Eötvös Loránd Tudományegyetem

Természettudományi Kar

Bináris keresőfák kiegyensúlyozásai

BSc szakdolgozat

Egyed Boglárka

Matematika BSc, Alkalmazott matematikus szakirány

Témavezető: Fekete István, egyetemi docens

Informatikai Kar, Algoritmusok és Alkalmazásaik Tanszék

Budapest, 2010

Tartalomjegyzék

1. fejezet

Bevezetés

Véges adathalmazokkal kapcsolatos feladatok során gyakran merül fel az igény bizonyos adatok keresésére, törlésére, vagy éppen új elem beszúrására. A minél hatékonyabb munkavégzés megköveteli, hogy ezek a műveletek gyorsak és lehetőleg egyszerűek legyenek. A bináris keresőfák olyan adatszerkezetek, amelyek az adathalmazt láncolt struktúraként ábrázolják olyan módon, hogy minden csúcs az adathalmaz egy elemének feleljen meg. Felhasználva, hogy a legtöbb művelet a keresés elem-összehasonlító elvén alapszik, megmutatjuk, hogy a keresés, beszúrás és törlés műveletideje nagyságrendben megegyezik az általunk használt bináris keresőfa magasságával.

A gyakorlatban nyilván ennél többre is kíváncsiak vagyunk, bizonyos paraméterek – például a fa csúcsainak száma – ismeretében szeretnénk jó becslést adni a keresés várható műveletigényére, a műveletek átlagos futásidejére.

Első lépésként definiálni fogjuk a bináris keresőfa fogalmát, majd ismertetjük annak fontosabb tulajdonságait. A továbbiakban azonban a bináris keresőfa egy gyakorlati szempontból hasznosabb fajtáját, a véletlen építésű bináris keresőfát tekintjük, mivel ez a kulcssorrendek valószínűségeinek szempontjából közel áll a való életben előforduló bináris keresőfákhoz. Ezt követően matematikai módszerekkel elemezzük a keresőfa átlagos magasságát, illetve átlagos csúcsmagasságát, amelyek éppen az általunk keresett becsléseket adják meg. A későbbiekben látni fogjuk, hogy egy n különböző elemből álló véletlen építésű bináris keresőfa átlagos magassága $O(\log_2 n)$ – precízebben legfeljebb $2.98 \log_2 n$ –, átlagos csúcsmagassága pedig nem

nagyobb, mint $1.39 \log_2 n$. Fontos megjegyezni, hogy ezek a kedvező eredmények mind annak köszönhetők, hogy feltesszük, a keresőfába beszúrt elemek egyenletesen oszlanak el az adathalmazon, azaz teljesülnek rájuk az egyenletes eloszlásra vonatkozó előnyös tulajdonságok.

Ennyivel azonban még nem elégszünk meg, célunk, hogy a famagasságot, ezzel együtt pedig a műveletek futásidejét leszorítsuk, de nem csak egyenletes eloszlás, hanem az input bármely eloszlása esetén. Az erre irányuló törekvéseket valósították meg az úgynevezett kiegyensúlyozó eljárások, amelyek a bináris keresőfa minden egyes módosítását követően helyreállítják a keresőfa jó szerkezetét, azaz a kiegyensúlyozottságát. Ilyen kiegyensúlyozó eljárást többet is megvizsgálunk, mivel a kiegyensúlyozottság fogalma többféle módon értelmezhető, és ennek megfelelően többféle módszer is alkalmazható.

Az első kiegyensúlyozó eljárás az AVL-fa módszere, amely azon az elven alapszik, miszerint egy bináris keresőfa akkor kiegyensúlyozott, ha a bal-, illetve jobboldali részfájának magassága közötti különbség a lehető legkisebb. Megmutatjuk, hogy egy ilyen módon kiegyensúlyozott n csúcsú keresőfa magassága felülről becsülhető $1.44 \log_2 n$-nel, ami nagyon jó műveleti időt eredményez.

Egy másik, az AVL-fához hasonló fastruktúra a piros-fekete fa, amely a keresőfa csúcsainak színezésével próbálja a fa kedvező alakját fenntartani. Látni fogjuk, hogy az ilyen módon felépülő n csúcsú keresőfák magassága legfeljebb $2 \log_2(n+1)$.

Az előző kettő struktúrától merőben eltérő kiegyensúlyozott típus az S-fa, vagy idegen szóval *splay tree*, amelynek érdekessége, hogy nem a fa alakjának újbóli helyreállításával tartja elfogadható szinten a műveletek futási idejét. A módszer lényege, hogy az egyes elemek elérési gyakoriságát figyelve igyekszik a leggyakrabban használt elemeket a fa gyökerének közelébe, míg a ritkán használtakat a levelek felé mozgatni. Értelemszerűen ezzel a várható keresési idő csökken, ami pedig azt jelenti, hogy a beszúrás és törlés műveleti ideje is, hiszen mindkettő eljárás a keresés algoritmusán alapszik. Megmutatjuk még, hogy az S-fák építése nem igényel nagy költséget, ami a gyakorlatban igen hasznos tulajdonságnak bizonyul.

Végül bemutatunk egy új eljárást, az RBS-fát, amely bizonyos paraméterek mellett képes az AVL-fa hatékonyságát közelíteni. Lényege, hogy nem engedi a keresőfát

túl magasra nőni, azaz beállít egy $c \log_2 n$-es felső határt a fa magassága számára, ahol a c konstanst előre rögzítjük. Az eljárás ezt a bináris keresőfa költségesebb – lineáris idejű –, de várhatóan ritkább teljes átrendezésével éri el. Számítógépes program segítségével megpróbálunk olyan becslést adni c értékére, hogy az RBS-fa hatékonysága valóban jó legyen.

A kiegyensúlyozó eljárások bemutatása után ugyanazon számítógépes program segítségével, egyszerű mintavételezéssel összevetjük a futási eredményeinket a számítással becsült paraméterekkel. Látni fogjuk, hogy általában a gyakorlatban kedvezőbb eredményeket kapunk a paraméterekre, mint az elmélet keretében végzett becslések során.

2. fejezet

Bináris keresőfák

Tegyük fel, hogy egy adott véges halmazon olyan műveleteket kívánunk végrehajtani, amelyek során az adott halmaz bővülhet, zsugorodhat, vagy egyéb értelemben módosulhat. Az ilyen halmazokat dinamikus halmazoknak nevezzük, mivel a műveletek végzése során folyton változhatnak. Azokat a dinamikus halmazokat, amelyeken értelmezve vannak a keresés, beszúrás és törlés műveletei, szótárnak nevezzük. A szótárak általában egy értékhalmaz lehetséges kulcsok egy halmazával vett direktszorzataként értelmezhetők, így minden halmazbeli elem egy kulccsal azonosítható. A kulcsütközés elkerülése végett tegyük fel, hogy a kulcshalmazon értelmezve van egy $<$ rendezés, ezzel garantáljuk, hogy a kulcsok egyediek legyenek, azaz hogy a kulcsmezőnek *kulcs-tulajdonsága* legyen. Az előbb definiált szótárat implementálhatjuk bináris keresőfa segítségével, amelynek fogalmát, tulajdonságait az *Algoritmusok* c. könyvben [6] található módon definiáljuk.

2.1. Mi a bináris keresőfa?

Definíció 2.1.1 *A bináris keresőfa egy olyan bináris faként szervezett adatszerkezet, amelyben a kulcshalmazt láncolt struktúraként ábrázoljuk. A bináris keresőfában az elemeket olyan módon tároljuk, hogy teljesüljön az úgynevezett bináris keresőfa-tulajdonság:*

> *A bináris keresőfa tetszőleges x csúcsa esetén, ha y az x baloldali részfájának egy csúcsa, akkor $kulcs(y) < kulcs(x)$, ha viszont a jobboldali részfá-*

nak eleme, akkor $kulcs(x) < kulcs(y)$.

Megjegyezzük, hogy a fenti definíciót nem elég szülő-gyerek párokra megfogalmazni.

A kulcsmezőn túl minden egyes csúcs tartalmaz egy bal, egy jobb, illetve egy szülő mutatót – idegen szóval pointert –, amelyek rendre az adott csúcs bal-, és jobboldali gyerekére, valamint a csúcs szülőjére mutatnak. Amennyiben nem létezik valamelyik gyerek, vagy a szülő, akkor a megfelelő mutató helyébe NIL kerül, ami a null pointert jelöli. Általában a dinamikus halmazok elemei egyéb, kísérő adatokat is tartalmaznak, ezek azonban a szótár megengedett műveleteinek elvégzésének szempontjából lényegtelenek, így a továbbiakban eltekintünk tőlük.

Megjegyzés. A bináris keresőfák nem csak adatfeldolgozás, de rendezési feladatok elvégzésének céljából is felhasználhatók, mivel felépítésükből adódóan, alkalmazva az *inorder* fabejárást, a kulcsokat rendezett sorrendben kaphatjuk vissza. Ez egy olyan rekurzív algoritmus, amelynek lényege, hogy a fa gyökerében levő kulcsot a baloldali részfájában levő értékek után, és a jobboldali részfájában levő értékek előtt írja ki. Rendezési feladatok esetén viszont megengedett a kulcsok egyenlősége, azaz a kulcshalmazon elegendő egy $\leq$ rendezést értelmeznünk.

2.2. Műveletek

A bináris keresőfákon értelmezett műveleteket két csoportba sorolhatjuk, a lekérdező, illetve a módosító műveletek csoportjába. A lekérdező műveletek a keresés, a minimum-, és maximumkeresés, valamint egy elem megelőzőjének, illetve rákövetkezőjének keresése. Ezek mind csupán információt közölnek a keresőfáról, módosítást nem végeznek azon. A módosító műveletek a beszúrás, és a törlés, amelyek viszont már megváltoztatják a keresőfát. A bináris keresőfák műveleteinek futásidejét a keresőfa elemszámának függvényében határozzuk meg.

Keresés

A bináris keresőfákon leggyakrabban használt és legfontosabb művelet a keresés, amelynek segítségével megtalálható egy adott kulcsú elem a fában tárolt elemek között. Ez egy rekurzív eljárás, inputként a keresőfa gyökerének mutatóját és a keresendő k kulcsértéket kell megadnunk. Amennyiben a fa nem üres, első lépésben összehasonlítja a gyökérhez tartozó kulcsértéket k-val, ha ezek megegyeznek, a keresés leáll, ha nem, akkor pedig a bal, vagy a jobb részfában halad tovább, attól függően, hogy k kisebb, vagy nagyobb volt a gyökérhez tartozó kulcsnál, hiszen a bináris keresőfa-tulajdonság miatt balra helyezkednek el a kisebb, jobbra pedig a nagyobb kulcsú elemek. Ha a keresendő elem megtalálható a fában, az eljárás a k kulcsú elem mutatóját fogja visszaadni, ha viszont nem, akkor pedig NIL-t.

Műveletigény: Egy lépésben egy összehasonlítás, majd pedig egy mutató mentén történő elmozdulás hajtódik végre, ezzel az eredetinél egy szinttel alacsonyabb fában folytatódik a keresés. A rekurzió során tehát egy gyökérből induló, a vizsgált csúcsokat érintő út keletkezik, ezért a keresés műveletigénye $O(h(t_n))$, ahol t_n az n csúcsból álló keresőfát jelöli, h pedig a magasságot.

A bináris keresőfákon egyéb keresési algoritmusok is megvalósíthatók, így például a minimum-, és maximumkeresés, vagy egy elem rákövetkezőjének, megelőzőjének keresése. Minimumkeresésnél a gyökértől indulva addig haladunk a mutatók mentén balra lefelé, amíg NIL-hez nem érünk, maximumkeresésnél pedig a jobboldali mutatók mentén ugyanígy, mivel a bináris keresőfa-tulajdonság miatt ekkor éppen a minimumhoz, illetve a maximumhoz érkezünk. Mivel mindkét eljárás egy gyökérből induló utat vesz végig, így a műveletigény továbbra is $O(h(t_n))$.

Egy x elem rákövetkezőjének keresésekor azt a kulcsot keressük, amely a legkisebb olyan kulcs, ami x kulcsánál nagyobb. Ehhez nem kell kulcs-összehasonlításokat végeznünk, mivel a bináris keresőfa konstrukciójából adódóan elegendő a mutatók mentén végiglépdelnünk. Történetesen, ha x jobboldali részfája nem üres, akkor x rákövetkezője a jobboldali részfa legkisebb kulcsú eleme, amit az előbb említett minimumkereséssel könnyen megtalálhatunk. Ha pedig a jobboldali részfa üres, akkor a rákövetkező csakis az az elem lehet, amely a legalacsonyabban fekvő olyan őse x-nek, hogy annak baloldali gyereke szintén őse x-nek. Ezt egyszerűen megtalálhatjuk,

ha addig haladunk felfelé x-től, amíg egy olyan csúcsot nem találunk, amely már baloldali gyereke a szülőjének. Így azt az y csúcsot keressük meg, amelynek x a közvetlen megelőzője, azaz y rákövetkezője az x-nek.

A műveletigény ugyancsak $O(h(t_n))$ lesz, mivel ismét egy utat veszünk végig, az első esetben egy felülről lefelé, a másodikban pedig egy alulról felfelé haladót. Elem megelőzőjének keresése hasonlóan leírható, ekkor ugyanis a rákövetkező keresésének inverz útját járja be az algoritmus, így a műveletigény szintén $O(h(t_n))$ lesz.

