S \in U_{h,i}$ kulcshalmaz esetén, vagy olyan esetben, mikor az S halmaz nagyrészt $U_{h,i}$ elemeiből áll, a h hasítófüggvény nem hasítja jól S-et T-be (sőt, kifejezetten rosszul hasítja), szinte a lehető legrosszabb eredményt produkálja. Ilyenkor például a keresés műveletigénye $O(\mathrm{n})$ lenne. Ez nyilvánvalóan elrontaná a törekvéseinket, így annak érdekében, hogy (ii)-t teljesítsük, olyan hasítófüggvényre van szükségünk, mely egyenletesen hasítja U összes kulcsát T-be. Így tehát meg kell követelnünk azt, hogy minden $i \in \{0, 1, \ldots, m-1\}$-re

$$P(h(z) = i) \quad = \quad \frac{1}{m} \tag{2.1}$$

legyen minden véletlenszerűen választott $z \in U$ elem esetén. Ilyen hasítófüggvény létezik, vegyük például azt a $h_m : \; U \to T$ függvényt, melyet a $h_m(z) = z \bmod m$ képlet definiál. Most megmutatjuk, hogy a (2.1) teljesülése esetén tetszőleges kulcshalmazt várhatóan egyenletesen hasít a hasítófüggvény T slotjaira.

1. Lemma. *Legyen $U = \mathbb{N}$ az univerzum és legyen $T = \{0, 1, \ldots, m-1\}$, $m \in \mathbb{N} - \{0, 1\}$. Legyen n pozitív egész, h pedig $U \mapsto T$ egy olyan hash függvény, mely azonos valószínűséggel hasít egy véletlenszerűen választott kulcsot minden slotra, vagyis teljesíti a (2.1)-et. Ekkor T minden egyes slotjára teljesül, hogy:*

(i) az l. slotra hasított $S \in P_n(U)$-beli rekordok várható száma (vagyis azon S-beli x rekordok száma, melyekre $h(x) = l$) kisebb, mint

$$\frac{n}{m} + 1$$

(ii) és $n = m$ esetén P(egynél több S-beli kulcs kerül az l. slotra) $< \frac{1}{2}$.

Bizonyítás. Tekintsük a $(P_n(U), \mathrm{P})$ valószínűségi mezőt, ahol P az egyenletes valószínűségeloszlás $P_n(U) = \{S \subseteq U \mid |S| = n\}$ felett. Válasszunk ki n kulcsot U-ból véletlenszerűen, így egy n elemű $S \subseteq U$ halmazt kapunk. Az $S = \{s_1, s_2, \ldots, s_n\}$ jelölést fogjuk használni, ahol s_k a k-adik véletlenszerűen kiválasztott elem. Tekintsük az alábbi $X_{ij}^l(S)$ valószínűségi változót minden $i, j \in \{1, \ldots, n\}, i < j$, $l \in \{0, 1, \ldots, m-1\}$ esetén:

$$X_{ij}^l(S) = \begin{cases} 1 & \text{ha } h(s_i) = h(s_j) = l \\ 0 & \text{különben} \end{cases}$$

Mivel X_{ij}^l indikátorváltozó, a várható értéke ($\mathrm{E}\left[X_{ij}^l\right]$) annak a valószínűsége, hogy $h(s_i) = h(s_j) = l$ (vagyis annak, hogy h s_i-t és s_j-t egyaránt az l-edik slotra képezi le, így s_i és s_j ütköznek l-ben. Mivel a hasítófüggvénytől megköveteltük, hogy egyenletesen hasítson, így ez a valószínűség az egyes hasítások függetlensége miatt nem más, mint

$$\begin{aligned} \mathrm{P}(h(s_i) = l \ \wedge \ h(s_j) = l) &= \mathrm{P}(h(s_i) = l) \cdot \mathrm{P}(h(s_j) = l) \\ &= \frac{1}{m} \cdot \frac{1}{m} = \frac{1}{m^2}. \end{aligned} \tag{2.2}$$

Most kiszámítjuk az ütközések számának várható értékét minden T-beli l slotra. Az indikátorváltozó

$$X^l = \sum_{1 \le i < j \le n} X_{ij}^l = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}^l$$

az l-edik slotbeli ütközések számát adja meg. A (2.2), valamint a várható érték linearitása alapján

$$\begin{aligned} \mathrm{E}\left[X^l\right] &= \mathrm{E}\left[\sum_{1 \le i < j \le n} X_{ij}^l\right] \\ &= \sum_{1 \le i < j \le n} \mathrm{E}\left[X_{ij}^l\right] \\ &= \sum_{1 \le i < j \le n} \frac{1}{m^2} \\ &= \frac{\binom{n}{2}}{m^2} = \frac{n(n-1)}{2m^2} < \frac{n^2}{2m^2} \end{aligned} \tag{2.3}$$

minden $l \in 0, 1, \ldots, m-1$ esetén. Speciálisan, az $n = m$ esetben azt kapjuk, hogy

$$\mathrm{E}\left[X^l\right] < \frac{1}{2},$$

vagyis annak a valószínűsége, hogy h egynél több kulcsot képez T l-edik slotjára kisebb, mint $\frac{1}{2}$ minden l-re. Ezzel a lemma második állítását beláttuk.

Felhasználva a (2.3) egyenlőtlenséget, megkaphatjuk az l-edik slotra hasított S-beli kulcsok számának várható értékét bármely $n > m$-re. Ha S-nek pontosan k eleme kerül az l-edik slotra, akkor ott pontosan

$$\binom{k}{2} = \frac{k(k-1)}{2}$$

ütközést kapunk. Ekkor válasszuk k-t úgy, hogy

$$\mathrm{E}\left[X^l\right] = \frac{k(k-1)}{2},$$

így felhasználva ismét a (2.3)-t, azt kapjuk, hogy

$$\frac{n^2}{2m^2} > \mathrm{E}\left[X^l\right] = \frac{n(n-1)}{2m^2} = \frac{k(k-1)}{2} > \frac{(k-1)^2}{2}.$$

$$\text{Ebből } k < \frac{n}{m} + 1,$$

így éppen a kívánt egyenlőtlenséget kaptuk. ♣

Mivel az előző lemma alapján az egy slotra hasított kulcsok számának várható értéke kisebb, mint $\frac{n}{m} + 1$, elértük azon kitűzött célunkat, hogy olyan szótárszerkezetet találjunk, mely használata esetén a három lényeges szótárművelet - keresés, beszúrás, törlés - műveletigénye egyaránt $O(\frac{n}{m})$.
Az 1. lemmában feltételeztük, hogy a kulcsuniverzum a természetes számok halmaza. A valóságban azonban a legtöbbször ez nem áll fenn, hiszen csak véges sok potenciális elemet vizsgálunk/rendezünk. Így lényeges megvizsgálnunk azt az esetet, amikor a kulcsuniverzum számossága a hasítótáblázat méretének (nem feltétlenül egész) többszöröse. A véges esetben ugyanis felvetődik egy kisebb probléma. Tegyük fel, hogy az elsőnek kiválasztott kulcsunk x, melyre $h(x) = i$ valamely $i \in T$-re Ekkor a (2.1) teljesülése esetén

$$|\{a \in U - \{x\} \mid h(a) = i\}| < |\{b \in U - \{x\} \mid h(b) = j\}| = \frac{|U|}{m}$$

fennáll minden $j \in \{0, 1, \ldots, m-1\} - \{i\}$-re. Vagyis második kulcsként egy olyan $y \in U - \{x\}$ kulcs választásának valószínűsége, melyre $h(y) = i$ kisebb, mint $\frac{1}{m}$. Ez komoly gondot jelenthet, hiszen ekkor nem teljesül a (2.1) feltétel, ezáltal pedig nem igaz az 1. lemma. Azonban amennyiben $\{|U| \gg |S| > |T|\}$, úgy feltehető, hogy

$\frac{1}{m}$ elfogadhatóan jó közelítése a $\mathrm{P}(h(a) = s \mid a \in U - A)$ valószínűségnek minden $s \in T$ slot és minden $\{A \mid |A| < |S|\}$ halmaz esetén, így problémánk csak látszólagos. Persze akármilyen nagy is U, a második elem kiválasztásának valószínűsége sosem lesz ugyanakkora minden egyes slotra nézve, mint az első elemé (hiszen azon slotra képzés valószínűsége értelemszerűen kisebb, melyhez az első kulcsot rendelte h). Ugyanakkor az 1. lemma nem azt mondja ki, hogy várhatóan egyenletes lesz a slotokra képzett elemek eloszlása, mindössze azt, hogy jól megközelíti azt. Nem célunk véges univerzumra is megfogalmazni és bizonyítani az 1. lemmát, de az előbbi gondolatmenettel jól érzékeltethető, hogy az állítás ilyenkor is igaz lesz.

2.3. Nyílt címzés

A nyílt címzés módszerének használatakor az adatrekordokat magában a hasítótáblában tároljuk. A módszer előnye a láncolással szemben az, hogy mellőzi a pointerek használatát, ezáltal kisebb a memóriaigénye, mint a láncolt listás hasításnak, vagyis általa ugyanakkora memóriaterületen nagyobb méretű hasítótáblát tárolhatunk. Így, ha ugyanazt az S inputhalmazt szeretnénk eltárolni a két módszerrel, a nyílt címzés esetén várhatóan kevesebb ütközést, ezáltal pedig elméletileg gyorsabb adatkezelést kaphatunk. Ugyanakkor kétségtelen hátránya a módszernek, hogy a hasítótáblában legfeljebb $|T|$ db adatrekordot tudunk eltárolni, vagyis csak akkor érdemes a nyílt címzéses hasítást alkalmazni, ha előre (legalább közelítőleg) meg tudjuk mondani, hogy mekkora lesz az S kulcshalmaz és ehhez választhatunk megfelelő méretű T-t. Ennél a módszernél tehát $|S| \leq |T|$-nek mindig teljesülnie kell. Szintén hátrány továbbá az, hogy - mint azt látni fogjuk - az egyes adatrekordok elhelyezése nem (mindig) olyan gyors és egyértelmű, mint a láncolás esetében.
Nézzük meg, hogy az egyes műveleteket hogyan lehet végrehajtani a nyílt címzéses hasítás esetében!
Ha egy adott rekordot keresünk, végignézzük T slotjait, amíg megtaláljuk a keresett elemet, üres mezőt találunk, vagy pedig a táblázat végére nem érünk.
A beszúrás során adott sorrendben végigjárjuk a slotokat és az első üres helyre hasítjuk a beszúrandó rekordot. Ha $|T| = m$ lépés során sem találtunk üres (vagy csillagozott) helyet, akkor a beszúrás nem lehetséges, a táblázat tele van. Az adott sorrend itt azt jelenti, hogy a kipróbált pozíciók sorrendje a beszúrandó rekordtól függ. Így sokkal hatékonyabbak lesznek a műveleteink, mintha egy fix, előre rögzített, minden kulcsra azonos sorrendet használnánk (mely $\Theta(n)$-es műveletigényt jelentene). Pontosabban fogalmazva arról van szó, hogy a k kulcsú adatrekord $h(k)$

hasított értékéhez rendre hozzáadjuk a {0,1,. . . ,m-1} halmaz egy elemét és megköveteljük, hogy ez az elem minden lépésben egy korábban még nem szereplő legyen. Az így nyert m hosszú számsorozatot a k kulcshoz tartozó **kipróbálási sorozatnak**, vagy röviden csak próbasorozatnak nevezzük. A beszúrás során tehát a kipróbálási sorozat elemeit vesszük sorra és megnézzük, hogy a T általuk indukált/mutatott slotjai üresek-e. Ezt addig folytatjuk, amíg üres helyet nem találunk, vagy elérünk a kipróbálási sorozat végére.

Az egyedüli nehézséget a nyílt címzéses hasítótáblából való törlés jelenti, melynek során körültekintőnek kell lennünk. Ugyanis ha például egyszerűen csak törlünk a tábla i-edik slotjából egy sikeresen megkeresett elemet és a helye ezáltal üres lesz, akkor utána sikertelen lesz egy olyan táblabeli elem keresése, melynek beszúrása során a próbasorozat érintette, ám foglaltnak találta az i-edik slotot.

