3.

Keresőfák

Vagy találunk utat, vagy építünk egyet.
HANNIBÁL

Az egyik legalapvetőbb adatkezelési igény, hogy adatok véges halmazait *hatékonyan* tárolni tudjuk. Hatékonyságon elsősorban az értendő, hogy a *keresés*, *beillesztés*, *törlés* és *módosítás* műveletei gyorsak legyenek. Ebben a fejezetben a problémának azzal a fontos speciális esetével foglalkozunk, amikor a tárolt S halmaz elemei egy $(U, <)$ rendezett típusból valók. Ebben az esetben további természetes igények is felmerülnek: érdekelhet bennünket az S legkisebb vagy éppen legnagyobb eleme, esetleg az S-nek az $[a, b]$ intervallumba eső elemei $(a, b \in U)$.

Az így körvonalazott feladatcsokornak megfelelő adatszerkezet a *keresőfa*, amihez egyetlen $(U, <)$ rendezett típus tartozik. A szerkezet célja az U egy véges S részhalmazát tárolni úgy, hogy BESZÚR, TÖRÖL, KERES, MIN, MAX, TÓLIG hatékonyak legyenek.

Ebben a megközelítésben jelentős egyszerűsítés van. A legtöbb esetben a tárolni kívánt adataink összetett szerkezetűek, több különböző típusú információelemből állnak. Ezek közül általában egy vagy csak néhány adja a keresés/tárolás alapját jelentő kulcsot. A terítékre kerülő algoritmusok szempontjából azonban eléggé közömbös, hogy a kulcs mellett vannak-e még további adatmezők is. A legtöbb esetben mit sem vesztünk azzal, ha feltesszük, hogy csupán U bizonyos elemeit kell tárolnunk. Ezzel az egyszerűsítéssel lehetővé válik, hogy csak az algoritmikus szempontból lényeges tényezőkre figyeljünk.

A műveletek pontos értelmezésében – elsősorban a kivételek kezelésénél – többféle változattal találkozhatunk. Ezeket nem célunk itt áttekinteni; csak a műveletekhez kapcsolódó jellegzetes teendőkről ejtünk szót.

A BESZÚR(x, S) eljárás(hívás) beilleszti az $x \in U$ elemet a tárolt S halmazba. TÖRÖL(x, S) hatására x kikerül az S halmazból. KERES(x, S) megadja

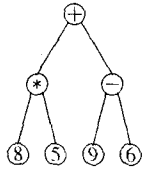
az x elem helyét az S-ben, feltéve, hogy $x \in S$. MIN(S) megadja az S-nek a $<$ rendezés szerinti legkisebb, MAX(S) pedig a legnagyobb elemét. Az intervallumkeresésnek az S halmaz mellett van még két paramétere. Ezek az U elemei; jelöljük őket a-val és b-vel. TÓLIG(a, b, S) eredménye azon $s \in S$ elemek összessége, melyekre teljesül, hogy $a \leq s \leq b$.

Az adatszerkezet nevében a *fa* szó arra hivatott utalni, hogy a legjobb ismert implementációk fákra épülnek, faszerű struktúrákat használnak. Mielőtt ezekből a megoldásokból ismertetnénk egy csokorra valót, egy kis kitérőt teszünk, rögzítve néhány bináris fákra vonatkozó tényt.

3.1. Bináris fák

Bináris fákkal már találkoztunk az előző fejezetben a kupac adatszerkezet kapcsán. Ott érintettük egy tárolási módjukat is, ami csak bizonyos speciális, terebélyes fákra működik. Itt egy általánosan használható megadási módot szeretnénk bemutatni. Ebben a fa egy csúcsa összetett szerkezetű; úgy gondolhatunk rá, mint egy rekord-típusú objektumra mondjuk a Pascal nyelvből. A x csúcs fontosabb összetevői, „mezői" a következők: $elem(x)$ az x csúcsban tárolt érdemi információ; ez a későbbiekben legtöbbször egy U-beli kulcs lesz. A $bal(x)$ mező egy mutató az x csúcs bal fiára, $jobb(x)$ pedig a jobboldali fiára. A módszerek tárgyalásakor ezeket a mutatókat olykor azonosítani fogjuk a mutatott csúccsal. Nagyjából ennyi tartozik a bináris fák *alapvető reprezentációjához*. Bizonyos esetekben további mezők is hasznosak lehetnek. Ilyen például az x csúcs apjára mutató $apa(x)$ mező vagy az x-ből és leszármazottaiból álló x-gyökerű részfa csúcsainak számát tartalmazó $s(x)$ mező.

Példa: A következő ábra egy bináris fát mutat; a tárolt elemeket - ezek számok és műveleti jelek - a csúcsokba írtuk. Jelölje x a fa gyökerét, y pedig a 9-et tartalmazó csúcsot. Érvényesek a következők: $bal(jobb(x)) = y$, $apa(apa(y)) = x$, $elem(bal(x)) = *$ és $s(x) = 7$.



A fejezetben fontos szerep jut majd a fák alakjának. Mint látni fogjuk, azok a fák lesznek hasznosak, amelyek eléggé telt alakúak. Ebből a szempontból azok

a fák ideálisak, melyeknek a szintjeik számához képest a lehető legtöbb csúcsuk van. Ha a szintek száma l, akkor ez $2^l - 1$ csúcsot jelent. Az ilyen fákat ebben a részben *teljes fáknak* fogjuk nevezni. Ez némiképp eltér a kupac adatszerkezetnél bevezetett használattól. A terminológiai kettősséget azért vállaljuk, mert eléggé általános a szakirodalomban.

A mostani meghatározás szerinti teljes fák úgy jellemezhetők, hogy minden levelük ugyanazon a szinten helyezkedik el, és minden belső csúcsuknak két fia van.

A bináris fák alkalmazásainál fontos szerepet játszanak az olyan módszerek – *bejárások* – amelyek valamilyen értelmes sorrendben végigmennek a fa csúcsain. Általában arról van szó, hogy amikor a bejárás során az y csúcs kerül sorra, akkor valami y-nal kapcsolatos teendőt végzünk el (pl. kinyomtatjuk a benne tárolt információt). A bejárásokat e teendőtől függetlenül érdemes tárgyalni, ezért egyszerűen csak azt mondjuk, hogy *meglátogatjuk* y-t. A legismertebb bejáró módszerek a *preorder*, *inorder* és *postorder* bejárások. Itt következik a rekurzív definíciójuk; x jelöli a fa gyökerét, az eljárások nevei pedig pre(), in() és post().

pre(x)	in(x)	post(x)
begin	**begin**	**begin**
látogat(x);	in($bal(x)$);	post($bal(x)$);
pre($bal(x)$);	*látogat*(x);	post($jobb(x)$);
pre($jobb(x)$)	in($jobb(x)$)	*látogat*(x)
end	**end**	**end**

A három eljárás csak a *látogat*(x); helyében különbözik egymástól. A preorder bejárás először meglátogatja a gyökeret, majd rekurzíve hívja önmagát először a bal, majd a jobb részfára. Az inorder bejárásnál a látogatás a két önhívás közé került, a postorder-módszernél pedig a hívások mögé.

Írjuk le az előző példa fájában levő elemeket abban a sorrendben, ahogy az eljárások meglátogatják a csúcsokat!

preorder sorrend: $+ * 85 - 96$;

inorder sorrend: $8 * 5 + 9 - 6$, megfelelően zárójelezve: $(8 * 5) + (9 - 6)$;

postorder sorrend: $85 * 96 - +$.

Az egyes sorrendek a fa által meghatározott aritmetikai kifejezés különböző írásmódjainak felelnek meg. A középső, az inorder sorrend adja a szokásos írásmódot.

Nézzük most a módszerek költségét. Elég a preorder bejárásra szorítkozni; a másik kettő ugyanúgy kezelhető. Tekintsük egységnyinek a pre(y)-hívás kezdésével és befejezésével járó költségeket; itt y a fa egy tetszőleges csúcsa. Ugyancsak legyen 1 a $bal(y)$ és $jobb(y)$ mutatók mentén való lépés, valamint *látogat*(y)

időigénye. Jelölje $T(n)$ a módszer maximális költségét legfeljebb n-pontú fákra. Feltehetjük, hogy $T(0) = 0$ és a rekurzió alapján

$$T(n) \leq c + \max_{0 \leq i \leq n-1} \{T(i) + T(n-1-i)\}.$$

A második tag a két rekurzív hívás költsége, az első pedig a hívásokon kívüli munka időigényét becslő n-től független c állandó. Egyszerű indukcióval adódik innen, hogy $T(n) \leq cn$. Ez ugyanis nyilván igaz, ha $n = 0$. Nagyobb n-ekre pedig az indukciós feltevés szerint

$$T(n) \leq c + \max_{0 \leq i \leq n-1} \{ci + c(n-1-i)\} = c + c(n-1) = cn.$$

A bejárások tehát $O(n)$, azaz lineáris időben megvalósíthatók.

Feladat: Tegyük fel, hogy adott az l szintből álló f bináris fa y csúcsa. Javasoljunk $O(l)$ költségű módszert a preorder (inorder, postorder) sorrendben az y után következő csúcs megtalálására.

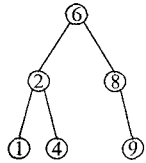
3.2. Bináris keresőfák, naiv algoritmusok

A keresőfa adatszerkezet megvalósításai közül első helyre kívánkoznak a *bináris keresőfák*. Ezek legfőbb közös vonása, hogy a tárolni kívánt $S \subseteq U$ halmaz elemei egy bináris fa csúcsaiban helyezkednek el, csúcsonként pontosan egy elem. Teljesül ezen felül a *keresőfa-tulajdonság*:

> *Tetszőleges x csúcsra és az x baloldali részfájában levő y csúcsra igaz, hogy $elem(y) \leq elem(x)$. Hasonlóan, ha z egy csúcs az x jobb részfájából, akkor $elem(x) \leq elem(z)$.*

Egy kényelmes megállapodás: a továbbiakban feltesszük, hogy nincsenek ismétlődő elemek a keresőfában. Ez a megállapodás jelentősen egyszerűsíti a módszerek leírását, a bennük levő gondolatok érzékeltetését. Az algoritmusok, amiket tárgyalni fogunk, kisebb-nagyobb módosításokkal működnek akkor is, ha megengedjük az ismétlődéseket. A hatékonyságuk viszont romlik, ha bizonyos elemekből túl sok példány van; gondoljunk arra, hogy milyen használhatatlan lenne a telefonkönyv, ha mindenkinek ugyanaz lenne a neve.

