Eötvös Loránd Tudományegyetem
Informatikai Kar
Algoritmusok és Alkalmazásaik Tanszék

Bináris keresőfák műveleteinek szemléltetése

Témavezető:
dr. Ásványi Tibor
egyetemi docens

Készítette:
Maleskovits Dávid Péter
programtervező informatikus BSc, nappali

Budapest, 2009

Tartalomjegyzék

Bevezetés

Az algoritmusok és adatszerkezetek oktatásában nagy szerepet kapnak a bináris keresőfák, illetve azok speciális változatai, az AVL fák és a piros-fekete fák. Szakdolgozatom tárgya egy olyan program elkészítése, mely ezen fontos adatszerkezetek tanulásában és megértésében nyújt segítséget.

Dolgozatom része egy elméleti összefoglalás, amiben röviden ismertetem a program használatához szükséges fogalmakat, algoritmusokat. Az algoritmusok leírása során célom, hogy az azok lényegét mutassam be.

A program különlegessége, hogy a tárgyalt adatszerkezeteket képes akár egyszerre, párhuzamosan szemléltetni.

1 Elméleti összefoglalás

1.1 Keresőfák

A keresőfák olyan adatszerkezetek, melyek adataink hatékony tárolását segítik elő, azaz a keresés, beszúrás és törlés műveleteik gyorsak. Adatainkhoz ekkor egy halmazból – melyen teljes rendezés van - egy (esetleg több) kulcsot rendelünk. A keresőfák algoritmusai szempontjából ezen kulcsok a lényegesek, ezért élhetünk azzal az egyszerűsítéssel, hogy a szemléltető programban úgy tekintünk adatainkra, mintha azok csak kulcs mezőből állnának.

1.2 Bináris fák és lehetséges bejárásaik

A bináris fák rendelkeznek egy kitüntetett csúccsal, amit a fa gyökerének nevezünk. Minden csúcsból legfeljebb két él indulhat ki, melyek eggyel mélyebb szinten elhelyezkedő csúcsokba futnak. Ezen csúcsokat gyerek csúcsoknak (jobb gyerek, bal gyerek) nevezzük, a csúcsot melyből az élek kiindulnak, a gyerek csúcsok szülőjének. Egy csúcs levél, ha egyetlen gyereke sincs. Egy bináris fa magasságán a szintjeinek a számát értjük [2].

A bináris fákat általában pointeresen ábrázoljuk, ez alól jellegzetes kivétel a kupac adatszerkezet, amit többnyire tömbösen szoktunk.

Gyakori feladat, hogy egy bináris fában tárolt elemeken valamilyen műveletet kell végrehajtani, például ki kell írnunk képernyőre a bennük tárolt információt. Erre nyújtanak eszközt a bejárások. A preorder, inorder, posztorder és szintfolytonos bejárások algoritmusai a következők:

```
Preorder(x)
 if x ≠ null then
  feldolgoz(x.cimke)
  preorder(x.bal)
  preorder(x.jobb)
```

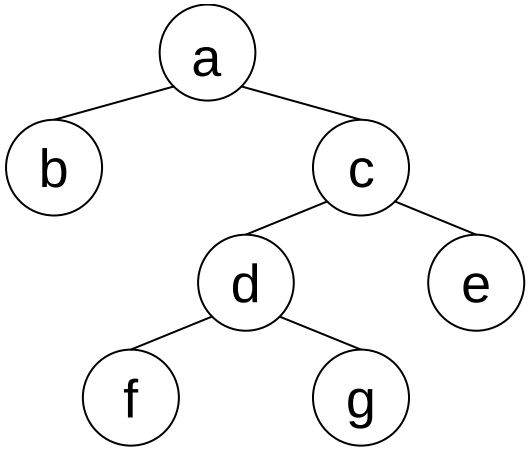
```
Inorder(x)
 if x ≠ null then
  Inorder(x.bal)
  feldolgoz(x.cimke)
  Inorder(x.jobb)
```

```
Posztorder(x)
 if x ≠ null then
  Posztorder(x.bal)
  Posztorder(x.jobb)
  feldolgoz(x.cimke)
```

```
Szintfolytonos(x)
 Üres(s)
 p:=x
 while (p ≠ null)
  feldolgoz(p.cimke)
  if p.bal ≠ null then
   Sorba(s, p.bal)
  if p.jobb ≠ null then
   Sorba(s, p.jobb)
  if ¬Üres-e(s) then
   Sorból(s, p)
  else
   p=null
```

Az algoritmusokat egy csúccsal paraméterezzük. Mind a négy bejárás műveletigénye Θ(n). Az 1.1. ábrán látható bináris fa csúcsainak felsorolása a különböző bejárások szerint:

- Preorder: a,b,c,d,f,g,e
- Inorder: b,a,f,d,g,c,e
- Posztorder: b,f,g,d,e,c,a
- Szintfolytonos: a,b,c,d,e,f,g



1.1 ábra.

1.3 Bináris keresőfák

definíció

A bináris keresőfák olyan bináris fák, melyekre érvényes a keresőfa-tulajdonság: minden x csúcsra igaz, hogy x bal részfájában minden elem kulcsa kisebb x kulcsánál, és x jobb részfájában minden elem kulcsa nagyobb x kulcsánál.

Ahogy a definícióból is látszik, mi olyan keresőfákkal foglalkozunk, melyekben a kulcsismétlődés nem megengedett.

Az egyes csúcsok rendelkeznek gyerekekre mutató pointerekkel, valamint rendelkezhetnek szülőre mutató pointerrel is. A gyökérelem szülő pointere, illetve nem létező gyerek esetén az adott csúcs megfelelő gyerek pointere null.

1.3.1 Keresés

Egy elem keresése során a fa gyökéreleméből indulunk. Ha az adott csúcs kulcsa nagyobb, mint amit keresünk, akkor a bal gyerekére lépünk, ha kisebb, akkor a jobb

gyerekére, ha egyenlő, akkor megtaláltuk a keresett elemet. Ezt addig ismételjük, amíg meg nem találjuk a keresett elemet, vagy nem létezik a csúcs, ahova lépnünk kellene. Az utóbbi eset azt jelenti, hogy nem tartalmazza a fa a keresett elemet.

Mivel minden iterációban eggyel mélyebb szintre lépünk a keresőfában, az eljárás maximum a fa magasságával megegyező számú lépésben véget ér, azaz a keresés műveletigénye O(h). (h-val a fa magasságát jelöljük)

1.3.2 Beszúrás

Beszúrás során először egy keresést hajtunk végre a beszúrandó elemre. Ha megtaláltuk, akkor nem csinálunk semmit (kulcs duplikátumokat nem engedünk meg), ha nem találtuk, akkor beszúrjuk az elemet oda, ahol a keresés leállt, azaz létrehozunk egy új csúcsot valamelyik levélelem gyerekeként. Abban az esetben, ha a fa még üres volt, akkor a fa gyökérelemeként szúrjuk be az új elemet. A beszúrás műveletigénye O(h).

1.3.3 Minimum

Minimumot számolhatunk bármelyik csúcs (mint gyökérelem) által meghatározott részfára. A gyökérelemből indulva minden lépésben megpróbálunk az aktuális csúcs bal gyerekére lépni. Ha már bal gyerek hiányában nem tudunk lépni, akkor megtaláltuk a részfa minimumát. Műveletigénye O(h).

1.3.4 Maximum

A maximumkeresés művelete a minimumkereséssel analóg módon definiálható. Ebben az esetben minden iterációban a jobb oldali gyerek felé lépünk.

1.3.5 Törlés

Törlés során először egy keresést hajtunk végre a törlendő elemre. Ha nem találtuk meg, akkor nem csinálunk semmit, ha megtaláltuk, akkor három lehetőség van. Ha a törlendő csúcsnak nincs gyereke (azaz egy levélelem), akkor egyszerűen kitöröljük a csúcsot. Ha egy gyereke van, akkor a csúcs törlése után a gyereke kerül a helyébe. Ha két gyereke van, akkor visszavezetjük a problémát az egy gyerekes esetre következő módon: megkeressük a törlendő csúcs jobb részfájának minimumát. Ezen csúcsnak már legfeljebb csak egy gyereke lehet, azaz ezt már ki tudjuk törölni, de mielőtt

megtennénk, másoljuk át értékét a törlendő csúcsba. Belátható hogy ezzel a cserével a keresőfa-tulajdonság nem sérül. A törlés műveletigénye O(h).

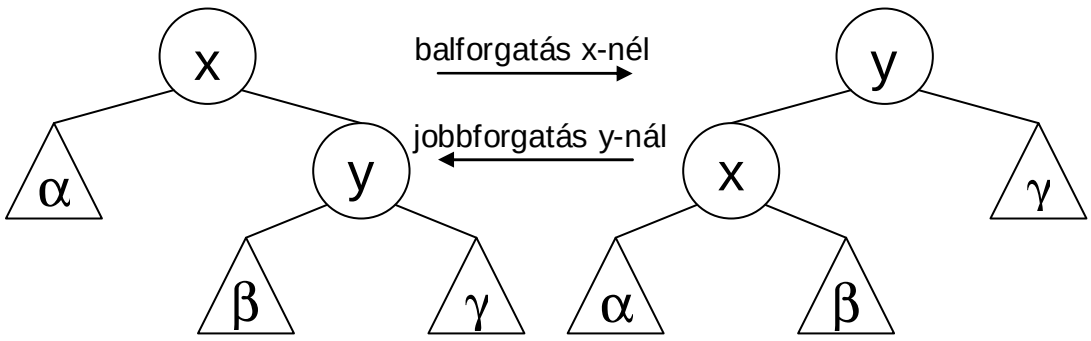
1.3.6 Műveletigény

A bináris keresőfa alapműveleteinek végrehajtási ideje a fa magasságától függ. Egy teljes fa esetén ez O(lg n), eltorzult fáknál O(n), a legrosszabb esetben Θ(n) idő szükséges egy alapművelet végrehajtásához (például, ha a fa építése során kulcsuk szerint növekvő sorrendben szúrjuk be az elemeket). A gyakorlatban fontos, hogy egy adott keresőfán a műveletek végrehajtási ideje nagyságrendileg csak a fában tárolt elemek számától függjön, a műveletsortól - ami a fa kialakulásához vezetett – ne. Ugyanakkor a keresőfák esetleges műveletek utáni karbantartásának nem lehet akkora a költsége, hogy a végrehajtási idő nagyságrendben romoljon

A következőkben két speciális bináris keresőfáról lesz szó, melyek megoldást jelentenek a fentebb vázolt problémára. Előbb azonban ismerkedjünk meg a forgatásokkal, melyek segítségével úgy módosíthatjuk egy keresőfa szerkezetét, hogy közben nem rontjuk el a keresőfa-tulajdonságot.

1.3.7 Forgatások

Bináris fák bizonyos csúcsainál végezhetünk forgatásokat. Balforgatás esetén feltétel, hogy a kiválasztott csúcsnak legyen jobb gyereke, jobbforgatásnál pedig legyen bal gyereke. Látható, hogy a forgatások a fa csúcsainak elhelyezkedését csak a kiválasztott elem által meghatározott részfában változtatják meg, valamint ha a fa bináris keresőfa volt, akkor a keresőfa-tulajdonság a forgatás után is fennáll.