Az eddigiek szemléltetésére nézzük meg az alábbi példát. Legyen $m = 11$ és alkalmazzuk a $h(k) = k \bmod 11$ hasítófüggvényt. Ütközés esetén lépjünk egyet balra mindaddig, míg üres helyet nem találunk (vagy körbe nem érünk). Az eltárolandó kulcssorozat legyen a következő: 17, 12, 6, 35, 50, 24, 23. A beszúrások után a hasítótáblánk az alábbi:

0	1	2	3	4	5	6	7	8	9	10
24	12	35		50	6	17				23

A felső sorban az egyes slotok sorszáma, alattuk az adott slotra hasított adatrekord (ha van ilyen) kulcsa látható. Most töröljük a táblából a 17-es és a 24-es kulcsokat. Előbbit azonnal megtaláljuk a $h(17) = 6$ hasított érték alapján a $T[6]$ helyen, utóbbi esetében pedig $h(24) = 2$, így $T[2]$, majd $T[1]$ után a $T[0]$ helyen lévő rekordot kell kitörölnünk. Próbáljuk meg most megkeresni a $k = 50$ kulccsal ellátott adatrekordot. Mivel $h(50) = 6$, először a $T[6]$ helyen vizsgálódunk, ám az üres. Ekkor leállunk és közöljük, hogy a keresett elem nem található meg a hasítótáblában, hiszen a hasított értékének megfelelő slot üres, vagyis oda az input egyetlen eleme sem kerülhetett. Következtetésünk persze téves. Látható, hogy a fentiekkel összecsengően a hibát az okozta, hogy a törlés során nem voltunk eléggé körültekintőek. A hasonló problémák elkerülése végett az elemek törlése során NIL érték helyett egy csillag karaktert szoktak a törölt rekord helyére írni. Ezzel a kiegészítő szabállyal tehát a törlés után így festene a hasítótábla:

0	1	2	3	4	5	6	7	8	9	10
*	12	35		50	6	*				23

Ha a keresőalgoritmus csillagot talál, akkor azt olyan üres mezőnek veszi, melyre írni

lehet, ám a keresés során ott leállni nem. Tehát csillag esetén folytatja a keresést az algoritmus és csak akkor áll le, ha a keresett kulcsot vagy valóban üres mezőt talál, illetve ha elérte a próbasorozat végét.

Elméletben olyan próbamódszert keresünk, mely egyenlő valószínűséggel próbálja ki a {0, 1, ..., m-1} halmaz összes permutációját (pontosabban minden esetben ezek közül véletlenszerűen választ egyet). Ez felelne meg a korábbiakban tárgyalt egyenletesség feltételének. A gyakorlatban azonban ilyen hasítófüggvény alkalmazása nehézkes és körülményes, ezért egyszerűbb, közelítő módszerek használatosak. Mi a három leggyakrabban használt eljárást mutatjuk be a kipróbálási sorozatok létrehozására, a lineáris és a négyzetes próbálást, illetve a kettős hasítást. jelöljük n-nel a hasított elemek számát, α-val pedig az $\alpha = \frac{n}{m}$ telítettségi tényezőt. Sikeres keresésnek azt nevezzük, amikor a keresett rekordot megtaláltuk a táblázatban, míg sikertelen keresésen azt értjük, hogy a keresett kulcs nem szerepel T-ben. Legyen C_n a sikeres, C'_n pedig a sikertelen keresések során megvizsgált slotok átlagos száma. Utóbbi esetében feltesszük, hogy $h(k)$ ugyanakkora valószínűséggel lehet a 0, 1, ..., m-1 számok bármelyike.

2.3.1. Lineáris próbálás

A lineáris próbálás során a kipróbálási sorozat a lehető legtermészetesebb: először a hasított érték helyét nézzük meg, majd - amennyiben ez célunknak nem felel meg - rendre eggyel balra lépve folytatjuk a próbálást (úgy, hogy a lista elejéhez érve a végére ugrunk és onnan lépegetünk balra ismét), egészen addig, amíg el nem érjük a célunkat vagy vissza nem érünk a hasított értékhez. Vagyis, ha a k kulcsra a $h : U \mapsto T$ hasítófüggvény a $h(k)$ hasított értéket adja, akkor a $T[h(k)],\ T[h(k)-1], \ldots,\ T[0],\ T[m-1],\ T[m-2],\ \ldots,\ T[h(k)+1]$ kipróbálási sorozatot kapjuk. Itt (az "elvárt" $m!$ helyett) csupán m db próbasorozat fordul elő. Ha egy hasított értéknek megfelelő hely már foglalt, akkor a próba a mellette levő helyet vizsgálja meg, majd a mellette lévő elem mellettit, stb. Az eljárás hátránya, hogy általában nem szórja szét a rekordokat, így úgynevezett elsődleges csomósodás alakul ki. Ennek az az oka, hogy ha egy R_2 rekord beszúrása közben a próbasorozat eléri a már beillesztett R_1 rekord keresési útját, akkor jellemzően végig is járja azt, növelve ezáltal az egymás melletti elemek számát. A fenti példában is a lineáris próbálás módszerét használtuk és ott is megfigyelhetőek a kialakult elsődleges csomók: {23, 24, 12, 35} és {50, 6, 17}.

A lineáris próbálással végzett hasítás esetén a sikeres és a sikertelen keresés műveletigénye az alábbiak szerint alakul:

$$C_n = \frac{1}{2}(1 + \frac{1}{1-\alpha})$$

és

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

2.3.2. Álvéletlen próbálás

Egy speciális esetet mutatunk be, a kvadratikus maradékpróbát. Legyen m egy $4t + 3$ alakú prímszám, ahol $t \in \mathbb{Z}$. Tekintsük a k kulcsra a $h(k) + r_i$ próbasorozatot, ahol $r_i = \pm i^2 \; \forall i \in 0, 1, \ldots, m-1$ esetén. Vagyis a hasított érték után a jobb, majd a baloldali szomszéd, azután a jobbra négy, majd a balra négy mezőre lévő slot következik, stb. Mindezt persze úgy kell végrehajtani, hogy ha elérjük T első vagy utolsó slotját, akkor a hasítótábla másik végétől folytatjuk a számolást. A próbálás során maximum m lépést végzünk, majd, ha nem találtuk meg a keresett elemet (illetve ha nem találtunk üres helyet), az algoritmus leáll. Belátható, hogy ilyen feltételek mellett a kipróbálási sorozat a modulo m maradékosztályok egy permutációját adja, vagyis a próbálás során nincs ismétlődő slot addig, amíg körbe nem érünk. Természetesen így az r_i sorozat is a maradékosztályok egy permutációját adja, mely ráadásul a k kulcstól független. Ez általános az álvéletlen próbálás módszerénél. Az eljárás lényege éppen az, hogy egy gyorsan és hatékonyan számítható kipróbálási sorozatot biztosít, mely ugyanakkor közel véletlenszerűnek hat. Ezen módszernek is komoly hátránya, hogy mindössze m különböző kipróbálási sorozatot képes előállítani. Ugyanakkor az álvéletlen próbálás mentes az elsődleges csomósodás jelenségétől. Ugyanakkor megjelenik az úgynevezett másodlagos csomósodás, melynek lényege az, hogy ha két kulcs hasított értéke eleve megegyezik, akkor a teljes próbasorozatuk is, és így nagy valószínűséggel kerülnek majd egymás közelébe a hasítótáblában. Megjegyezzük, hogy a másodlagos csomósodás természetesen a lineáris próbálás esetében is jelen van, de kevésbé jelentős, mint az elsődleges. Úgy is fogalmazhatunk, hogy az elsődleges csomósodás problémája magában foglalja a másodlagos csomósodást. A műveletigény a legjobb megvalósítások esetében

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

a sikeres keresés esetén, illetve

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

a sikertelen keresés során bejárt slotok átlagos számára.

2.3.3. Kettős hasítás

Az egyik legjobb nyílt címzéses, hatékonyan számítható módszer a kettős hasítás. Az eljárás lényege az, hogy az eredeti h hasítófüggvény mellett egy h' hasítófüggvényt is használunk, melyekkel a próbasorozat i-edik eleme $h(k) + i \cdot h'(k)$ lesz. A jó működéshez szükséges, hogy a $h'(k)$ értékek relatív prímek legyenek az m táblamérettel. Ez megvalósítható például úgy, hogy m-et prímnek választjuk, h' pedig mindig m-nél kisebb egészet állít elő. A korábban ismertetett két módszerrel szemben a kettős hasítás nem m, hanem m^2 különböző próbasorozatot ad, hiszen $h(k)$ és $h'(k)$ értéke egymástól függetlenül egyaránt m-féle lehet. Ezáltal a kettős hasítás várható eredménye lényegesen jobb is, mint a korábbi módszereké. Az eljárás kifinomultsága révén mentes mind az elsődleges, mind a másodlagos csomósodástól. A legjobb ismert implementációk műveletigénye (empirikus adatok alapján)

$$C_n \approx \frac{1}{\alpha} \cdot \log\frac{1}{(1-\alpha)}$$

a sikeres keresés, illetve

$$C'_n \approx \frac{1}{1-\alpha}$$

a sikertelen keresés esetében.

2.3.4. A nyílt címzéses módszerek összehasonlítása

Hasonlítsuk össze a bemutatott módszereket! Az alábbi táblázatban a sikeres (Sik) illetve a sikertelen (NemSik) keresés várható lépésszámait tüntettük fel $\alpha = \frac{1}{2}$, $\alpha = \frac{2}{3}$, $\alpha = 0,9$, valamint $\alpha = 0,99$ mellett. A lineáris próbálás, a kvadratikus maradékpróba és a kettős hasítás módszereket rendre a LP, az AP és a KH indexek jelölik.

α	Sik_{LP}	Sik_{KM}	Sik_{KH}	$NemSik_{LP}$	$NemSik_{KM}$	$NemSik_{KH}$
0,5	1,5	1,44	1,39	2,5	2,19	2
2/3	2	1,77	1,65	5	3,43	3
0,9	5,5	2,85	2,56	50,5	11,4	10
0,99	50,5	5,11	4,65	5000,5	103,62	100

Látható, hogy a lineáris próbálás csak a kis α értékekre versenyképes, $\alpha \geq 2/3$ esetén már lényegesen lassabb, mint a másik két módszer, főleg a sikertelen keresés várható műveletigényét tekintve. Némiképp meglepő módon a kvadratikus maradékmódszer nagyon jó eredményeket ad, mindössze 5-10%-kal lassabb, mint a kettős hasítás, mind a sikeres, mind a sikertelen keresés esetében, bármely α-ra. Utóbbi két módszer még 90%-os táblatelítettség esetén is igen gyors, a sikeres keresés várhatóan legfeljebb 3, a sikertelen 10-11 lépést vesz igénybe. További pozitívum, hogy

sikeres keresésre még a 99%-os telítettség esetén is nagyon jó értékeket produkál mind a kvadratikus maradékpróba, mind a kettős hasítás. Megjegyezzük, hogy a most tárgyalt nyílt címzéses hasítási technikák mindegyike, még a leggyorsabbnak bizonyult kettős hasítás is sokkal lassabb, mint a korábban ismertetett láncolásos megoldás. Ott ugyanis a táblaméretet az input függvényében megválasztva (például a fenti α értékek egyikét előirányozva) várhatóan konstans időben elvégezhető bármely szótárművelet. Ugyanakkor nem szabad elfelejtenünk, hogy a láncolt listás megvalósítás során sokkal nagyobb memóriaigény lép fel, mivel minden láncolt listát el kell tárolnunk a memóriában illetve a pointereket is kezelnünk kell. Összegezve tehát megállapíthatjuk, hogy mindkét megvalósításnak vannak előnyei és hátrányai is és rendszerint a megoldandó feladat természetétől függ, hogy melyiket választjuk.