A megállapodást figyelembe véve a keresőfa-tulajdonság követelményeiben szigorú egyenlőtlenségeket érthetünk: az x bal részfájában levő elemek kisebbek x eleménél, a jobb részfában levők pedig nagyobbak annál. A következő példában az elemek természetes számok, a rendezés a szokásos. A fában tárolt S halmaz $\{1, 2, 4, 6, 8, 9\}$.



Feladat: Igazoljuk, hogy egy bináris keresőfa elemeit a fa inorder bejárása *növekvő sorrendben* látogatja meg.

Keresés bináris keresőfákban

Tegyük fel, hogy az S bináris keresőfának n csúcsa és l szintje van, és legyen $s \in U$. A KERES(s, S) első lépésében összehasonlítjuk S gyökerének s' elemét s-sel. Ha $s = s'$, akkor megtaláltuk a keresett csúcsot (elemet). Ha $s < s'$, akkor a keresőfa-tulajdonság miatt a bal, ha $s > s'$, akkor a jobb részfába megyünk tovább, ismételve a fenti lépést. Az előző példa fájánál a KERES$(4, S)$ a gyökérnél balfelé lép, hiszen $4 < 6$, utána jobbra, mert $4 > 2$; végül megtaláljuk a keresett elemet. Ugyanezt az utat járjuk be a KERES$(5, S)$ kapcsán. A négyest tartalmazó levélnél kiderül, hogy a kérdéses elem nincs a fában, hiszen eddig nem találtuk meg, és már nem tudunk továbblépni.

Egy összehasonlítás és egy mutató mentén megtett lépés árán az eredetinél egy szinttel alacsonyabb fában folytathatjuk a keresést. Ebből következik, hogy a módszer költsége arányos a szintek számával: $O(l)$.

A MIN és MAX műveletek is eléggé egyszerűek. Az előbbinél addig megyünk a gyökértől balra a fában, amíg lehetséges, az utóbbinál pedig a jobboldali irányt tüntetjük ki figyelmünkkel. E két eljárás költsége is $O(l)$.

A TÓLIG(a, b, S) hívás az S halmaz a és b közé eső kulcsait hivatott összegyűjteni. Evégből KERES(a, S) segítségével megkeressük a szóban forgó intervallum elejét, majd inorder sorrendben ellépdelünk az utolsó olyan kulcsig, ami még nem nagyobb, mint b. Az inorder lépkedéshez hasznos a csúcsokban egy mutató – úgynevezett inorder fonal – az inorder bejárás szerinti következő csúcsra. Ha vannak ilyen mutatóink, akkor TÓLIG(a, b, S) költsége $O(l + k)$, ahol k az S a és b közé eső elemeinek a száma. Ha nincsenek inorder fonalaink, akkor inorder bejárással $O(n)$ időben kaphatjuk meg a kívánt eredményt.

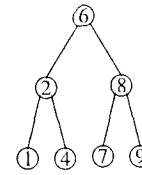
Naiv módszerek beszúrásra és törlésre

A különféle bináris keresőfa-típusok között a döntő különbség a beszúrás és a törlés algoritmusaiban van. Itt a legegyszerűbb, az úgynevezett naiv módszereket mutatjuk be. A BESZÚR(s, S) és TÖRÖL(s, S) végrehajtása is egy KERES(s, S)

hívással kezdődik. Beszúráskor a beillesztendő s elem helyét kell megkeresni S-ben, a TÖRÖL(s, S) esetén pedig a törlendő s elemet.

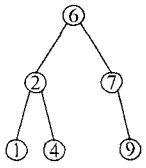
BESZÚR(s, S) végrehajtásakor a keresés után nincs teendőnk, ha már van olyan x csúcsa a fának, melyre $elem(x) = s$. Ellenkező esetben a keresés úgy ér véget, hogy nem tudunk továbblépni a csúcsból, ahol vagyunk. A megfelelő (jobb, vagy bal) fiú nincs a fában. A beszúrást úgy végezzük el ezután, hogy ezt a hiányzó fiút mint új levelet a fához adjuk. Az új csúcs eleme s lesz. Könnyű meggondolni, hogy ezután is érvényben marad a keresőfa-tulajdonság. Az eljárás költsége $O(l)$.

Példa: Az előző S fába illesszük be új elemként a hetest, vagyis hajtsuk végre a BESZÚR$(7, S)$ utasítást. A kereső fázisban arra jutunk, hogy a hetesnek a nyolcast tartalmazó csúcs bal fiában volna a helye. Ilyen fiú nincs a fában, ezzel bővíteni kell a szerkezetet. Az eredmény:



Ami a törlést illeti, nincs érdemi teendőnk, ha a szóban forgó s elem nincs a fában. Ellenkező esetben jelölje x az S-nek azt a csúcsát, amiben s ül. Ezek után TÖRÖL(s, S) könnyű, ha a szóban forgó x csúcsnak legfeljebb egy fia van. Ekkor elegendő, hogy x-et a fiával helyettesítjük: $x \leftarrow$ fiú(x). Ha viszont x-nek két fia van, akkor ez az út nem járható; csak egyik fiút tehetnénk x helyére, a másik árván maradna. A megoldás az lesz, hogy a feladatot visszavezetjük egy könnyű törlésre. Jelölje y azt a csúcsot, amiben az x bal részfájának a maximális eleme van. Az x ismeretében y-t MAX$(bal(x))$ $O(l)$ költséggel megadja. A recept ezután egyszerű. Az y-ban levő s' elemet tesszük s helyére x-be, majd töröljük az y csúcsot. Utóbbi egy könnyű törlést jelent, mert y-nak nem lehet jobboldali fia. Azt is könnyű meggondolni, hogy az így módosult fára érvényben marad a keresőfa-tulajdonság. Az s' definíciója miatt kisebb az x jobb részfájának minden eleménél, és nagyobb a bal részfa megmaradó elemeinél. Az összköltség ekkor is $O(l)$.

Példa: Az előző rajzon látható S fából töröljük a nyolcast. Ez nehéz törlésnek minősül, hiszen a szóban forgó x csúcsnak két fia van. Az x bal részfájának maximális (és egyetlen) eleme 7. Ez kerül x-be; a hetest tartalmazó csúcsot pedig kitöröljük:



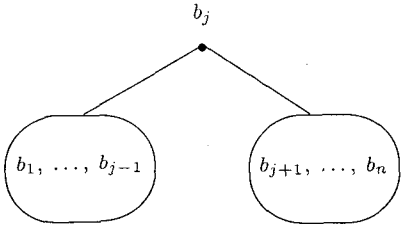
Naiv beszúrásokkal épített bináris fák átlagos költsége

Ha egy bináris fának l szintje van, akkor a csúcsok számára $n \leq 1+2+2^2+\cdots+2^{l-1} = 2^l - 1$ adódik, amiből $l \geq \log_2(n+1)$. A keresés szempontjából tehát a legjobb – azaz legkisebb – szintszám, amit n-pontú fánál elérhetünk, körülbelül $\log_2 n$.

A következőkben érvet mutatunk amellett, hogy a naiv beszúrásokkal épített fák átlagos értelemben nem rosszak; egy beillesztés átlagosan $O(\log_2 n)$ összehasonlításba kerül. A pontos modell a következő: tegyük fel, hogy az U univerzum $b_1, < b_2 < \ldots < b_n$ elemeiből bináris keresőfát építünk. A b_i elemeket egy véletlen $a_1, \ldots, a_n$ sorrendben szúrjuk be a naiv módszerrel a kezdetben üres fába. Legyen $T(n)$ a beszúrások során fellépő összehasonlítások átlagos száma. A $T(n)$ mennyiséggel szeretnénk mérni a fa építésének átlagos költségét. Az átlagot az elemek $n!$ lehetséges érkezési sorrendjére vesszük. Valószínűségi terminológiát használva úgy is fogalmazhatunk, hogy a $b_1, b_2, \ldots, b_n$ elemek minden érkezési sorrendje egyenlően valószínű. Ekkor tetszőleges j-re $\frac{1}{n}$ az esélye, hogy $a_1 = b_j$, hiszen mindegyik elem éppen ugyanannyi (nevezetesen $(n-1)!$) érkezési sorrendnél lesz első. Mindez azt jelenti, hogy

$$(*) \qquad T(n) = \frac{1}{n}\sum_{j=1}^{n} (\text{az olyan fa átlagos költsége, melyre } a_1 = b_j).$$

Egy fa költségén a felépítéséhez használt összehasonlítások számát értjük. Jelölje $F(j)$ a zárójelben levő tagot. Ha $a_1 = b_j$, akkor a keresőfa-tulajdonság miatt végül az a_1 bal részfájában $j-1$, a jobb részfában pedig $n-j$ elem lesz.