1.2 ábra. Forgatások

A forgatások műveletigénye O(1), hiszen minden esetben 6 pointert kell átállítani.

1.4 Az AVL fák

Az AVL fák minden csúcs esetén számon tartanak egy plusz tulajdonságot, a csúcs egyensúlyi állapotát, ami az adott csúcs jobb és bal oldali részfáinak magasságkülönbsége.

definíció

Egy bináris keresőfa kiegyensúlyozott (AVL-tulajdonságú), ha az összes csúcsának az egyensúlyi állapota -1 és 1 közé esik.

tétel

Egy AVL fa magassága legfeljebb $1{,}44*\log_2(n)$, ahol n a fa csúcsainak a száma.

A tétel bizonyítása megtalálható [2] 72. oldalán.

Beszúrás és törlés esetén az AVL-tulajdonság elromolhat, ezért ha szükséges, forgatások segítségével helyre kell azt állítani. A csúcsok egyensúly-faktorának jelölésére vezessük be a szemléletes --, -, = , +, és ++ szimbólumokat (--, ha az adott csúcs bal oldali részfája kettővel mélyebb a jobb oldalinál, ++ ha a jobb oldali részfája kettővel mélyebb a balnál). Az AVL fák vizsgálata során más egyensúlyi állapot nem fordulhat elő, hiszen mindkét művelet előtt egy kiegyensúlyozott fánk van, mely csúcsainak egyensúlyfaktora - az AVL fa definíciójából adódóan - csak -, =, vagy + lehet. Egy beszúrás elvégzése után néhány részfa magassága eggyel nő, törlés elvégzése után pedig eggyel csökken, vagyis a csúcsok egyensúlyi állapota legfeljebb eggyel változhat az eredeti állapothoz képest (így keletkezhet --, és ++ állapot). Az algoritmusok tanulmányozása során kiderül majd, hogy a helyreállítás során sem alakulhat ki az említetteken kívüli egyensúly-faktor.

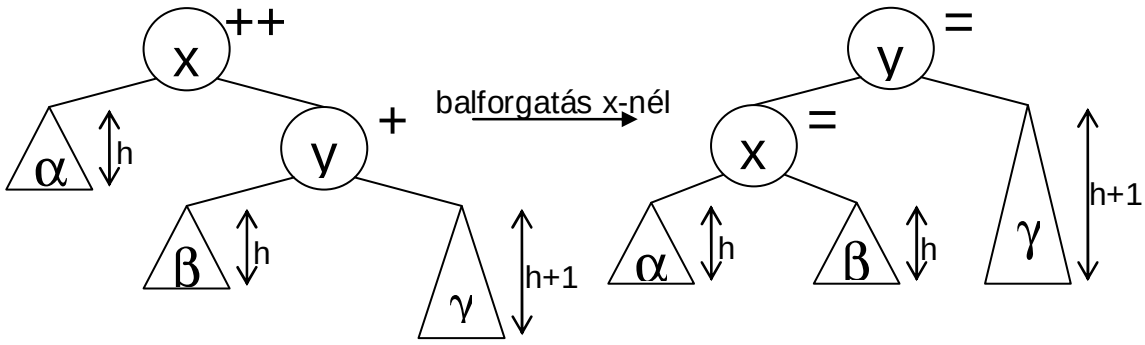
Most megnézzük, milyen módon romolhatnak el az AVL fák egy beszúrás vagy törlés következtében. Látni fogjuk, hogy minden esetnek bekövetkezhet a tükörképe is, de ezek helyreállítása a jobb és bal felcserélésével adódik az alapalgoritmusokból (a helyreállító algoritmus ”tükrözésével”).

1.4.1 Beszúrás

A beszúrást követően elindulunk a beszúrt csúcstól fölfele a fán, és módosítjuk a szülők egyensúlyi állapotát, ha szükséges. Ha egy csúcs új egyensúly-faktora ++ vagy -- lenne, akkor forgatnunk kell. Kétféle módon romolhatott el az AVL fa az adott pontban.

1. (++, +) eset (tükrözése a (--, -))

Ekkor, az 1.3. ábrán látható módon, egy a ++-os csúcsra végrehajtott balforgatással helyreállítjuk az AVL-tulajdonságot a részfában. Nyilván ebben az esetben a γ részfába szúrtuk be az új elemet, tehát az eredeti fában az x gyökerű részfa magassága h+2 volt. Figyeljük meg, hogy a forgatás után a részfa magassága ismét h+2, azaz nem kell tovább fölfelé haladnunk a fában, hiszen biztosan nem lennének további egyensúlyi állapot módosítások.



1.3 ábra. (++,+) eset

2. (++, -) eset (tükrözése a (--, +))

Ekkor két forgatást is végre kell hajtanunk az 1.4. ábrán látható módon. A részfák magassága nem egyértelmű, attól függően, hogy hova került az új csúcs, különböző párosítások lehetségesek. Ha az új elem a Z csúcs, akkor h=0 (Z-nek nincsenek gyerekei). Ha az új elemet a γ részfába szúrtuk be, akkor m(β)=h-1 és m(γ)=h, ha pedig a β részfába szúrtuk be, akkor m(β)=h és m(γ)=h-1. Az x gyökerű részfa magassága eredetileg h+2 volt, és a forgatások eredményeként ismét h+2 lesz, vagyis ahogy a (++, +) esetben sem kellett, most sem kell tovább haladni a fán, hiszen egyensúly-faktor módosításokra már nem kerülne sor.

3**1.4 ábra.** (++,-) eset

1.4.2 Törlés

A törlést követően elindulunk a kivágott csúcstól fölfele a fán, és aktualizáljuk a szülők egyensúlyi állapotát. Ha egy csúcs új egyensúly-faktora ++ vagy -- lenne, akkor forgatnunk kell. Törlés következtében háromféle módon romolhat el az AVL fa.

1. (++, +) eset (tükrözése a (--, -))

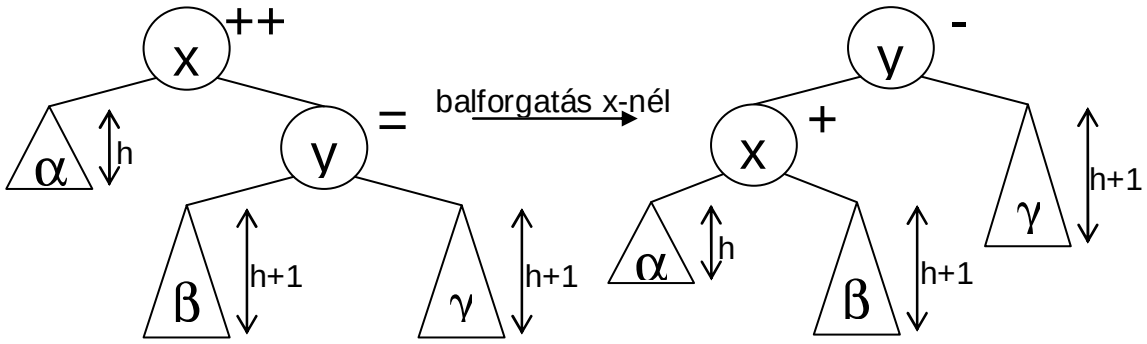
Ekkor ugyanazt kell tennünk, mint amit a beszúrásnál (ahogy az 1.3. ábra mutatja), a különbség az, hogy törlésnél az eredeti részfa magassága nem h+2, hanem h+3, azaz forgatás után a részfa magassága eggyel csökken, ezért tovább kell haladnunk fölfele a fán.

2. (++, -) eset (tükrözése a (--, +))

Ekkor szintén ugyanazt kell tennünk, mint amit a beszúrás (++,-) esetében, azonban törlésnél most is tovább kell haladnunk a fán, ugyanis a részfa magassága h+3-ról h+2-re csökken (1.4. ábra).

3. (++, =) eset (tükrözése a (--, =))

Ez az eset beszúrásnál nem fordulhat elő. Az 1.5. ábrán látható módon, egy a ++-os csúcsra végrehajtott balforgatással helyreállítjuk az AVL-tulajdonságot a részfában. Mivel a részfa magassága kezdetben is, és a forgatás után is h+3 (hiszen csak az α-ból törölhettük ez elemet) nem kell tovább fölfelé haladnunk a fában.



1.5 ábra. (++,=) eset

1.5 A piros-fekete fák

A piros-fekete fák is plusz információt tartanak számon a csúcsokról, mégpedig a színüket, ami piros vagy fekete lehet. A piros-fekete fák esetében úgy tekintjük, hogy a NULL gyerekek a fák levelei, azaz értékes információt csak a belső csúcsok tárolnak.

definíció

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

- Minden csúcs színe piros vagy fekete.
- A gyökércsúcs színe fekete.
- Minden levél (NULL) színe fekete.
- Minden piros csúcsnak mindkét gyereke fekete.
- Bármely csúcsból bármely levélig vezető úton ugyanannyi fekete csúcs van.

Beszúrás és törlés következtében a definícióban felsorolt tulajdonságok elromolhatnak, ezért ha szükséges, forgatások és átszínezések segítségével helyre kell azt állítani.

Vezessük be a piros-fekete fák csúcsaira a fekete-magasság fogalmát. Egy x csúcsból kiinduló levélig vezető tetszőleges úton található fekete csúcsok számát (x csúcsot nem számolva) az x csúcs fekete-magasságának nevezzük. Ez a szám a piros-fekete fák 5. tulajdonsága miatt egyértelműen meghatározott. Egy részfa fekete-magasságán a részfa gyökerének fekete-magasságát értjük.

tétel

Egy piros-fekete fa magassága legfeljebb $2*\log_2(n + 1)$, ahol n a fa belső csúcsainak a száma.

A tétel bizonyítása megtalálható [1] 251. oldalán.

Megjegyzés az ábrákhoz: A piros-fekete fák algoritmusait szemléltető ábrákon csak az algoritmusok szempontjából lényeges elemeket tüntettem fel. A fehér színnel jelölt csúcsok valódi színe lehet akár piros, akár fekete.

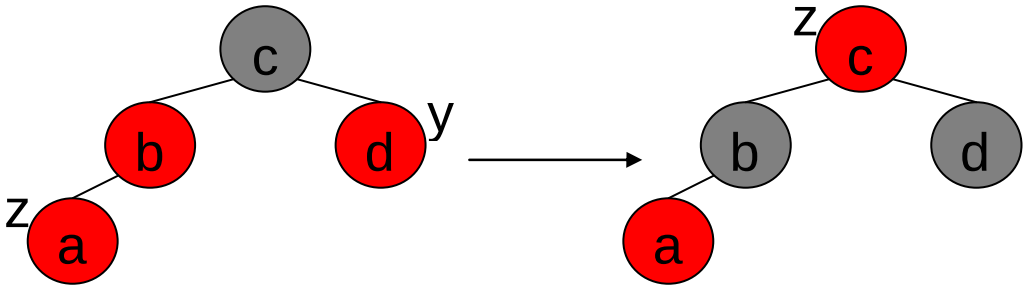
1.5.1 Beszúrás

Végrehajtjuk a bináris keresőfákra alkalmazott beszúrást. Jelöljük az új elemet z-vel.