2.4. Hasítás vagy keresőfák?

Számos jó tulajdonságuk miatt adatkezelésre széles körben használatosak az AVL fák és a B-fák is. Joggal vetődhet fel a kérdés, hogy vajon a hasításos módszerek mennyire versenyképesek a keresőfákkal történő adatkezeléssel szemben? Melyiket érdemesebb használni? Az alábbiakban erre fogunk választ kapni.
A fent említett keresőfa adatstruktúrák segítségével mindhárom szótárművelet végrehajtható

$$\Theta(\log n)$$

időben. Ez az eredmény a legjobb, legrosszabb és a várható esetre egyaránt fennáll. A hasítás bevezetésének egyik célja épp a várható műveletigény $O(1)$-re csökkentése volt. Az 1. lemma segítségével megmutattuk, hogy bármely szótárművelet műveletigénye $O(\frac{n}{m})$. Speciálisan, ha m-et n-nél nagyobbnak választjuk, akkor $O(1)$-et, vagyis konstans műveletigényt kapunk. Így a hasítás az adatkezelés terén egy igen versenyképes alternatíva, mely átlagosan lényegesen gyorsabb, mint a keresőfák. Olyan alkalmazások esetén tehát, melyeknél az átlagos műveleti idő döntő fontosságú, a hasítás alkalmazása kifizetődőbb megoldás. A valóságban a megoldandó feladatok jelentős hányada ilyen, például azok, melyek során sok hasonló részfeladatot kell megoldani egymás után, és csak az számít, hogy összességében minél gyorsabban végezzünk a részfeladatokkal.
Azonban a hasítás esetében a legrosszabb esetre nemhogy az $O(1)$ eredmény nem tartható, de még a $\Theta(\log n)$ sem, mivel sajnos esetenként akár $\Theta(n)$ is lehet a műveleti bonyolultság. Gondoljunk csak arra az esetre, amikor a hasítandó kulcshalmaz mind az n eleme egyazon $T[j]$ slotra képeződik le. Ilyenkor egy szintén a j-edik slotra hasítandó, ám a hasítótáblában nem szereplő k kulcs (sikertelen) keresése $n+1$

lépést igényel, a $h(k)$ érték kiszámítását is figyelembe véve. Ezen körülmény miatt például az úgynevezett *valós idejű* feladatok esetében inkább a keresőfák használata javasolt. Ezen programok jellemzője, hogy a programnak bizonyos külső folyamatok által adott fix időbeli korlátoknak kell megfelelnie minden esetben. Tekintsük például az önmagát vezető autót, melyből már manapság is léteznek kiváló eredményeket produkáló prototípusok. Egy ilyen autótól értelemszerűen elvárjuk, hogy az esetleges vészhelyzetekre azonnal, késlekedés nélkül reagáljon, elkerülve ezzel a balesetet. Vészhelyzetből pedig igen sokféle adódhat, például ha az előttünk haladó jármű váratlanul nagyot fékez, vagy ha egy kereszteződésben nem adják meg nekünk a jogos elsőbbséget. Ha súlyos balesetet szenvedünk, mert az automatika nem fékezett vagy kanyarodott időben, akkor aligha nyugtat meg bennünket a tudat, hogy a program jól működött, csak éppen kifogtuk azt az egy esetet a millióból, amikor az algoritmus egy nagyon hosszú listában keresgélt. Világos, hogy ilyen jellegű problémák esetén sokkal jobb, ha a program kicsit lassabb, de egyenletesen jó sebességgel működik, mintha nagyon gyors, de nagyon ritkán kivételesen lassú.

Szintén a keresőfák adják a jobb eredményt akkor, ha az eltárolt adatok rendezettsége fontos szempont, például ha sokszor szeretnénk lekérdezni a minimális vagy maximális kulcsú elemet. A hasításos technikák ugyanis nem, vagy csak igen körülményes kiegészítő módszerekkel tudják figyelembe venni a rendezettséget, míg a keresőfák kifejezetten jól rendezett tárolást biztosítanak.

Végül megemlítünk még egy szempontot, mely komoly szerepet játszhat a megfelelő eljárás kiválasztásában. A hasítás mindegyik megvalósítás esetén igen érzékeny az input növekedésére. A láncolásos megoldásnál az $\frac{n}{m}$ hányados növekedésével jelentősen romlik a keresési idő, míg a nyílt címzés esetében még az is előfordulhat, hogy az adatokat egész egyszerűen nem tudjuk eltárolni, mert betelt a hasítótábla. Ezzel szemben a keresőfák igen jól kezelik az állomány dinamikus növekedését, így olyankor, ha az input mérete várhatóan jelentősen nőni fog vagy nem tudunk rá realisztikus felső korlátot adni, inkább a keresőfák használata javasolt.

3. fejezet

Univerzális hasítás

3.1. Alapgondolat

Láttuk, hogy
(i) minden h hasítófüggvényhez létezik "rossz" S inputhalmaz (például egy $S \subseteq U_{h,i} = \{a \in U \mid h(a) = i\}$ halmaz), ugyanakkor
(ii) minden olyan hasítófüggvény, mely (2.1)-et kielégíti, az 1. lemma alapján "jól", vagyis várhatóan közel egyenletesen hasítja a véletlenszerű S kulcshalmazt T-re.
Bár a (ii) állítás bíztatóan hangzik, tudomásul kell vennünk, hogy az adatrekordok kulcsainak (vagyis az S halmaz elemeinek) választása korántsem egyenletes eloszlás alapján történő kiválasztás a kulcsuniverzumból. Például, ha egy főként férfiakat foglalkoztató cégnél (pl. futballklubok) az alkalmazottak adatrekordjának egy bitje a nemet írja le, másik 8 pedig a kort, akkor értelemszerűen nem tekinthetjük egyenletesnek a lehetséges kulcsok halmazából történő kiválasztást, hiszen jóval kisebb valószínűséggel foglalkoztat a futballklub 20 db 100 év feletti nőt, mint 20 db 30 alatti férfit... Hasonlóan, ha az ETR kódokból egész számokat képezünk oly módon, hogy minden betűt az angol abc szerint kétjegyű számnak feleltetünk meg, úgy 14 jegyű kulcsokat kapunk (ABECAAT = 01020503010120), ahol érthető módon a 11-gyel kezdődő kódok kiválasztásának valószínűsége messze nagyobb lesz, mint a 25-tel kezdődő kódoké, hiszen K-val kezdődő névből sokkal több van, mint Y-nal kezdődőből és a többi számjegy eloszlása már közelítően egyező a két esetben. A fenti példák azt mutatják, hogy szükséges a hasítási stratégiánk fejlesztése, így ezt tűzzük ki célunknak.
Az (i) és (ii) alapján úgy is tekinthetünk a szituációra, mint egy ellenfelünkkel való játékra. Az ellenfél adja nekünk az S inputot, mi pedig a megfelelő hasítófüggvényt kell biztosítsuk, mely S-et egyenletesen hasítja. A probléma az, hogy azelőtt kell megadnunk a "megfelelő" h-t, hogy ismernénk S-et. Ugyanis S-ről csak annyit tu-

dunk, hogy U-nak egy részhalmaza. Épp ezért a célunk nem lehet más, mint nyerő stratégiát találni, vagyis olyat, mellyel az ellenfél által adott tetszőleges $S \in U$ T-re való hasítása "jó" lesz, legalábbis nagy valószínűséggel. Ennek érdekében egy olyan H halmazt kell keresnünk, melynek elemei hasítófüggvények és melyből véletlenszerűen választva egy h függvényt, az várhatóan jól hasít tetszőleges $S \in U$ kulcshalmazt.

Legyen a valószínűségi mezőnk (H, P_H), ahol H-t úgy kell megválasztanunk, hogy (H, P_H) minden lehetséges S-re megfeleljen. Ennek érdekében a korábbiakban látott (2.1) kritériummal analóg feltevést teszünk H-ra, mely szerint két véletlenszerűen választott U-beli kulcsot (x-et és y-t) ugyanazon slotra legfeljebb $\frac{1}{m}$ valószínűséggel képezünk le. Vagyis bármely $x, y \in U$ $x \neq y$ kulcspár esetén H-ra teljesül, hogy

$$\mathrm{P}_H(h(x) = h(y)) \quad \leq \quad \frac{1}{m}. \tag{3.1}$$

1. Definíció. *Legyen H hasítófüggvények egy véges halmaza U-ból T-be, ahol $T = \{0, 1, \ldots, m-1\}$. A H halmazt univerzálisnak nevezzük, ha minden olyan $x, y \in U$ elempárra, melyre $x \neq y$,*

$$|\{h \in H \mid h(x) = h(y)\}| \leq \frac{|H|}{m}$$

fennáll, vagyis legfeljebb minden m-edik H-beli hasítófüggvény képezi x-et és y-t T ugyanazon slotjára.

Figyeljük meg, hogy hasítófüggvények minden univerzális halmazára teljesül (3.1) a (H, P_H) valószínűségi mezőn. Most megmutatjuk, hogy minden univerzális hasítófüggvény-halmaz eleget tesz a fő célunknak, vagyis minden $S \in U$ esetén egyenletesen hasítja S-et T-re.

3.2. Az univerzális hasítás főtétele

2. Tétel. *Legyen $S \in U$ tetszőlegesen választott kulcsok halmaza, melyre $|S| = n$. Legyen H U-ból T-re képező hasítófüggvények egy univerzális halmaza, ahol $T = \{0, 1, \ldots, m-1\}$. Ekkor minden $x \in S$ és minden véletlenszerűen választott $h \in H$ esetén az $S_x(h) = \{a \in S \mid a \neq x, h(a) = h(x)\}$ halmaz elemeinek várható száma legfeljebb*

$$\frac{|S|}{|T|} = \frac{n}{m}.$$

Bizonyítás. Legyen S tetszőlegesen választott n-elemű részhalmaza U-nak. Tekintsük a (H, P_H) valószínűségi mezőt. Legyen $Z_{x,y}$ indikátorváltozó, melyre minden

$x, y \in S, x \neq y$ esetén

$$Z_{x,y}(h) = \begin{cases} 1 & \text{ha } h(x) = h(y) \\ 0 & \text{ha } h(x) \neq h(y). \end{cases}$$

Mivel $Z_{x,y}$ indikátorváltozó, a várható értéke megegyezik azzal a valószínűséggel, mellyel az 1 értéket felveszi. Így a (3.1) alapján $Z_{x,y}$ várható értékét felülről becsülhetjük az alábbi módon:

$$\mathrm{E}[Z_{x,y}] \quad = \quad = \mathrm{P}_H(h(x) = h(y)) \leq \frac{1}{m}. \tag{3.2}$$

A valószínűségi változó

$$Z_x = \sum_{y \in S, y \neq x} Z_{x,y}$$

megszámolja $S_x(h)$ elemeit minden $x \in S$-re, vagyis azon S-beli x-től különböző kulcsokat, melyeket ugyanazon slotra hasít h, mint x-et.
Felhasználva a várható érték linearitását és a (3.2) becslést, az

$$\mathrm{E}[Z_x] = \sum_{y \in S, y \neq x} \mathrm{E}[Z_{x,y}] \leq \sum_{y \in S, y \neq x} \frac{1}{m} = \frac{|S| - 1}{m} < \frac{n}{m}$$

egyenlőtlenséget kapjuk minden $x \in S$ kulcsra. ♣

Az előbbi tétel alapján megállapíthatjuk, hogy bármely univerzális H hasítófüggvény-halmaz esetén a T egyes slotjaira hasított kulcsok számának várható értéke kisebb, mint $\frac{n}{m} + 1$. Így univerzális hasítófüggvény-halmazok jól használhatóak dinamikus adatkezelés megvalósítására. Az eddigiek során azonban nem tisztáztuk, hogy léteznek-e egyáltalán ilyen halmazok? Vajon a H halmazzal szemben támasztott elvárásaink nem túl nagyok-e? A következő lemmában megmutatjuk, hogy található megfelelő H halmaz.

2. Lemma. *Legyen U véges halmaz és legyen*

$$H_{U,T} = \{h \mid h : U \to T\}.$$

(Tehát $H_{U,T}$ nem más, mint az összes U-ból T-be képező függvény halmaza.)
Ekkor $H_{U,T}$ univerzális.