Itt a baloldali részfán belüli összehasonlítások száma $T(j-1)$ lesz, amihez hozzá kell adnunk a b_j gyökérrel való összehasonlításokat. A bal részfa átlagos költsége tehát $j-1+T(j-1)$. Hasonló meggondolással a jobb részfa költsége $n-j+T(n-j)$. A kettőt összegezve kapjuk, hogy $F(j)=j-1+T(j-1)+n-j+T(n-j)=n-1+T(j-1)+T(n-j)$. Mindezeket visszaírhatjuk a $(*)$ formulába. Használva azt is, hogy az üres fa ingyen van, azaz $T(0)=0$, $n>0$-ra a következő rekurziót nyerjük:

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

Vegyük észre, hogy ez megegyezik a gyorsrendezés elemzésekor kapott $(+)$ rekurzióval. Az ott taglalt módon adódik tehát, hogy

$$T(n)<2n\ln n+O(n)\approx 1,39n\log_2 n+O(n).$$

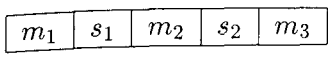
Egy elem beszúrásához átlagosan tehát legfeljebb $1,39\log_2 n$ összehasonlítás elegendő. A becslés érvényes a keresés átlagos költségére is. Hasonló módszerekkel megmutatható, hogy az átlagos szintszám (azaz l várható értéke) is $O(\log n)$. Ez nem rossz, ha figyelembe vesszük, hogy adott n mellett a legjobb, amit elérhetünk $l=\log_2(n+1)$. Ez akkor teljesül, ha a fa teljes (l szintje és összesen 2^l-1 csúcsa van).

3.3. 2-3-fák

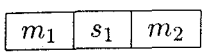
Ebben a szakaszban egy igen hatékony keresőfa-konstrukciót mutatunk be. Ennek is fa a tárolási szerkezete, de a binárisnál valamivel bonyolultabb; egy nem-levél csúcsnak 2 vagy 3 fia lehet. Innen származik a szerkezet elnevezése. Egy kulcs (U eleme) a fa több csúcsában is előfordulhat, és egy csúcsban egy vagy két kulcs is lehet. Úgy érdemes elképzelni a helyzetet, hogy U elemei egy rekordtípus kulcsai (pl. személyi szám egy személyi adatokat tartalmazó rekordban). A cél a rekordok hatékony elérése. Maguk a rekordok a fa legalsó szintjén helyezkednek el, a fa belső csúcsai csak kulcsokat és mutatókat tartalmaznak (kivéve, ha csak egy rekordunk van). A belső csúcsokban levő kulcsok a levelekben tárolt rekordok kulcsai közül kerülnek ki. Szerepük az, hogy jelzőtáblák módjára segítsék a fában való közlekedést.

A *2-3-fa* egy (lefelé) irányított gyökeres fa, melyre igazak az alábbiak:
1. A tárolni kívánt rekordok a fa leveleiben helyezkednek el, a kulcs értéke szerint balról jobbra növekvő sorrendben. Egy levél egy rekordot tartalmaz.

2. Minden belső (azaz nem levél) csúcsból 2 vagy 3 él megy lefelé; ennek megfelelően a belső csúcsok egy illetve két $s \in U$ kulcsot tartalmaznak. A belső csúcsok szerkezete tehát kétféle lehet. Az egyik típus így ábrázolható:



Itt m_1, m_2, m_3 mutatók a csúcs részfáira, s_1, s_2 pedig U-beli kulcsok, melyekre teljesül, hogy $s_1 < s_2$. Az m_1 által mutatott részfa minden kulcsa kisebb, mint s_1, az m_2 részfájában s_1 a legkisebb kulcs, és minden kulcs kisebb, mint s_2. Végül m_3 részfájában s_2 a legkisebb kulcs. Előfordulhat, hogy egy csúcsból az utolsó két mező hiányzik. A belső csúcsoknak ez a típusa valamivel egyszerűbb alakú:



Ez esetben a csúcsnak csak két részfája van; a baloldaliban vannak az s_1-nél kisebb kulcsú rekordok, a jobboldaliban pedig s_1 a legkisebb kulcs.
3. A fa levelei a gyökértől egyforma távolságra vannak.

A következő állítás a műveletek költségének becslésekor lesz hasznos. Lényegében azt mondja, hogy a fa szintjeinek száma barátságosan alacsony a fában levő rekordok számához mérten.

Állítás: *Ha a fának m szintje van, akkor a levelek száma legalább 2^{m-1}. Megfordítva, ha $|S| = n$ (itt $S \subseteq U$ a fában tárolt kulcsok halmaza; $|S|$ megegyezik a tárolt rekordok számával), akkor $m \leq \log_2 n + 1$.*

Bizonyítás: A 3. tulajdonság szerint a fa i-edik szintjén levő csúcsok mind belső csúcsok, ha $1 \leq i \leq m-1$. A 2. tulajdonság szerint minden ilyen csúcsnak legalább két fia van. Ezekből könnyű indukcióval adódik, hogy az i-edik szinten legalább 2^{i-1} csúcs van ($1 \leq i \leq m$). Ezt $i = m$-re alkalmazva $2^{m-1} \leq n$, amiből logaritmusokat véve $m - 1 \leq \log_2 n$. □

Keresés 2-3-fákban

A kulcsoknak a részfákban való ilyen elosztása lehetővé teszi, hogy a bináris fáknál megismert módszerhez hasonlóan végezzük a keresést. Legyen S egy 2-3-fa, és $s \in U$. A KERES(s, S) első lépéseként az s kulcsot összehasonlítjuk az S gyökerében levő s_1 és esetleg s_2 kulcsokkal. Ennek eredményeképpen megtudjuk, hogy a három (vagy két) részfa közül melyikben kell folytatni a keresést. Ha $s < s_1$, akkor az m_1 mutató mentén találjuk meg az érdekes részfa gyökerét. Ha $s_1 \leq s < s_2$, akkor a második mutatót követjük. Ha pedig $s_2 \leq s$, akkor m_3

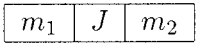
mutatja a helyes irányt. Természetesen, ha s_2 és m_3 hiányzik, akkor csak két eset van: $s < s_1$, illetve $s_1 \leq s$.

A keresést aztán ugyanígy folytatjuk egy szinttel lejjebb. Legyen n a fában tárolt rekordok száma, m pedig a fa szintjeinek a száma. Az elmondottak szerint szintenként nem több, mint két kulcs-összehasonlítással, összesen tehát legfeljebb $2m$-mel kideríthetjük, hogy a keresett rekord benne van-e a fában. Igenlő válasz esetén meg is találjuk az s kulcsú rekordot. Az állítást figyelembe véve látjuk, hogy az összehasonlítások száma legfeljebb $2(\log_2 n + 1)$; a többi költség arányos ezzel. Így a keresés összköltségére igen kedvező korlátot kapunk: $O(\log n)$. A korlát *minden esetben* (tehát nem csak átlagos értelemben) érvényes.

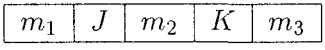
Beszúrás és törlés

A 2. és 3. követelmények igen szigorúan szabályozzák a 2-3-fák alakját. Ennek a két tulajdonságnak köszönhető a versenyképes keresési idő. A beszúrás és törlés kapcsán a fő gondot az jelenti, hogy megőrizzük érvényességüket. A módszerek lényeges ötleteit példákon keresztül mutatjuk be. Ennek során feltesszük, hogy U elemei nagybetűk: $A, B, C \ldots$, a szokásos rendezéssel.

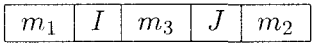
A BESZÚR(s, S) végrehajtása kereséssel kezdődik, és csak akkor van érdemi munka, ha nincs a fában s kulcsú levél. Tegyük fel, hogy ez a helyzet, és jelölje x a legalsó belső csúcsot a kereső út mentén. Tegyük fel, hogy x így néz ki:



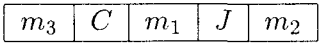
Az m_1 által meghatározott levél kulcsa legyen C (a másik levélé szükségképpen J). Ha például $s = K$, akkor x így módosul:



Az m_3 mutató fog az új levélre mutatni. Ha viszont $s = I$, akkor x tartalma a következő lesz:



Ha s a legkisebb kulcsot is megelőzi, mondjuk $s = A$, akkor x így alakul:



Az új rekord beillesztése tehát elég egyszerű, ha x-nek csak két fia van.

Igazán érdekessé akkor válik a helyzet, ha a beszúrás előtt x már két kulcsot tartalmaz. Tegyük fel, hogy x a legutóbbi ábra szerinti állapotban van, és nézzük,

mi a teendő, ha most az $s = H$ kulcsú rekordot kell beilleszteni. Ekkor alkalmazzuk a *csúcsvágás* elnevezésű elegáns ötletet: az x mellé egy további y csúcsot veszünk. A két csúcs tartalma

m_3	C	m_1

illetve

m_4	J	m_2

lesz. Az m_4 fog a H kulcsú rekordra mutatni. Ezután az új y csúcsot is a fába kell illeszteni. Ez azt jelenti, hogy az x apjába kell egy új kulcs–mutató párt tennünk. A beillesztendő kulcs a három érintett (a két régi és az új) közül mindig a nagyság szerint középső. Ez a kulcs a jelen esetben H. Az egy szinttel feljebb történő beszúrást *ugyanazzal* a módszerrel végezzük, ahogy x-nél eljártunk. Tehát ha x apjának csak két gyermeke volt, akkor az egyszerű ágon fejezzük be a munkát. Ha nem, akkor itt is csúcsvágást alkalmazunk. A csúcsvágások sora elgyűrűzhet esetleg a fa gyökeréig. Mi ilyenkor a teendő? Tegyük fel, hogy az utolsó példában x a fa gyökere volt. (Ekkor az m_i mutatók esetleg nagyobb részfákra mutatnak.) A csúcsvágás után keletkező x és y rekordok fölé egy új gyökeret teszünk:

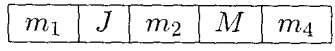
m	H	m'

Itt m az x, m' pedig az y csúcsra mutat. A fa szintjeinek száma eggyel nőtt. A lényeges mozzanat ebben, hogy a növekedés a fa tetején történt; ezáltal megtartottuk a 2-3-fákkal kapcsolatos 3. követelményt. A gyökértől levélig menő utak mindegyike hosszabb lett. A második követelményről is gondoskodtunk: az eseteket végignézve az olvasó meggyőződhet arról, hogy sehol sem hagytunk belső csúcsot kettőnél kevesebb gyermekkel.