0. Ha z a keresőfa gyökéreleme, akkor színezzük feketére (és ezzel készen is vagyunk), egyébként színezzük pirosra. Figyeljük meg, hogy ha kezdetben egy piros-fekete fánk volt, akkor ezt követően csak a 4. tulajdonság lehet elromolva. Ha z szülője

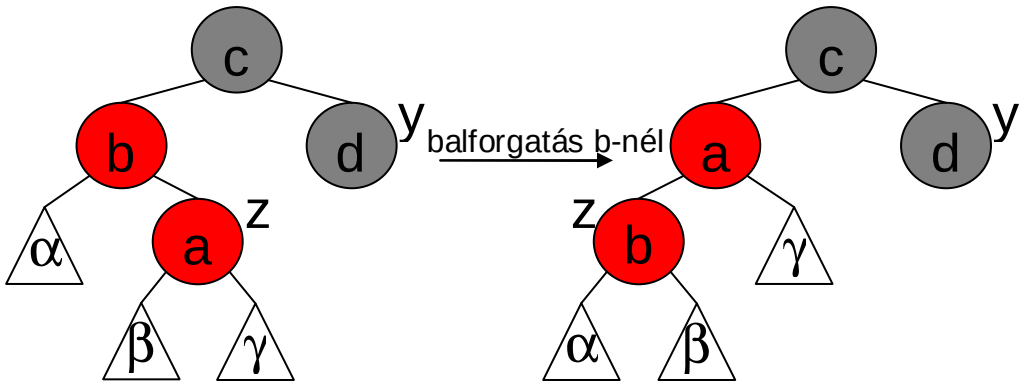
fekete, akkor készen vagyunk, ha piros, akkor hat eset valamelyike áll fenn. Legyen y z nagybátyja, azaz z szülője szülőjének a másik gyereke. Ez a csúcs nyilván létezik, hiszen z szülője piros, ezért nem lehet a gyökérelem. Az viszont nincs kizárva, hogy y egy (a piros-fekete fák kontextusában értéktelen) levélelem. Attól függően, hogy y jobb gyerek, vagy bal gyerek, 3-3 szimmetrikus esetről beszélhetünk, melyekhez tartozó algoritmusok a jobb és bal felcserélésével adódnak a párjukból, ezért mi csak azon három esettel foglalkozunk, mikor az y jobb gyerek.

1. Ha y színe piros, akkor az 1.6. ábrán látható átszínezéssel a problémát fentebb toljuk a fában (z szülőjét és y-t feketére, y szülőjét pirosra színezzük, az új z pedig legyen y szülője), majd kezdjük újra az 0-tól.



1.6 ábra. Beszúrás, 1. eset

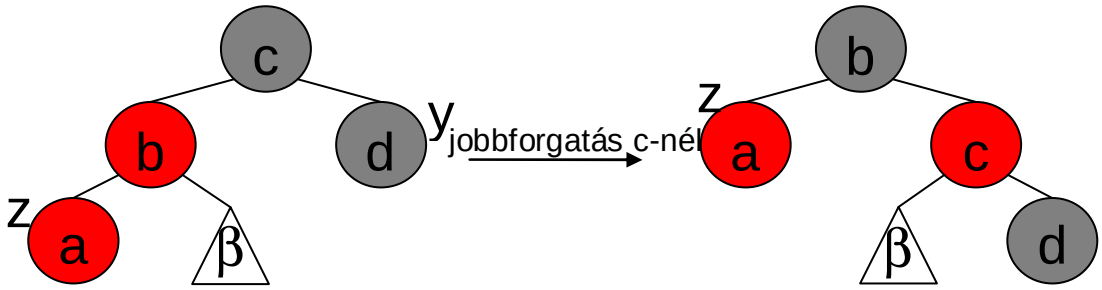
2. Ha y színe fekete, és z jobb gyerek, akkor z szülőjénél balra forgatunk (1.7. ábra), és z bal gyerekét tesszük meg új z-nek, így eljutunk a 3. esethez.



1.7 ábra. Beszúrás, 2. eset

3. Ha y színe fekete, és z bal gyerek, akkor az 1.8. ábrán látható forgatással és átszínezéssel biztosítjuk, hogy a piros-fekete fa tulajdonságai közül egyik se sérüljön. (z

szülőjét feketére, nagyszülőjét pirosra színezzük, majd z nagyszülőjénél jobbra forgatunk)



1.8 ábra. Beszúrás, 3. eset

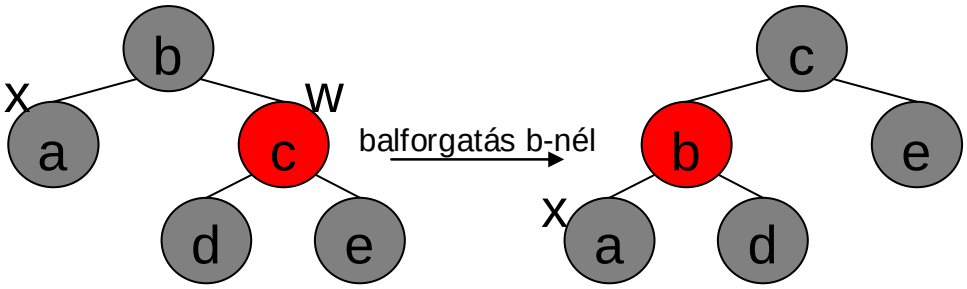
1.5.2 Törlés

Végrehajtjuk a bináris keresőfákra alkalmazott törlést. Jelöljük a kezdeti piros-fekete fa ténylegesen törölt elemét y-nal. Ha y piros volt, akkor a törléssel nem sérülhetnek a piros-fekete fa definiáló tulajdonságai, hiszen y nem lehetett a gyökérelem, nem jöhet létre piros szülő – piros gyerek kapcsolat és y törlése nem változtatja semelyik csúcs fekete-magasságát sem, tehát készen vagyunk. Ha y fekete volt, akkor az y helyére került elemet jelöljük x-el. Az x-el jelzett csúcsnak mindig legyen egy extra fekete értéke (ellensúlyozva a kiesett fekete csúcsot). Ezt az extra feketét fogjuk megfelelő módon fölfele vinni a fában, amíg

- x egy piros csúcsra nem mutat, ekkor színezzük azt feketére az extra fekete értéket felhasználva, vagy
- x már a piros-fekete fa gyökere, ekkor az extra fekete értéket egyszerűen elhagyjuk (csökkentve a fa fekete-magasságát).

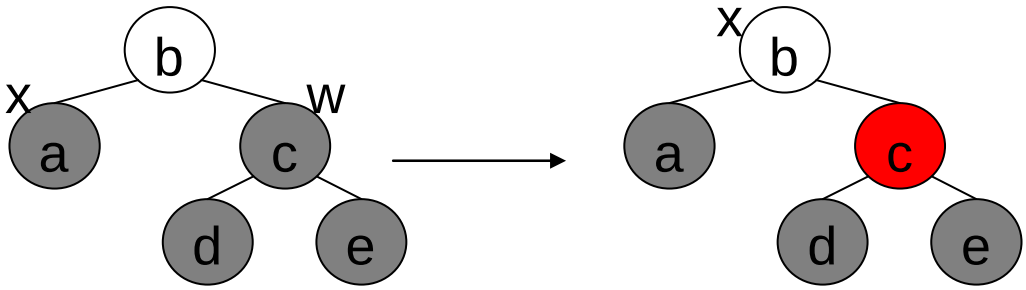
Abban az esetben, ha az x által mutatott csúcs fekete, de még nem a fa gyökere a következő négy eset állhat fenn. (feltételezzük, hogy x bal gyerek, ha jobb gyerek lenne, akkor a korábbiakhoz hasonlóan ”tükröznünk” kell) Jelöljük w-vel x testvérét. Vegyük észre, hogy x-nek létezik testvére, és az csak belső csúcs lehet, mivel w szülőjének feketemagassága legalább 2.

1. Ha x fekete, és w piros, akkor szülőjük, és w gyerekei csak feketék lehetnek. Színezzük pirosra w szülőjét, w-t pedig feketére, majd forgassunk balra w szülőjénél (1.9. ábra). Ezt követően a 2., 3. vagy 4. esethez jutunk.



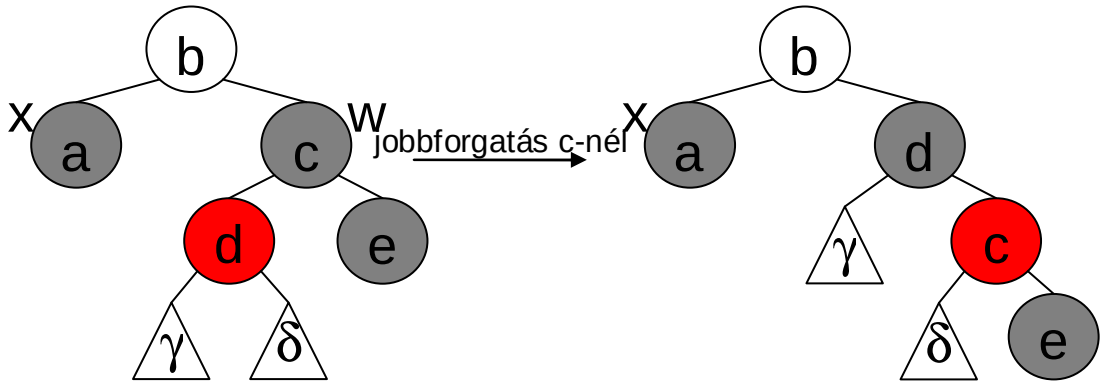
1.9 ábra. Törlés, 1. eset

2. Ha x, w, és w mindkét gyereke fekete, akkor w-t pirosra színezzük, az új x pedig legyen w szülője. Így a részfa fekete-magassága nem változik, a problémát azonban fentebb toltuk a fában (1.10. ábra). Vegyük észre, hogy w szülőjének színe nem ismert, de ha az 1. esettől kerültünk ide, akkor az új x piros, mely feketére színezésével készen is vagyunk.