Beszúrás

Egy elem beszúrása során fontos, hogy a beszúrni kívánt elem még ne legyen eleme a keresőfának, azaz ne sérüljön a kulcs mező kulcs-tulajdonsága, továbbá, hogy a bináris-keresőfa tulajdonság megmaradjon. Legyen x a beszúrandó elem, kulcsértéke k, és a bal, illetve a jobb mutatója egyelőre NIL. Az előzőekben bemutatott keresés művelet segítségével megkeressük x helyét, azaz a gyökértől indulva mindaddig haladunk lefelé, amíg egy olyan csúcsot nem találunk, amelynek kulcsértéke megegyezik k-val – ez esetben kulcsütközés jön létre, azaz az elemet nem tudjuk beszúrni – , vagy NIL-hez nem érünk, ekkor ennek a NIL-nek a helyére szúrjuk be x-et. Műveletigénye a keresés műveletéhez hasonlóan $O(h(t_n))$.

Törlés

Törlés során az eljárás a törlendő elem mutatóját kapja meg argumentumként. A mutató meghatározását értelemszerűen egy keresés előzi meg, így garantált, hogy értéke helyes, vagyis a keresőfa egy csúcsára mutat. Legyen x a törölni kívánt elem. Törlés során három esetet különböztetünk meg aszerint, hogy x-nek hány gyereke van: ha nulla, akkor egyszerűen töröljük őt, ha egy, akkor x-et a gyerekével helyettesítjük, ha pedig kettő, akkor kivágjuk x azon legközelebbi rákövetkezőjét, amelynek nincs baloldali gyereke és annak kulcsával, adataival helyettesítjük x kulcsát, illetve adatait. A törlés algoritmusa hasonló a beszúráséhoz, egyetlen különbsége, hogy amint megtalálja azt a csúcsot, amelynek kulcsértéke k, kiláncolja, és a fent említett módon, pointerek átállításával helyettesíti egy másikkal. Műveletigénye a keresési eljárásokhoz hasonlóan $O(h(t_n))$.

Fontos látnunk, hogy az elvégzendő műveletek hatékonysága nagyban függ a bináris keresőfa felépítésétől, hiszen, mint azt megállapítottuk, a műveletigények aszimptotikusan felülről becsülhetők a keresőfa magasságával. Ha például a beszúrandó kulcsok szigorúan növekvő – illetve csökkenő – sorrendben érkeznek, akkor minden kulcs az előző jobb – illetve bal – gyereke lesz, azaz egy elfajult fát, egy (lineáris) láncot kapunk. Ez tehát a legrosszabb eset, ami igen rossz hatékonyságot eredményez. Egy n csúcsú, $h(t_n)$ magasságú bináris keresőfa magasságára általánosan a következő becslés áll fent:

$$\lfloor \log_2 n \rfloor \leq h(t_n) \leq n - 1.$$

A legjobb esetben a bináris keresőfa majdnem teljes, azaz legfeljebb az alsó szinten hiányozhatnak csúcsok.

3. fejezet

Véletlen építésű bináris keresőfák

A továbbiakban, a bináris keresőfák paramétereinek meghatározásának céljából, véletlen beszúrások sorozatával felépülő bináris keresőfákon vizsgálódunk. Megjegyezzük, hogy azért nem olyan keresőfákat tekintünk, amelyek beszúrások és törlések sorozatával épül fel, mert annak átlagos magasságáról igen keveset tudnánk csak mondani. Elsőként tekintsük a véletlen építésű bináris keresőfák fogalmát:

Definíció 3.0.1 *Véletlen építésű bináris keresőfának nevezünk minden olyan bináris keresőfát, amely n darab különböző elemből épül fel olyan módon, hogy az elemeket tetszőleges sorrendben szúrjuk be a fába, és az n elem mind az $n!$ permutációja egyforma valószínűségű.*

Megjegyzés. A beszúrandó n különböző elemet leggyakrabban az $\{1, 2, ..., n\}$ halmaz elemeiként képzeljük el, ezzel egyszerűsítve az elemzést. Fontos még látnunk, hogy a véletlen építésű bináris keresőfa fogalma nem azonos egy n csúcsú véletlen módon választott bináris keresőfa fogalmával. Nézzünk $n = 3$-ra egy példát: ekkor az $1, 2, 3$ elemeknek $3! = 6$ különböző sorrendje van, de 3 csúcsú bináris keresőfa alakra nézve csak 5, mivel a $2, 1, 3$ és a $2, 3, 1$ elemek ugyanazt a keresőfát építik fel.

3.1. Átlagos famagasság

Az előző fejezet végén tett megjegyzés alapján vizsgáljuk meg egy véletlen beszúrásokkal felépített keresőfa átlagos magasságát. Megmutatjuk, hogy ez az érték egy

n különböző elemet tartalmazó véletlen építésű bináris keresőfa esetén $O(\log_2 n)$. Érdekesség, hogy a majdhogynem enciklopédikusnak tartott *Algoritmusok* c. könyv első [6], illetve második [5] kiadása erre más-más bizonyítást közöl. Az alábbiakban mindkét bizonyítást bemutatjuk, amelyek közös tulajdonsága, hogy a valószínűségszámítás és a matematikai statisztika elemeit használják fel. Az első kiadásbeli bizonyítás előtt lássunk néhány lemmát.

Lemma 3.1.1 *Legyen t olyan fa, amelyet az n különböző értéket tartalmazó $k_1, k_2, \ldots, k_n$ kulcssorozat elemeinek ilyen sorrendben való egymás utáni beszúrásával kapunk. Ekkor $1 \leq i < j \leq n$ esetén k_i a k_j-nek t-ben akkor és csak akkor őse, ha*

$$k_i = min\{k_l : 1 \leq l \leq i \text{ és } k_l > k_j\} \text{ vagy } k_i = max\{k_l : 1 \leq l \leq i \text{ és } k_l < k_j\}.$$

Bizonyítás. $\Rightarrow$: Tegyük fel, hogy k_i a k_j őse. Tekintsük azt a fát, amelyet a $k_1, k_2, \ldots, k_i$ kulcsok beszúrása után kapunk. Ebben, az eredetinél kisebb fában a gyökérből k_i-be vezető út megegyezik az eredeti fa gyökérből k_i-be vezető útjával. Ekkor, ha k_j-t beszúrjuk a $k_1, k_2, \ldots, k_i$ kulcsokat tartalmazó kisebb fába, akkor éppen a k_i egyik gyereke lesz, hiszen feletettük, hogy k_i őse k_j-nek. Egyszerűen belátható, hogy ebben az esetben k_i vagy a k_j-nél nagyobb $k_1, k_2, \ldots, k_n$-beliek közül a legkisebb, vagy a k_j-nél kisebb $k_1, k_2, \ldots, k_n$-beliek közül a legnagyobb.
$\Leftarrow$: Tegyük fel, hogy k_i a k_j-nél nagyobb $k_1, k_2, \ldots, k_n$-beliek közül a legkisebb. Ekkor a gyökérből k_i-be vezető út bármely elemét k_j-vel összehasonlítva ugyanazt az eredményt kapjuk, mintha k_i-vel hasonlítottuk volna össze őket. Így, amikor beszúrjuk k_j-t, az algoritmus egy k_i-n keresztülhaladó úton halad végig és ott helyezi el k_j-t, tehát k_i valóban k_j őse lesz. Hasonlóan bizonyítható arra az esetre is, amikor k_i a k_j-nél kisebb $k_1, k_2, \ldots, k_n$-beliek közül a legnagyobb. □

Következmény 3.1.1 *Legyen t olyan fa, amelyet az n különböző értéket tartalmazó $k_1, k_2, \ldots, k_n$ kulcssorozat elemeinek ilyen sorrendben való egymás utáni beszúrásával kapunk. Egy adott k_j kulcsra, ahol $1 \leq j \leq n$, definiáljuk a következő halmazokat:*

$$G_j = \{1 \leq i < j \text{ és } k_l > k_i > k_j \ \forall\ l < i\text{-re, ha } k_l > k_j\}$$

és

$$L_j = \{1 \leq i < j \text{ és } k_l < k_i < k_j \ \forall\ l < i\text{-re, ha } k_l < k_j\}$$

Ekkor a gyökérből a k_j-be vezető út kulcsai pontosan a $G_j \cup L_j$ elemei, és a k_j kulcs t-beli mélysége $d(k_j, t) = |G_j| + |L_j|$.

A G_j halmaz tehát azokat, a még k_j előtt beszúrt, k_i elemeket tartalmazza, amelyekre k_i a k_j-nél nagyobb $k_1, k_2, \ldots, k_n$-beliek közül a legkisebb. (Az L_j halmaz szerkezete ehhez hasonlóan leírható.) Tekintsük azt a G'_j halmazt, amely a $k_1, k_2, \ldots, k_{j-1}$-beli, k_j-nél nagyobb kulcsokból áll. Vegyük G'_j elemeit az indexük szerinti sorrendben, és végezzünk rajtuk ismételt minimum-kiválasztást. Ekkor G_j halmazba pontosan azok az elemek kerülnek bele, amelyek a minimum-kiválasztás során a futó minimumot átírják. Általánosítsuk a problémát: tekintsünk n különböző számot, amelyeket egy dinamikus halmazban szeretnénk elhelyezni egymás utáni beszúrásokkal. Felmerül a kérdés, vajon hányszor változik meg eközben a halmaz minimális eleme, ha tudjuk, hogy az n szám mind az $n!$ permutációja egyforma valószínűségű? Jelöljük k_i-vel az i-ediknek beszúrt számot, ahol $i = 1, 2, \ldots, n$. Annak valószínűsége, hogy az első i számból éppen k_i lesz a minimum, pontosan $1/i$, mivel az, hogy k_i a $k_1, k_2, \ldots, k_i$ számok közül nagyság szerint hányadik, egyforma valószínűségű a lehetséges sorszámokra. A minimum változásai számának várható értéke tehát:

$$\sum_{i=1}^{n} \frac{1}{i} = H_n.$$

Itt $H_n = \ln n + O(1)$ az n-edik harmonikus szám. A sejtésünk tehát az, hogy a minimum változásainak átlagos száma megközelítőleg $\ln n$, a következő lemma pedig rámutat arra, hogy annak valószínűsége, hogy a várható szám ennél sokkal nagyobb, nagyon kicsi.

Lemma 3.1.2 *Legyen adott n különböző szám valamely $k_1, k_2, \ldots, k_n$ véletlen permutációja, és legyen az $|S|$ valószínűségi változó az*

$$S = \{k_i : 1 \leq i \leq n \text{ és } k_l > k_i \ \forall\ l < i - re\}$$

halmaz elemszáma. Ekkor $P(|S| \geq (\beta+1)H_n) \leq 1/n^2$, ahol H_n az n-edik harmonikus szám és a $\beta \approx 4.32$ szám kielégíti az $(\ln \beta - 1)\beta = 2$ egyenletet.

Bizonyítás. S halmaz elemszámát tekinthetjük úgy, mint amit n számú Bernoulli-kísérlet határoz meg, azaz, ha tekintjük azt az eseményt, hogy k_i a $k_1, k_2, \ldots, k_i$ számok minimuma ($i = 1, 2, \ldots, n$), akkor ez n kísérletből pontosan $|S|$ alkalommal fog bekövetkezni. Az i-edik kísérlet sikerének valószínűsége $1/i$. A kísérletek függetlenek, mivel annak a valószínűsége, hogy k_i a $k_1, k_2, \ldots, k_i$ számok minimuma, független a $k_1, k_2, \ldots, k_{i-1}$ sorrendjétől.

Az $|S| \geq (\beta + 1)H_n$ valószínűségének felső becslésére az alábbi segédtételt alkalmazzuk, amelynek bizonyítása az [6] könyvben található:

Segédtétel 3.1.1 *Tekintsük az n számú Bernoulli-kísérletnek egy olyan sorozatát, ahol a sikeres kísérlet valószínűsége p_i, a sikertelené pedig $q_i = 1 - p_i$, ahol $i = 1, 2, \ldots, n$. Jelölje az X valószínűségi változó a sikeres kísérletek számát, és legyen $\mu = E(X)$. Ekkor $\forall r > \mu$ esetén $P(X - \mu \geq r) \leq (\mu e/r)^r$, ahol e jelöli az Euler-féle számot.*

Jelölje most μ az $|S|$ várható értékét, ekkor $\mu = H_n \leq \ln n$. Mivel $\beta > 1$, alkalmazva a 3.1.1. Segédtételt azt kapjuk, hogy
$P(|S| \geq (\beta + 1)H_n) = P(|S| - \mu \geq \beta H_n) \leq (eH_n/\beta H_n)^{\beta H_n} = e^{(1-\ln\beta)\beta H_n} \leq e^{-(\ln\beta - 1)\beta \ln n} = n^{-(\ln\beta - 1)\beta} = 1/n^2$, ahol felhasználtuk β definícióját. □

A lemmák igazolását követően tételben is megfogalmazhatjuk a véletlen építésű bináris keresőfák várható magasságáról szóló becslésünket.