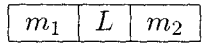
Nézzük mármost a TÖRÖL(s, S) végrehajtását. Legyen ismét x a legalsó belső csúcs a kereső út mentén. Ha x-nek három fia van, akkor az s kulcsú levél törlése után x-ben az értelemszerű változtatásokat elvégezve készen vagyunk. Gondot jelent viszont, ha x-nek csak két fia van. Ekkor a törlés után megsérülne a 2-3-fákkal szembeni 2. követelmény, hiszen x-nek csak egy fia maradna. Még mindig egyszerű a dolgunk, ha x (valamelyik) szomszédos testvérének 3 fia van; ekkor ugyanis ezek közül egyet áttehetünk x-be. Csak x-et, adakozó kedvű testvérét és az apjukat kell módosítani. Ezeket a módosításokat elvégezve ismét készen vagyunk.

Mindezek nem lehetségesek, ha x egyik szomszédos testvérének sincs három részfája. Ekkor használható a csúcsvágás fordítottját jelentő gondolat, a *csúcsösszevonás*. A csúcsösszevonás lényege, hogy x-et és az egyik testvérét egyetlen csúccsal helyettesítjük. Ennek a csúcsnak három fia lesz. A csúcsösszevonás kivitelezését is egy példán keresztül mutatjuk be. Legyenek x és a fában szomszédos testvére, y az alábbiak:

Legyenek az m_1, illetve m_3 által mutatott levelek kulcsai A és K. (Az m_2-höz és m_4-hez tartozó kulcsok szükségképpen J, illetve M.) Ha $s = K$ a törlendő kulcs, akkor az összevonás után x így néz ki.



Az y csúcsot töröljük. A K kulcsértéket és a hozzá tartozó mutatót törölni kell x apjából (törlés egy szinttel feljebb). A többi három eset (azaz A vagy J vagy M törlése) hasonlóan végezhető. Csúcsösszevonáshoz vezető törlés után egy szinttel magasabban is felmerül egy törlési igény. Ezt az itt ismertetett módon kezeljük. Ennek során is szükség lehet csúcsösszevonásra, ami törlést igényel a nagyapák szintjén; és így tovább. A beszúrásnál tapasztalt jelenséghez hasonlóan a törlések is elgyűrűzhetnek egészen a gyökérig. Külön figyelemre akkor van szükség, ha a törlés után a gyökérnek csak egyetlen fia maradna. Például tegyük fel, hogy a gyökér



alakú, és törölnünk kell az L kulcsot és a hozzá tartozó m_2 mutatót. Ekkor az m_1 által mutatott részfa gyökere lesz az új gyökér. A fa szintjeinek száma eggyel kevesebb lesz. A változás ismét a fa tetején történt; ebből könnyen következik, hogy a törlés algoritmusa is megtartja a 2-3-fákkal kapcsolatos követelményeket.

Mind a BESZÚR, mind a TÖRÖL költsége $O(\log n)$, hiszen munkát csak a kereső út csúcsain, illetve azok közvetlen szomszédain kell végezni; továbbá egy csúcsra konstans mennyiségű összehasonlítás és mezőmódosítás jut.

TÓLIG(a, b, S) megvalósítása hasonló a bináris fáknál tárgyalt megoldáshoz. Először megkeressük S-ben a keresési intervallumba eső legkisebb elemet; ennek költsége az előbbiek szerint $O(\log n)$, majd végiglépkedünk a felső korlátig. Ez utóbbi fázis a fa leveleinek növő sorrendű láncolásával segíthető. Az összköltség $O(\log n + d)$, ahol d az eredményül kapott rekordok száma.

Összegezésül elmondhatjuk, hogy a 2-3-fák a keresőfa adatszerkezet igen jó megvalósítását adják. A három fő művelet – a keresés, a beszúrás és a törlés – elvégezhető $O(\log n)$ költséggel, ahol n a tárolt rekordok száma. A 2-3-fák algoritmusai viszonylag egyszerűen programozhatók, és a gyakorlatban is kedvező viselkedést mutatnak.

3.4. B-fák

A B-fák és különböző változataik (B^*-fák, stb.) adják a keresőfa adatszerkezet ma ismert legjobb külső táras megvalósítását. A módszerek olyannyira fontosak és elfogadottak, hogy szabványok részeivé váltak. Ilyen megoldást ír elő egyebek között az IBM VSAM szabványa indexelt szekvenciális állományok létrehozására, kezelésére. A B-fákat és algoritmusaikat a feladat jelentőségéhez és a megoldások egyszerűségéhez képest meglehetősen későn fedezték fel (R. Bayer, E. McCreight, 1972).

A cél tehát egy rekordokból álló állomány tárolása külső tárban úgy, hogy az elérés alapjául egy $(U, <)$ rendezett halmazból való kulcsok szolgálnak. A kitüntetett műveletek itt is a következők: BESZÚR, TÖRÖL, KERES, MIN, MAX, TÓLIG. Ezeket szeretnénk hatékonyan megvalósítani.

A külső tárak alapvető sajátosságairól már szóltunk a *rendezés* kapcsán. A tárat lapokba szervezettnek fogjuk elképzelni. Ennek folyománya például, hogy a fák csúcsai is lapok lesznek. A költségtényező pedig, amivel a hatékonyságot mérjük, a lapelérések száma.

A B-fák a 2-3-fák természetes általánosításai. A lényegi különbség abban van, hogy a B-fák belső csúcsainak háromnál több fia is lehet. Kevésbé fontos különbség, hogy a fa egy levelében (amit egy lapnak képzelhetünk el) egynél több rekord is helyet kaphat. Legyen $m \geq 3$ egy egész.

Egy *m-edrendű B-fa*, röviden B_m-fa egy gyökeres, (lefelé) irányított fa, melyre érvényesek az alábbiaknak:

(a) A gyökér foka legalább 2, kivéve esetleg, ha a fa legfeljebb kétszintes.
(b) Minden más belső csúcsnak legalább $\lceil \frac{m}{2} \rceil$ fia van.
(c) A levelek a gyökértől egyforma messze vannak.
(d) Egy csúcsnak legfeljebb m fia lehet.

A 2-3-fák esetéhez hasonlóan a tárolni kívánt rekordok itt is a fa leveleiben helyezkednek el; egy levélben a lapmérettől és a rekordhossztól függően több rekord is lehet. A leveleket jelentő lapok balról jobbra, kulcsérték szerint növekvő sorrendben láncolva helyezkednek el.

A B-fák belső csúcsai is hasonlítanak a 2-3-fák belső csúcsaira. Az eltérés annyi, hogy itt több bejegyzés lehetséges. Egy belső csúcs így néz ki:

m_0	s_1	m_1	s_2	m_2	...	s_i	m_i

ahol a (b) és (d) követelményeknek megfelelően $\lceil \frac{m}{2} \rceil - 1 \leq i \leq m - 1$. Korábbi jelöléseinkkel összhangban az $s_i \in U$ egy kulcs, m_j pedig mutató egy részfa gyökerére. Az m_j mutató által meghatározott részfa minden s kulcsára igaz, hogy

$s_j \leq s$ és $s < s_{j+1}$. Az m_0, illetve m_i részfák kulcsaira csak egy feltétel van: az elsőben a kulcsoknak kisebbeknek kell lenniük, mint s_1, az utolsóban pedig legalább akkoráknak, mint s_i. Előírható még az is (bár az algoritmusok működéséhez nem feltétlenül szükséges), hogy $j > 0$ esetén m_j részfájában a minimális kulcs s_j legyen. A fának azon szintjeit, ahol a belső csúcsok helyezkednek el, *index-szinteknek* szokás nevezni.

A meghatározó tulajdonságokból közvetlenül látszik, hogy a B_3-fák lényegében a 2-3 fa. A B-fák a 2-3-fák természetes általánosításainak tekinthetők. Ahogy már utaltunk rá, a B-fákat elsősorban külső táras adatszerkezetként alkalmazzák, és ebből adódóan egy csúcsnak általában egy lapot foglalnak le. Így m aktuális értéke a lapméret, a kulcshossz és a mutatók hosszának függvénye. Ugyanakkor az is gyakran előfordul, hogy egy levélben több, mint egy rekord helyezkedik el.

Tegyük fel, hogy egy B-fának n levele és k szintje van, és keressünk összefüggést e két paraméter között. Az érdektelen kicsi fáktól eltekintve a gyökérnek legalább két fia van, a többi belső csúcsnak pedig legalább $\lceil \frac{m}{2} \rceil$. A 2-3-fákra vonatkozó állításhoz hasonlóan kapjuk ezekből, hogy $n \geq 2\lceil \frac{m}{2} \rceil^{k-2}$, amiből $\log_{\lceil \frac{m}{2} \rceil} \frac{n}{2} + 2 \geq k$. Innen kettes alapú logaritmusra térve

$$k \leq \frac{\log_2 n - 1}{\log_2 \lceil \frac{m}{2} \rceil} + 2.$$

Keresés során itt is a gyökértől haladunk a levelek felé; az aktuális csúcsban levő kulcsokkal való összehasonlítások mondják meg, hogy melyik részfába kell továbblépni. Vegyük észre, hogy k éppen a kereséshez szükséges lapelérések száma, feltéve, hogy minden lap külső tárban van. (Nem szokatlan gyorsítási ötlet a fa felső néhány szintjének a belső memóriában való tárolása.) A kapott korlát mutatja, hogy nagy m érték (nagy elágazási tényező) az előnyös.

Például, ha $m = 32$, $n = 2^{20}$ (itt n az alsó szint *lapjainak* száma), akkor $k \leq \frac{19}{4} + 2 < 7$. Egy rekord keresése tehát legfeljebb 6 lap elérését igényli. Megjegyezzük még itt, hogy a becslésben úgy vettük, hogy a lapok éppen a (b) feltételbeli minimális mértékben vannak kitöltve. A gyakorlatban ez a szélsőséges helyzet meglehetősen ritkán fordul elő; a tényleges keresési idő valamivel kedvezőbb a becsültnél.