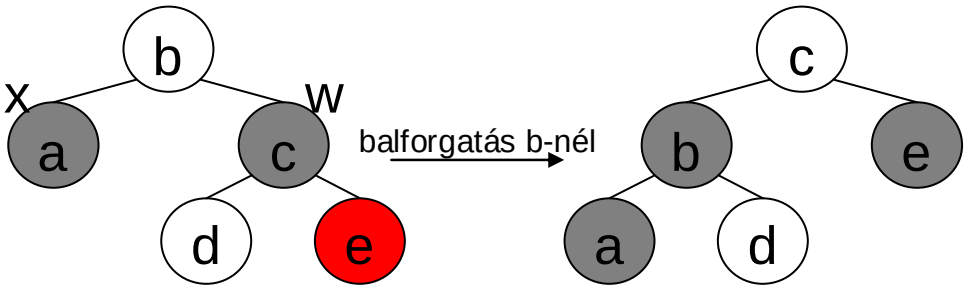
1.10 ábra. Törlés, 2. eset

3. Ha x és w fekete, w bal gyereke piros, jobb gyereke pedig fekete akkor az 1.11. ábrán látható forgatással és átszínezéssel a 4. esethez jutunk. (w bal gyerekét feketére, w-t pirosra színezzük, majd w-nél jobbra forgatunk)



1.11 ábra. Törlés, 3. eset

4. Ha x fekete, w fekete, és w jobb gyereke piros, akkor az 1.12. ábrán látható forgatással, átszínezéssel valamint az extra fekete érték elhagyásával biztosítjuk, hogy a piros-fekete fa tulajdonságai közül egyik se sérüljön. (w színét beállítjuk olyanra, mint amilyen a szülője színe, majd w jobb gyerekét és szülőjét feketére színezzük és forgatunk w szülőjénél balra)



1.12 ábra. Törlés, 4. eset

2 Felhasználói dokumentáció

2.1 Előkészületek

2.1.1 Rendszerkövetelmények

- Windows XP SP3 operációs rendszer
- .NET Framework 3.5
- Minimum 1024*768-as felbontás (1280*1024 ajánlott)
- Minimum 1 GHz processzor
- Minimum 512 MB memória

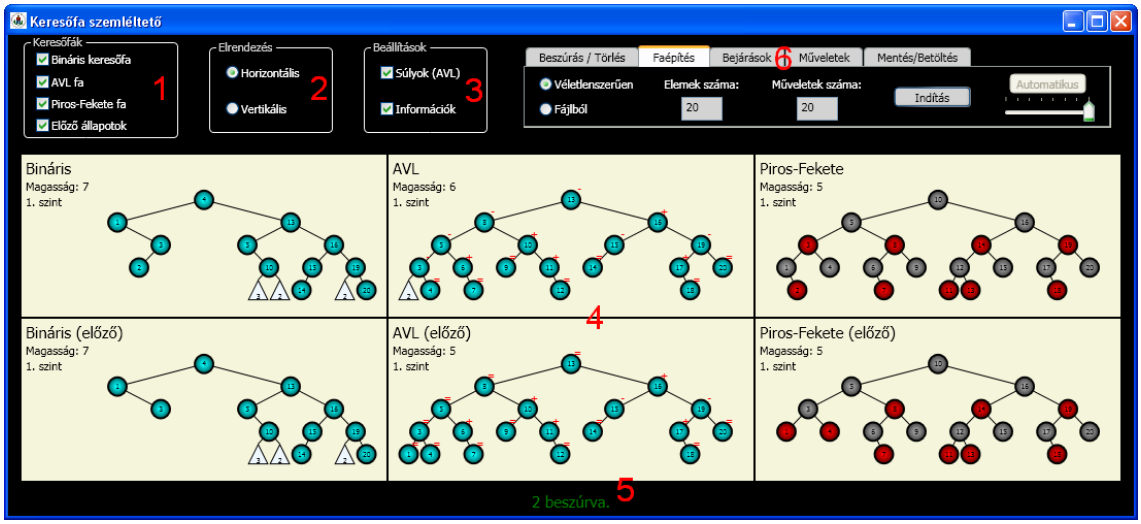
2.1.2 Telepítés

A szoftver telepítést nem igényel, egyszerűen másoljuk fel a cd-n található fájlokat a számítógépre.

2.1.3 A program indítása

A programot a keresofak.exe fájl futtatásával indíthatjuk.

2.2 A felhasználói felület



2.1. ábra A felhasználói felület részei

1. Keresőfák mező

Itt adhatjuk meg, hogy mely típusú keresőfákat szeretnénk látni a képernyőn, valamint azt, hogy az egyes fák előző állapotai is meg legyenek-e jelenítve. Ezeket billentyűkombinációkkal is módosíthatjuk: SHIFT+F1, SHIFT+F2, SHIFT+F3, SHIFT+F5.

2. Elrendezési mező

Horizontális és vertikális elrendezések közül választhatunk. Ennek csak akkor van jelentősége, ha egyszerre többféle fa megjelenítése is aktív.

3. Beállítások mező

Megadhatjuk, hogy az AVL fa esetén az alkalmazás feltüntesse-e az egyes csúcsok egyensúlyi állapotát. A fák konstrukciójával kapcsolatos információk kijelzését is itt állíthatjuk.

4. Demonstrációs terület

Itt helyezkednek el a megjelenített fák. Minden egyes fához tartozik egy információs rész, mely a keresőfa-megjelenítő bal felső részén helyezkedik el. Ez a rész tartalmazza az adott fa nevét, magasságát és azt, hogy a megjelenített részfa gyökéreleme a fa hányadik szintjén helyezkedik el.

5. Információs terület

A legutóbb elvégzett beszúrásról, törlésről vagy keresésről nyújt tájékoztatást, sikeres művelet esetén zöld, sikertelen esetén piros színnel. Faépítés során itt láthatjuk azt is, hogy hányadik lépésnél járunk.

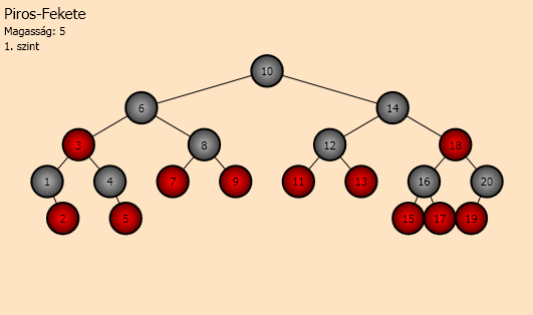
6. Menüsor

Kiválaszthatjuk, hogy program mely funkcióját szeretnénk használni.

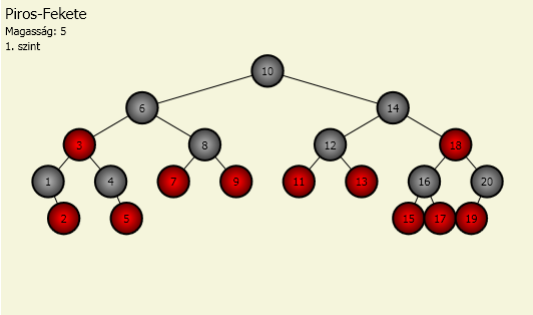
2.3 A keresőfa-megjelenítő

Összesen hat keresőfa-megjelenítő lehet a demonstrációs területen. Ezek közül legfeljebb egy lehet aktív állapotban, ami szükséges lehet bizonyos műveletek elvégzéséhez. Egy megjelenítőt kétféleképpen hozhatunk aktív állapotba:

- Bal vagy jobb egérgombbal kattintunk a megjelenítő területére.
- Funkcióbillentyűk segítségével (F1, F2, F3 és F5, F6, F7)



2.2. ábra Aktív állapot



2.3. ábra Normál állapot

Az ESC billentyű megnyomásával megszüntethetjük egy megjelenítő aktív állapotát. Abban az esetben, ha egy aktív megjelenítőt eltávolítunk (a háttérben) szintén normál állapotba kerül.

Minden megjelenítőhöz tartozik egy logikai keresőfa, aminek mi egy maximum 5 szintből álló részfáját látjuk. Ha az 5. szinten van olyan csúcs, melynek van gyereke, akkor a csúcs helyett egy részfaszimbólum jelenik meg, aminek a felirata nem a csúcs kulcsa, hanem a részfa szintjeinek a száma.

2.3.1 Navigálás egy keresőfában

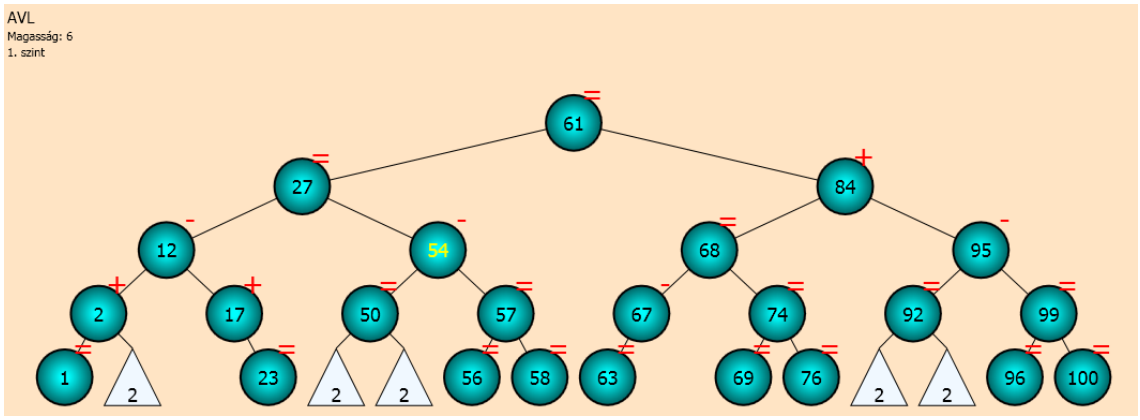
Mivel a megjelenítőkhöz tartozó logikai fa akár nagyon nagy is lehet, elengedhetetlen, hogy tudjunk navigálni benne, azaz meg tudjuk mondani, hogy a logikai fa mely csúcsától (mint gyökérelem) akarjuk látni a fát. A navigációra is több lehetőségünk van, de csak az aktív megjelenítőn:

- Kattintsunk bal egérgombbal egy csúcsra vagy egy részfaszimbólumra, ekkor a kiválasztott csúcs vagy részfaszimbólum gyökéreleme lesz a megjelenített részfa új gyökere.
- Az aktív megjelenítő felett mozgassuk az egér görgőjét előre vagy hátra. Ha előre görgetünk, akkor egyel fentebb lépünk a fában (az aktuális gyökérelem szülője lesz az új gyökér), ha hátra akkor lefele lépünk, de csak abban az esetben, ha az aktuális gyökérelemnek pontosan egy gyereke van.
- A kurzorbillentyűk segítségével. A felfele nyíllal eggyel fentebb lépünk a fában az aktuális elem szülőjére, a jobbra nyíllal a jobb gyerekre, a balra nyíllal a bal

gyerekre, a lefele pedig eggyel lentebb lépünk a fában abban az esetben ha az aktuális csúcsnak pontosan egy gyereke van.

2.3.2 Kijelölt csúcs

Az aktív megjelenítőn ki is jelölhetünk egy csúcsot, ha jobb egérgombbal kattintunk rá. Ekkor az elem kulcsa sárga színnel lesz kiírva (2.4. ábra). Kijelölt elemre a menüsor „Műveletek” pontjában lehet szükség.



2.4. ábra Az 54 kulcsú csúcs kijelölve egy AVL fában

2.4 Beszúrás/Törlés menüpont

2.5. ábra Beszúrás/Törlés menüpont

Ezen menüpontban van lehetőségünk egyesével történő beszúrásra és törlésre, valamint keresésre. A beszúrás és törlés műveletek mind a hat megjelenítőhöz tartozó logikai fára hatással lesznek, azokra is, melyeket esetleg nem is látunk. A program a keresést csak az aktív állapotban lévő megjelenítőhöz tartozó logikai fán hajtja végre.

