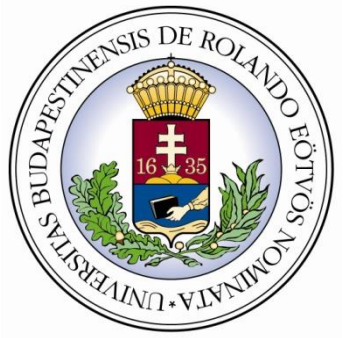


Eötvös Loránd Tudományegyetem
Informatikai Kar
Algoritmusok és Alkalmazásaik Tanszék

AVL fák műveleteinek szemléltetése

Témavezető:
dr. Ásványi Tibor
egyetemi docens

Készítette:
Képes Gyula
programozó matematikus, nappali

Budapest, 2012

Tartalomjegyzék

1. Bevezetés

Az algoritmusok és adatszerkezetek oktatásában nagy szerepet kapnak a különböző adatszerkezetek, ezeken belül is keresőfák. Ezeknek egy speciális változatát szeretném bemutatni, az AVL fákat. Szakdolgozatom témája egy olyan program elkészítése, amely AVL fákat tud ábrázolni és építeni, segítve ezzel a tanulást és megértést.

Dolgozatom része egy elméleti összefoglalás, amiben röviden ismertetem a program használatához szükséges fogalmakat és algoritmust.
A program képes az egyes műveleteket lépésenként bemutatni.

2. Elméleti összefoglalás

2.1. Keresőfák

A keresőfák olyan adatszerkezetek, melyek adataink hatékony tárolását segítik elő. Az alapvető műveleteik hatékonyak és gyorsak. Felsorolva a keresés, beszúrás és törlés műveletek a legfontosabbak nagy adatmennyiségek esetén. Adatainkhoz ekkor egy vagy több kulcsot rendelünk. A keresőfák algoritmusai szempontjából ezek kulcsok a lényegesek. A szemléltető programban úgy tekintünk adatainkra, mintha azok csak kulcs mezőből állnának. A keresőfáknak a fajtái közül először a bináris fákat jellemzem, majd az AVL fákat mutatom be.

A bináris fák rendelkeznek egy kitüntetett csúccsal, amit a fa gyökerének nevezünk. Minden csúcsból legfeljebb két él indulhat ki, melyek eggyel mélyebb szinten elhelyezkedő csúcsokhoz csatlakoznak. Ezeket a csúcsokat gyerek csúcsoknak nevezzük, a bal oldali a bal gyerek valamint a jobb oldali a jobb gyerek.
A csúcsot amelyből az élek kiindulnak, a gyerek csúcsok szülőjének nevezzük.
Egy csúcs levél, ha egyetlen gyereke sincs. Egy bináris fa magasságán a szintjeinek a számát mínusz egyet értünk. A bináris fákat általában pointeresen ábrázoljuk.

2.2 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.

A definícióból következik, hogy 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.

2.2.1. 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 szintjeinek számával megegyező számú lépésben véget ér, azaz a keresés műveletigénye O(h) ahol „h” a fa szintjeinek számát jelöli

2.2.2. Beszúrás

Beszúrás során először egy keresést hajtunk végre a beszúrandó elemre.
Ha megtaláltuk, akkor nem csinálunk semmit, mert kulcs duplikátumokat nem engedünk meg. Tegyük fel nem találtuk a keresett elemet, akkor beszúrjuk az elemet oda, ahol a keresés leállt. Vagyis 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).

2.2.3. 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, vagyis 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.
Végigmegyünk a törlendő csúcs jobb részfájának bal gyerekein lefelé haladva, így megtaláljuk a részfa minimális kulcsú csúcsát. Ennek a 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).

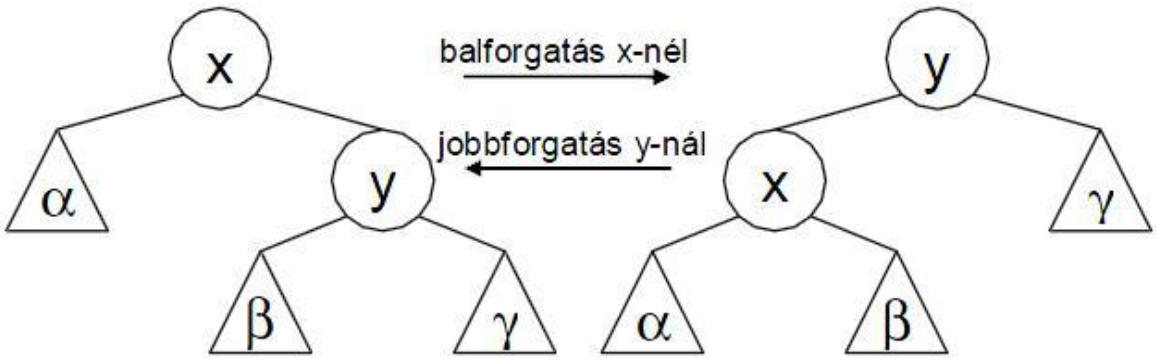
2.2.4. **Műveletigény**

A bináris keresőfa alapműveleteinek végrehajtási ideje a fa magasságától függ. Egy teljes fa esetén ez O(log n), eltorzult fáknál O(n), a legrosszabb esetben $\Theta(n)$ 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 magától ne. Ugyanakkor a keresőfák esetleges műveletek utáni karbantartásának nem lehet akkora a költsége, hogy a végrehajtási idő nagyságrendben romoljon.

A következő részben az AVL fáról, mint speciális bináris keresőfáról lesz szó. Előbb azonban bemutatom a forgatásokat, melyek segítségével úgy módosíthatjuk egy keresőfa szerkezetét, hogy közben nem rontjuk el a keresőfa-tulajdonságot.

2.2.5. 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.
A forgatások műveletigénye O(1), hiszen minden esetben 6 pointert kell átállítani.



2.3. 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.