Tétel 3.1.1 *Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfa átlagos magassága $O(\log_2 n)$.*

Bizonyítás 1. Legyen $k_1, k_2, \ldots, k_n$ az n kulcsérték egy véletlen permutációja, és legyen t az a bináris keresőfa, amelyet ezen kulcsok felsorolás szerinti sorrendben való beszúrásával kapunk. Nézzük annak az eseménynek a valószínűségét, hogy egy adott k_j kulcs $d(k_j, t)$ mélysége legalább akkora, mint egy tetszőlegesen adott ϑ érték. Ha k_j mélysége legalább ϑ, akkor $d(k_j, t)$-nek az 3.1.1 Következményben adott jellemzése alapján G_j és L_j halmazok valamelyikének elemszáma legalább $\vartheta/2$. Így

$$P(d(k_j, t) \geq \vartheta) \leq P(|G_j| \geq \vartheta/2) + P(|L_j| \geq \vartheta/2). \tag{3.1}$$

Vizsgáljuk meg először $P(|G_j| \geq \vartheta/2)$-t. Azt kapjuk, hogy

$$\begin{aligned} P(|G_j| \geq \vartheta/2) &= P(|\{k_i : 1 \leq i < j \text{ és } k_l > k_i > k_j \ \forall\ l < i\text{-re}\}| \geq \vartheta/2) \\ &\leq P(|\{k_i : i \leq n \text{ és } k_l > k_i \ \forall\ l < i\text{-re}\}| \geq \vartheta/2) \\ &= P(|S| \geq \vartheta/2), \end{aligned}$$

ahol $S = \{k_i : 1 \leq i \leq n \text{ és } k_l > k_i \forall l < i - re\}$ az 3.1.2 Lemmában definiáltak szerint. Megjegyezzük, hogy a valószínűséget nem csökkentettük azzal, hogy i értékeinek halmazát $i < j$-ről $i \leq n$-re kiterjesztettük, hiszen ezzel további elemeket vettünk a halmazhoz. Hasonló módon, a $k_i > k_j$ feltétel elhagyásával sem csökkent a valószínűség, mivel így n-nél kisebb elemszámú véletlen permutációt (azon k_i-ket, amelyek nagyobbak k_j-nél) egy bővebb n elemű véletlen permutációra cseréltük. Hasonlóan igazolható, hogy

$$P(|L_j| \geq \vartheta/2) \leq P(|S| \geq \vartheta/2),$$

amiből a (3.1) egyenlőtlenség alapján adódik, hogy

$$P(d(k_j, t) \geq \vartheta) \leq 2P(|S| \geq \vartheta/2).$$

Ha most a speciális $\vartheta = 2(\beta+1)H_n$ értéket választjuk, ahol H_n az n-edik harmonikus szám és $\beta \approx 4.32$ állandó kielégíti az $(\ln \beta - 1)\beta = 2$ egyenletet, akkor a 3.1.2 Lemma felhasználásával a következő eredményt kapjuk:

$$P(d(k_j, t) \geq 2(\beta + 1)H_n) \leq 2P(|S| \geq (\beta + 1)H_n) \leq 2/n^2.$$

Most a véletlen építésű bináris keresőfánkban legfeljebb n csúcs van, annak a valószínűségét, hogy valamelyik csúcs mélysége legalább $2(\beta+1)H_n$ felülről tudjuk becsülni az úgynevezett *Boole-egyenlőtlenség* segítségével, amely az alábbi szabályszerűséget mondja ki: az $A_1, A_2, \ldots$ események bármely véges vagy megszámlálhatóan végtelen sorozata esetén $P(A_1 \cup A_2 \cup \ldots) \leq P(A_1) + P(A_2) + \ldots$. Az említett felső becslés ez alapján $n(2/n^2) = 2/n$, így legalább $1 - 2/n$ a valószínűsége annak, hogy $2(\beta+1)H_n$-nél kisebb magasságú véletlen építésű bináris keresőfát kapunk, és legfeljebb $2/n$ annak a valószínűsége, hogy legfeljebb n magasságút. A várható magasságra tehát az alábbi felső becslés adódik: $(2(\beta + 1)H_n)(1 - 2/n) + n(2/n) = O(\log_2 n)$. □

A bizonyításban szereplő β konstans alapján a véletlen építésű bináris keresőfa átlagos magasságára vonatkozóan egy pontosabb becslés is nyerhető, nevezetesen $2,98 \log_2 n$. Most pedig lássuk, milyen bizonyítást közöl az *Algoritmusok* c. könyv második kiadása [5].

Bizonyítás 2. Első lépésként definiáljunk három valószínűségi változót, hogy mérni tudjuk a véletlen építésű bináris keresőfa magasságát. Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfa magasságát jelöljük X_n-nel, és definiáljuk az *exponenciális magasságot*, ez 2^{X_n}, amit Y_n-nel fogunk jelölni. A harmadik valószínűségi változónk R_n lesz, ami a gyökérnek választott kulcs rangját kapja meg, azaz ha $R_n = i$, akkor a gyökér bal részfája $i-1$, míg jobb részfája $n-i$ kulcsot tartalmaz egy véletlen építésű bináris keresőfában. A véletlen építésű bináris keresőfa definíciójából adódóan R_n egyenlő valószínűséggel veszi fel az $\{1, 2, \ldots, n\}$ halmaz bármely elemét. Mivel egy bináris fa magassága eggyel nagyobb a gyökérhez kapcsolódó két részfa nagyobbikának magasságánál, ezért egy bináris fa exponenciális magassága kétszerese a nagyobbik részfa exponenciális magasságának. Formulával kifejezve, ha tudjuk, hogy $R_n = i$, akkor

$$Y_n = 2 \cdot max(Y_{i-1}, Y_{n-i}).$$

Definíció szerint legyen $Y_0 = 0$, valamint könnyen belátható, hogy az egy kulcsból álló bináris keresőfára $Y_1 = 1$, mivel $2^0 = 1$.

Definiáljuk indikátor valószínűségi változók egy sorozatát, nevezetesen $Z_{n,1}, Z_{n,2}, \ldots, Z_{n,n}$ sorozatot, ahol

$$Z_{n,i} = I\{R_n = i\}.$$

Mivel R_n egyenlő valószínűséggel veszi fel az $\{1, 2, \ldots, n\}$ halmaz bármely elemét, ezért $P(R_n = i) = 1/n$, ahol $i \in \{1, 2, \ldots, n\}$. Ismeretes, hogy esemény indikátor valószínűségi változójának várható értéke megegyezik az esemény valószínűségével, így

$$E(Z_{n,i}) = \frac{1}{n}. \tag{3.2}$$

Mivel $Z_{n,1}, Z_{n,2}, \ldots, Z_{n,n}$ indikátor valószínűségi változók, így értékük 0 vagy 1, sőt,

pontosan egy $Z_{n,i}$ érték lesz 1, a többi pedig 0. Ebből következően

$$Y_n = \sum_{i=1}^{n} Z_{n,i}(2 \cdot max(Y_{i-1}, Y_{n-i})).$$

Megmutatjuk, hogy $E(Y_n)$ egy polinomja n-nek, amelyből következik, hogy $E(X_n) = O(\log_2 n)$.

A $Z_{n,i} = I\{R_n = i\}$ indikátor valószínűségi változó független az Y_{i-1} és Y_{n-i} értékektől. $R_n = i$ választás esetén az Y_{i-1} exponenciális magasságú bal részfába $i-1$ darab i-nél kisebb rangú kulcs kerül véletlenszerűen. Ez a részfa szintén tekinthető úgy, mint egy véletlen építésű bináris keresőfa, amely $i-1$ különböző kulcsot tartalmaz. Ekkor, eltekintve a tartalmazott kulcsok számától, ennek a részfának a struktúrájára egyáltalán nincs hatással az $R_n = i$ választás, azaz Y_{i-1} valószínűségi változó független $Z_{n,i}$-től. Hasonlóan, a jobb részfa $n - i$ különböző, i-nél nagyobb rangú kulcsot tartalmazó véletlen építésű bináris keresőfa, amelynek exponenciális magassága Y_{n-i}. Ennek struktúrája szintén független az R_n értékétől, azaz az Y_{n-i} és $Z_{n,i}$ valószínűségi változók függetlenek. Ekkor pedig fennáll

$$\begin{aligned}
E(Y_n) &= E[\sum_{i=1}^{n} Z_{n,i}(2 \cdot max(Y_{i-1}, Y_{n-i}))] \\
&= \sum_{i=1}^{n} E[Z_{n,i}(2 \cdot max(Y_{i-1}, Y_{n-i}))] \text{ (várható érték linearitása miatt)} \\
&= \sum_{i=1}^{n} E[Z_{n,i}]E[2 \cdot max(Y_{i-1}, Y_{n-i})] \text{ (változók függetlensége miatt)} \\
&= \sum_{i=1}^{n} \frac{1}{n} E[2 \cdot max(Y_{i-1}, Y_{n-i})] \text{ ((3.2) egyenlőség miatt)} \\
&= \frac{2}{n} \sum_{i=1}^{n} E[2 \cdot max(Y_{i-1}, Y_{n-i})] \text{ } (E(aX) = aE(X) \text{ miatt}) \\
&\leq \frac{2}{n} \sum_{i=1}^{n} [E(Y_{i-1}) + E(Y_{n-i})] \text{ } (E(max(X,Y)) \leq E(X) + E(Y) \text{ miatt}).
\end{aligned}$$

Az utolsó összegben az $E(Y_0), E(Y_1), \ldots, E(Y_{n-1})$ tagok kétszer fordulnak elő, egyszer $E(Y_{i-1})$-ként és még egyszer $E(Y_{n-i})$ alakban, tehát

$$E(Y_n) \leq \frac{4}{n} \sum_{i=0}^{n-1} E(Y_i). \tag{3.3}$$

Helyettesítési módszerrel megmutatjuk, hogy minden pozitív egész n esetén a

(3.3) rekurzív egyenlőtlenség megoldása

$$E(Y_n) \leq \frac{1}{4}\binom{n+3}{3}.$$

A bizonyításban felhasználjuk a következő azonosságot:

$$\sum_{i=0}^{n-1}\binom{i+3}{3} = \binom{n+3}{4}, \tag{3.4}$$

amely könnyen belátható az alábbi, binomiális együtthatókra vonatkozó azonosság segítségével, ahol $n > k$ pozitív egészek és

$$\binom{n+1}{k+1} = \binom{n}{k}\binom{n}{k+1}.$$

Az $i = 0$ esetben triviálisan igaz az egyenlőtlenség, mivel $Y_0 = 0$, azaz $E(Y_0) = 0$ és így

$$E(Y_0) \leq \frac{1}{4}\binom{3}{3} = \frac{1}{4}.$$

Ugyanígy $i = 1$ esetben $Y_1 = 1$, azaz $E(Y_1) = 1$ és így

$$E(Y_1) \leq \frac{1}{4}\binom{1+3}{3} = 1.$$

Általános esetben pedig a következőket kapjuk:

$$\begin{aligned}
E(Y_n) &\leq \frac{4}{n}\sum_{i=0}^{n-1} E(Y_i) \\
&\leq \frac{4}{n}\sum_{i=0}^{n-1}\frac{1}{4}\binom{i+3}{3} \text{ (indukciós feltétel miatt)} \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\binom{i+3}{3} \\
&= \frac{1}{n}\binom{n+3}{4} \text{ (a (3.4) egyenlőség szerint)} \\
&= \frac{1}{n}\cdot\frac{(n+3)!}{4!(n-1)!} \\
&= \frac{1}{4}\cdot\frac{(n+3)!}{3!n!} \\
&= \frac{1}{4}\binom{n+3}{3}.
\end{aligned}$$

Ezzel tehát felső korlátot kaptunk $E(Y_n)$-ra, aminek segítségével $E(X_n)$-t is meg tudjuk becsülni, ugyanis $f(x) = 2^x$ konvexitása miatt alkalmazható a várható értékre vonatkozó *Jensen-egyenlőtlenség*, amely szerint

$$2^{E(X_n)} \leq E(2^{X_n}) = E(Y_n),$$

amiből pedig azt kapjuk, hogy

$$\begin{aligned} 2^{E(X_n)} &\leq \frac{1}{4}\binom{n+3}{3} \\ &= \frac{1}{4} \cdot \frac{(n+3)(n+2)(n+1)}{6} \\ &= \frac{n^3+6n^2+11n+6}{24}. \end{aligned}$$

Mindkét oldal logaritmusát véve adódik a tétel állítása, azaz $E(X_n) = O(log_2 n)$. □

Az elemzés azt mutatja tehát, hogy n kulcsú véletlen építésű bináris keresőfa átlagos magassága $O(log_2 n)$, ami egy szótár megengedett műveleteinek elvégzése szempontjából igen nagy hatékonyságot eredményez, vagyis az ilyen módon felépített bináris keresőfák átlagos értelemben nem rosszak. Mivel azonban várható értékről van szó, sajnálatos módon előfordulhatnak a rossz, lineárishoz közelítő esetek is, amelyeket – a későbbiekben majd látni fogjuk – a bináris keresőfák kiegyensúlyozásával tudunk orvosolni.

3.2. Átlagos csúcsmagasság

A megengedett műveletek közül kiemeljük a keresés műveletét, mivel a beszúrás, illetve a törlés is a keresés algoritmusán alapszik, hiszen előbbinél a beszúrandó elem helyét, utóbbinál pedig a törölni kívánt elemet magát keressük meg. Mint azt már láttuk, ennek műveletigénye aszimptotikusan felülről becsülhető a bináris keresőfánk magasságával. Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfára megmutattuk, hogy ez várhatóan $O(log_2 n)$. Kérdés, hogy mennyi lesz egy véletlen építésű bináris keresőfában a keresőutak átlagos hossza? Egy keresőút hossza megegyezik a keresendő elem fán belüli magasságával, tehát úgy is feltehetnénk a kérdést, hogy mennyi lesz egy véletlen építésű bináris keresőfán belüli átlagos csúcsmagasság?

A problémát ismét két irányból, két különböző szakirodalmi forrás alapján közelítjük meg. Az elsőben [7] azt vizsgáljuk, hogy véletlen építésű bináris keresőfa építése során egy beszúrás átlagosan hány összehasonlításba kerül. A másodikban [9] viszont azt vesszük szemügyre, hogy egy véletlen módon felépített bináris keresőfában a szükséges összehasonlítások száma mennyivel nagyobb $log_2 n$-nél, amiről tudjuk,

hogy a legjobb – azaz tökéletesen kiegyensúlyozott – esetben a keresőfánk magassága éppen ennyivel lenne egyenlő.