A további keresőfa-műveletek (MIN, MAX, TÓLIG, BESZÚR , TÖRÖL) is a 2-3-fák műveleteinek kézenfekvő általánosításai. Így például a beszúrásnál csúcsvágást, a törlésnél csúcsösszevonást alkalmaznak. Ezeket a technikákat ebben az összefüggésben *lapvágásnak*, illetve *lapösszevonásnak* nevezzük. Az algoritmusok részleteit mellőzzük; csak annyit jegyzünk itt meg, hogy a költségük – a TÓLIG kivételével – ez esetben is arányos a szintek számával.

Példa: A B_{11}-fák esetén keresztül szeretnénk érzékeltetni, hogy a (b) és (d) feltételek tényleg megtarthatók lapvágás/összevonás során. Ekkor ugyanis $m = 11$ és $\lceil \frac{m}{2} \rceil = 6$. Lapvágásra akkor kerül sor, ha egy olyan csúcsba kell új gyereket illeszteni, amelyben már 11 mutató van. A vágás után az összesen 12 gyerek 2 hatgyermekes csúcsot fog alkotni, tehát (b) éppen teljesül.

Lapösszevonásra akkor kényszerülünk, ha egy olyan hatgyermekes csúcsból kell törölni, amelynek a szomszédos testvéreiben is csak hat mutató van. A törlés és összevonás után két csúcs helyett egyet kapunk; ebben 11 mutató lesz.

3.5. *AVL*-fák

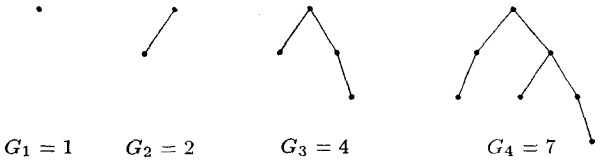
Egy időre visszatérünk a bináris keresőfákhoz. Korábban láttuk, hogy a keresés és a többi művelet sebességének szemszögéből a lényeges jellemző a fa *szintjeinek száma*, más szóval *magassága*. Minél kisebb a magasság, annál jobb becslés adható a keresés idejére. Mi ezt a legkedvezőtlenebb esetekre állapítottuk meg, de hasonló a helyzet az átlagos viselkedéssel is. Egy fa magassága akkor mondható jónak, ha legfeljebb $c \log_2 n$, ahol n a csúcsok száma és c egy kis pozitív állandó. A legterebélyesebb fáknál ez a c érték 1 körül van: gondoljunk a teljes fákra, amelyek magassága $\log_2(n+1)$. Az olyan fa-konstrukciókat, amelyeknél c 1-nél nem sokkal nagyobb, *kiegyensúlyozott* fáknak nevezzük. A c értéket illetően két ellentétes irányú tényezőt kell figyelembe vennünk. A keresés szempontjából a kisebb c a jobb; ugyanakkor nagyobb c-vel a feltétel könnyebben teljesíthető, algoritmikusan kényelmesebben elérhető.

Az első kiegyensúlyozott keresőfa-konstrukciót, amit szemügyre veszünk, *AVL*-fának nevezik. Az elnevezés a szerkezet szerzőinek névbetűiből alkotott mozaikszó (G. M. Adelszon-Velszkij, E. M. Landisz, 1962). Hasznos lesz a következő: jelölje $m(f)$ az f bináris fa magasságát (szintjeinek számát). Legyen x az f fa egy csúcsa; ekkor $m(x)$ jelöli az x-gyökerű részfa magasságát.

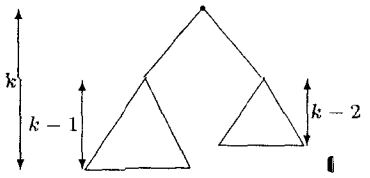
Definíció (*AVL*-tulajdonság): *Egy bináris keresőfa AVL-fa, ha minden x csúcsára teljesül, hogy*

$$|m(bal(x)) - m(jobb(x))| \leq 1.$$

A feltétel szerint egy csúcs jobb és bal részfájának a magassága közel egyenlő. Most megmutatjuk, hogy az előírás tényleg egy kiegyensúlyozottsági feltétel. Látszólag messziről kezdjük: legyen G_k a k magasságú (szintszámú) *AVL*-fák minimális csúcsszáma. Próbáljuk meghatározni G_k nagyságrendjét! Az első néhány k-ra könnyen kapjuk a pontos értékeket: $G_1 = 1$, $G_2 = 2$, $G_3 = 4$ és $G_4 = 7$.



A rajzon látható szabályszerűség általában is érvényes. A k szintszámú minimális csúcsszámú *AVL*-fa gyökerének egyik részfája $k-1$, a másik $k-2$ szintű; az eredeti fa minimalitása miatt pedig mindkét részfa minimális csúcsszámú.



Innen $k \geq 3$ esetén az alábbi rekurziót nyerjük:

$$G_k = 1 + G_{k-1} + G_{k-2}.$$

Emlékeztetjük az olvasót, hogy F_i jelöli az i-edik Fibonacci-számot. Érvényes a következő:

Tétel: $G_k = F_{k+2} - 1$ *ha* $k \geq 1$.

Bizonyítás: $k = 1,\ 2$ esetén az állítás nyilvánvaló. Ha pedig $k > 2$, akkor a rekurzió alapján indukciót használhatunk:

$$G_k = 1 + G_{k-1} + G_{k-2} = 1 + F_{k+1} - 1 + F_k - 1 = F_{k+2} - 1.$$

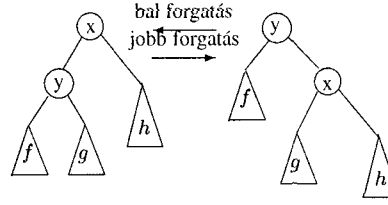
Az utolsó lépésben alkalmaztuk a Fibonacci-számokra teljesülő $F_{k+1} + F_k = F_{k+2}$ összefüggést. □

Következmény: *Egy n-pontú AVL-fa szintjeinek k száma nem több mint* $O(\log n)$, *pontosabban* $k \leq 1.44 \log_2(n+1)$.

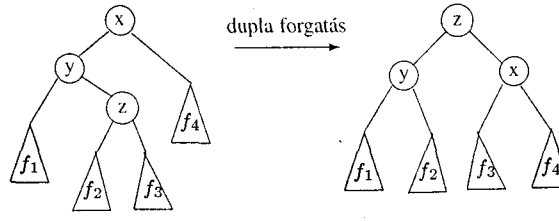
Bizonyítás: A tétel szerint $n \geq F_{k+2} - 1$. Innen a Fibonacci-számokra vonatkozó alsó becslésből $n+1 \geq \phi^k$ és ezért $\log_\phi(n+1) \geq k$, amiből $k \leq 1.44 \log_2(n+1)$. □

Az *AVL*-fák tényleg tekinthetők kiegyensúlyozott fáknak. A kérdéses c érték 1.44 körül van. Úgy is fogalmazhatunk, hogy az *AVL*-fákban való keresés hatékony, mert a magasságuk legfeljebb 1.44-szer nagyobb az ugyanannyi csúcsból álló tökéletes alakú bináris fáénál.

A kérdés ezek után, hogy miként lehet az *AVL*-tulajdonságot megőrizni, karbantartani? Hogyan lehet a BESZÚR és TÖRÖL eljárásokat úgy megvalósítani, hogy megtartsák az *AVL*-tulajdonságot? Az alapvető eszköz a *forgatás*. Legyen S egy bináris keresőfa, melynek gyökere az x csúcs, ennek bal fia y. Jelölje f az y bal, g pedig a jobb részfáját. Legyen h az x jobb részfája. A bináris keresőfa-tulajdonság megmarad, ha az S fát átalakítjuk úgy, hogy y lesz a gyökere, ennek bal részfája marad f, az y jobb fia x, ennek részfái g (bal) és h (jobb). Ezt az átalakítást jobb forgatásnak nevezzük, az inverzét pedig bal forgatásnak.



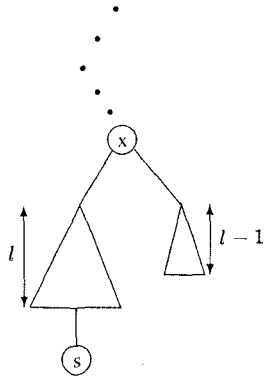
Legyen az S bináris keresőfa, gyökere az x csúcs, ennek bal fia y, y jobb fia z. Végezzünk el egy bal forgatást az y gyökerű részfára, majd egy jobb forgatást az így kapott S-re. Ennek az operációnak a neve dupla forgatás.



Dupla forgatásnak nevezzük ennek a lépéspárnak a tükörképét is, amikor y az x jobb fia, és z az y bal fia. A beszúrás és törlés megvalósítására a naiv algoritmusokat használjuk, kiegészítve azzal, hogy a művelet elvégzése után forgatásokkal visszaállítjuk (ha szükséges) az *AVL*-tulajdonságot.

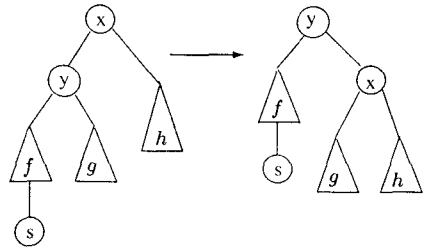
Tétel: *Legyen S egy n csúcsból álló AVL-fa.* BESZÚR(s, S) *után legfeljebb egy (esetleg dupla) forgatással helyreállítható az AVL-tulajdonság. A beszúrás költsége ezzel együtt is* $O(\log n)$.

Bizonyítás: A korábbiakkal összhangban egy f fa (z csúcs) esetén jelölje $m(f)$ ($m(z)$) az f fa (a z gyökerű részfa) magasságát. Az S fa minden csúcsában feljegyezzük az itt gyökerező részfa szintjeinek számát. Ez a mező könnyen karbantartható a naiv beszúrás és törlés során. (Valójában mindössze 2 bit/csúcs extra információ is elég az *AVL*-tulajdonság helyreállításához. Ezt itt nem részletezzük.) A naiv beszúrás után a keresési út ismételt bejárásával megkapjuk a *legalsó* olyan csúcsot (ez legyen x), ahol az *AVL*-tulajdonság megsérül.