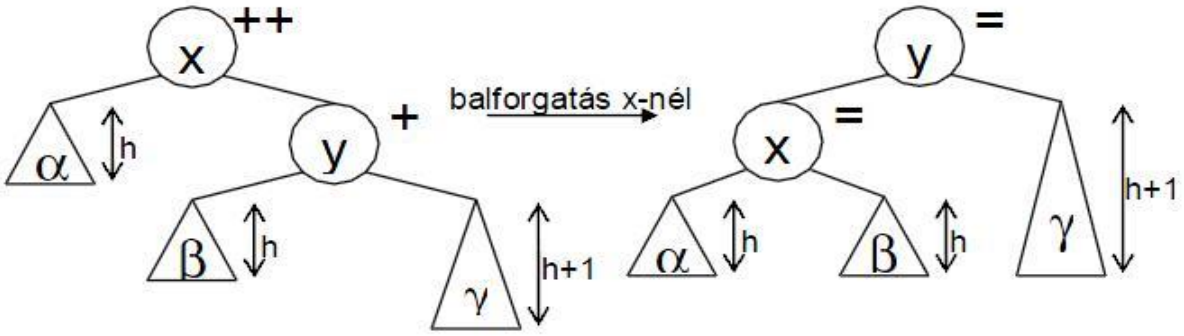
Beszúrás és törlés esetén az AVL-tulajdonság elromolhat, ezért ha szükséges, forgatások segítségével helyre kell azt állítani. A csúcsok egyensúly-faktorának jelölésére vezessük be a szemléletes --, -, = , +, és ++ szimbólumokat.
Az egyensúly --, ha az adott csúcs bal oldali részfája kettővel mélyebb a jobb oldalinál, az egyensúly ++ ha a jobb oldali részfája kettővel mélyebb a bal oldalinál.
Az AVL fák vizsgálata során más egyensúlyi állapot nem fordulhat elő, hiszen mindkét művelet előtt egy kiegyensúlyozott fánk volt, 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.

2.3.1. Beszúrás

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

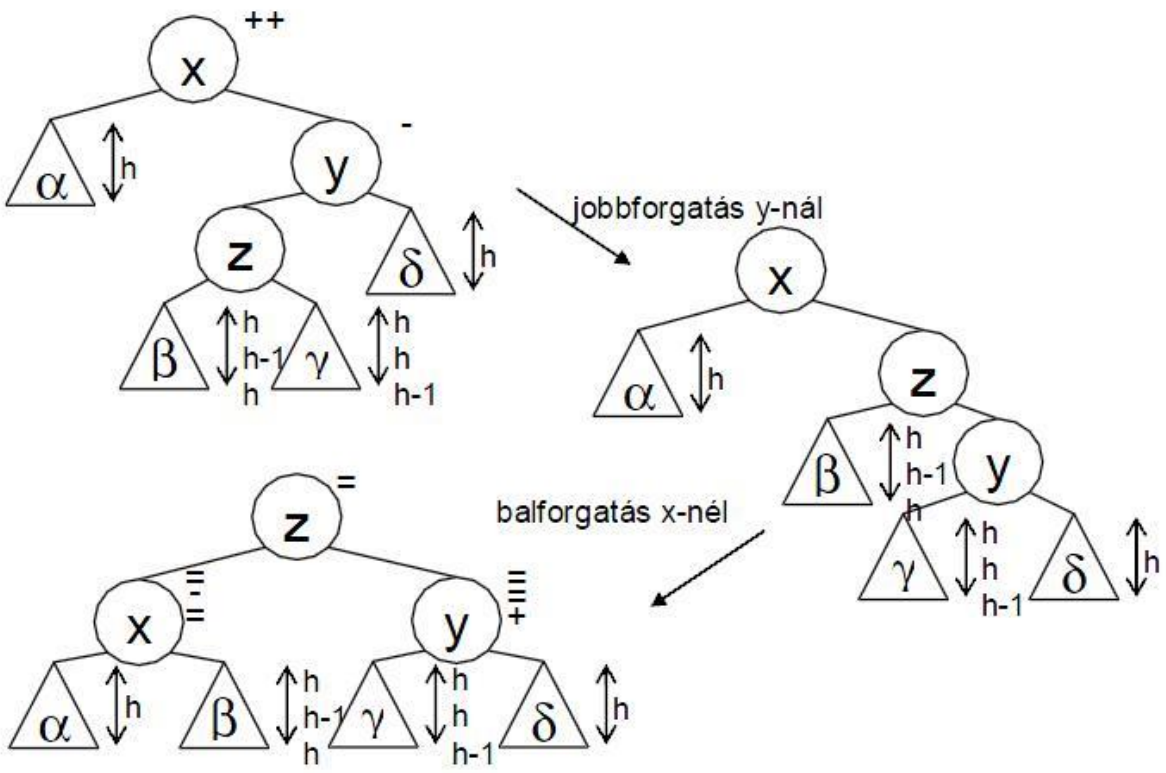
1. (++, +) eset (tükrözése a (--, -))

Ekkor, az 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.



2. (++, -) eset (tükrözése a (--, +))

Ekkor két forgatást is végre kell hajtanunk az 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 azaz 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.



2.3.2. 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 (--, -))

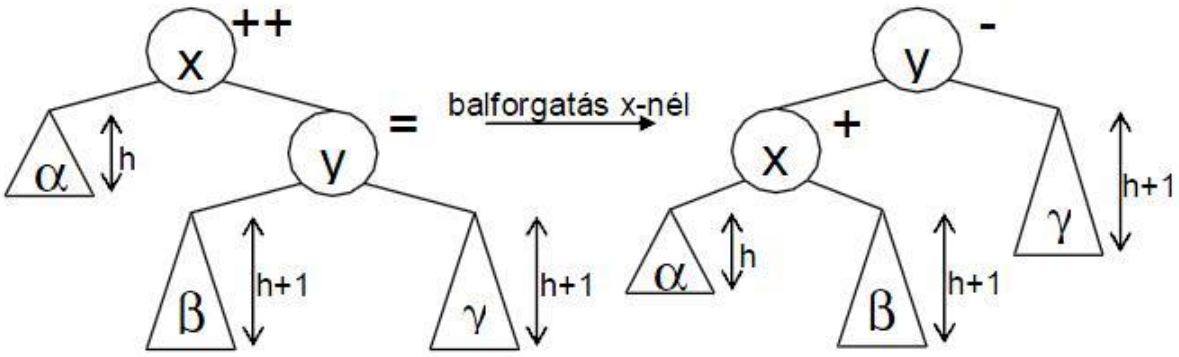
Ekkor ugyanazt kell tennünk, mint amit a beszúrásnál ahogy az első ábrán látható. A különbség az, hogy törlésnél az eredeti részfa magassága nem h+2, hanem h+3, azaz forgatás után a részfa magassága eggyel csökken, ezért tovább kell haladnunk fölfelé a fán.