Tétel 3.2.1 *Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfa építése során egy elem beszúrásához átlagosan legfeljebb $1.39 \log_2 n$ összehasonlítás elegendő.*

Bizonyítás. Tegyük fel, hogy adott $e_1 < e_2 < \cdots < e_n$ kulcsok egy sorozata. Legyen $k_1, k_2, \ldots, k_n$ az n kulcsérték egy véletlen permutációja, és legyen t az a bináris keresőfa, amelyet ezen kulcsok felsorolás szerinti sorrendben való beszúrásával kapunk. Jelölje $t(n)$ a beszúrások során fellépő összehasonlítások átlagos számát. A $t(n)$ mennyiséggel egyben a keresőfa építésének átlagos költségét is mérjük. Mivel n különböző elemnek $n!$ különböző sorrendje lehet, így az átlagot erre az $n!$ lehetséges érkezési sorrendre vesszük, ahol minden érkezési sorrend egyforma valószínűségű. Ekkor tetszőleges i esetén, ahol $i = 1, 2, \ldots, n$, $P(k_1 = e_i) = 1/n$, hiszen minden elem éppen ugyanannyi – nevezetesen $(n-1)!$ – érkezési sorrendnél lesz első. Mindebből az következik, hogy

$$t(n) = \frac{1}{n} \sum_{i=1}^{n} (\text{az olyan fa átlagos költsége, amelyre } k_1 = e_i) \tag{3.5}$$

Egy fa költségén a felépítéséhez használt összehasonlítások számát értjük. Jelölje $K(i)$ a zárójelben szereplő tagot. Ha $k_1 = e_i$, akkor a bináris keresőfa-tulajdonság miatt k_1 bal részfájában $i-1$, jobb részfájában pedig $n-i$ elem lesz, mivel e_i nagyság szerinti sorrendben az i-edik legnagyobb kulcs. Ekkor a baloldali részfában elvégzett összehasonlítások száma $t(i-1)$ lesz, amihez még hozzá kell adnunk az e_i gyökérrel való összehasonlításokat. A baloldali részfa átlagos költsége tehát $i - 1 + t(i-1)$. Hasonló módon adódik, hogy a jobboldali részfa átlagos költsége $n - i + t(n-i)$. A kettő összegzésével megkaphatjuk $K(i)$ értékét, azaz

$$K(i) = i - 1 + t(i-1) + n - i + t(n-i) = n - 1 + t(i-1) + t(n-i).$$

Mindezt visszahelyettesítve a (3.5) egyenletbe, valamint felhasználva azt, hogy $t(0) = 0$, hiszen az üres keresőfa építésének költsége 0, a következő rekurziót kapjuk $n > 0$ esetén:

$$t(n) = n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} t(i).$$

Szorozzuk fel az egyenletet n-nel:

$$nt(n) = n(n-1) + 2\sum_{i=0}^{n-1} t(i), \tag{3.6}$$

majd pedig írjuk fel a formulát n helyett $n-1$-re is, és vonjuk ki a (3.6) egyenletből, ezzel azt kapjuk, hogy

$$nt(n) - (n-1)t(n-1) = 2(n-1) + 2t(n-1).$$

Innen átrendezés és $n(n+1)$-gyel való osztás után a következő egyenlet adódik:

$$\frac{t(n)}{n+1} = \frac{2(n-1)}{n(n+1)} + \frac{t(n-1)}{n},$$

ami egy egyszerű rekurzió a $t(n)/(n+1)$ mennyiségre. Mivel most felső korlátot keresünk, növeljük a jobboldal első tagját úgy, hogy a számlálóba $n-1$ helyett $n+1$-et írunk. Ekkor azt kapjuk, hogy

$$\frac{t(n)}{n+1} < \frac{2}{n} + \frac{t(n-1)}{n}.$$

Ezt ismételten önmagába helyettesítve végül azt nyerjük, hogy

$$\frac{t(n)}{n+1} < \frac{2}{n} + \frac{2}{n-1} + \cdots + \frac{2}{3} + \frac{2}{2} + \frac{2}{1},$$

ahol a jobboldal éppen $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ n-edik harmonikus szám kétszerese. Mint azt már a korábbi alfejezetben említettük, $H_n = \ln n + O(1)$, azaz a következő felső korlátot kapjuk a $t(n)$ mennyiségre:

$$t(n) < 2n \ln n + O(n) \approx 1.39n \log_2 n + O(n),$$

ahol kihasználtuk, hogy $\ln n = \ln 2 \cdot \log_2 n$ és $\ln 2 \approx 0.69$. Egy elem beszúrásához átlagosan tehát valóban legfeljebb $1.39 \log_2 n$ összehasonlítás elegendő, és mivel a becslés érvényes a keresés átlagos költségére, így az átlagos csúcsmélységre is. □

Tétel 3.2.2 *Egy véletlen építésű bináris keresőfa esetén a keresőutak hossza várhatóan* 39 *százalékkal lesz nagyobb, mint egy teljesen kiegyensúlyozott bináris keresőfa esetén.*

Bizonyítás. Az egyszerűség kedvéért legyen most az n különböző kulcs $1, 2, \ldots, n$, és tegyük fel, hogy a kulcsok véletlenszerűen érkeznek. Tudni szeretnénk a keresőutak átlagos a_n hosszát, az átlagot az n kulcsra és az $n!$ féle lehetséges sorrendre nézve. Annak a valószínűsége, hogy az első kulcs, azaz a gyökér kulcsa, i pontosan $1/n$. Az első kulcs baloldali részfája ekkor $i-1$, jobboldali részfája pedig $n-i$ csúcsot tartalmaz. Jelölje a_{i-1} a baloldali részfában, a_{n-i} pedig a jobboldali részfában a keresőutak átlagos hosszát. Az átlagos úthossz egy n csúcsú fában az egyes csúcsok szintjének és hivatkozási gyakoriságának szorzatából vett összeggel fejezhető ki. Ha minden csúcs hivatkozása egyformán valószínű, akkor

$$a_n = \frac{1}{n}\sum_{i=1}^{n} u_i, \tag{3.7}$$

ahol u_i az i csúcs úttávolsága. A csúcsokat ekkor három osztályba sorolhatjuk, ha feltesszük, hogy i kerül a gyökérbe:

1. A baloldali részfa $i-1$ csúcsának átlagos úttávolsága $a_{i-1}+1$.
2. A gyökér úttávolsága 1.
3. A jobboldali részfa $n-i$ csúcsának átlagos úttávolsága $a_{n-i}+1$.

Tehát a (3.7) egyenlet három tag összegéből áll:

$$a_n^{(i)} = (a_{i-1}+1)\frac{i-1}{n} + 1\cdot\frac{1}{n} + (a_{n-i}+1)\frac{n-i}{n}.$$

A keresett a_n mennyiség az $a_n^{(i)}$ érték átlaga lesz, ahol $i = 1, 2, \ldots, n$, tehát az összes $i = 1, 2, \ldots, n$ gyökerű fára véve az átlagot:

$$\begin{aligned} a_n &= \sum_{i=1}^{n}[(a_{i-1}+1)\frac{i-1}{n} + 1\cdot\frac{1}{n} + (a_{n-i}+1)\frac{n-i}{n}] = \\ &= 1 + \frac{1}{n^2}\sum_{i=1}^{n}[(i-1)a_{i-1} + (n-i)a_{n-i}] = \\ &= 1 + \frac{2}{n^2}\sum_{i=1}^{n}(i-1)a_{i-1} \\ &= 1 + \frac{2}{n^2}\sum_{i=1}^{n-1} i\cdot a_i. \end{aligned} \tag{3.8}$$

A (3.8) egy $a_n = f_1(1_1, a_2, \ldots, a_{n-1})$ alakú rekurzív kifejezés a_n értékre, ebből egy egyszerűbb $a_n = f_2(a_{n-1})$ alakú rekurzív reláció nyerhető közvetlenül (3.8) alapján,

nevezetesen

$$(i) \qquad a_n = 1 + \frac{2}{n^2}\sum_{i=1}^{n-1} i \cdot a_i = 1 + \frac{2}{n^2}(n-1)a_{n-1} + \frac{2}{n^2}\sum_{i=1}^{n-2} i \cdot a_i$$

$$(ii) \qquad a_{n-1} = 1 + \frac{2}{(n-1)^2}\sum_{i=1}^{n-2} i \cdot a_i$$

$$(iii) \qquad \frac{2}{n^2}\sum_{i=1}^{n-1} i \cdot a_i = \frac{(n-1)^2}{n^2}(a_{n-1} - 1),$$

ahol (iii)-t úgy kaptuk, hogy (ii)-t megszoroztuk $(\frac{n-1}{n})^2$-nel. Helyettesítsük be (iii)-t (i)-be:

$$a_n = \frac{1}{n^2}((n^2-1)a_{n-1} + 2n - 1).$$

Az n-edik harmonikus függvény segítségével a_n nem rekurzív formában is kifejezhető:

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

Mivel $H_n = \ln n + O(1)$, azért elég nagy n-ekre az adódik, hogy

$$a_n \approx 2\ln n - 3 = 2\ln n - c.$$

Minthogy a legjobb – azaz tökéletesen kiegyensúlyozott – esetben a fa úttávolsága közelítőleg $a'_n = \log_2 n - 1$, így nagy n-re, az elhanyagolható konstansok nélkül számolva azt kapjuk, hogy

$$\lim_{n\to\infty} \frac{a_n}{a'_n} = \frac{2\ln n}{\log_2 n} \approx 1.386.$$

Az eredményből kiolvasható, hogy ha véletlen építésű bináris keresőfa helyett teljesen kiegyensúlyozott keresőfát szeretnénk építeni, akkor a keresőutak hosszának várható javulása legfeljebb 39 százalékos. Hangsúlyozzuk, hogy itt csak várható értékről van szó, a javulás nyilvánvalóan sokkal nagyobb abban az esetben, amikor az előállított fa lineáris listává fajul. Ennek valószínűsége azonban igen csekély, mivel a beszúrandó n kulcs összes permutációja egyformán valószínű. □

A 39 százalékos érték határt szab minden olyan törekvésnek, amely a faszerkezet bármilyen átrendezésével a kulcsok beszúrásában még megtérülhet. Természetesen a csúcsok visszakeresésének és a beszúrások számának aránya döntő ebben a kérdésben, hiszen minél nagyobb ez az arány, annál jobban járunk, ha átrendezzük a

keresőfánkat. A legtöbb alkalmazásban ez a 39 százalék azonban nem olyan számottevő, hogy érdemes lenne módosítani a beszúrás algoritmusát, kivéve, ha a csúcsok száma és a beszúrt elemek hányadosa nagy, vagy ha tartunk a legrosszabb esettől.

4. fejezet

AVL-fák

Az előzőekben láttuk, hogy minél kisebb egy keresőfa magassága, annál jobb becslés adható a keresés idejére. Célunk tehát, hogy ilyen, keresőfa értelemben alacsony fákat építsünk, valamint, hogy a beszúrás és törlés során ezen tulajdonságot megtartsuk. Az olyan fa-konstrukciókat, ahol a fa magassága legfeljebb $c \log_2 n$ és ez a c állandó 1-nél nem sokkal nagyobb, kiegyensúlyozott fáknak szokás nevezni. Egy ilyen kiegyensúlyozási keresőfa-konstrukció az G. M. Adelszon-Velszkij és E. M. Landisz-féle, amelyet a következőképpen definiálunk:

Definíció 4.0.1 *Egy bináris keresőfa akkor és csak akkor AVL-fa, ha minden x csúcsára teljesül, hogy az x-hez tartozó bal-, és jobboldali részfa magasságának különbsége legfeljebb* 1.

A definícióból adódóan tehát egy olyan bináris keresőfát kapunk, amelyben egy csúcs bal-, és jobboldali részfájának magassága közel egyenlő. Ez valóban kiegyensúlyozott keresőfa lesz, mivel legfeljebb 44 százalékkal magasabb, mint egy azonos csúcsszámú, tökéletesen kiegyensúlyozott bináris keresőfa. Ezt az állítást tételben is megfogalmazzuk, a bizonyítást a [9]-ben szereplő módon végezzük.

Tétel 4.0.3 *Egy n különböző kulcsból álló AVL-fa magassága legfeljebb* $1.44 \log_2 n$.