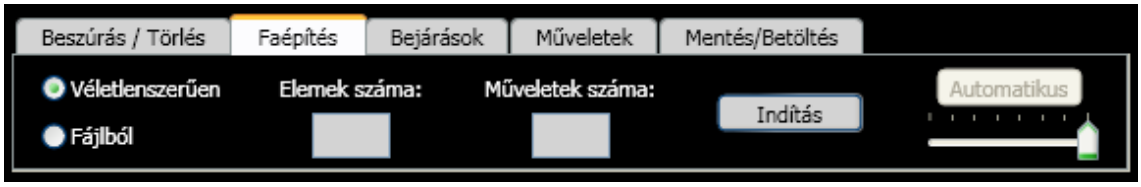
Egy legfeljebb háromjegyű pozitív egész számot szúrhatunk be vagy törölhetünk ki a fákból. Ezt az értéket a 2.5. ábrán látható szövegmezőbe kell beírnunk. Ha a beírt érték megfelel a kritériumoknak, akkor a művelet elvégzése után az információs területen megjelenik, hogy a művelet sikeres volt-e vagy sem (abban az esetben, ha a beállítások mezőben az információk jelölődoboz be van jelölve).

Ha beírtuk a megfelelő kulcsértéket, akkor beszúrást a „Beszúrás” gombra kattintással, vagy az ENTER billentyű lenyomásával hajthatjuk végre, törölni pedig a „Törlés” gombra kattintással, vagy a SHIFT+ENTER billentyűkombinációval tudunk.

2.5 Faépítés menüpont

A fákba egyesével történő beszúrásnál vagy törlésnél sokkal hatékonyabb, ha a faépítés menüpontot választjuk. Ily módon véletlenszerűen, vagy fájlban megadott műveletsorozattal építhetjük fel a keresőfákat.

2.5.1 Faépítés véletlenszerűen



2.6. ábra Faépítés véletlenszerűen

Véletlenszerű faépítés során megadhatjuk, hogy az elkészült fák hány csúcsból álljanak, valamint azt, hogy hány művelet eredményeként jöjjenek létre a fák. Nyilván legalább annyi műveletnek kell lennie, mint ahány elem van, és a két érték különbségének párosnak kell lennie (a hibás értékeket a program automatikusan korrigálja) . Ha a két szám megegyezik, az azt jelenti, hogy csak beszúrások lesznek. Az elemek száma és a műveletek száma is maximum 999 lehet.

Paraméterezés után az indítás gombbal kezdhetjük futtatni a demonstrációt. Ekkor választhatunk, hogy mi szeretnénk léptetni, vagy automatikusan a beállított időközönként hajtódjon végre egy művelet. A demonstráció során bármikor válthatunk automatikus és manuális módok között, illetve a műveletsor végrehajtásának sebességét is módosíthatjuk a csúszka segítségével. Ha a csúszka a minimális értéken áll, akkor 5 másodperc telik el két művelet végrehajtása között, míg a maximális értéken a műveletsor azonnal végrehajtódik.

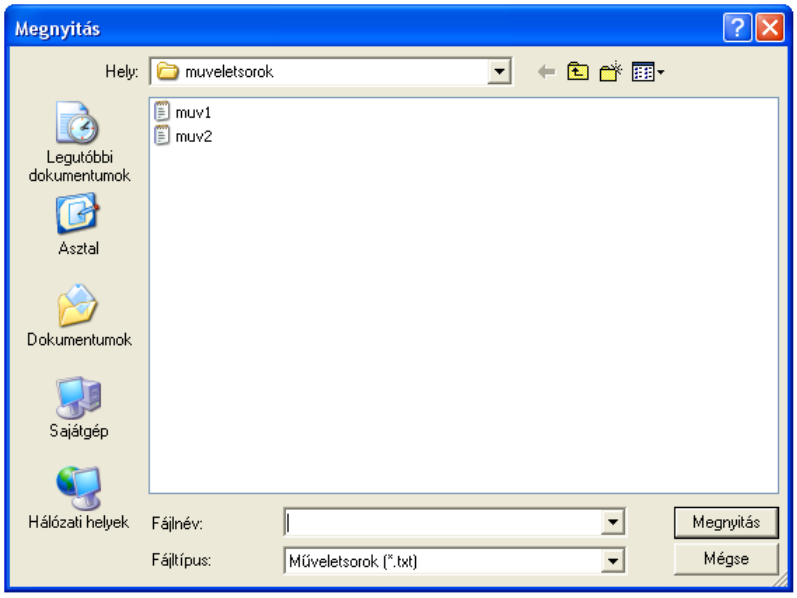
Az információs területen nyomon követhetjük, hogy a konstrukció hányadik lépésénél járunk (2.7. ábra).

2.7. ábra 36. lépés a 100-ból

Fontos megjegyeznünk, hogy ha a demonstráció alatt ellépünk a „Faépítés” fülről, vagy elváltunk a véletlenszerű építési módról, akkor a szemléltetés leáll.

2.5.2 Faépítés fájlból

2.8. ábra Faépítés fájlból



2.9. ábra Műveletsor megnyitása

Fájlból való faépítéshez először kattintsunk Betöltés gombra. A megjelenő dialógusablakban (2.9. ábra) válasszuk ki a megfelelő txt kiterjesztésű műveletsor fájlunkat. A fájlt bármilyen szövegszerkesztőben elkészíthetjük, de szerkezetére a következő megkötések vonatkoznak:

- Minden műveletnek külön sorban kell szerepelnie.
- Egy sor minimum 2 maximum 4 karakterből állhat.
- A sor első betűje „b” vagy „t” legyen.

- A sor első betűjét egy legfeljebb háromjegyű pozitív egész szám kövesse.

Ha egy sor „b” betűvel kezdődik, az azt jelenti, hogy be kell szúrni az utána álló számot, ha „t” betűvel akkor pedig ki kell törölni.

```
b10
b20
t3
b121
t20
```

2.10. ábra

A 2.10. ábrán látható fájl megfelel a formai követelményeknek. Ha lefuttatjuk, akkor a program először beszúrja a fákba a 10-et, majd a 20-at, utána törölni próbálja a 3-at, de ez nem fog sikerülni, hiszen nem szerepel a fákban 3 kulcsú elem, ezt követően beszúrja a 121-et, végül törli a 20-at.

A program azokat a sorokat, amelyek nem felelnek meg a fenti specifikációnak, figyelmen kívül hagyja.

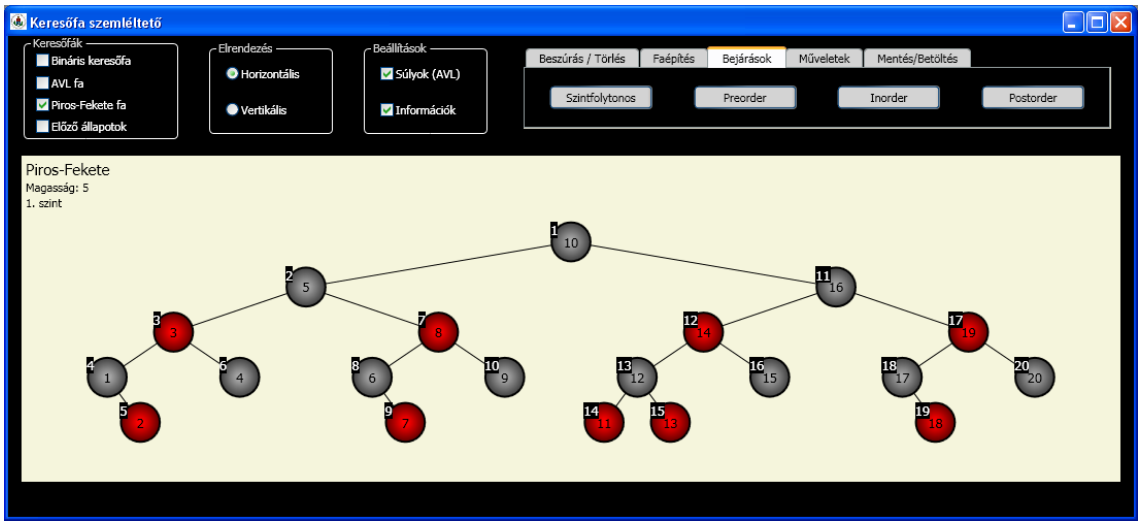
A demonstráció vezérlése a véletlenszerű esetben látottakkal megegyezően történik.

2.6 Bejárások menüpont

2.11. ábra Bejárások menüpont

Négy fabejárás közül választhatunk, a szintfolytonos, a preorder, az inorder és a postorder közül (2.11. ábra).

A megfelelő bejárás gombjára kattintva az összes fa összes megjelenített csúcsa mellett megjelenik egy szám. Ez a szám megmutatja, hogy a kiválasztott bejárás szerint az adott csúcs hányadikként kerül sorra.



2.12. ábra Egy piros-fekete fa preorder bejárása

A bejárásokat követően szabadon navigálhatunk a keresőfákban, de ha elváltunk a bejárások fülről a címkék eltűnnek.

2.7 Műveletek menüpont

2.13. ábra Műveletek menüpont

Az itt szereplő funkciók használatához szükséges, hogy legyen aktív állapotban lévő keresőfa-megjelenítőnk, az „Előző” és „Következő” műveletekhez pedig kijelölt csúcs is szükséges.

A maximum és minimum műveletek kijelölik az aktív megjelenítőhöz tartozó logikai fa legnagyobb, illetve legkisebb elemét, az előző és következő műveletek pedig az aktív megjelenítő kijelölt elemének a (inorder bejárás szerinti) rákövetkezőjére, illetve megelőzőjére lépteti a kijelölést (abban az esetben, ha ilyen elem nincs, a kijelöltség megszűnik).

2.8 Mentés/Betöltés menüpont

2.14. ábra Mentés/Betöltés menüpont

Lehetőségünk van elmenteni és betölteni a megalkotott fáinkat. A mentéshez kattintsunk a mentés gombra, és a megjelenő dialógusablakban adjuk meg a létrehozni kívánt fájl nevét és helyét.

Betöltéshez kattintsunk a betöltés gombra, majd a megjelenő dialógusablakban válasszuk ki a megfelelő bst kiterjesztésű fájlt.

Ha a mentés vagy betöltés során valamilyen hiba történne az alkalmazás egy felugró ablakkal értesít minket.

Az „Új” gombra kattintva új munkamenetet kezdhetünk.

3 Fejlesztői dokumentáció

A program WPF (Windows Presentation Foundation) alkalmazásként készült Visual Studio 2008-ban C# nyelven, .NET Framework 3.5 platformon.

Ezen fejezet célja, hogy elősegítse a programban való tájékozódást, a program továbbfejlesztését és karbantartását.

3.1 A probléma specifikációja és a program felépítése

A feladat egy olyan alkalmazás elkészítése, mely három típusú keresőfát, illetve azok műveleteit is tudja szemléltetni. Ezek az általános bináris keresőfa, az AVL fa, és a piros-fekete fa. A képernyőn egyszerre maximum 6 fát akarunk megjeleníteni, mert minden fának mutatni szeretnénk az előző állapotát is. Az összes konstrukciós műveletnek hatással kell lennie az összes fára. Mivel szemléltető programról van szó, az alapfunkciók elkészítésén kívül a megjelenítés és az elrendezés nagymértékű testre szabását is szeretnénk biztosítani.

A programban három réteget különíthetünk el, melyek a következők:

Logikai réteg

Itt helyezkednek el a különböző keresőfa adatszerkezetek. Az elméleti részben leírt algoritmusok is ebben a rétegben kerülnek megvalósításra. A logikai rétegben található osztályok semmilyen módon nem veszik igénybe a másik két réteg szolgáltatásait.