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. A második ábra mutatja.

3. (++, =) eset (tükrözése a (--, =))

Ez az eset beszúrásnál nem fordulhat elő. A lenti 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, mert csak az α-ból törölhettük ez elemet, így nem kell tovább fölfelé haladnunk a fában.



3. Felhasználói dokumentáció

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

Telepítés

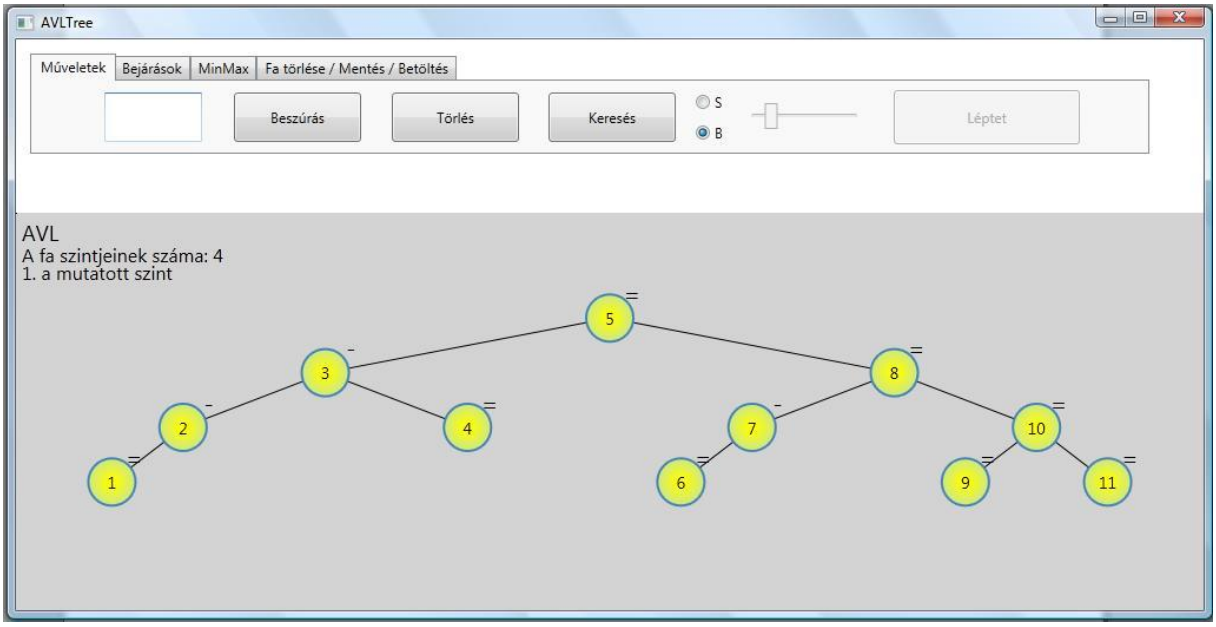
A szoftver telepítést nem igényel, egyszerűen másoljuk fel a cd-n található fájlokat a számítógépre.

A program indítása

A programot az AVLTree.exe fájl futtatásával indíthatjuk.