Bizonyítás. Próbáljunk olyan AVL-fát konstruálni, amelynek a struktúrája a legrosszabb, azaz a magassága a legnagyobb a többi AVL-fáéhoz képest. Az n csúcsú AVL-fák közül a maximális h magasságú kiválasztásához tekintsük h-t rögzítettnek,

majd konstruáljunk olyan h magasságú AVL-fát, amelynek minimális a csúcsszáma. Azért célszerű ezt a stratégiát választani, mivel rögzített h magasság csak speciális n-ekre érhető el. Jelölje t_h a h magasságú, minimális csúcsú AVL-fát. Nyilván t_0 az üres fa, t_1 pedig az egypontú fa, azaz csak a gyökérből áll. A t_h-t úgy állítjuk elő $h >$ 1-re, hogy a gyökérhez minimális csúcsszámú részfákat illesztünk. Ez azt jelenti tehát, hogy a részfák is minimális csúcsszámú fák. Ekkor az egyik részfa magassága $h-1$, a másiké pedig $h-2$. Ez a konstrukciós elv hasonlatos a Fibonacci számok képzéséhez, éppen ezért ezeket a fákat *Fibonacci-fáknak* nevezzük, pontos definíciójuk:

Definíció 4.0.2 *Az üres fa a* -1 *magasságú, az egypontú fa a* 0 *magasságú Fibonacci-fa. Ha* t_{h-1} *és* t_{h-2} $h-1$*, valamint* $h-2$ *magasságú Fibonacci-fák, akkor* $t_h = (t_{h-1}, x, t_{h-2})$ *egy* h *magasságú Fibonacci-fa. Más Fibonacci-fa nincs.*

Jelölje t_h csúcsainak számát N_h, ekkor $N_{-1} = 0$ és $N_0 = 1$ triviálisan, $h > 0$ esetén pedig a következő rekurzív kifejezés adódik:

$$N_h = 1 + N_{h-1} + N_{h-2}$$

$$N_h + 1 = N_{h-1} + 1 + N_{h-2} + 1$$

Legyen $G_h = G_{h-1} + G_{h-2}$, ahol $G_h = N_h + 1$. Ekkor G_h a 3-mal "eltolt" Fibonacci-sorozat, és így

$$N_h = G_h - 1 = F_{h+3} - 1.$$

Ismeretes, hogy

$$F_k = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^k - (\frac{1-\sqrt{5}}{2})^k),$$

ezt behelyettesítve a következőket kapjuk:

$$\begin{aligned} N_h = G_h - 1 &= \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^{h+3} - (\frac{1-\sqrt{5}}{2})^{h+3}] - 1 \\ &\geq \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^3(\frac{1+\sqrt{5}}{2})^h - 2 \\ &= \frac{1}{\sqrt{5}} \cdot \frac{1+3\sqrt{5}+3\cdot 5+5\sqrt{5}}{8}(\frac{1+\sqrt{5}}{2})^h - 2 \\ &= \frac{2+\sqrt{5}}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^h - 2 \\ &= (\frac{1+\sqrt{5}}{2})^h + \frac{2}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^h - 2 \\ &\geq (\frac{1+\sqrt{5}}{2})^h. \end{aligned}$$

Ebből pedig:

$$\log_2 N_h \geq \log_2(\frac{1+\sqrt{5}}{2})^h$$
$$\log_2 N_h \cdot \frac{1}{\log_2(\frac{1+\sqrt{5}}{2})^h} \geq h,$$

és mivel $\frac{1}{\log_2(\frac{1+\sqrt{5}}{2})^h} \approx 1.4404$, így valóban $h \leq 1.44 \log_2 n$. □

Mi történik azonban, ha egy AVL-kiegyensúlyozott fába új elemet szúrunk be, vagy éppen ellenkezőleg, egy csúcsát töröljük? A keresőfa efféle módosítása során sérülhet a kiegyensúlyozottsági feltétel, amivel a fa magasságára vonatkozó $1.44 \log_2 n$-es becslés érvényét veszti. Szükségünk van tehát olyan kiegyensúlyozó eljárásokra, amelyek a keresőfa szerkezetét beszúrás és törlés után helyreállítják.

4.1. Beszúrás AVL-fába

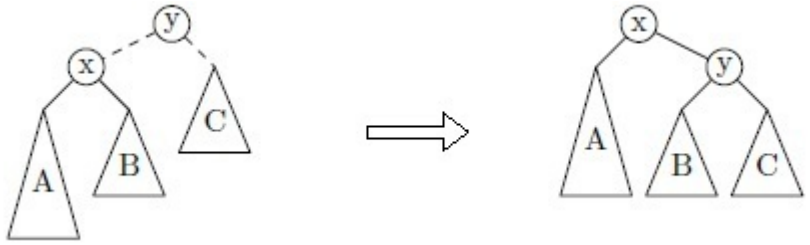
Jelölje g az AVL-fa gyökerét, B a baloldali, J pedig a jobboldali részfáját, h pedig a famagasságot. Tegyük fel, hogy a beszúrt elem B-be kerül és 1-gyel növeli B magasságát. A beszúrás elvégzése után három esetet különböztetünk meg:

1. $h_B = h_J$ volt a beszúrás előtt, az egyenlőség megszűnik, de az AVL-kiegyensúlyozottság megmarad

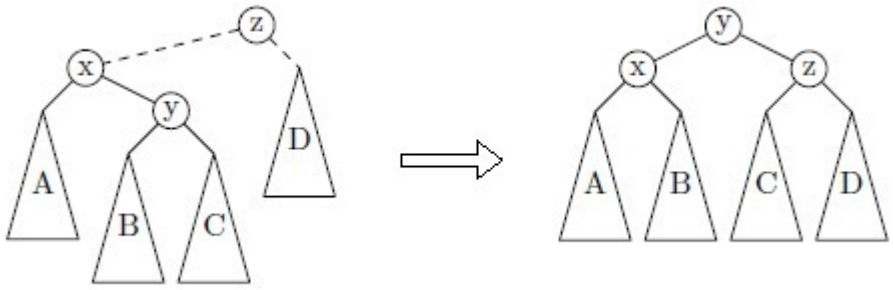
2. $h_B < h_J$ volt a beszúrás előtt, B és J magassága egyenlővé válik, az AVL-kiegyensúlyozottság erősödik

3. $h_B > h_J$ volt a beszúrás előtt, az AVL-kiegyensúlyozottság elromlik, a fát át kell alakítani

Definíció 4.1.1 *Azokat az eljárásokat, amelyekkel egy bináris keresőfa AVL-kiegyensúlyozottságát állítjuk vissza, forgatásoknak nevezzük. Alapvetően két forgatástípust különböztetünk meg, az egyszerű – jobb, illetve szimmetrikus esetben bal – forgatást, illetve a kettős forgatást. A forgatásokat ábrán szemléltetjük:*



4.1. ábra. *Egyszerű forgatás*



4.2. ábra. *Kettős forgatás*

Fontos tehát, hogy a keresőfán csak függőleges irányú mozgást szabad végezni, a csúcsok és részfák relatív vízszintes helyzete változatlan kell maradjon, máskülönben sérülne a bináris keresőfa-tulajdonság. Kérdés, hogy hogyan valósíthatjuk meg ezen forgatásokat? A megoldást egy úgynevezett egyensúlyi faktor bevezetése adja meg, amelyet minden csúcsra külön kiszámítunk, és ami a bal-, illetve jobboldali részfa különbségét tárolja. Ezek után az AVL-fába való beszúrás a következőképpen fog kinézni:

1. A keresés algoritmussal megkeressük a beszúrandó elem helyét, megbizonyosodunk róla, hogy még nem szerepel a fában.

2. Beszúrjuk az elemet, kiszámoljuk a hozzá tartozó egyensúlyi faktort.

3. A keresőút mentén leellenőrizzük minden csúcsra, miként módosultak az egyensúlyi faktorok.

Ez a módszer azonban elvégez néhány felesleges ellenőrzést is, hiszen az egyensúly észlelése után már nem kell az elődöket végigvizsgálni, éppen ezért bevezetünk egy újabb paramétert, nevezetesen egy l logikai változót, amelynek jelentése: "a részfa növekedett".

Tegyük fel, hogy a p mutatóval jelzett csúcshoz a baloldali ágról az a jelzés kerül vissza, hogy a részfa magassága növekedett. Ekkor ismét három esetet különböztetünk meg:

1. $h_B < h_J$ volt a beszúrás előtt, p egyensúlyi faktora 1, most tehát p-ben beáll a teljes egyensúly;

2. $h_B = h_J$ volt a beszúrás előtt, p egyensúlyi faktor 0, az egyensúly balra billen;

3. $h_B > h_J$ volt a beszúrás előtt, p egyensúlyi faktora -1, az AVL-kiegyensúlyozottság elromlik, a fát át kell alakítani.

A harmadik esetben a p csúcshoz tartozó baloldali részfa gyökerének egyensúlyi faktora mondja meg, hogy a két forgatástípusból melyiket kell alkalmazni. Ha ennek a csúcsnak szintén magasabb a baloldali részfája, mint a jobboldali, akkor jobb forgatást, egyébként pedig kettős forgatást alkalmazunk. Az AVL-kiegyensúlyozottság visszaállításához szükséges forgatások elvégezhetők a mutatók átállításával, az egyszerű forgatás megoldható hat, a kettős forgatás pedig tíz pointer átállításával. Ekkor természetesen az érintett csúcsok egyensúlyi faktorát is újra ki kell számolni.

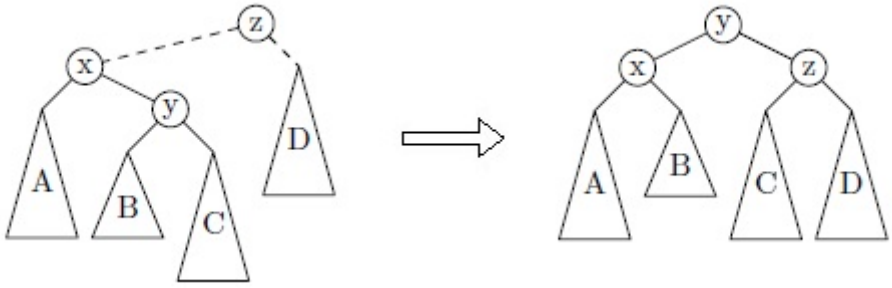
Tétel 4.1.1 *Egy n különböző kulcsból álló AVL-fába való beszúrást követően legfeljebb egy forgatással helyreállítható az AVL-kiegyensúlyozottság. A beszúrás költsége ezzel együtt is $O(\log_2 n)$.*

A tétel bizonyítása a [7] könyvben található.

Az AVL-fák beszúró algoritmusának hatékonyságával kapcsolatosan felmerül a kérdés, vajon mi a valószínűsége annak, hogy a beszúrást követően kiegyensúlyozást kell végrehajtani? A tapasztalati eredmények azt mutatják, hogy átlagosan minden második beszúrást követően van szükség kiegyensúlyozásra. Hozzátesszük, hogy itt az egyszerű, illetve a kettős forgatás valószínűsége megegyezik. Ez azt jelenti, hogy az AVL-fák gyakorlatilag ugyanolyan jók, mint tökéletesen kiegyensúlyozott társaik, fenntartásuk ráadásul sokkal egyszerűbb.

4.2. Törlés AVL-fából

Egy elem törlését követően a keresőfa AVL-kiegyensúlyozottságának visszaállítása lényegében ugyanúgy zajlik, mint ahogyan azt beszúrás esetén láttuk, a kiegyensúlyozó eljárások, azaz a forgatások megegyeznek a előbb látottakkal, valamint kiegészülnek egy harmadik forgatás-típussal, amelynek során a keresőfa magassága csökken:



4.3. ábra. *Harmadik forgatás törlés során*

A szélső csúcsok és az egyetlen utóddal rendelkező csúcsok esete könnyen kezelhető, előbbi esetben egyszerűen töröljük az elemet, utóbbiban pedig a gyerekével helyettesítjük. Ha pedig a törölni kívánt csúcsnak két gyereke van, akkor azzal a legközelebbi rákövetkezőjével helyettesítjük, amelynek nincs baloldali gyereke. A törlés folyamán bevezetünk egy l logikai változót, amelynek jelentése: "a részfa magassága csökkent". Kiegyensúlyozás akkor történik, ha l értéke igaz, ami akkor fordulhat

elő, ha éppen töröltünk egy elemet, vagy pedig a kiegyensúlyozás során csökkent valamelyik részfa magassága.

Tétel 4.2.1 *Egy n különböző kulcsból álló AVL-fából való törlést követően legfeljebb $O(\log_2 n)$ forgatás helyreállítja az AVL-kiegyensúlyozottságot. A törlés költsége ezzel együtt is $O(\log_2 n)$.*

Akad azonban egy lényeges különbség a beszúrás és törlés eljárások között. Míg az egyes kulcsok beszúrása legfeljebb egy forgatást tesz szükségessé, addig törléskor a keresőút minden elemén szükség lehet forgatásra. Tekintsük például az előzőekben definiált Fibonacci-fák esetét. Ilyenkor minden egyes csúcs törlése csökkenti a fa magasságát, a legalacsonyabban fekvő szélső csúcs törlése pedig a lehető legtöbb forgatást vonja maga után. De mekkora a törlések utáni forgatások valószínűsége általában? A tapasztalati eredmények alapján azt mondhatjuk, hogy amíg körülbelül minden második beszúrásnál kell forgatást végezni, addig csupán minden ötödik törlésnél lesz szükség forgatásra. A törlés az AVL-fákon tehát körülbelül olyan nehézségű művelet, mint a beszúrás.

Az AVL-fákról elmondható tehát, hogy a keresés, beszúrás és törlés műveletigénye egyaránt $O(\log_2 n)$ nagyságrendű, még a legrosszabb esetben is, azaz igen jó önkiegyensúlyozó fastruktúrák. A keresőfák csúcsait azonban erősen tömörített rekordokkal szokás ábrázolni a memóriatárral való takarékosság miatt, éppen ezért az AVL-fákat abban az esetben érdemes használni, ha a keresés műveletét jóval gyakrabban kívánjuk futtatni, mint a beszúrását, hiszen a kiegyensúlyozó eljárások összetettsége miatt sok pluszinformációt is el kell tárolnunk az egyes csúcsokhoz tartozó rekordokban, ami sokat ronthat a hatékonyságon.