Megjelenítési réteg

Ez a réteg felel a logikai rétegben megvalósított keresőfák megjelenítésért. Csak a logikai réteg szolgáltatásait használja.

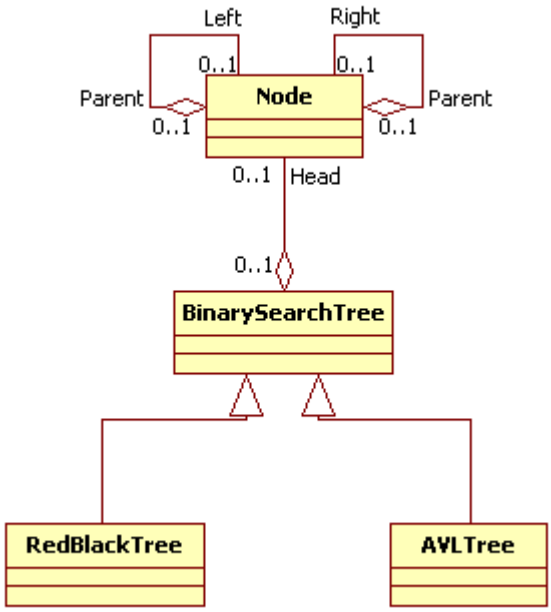
Vezérlési réteg

A felhasználóval történő kapcsolattartás, valamint a program kínálta funkciók megvalósítása a feladata. Használja mind a megjelenítési, mind a logikai réteg szolgáltatásait.

A következőkben a program különböző rétegeiben található osztályokat, azok fontosabb metódusait és adattagjait fogjuk megvizsgálni.

3.2 A logikai réteg

A logikai rétegben négy osztály szerepel, a Node, a BinarySearchTree, az AVLTree, és a RedBlackTree. A bináris keresőfák rendelkeznek egy a gyökércsúcsra mutató referenciával, és a bináris keresőfákból származtatjuk az AVL fákat, valamint a piros-fekete fákat (3.1 ábra). Az alkalmazott reprezentációban a csúcsoknak van referenciájuk szülőre.



3.1. ábra A logikai réteg osztálydiagramja

3.2.1 A Node osztály

Fontosabb adattagok

- private int **_key** : A csúcs kulcsértéke.
- private Node **_right** : Referencia a csúcs jobb gyerekére.
- private Node **_left** : Referencia a csúcs bal gyerekére.
- private Node **_parent** : Referencia a csúcs szülőjére.
- private sbyte **_balance** : A csúcs egyensúlyi állapota. Értéke „–” esetben -1, „=” esetben 0, és „+” esetben 1. (AVL fák esetén lesz szerepe)
- private byte **_color** : A csúcs színe. Értéke 0 ha piros, 1 ha fekete és 2, ha színtelen. (piros-fekete fák esetén lesz szerepe)

Az adattagokból property-k készültek a kényelmes használhatóság érdekében.

Metódusok

- public **Node**(int key) : A konstruktor. Inicializálja az adattagokat, valamint beállítja a kulcsértéket.
- public int **height**() : Visszaadja a csúcs által meghatározott részfa magasságát.
- public void **preOrder**(List<Node> traversalResult) : A csúcsok felsorolását készíti el preorder bejárás szerint. Az eddig elkészült listával paraméterezzük, és azt bővíti.
- public void **inOrder**(List<Node> traversalResult) : A csúcsok felsorolását készíti el inorder bejárás szerint.
- public void **postOrder**(List<Node> traversalResult) : A csúcsok felsorolását készíti el posztorder bejárás szerint.
- public void **levelOrder**(List<Node> traversalResult, Queue<Node> nodeQueue) : A csúcsok felsorolását készíti el szintfolytonos bejárás szerint. Használja az eddig elkészült listát, valamint az algoritmushoz szükséges sort.

3.2.2 A BinarySearchTree osztály

Adattagok

- private List<Node> **_traversalResult** : A bejárások megvalósítása során használt csúcslista.
- protected Node **_head** : Referencia a bináris fa gyökérelemére.

Metódusok

Az osztály tartalmaz négy virtuális metódust, amik majd a leszármazott osztályokban lesznek megvalósítva.

- public **BinarySearchTree**() : A konstruktor. Beállítja a _head értékét null-ra.
- public Node **search**(int value) : Megkeresi, és visszaadja value kulcsú csúcsot. Null-t ad vissza, ha nem létezik a keresett elem.
- public Node **insert**(int value) : Beszúr egy új csúcsot value kulcsértékkel. Az új csúccsal tér vissza, ha a beszúrás sikeres, null-al, ha nem. Az [1] 239. oldalán található algoritmus implementációja, kiegészítve két virtuális függvény meghívásával.

- public Node **delete**(int value) : Törli a fából a value kulcsú csúcsot. A törölt csúccsal tér vissza, ha a törlés sikeres, null-al ha nem. Az [1] 240. oldalán található algoritmus implementációja, kiegészítve két virtuális függvény meghívásával.
- public Node **maximum**(Node x) : Visszaadja az x gyökerű részfa maximális kulcsú csúcsát. Előfeltétel, hogy x nem lehet null.
- public Node **minimum**(Node x) : Visszaadja az x gyökerű részfa minimális kulcsú csúcsát. Előfeltétel, hogy x nem lehet null.
- public Node **successor**(Node x) : Visszaadja az x csúcs inorder bejárás szerinti rákövetkezőjét. Előfeltétel, hogy x nem lehet null.
- public Node **predecessor**(Node x) : Visszaadja az x csúcs inorder bejárás szerinti megelőzőjét. Előfeltétel, hogy x nem lehet null.
- public void **rightRotation**(Node x) : Az x csúcsnál jobbforgatást hajt végre. Előfeltétel, hogy x-nek legyen bal gyereke.
- public void **leftRotation**(Node x) : Az x csúcsnál balforgatást hajt végre. Előfeltétel, hogy x-nek legyen jobb gyereke.
- public List<Node> **preOrder**() : Visszaadja a fa csúcsait preorder sorrendben.
- public List<Node> **inOrder**() : Visszaadja a fa csúcsait inorder sorrendben.
- public List<Node> **postOrder**() : Visszaadja a fa csúcsait posztorder sorrendben.
- public List<Node> **levelOrder**() : Visszaadja a fa csúcsait szintfolytonos sorrendben.
- public List<bool> **pathTo**(int value) : Visszaadja az útvonalat, amin a value kulcsú elem elérhető a fában. A hamis balra, az igaz jobbra lépést reprezentál. Abban az esetben ha a keresett kulcsú csúcs nem szerepel a fában, a visszatérési érték null.

3.2.3 Az AVLTree osztály

Az AVLTree osztály a BinarySearchTree osztályból származik, és mindössze két felüldefiniált metódust tartalmaz.

Metódusok

- override protected void **CheckStateInsertAVL**(Node node) : Az 1.4.1 részben leírt algoritmus implementációja. A node mutatja, hogy melyik csúcsnál járunk a fában. Az 1.4.1. fejezetben felsorolt elromlott eseteken kívül, akkor sem kell tovább haladnunk a fában fölfele, ha nem mélyült a részfa.
- override protected void **CheckStateDeleteAVL**(Node node, bool right) : Az 1.4.2 részben leírt algoritmus implementációja. A node mutatja, hogy melyik csúcsnál járunk a fában, a right pedig azt, hogy a csúcs jobb részfájának csökkent-e a mélysége vagy a balnak. Abban az esetben, ha az aktuális részfa mélysége nem csökkent, nem kell tovább haladnunk a fán fölfelé.

3.2.4 A RedBlackTree osztály

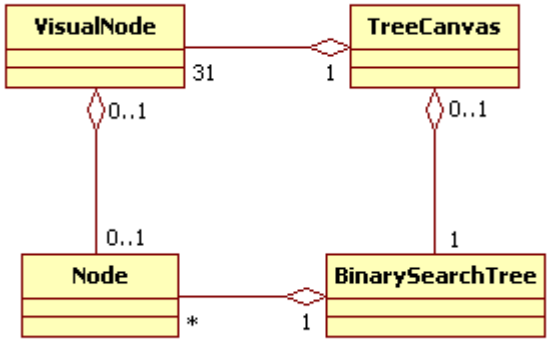
Az RedBlackTree osztály a BinarySearchTree osztályból származik. Mivel a piros-fekete fák esetében a null gyerekek számítanak levélelemnek, az algoritmusokban pedig szükség van az összes csúcs színének, illetve szülő referenciájának az ismeretére, nem mindenhol lehet direkt módon lekérdezni az adott csúcs színét vagy szülőjét.

Metódusok

- private bool **isBlack**(Node node) : Ha node null vagy színe fekete, akkor igazat ad vissza, egyébként hamisat.
- private bool **isRed**(Node node) : Ha node null vagy színe fekete, akkor hamisat ad vissza, egyébként igazat.
- override protected void **CheckStateInsertRedBlack**(Node z) : Piros-fekete fába történő beszúrás utáni helyreállítást végzi. Az [1] 255. oldalán található algoritmus implementációja.
- override protected void **CheckStateDeleteRedBlack**(Node x, Node parent) : Piros-fekete fából való törlés utáni helyreállítást végzi. Az [1] 262. oldalán található algoritmus implementációja. A parent referencia mindig az aktuális x szülőjét azonosítja. Erre azért van szükség, mert x lehet null is (levélelem), és ekkor nem lehetne eldönteni, hogy melyik csúcs gyereke.

3.3 A megjelenítési réteg

A megjelenítési réteg két osztályt tartalmaz, a VisualNode-ot és a TreeCanvas-t. Minden TreeCanvas objektumhoz tartozik egy BinarySearchTree példány (ennek a megjelenítését végzi), valamint 31 db VisualNode objektum (egyszerre maximum ennyi csúcsot jelenít meg). A VisualNode objektumokhoz legfeljebb egy Node objektum tartozhat, ami a megjelenített fának egy csúcsa. A 3.2. ábrán látható osztálydiagram szemlélteti a megjelenítési és a logikai réteg között fennálló kapcsolatot.



3.2. ábra Kapcsolat a megjelenítési és a logikai réteg között

3.3.1 A VisualNode osztály

A VisualNode osztály a beépített Button osztályból származik. Egy WPF nyújtotta lehetőséget kihasználva egy VisualNode objektum négy különböző típusú logikai csúcs megjelenítésére is szolgál. A WPF App.xaml fájljában négyféle button controltemplate (vezérlősablon) található. A „VisualNode” nevű a színtelen csúcs sablonja, a „VisualNodeRed” és „VisualNodeBlack” nevű a piros-fekete fák piros és fekete csúcsainak sablonja, a „VisualNodeTriangle” sablon pedig a részfaszimbólumokhoz tartozik.

Fontosabb adattagok

- private Node **_node** : Referencia a VisualNode-hoz tartozó logikai node-ra.
- private Label **_TraversalLabel** : Bejárások szemléltetésekor ezen a labelen jelenik meg az, hogy VisualNode-hoz tartozó logikai node a bejárás során hányadikként lett érintve.
- private Label **_Ballance** : AVL fához tartozó logikai csúcs esetén ezen a labelen jelenik meg a csúcs egyensúlyi állapota (-,=,+).