Az x definíciójából adódik, hogy x bal és jobb részfájának nem lehet ugyanaz a magassága. Az általánosság csorbítása nélkül feltehetjük, hogy a bal részfa magasabb, jelesül nem üres. (Ha a jobb részfa a magasabb, akkor az alább következő recept tükörképével, benne a *jobb* és a *bal* felcserélésével érhetünk célt.) Legyen ezek után a bal részfa gyökérpontja y, ennek a részfái pedig f (bal) és g (jobb). Az x jobb részfája legyen h. Legyen $m(y) = l$; ekkor szükségképpen $m(h) = l - 1$. Két esetet különböztetünk meg: a beszúrás során
(a) az s az f-be kerül;
(b) az s a g-be kerül.

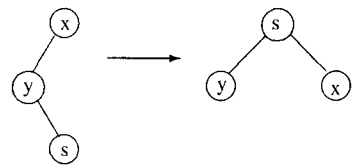
Nézzük először az (a) esetet. Ekkor $m(f) = m(g) = l - 1$. Ugyanis $m(f) < m(g)$ esetén a beszúrás nem tudná megsérteni az *AVL*-tulajdonságot x-ben. Másfelől $m(f) > m(g)$ azt jelentené, hogy y-ban is – tehát x alatt – sérül az *AVL*-tulajdonság. Innen adódik, hogy $m(f) = m(g) = m(y) - 1 = l - 1$. Eme ismeretek birtokában állítjuk, hogy az x-nél elvégzett jobb forgatás megoldja a problémát.



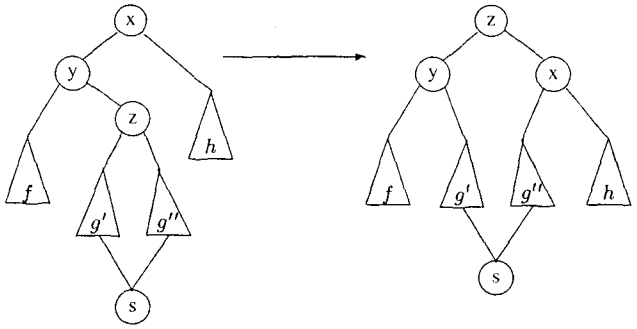
A forgatás után y mindkét részfájának a magassága l lesz, x új részfái g és h, mindkettő szintszáma $l-1$. Ezen csúcsok és részfák rendben vannak. Az x definíciója miatt f-ben az s beillesztése után sem sérülhet az *AVL*-tulajdonság. Végül megjegyezzük, hogy az új helyre került y feletti csúcsok magassága ugyanannyi marad a beszúrás és forgatás után, mint amennyi eredetileg volt; így az *AVL*-feltétel feljebb is megmarad a kereső út mentén.

Nézzük most a (b) lehetőséget, amikor is s az y jobb részfájába kerül. Ezt mindjárt két alesetre bontjuk.
(b1) az új csúcs y fia, azaz $l = 1$;
(b2) az új csúcs y-nak nem fia, más szóval $l > 1$.

A (b1) esetben y bal részfája és x jobb részfája is üres, ezért elegendő egy dupla forgatás x-nél.



A (b2) esetben a g részfa nem üres, hiszen $m(g) = l - 1 > 0$. Legyen a részfa gyökere z, ennek részfái g' és g''. Az s kulcs e két részfa valamelyikébe kerül, a továbbiak szempontjából közömbös, hogy melyikbe. Ekkor $m(f) = l - 1$ (mert y-ban az *AVL*-feltétel teljesül), és $m(g') = m(g'') = l - 2$ (mert z-ben sincs baj az *AVL*-tulajdonsággal). Egy kettős forgatás mindent helyretesz.



A részfa új gyökere z kiegyensúlyozott lesz, $m(z) = l + 1$. Ennyi volt $m(x)$ a beszúrás előtt. A z bal fia y balsúlyos vagy kiegyensúlyozott lesz aszerint, hogy s a g' vagy g'' részfába kerül-e. Ugyanígy z jobb fia x lesz, ami jobbsúlyos vagy kiegyensúlyozott. Ezzel az algoritmus ismertetését befejeztük. Az itt vázolt eljárás költsége nyilvánvalóan arányos a naiv beszúrás költségével; az $m(S) \leq 1.44 \log_2(n+1)$ egyenlőtlenségből látszik, hogy ez $O(\log n)$. □

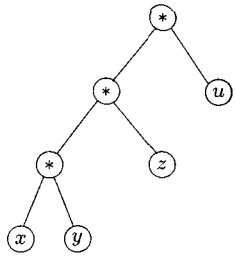
A törlés algoritmusa is hasonló felépítésű. Először a naiv módszerrel eltávolítjuk a törölni kívánt elemet, majd – ha szükséges – alkalmas forgatásokkal helyreállítjuk az *AVL*-tulajdonságot.

Tétel: *Az n szögpontú AVL-fából való naiv törlés után legfeljebb $1.44 \log_2 n$ (szimpla vagy dupla) forgatás helyreállítja az AVL-tulajdonságot.*

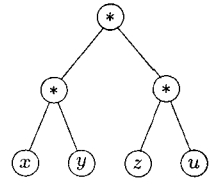
Ezt nem bizonyítjuk. A helyreállító fázis valamivel bonyolultabb, mint a beszúrásnál alkalmazott módszer. Előfordulhat, hogy a törlési út mentén több csúcsnál is kell forgatást végezni. A törlés költsége evvel együtt is $O(\log n)$.

Feladat: Adjunk példát egy olyan *AVL*-fára, melynél egy alkalmas törlés után nem állítható helyre az *AVL*-tulajdonság egyetlen (szimpla vagy dupla) forgatással.

Megjegyzés: A forgatások szoros kapcsolatban vannak az *asszociatív szabállyal*. Képzeljük el, hogy van egy kétváltozós műveletünk, mondjuk $*$. E művelet segítségével kifejezéseket készíthetünk (pl. $(x * y) * (z * u)$, ahol x, y, z, u változók). Egy kifejezésnek természetes módon megfelel egy bináris fa, melynek levelei a változók. Ebben a kifejezésfában forgatások felelnek meg a kifejezés asszociatív szabály szerinti átalakításainak.



$((x * y) * z) * u$



$(x * y) * (z * u)$

Az ábrán baloldalt levő kifejezésfából a gyökérnél elvégzett jobbforgatással keletkezett a másik fa. Látható, hogy az új kifejezés a régiből az asszociatív szabály egyszeri alkalmazásával származtatható.

3.6. További megjegyzések kiegyensúlyozott fákról

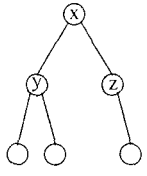
Az *AVL*-tulajdonság olyan szabályt, előírást jelent, melyet ha megtartunk, akkor kiegyensúlyozott fát kapunk. Az AVL-fák magassága legfeljebb mintegy 1.44-szerese az ideális fákénak. Ez biztosítja, hogy a bennük való keresés hatékony. Az *AVL*-tulajdonság csak egy a lehetséges kiegyensúlyozottsági feltételek közül. Más olyan tulajdonságok is vannak, amelyek hasonló módon szabályozzák a fa alakját, és a megőrzésük sem jelent túl nagy algoritmikus nehézséget. E tulajdonságok (konstrukciók) közül kettőt érintünk a továbbiakban.

HB[k]-fák (C. C. Foster, 1973)
Legyen $k \geq 1$ egy egész szám. Egy bináris keresőfa $HB[k]$-fa, ha minden x csúcsára teljesül, hogy $|m(bal(x)) - m(jobb(x))| \leq k$. Az előírás az *AVL*-feltétel általánosítása. Speciálisan a $HB[1]$-fák éppen az *AVL*-fák. Az *AVL*-fák esetéhez hasonló, bár valamivel bonyolultabb módon megmutatható, hogy a $HB[k]$-feltétel általában is kiegyensúlyozott fákat eredményez.

Feladat: Az előbbi definícióból kizártuk a $k = 0$ esetet. Mi lehet ennek a magyarázata? Miért alkalmatlanok a $HB[0]$-fák a keresőfa adatszerkezet megvalósítására?

Súlyra kiegyensúlyozott fák (J. Nievergelt, B. Reingold, C. Wong, 70-es évek)
Az előző feltétel a részfák magasságának a szabályozásával biztosította a kellően telt alakú fákat. Erre a célra más kombinatorikus tulajdonságok is alkalmasak. Ilyen jellemző lehet például a részfák csúcsainak száma.

Legyen x egy bináris fa csúcsa. Az x csúcs *súlya* az x-gyökerű részfában levő csúcsok száma. Az x csúcs súlyának a jele $s(x)$. Például a következő fában $s(x) = 6$, $s(y) = 3$ és $s(z) = 2$.



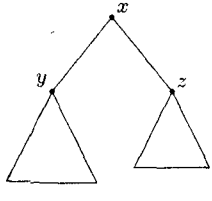
Definíció: *Egy bináris keresőfát súlyra kiegyensúlyozott fának (röviden SK-fának) nevezünk, ha minden x belső csúcsára teljesül, hogy*

$$\sqrt{2} - 1 < \frac{s(bal(x))}{s(jobb(x))} < \sqrt{2} + 1.$$

Láthatjuk, hogy az SK-feltétel jobb-bal szimmetrikus. Az egyenlőtlenségek reciprokát véve kiderül, hogy $\sqrt{2} - 1 < \frac{s(jobb(x))}{s(bal(x))} < \sqrt{2} + 1$ is igaz.

Feladat: Igazoljuk, hogy a leheletnyivel szigorúbb $1/2 < \frac{s(bal(x))}{s(jobb(x))} < 2$ korlátokat már csak az l szintből álló, $2^l - 1$ pontú bináris fák a teljesítik.