5. fejezet

Piros-fekete fák

A bináris keresőfák kiegyensúlyozottságát természetesen értelmezhetjük az előző fejezetben látottaktól eltérő módon is. Tekintsük például a következő, [8]-beli definíciót:

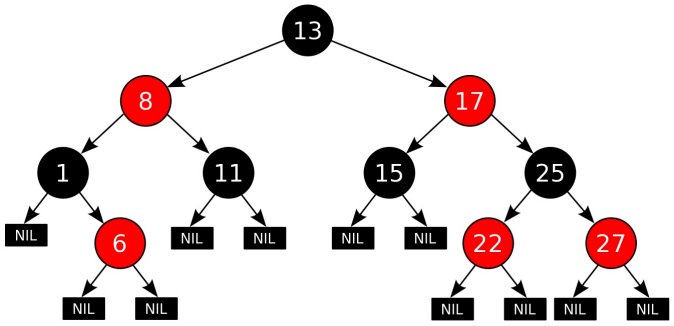
Definíció 5.0.1 *Egy bináris keresőfa minden x csúcsához rendeljünk hozzá egy egész számot, amelyet $rang(x)$-szel jelölünk, és legyen x szülője $p(x)$, nagyszülője pedig $p^2(x)$. A bináris keresőfát kiegyensúlyozottnak nevezzük, ha minden x-re teljesülnek az alábbi kritériumok:*

1. *Ha x-nek van szülője, akkor $rang(x) \leq rang(p(x)) \leq rang(x) + 1$.*
2. *Ha x-nek van nagyszülője, akkor $rang(x) < rang(p^2(x))$.*
3. *Ha x egy szélső csúcs, akkor $rang(x) = 0$ és $rang(p(x)) = 1$, ha x-nek van szülője.*

Ezen definíció alapján eljuthatunk a piros-fekete fa fogalmához, amely R. Bayertől származik, nevét viszont L. J. Guibas és R. Sedgewick tette használatossá, és amely a csúcsok pirossal és feketével való színezésére utal. A színezési szisztéma a következőképpen néz ki: legyen x fekete, ha $rang(p(x)) = rang(x) + 1$, vagy $p(x) = NIL$, illetve legyen x piros, ha $rang(p(x)) = rang(x)$. A színezés alapján tehát egy alternatív definíciót fogalmazhatunk meg a kiegyensúlyozott bináris keresőfákat illetően:

Definíció 5.0.2 *Egy bináris keresőfa piros-fekete fa, ha rendelkezik a következő piros-fekete tulajdonságokkal:*

1. *Minden csúcs színe piros vagy fekete.*
2. *Minden levél színe fekete.*
3. *Minden piros csúcsnak mindkét fia fekete.*
4. *Bármely két, azonos csúcsból induló és levélig vezető út ugyanannyi fekete csúcsot tartalmaz.*



5.1. ábra. Példa piros-fekete fára

Szokás még félig kiegyensúlyozott fának is nevezni, mivel a csúcsok színezésének korlátozásával biztosítható, hogy a keresőfában minden, a gyökértől levélig vezető út hossza nem nagyobb, mint a legrövidebb ilyen út hosszának kétszerese. Ennek tudatában az alábbi, [6]-ban megtalálható tétel mondható ki a piros-fekete fa magasságáról:

Tétel 5.0.2 *Bármely n belső csúcsot tartalmazó piros-fekete fa magassága legfeljebb* $2\log_2(n+1)$.

Bizonyítás. Jelölje $fm(x)$ egy x csúcs fekete-magasságát, amely az x csúcsból induló, levélig vezető úton található, x-et nem tartalmazó fekete csúcsok száma. Teljes indukcióval megmutatjuk, hogy a piros-fekete fa minden x gyökerű részfája legalább $2^{fm(x)}-1$ belső csúcsot tartalmaz. Az indukciót x levelektől való távolsága

szerint hajtjuk végre: ha x ilyen módon meghatározott "magassága" 0, akkor x levél, tehát az x gyökerű részfának valóban $2^{fm(x)}-1 = 2^0-1 = 0$ belső csúcsa van. Tegyük fel, hogy x-nek a levelektől való távolsága pozitív, és x-nek két gyereke van. Ekkor attól függően, hogy a gyerekek színe piros, vagy fekete, a gyerekek levelektől való távolsága $fm(x)$ vagy $fm(x)-1$, de mindenképpen kisebb, mint x-é. Az indukciós feltevés szerint ekkor mindkét részfa legalább $2^{fm(x)-1}-1$ belső csúcsot tartalmaz. Tehát az x gyökerű részfa belső csúcsainak száma legalább $2^{fm(x)-1}-1+2^{fm(x)-1}-1+1 = 2^{fm(x)}-1$.

Jelölje a piros-fekete fa magasságát h. A 3. piros-fekete tulajdonság miatt minden olyan út, amelyik a gyökértől egy levélig halad, legalább fele annyi fekete csúcsot tartalmaz, mint ezen út csúcsainak száma, nem számítva a gyökeret. A gyökér feketemagassága így legalább $h/2$, azaz $n \geq 2^{h/2}-1$, amiből $\log_2(n+1) \geq h/2$, és így $h \leq 2\log_2(n+1)$. $\square$

A tételből láthatjuk, hogy a keresés, beszúrás és törlés műveletigénye a kiegyensúlyozó eljárásokkal együtt is $O(\log_2 n)$, tehát a piros-fekete fa valóban hatékony adattárolási struktúrát ad. Megjegyezzük, hogy piros-fekete fák esetén az alábbi kiegyensúlyozó eljárásokat szokás használni: egyszerű, vagy kettős forgatás, amelyek a korábbi fejezetben látottakhoz hasonló elven működnek. Most vizsgáljuk meg részletesebben az alapműveleteket, hogy jobban megértsük a piros-fekete fák felépítését. A továbbiakban a [8]-ból megismert szemléletet követjük.

5.1. Beszúrás piros-fekete fába

Beszúrás során az általunk hozzáadni kívánt elemet egy NIL levél helyére szúrjuk be, amelynek rangja a beszúrás előtt 0 volt, a beszúrást követően pedig 1 lesz. Jelöljük most ezt az elemet x-szel, ekkor tehát $rang(x) = 1$. Az eljárás közben sérülhet a 3. piros-fekete tulajdonság, amennyiben egy olyan piros csúcsot hozunk létre, amelynek a szülője is piros. Hogy visszaállítsuk a piros-fekete fa megfelelő szerkezetét, leellenőrizzük, hogy x nagyszülőjének két piros gyereke van-e. Ha igen, megnöveljük $p^2(x)$ rangját 1-gyel, kicseréljük x-et $p^2(x)$-szel, az új rang-kapcsolatoknak megfelelően átszínezzük az érintett csúcsokat, és megnézzük, még mindig sérül-e

a 3. piros-fekete tulajdonság. Ha azonban $p^2(x)$-nek van fekete színű gyereke, akkor végrehajtunk egy egyszerű, vagy egy kettős forgatást, amely helyreállítja a keresőfa kiegyensúlyozottságát. Ezzel a módszerrel a beszúrást egy sor rangnövelés, és legfeljebb két egyszerű – azaz egy kettős – forgatás követi.

5.2. Törlés piros-fekete fából

Egy elem törlésekor a törölt elemet egy 1-gyel kisebb rangú elemmel helyettesítjük, jelöljük ezt x-szel. Előfordulhat, hogy $rang(x)$ 2-vel kisebb, mint $rang(p(x))$, amely sértené az 5.0.1 Definíció 1. pontját, ezért ekkor a fa szerkezetének helyreállítására volna szükség. Jelölje y az x testvérét, ekkor két esetet különböztetünk meg:

1. eset: y fekete

   (a) Ha y mindkét gyereke fekete, akkor csökkentjük $p(x)$ rangját 1-gyel, x-et kicseréljük $p(x)$-szel, átszínezzük az érintett csúcsokat, és leellenőrizzük, még mindig sérül-e az 1. pont.

   (b) Ha y x-től legtávolabbi gyereke piros, akkor $p(x)$-nél egy egyszerű forgatást hajtunk végre és készen vagyunk.

   (c) Ha y x-hez legközelebbi csúcsa piros, és annak testvére fekete, akkor $p(x)$-nél egy kettős forgatást hajtunk végre és készen vagyunk.

2. eset: y piros

   Ekkor y mindkét gyereke fekete a 3. piros-fekete tulajdonság miatt. $p(x)$-nél egy egyszerű forgatást hajtunk végre, majd innentől ugyanúgy, mint az 1. esetben, hiszen a forgatást követően x új testvére fekete lesz.

Ezzel a módszerrel a törlést egy sor rangcsökkentés és legfeljebb három forgatás követi.

5.3. Egyesítés piros-fekete fákon

Az egyesítés művelet inputja két dinamikus halmaz, azaz piros-fekete fa, jelölje ezeket S_1 és S_2, valamint egy x elem, outputja pedig az $S = S_1 \cup \{x\} \cup S_2$ halmaz.

Célravezető, ha az egyesítendő fáink gyökerét a rangjukkal azonosítjuk, majd lefelé haladva $O(1)$ idő alatt kiszámítjuk a többi csúcs rangját is. Az egyesítéshez hasonlítsuk össze S_1 és S_2 rangját. Ha $rang(S_1) \geq rang(S_2)$, követjük a jobboldali mutatókat S_1-től egészen egy olyan q csúcsig, amelyre $rang(q) = rang(S_2)$, majd q-t és az ő részfáját kicseréljük x-re, ezzel x az S_2 baloldali, valamint q jobboldali gyereke lesz. Az x csúcs rangját $rang(q)+1$-nek definiáljuk, ezzel x és S_2 is fekete lesz. Ezek után helyreállítjuk a fa szerkezetét, pontosan úgy ahogy a beszúrás során, x-től indulva. A $rang(S_1) < rang(S_2)$ eset szimmetrikus. Az egyesítés művelete $O(|rang(S_1) - rang(S_2)|) = O(\log_2 n)$ műveletigényű és egy olyan fát hoz létre, amelynek gyökerére a rang $\max\{rang(S_1), rang(S_2)\}$, vagy $\max\{rang(S_1), rang(S_2)\} + 1$, utóbbi esetben a gyökér mindkét gyereke fekete.

Láthatjuk tehát, hogy az AVL-fákhoz hasonlóan, itt is szükséges egy, vagy több egyensúlyi érték tárolása a csúcsokban, különben nem tudnánk megfelelően alkalmazni a kiegyensúlyozó eljárásokat. A piros-fekete fák alapjában véve nagyon hasonlítanak az AVL-fákhoz, viszont másfajta forgatásokat használnak a kiegyensúlyozásra, valamint egy kicsivel nagyobb kiegyensúlyozatlanságot engednek meg. Mindezek ellenére, ahogy már fentebb említettük, ugyanúgy biztosítják az $O(\log_2 n)$ idejű keresést, beszúrást és törlést, tehát igen jó struktúrájú bináris keresőfáknak mondhatók.

6. fejezet

Egy önszervező bináris keresőfa: az S-fa

Az eddigi hatékony keresőfák, mint az AVL-fa vagy a piros-fekete fa, folyamatosan ügyeltek a fa alakjának, precízebben kiegyensúlyozottságának megtartására, ezzel biztosítva a kedvező $O(\log_2 n)$-es műveletigényt. Most azonban egy olyan önszervező struktúrát vizsgálunk meg, amely más módon tartja fent a keresőfa hatékonyságát, teljesen figyelmen kívül hagyva a fa alakját. Elsőként B. Allen, I. Munro és J. R. Bitner próbálkoztak ilyen struktúra kidolgozásával, de csak $O(n)$-es futásidőt tudtak elérni, D. D. Sleator és R. E. Tarjan fedezték fel azt az önszervező bináris keresőfa-típust, amellyel most foglakozni fogunk, és amely már a kívánt $O(\log_2 n)$-es futásidőt eredményezi. Ezt a konstrukciót *splay tree*-nek nevezték el, a magyar szakirodalomban S-fának [7], vagy ferde fának szokás nevezni.

Az S-fa azon az egyszerű elven alapszik, hogy a gyakrabban keresett, vagy használt elemek magasabban helyezkednek el a keresőfán belül, mint kevésbé "fontos" társaik. Szemléletes példaként megemlíthetjük egy kórház betegnyilvántartását, ahol a naponta, vagy hetente visszajáró betegek adatai magasabban helyezkednek el a keresőfában, mint azoké, akik mondjuk csak éves kontrollra járnak. Ekkor tehát, ha keresést végzünk a fán, nagyobb valószínűséggel egy rövidebb keresőutat kell csak bejárnunk, azaz a műveletigény ebben az esetben nem lesz túl nagy.

Ezen egyszerű elv megvalósítását a következő mechanizmus biztosítja: ha a fában tárolt x elemre rákeresünk, akkor az algoritmus ezt úgy értelmezi, hogy x fontossága

nőtt, ezért x-et forgatások segítségével a fa gyökerébe mozgatja. Ezen felül az x-hez közeli elemek is fontosabbak lesznek, a módszer őket is a gyökér felé közelíti, tehát a végeredmény egy olyan keresőfa lesz, amelyben a gyakran használt elemek a gyökér közelében, a ritkán használtak pedig a levelekben, illetve azok közelében lesznek.

Az említett mechanizmus megvalósítása az úgynevezett *splay* operáció [8], amelynek működése viszonylag egyszerűen leírható. Legyen x egy belső csúcs az S-fában, amelyet a gyökérbe szeretnénk mozgatni. Az x csúcsból kiindulva visszafelé lépkedünk a gyökérig, miközben egyes csúcsoknál egyszerű forgatást hajtunk végre. Ezeket a forgatásokat párban végezzük, olyan sorrendben, amely a fa szerkezetéből adódik. Jelölje $p(x)$ az x szülőjét, $p^2(x)$ pedig az x nagyszülőjét. Ekkor a splay operáció a következő lépések ismételt elvégzéséből áll:

1. Ha x-nek van szülője, de nincs nagyszülője, akkor $p(x)$-nél forgatunk.

2. Ha x-nek van nagyszülője, valamint x és $p(x)$ is jobboldali – vagy baloldali – gyerekek, akkor $p^2(x)$-nél forgatunk.

3. Ha x-nek van nagyszülője, valamint x baloldali, $p(x)$ pedig jobboldali gyerek – vagy fordítva –, akkor először $p(x)$-nél, majd x új szülőjénél – azaz az eredeti $p^2(x)$-nél – forgatunk.