Fontosabb metódusok

- **VisualNode_Click** : Eseménykezelő, ami egy VisualNode-ra történő kattintáskor fut le. Aktív állapotba hozza a VisualNode-hoz tartozó TreeCanvast, és a hozzá tartozó logikai node-ra pozícionálja azt.
- **VisualNode_MouseDown** : Eseménykezelő, ami jobb egérkattintás esetén kijelöli a VisualNode-hoz tartozó logikai node-ot, valamint aktív állapotba hozza a TreeCanvast.

3.3.2 A TreeCanvas osztály

A TreeCanvas osztály a beépített Canvas osztályból származik, és ez felel a logikai réteg keresőfáinak megjelenítéséért. A logikai fa egy tetszőleges csúcsától legfeljebb öt szintet mutat (azaz maximum 31 csúcsot). Lehetőség van módosítani ezt a tetszőleges csúcsot, így lehet navigálni a fában.

Fontosabb adattagok

- public static TreeCanvas **FocusedCanvas** : Egy osztályszintű adattag. Referencia az aktív állapotban lévő TreeCanvasra. Ha az összes TreeCanvas normál állapotban van, akkor az értéke null.
- private Label **_nameLabel** : A megjelenített fa neve.
- private Label **_positionLabel** : Hányadik szinttől jelenítjük meg a fát.
- private Label **_heightLabel** : A fa magassága.
- private BinarySearchTree **_tree** : Referencia a hozzá tartozó logikai fára.
- private List<VisualNode> **_VisualNodeList** : A VisualNode-ok listája.
- private List<Line> **_EdgeList** : Kirajzolandó élek listája.
- private List<bool> **_basePosition** : Egy utat kódol, ami meghatározza azt a csúcsot a logikai fában, ahonnan meg kell jelenítenünk a fát. Az igaz érték jobbra, a hamis érték balra lépést jelent.
- private bool **_showBallance** : Egyensúlyi állapotok kijelzése. (AVL fánál) Property-n át történő módosítás során automatikusan újrarajzolja a fát.
- private bool **_showTraversalLabels** : Bejárási címkék megjelenítése. Property-n át történő módosítás során automatikusan újrarajzolja a fát.
- private bool **_shown** : Látszódik-e az adott TreeCanvas a képernyőn.

- private Node **_selectedNode** : Referencia a kijelölt logikai csúcsra. Értéke null, ha nincs kijelölt cúscs.

Fontosabb metódusok

- public **TreeCanvas**(BinarySearchTree tree, string text) : A konstruktor. Inicializálja az adattagokat, beállítja a logikai fát (tree) illetve a fa címkéjét (text).
- public void **setFocus**() : Aktív állapotba hozza a TreeCanvast.
- public void **setTree**(BinarySearchTree tree, string text) : Beállítja a logikai fát, és a fa címkéjét.
- public void **refresh**() : Frissíti a logikai node-ok VisualNode-okhoz rendelését a baseposition alapján, majd újrarajzolja a fát.
- public void **redraw**() : Újrarajzolja a fát. Meghatározza az összes VisualNode helyét és méretét, eldönti, hogy melyek látszódnak (tartozik-e hozzájuk logikai node), illetve meghatározza, hogy milyen vezérlősablont alkalmazzon rájuk. Ha a VisualNode az utolsó sorban van, és a hozzá tartozó logikai node-nak van gyereke, akkor részfaszimbólum sablont rendel hozzá, egyébként értelemszerűen a logikai node színe dönti el, hogy melyik sablont rendeli hozzá. A létező éleket is kirajzolja a megfelelő koordinátákra, valamint szükség esetén az egyensúlyi és a bejárási címkéket is megjeleníti.
- public void **moveUp**() : Fentebb lépteti a megjelenített részfát.
- public void **moveRight**() : Jobbra lépteti a megjelenített részfát.
- public void **moveLeft**() : Balra lépteti a megjelenített részfát.
- public void **moveDown**() : Lentebb lépteti a megjelenített részfát, ha az aktuális csúcsnak pontosan egy gyereke van.
- public void **moveTo**(List<bool> path) : A megjelenített részfát a logikai fa path által meghatározott csúcsára lépteti.
- public void **showPreOrder**() : Bejárja a logikai fát preorder módon, majd megjeleníti a bejárási címkéket.
- private void **posToSelected**() : Úgy pozícionál a fában, hogy a kijölt csúcs (ha van ilyen) látható legyen.
- public void **showMaximum**() : Kijelöli a fa maximális kulcsú elemét, majd rápozícionál.

- public void **showSuccessor**() : Kijelöli a logikai fa kijelölt elemének az inorder bejárás szerinti rákövetkezőjét, majd rápozícionál.
- public void **showNode**(int key) : Kijelöli a key kulcsú csúcsot (ha van ilyen), majd rápozícionál.
- **TreeCanvas_MouseWheel** : Eseménykezelő, ami az egérgörgő görgertésekor fut le. Ha ez az aktív TreeCanvas, akkor előregörgetés esetén felfele, hátragörgetés esetén lefele próbál meg lépni a fában.
- **TreeCanvas_MouseDown** : Eseménykezelő, ami valamelyik egérgomb lenyomásakor fut le. Aktívvá teszi a TreeCanvast.
- **TreeCanvas_SizeChanged** : Eseménykezelő, ami akkor fut le, ha megváltozik a TreeCanvas mérete. Ekkor újrarajzolja a fát.

3.4 A vezérlési réteg

A vezérlési réteg bonyolítja a felhasználóval való kapcsolattartást, valamint használva a másik két réteg szolgáltatásait, ez a réteg valósítja meg program kínálta funkciókat.

3.4.1 A felhasználói felület kialakítása

A felhasználói felület kialakítása nagyrészt a Window1.xaml fájlban történik. Az xml struktúráltsága miatt a kód könnyen átlátható. Az ablak felső részén kapnak helyet a különböző vezérlőelemek, bal oldalon három groupbox, melyek a megjelenítés testre szabásáért felelősek, jobb oldalon pedig egy tabcontrol, amivel a program különböző funkciói elérhetők (lásd 2.4-2.8). Az ablak alsó részén egy 3*3-as grid (TreeGrid) található, melynek hat cellájába kerül a hat TreeCanvas (melyek a hat különböző keresőfát jelenítik meg). Azt, hogy a grid kilenc cellája közül melyik hat foglalt, illetve melyik oszlopok szélessége (sorok magassága) nulla, a „Keresőfák”, illetve az „Elrendezés” címkéjű groupboxokban található controlok aktuális állapota határozza meg.

A tabcontrol lapjainak felépítése nagyon egyszerű, egyedül a „Faépítés” fülhöz tartozó lap szorulhat magyarázatra. Ezen a lapon két alfunkció („Véletlenszerűen”, illetve „Fájlból”) is elérhető, és közöttük két rádiógombbal lehet váltani. Bizonyos vezérlők láthatósága a rádiógombok IsChecked attribútumához van kötve. Hasonlóan,

az ablak alján található információs címke (lblInfo) láthatósága a CbxInfo IsChecked adattagjához van kötve.

3.4.2 A Window1 osztály

Fontosabb adattagok

- private int **_lastValue** : Az előző (sikeres) művelete operandusa.
- private byte **_lastOperation** : Az előző (sikeres) művelet típusát mutatja. Értéke 1, ha beszúrás volt, 2, ha törlés volt.
- private Timer **_timer** : Az automatikus faépítéshez szükséges időzítő.
- private List<string> **_opFileRows** : Fáljból történő faépítés során a fájl specifikációnak megfelelő sorait tárolja.
- private int **_currentFromFileRow** : Fáljból történő faépítés során az _opFileRows hányadik eleménél járunk.
- private int **_nodeCount** : Véletlenszerű faépítéshez az elemek száma.
- private int **_opCount** : Véletlenszerű faépítéshez az műveletek száma.
- private List<int> **_keysNotInTree** : Véletlenszerű faépítés során azokat a lehetséges kulcsértékeket tárolja, amelyek éppen nem szerepelnek a fában. Ezen lehetséges kulcsértékek az 1, és az „Elemek száma”-ként megadott paraméter (maximum 999) közé eső számok.
- private List<int> **_keysInTree** : Véletlenszerű faépítés során azokat a kulcsértékeket tárolja, amelyek éppen szerepelnek a fában.
- private int **_deletesLeft** : A hátralévő törlések száma véletlenszerű faépítés esetén.
- private int **_insertsLeft** : A hátralévő beszúrások száma véletlenszerű faépítés esetén.

Ezeken kívül tartalmaz még az osztály hat TreeCanvast, illetve mindhárom keresőfa típusból kettőt.

Fontosabb metódusok

- public **Window1**() : A konstruktor. Egymáshoz rendeli a TreeCanvasokat és a logikai fákat, valamint elhelyezi őket a TreeGridben.

- private bool **insert**(int value) : Megpróbálja beszúrni a value kulcsú új csúcsot az összes fába. Igazzal tér vissza, ha sikerült, hamissal, ha nem.
- **BtnInsert_Click** : Eseménykezelő, ami a Beszúrás/Törlés tabon elhelyezkedő BtnInsert gomb megnyomásakar fut le. Megpróbálja számmá konvertálni TbxValue tartalmát. Ha sikerül, meghívja az insert metódust.
- private bool **delete**(int value) : Megpróbálja kitörölni a value kulcsú csúcsot az összes fából. Igazzal tér vissza, ha sikerült, hamissal, ha nem.
- **BtnDelete_Click** : Eseménykezelő, ami a Beszúrás/Törlés tabon elhelyezkedő BtnDelete gomb megnyomásakar fut le. Megpróbálja számmá konvertálni tbxValue tartalmát. Ha sikerül, meghívja a delete metódust.
- **BtnSearch_Click** : Eseménykezelő, ami a Beszúrás/Törlés tabon elhelyezkedő BtnSearch gomb megnyomásakar fut le. Megpróbálja számmá konvertálni tbxValue tartalmát. Ha sikerül, és van aktív TreeCanvas akkor megkeresi, és kijelöli a kért kulcsú elemet az aktív megjelenítőn.
- private void **refreshPrevCanvases**(int newValue, byte operation) : Frissíti az előző állapotokat kijelző TreeCanvasokat. Paramétere az új operandus, illetve az új művelet kódja.
- private void **refresh**() : Az összes TreeCanvasnak meghívja a refresh metódusát.
- private void **DoRearrange**() : Újrarendezi a a TreeCanvasokat a TreeGridben az aktuális beállításoknak megfelelően.
- **btnPreOrder_Click, btnInOrder_Click, btnPostOrder_Click, btnLevelOrder_Click** : Eseménykezelők, amik az összes TreeCanvason megmutatják a megfelelő bejárást.
- **TbxValue_KeyUp** : Eseménykezelő, ami az egyenkénti beszúrás és törlés hotkey-eit valósítja meg (ENTER, SHIFT+ENTER).
- **Window_KeyDown** : Eseménykezelő, ami az összes többi gyorsbillentyűt valósítja meg.
- **tbcChoices_SelectionChanged** : Eseménykezelő, ami a tabcontrolon történő váltáskor fut le. Megszünteti a bejárások kijelzését, leállítja a faépítést, és törli az információs label tartalmát.
- **btnSave_Click** : Eseménykezelő, ami a fák elmentését valósítja meg. Kivétel esetén egy felugró ablakot jelenít meg.