Bizonyítás. A $H_{U,T}$-beli függvények száma $m^{|U|}$, mivel U minden elemét T bármely (vagyis mind az m) slotjához hozzárendelhetjük. Minden $H_{U,T}$-beli függvény kiválasztásának valószínűsége legyen azonos. Vezessük be a

$$H(x, y, i) = \{h \mid h : U \to T \text{ és } h(x) = h(y) = i\}$$

halmazt minden $x, y \in U, x \neq y$-re és $i \in T$-re. Ha feltesszük, hogy x-et és y-t T i-edik slotjára képezte h, a maradék $|U| - 2$ kulcs hasítását még $m^{|U|-2}$ féleképpen tudjuk megtenni. Azt kapjuk tehát, hogy

$$|H(x, y, i)| = m^{|U|-2}.$$

Célunk meghatározni a

$$H(x, y) = \{h : U \to T \mid h(x) = h(y)\}$$

halmaz számosságát. Mivel

$$H(x, y) = \bigcup_{i=0}^{m-1} H(x, y, i),$$

valamint

$$H(x, y, i) \cap H(x, y, j) = \emptyset \text{ minden } i \neq j, i, j \in T\text{-re, így}$$

$$|H(x, y)| = \sum_{i=0}^{m-1} |H(x, y, i)| = \sum_{i=0}^{m-1} m^{|U|-2} = m^{|U|-1} = \frac{|H_{U,T}|}{m}$$

minden $x, y \in U, x \neq y$ esetén. Tehát az 1. Definíció értelmében $H_{U,T}$ hasítófüggvények univerzális halmaza. ♣

Most már biztosan tudjuk tehát, hogy létezik univerzális halmaz, ám sajnos ez a tény önmagában még nem jelent garanciát a sikeres megvalósításra. Sőt, a valós gyakorlati problémák megoldására sajnos nem is tudjuk (ill. nem célszerű) használni $H_{U,T}$-t. A fő gond $H_{U,T}$-vel az, hogy egyrészt sokkal több eleme van, mint amennyiből még effektíven tudunk véletlenszerű h-t választani, másrészt az egyes $h \in H_{U,T}$ függvények jellemzően teljesen "véletlenszerűen" rendelik U elemeit T slotjaihoz, így leírásuk nehézkes és nehezen kiszámíthatóak. Ha például $U = 10000$ és $m = 10$, akkor $|H_{U,T}| = 10^{10000}$, vagyis elképzelhetetlenül és kezelhetetlenül nagy. Tehát találtunk egy univerzális hasítófüggvény halmazt, mely azonban a gyakorlatban sajnos használhatatlan. Most olyan H univerzális hasítófüggvényt szeretnénk találni, mely már a gyakorlati felhasználás szempontjainak is megfelel. Ennek érdekében két feltételt támasztunk H-val szemben:

(i) H nem túl nagy, azaz $|H|$ felülről korlátozható $|U|$ valamely polinomjával, valamint

(ii) minden egyes $h \in H$ hasítófüggvény nagyon hatékonyan számítható.

3.3. Egy jól használható univerzális hasítófüggvény-halmaz

Legyen $U = \{0, 1, 2, \ldots, p-1\}$ valamely p prímre. Minden $x \in U$-re és minden $a, b \in U$ természetes számpárra legyen a $h_{a,b} : U \to T$ lineáris hasítófüggvény az alábbi módon definiálva:

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m.$$

Mivel modulo m számolunk, $h_{a,b}$ valóban U-ból T-re képez. Legyen

$$H^p_{lin} = \{h_{a,b} \mid a \in \{1, 2, \ldots, p-1\} \text{ és } b \in \{0, 1, \ldots, p-1\}\}$$

hasítófüggvények egy halmaza. Könnyen látható, hogy

$$|H^p_{lin}| = p \cdot (p-1).$$

3. Tétel. *Legyen $U = \{0, 1, \ldots, p-1\}$ és $T = \{0, 1, \ldots, m-1\}$. Bármely p-re a H^p_{lin} hasítófüggvény-halmaz U-ból T-re képező hasítófüggvények univerzális halmaza.*

Bizonyítás. Legyen x és y U két tetszőleges különböző eleme. Azt kell megmutatnunk, hogy

$$|\{h_{a,b} \mid \in H^p_{lin} \text{ és } h_{a,b}(x) = h_{a,b}(y)\}| \leq \frac{H^p_{lin}}{m} = \frac{p \cdot (p-1)}{m}.$$

Vezessük be a

$$h'_{a,b}(x) = (ax + b) \bmod p \in \mathbb{Z}_p$$

lineáris segédfüggvényeket. A $(\mathbb{Z}_p, \oplus_{\text{mod}p}, \odot_{\text{mod}p})$ algebra test, mivel p prím. Legyen

$$r = h'_{a,b}(x) = (ax + b) \bmod p \quad \text{és} \quad s = h'_{a,b}(y) = (ay + b) \bmod p \tag{3.3}$$

a mod p maradék $h'_{a,b}$ alkalmazása után. Most megmutatjuk, hogy $x \neq y \Rightarrow r \neq s$. Tegyük indirekt fel ugyanis, hogy $x \neq y$, de $r = s$. Mivel

$$r - s \equiv ax - ay \quad \equiv \quad a(x - y) \bmod p \tag{3.4}$$

és feltettük, hogy $r = s$, az

$$a(x - y) \equiv 0 \bmod p$$

egyemlőséget kapjuk. A számelmélet alaptétele alapján ekkor

$$a|p \text{ vagy } (x - y)|p.$$

Felhasználva, hogy a 1 és $p-1$ közé eső egész, azt kapjuk, hogy a kettő közül csak $(x-y)|p$ teljesülhet. Mivel azonban mind x, mind y kisebb p-nél (és pozitív), ez csak $x = y$ esetén lehetséges.

Mivel feltettük, hogy x és y különböző, az előbbi állítás értelmében $r \neq s$. Vagyis beláttuk, hogy bármely egymástól különböző fixen rögzített $x, y \in U$ kulcspárhoz egyértelműen létezik egy $r, s \in U \times U$ pár, ahol $r \neq s$. Legyen

$$f_{x,y}(a,b) = ((ax+b) \bmod p, (ay+b) \bmod p) = (h'_{a,b}(x), h'_{a,b}(y)) = (r.s)$$

a függvény, mely a fenti hozzárendelést írja le. Világos, hogy $f_{x,y}$ $(U - \{0\}) \times U$-ból $\{(r,s) \mid r, s \in U, r \neq s\}$-be képez. Szeretnénk megmutatni, hogy ennél több is igaz. Valójában $f_{x,y}$ bijekció, vagyis $f_{x,y}$ és $f_{x,y}^{-1}$ injektív függvények. Ennek belátásához elég megmutatnunk, hogy $f_{x,y}^{-1}$ injektív, mivel

$$|(U - \{0\}) \times U| = (p-1) \cdot p = |\{(r,s) \mid r, s \in U, r \neq s\}|$$

fennáll. Legyen $r, s \in U, r \neq s$. Mivel a $\mathbb{Z}_p$ test felett számolunk, a (3.3) kongruenciaegyenleteknek egyértelmű megoldása van $\mathbb{Z}_p$-ben ismert x, y, r és s értékekre:

$$a = (r-s) \cdot (x-y)^{-1} \bmod p,$$

valamint

$$b = (r - ax) \bmod p,$$

ahol $(x-y)^{-1}$ az $(x-y)$ egyértelmű inverze a $\odot_{\bmod p}$ műveletre $\mathbb{Z}_p$-ben. Így tehát az

$$f_{x,y}^{-1}(r,s) = (a,b) = ((r-s) \cdot (x-y)^{-1} \bmod p, (r-ax) \bmod p)$$

hozzárendelés injektív függvény.
Mivel $h_{a,b}$ véletlen választása (vagyis (a,b) véletlen választása, hiszen minden párhoz pontosan egy h függvény tartozik) egyértelműen meghatározza az

$$(r,s) = (h'_{a,b}(x), h'_{a,b}(y)) = ((ax+b) \bmod p, (ay+b) \bmod p)$$

párt, a hasítófüggvények száma a

$$H_{lin}^p(x,y) = \{h_{a,b} \in H_{lin}^p \mid h_{a,b}(x) = h_{a,b}(y)\}$$

halmazban megegyezik az

$$M(x,y) = \{(r,s) \in U \times U \mid r \neq s \text{ és } r \equiv s (\bmod m)\}$$

halmaz számosságával, vagyis

$$|H_{lin}^p(x,y)| \quad = \quad |M(x,y)|. \tag{3.5}$$

Tudjuk, hogy U elemeinek száma p és ezek maradékosztályokat alkotnak modulo m. Bármely ilyen maradékosztály legfeljebb

$$\left\lceil \frac{p}{m} \right\rceil \leq \frac{p+m-1}{m} = \frac{p-1}{m} + 1$$

elemet tartalmaz. Ez azt jelenti, hogy r minden értékére $s \in U$ legfeljebb

$$\frac{p-1}{m}$$

értéket vehet fel, ahol $r \equiv s(\bmod m)$ és $r \neq s$. Mivel $r \in U$ tetszőleges és $|U| = p$, azt kapjuk, hogy

$$|M(x,y)| \quad \leq \quad \frac{p \cdot (p-1)}{m} \tag{3.6}$$

minden $(x,y) \in U \times U,\ x \neq y$ kulcspárra. Tehát (3.5) és (3.6) alapján fennáll, hogy

$$|H^p_{lin}(x,y)| = |M(x,y)| \leq \frac{p \cdot (p-1)}{m} = \frac{|H^p_{lin}|}{m}$$