3.1. A felhasználói felület

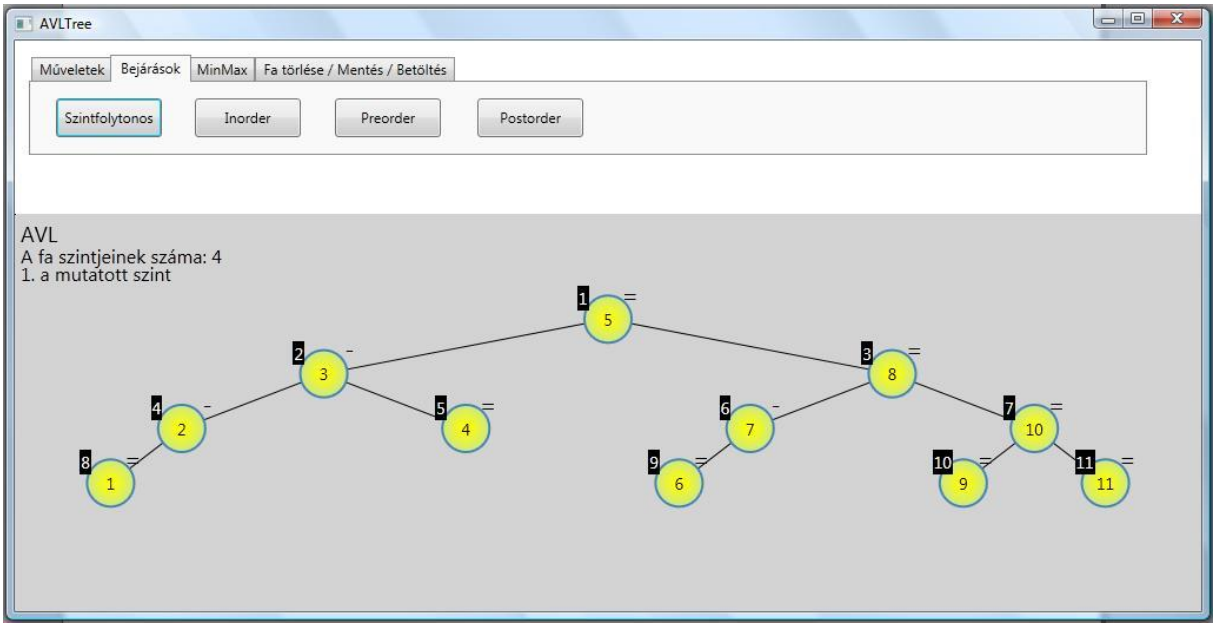
3.1.1. Műveletek felület



A program főfelülete indításnál a műveletek gombsorral fogadja a felhasználót. A szövegmezőbe adhatjuk meg azt a számot amivel műveletet kívánunk végezni az AVL fán. A számot a program [1, 999] zárt intervallumon fogadja el. A beszúrással új elemet tudunk a fához adni, ezt lépésenként mutatja be a program. Először megkeresi az új csúcs leendő helyét, majd végrehajtja a beszúrást és ellenőrzi, hogy kiegyensúlyozott maradt-e a fa. Ha kiegyensúlyozásra van szükség a következő lépésben végrehajtja a megfelelő forgatást. A törléssel törölni tudjuk a fából a megadott elemet. Hasonlóan először lépésenként megkeresi a törlendő csúcsot, majd töröl és annyi forgatást végez a fán amennyire szükség van, hogy kiegyensúlyozott fát kapjunk eredményül. A programot a művelet végrehajtásában pontról pontra lehet léptetni. A kereséssel megkereshetünk egy adott csúcsot, ezt a program pirosra festett számmal jelzi. Az S betűvel jelölt rádiógombbal tehetjük aktívvá az időzítőt. Az aktívvá váló csúszkán állítható az automatikus léptetés időzítése, egészen öt másodpercig. A B betűvel jelölt rádiógomb kikapcsolja az időzítőt, így a felhasználó manuálisan léptetheti tovább a programot a léptet gombbal. Az AVL fa elemeit sárga körökben, élekkel összekötve jeleníti meg a program. A rajzfelület bal felső sarkában mindig megjelenik a fa szintjeinek a száma, valamint hogy hányadik szintjétől jelenítettük meg.

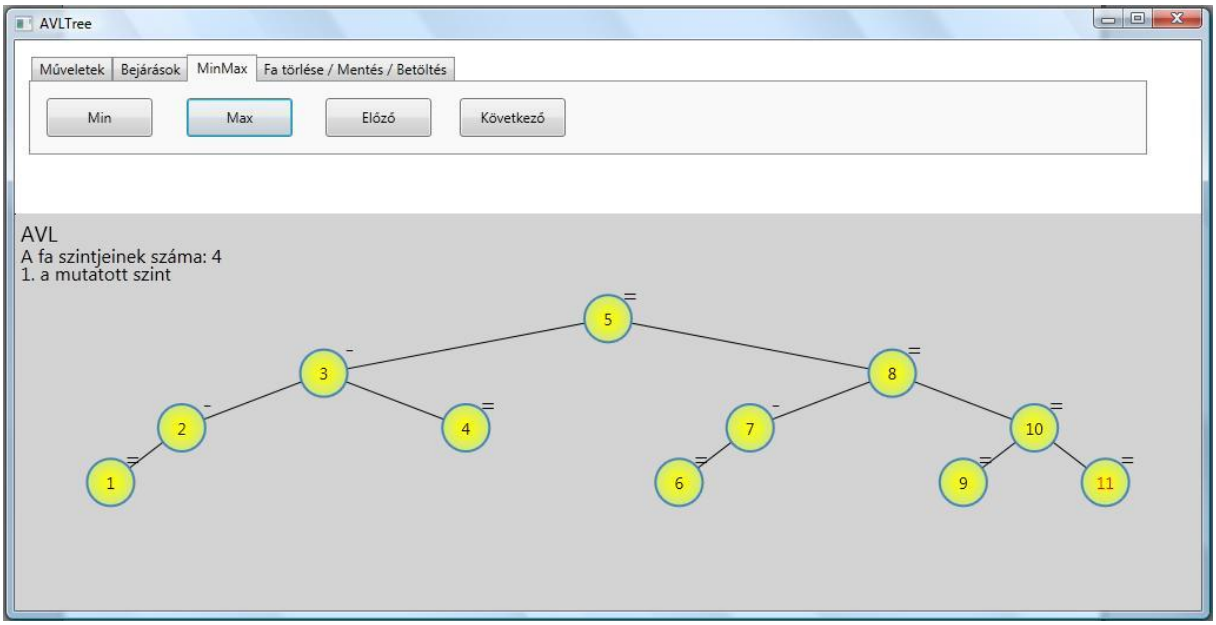
A program öt szint magasan tudja kirajzolni az elemeket. Az ötödik szint alatti elemeket összefogja egybe, amit egy háromszöggel jelöl. Bármely tetszőleges elemre kattintva a fát a választott elem szintjétől és nézőpontjából láthatjuk lefelé. Tetszőleges mélységig navigálhatunk a fában. Egy szinttel feljebb az egér görgőjének egyszeri felfelé fordításával léphetünk. A - , + , = címkék az adott csúcs egyensúlyi állapotát jelzik az AVL fában.

3.1.2. Bejárások felület



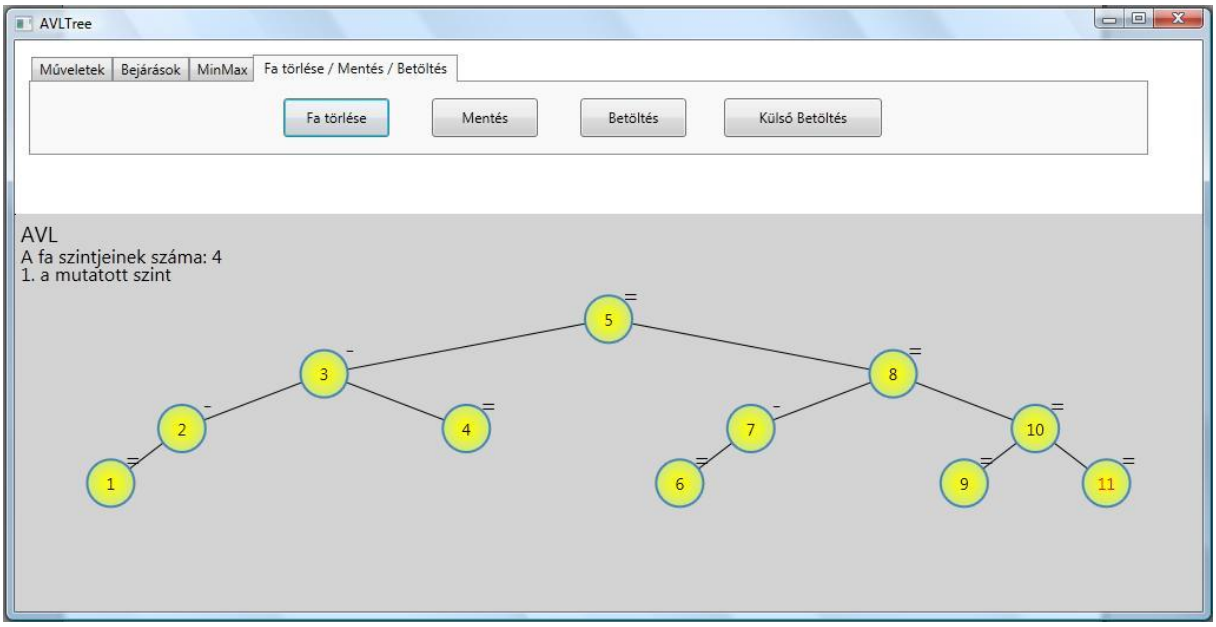
A bejárások fülön négy gomb látható. A szint folytonos bejárás az első gombbal érhető el, ez látható az ábrán az előző példafára.
A további bejárások az inorder, preorder és posztorder szintén hasonló címkézéssel érhető el sorban az adott gombokon. Ha elváltunk a bejárások fülről a bejárás címkéi eltűnnek.
A fában a bejárási nézetben is nyugodtan navigálhatunk.