Most megmutatjuk, hogy az SK-feltétel tényleg kiegyensúlyozott fákhoz vezet; a magasságkorlátban szereplő c állandó 2-nek adódik. Legyen evégből S egy SK-fa, melyre $s(S) = n$ és $m(S) = k$. Legyen x az S egy belső csúcsa, melynek bal fia y, a jobb fia pedig z.



Ekkor $s(x) > s(y) + s(z) > (\sqrt{2} - 1)s(z) + s(z) = \sqrt{2}s(z)$. Az első egyenlőtlenség a súly definíciója, a második pedig az SK-feltétel miatt igaz. Innen az SK-feltétel jobb-bal szimmetriáját használva megállapíthatjuk, hogy érvényes az $s(x) > \sqrt{2}s(y)$ egyenlőtlenség is. Legyenek most $x_1, x_2, \ldots, x_k$

egy k-hosszúságú gyökértől levélig menő út csúcsai. Az S gyökere x_1, amiből $n = s(S) = s(x_1)$, továbbá $x_1, x_2, \ldots, x_{k-1}$ belső csúcsai S-nek. Az előbb nyert egyenlőtlenségek ismételt alkalmazásával kapjuk, hogy

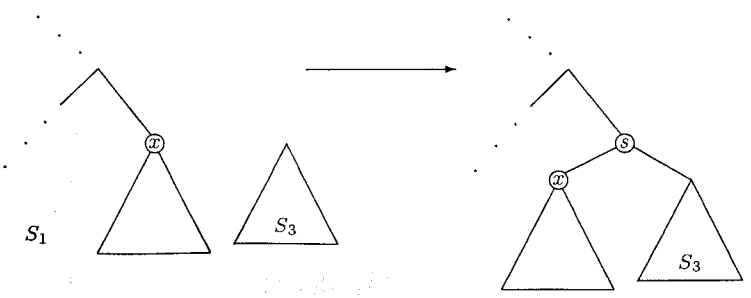
$$n = s(x_1) > \sqrt{2}s(x_2) > (\sqrt{2})^2 s(x_3) > \cdots > (\sqrt{2})^{k-1} s(x_k) = (\sqrt{2})^{k-1}.$$

A lánc elejét a végével összevetve $n > (\sqrt{2})^{k-1}$, amiből logaritmusra térve $2\log_2 n + 1 > k$ adódik. Egy n csúcsból álló SK-fa magasságának maximuma legfeljebb $2\log_2 n + 1$.

Egy további alkalmazás

A kiegyensúlyozott fák alkalmasak a tárgyalt alapfeladatokon kívül más problémák hatékony megoldására is. Példaként említjük a rendezett listák összefűzésének feladatát. Az elemek inorder sorrendjére gondolva egy *AVL*-fa felfogható rendezett listának. Tegyük fel, hogy van két rendezett listánk, S_1 és S_2, melyeket egy-egy *AVL*-fában tárolunk. Tegyük fel továbbá, hogy az S_2-ben szereplő kulcsok mind nagyobbak az S_1 kulcsainál. Célunk a két lista egyesítése; a két *AVL*-fát szeretnénk minél hatékonyabban egyetlen *AVL*-fává összefűzni. Megmutatjuk, hogy ez megtehető $O(\log n)$ költséggel, ahol n a két lista összesített elemszáma. A fákról feltételezzük, hogy a csúcsai tartalmazzák az ott gyökerező részfák magasságát. Legyen $m(S_1) \geq m(S_2)$. (A fordított eset hasonlóan kezelhető.) Az eljárás fő lépései a következők:

1. Megkeressük és töröljük S_2 legkisebb elemét (legyen ez s). A törlés eredményeként kapott fát jelölje S_3.
2. S_1 gyökerétől indulva, és rendületlenül jobbfelé haladva megkeressük az első olyan x csúcsot, melyre $m(x) - m(S_3) = 0$ vagy 1. (A feltétel azért tartalmaz két lehetőséget, mert egy lépésnél a magasság 1-gyel vagy 2-vel csökkenhet.)
3. Ezután x helyére egy új csúcsot teszünk, melynek kulcsa s. Az új csúcs jobb részfája S_3, bal részfája pedig az S_1 fa x-gyökerű részfája lesz. Végül helyreállítjuk az *AVL*-tulajdonságot. Erre a beszúró algoritmus ötletei alkalmazhatók: úgy járunk el, mintha az s csúcsot szúrtuk volna be a fába.



Könnyen ellenőrizhető, hogy az így kapott fa teljesíti a keresőfa-tulajdonságot és az *AVL*-feltételt is. A költségkorlát azonnal következik abból, hogy S_1 és S_2 magassága is $O(\log n)$.

3.7. Egy önszervező megoldás: az S-fák

Kvasztics felriadt álmából, meglátta az újult őrt, és fáradtan körülnézett.
– Csak menjetek nyugodtan... – mondta – az őrt megvizsgálom és ellátom kötéssel.
– Az nem elég – súgta Tuskó – a száját is tömje be, ha már ápolás alá veszi.
REJTŐ JENŐ: A három testőr Afrikában

Az eddig vett hatékony keresőfák algoritmusai zord szigorúsággal vigyáznak a fák alakjára. Ezáltal a műveletek legkedvezőtlenebb eseteire is biztosítani tudják a $O(\log n)$ nagyságrendű korlátot, ahol n a tárolt elemek száma. Egészen másféle szemléletet képvisel az itt bemutatásra kerülő önszervező bináris keresőfa-konstrukció, melynek neve S-fa. Az elnevezés az angol *splay tree* (kifordított fa) kezdőbetűjéből származik. Az S-fákat D. D. Sleator és R. E. Tarjan javasolták 1983-ban.

Egy S-fa a tárolási szerkezetét illetően egyszerűen egy bináris keresőfa. Érdekessé és hatékonnyá az alapműveletekre javasolt módszerek teszik. Ezek az algoritmusok semmiféle közvetlen figyelmet nem szentelnek a fa alakjának. Előfordulhat például, hogy a fában hosszú utak vannak, és ezért bizonyos keresések lassúak. Az S-fák tehát nem feltétlenül kiegyensúlyozottak. Ezzel szemben hosszabb operációsor (keresések, beszúrások, törlések, stb.) alatt „tanulnak": az egyes elemek elérési gyakoriságai és ezen elérések időbeli eloszlása szerint változtatják alakjukat. Úgy is mondhatjuk, hogy a fa idomul a felhasználói igényekhez.

Az S-fák algoritmusai mögött egy nagyerejű és sok esetben használható gondolat húzódik meg. Ha a felhasználói igények teljesítése (mondjuk keresés) közben eljutunk valahova, akkor nem távozunk onnan szűkkeblűen, pusztán csak a kötelezőket elvégezve. Rászánunk még valamennyi időt, hogy szépítsük, alakítsuk a környéket ahová kerültünk. Ez az idő (nagyságrendileg) nem több, mint amit a felhasználói kérésre amúgy is fordítanunk kell. A ráfordítás javítja a szerkezet hatékonyságát, aminek az eredménye az lesz, hogy a jövőbeli kéréseket gyorsabban tudjuk teljesíteni. Ezzel a filozófiával később is találkozni fogunk a szekvenciális keresés önszervező módszereinél és az UNIÓ-HOLVAN adatszerkezetnél.

Az S-fák alkalmazkodó képességének két konkrétabb megnyilvánulását említjük:

1. Egy hosszú műveletsor esetén az egy műveletre eső költség konstans szorzó erejéig optimális lesz. Ha tehát a fa elég sokáig él, akkor az összesített időigénye nem lesz számottevően rosszabb a legjobb kiegyensúlyozott fákénál. E tulajdonság pontosabb megfogalmazása a szakasz végén található tétel.

2. A kiegyensúlyozott fáknál jobb teljesítményt nyújtanak olyan alkalmazási helyzetekben, ahol az egyes elemekkel kapcsolatos elérési igények „időben csomósodnak". Az ilyen csomós helyzetre suta, de szemléletes példa egy kórház betegnyilvántartása. (A sutaság abból ered, hogy egy kórházi nyilvántartást külső táron érdemes kezelni, míg a bináris keresőfák belső memóriás szerkezetek.) Ha valaki bekerül a kórházba, akkor a róla szóló rekord igen aktív lesz; viszonylag gyors egymásutánban kerülnek bele a különféle vizsgálatokkal, kezelésekkel kapcsolatos feljegyzések. Ha viszont az illető éppen nem beteg, akkor meglehetősen ritkán van szükség erre a rekordra.

Az S-fák ezen alkalmazkodási képességét egy igen egyszerű, ugyanakkor csillogóan elegáns visszacsatolási mechanizmus biztosítja. Ennek lényege, hogy ha a fában tárolt $s \in U$ elemmel kapcsolatos elérési igény érkezik (pl. KERES(s, f)), akkor az igényt kezelő algoritmus ezt úgy „értelmezi", hogy az s fontossága nőtt. Az s elemet alkalmas forgatásokkal a fa gyökerébe mozgatja. Ezen felül az s-hez vezető kereső út mentén levő elemek is valamivel fontosabbak lettek; a módszer őket is közelíti a gyökérhez. Ennek eredményeként az adott időszakban gyakran használt elemek közel lesznek a fa tetejéhez, a kevésbé aktívak pedig lassan a levelek felé vándorolnak. A gyakran használt elemek ezért nagyon gyorsan elérhetők. A kórházi példánál maradva a fa csaknem olyan teljesítményt nyújt, mintha csupán az éppen kezelt betegek rekordjaiból állna a nyilvántartás.

Az S-fák műveleteinek pontos leírásához bevezetünk néhány jelölést. Legyenek f, f' S-fák, x, y, z kulcsok, az U rendezett univerzum elemei. Utóbbiakat azonosítani fogjuk az őket tartalmazó fabeli csúcsokkal. Ez nem fog félreértést okozni.