minden $(x,y) \in U \times U,\ x \neq y$ kulcspár esetén. ♣

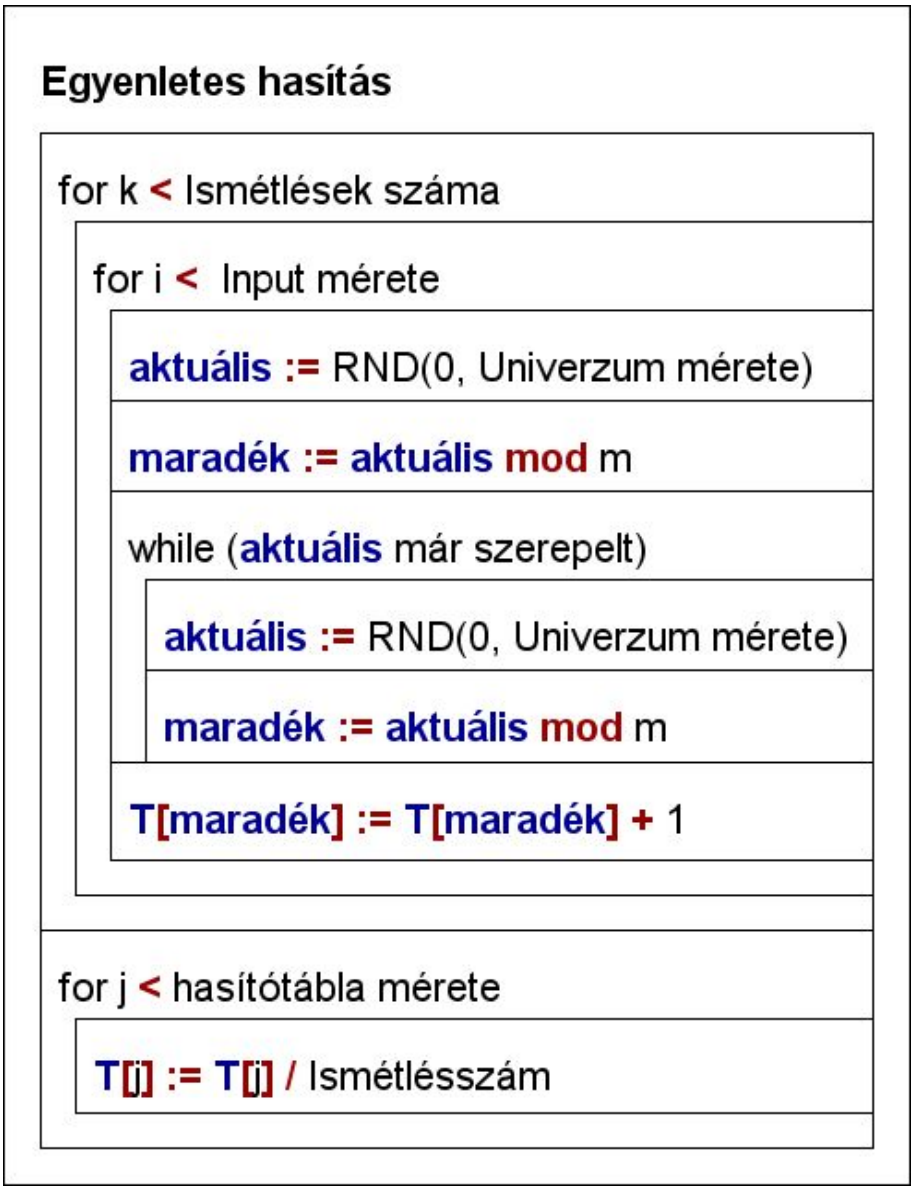
4. fejezet

Az elméleti eredmények tesztelése

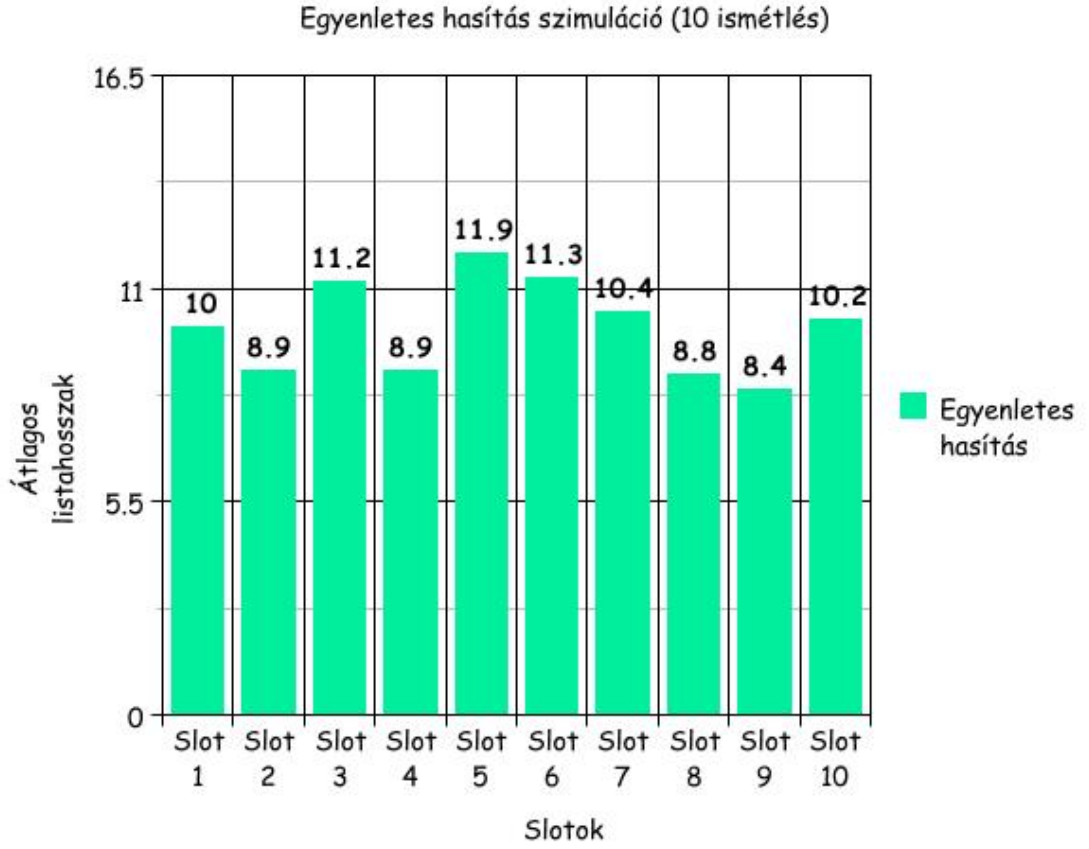
A következőkben általam írt programokkal igyekszem bemutatni azt, hogy a fentebb részletezett eredmények nem csupán elméletben jók, hanem a gyakorlatban is igen jól megállják a helyüket. Az első programot a láncolt listás hasítás egyenletességét leíró 1. Lemma, míg a másodikat a H^p_{lin} hasítófüggvény-halmaz univerzálisságáról szóló 3. Tétel szemléltetése céljából írtam.

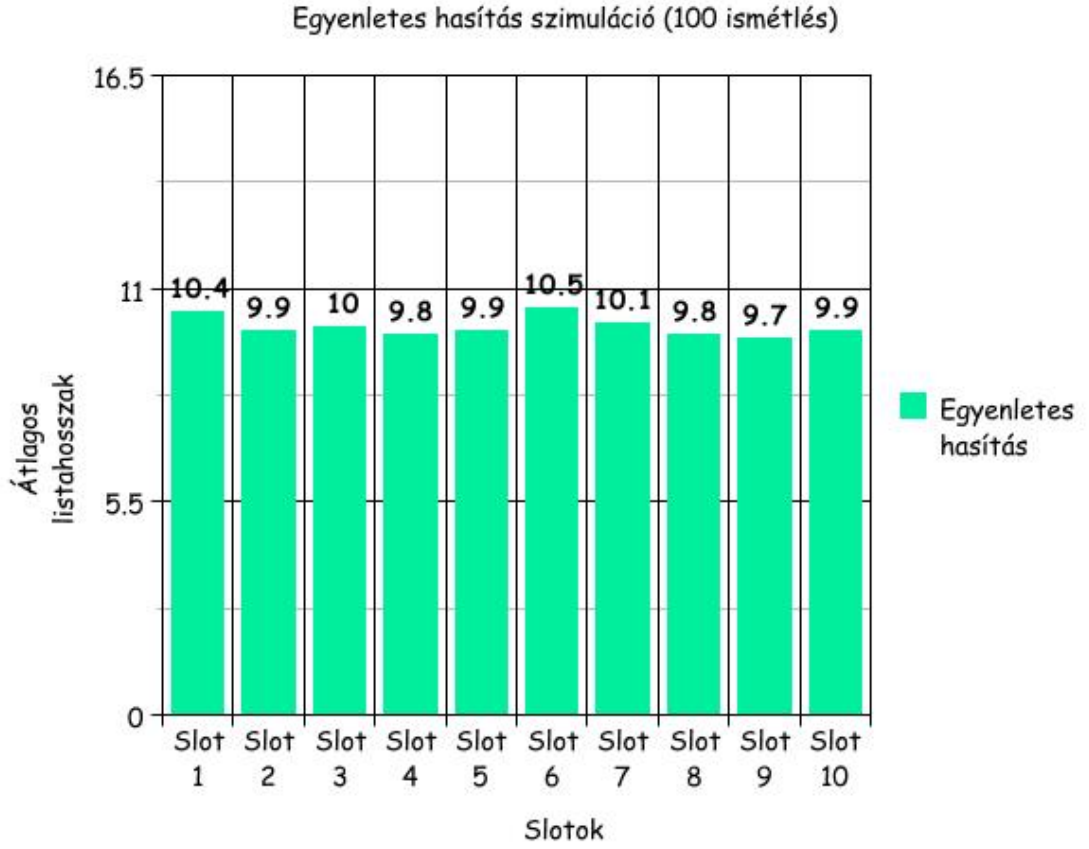
4.1. Egyenletes hasítás

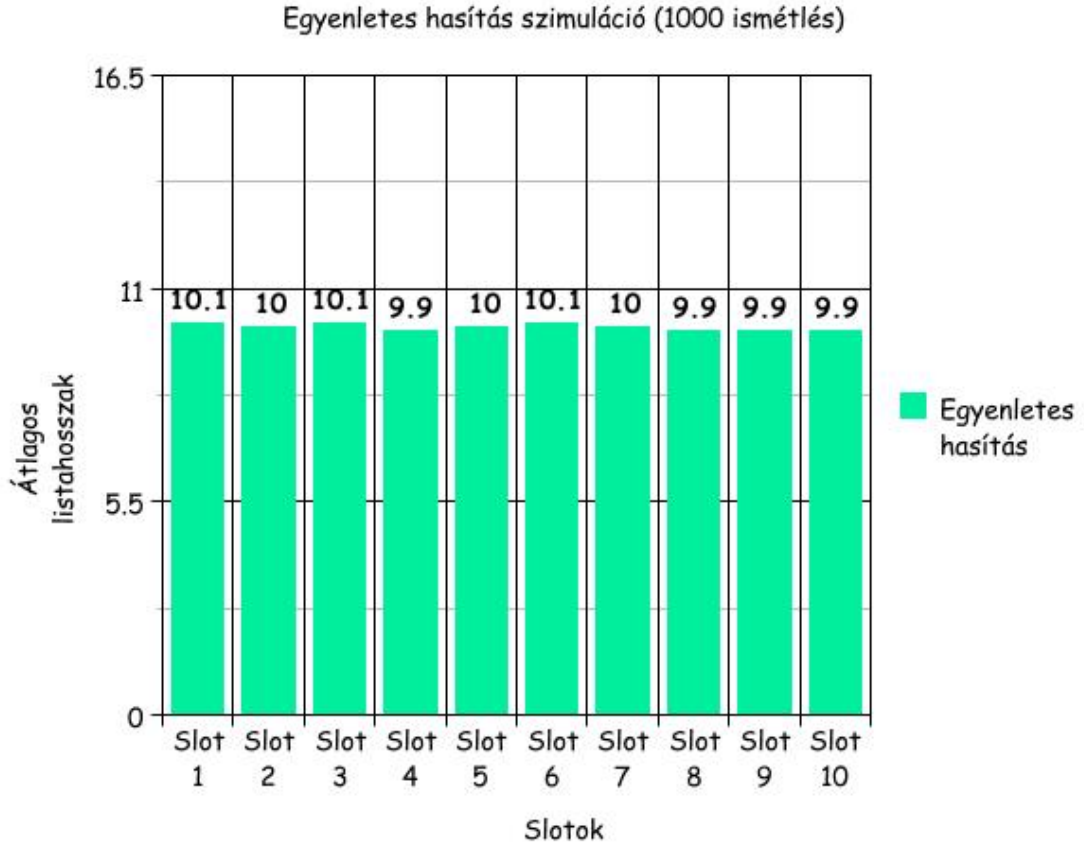
Az 1. Lemma azt mondja ki, hogy ha egy U kulcsuniverzumból véletlenszerűen kiválasztunk n elemet, akkor egy, az egész kulcsuniverzumot egyenletesen hasító hasítófüggvény várhatóan egyenletesen fogja hasítani a véletlen S inputhalmazt is. A programban az egyszerűség kedvéért az $U = \{0, 1, \ldots, r\}$, $T = \{0, 1, \ldots, m-1\}$ halmazokkal dolgoztam, az input, vagyis a hasítandó kulcshalmaz pedig n db kulcsból állt. A program minden egyes lépésben n db véletlenszerű és egymástól különböző kulcsot választ az univerzumból, majd ezeket T-re hasítja. A lépések száma, az n, m és az r (az univerzum legnagyobb eleme) a felhasználó által megadható paraméterek. Az algoritmus (egyszerűsített) struktogramját mutatja a következő ábra.