3.1.3 MinMax felület



A MinMax fül is négy gombot és funkciót tartalmaz. A Min gombbal megkereshetjük a fa minimumát, minimális csúcsát. Az előző és következő gombokkal bejárhatjuk a fa elmeit, a fa csúcsainak rendezése szerint. A Max gomb a maximális elemet jelöli ki, ez látható az ábrán. A navigálás a fában ezen a fülön is elérhető.

3.1.4. Fa törlése / Mentés / Betöltés felület



A mentés fülön is négy funkció érhető el. A fa törlése gomb segítségével törölhetjük a fát a felületről. A mentés gombbal menthetjük a fát, ehhez a program saját speciális formátumot és kiterjesztést (.avl) használ. A betöltés gomb segítségével újra megjeleníthetjük bármelyik a program által mentett fánkat. A külső betöltés a txt formátumú bemenő fájlok megjelenítésére szolgál. Az adatokat soronként kell megadni a fájlban, a számoknak [1,999] zárt intervallumba kell esnie. A beolvasott értékeket egyesével szúrja be egy üres fába, majd ha a fájl végére ért a program megjeleníti a fát. Megadhatunk az elvárttól különböző, hibás bemenő adatokat is, ezeket a program nem veszi figyelembe, ahogy a duplikációkat sem.

4. Fejlesztői dokumentáció

A program WPF (Windows Presentation Foundation) alkalmazásként készült Visual Studio 2010-ban C# nyelven, .NET Framework 3.5 platformon.
Ennek a fejezetnek a célja, hogy elősegítse a programban való tájékozódást, a program továbbfejlesztését és karbantartását.

4.1. A probléma specifikációja és a program felépítése

A feladat egy olyan alkalmazás elkészítése, amely az AVL fa műveleteit szemlélteti, valamint a különböző bejárási módokat is bemutatja. A minimum és maximumkeresés mellett, képes az AVL fákat menteni és betölteni saját és külső forrásból.
A programban három réteget különítünk el egymástól.

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 AVL fa megjelenítésért. Csak a logikai réteg szolgáltatásait használja.

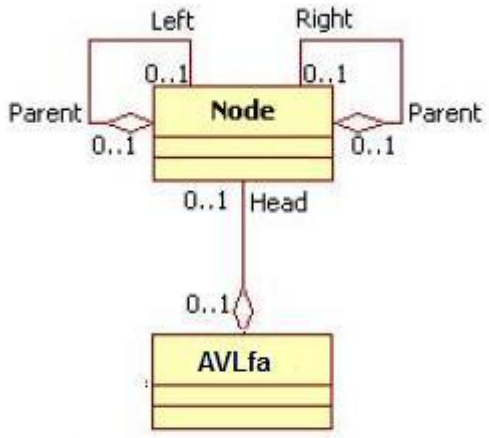
Vezérlési réteg

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 fogom bemutatni.

4.2. A logikai réteg

A logikai rétegben két osztály szerepel, a Node, és az AVLfa.
Az AVLfa rendelkezik egy a gyökércsúcsra mutató referenciával.
Az alkalmazott reprezentációban a csúcsoknak van referenciájuk a szülőre.



A logikai réteg osztálydiagramja

4.3. A Node osztály

4.3.1. 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.

- private byte **_color**;

  A csúcs színe.

Az adattagokból property-k készültek a kényelmes használhatóság érdekében.

4.3.2. Metódusok

- public **Node**(int key)

  A konstruktor inicializálja az adattagokat, valamint beállítja a kulcsértéket.

- public int **height**()

Visszaadja a csúcs által meghatározott részfa szintjeinek számát.

- public void **preOrder**(List<Node> traversalResult)

  A csúcsok felsorolását készíti el preorder bejárás szerint, rekurzívan a csúcsokon keresztül.

- public void **inOrder**(List<Node> traversalResult)

  A csúcsok felsorolását készíti el inorder bejárás szerint, rekurzívan a csúcsokon keresztül.

- public void **postOrder**(List<Node> traversalResult)

  A csúcsok felsorolását készíti el posztorder bejárás szerint, rekurzívan a csúcsokon keresztül.

- public void **levelOrder**(List<Node> traversalResult, Queue<Node> nodeQueue)

  A csúcsok felsorolását készíti el szintfolytonos bejárás szerint, rekurzívan a csúcsokon keresztül. Az algoritmushoz szükséges a sor adattípus.

4.4. Az AVLTree osztály

4.4.1. Adattagok

- private List<Node> **_traversalResult;**

  A bejárások megvalósítása során használt csúcslista.

- protected Node **_head;**

  Referencia a bináris fa gyökérelemére.

- public Node **hold**;

  Jelzi, hogy melyik csúcsnál járunk a fában.

- public int **move**;

  Jelzi, hogy melyik forgatási függvény végrehajtása következik.

- public bool **benne**;

  Mutatja, hogy az adott elem szerepel-e a fában.

- public bool **side**;

  Mutatja, hogy bal vagy jobb gyerek az adott csúcs.

- public bool **ujra**;

  Jelzi, hogy kell-e újra forgatni a fát.

Az egyetlen property a head, ami visszaadja a fejelemet.

4.4.2. Metódusok

- public **AVLfa**()

  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.

- 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.

- 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 minimáliskulcsú 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 jobbraforgatást hajt végre. Előfeltétel, hogy x-nek legyen bal gyereke.

- public void **leftRotation**(Node x)

Az x csúcsnál balraforgatást 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.

- public Node **nextTo**(Node source, int targetKey)

  A függvény egy Node és targetKey kulcs paramétert kap, ha a kulcs kisebb mint a source kulcsa akkor visszaadja a source bal gyerekét, egyébként meg a jobb gyerekét.