Mindezek hatására x az S-fa gyökerébe kerül, miközben az eredetileg x-be vezető, gyökérből induló út újrarendeződik. A splay operációt természetesen nem csak egy elem keresésekor, de beszúráskor, vagy törléskor is használjuk. Beszúrás esetén a beszúrt elemnél alkalmazzuk a splay műveletét, törléskor pedig a törlendő elem szülőjénél. Az S-fákon értelmezünk egy szétvágó és egy egyesítő műveletet is, előbbi az eredeti fát kisebb S-fákra bontja, utóbbi pedig az adott S-fákból egyetlen S-fát szervez. Ha egy x elemnél kívánunk szétvágni, akkor előbb elvégzünk x-nél egy splay műveletet, mivel ekkor x a gyökérbe kerül, tehát a szétvágás eredménye a hozzá tartozó baloldali, illetve a jobboldali részfa lesz. Minden egyes művelet esetén a futásidő az úgynevezett *splay út* hosszával arányos, amely azon út, amely mentén a splay működik.

Tétel 6.0.1 *Egy üres S-fából induló olyan m műveletből álló sorozat költsége, amelyben n beszúrás és egyesítés van, $O(m \log_2 n)$.*

A tétel bizonyítása a [8] könyvben található. Az $O(m \log_2 n)$-es becslés valóban kedvező műveletigényt biztosít a keresőfa konstrukciója során, így tehát abban az esetben, ha az implementált adathalmazon túlnyomórészt csak keresést akarunk végrehajtani, érdemes S-fákat alkalmazni.

7. fejezet

A kiegyensúlyozás egy naiv módszere: az RBS-fa

A bináris keresőfák kiegyensúlyozásának egy másik módszere az az alábbiakban bemutatott fastruktúra, amely Fekete Istvántól [4] származik. A módszer alapvetően egy játékos elgondolás alapján született, ám egy Giachetta Roberto által megírt program [3], amely nem csak megvalósítja ezt a faszerkezetet, de statisztikai elemzést is végez rajta, mégis azt mutatja, hogy hatékonyságát tekintve meglepően jól működik. A fa a *Rebuilding Binary Search tree* elnevezés alapján az RBS-fa nevet viseli, célja, hogy megőrizze a bináris keresőfa $O(\log_2 n)$-es magasságát, mivel tudjuk, hogy ebben az esetben a keresőfa kellően hatékonynak mondható. De hogyan is működik a módszer?

Építsünk fel véletlen beszúrások, illetve törlések sorozatával egy bináris keresőfát n különböző kulcsból, ahol a kulcsok eloszlása egyenletes. Jelölje t_n a keresőfát, $h(t_n)$ pedig a fa magasságát, valamint rögzítsünk le egy nem túl nagy, 1-nél nagyobb vagy azzal egyenlő c konstanst. Ekkor az RBS-fa minden egyes lépésben ellenőrzi a famagasságot, és annak eredménye szerint jár el:

(a) Ha $h(t_n) \leq c \log_2 n$, akkor az eljárás nem tesz semmit.

(b) Ha $h(t_n) > c \log_2 n$, romlott a kiegyensúlyozottság, tehát átszervezésre van szükség.

A (b) esetbeli átszervezés két lépésből áll. Az első lépésben a keresőfa inorder be-

járásával nagyság szerint növekvő sorrendben kiírjuk a kulcsokat egy tömbbe. A második lépésben a tömb elemeit a bináris keresés mechanizmusához hasonló módon, sorozatos felezések szerint indexelve, szintenként felépítjük az új keresőfát, ennek magassága pedig már minimális lesz.

Az átszervezés lineáris idejű, ráadásul a kulcsok egyenletes eloszlása szerint a fa csak nagyon ritka esetekben lesz rosszul kiegyensúlyozott, tehát ha c-t alkalmasan választjuk meg, akkor átszervezésre is csak ritkán kerül sor, azaz a műveletigény alacsony lesz, közelíteni fogja, esetleg meg is haladja az AVL-fa hatékonyságát.

Ez a módszer azonban magában hordoz néhány nem elhanyagolható problémát. Egyrészt, elveszíti a valós idejűséget, azaz előfordulhat, hogy hosszabb ideig egyáltalán nem csinál semmit, aztán egyszerre sokat dolgozik. Másrészt, elméleti számítások helyett csak a tapasztalat oldaláról, számítógépes szimulációval sikerült alátámasztani az RBS-fa hatékonyságát, pontos matematikai elemzése még nyitott terület. Ráadásul, mivel a problématér $n!$ méretű, ezért csak mintavételezésre van alkalmunk, ahol az $n = 12$ egy lehetséges határesetet valósít meg, mivel a 12! még integer-ként ábrázolható, az összes esetben történő faépítés még kivárható egy asztali számítógépen.

Mivel a gyakorlatban mégis könnyen alkalmazható abban az esetben, ha átszervezés ritkán fordul csak elő, ezért megvizsgáljuk, hogy általában c milyen értékeire teljesül ez a kívánalom. Érdekes kérdés az is, hogy milyen c-re nem lesz egyáltalán átszervezés? Mindezeket a kérdéseket a következő fejezetben válaszoljuk meg, ahol ezen túl összehasonlítjuk az RBS-fára kapott eredményeket az AVL-fa eredményeivel, majd ezek alapján egy szigorúan tapasztalaton alapuló véleményt fogalmazunk meg.

8. fejezet

Futási eredmények

Az alábbiakban a már említett program segítségével végzett mintavételezések eredményeit közöljük. Alapvetően négy kérdésre kerestük a választ. Egyrészt, hogy a véletlen építésű bináris keresőfák átlagos magasságára vonatkozó $2,98 \log_2 n$-es becslés mennyire helytálló, másrészt, hogy az átlagos csúcsmagasságot megbecsülni kívánó $1,39 \log_2 n$ kifejezés valóban jól tükrözi-e a gyakorlatban tapasztalt eredményeket.

Hasonlóan, megvizsgáltuk, hogy az AVL-fák esetében mennyire szigorú a famagasságra vonatkozó $1,44 \log_2 n$ felsőbecslés, vagyis, hogy valóban indokolt-e ekkora konstans szorzót használni.

Végül, az RBS-fákkal kapcsolatos kérdésekre kerestük a választ, azaz, hogy milyen c konstans esetén közelítik az AVL-fák hatékonyságát, illetve, hogy c mekkora értéke esetén nem fordul elő egyáltalán átszervezés.

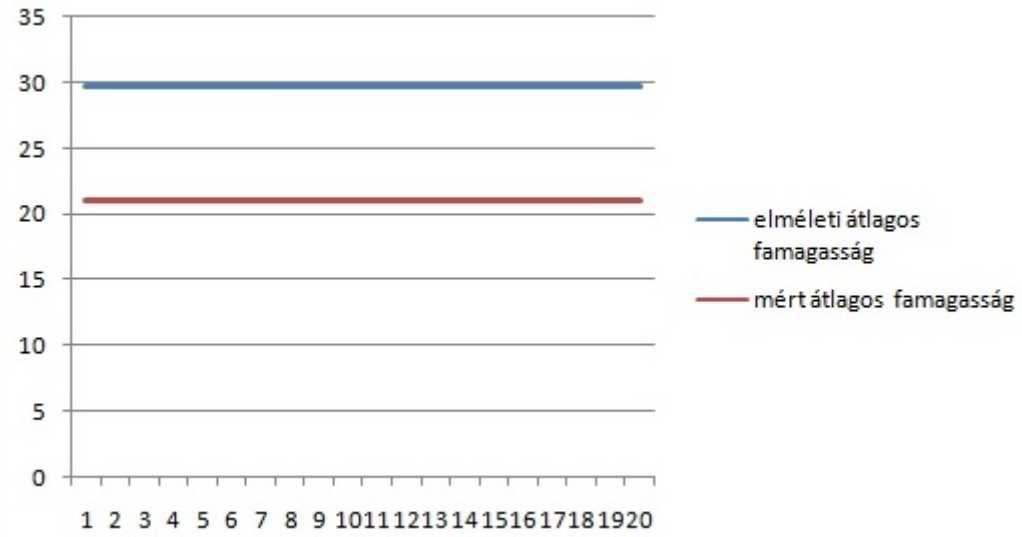
A program működése egyszerűen leírható, általában két inputot kell megadnunk, a kulcshalmaz méretét – azaz a csúcsok számát –, illetve a minta elemszámát. Például $n = 256$ esetén a kulcshalmazunk az $\{1, 2, \ldots, 256\}$ halmaz lesz és ezen halmaz elmeinek különböző permutációiból generálódik annyi keresőfa, amekkora értéket megadtunk. Ezen kívül az RBS-fák generálása során megadhatjuk a c konstans értékét, vagy egy számként, vagy pedig egy intervallum egymást követő elemeiként. Utóbbi esetben értelemszerűen annyi minta jön majd létre, ahány különböző értéket kap a c konstans.

A megfigyelések során a minta elemszámát mindvégig tízezernek választottuk,

mivel ez kellően nagynak mondható az eredmények elfogadásához, ugyanakkor elég alacsony ahhoz, hogy a program rövid idő alatt kivárható működést produkáljon.

Véletlen építésű bináris keresőfák átlagos magassága és csúcsmagassága

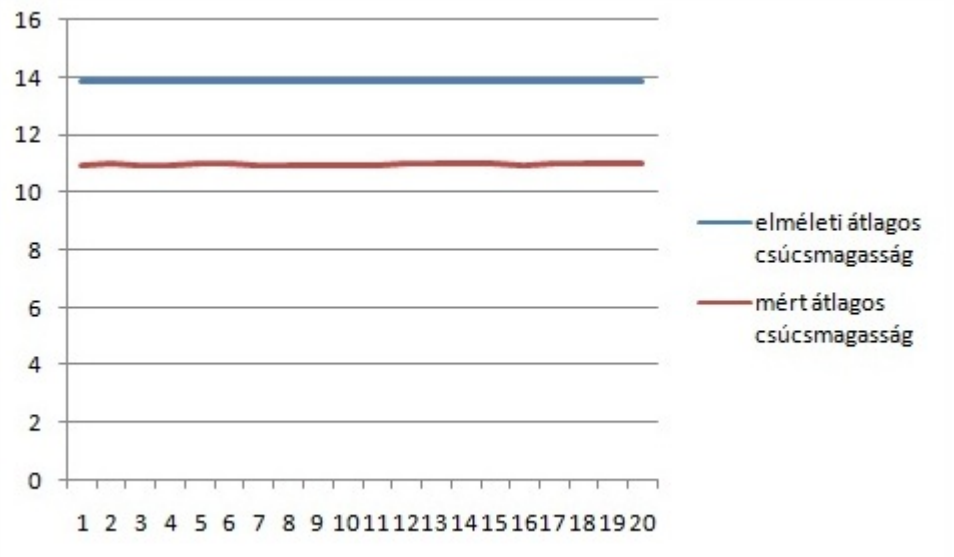
Jelölje az átlagos famagasságot n csúcsú keresőfa esetén $E(h(t_n))$, az átlagos csúcsmélységet pedig $E(d(t_n))$. A véletlen építésű bináris keresőfák átlagos magasságára vonatkozó első megfigyelésünkben az $\{1, 2, \ldots, 1000\}$ halmazt választottuk kulcshalmaznak, és húszszor legyártottunk egy tízezer méretű mintát, azaz megnéztük, egyazon kulcshalmazon végzett többszöri mintavétel esetén milyen eredmények realizálódnak. Az elméleti paraméterek jelen esetben $E(h(t_{1000})) = 29,70$ és $E(d(t_{1000})) = 13,85$. Elsőként összehasonlítottuk a mért és az elméleti átlagos famagasságot, ezt az alábbi ábárn szemléltetjük:



8.1. ábra.