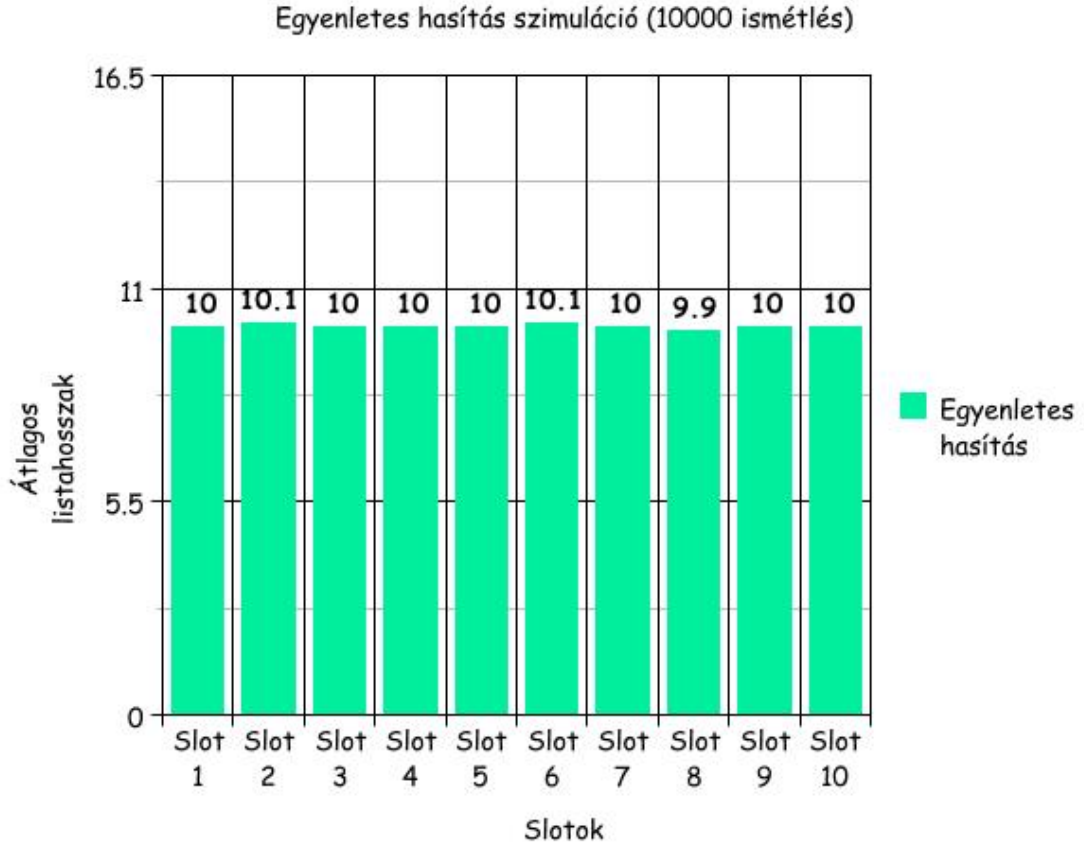


A szemléltetés végett egy konkrét példára elkészítettem az egyes slotra hasított kulcsok átlagos számát különböző ismétlésszámok esetén. Az alábbi eredmények az $|U| = 1000, |S| = 100, |T| = 10$ választás mellett készültek. A programot úgy módosítottam, hogy négy ismétlésszámot is megadhasson a felhasználó és a program minden így kapott ismétlési ciklus után kiírja az addigi összesített eredményt, vagyis a slotokra hasított elemek átlagos számát. Jelen szimulációban egymás után 10, 90, 900, majd végül 9000 ismétlést végeztem, ezáltal a kapott tényleges ismétlésszámok rendre 10, 100, 1000 és 10 000 lettek. Az egyes ismétlésszámokhoz tartozó eredményről hisztogramokat készítettem. Jól látható, hogy $n \longrightarrow \infty$ mellett az ábra egyre jobban "kisimul".


Egyenletes hasítás szimuláció (10 ismétlés)
16.5
11
5.5
0
Átlagos listahosszak
10
8.9
11.2
8.9
11.9
11.3
10.4
8.8
8.4
10.2
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Egyenletes hasítás


Egyenletes hasítás szimuláció (100 ismétlés)
16.5
11
5.5
0
Átlagos listahosszak
10.4
9.9
10
9.8
9.9
10.5
10.1
9.8
9.7
9.9
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Egyenletes hasítás





Az 1. Lemma alapján minden slotra teljesül, hogy az elemek várható száma nem éri el az $\frac{n}{m}+1$ értéket, mely jelen esetben 11. Világos, hogy minden egyes ismétlésre

nem fog minden slot megfelelni, ám a nagy számok törvénye alapján azt várhatjuk, hogy ha a lemma állítása helyes, akkor az ismétlésszám növelésével előbb-utóbb minden slot meg fog felelni. Mivel a lemmát bebizonyítottuk, ennek valóban így kell lennie. A szimuláció eredményeit vizsgálva megfigyelhető, hogy 10 ismétlést követően még több slot is a "nem felelt meg" kategóriába esett, de az ismétlésszám növelésével már mindegyik (bőven) megfelelt. Az is látható, hogy minél több ismétlést végeztünk, annál inkább az elméleti várható értékhez (≈ 10) közelítettek az értékek, ahogy ezt a nagy számok erős törvénye alapján el is vártuk.

4.2. Univerzális hasítás két nézőpontból

A 3. Tétel azt állítja, hogy ha p prím és $h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$, akkor a $H^p_{lin} = \{h_{a,b} \mid a \in \{1, 2, \ldots, p-1\} \text{ és } b \in \{0, 1, \ldots, p-1\}\}$ hasítófüggvény-halmaz univerzális. A Tétel gyakorlati alátámasztására két különböző, ám egyaránt helyes értelmezésen alapuló programot mutatok be, illetve ismét egy-egy konkrét példán keresztül szemléltetem az eredményeket.

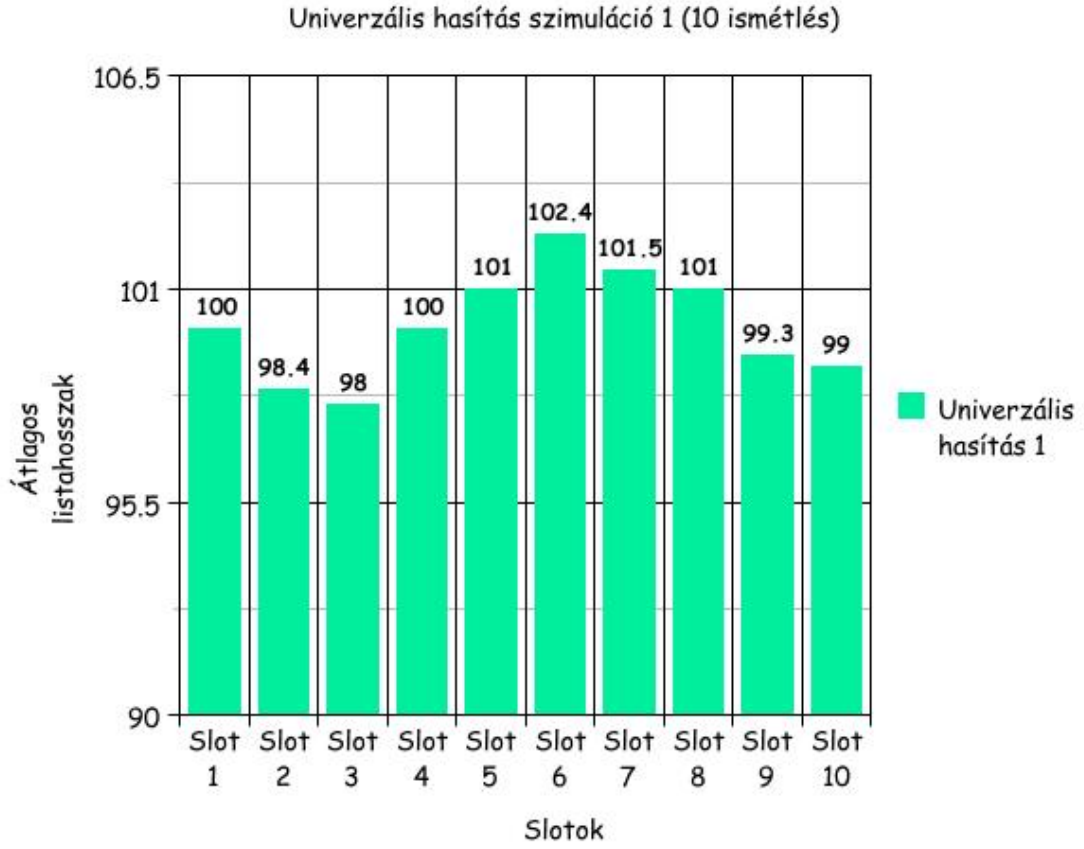
4.2.1. Első értelmezés

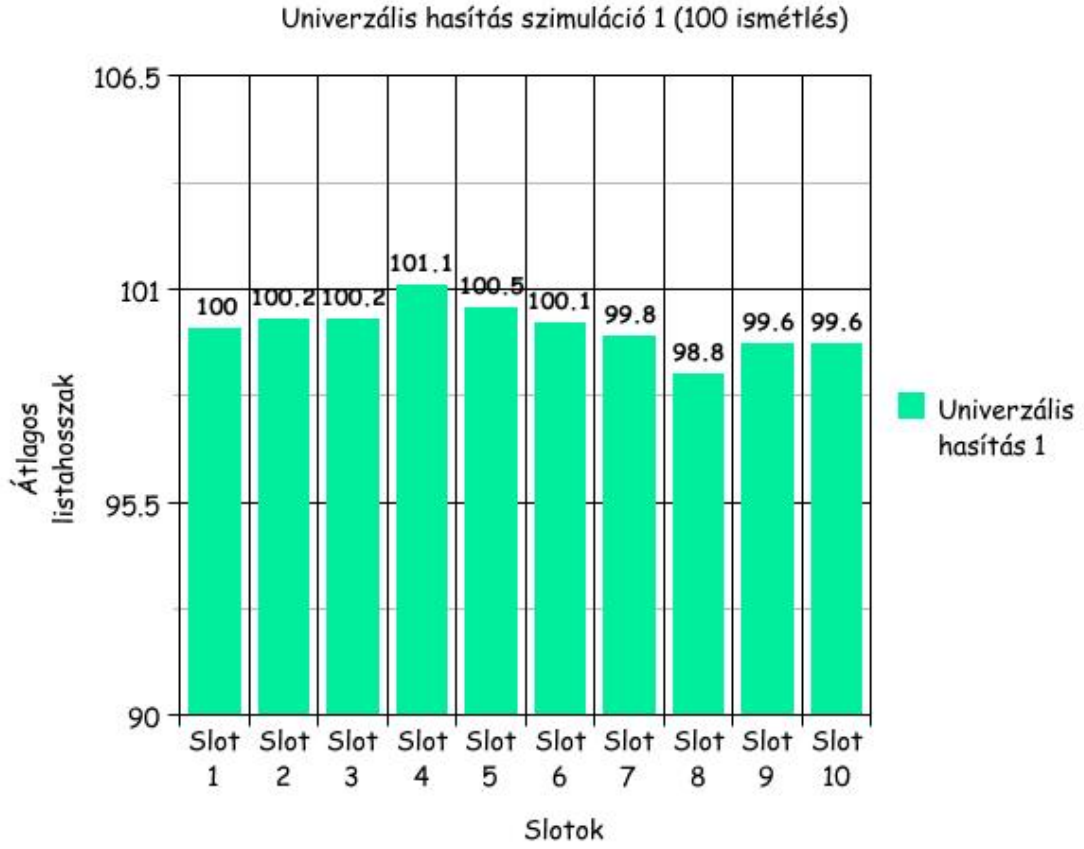
Az első értelmezés során azt vizsgáljuk meg, hogy igaz-e az, hogy egy tetszőleges inputot véve és a hasítófüggvényt véletlenszerűen választva a H^p_{lin} halmazból, a hasítás várhatóan egyenletes. Ennek érdekében a felhasználó megadja az univerzum p maximális elemét, melynek prímnek kell lennie, valamint a kiválasztandó rossz input elemszámát, az ismétlések számát, illetve a hasítótábla méretét, m-et. Ha nem prímnek választotta p-t, akkor a program ezt jelzi és megadja a beírt értékhez legközelebbi prímet, majd ismét megkérdezi, hogy mi legyen p, mindezt addig, amíg prímet nem kap paraméterként. A program a Tétel által meghatározott halmazokból kiválaszt egy véletlen A és egy véletlen B elemet, melyekkel megalkotja a rossz inputot. Ennek elemszáma épp a felhasználó által megadott lesz, tulajdonsága pedig az, hogy a kiválasztott A és B által meghatározott H^p_{lin}-beli hasítófüggvény minden elemét ugyanarra a (szintén véletlenszerűen kisorsolt) T-beli slotra hasítja. Azt szeretnénk tapasztalni, hogy ha véletlenszerűen választjuk az a és b értékeket (és ezáltal az $((ax+b) \bmod p) \bmod m$ hasítófüggvényt), akkor várhatóan minden slotra a hozzá rendelt láncolt lista elemeinek száma kisebb, mint $\frac{n}{m} + 1$, hiszen ezt jelenti az egyenletesség. Az algoritmus működését az alábbi (egyszerűsített) struktogram vázolja:

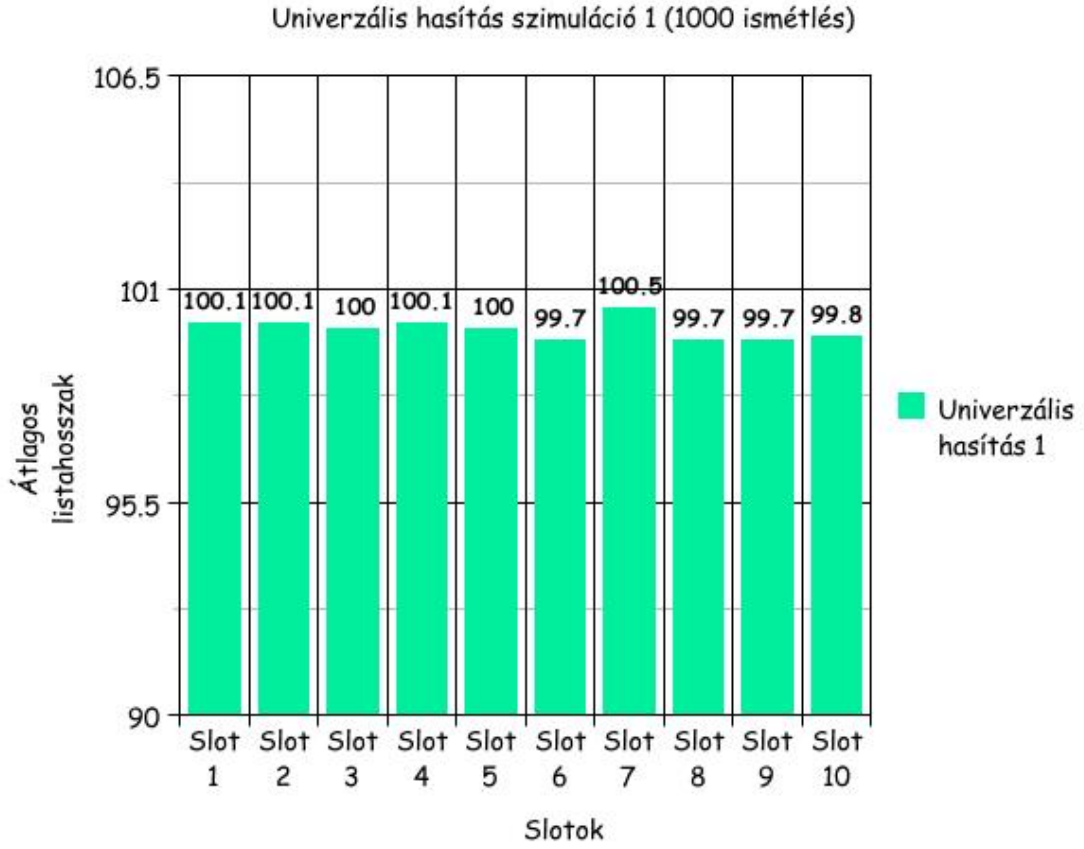
Univerzális hasítás 1

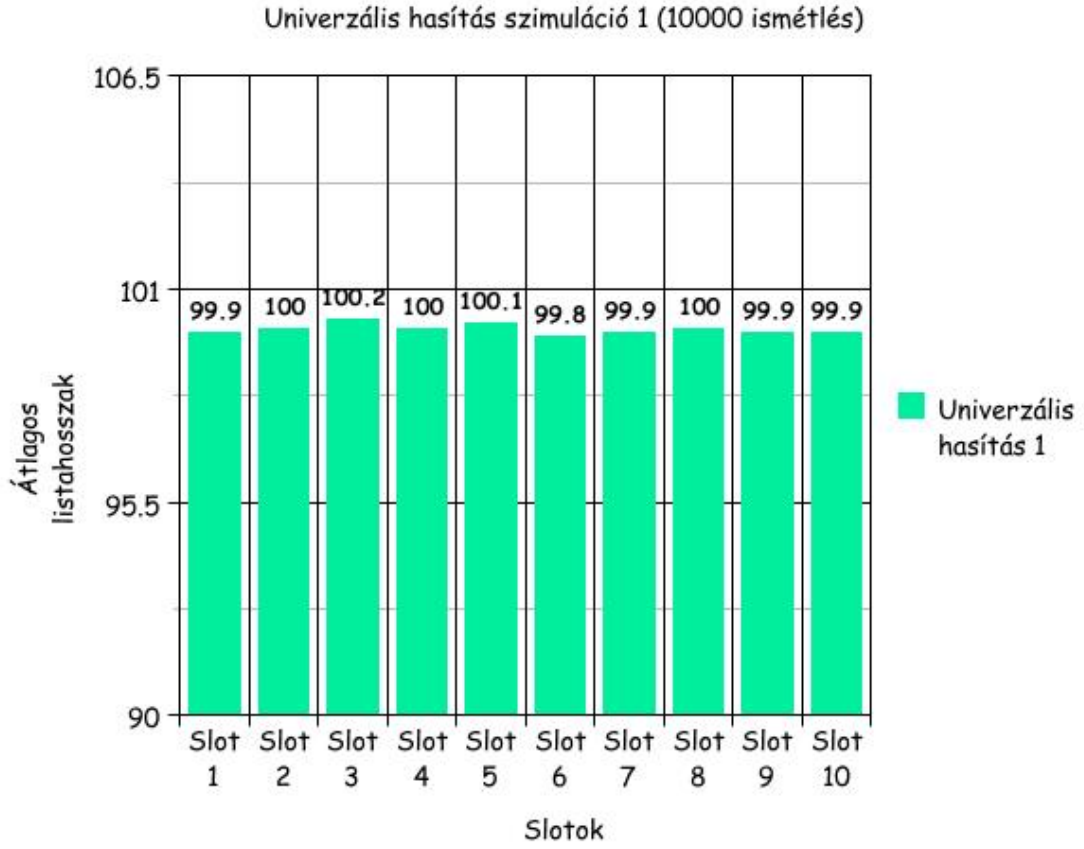
```
while p nem prím
    p értékének bekérése
rossz slot = RND(0, m-1)
A = RND(1, p - 1)
B = RND(0, p - 1)
for i < Input
    Választunk egy, a RosszInputban még nem szereplő
    x elemet az univerzumból, melyre
    ((A * x + B) mod p) mod m = rossz slot
    x -> RosszInput
for k < Ismétlések száma
    a = RND(1, p - 1)
    b = RND(0, p - 1)
    for j < Input
        r := (a * RosszInput[j] + b) mod p) mod m
        T[r] := T[r] + 1
for (i = 0, i < m, i++)
    T[i] := T[i] / Ismétlésszám
```

A konkrét példában az $r = 11113$, $m = 10$, $n = 1000$ paraméterek mellett 10, 100, 1000 és 10 000 ismétlés után kapott eredményeket a következő négy hisztogram szemlélteti.


Univerzális hasítás szimuláció 1 (10 ismétlés)
106.5
101
95.5
90
Átlagos listahosszak
100
98.4
98
100
101
102.4
101.5
101
99.3
99
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Univerzális hasítás 1


Univerzális hasítás szimuláció 1 (100 ismétlés)
106.5
101
95.5
90
Átlagos listahosszak
100
100.2
100.2
101.1
100.5
100.1
99.8
98.8
99.6
99.6
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Univerzális hasítás 1


Univerzális hasítás szimuláció 1 (1000 ismétlés)
106.5
101
95.5
90
Átlagos listahosszak
100.1
100.1
100
100.1
100
99.7
100.5
99.7
99.7
99.8
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Univerzális hasítás 1


Univerzális hasítás szimuláció 1 (10000 ismétlés)
106.5
101
95.5
90
Átlagos listahosszak
99.9
100
100.2
100
100.1
99.8
99.9
100
99.9
99.9
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slotok
Univerzális hasítás 1

Itt is megfigyelhető, hogy az ismétlésszám növekedésével "a tétel működik", mivel a nagy számok erős törvénye értelmében a várható érték az átlaghoz tart, mely körülbelül 100. Eleinte még itt sem felel meg minden slot, de 1000 ismétlésre már szép az eredmény, 10 000-re pedig már majdnem teljesen "egyenletes".

Ezzel tehát megmutattuk egy gyakorlati példán keresztül, hogy az univerzális hasítás segítségével várhatóan egyenletes hasítást érhetünk el olyan rossz inputok esetében is, melyeket katasztrofálisan rosszul is hasíthat a halmaz egy hasítófüggvénye. Ezáltal elértük, hogy nem kaphatunk olyan inputot, melyet nem hasítunk nagy valószínűséggel jól, míg a rendes hasítás esetében ilyen input könnyedén adható volt bármely hasítófüggvényhez.

4.2.2. Második értelmezés

Ezen verziónál a paraméterek és a jelölések megegyeznek az előző verzió paramétereivel és jelöléseivel. A felhasználónak meg kell adnia továbbá, hogy hány elemre végezze el a program a szimulációt. Jelölje ezt az értéket Ism. Próbáljuk meg a 3. Tételt szó szerint értelmezni, majd szemléltetni. A főtétel azt állítja, hogy az S input bármely eleme mellé várhatóan legfeljebb $\frac{n}{m}$ elemet hasítunk. A program ennek megfelelően úgy működik, hogy kiválaszt egy elemet véletlenszerűen a "rossz" inputból, majd elvégzi az S halmaz univerzális hasítását a megadott ismétlésszámra, és megszámolja, hogy hány elem került a kiválasztott elemmel azonos slotba. Ezt megismétli annyi különböző elemre a "rossz" inputból, amennyit kértünk tőle. A példában az előzővel megegyező paraméterekkel számolunk, a különbség mindössze annyi, hogy ezúttal 100, 10 000, illetve 100 000 ismétlés mellett nézzük meg az eredményt. Az új paraméter, mely a megvizsgált $x \in S$ elemek számát adja meg, legyen most 10. Ismét elkészítettem a program struktogramját a jobb áttekinthetőség kedvéért. A Tétel alapján tehát azt várjuk, hogy 100 alatt lesz mindegyik "rossz" elemre a mellékük hasított elemek száma. Ez valóban teljesül is, hiszen, mint azt a hisztogramok is mutatni fogják, 100 000 ismétlés mellett már minden slot megfelel és a hasítás nagyon egyenletes lesz. Jellemzően 99.8 körüli átlagos melléhasított elemszámot kaptunk. Ez megfelel a várakozásainknak, hiszen egy elemet kiválasztva a maradék 999 hasítását végezzük el 10 slotra, mely átlagosan 99.9 elem/slot eredményt adna. A kissé alacsonyabb értékek nem jelentenek problémát, épp ellenkezőleg, hiszen az egy kiválasztott elem befolyásolja a maradék 999 "összetételét", így azokra a hasítás már nem lesz teljesen egyenletes, a kiválasztott elem slotjára kicsit kevesebb hasított elem fog várhatóan esni. Az alábbi ábrákon tehát az algoritmus (egyszerűsített) struktogramja, illetve a kapott eredményeket bemutató hisztogramok láthatóak.