- public void **CheckInsertAVL**(Node node)

  Az elméleti részben leírt algoritmus implementációja. A node mutatja, hogy melyik csúcsnál járunk a fában. A kód vizsgálja, hogy mélyült-e a node gyökerű részfa, szüksége-e forgatást végezni. Nem kell tovább haladnunk a fában fölfele, ha nem mélyült a részfa.

- public void **CheckDeleteAVL**(Node node, bool right)

  Az elméleti 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.

- public Node DleftR1(Node node, bool right)

  Törlés utáni (++,+) típusú balra forgatás. Leírása a 9. oldalon található.

- public Node DleftR2(Node node, bool right)

  Törlés utáni (++,-) típusú balra forgatás. Leírása a 9. oldalon található.

- public Node DleftR3(Node node, bool right)

  Törlés utáni (++,=) típusú balra forgatás. Leírása a 10. oldalon található.

- public Node DrightR4(Node node, bool right)

  Törlés utáni (--,+) típusú jobbra forgatás. Leírása a 9. oldalon található.

- public Node DrightR5(Node node, bool right)

  Törlés utáni (--,-) típusú jobbra forgatás. Leírása a 9. oldalon található.

- public Node DrightR6(Node node, bool right)

  Törlés utáni (--,=) típusú jobbra forgatás. Leírása a 10. oldalon található.

- public void rightR2 (Node node)

  Beszúrás utáni (--,+) típusú jobbra forgatás. Leírása a 8. oldalon található.

- public void leftR4 (Node node)

  Beszúrás utáni (++,+) típusú balra forgatás. Leírása a 8. oldalon található.

- public void rightR1(Node node)

  Beszúrás utáni (--,-) típusú jobbra forgatás. Leírása a 8. oldalon található.

- public void leftR3(Node node)

  Beszúrás utáni (++,-) típusú balra forgatás. Leírása a 8. oldalon található.

- public void Qinsert(int value)

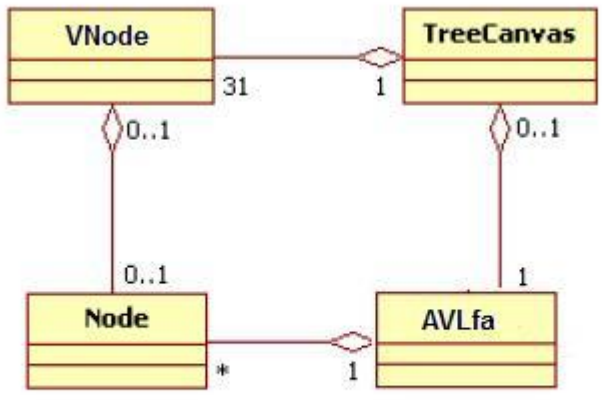
  A txt fájlból történő faépítésnél használt beszúrás. A művelet nincs lépésekre bontva.

- public void QInsertAVL(Node node)

  A txt fájlból történő faépítésnél használt ellenőrzés és forgatás. A művelet nincs lépésekre bontva.

4.5. A megjelenítési réteg

A megjelenítési réteg két osztályt tartalmaz, a VNode-ot és a TreeCanvas-t. Minden TreeCanvas objektumhoz tartozik egy AVLfa példány. Ennek a fának a megjelenítését végzi. Valamint VNode objektumok, egyszerre öt szintű fát tud megjeleníteni. A VNode objektumokhoz legfeljebb egy Node objektum tartozhat, ami a megjelenített fának egy csúcsa. Az alábbi ábrán látható osztálydiagram szemlélteti a megjelenítési és a logikai réteg között fennálló kapcsolatot.



Kapcsolat a megjelenítési és a logikai réteg között

4.6. A VNode osztály

A VNode osztály a beépített Button osztályból származik. Egy WPF nyújtotta lehetőséget kihasználva egy VNode objektum két különböző típusú logikai csúcs megjelenítésére is szolgál. A WPF App.xaml fájljában kétféle button controltemplate található. A „VisualNode” nevű a sárga csúcs sablonja, a „VisualNodeTriangle” sablon pedig a háromszög részfa szimbólumokhoz tartozik.

4.6.1. Adattagok

- private Node **_node;**

  Referencia a VNode-hoz tartozó logikai node-ra.

- private Label **_bejarascimke;**

  Bejárások szemléltetésekor ezen a labelen jelenik meg az, hogy VNode-hoz tartozó logikai csúcs a bejárás során hányadikként lett érintve.

- private Label **_egyensuly**;

  AVL fához tartozó logikai csúcs esetén ezen a labelen jelenik meg a csúcs egyensúlyi állapota (- , = , +) vagy (-- , ++) forgatás előtt.

A _node, _bejarascimke, _egyensuly adattagok property-ken keresztül elérhetők.

4.6.2. Metódusok

- public **VNode**()

  A VNode konstruktora. Kezdetiérték null és egér eseménykezelők beállítása.

- **void VNode_Click(**object sender, RoutedEventArgs e**)**

  Eseménykezelő, ami egy VNode-ra történő kattintáskor fut le. Aktív állapotba hozza a VNode-hoz tartozó TreeCanvast, és a hozzá tartozó logikai node-ra pozícionálja azt.

- **void VNode_MouseDown(**object sender, MouseButtonEventArgs e**)**

  Eseménykezelő, ami jobb egérkattintás esetén kijelöli a VNode-hoz tartozó logikai node-ot, valamint aktív állapotba hozza a TreeCanvast.

4.7. 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.

4.7.1. Adattagok

- private Label **_nameLabel;**

  A megjelenített fa neve.
- private Label **_positionLabel;**

  Hányadik szinttől jelenítjük meg a fát.
- private Label **_heightLabel;**

  A fa szintjeinek a száma.
- private AVLfa **_tree;**

  Referencia a hozzá tartozó logikai fára.
- private List<VNode> **_VNodeList;**

  A VNode-ok listája. A refresh metódus frissíti a listát.
- private List<Line> **_ellista;**

  Kirajzolandó élek listája. A refresh metódus frissíti a listát.
- private List<bool> **_utPozicio;**

  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 **_showegyensuly;**

  Egyensúlyi állapotok kijelzése. Property-n át történő módosítás során automatikusan újrarajzolja a fát.
- private bool **_showBejaras;**

  Bejárási címkék megjelenítése. Propertyn át történő módosítás során automatikusan újrarajzolja a fát.
- private bool **_mutat;**

  Látszódik-e az adott TreeCanvas a képernyőn.