- **btnLoad_Click** : Eseménykezelő, ami a fák betöltését valósítja meg. Kivétel esetén egy felugró ablakot jelenít meg.
- **btnNew_Click** : Eseménykezelő, ami új TreeCanvasokat és logikai fákat hoz létre. (új munkamenetet indít)
- **btnMinimum_Click, btnMaximum_Click, btnPrev_Click, btnNext_Click** : Eseménykezelők, amik kijelölik, és megmutatják az aktív TreeCanvas megfelelő elemét (lásd 2.7).
- private void **randomStep**() : Véletlenszerűen történő faépítés során egy lépést hajt végre.
- private void **fromFileStep**() : Fájlból történő faépítés során egy lépést hajt végre.
- **BtnStartStep_Click** : Eseménykezelő. Attól függően, hogy a BtnStartStep gomb felirata „Indítás” vagy „Léptetés”, elindítja a kiválasztott faépítési folyamatot, vagy lépteti azt. Véletlenszerű faépítés esetén indításkor beállítja a _insertsleft és _deletesleft változók értékeit a paramétereknek megfelelően, valamint feltölti _keysNotInTree listát.
- private void **DoBtnAuto_Click**() : Attól függően, hogy a BtnAuto gomb felirata „Automatikus” vagy „Manuális”, vált az automatikus és léptetéses faépítési módok között.
- **BtnFromFileOpen_Click** : Eseménykezelő, ami bekér egy műveletsort tartalmazó txt fájlt, majd feltölti az _opFileRows listát a fájl specifikációnak megfelelő soraival. Kivétel esetén egy felugró ablakot jelenít meg.
- private void **treeConstructReset**() : Leállítja a faépítési folyamatot, és beállítja az alapértelmezett értékeket a megfelelő elemeknek.
- **SlrTimer_ValueChanged** : Eseménykezelő, ami a demonstráció sebességét szabályozó csúszka értékének változásakor fut le. Beállítja az időzítő interval adattagját az értéknek megfelelően. 0 érték és automatikus faépítés esetén az összes hátralévő konstrukciós lépést végrehajtja.
- **timer_Elapsed** : Eseménykezelő, ami az időzítő jelzésekor fut le. Végrehajt egy, az aktuális faépítési módnak megfelelő lépést.
- **RdbFromFile_Checked** : Eseménykezelő, ami átvált fájlból történő faépítési módra.

- **RdbRandom_Checked** : Eseménykezelő, ami átvált véletlenszerű faépítési módra.

3.5 Tesztelés

A tesztelési folyamatot két részre bonthatjuk. Először a keresőfák algoritmusainak helyes működését ellenőrizzük, azaz azt, hogy az 1. fejezetben leírt esetekben a műveletek után a helyes végeredmény alakul-e ki. Ezután a felhasználói felületet és a programfunkciókat teszteljük potenciálisan problémásnak ítélt eseménysorozatok előidézésével.

Az egyes tesztesetekben leírjuk, hogy pontosan mit, és hogyan tesztelünk. Amennyiben az eredmény nem az elvárt működésnek megfelelő lenne, ki kell javítani a hibát, és megvizsgálni, hogy az addig letesztelt esetekben továbbra is helyesen működik-e a program. (A program végleges verziója minden tesztesetben az elvárásoknak megfelelően működött.)

A tesztelés a következő konfiguráción történt:

- Windows XP SP3, .NET Framework 3.5 SP1
- 1920*1200-as felbontás
- Core 2 Duo E6420 processzor
- 2 GB memória

3.5.1 A logikai réteg szolgáltatásainak tesztelése

Ezen tesztekben általában megadunk egy műveletsort (a 2.5.2 fejezetben leírt specifikáció szerint), melynek az utolsó művelete idézi elő a tesztelni kívánt eseményt. Az itt felsorolásra kerülő tesztek példák néhány fontosabb esetre. (Az összes esetben az elvárt fát kapjuk eredményül.)

Bináris keresőfa

1. Első elem beszúrása. (B5)

2. Sokadik elem beszúrása. (B5, B1, B2, B8)

3. Egy levélelem törlése. (B5, B1, B2, B8, T2)

4. Egy olyan csúcs törlése, aminek egy jobb gyereke van. (B5, B1, B2, B8, T1)

5. Egy olyan csúcs törlése, aminek egy bal gyereke van. (B5, B2, B1, B8, T2)

6. Egy olyan csúcs törlése, aminek két gyereke van. (B5, B2, B3, B1, T2)

7. A gyökérelem törlése. (B5, B1, B2, B8, T5)

AVL fa

8. Beszúrás (++,+) eset. (B1, B2, B3)

9. Beszúrás (--.-) eset. (B3, B2, B1)

10. Beszúrás (++,-) eset. (B1, B3, B2)

11. Beszúrás (--,+) eset. (B3, B1, B2)

12. Törlés (++,+) eset. (B5, B4, B6, B7, T4)

13. Törlés (--,-) eset. (B5, B4, B6, B3, T6)

14. Törlés (++,=) eset. (B5, B4, B7, B8, B6, T4)

15. Törlés (--,=) eset. (B5, B3, B6, B4, B2, T6)

16. Törlés (++,-) eset. (B4, B7, B3, B6, T3)

17. Törlés (--,+) eset. (B4, B2, B5, B3, T5)

18. Azok az esetek, amikor a fenti AVL fák jobb, illetve bal részfaként jelennek meg.

Piros-fekete fa

19. Beszúrás 1. eset (B5, B7, B3, B2)

20. Beszúrás 1b. eset (B5, B7, B3, B8)

21. Beszúrás 2. eset (B5, B3, B4)

22. Beszúrás 2b. eset (B5, B7, B6)

23. Beszúrás 3. eset (B5, B4, B3)

24. Beszúrás 3b. eset (B3, B4, B5)

25. Törlés 1. eset (B5, B3, B7, B6, B8, B9, T9, T3)

26. Törlés 1b. eset (B6, B4, B7, B3, B5, B2, T2, T7)

27. Törlés 2. eset (B5, B3, B7, B8, T8, T3)

28. Törlés 2b. eset (B5, B3, B7, B2, T2, T7)

29. Törlés 3. eset (B5, B3, B8, B7, T3)

30. Törlés 3b. eset (B5, B3, B8, B4, T8)

31. Törlés 4. eset (B5, B3, B7, B8, T3)

32. Törlés 4b. eset (B5, B3, B7, B2, T7)

33. Azok az esetek, amikor a fenti piros-fekete fák jobb, illetve bal részfaként jelennek meg.

3.5.2 A felhasználói felület és a programfunkciók tesztelése

34. Próbáljuk ki az összes lehetséges állapotát a keresőfák, illetve az elrendezés groupboxokban található vezérlőknek.

35. Próbáljunk a specifikációnak nem megfelelő értéket beszúrni a fákba. (például 1-999 intervallumon kívül eső számot; nem szám karaktert tartalmazó karaktersorozatot)

36. Navigáljunk az egyik megjelenítőn csúcsra kattintással, egérgörgővel, kurzorbillentyűkkel. Próbáljunk ”túlmenni” a fán (lelépni egy levélről, vagy a fa gyökeréből lépni felfele).

37. Töröljünk egy olyan elemet a fákból, amelyik az egyik megjelenítőn a részfa gyökere, és nincsen gyereke. (ekkor a megjelenítés a szülőre lép, ha tud)

38. Próbáljunk hibás értékeket megadni a véletlenszerű faépítés paramétereinek, majd indítsuk el a demonstrációt. (nem számot; negatív számot; kevesebb műveletet, mint elemet; műveletek és elemek különbsége legyen páratlan)

39. Fájlból történő faépítésnél töltsünk be egy olyan fájlt: aminek egyetlen sora sem felel meg a specifikációnak; ami üres; aminek néhány sora megfelel; minden sora megfelel.

40. Indítsunk úgy automatikus faépítést, hogy: a sebesség maximumon van; a sebességet menet közben állítgatjuk; a sebességet menet közben maximumra állítjuk.

41. Váltogassunk automatikus és manuális faépítés között.

42. Faépítés közben lépjünk el a faépítés fülről, majd térjünk vissza. (a faépítés leáll ellépéskor)

43. Faépítés közben váltsunk a faépítési módok között. (váltáskor a faépítés leáll)

44. Faépítés közben módosítsunk a megjelenítők elrendezésén, láthatóságukon.

45. Faépítés közben változtassuk az ablak méretét.

46. Faépítés közben navigáljunk valamelyik megjelenítőn.

47. Egy faépítés után hajtsunk végre egyesével történő beszúrásokat és törléseket.

48. Próbáljuk ki a „Műveletek” fül funkcióit, illetve a keresést úgy, hogy nincs aktív megjelenítő.

49. Próbáljuk ki az előző, és következő műveleteket úgy, hogy van aktív megjelenítő, de nincs kijelölt elem.

50. Navigáljunk egy megjelenítőn, miközben a bejárási címkék látszódnak.

51. Navigáljunk egy AVL fa megjelenítőn, miközben a súlyok látszódnak.

52. Mentsük el munkánkat, majd töltsük vissza.

53. Próbáljunk betölteni egy hibás bst fájlt.

3.6 Továbbfejlesztési lehetőségek

A program egyik lehetséges továbbfejlesztési módja, hogy a faépítés funkció használata közben legyen lehetőség visszafele is léptetni, valamint egy ”finom léptetés” opció hozzáadása, amellyel az egyes műveletek végrehajtása során kialakult részállapotok is megtekinthetők lennének.

Demonstrációs program lévén egy másik kézenfekvő lehetőség az, hogy a szoftvert animációkkal tegyük látványosabbá.

Összegzés

Úgy gondolom, a szakdolgozatomként elkészített program - az elméleti résszel kiegészítve - egy rendkívül jól használható oktatási segédanyag. Az alkalmazás elkészítése során fontosnak tartottam, hogy kezelése bárki számára nagyon könnyen elsajátítható legyen.

Az alkalmazás egyedülálló tulajdonsága, hogy képes a tárgyalt keresőfákat akár egyszerre, párhuzamosan szemléltetni, így a különböző adatszerkezetek eltérő viselkedése sokkal jobban megfigyelhető. A fák előző állapotainak kijelzésének lehetősége szintén nagyon hasznos lehet.

A megjelenítési mód teljes mértékben testre szabható, ami egy szemléltető program esetén különösen nagy jelentőséggel bír.

Irodalomjegyzék

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein : Új algoritmusok, Scolar, 2003, [992], ISBN: 9639193909

[2] Rónyai Lajos, Ivanyos Gábor, Szabó Réka : Algoritmusok, Typotex, 2005, [349], ISBN: 9639132164

[3] http://people.inf.elte.hu/fekete/ (2009. május)

[4] http://en.wikipedia.org/wiki/AVL_tree (2009. május)

[5] C. Bennage, R. Eisenberg : Tanuljuk meg a WPF használatát 24 óra alatt, Kiskapu, 2009, [463], ISBN: 9639637566

[6] http://people.inf.elte.hu/gt/eaf/eaf3/eaf3.html (2009. május)