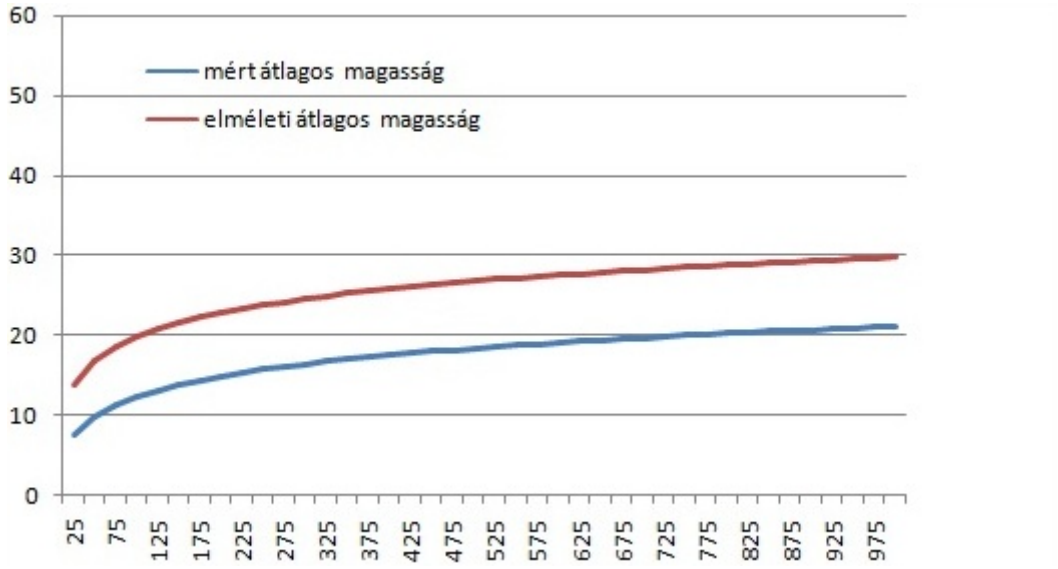
Az eredményt látva, kiszámoltuk a mért adatokra vonatkozó konstans szorzót és meglepő módon az elméletben felállított képletbeli $2,98$-es szorzó helyett csak $2,11$-at kaptunk, ami szignifikáns eltérésnek mondható. Hasonlóan jártunk el az átlagos csúcsmagasságot illetően, itt is kiszámoltuk a mért konstansok átlagát, és

azt tapasztaltuk, hogy $1,39$ helyett $1,10$ áll a formulában, ami nem annyira jelentős eltérés, mint az előbbi esetben, de így is figyelemreméltó. A mért és az elméleti csúcsmagasság összehasonlítása az alábbi ábrán látható:



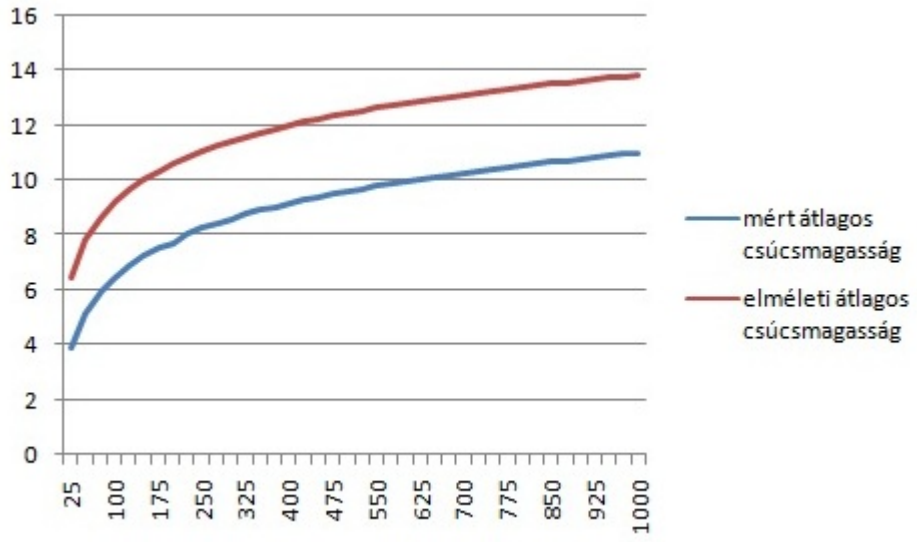
8.2. ábra.

Mivel azonban itt csak az $n = 1000$ esetet néztük, újabb kísérletet végeztünk, amely során $n = 25$-ről indulva n-et folyamatosan növeltük 25-tel, egészen 1000-ig.



8.3. ábra.

Ezzel tehát azt mértük, hogy a kulcshalmaz folyamatos növelésével, hogyan változnak a véletlen építésű bináris keresőfa paraméterei.



8.4. ábra.

Az ábrákon jól látható, hogy a gyakorlatban mindig kisebb értékeket kaptunk, mint az elméletben felállított paraméterek.

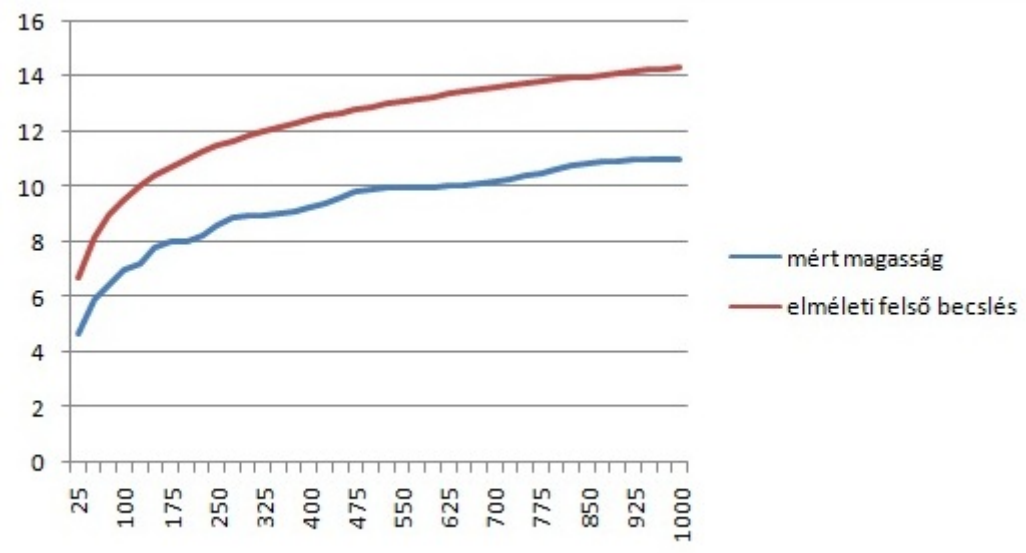
További megfigyelések azt mutatták, hogy az átlagos famagasság esetében a konstans szorzó általában 2 és 2, 5 közötti érték lesz, tehát a $2,98 \log_2 n$-es becslés negatív értelemben eltér a valós értékektől, azaz mindig rosszabb keresőfát jósol előre, mint amilyet a valóságban kapunk. Ugyanígy, az átlagos csúcsmagasságra vonatkozó konstans általában jóval alacsonyabban mozog, mint 1, 39.

AVL-fák magassága

Bebizonyítottuk, hogy n csúcsú AVL-fák esetén a keresőfa magassága felülről becsülhető $1,44 \log_2 n$-nel, ami már önmagában jó hatékonyságot jelent, de kíváncsiak voltunk, vajon a gyakorlatban tényleg ez a felső határ érvényesül leginkább? Megfigyelésünkben n-et ismét 25-től 1000-ig léptettük. A minták átlagos magasságát hasonlítottuk össze az elméleti felső becslésekkel, amik rendre számottevő eltérést mutattak. Az általunk mért konstans szorzók átlaga a kísérlet esetében 1, 08 lett,

ami azt jelenti, hogy az $1,44$-es felső korlát igen szigorú, az AVL-fák a gyakorlatban ennél sokkal "összenyomottabbnak", tehát használhatóbbnak bizonyultak.

A tapasztalatok azt mutatják, hogy az n csúcsú AVL-fák magassága általában $1,2 \log_2 n$ körül mozog, azaz a felső becslés valóban helytálló, ugyanakkor a magasság várható értéke soha nem éri el, vagy közelíti meg azt.



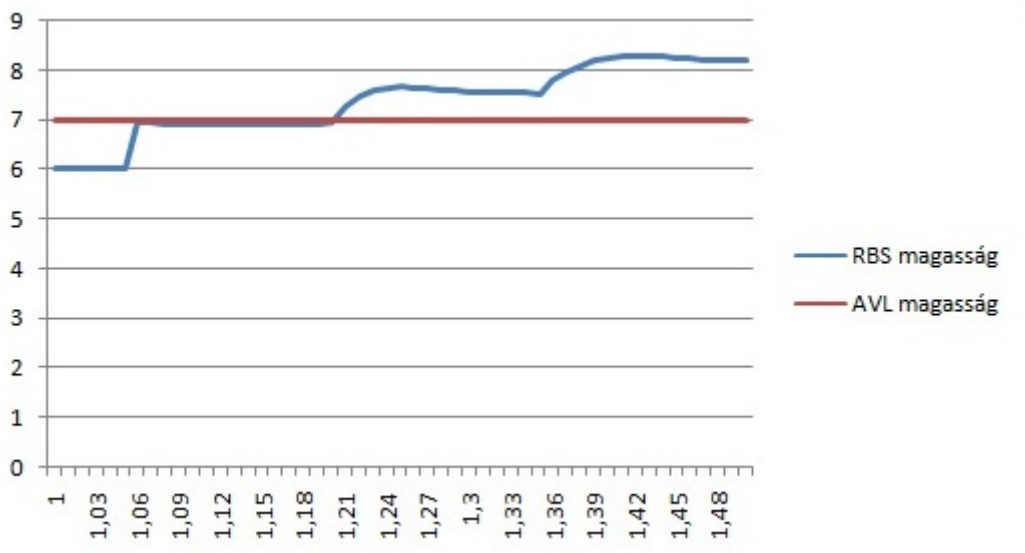
8.5. ábra.

RBS-fák hatékonysága

Az RBS-fák esetében az első vizsgálati szempont az RBS-fák és az AVL-fák hatékonyságának összehasonlítása volt, azaz megpróbáltuk megállapítani, hogy c konstans mely értékeire közelíti az RBS-fa az AVL-fa hatékonyságát. Ebben a kísérletben az $\{1, 2, \ldots, 100\}$ halmazt vettük kulcshalmaznak, c-t pedig előre rögzítettük olyan módon, hogy értéke 1-től $1,5$-ig fusson, $0,01$ lépésközzel. Az eredményt a 8.6. ábrán szemléltetjük, amiről könnyedén leolvasható, hogy ezen paraméterek esetén c konstans $1,06$ és $1,21$ közötti értékeire az RBS-fa ugyanazt a hatékonyságot produkálta, mint az AVL-fa.

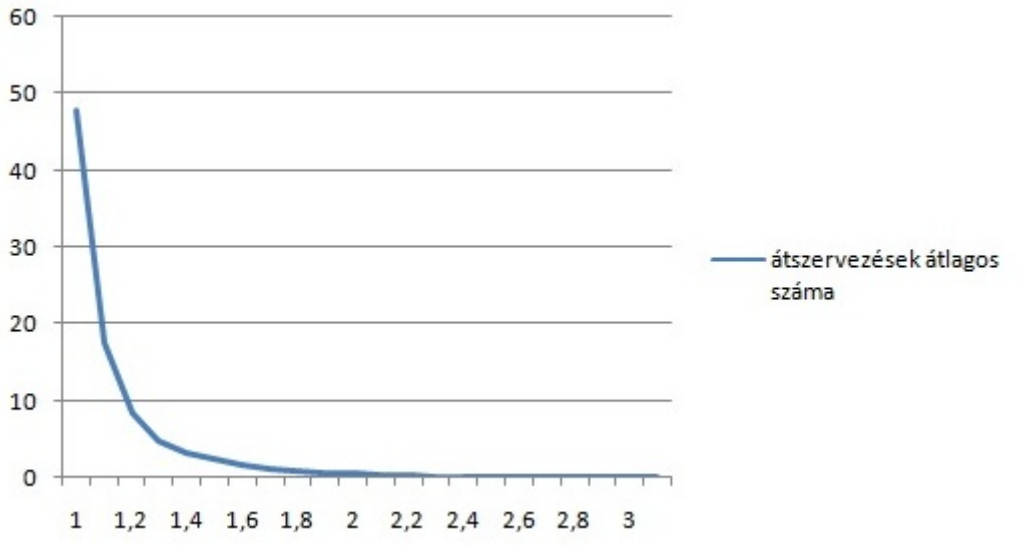
Ugyanakkor, ez viszonylag kis n esetén mondható, általánosságban, nagy n mellett, c konstans $1,3$ és $1,39$ közötti értéke esetén veszi fel a versenyt az RBS-fa az

AVL-fával.



8.6. ábra.

A második vizsgálati szempont c azon értékének keresése volt, amely mellett az RBS-fában egyáltalán nem történik átszervezés. Ismételten az $\{1, 2, \ldots, 100\}$ halmazt vettük kulcshalmaznak, c-t pedig 1-től 3-ig futtattuk, $0,1$ lépésközzel. Az eredmény magáért beszélt, az alábbi ábra szemlélteti:



8.7. ábra.

Azt mondhatjuk tehát, hogy $2,7-3$ -nál nagyobb értékek esetén várhatóan már nem történik átszervezés, azaz ekkor az RBS-fa hatékonysága csakis a felépítéséhez szükséges műveletektől függ.

A futási eredmények tehát rámutattak arra, hogy az elméletben felállított tézisek nem feltétlenül a legjobb becslést adják, sőt, olykor lényegesen eltérhetnek a valóságtól, ezzel biztonságos, de pesszimista korlátokat adva. Mindez arra enged következtetni, hogy bizonyos esetekben érdemes lehet inkább a gyakorlati tapasztalatokra hagyatkozni, mivel azok általában jobb eredményt mutatnak, ez pedig döntő fontosságú lehet egy feladat elvégzésében.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Fekete Istvánnak, aki számtalan teendője mellett mindig szakított rám időt, és tanácsaival, meglátásaival segített a dolgozatírás során. Köszönet illeti még Giachetta Robertót, akinek programját felhasználhattam munkámban, és ha egyéb problémám akadt, bátran felkereshettem őt kérdéseimmel. Továbbá köszönettel tartozom azon hallgatótársaimnak is, akik segítettek eligazodni a LaTeX használatában.

Irodalomjegyzék

[1] http://en.wikipedia.org/random binary tree, binary search tree.

[2] http://hu.wikipedia.org/avl-fa, piros-fekete fa.

[3] http://people.inf.elte.hu/fekete/specalg/vebk.pdf.

[4] I. Fekete, R. Giachetta, and P. Kovács. *To Balance or to Rebuild? - An Experimental Study of Randomly Built Binary Search Trees.* 8th International Conference on Applied Informatics, Eger, January 27-30, 2010.

[5] T. H. Cormen, C. E. Leiserson, R. L.Rivest, és C. Stein. *Új algoritmusok.* Scolar Kiadó, Budapest, 2003.

[6] T. H. Cormen, C. E. Leiserson, és R. L.Rivest. *Algoritmusok.* Műszaki Könyvkiadó, Budapest, 2001.

[7] Rónyai L., Ivanyos G., és Szabó R. *Algoritmusok.* Typotex Kiadó, Budapest, 1999.

[8] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial Applied Mathematics, Philadelphia, Pennsylvania, 1983.

[9] N. Wirth. *Algoritmusok + adatstruktúrák = programok.* Műszaki Könyvkiadó, Budapest, 1982.