- private Node **_selectedNode;**

  Referencia a kijelölt logikai csúcsra. Értéke null, ha nincs kijelölt csúcs.

- private int **_visualSorszam**;

  A megjelenített szintek száma a fából.

A _tree, _showegyensuly, _showBejaras, _mutat és a_selectedNode adattagok a nekik megfeleltetett property-ken keresztül érhetők el.

4.7.2. Metódusok

- public **TreeCanvas**(AVLfa 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**(AVLfa tree, string text)

  Beállítja a logikai fát, és a fa címkéjét.

- public void **refresh**()

  Frissíti a logikai csúcsok VNode-okhoz rendelését a baseposition alapján, majd újrarajzolja a fát.

- public void **redraw**()

  Újrarajzolja a fát. Meghatározza az összes VNode helyét és méretét, eldönti, hogy melyek látszódnak, tartozik-e hozzájuk logikai csúcs, illetve meghatározza, hogy milyen vezérlősablont alkalmazzon rájuk.
  Ha a VNode az utolsó sorban van, és a hozzá tartozó logikai node-nak van gyereke, akkor részfa szimbólum sablont rendel hozzá.
  A létező éleket is kirajzolja a megfelelő koordinátákra, valamint szükség esetén az egyensúlyi és a bejárási címkéket is megjeleníti.

- public void **moveUp**()

  Feljebb 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 fát a listában leírt jobbra balra lépésekkel az adott csúcsra lépteti.

- public void **showPreOrder**()

  Bejárja a logikai fát preorder módon, majd megjeleníti a bejárási címkéket.

- public void **showInOrder**()

  Bejárja a logikai fát inorder módon, majd megjeleníti a bejárási címkéket.

- public void **showPostOrder**()

  Bejárja a logikai fát postorder módon, majd megjeleníti a bejárási címkéket.

- public void **showLevelOrder**()

  Bejárja a logikai fát szintfolytonos módon, majd megjeleníti a bejárási címkéket.

- private void **posToSelected**()

  Úgy pozícionál a fában, hogy a kijelölt csúcs, ha van ilyen látható legyen.

- public void **showMaximum**()

  Kijelöli a fa maximális kulcsú elemét, majd rápozícionál.

public void **showMinimum**()

  Kijelöli a fa maximális kulcsú elemét, majd rápozícionál.

- public void **showKovetkezo**()

  Kijelöli a logikai fa kijelölt elemének az inorder bejárás szerinti rákövetkezőjét, majd rápozícionál.

- public void **showElozo**()

  Kijelöli a logikai fa kijelölt elemének az preorder bejárás szerinti rákövetkezőjét, majd rápozícionál.

- public void **showNode**(int key)

  Kijelöli a key kulcsú csúcsot ha van ilyen, majd rápozícionál.

- public Node **backNode**(int key)

  Visszaad egy referenciát az adott kulcsú csúcsra.

• public void **ShowRoot**()

Kijelöli a gyökér csúcsot.

• void **TreeCanvas_MouseWheel**(object sender, MouseWheelEventArgs e)

Eseménykezelő, ami az egérgörgő görgetésekor fut le. Ha aktív a TreeCanvas, akkor előre görgetés esetén felfele, hátragörgetés esetén lefele próbál meg lépni a fában.

• void **TreeCanvas_MouseDown**
(object sender, System.Windows.Input.MouseButtonEventArgs e)

Eseménykezelő, ami valamelyik egérgomb lenyomásakor fut le. Aktívvá teszi a TreeCanvast.

• void **TreeCanvas_SizeChanged**
(object sender, System.Windows.SizeChangedEventArgs e)

Eseménykezelő, ami akkor fut le, ha megváltozik a TreeCanvas mérete. Ekkor újrarajzolja a fát.

4.8. 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.

4.8.1. A felhasználói felület kialakítása

Az ablak felső részén kapnak helyet a különböző vezérlőelemek, egy tabcontrol-ba foglalva, amivel a program különböző funkciói elérhetők el.
A vezérlési felületért MainWindow osztály felel és a metódusai.
A részletes magyarázata az egyes funkcióknak a felhasználói dokumentációban olvasható.

4.9. A MainWindow osztály

4.9.1. Adattagok

- private AVLfa avlTree;

  Logikai fa.

- private TreeCanvas avlCanvas;

  VNode fa.

- private int lastInsertedVal;

  Utolsó beszúrt elem.

- private int lastDeletedVal;

  Utolsó törölt elem.

- bool InsertForgat;

  Beszúrás folyamatban van.

- bool DeleteForgat;

  Törlés folyamatban van.

- private int InsertForgatStep;

  Beszúrás folyamat lépés.

- private int DeleteForgatStep;

  Törlés folyamat lépés.

- private int lastSearched;

  Utolsó keresett csúcs.

- private Node success;

  Az adott művelet végrehajtása a success csúcsnál tart.

- private double ido;

  Az időzítő várakozási ideje.

4.9.2. Metódusok

- public **MainWindow**()

  A konstruktor. Egymáshoz rendeli a TreeCanvas-t és a logikai fát, inicializál.

- private bool **insert**(int value)

  Megpróbálja beszúrni a value kulcsú új csúcsot a fába.

Igazzal tér vissza, ha sikerült, hamissal, ha nem.

- private bool **delete**(int value)

  Megpróbálja kitörölni a value kulcsú csúcsot a fából.

Igazzal tér vissza, ha sikerült, hamissal, ha nem.

- private void **beszuras_Click**(object sender, RoutedEventArgs e)

  Eseménykezelő, ami a műveletek tabon elhelyezkedő beszúrás gomb megnyomásakor fut le. Megpróbálja számmá konvertálni a textbox tartalmát. Ha sikerül, meghívja az insert metódust.

- void **timer_Tick**(object sender, EventArgs e)

  Ha a timer időzítője megáll, meghívja a léptet metódust.

- private void **torles_Click**(object sender, RoutedEventArgs e)

  Eseménykezelő, ami a műveletek tabon elhelyezkedő törlés gomb megnyomásakor fut le. Megpróbálja számmá konvertálni a textbox tartalmát. Ha sikerül, meghívja a delete metódust.