A keresőfákra jellemző KERES(x, f), BESZÚR(x, f) és TÖRÖL(x, f) műveleteket a szokásos módon értelmezzük. A RAGASZT(f, f') művelet az f és f' S-fákból egyetlen S-fát szervez, feltéve, hogy $x < y$ teljesül minden $x \in f$ és $y \in f'$ kulcsra. A VÁG(x, f) művelet szétvágja f-et az f' és f'' S-fákra úgy, hogy $y \leq x \leq z$ teljesül minden $y \in f'$ és $z \in f''$ csúcsra.

A korábban említett önszervező-szépítő képesség a KIFORDÍT elnevezésű eljárásba van építve. Ennek a definíciója az erejéhez képest meglepően egyszerű.

KIFORDÍT(x, f) átszervezi az f S-fát úgy, hogy x lesz az új gyökér, ha $x \in f$; különben f gyökere x valamelyik szomszédja lesz:
vagy $\max\{y \in f;\ y < x\}$ vagy $\min\{y \in f;\ y > x\}$.

A KIFORDÍT művelet az egész módszercsalád lelke. Segítségével a többi eljárás könnyen megvalósítható:

Állítás: *Az ismertetett műveletek mindegyike megvalósítható konstans számú KIFORDÍT és konstans számú elemi operáció (összehasonlítás, mutató beállítás, stb.) segítségével.*

Bizonyítás: Csak két művelet, a RAGASZT és a TÖRÖL esetét nézzük meg közelebbről. A többi módszer összerakását feladatként az olvasóra hagyjuk.

RAGASZT(f, f') végrehajtását a KIFORDÍT($+\infty, f$) hívással kezdjük. Itt $+\infty \in U$ egy az f minden eleménél nagyobb kulcsot jelent. Legyen x az eredményül kapott f^* fa gyökere. A KIFORDÍT specifikációja szerint az x az f^* legnagyobb kulcsa. Ebből következik, hogy x-nek nincs jobboldali fia. Legyen tehát az x jobboldali fia az f' fa gyökere. Az így kapott fára teljesül a keresőfa-tulajdonság, mert a hívási feltétel szerint f' kulcsai nagyobbak x-nél.

TÖRÖL(x, f) első lépése KIFORDÍT(x, f). Ha az ekkor kapott fa gyökere nem x, akkor készen vagyunk; ez azt jelenti, hogy $x \notin f$. Feltehetjük ezután, hogy a fa gyökere x. Legyenek f_1 és f_2 az x gyökér részfái. A törlést RAGASZT(f_1, f_2) végrehajtásával fejezhetjük be. □

KIFORDÍT(x, f) **implementációja**

Hasznos lesz a következő jelölés: legyen x az f fa egy csúcsa, melynek apja y; ez esetben FORGAT(x) jelölje azt az egyszeres forgatást, mely x-et y apjává teszi.

Először a bináris keresőfákban szokásos módszerrel megpróbáljuk megtalálni x-et f-ben. Ez vagy sikerül, vagy pedig ha $x \notin f$, akkor az x-nek a rendezés szerinti egyik szomszédjánál ($\max\{y \in f;\ y < x\}$ vagy $\min\{y \in f;\ y > x\}$) végzünk. Mindegyik esetben azt az elemet kell majd felvinnünk a fa tetejére, amelyet a keresés során utoljára elértünk. Feltehetjük ezért, hogy $x \in f$, és már meg is leltük x-et f-ben.

Ezután a kezünkben levő x elemet az alább következő recept ismételt alkalmazásával felvisszük a fa tetejére. Az eljárásdarab egy végrehajtása maximum két szinttel viszi feljebb x-et. Egy menetben a 0-3. lépések közül pontosan egy hajtódik végre. A döntéshez szükséges ellenőrzés könnyű, mert az x-hez vezető kereső út utolsó néhány (legfeljebb három) csúcsának és a köztük menő éleknek az ismeretében elvégezhető.

(0) Ha x gyökér, akkor készen vagyunk.
(* A továbbiakban jelölje y az x apját. *)
(1) Ha x-nek nincs nagyapja, akkor FORGAT(x), különben
(2) ha x és y is baloldali (jobboldali) gyerek,
akkor FORGAT(y), majd FORGAT(x), különben
(3) ha x és y különböző oldali gyerekek,
akkor FORGAT(x), majd ismét FORGAT(x).

KIFORDÍT(x, f) fontos járulékos hatása, hogy az x kereső útján levő csúcsok közelebb kerülnek a fa tetejéhez. A keresőfa-műveletek – a bennük levő KIFORDÍT-hívásnak köszönhetően – változtatják a fa alakját. Ez a változtatás annál több csúcsot érint, minél mélyebben levő elemre alkalmazzuk a kifordító eljárást.

A következő eredmény a korábban említett 1. tulajdonság pontosabb megfogalmazása. Lényegében azt mondja, hogy egy hosszú műveletsor összköltsége állandó szorzó erejéig a kiegyensúlyozott fákra jellemző korláton belül marad: az egy műveletre eső átlagos költség $O(\log n)$, ahol n a fa csúcsainak a száma. A bizonyítást mellőzzük.

Tétel: *Egy üres S-fából induló olyan m műveletből álló sorozat költsége, melyben n beszúrás van,* $O(m \log n)$. □

3.8. Szófák

Az eddigi keresőfa-konstrukciók csak annyit tételeztek fel az U kulcshalmazról, hogy az rendezett halmaz. A rendező módszereknél a kulcsokról való finomabb ismeretek hatékonyabb algoritmusokhoz vezettek; gondoljunk csak a láda- vagy a radixrendezésre. Némiképp hasonló a helyzet a keresőfa-műveleteknél. Bizonyos speciális szerkezetű kulcsokra érdekes *ad hoc* megoldások adhatók. Szép és hasznos példaként említhetők a *szófák*. A szerkezet angol neve egy fantáziaszó: *trie*, amely a retrieval szó közepéből való négy betűre szándékozik utalni (kiejtése viszont a *try* mintáját követi).

Legyen Σ egy véges halmaz, amit úgy tekintünk, mint egy nyelv betűkészletét (*abc*-jét). Jelölje Σ^* a Σ-beli elemekből alkotott véges hosszú sorozatok, azaz szavak halmazát. Itt most feltesszük, hogy adott egy rendezés (amit *abc*-sorrendnek tekintünk) a Σ halmazon. Ebből azonnal adódik egy rendezés Σ^*-on, nevezetesen a betűrenden alapuló lexikografikus rendezés. A szófák hatékony keresőfa-

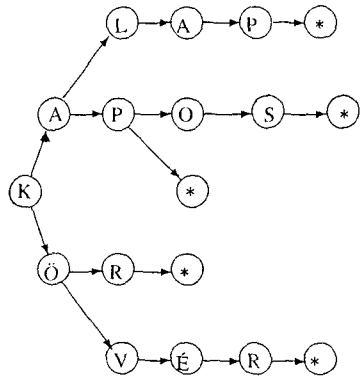
konstrukciót adnak arra az esetre, amikor $U = \Sigma^*$, a rendezés pedig a lexikografikus. A szófák tehát akkor alkalmazhatók, ha a kulcsok szavak.

A szófát gyökeres, a gyökértől távolodóan irányított fának képzelhetjük el. A fa csúcsai lényegében mutató típusú tömbök, $\Sigma \cup \{*\}$ elemeivel indexelve. Itt $* \notin \Sigma$ egy speciális végjel, amiről feltesszük, hogy a betűrend szerint Σ minden betűje megelőzi. Úgy is fogalmazhatunk, hogy a tömbök utolsó, $|\Sigma|+1$. elemének az indexe $*$. A tömbök elemeiben levő mutatók adják a fa éleit. Egy csúcsnak ezek szerint maximum $|\ \Sigma\ | + 1$ fia lehet. Egy tömbelem vagy NULL (jelezve hogy üres, tehát nem tartalmaz igazi mutatót) vagy egy mutató egy fiúra (azaz egy másik tömbre) vagy a $*$ végjel. A fában tárolt szavakat gyökértől levélig, azaz a $*$ végjelig menő utak jelentik.

Például ha Σ a magyar *abc*, és a KAP szót szeretnénk tárolni, akkor a gyökeret jelentő tömb K indexű eleméből mutató mutat egy másik tömbre, ennek A indexű elemében mutató van egy további x tömbre, melyre $x[\mathrm{P}] = *$. Bonyolultabb a helyzet, ha a KAPU szó is szerepel a szófánkban. Ekkor $x[\mathrm{P}]$ valódi mutató, mondjuk az y tömbre. Azt a tényt pedig, hogy a KAP szó is szerepel, úgy jelezhetjük, hogy $*$-ot írunk $y[*]$-ba. Másképpen fogalmazva: a problémát, hogy egyik szó prefixe a másiknak – mint a KAP a KAPU-nak – úgy kezeljük, hogy minden szó végére egy $*$-ot képzelünk, és $*$ csak a szó végén fordulhat elő. Az $x[\mathrm{P}]$-ben gyökerező részfa azoknak a tárolt szavaknak felel meg, amelyeknek az első három betűje KAP.

A tömbök helyett lineáris listák is használhatók. Ezáltal megtakaríthatjuk a NULL értékű tömbelemek helyét. Ennél a megközelítésnél viszont némi gondot okoz a listák hatékony kezelése.

Legyen Σ továbbra is a magyar *abc*. A KÖR, KÖVÉR, KAPOS, KAP, KALAP szavakat tartalmazó fa listás ábrázolása így képzelhető el:



Itt egy adott csúcs fiai tekinthetők egy listán levőknek. Így például a gyökér egy

egyelemű lista, a második szinten levő A betű fiai, L és P szintén egy (kételemű) listát képeznek.

A szófa adatszerkezet érzéketlen a beszúrások sorrendjére, a fa alakja csak a tárolt szavak összességétől függ. A tömbbel megvalósított szófában való keresés igen hatékony. Ugyanis a bemenő szó hosszához mérten nem kell többet lépnünk a mutatók mentén, mint amennyi a szóban a betűk száma. Megmutatható, hogy m betűs *abc* feletti n véletlen kulcsból álló szófa esetén a keresés átlagosan mintegy $\log_m n + c$ mutató követésével befejeződik, ahol c az m-től és n-től is független állandó.