Univerzális hasítás 2

while p nem prím

p értékének bekérése

rossz slot = RND(0, m-1)

A = RND(1, p - 1); B = RND(0, p - 1)

for i < Input

Választunk egy, a RosszInputban még nem szereplő
x elemet az univerzumból, melyre
((A * x + B) mod p) mod m = rossz slot

x -> RosszInput

for k < Ism

Választunk egy még nem vizsgált x-et a RosszInputból

d[k] := 0

for j < Ismétlések száma

a = RND(1, p - 1); b = RND(0, p - 1)
f(s) := ((a * s + b) mod p) mod m

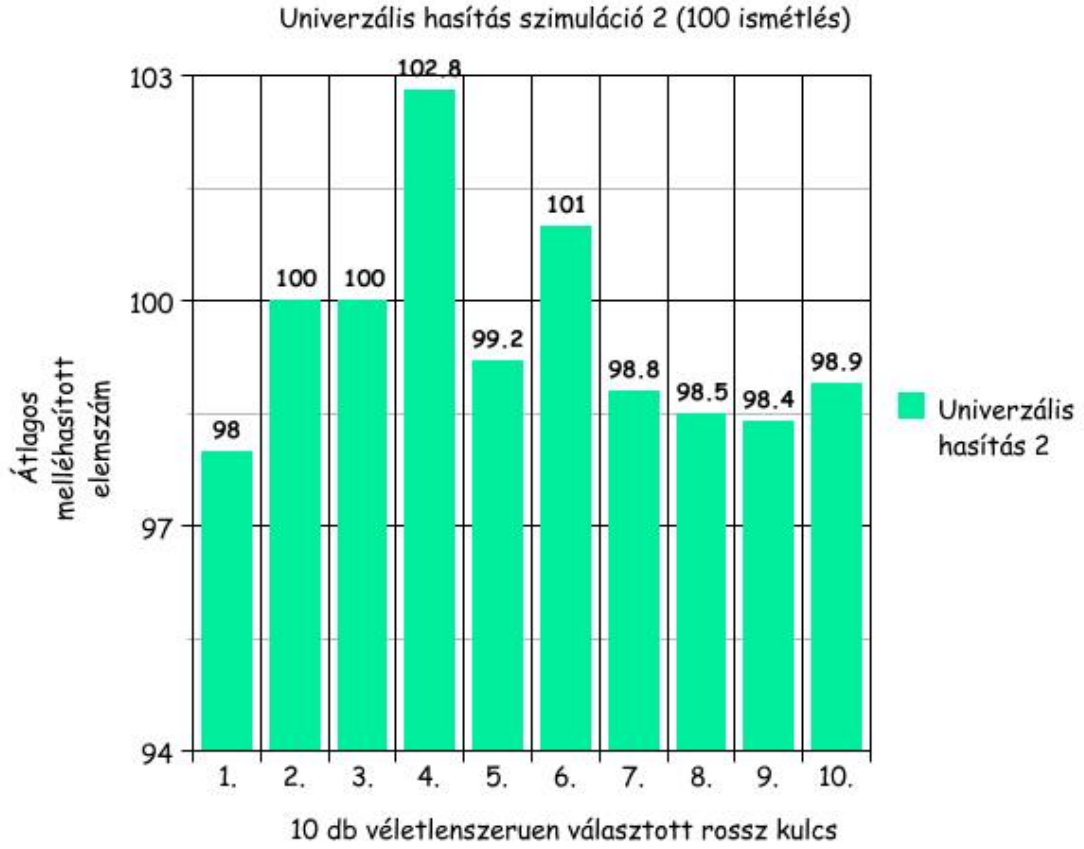
for j < Input

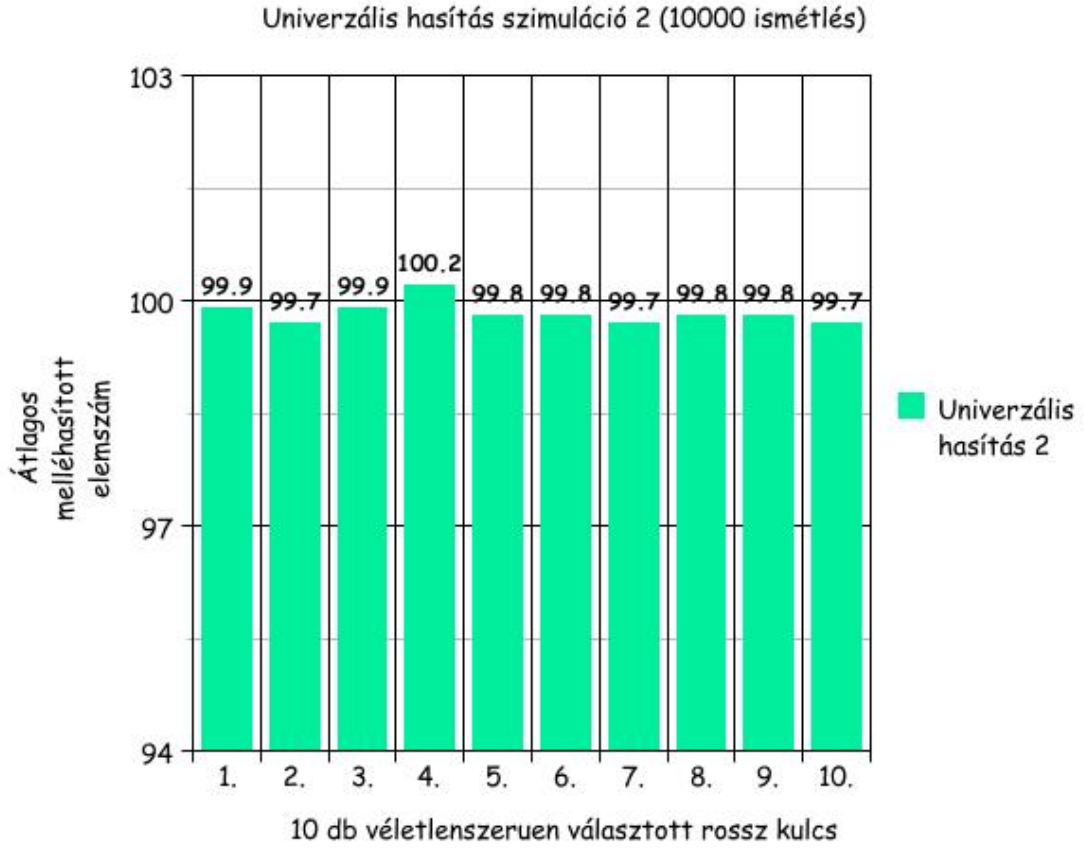
if f(s) x-et és RosszInput[j]-t ugyanoda hasítja

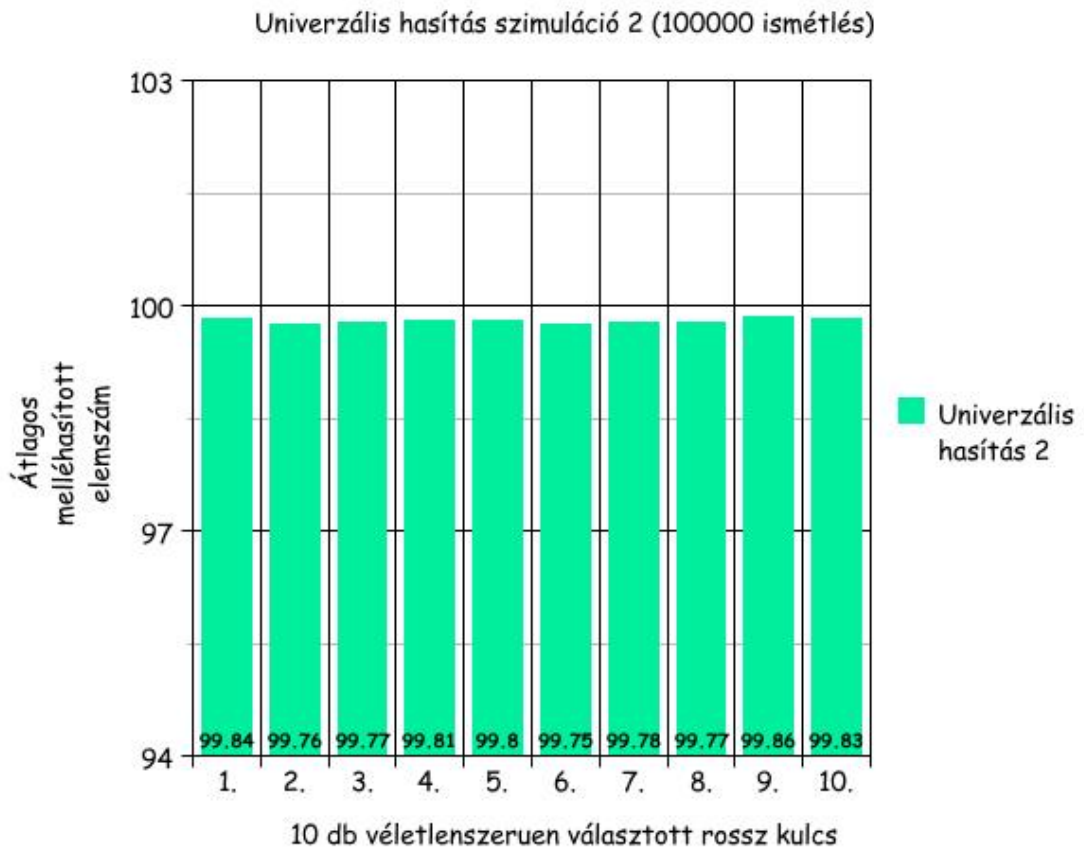
igaz: d[k] := d[k] + 1

hamis: SKIP

d[k] := d[k] / Ismétlések száma


Univerzális hasítás szimuláció 2 (100 ismétlés)
Átlagos melléhasított elemszám
103
100
97
94
98
100
100
102.8
99.2
101
98.8
98.5
98.4
98.9
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
10 db véletlenszeruen választott rossz kulcs
Univerzális hasítás 2


Univerzális hasítás szimuláció 2 (10000 ismétlés)
Átlagos melléhasított elemszám
103
100
97
94
99.9
99.7
99.9
100.2
99.8
99.8
99.7
99.8
99.8
99.7
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
10 db véletlenszeruen választott rossz kulcs
Univerzális hasítás 2



Így a Tételt két különböző nézőpontból is megvizsgáltuk és szemléltettük, hogy az univerzális hasítás elve nagyon jól működik a gyakorlatban. Megjegyzem, hogy mindkét példára igaz, hogy a kis ismétlésszámra meg nem felelő slotok is alig tértek el az ideálistól, vagyis az univerzális hasítás már 10 ismétlésre is nagyon jó eredményt ad, sőt, egyetlen egy futásra is jó a kapott eloszlás a legtöbb esetben. A gyakorlati alkalmazhatóságot ez utóbbi tény nagymértékben javítja.

Irodalomjegyzék

[1] Juraj Hromkovic: *Design and Analysis of Randomized Algorithms*, Springer-Verlag, 2010

[2] Thomas H. Cormen - Charles E. Leiserson - Ronald L. Rivest - Clifford Stein: *Új algoritmusok*, Scolar kft., 2003

[3] Rónyai Lajos - Ivanyos Gábor - Szabó Réka: *Algoritmusok*, TYPOTEX, 2005

[4] Niklaus Wirth: *Algoritmusok + Adatstruktúrák = Programok*, Műszaki Könyvkiadó, 1982

[5] Robert Sedgewick - Kevin Wayne: *Algorithms (4th edition)*, Addison-Wesley Professional, 2011

[6] Freud Róbert: *Lineáris algebra*, ELTE Eötvös Kiadó, 2006

Köszönetnyilvánítás

Köszönöm az érdekes témajavaslatot témavezetőmnek, Fekete Istvánnak, valamint azt is, hogy rengeteg elfoglaltsága mellett is tudott időt szakítani rám és ötleteivel, javaslataival segítette a szakdolgozat elkészültét. Köszönettel tartozom továbbá családomnak és szeretteimnek támogatásukért és erőt adó biztatásukért.