- private void **kereses_Click**(object sender, RoutedEventArgs e)

  Eseménykezelő, ami a műveletek tabon elhelyezkedő kereses gomb megnyomásakor fut le. Megpróbálja számmá konvertálni a textbox tartalmát. Ha sikerül, akkor megkeresi, és kijelöli a kért kulcsú elemet.

- private void **preorder_Click**(object sender, RoutedEventArgs e)

  Az AVL fán preoder bejárást hajt végre.

- private void **inorder_Click**(object sender, RoutedEventArgs e)

  Az AVL fán inorder bejárást hajt végre.

- private void **postorder_Click**(object sender, RoutedEventArgs e)

  Az AVL fán postorder bejárást hajt végre.

- private void **szintfolytonos_Click**(object sender, RoutedEventArgs e)

  Az AVL fán szintfolytonos bejárást hajt végre.

- private void **tabChanged**(object sender, SelectionChangedEventArgs e)

  Eseménykezelő, ami a tabkontrolon történt váltáskor fut le. Megszünteti a bejárások kijelzését, és a szövegdobozt törli.

- private void **mentes_Click**(object sender, RoutedEventArgs e)

  Eseménykezelő, ami a fák elmentését valósítja meg. Kivétel esetén egy felugró ablakot jelenít meg.

- private void **betolt_Click**(object sender, RoutedEventArgs e)

  A program által lementett .avl kiterjesztésű fáinkat tudjuk megjeleníteni a segítségével.

- private void **uj_Click**(object sender, RoutedEventArgs e)

  A fa törlését valósítja meg.

- private void **min_Click**(object sender, RoutedEventArgs e)

  A minimális kulcsértékű elemet keresi meg.

- private void **max_Click**(object sender, RoutedEventArgs e)

  A maximális kulcsértékű elemet keresi meg.

- private void **elozo_Click**(object sender, RoutedEventArgs e)

  A kijelölt csúcshoz képest visszaadja az inorder bejárás szerinti megelőző csúcsot.

- private void **kovetkezo_Click**(object sender, RoutedEventArgs e)

  A kijelölt csúcshoz képest visszaadja az inorder bejárás szerinti következő csúcsot.

- private void **buttonNextStep_Click**(object sender, RoutedEventArgs e)

  Az adott művelet (beszúrás, törlés, keresés) végrehajtását a következő pontra lépteti.

- private void **TorlesLabel**()

  Kiírja a szövegdobozba, hogy törlés után milyen forgatás fog következni.

- private void **radioButton1_Checked**(object sender, RoutedEventArgs e)

  Az időzítő aktív, a program automatikusan léptet.

- private void **radioButton2_Checked**(object sender, RoutedEventArgs e)

  Az időzítő inaktív, a programot manuálisan kell léptetni.

- private void **txt_Click**(object sender, RoutedEventArgs e)

  Külsőleg szerkesztett txt fájlból történő faépítést hajt végre. Csak a helyes bemenő adatokat veszi figyelembe, [1, 999] zárt intervallumon.

- private void **sliderInterval_ValueChanged**

(object sender, RoutedPropertyChangedEventArgs<double> e)

Változtatja az időzítő várakozási idejét, a csúszka segítségével.

5. Tesztelés

A tesztelési folyamatot két részre bonthatjuk. Először a keresőfák algoritmusainak helyes működését ellenőrizzük. Vagyis az első 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 tesztelés a következő konfiguráción történt:

- Windows Vista SP1, .NET Framework 4.0 SP1
- 1280*800-as felbontás
- Core 2 Duo E6420 processzor
- 2 GB memória

5.1. A logikai réteg szolgáltatásainak tesztelése

A teszteken a fa módosítására szolgáló függvények egyes eseteit vettem alapul. Az itt felsorolásra kerülő tesztek például szolgálnak néhány fontosabb esetre.

1. Első elem beszúrása.
2. Sokadik elem beszúrása.
3. Egy levélelem törlése.
4. Egy olyan csúcs törlése, aminek egy jobb gyereke van.
5. Egy olyan csúcs törlése, aminek egy bal gyereke van.

6. Egy olyan csúcs törlése, aminek két gyereke van.
7. A gyökérelem törlése.
8. Beszúrás (++,+) eset.
9. Beszúrás (--.-) eset.
10. Beszúrás (++,-) eset.
11. Beszúrás (--,+) eset.
12. Törlés (++,+) eset.
13. Törlés (--,-) eset.
14. Törlés (++,=) eset.
15. Törlés (--,=) eset.
16. Törlés (++,-) eset.
17. Törlés (--,+) eset.
18. Olyan csúcs törlése, ami után több forgatás szükséges.

5.2. A felhasználói felület és a programfunkciók tesztelése

1. Próbáljunk a specifikációnak nem megfelelő értéket beszúrni a fába. Például 1-999 intervallumon kívül eső számot; nem szám karaktert tartalmazó karaktersorozatot.
2. Navigáljunk az egyik csúcsra kattintással, egér görgővel. Próbáljunk túlmenni a fán. Lelépni egy levélről, vagy a fa gyökeréből lépni felfele.
3. Töröljünk egy olyan elemet a fából, amelyik a megjelenítőn a részfa gyökere, és nincsen gyereke. Ekkor a megjelenítés a szülőre lép, ha létezik.
4. Külső fájlból történő faépítésnél töltsünk be egy olyan fájlt, aminek egyetlen sora sem felel meg a specifikációnak, ami üres; aminek néhány sora megfelelő, minden sora megfelelő.
5. Váltogassunk automatikus és manuális léptetés között.
6. Faépítés közben próbáljuk változtatni az ablak méretét.
7. Faépítés közben navigáljunk a fában.
8. Egy fájl betöltése után hajtsunk végre egyesével történő beszúrásokat és törléseket.
9. Próbáljuk ki az előző, és következő műveleteket úgy, hogy nincs kijelölt elem. Ilyenkor nincs hatása műveleteknek.

10. Navigáljunk a fában, miközben a bejárási címkék látszódnak.
11. Mentsük el munkánkat, majd töltsük vissza.
12. Fájl betöltésnél a felugró ablakon kattintsunk a mégse gombra, és vizsgáljuk hogy van-e nem várt hatása.
13. Fájl mentésnél a felugró ablakon kattintsunk a mégse gombra, és vizsgáljuk hogy van-e nem várt hatása.
14. Próbáljunk betölteni egy hibás avl fájlt.
15. Próbáljunk ki minden gombot és funkciót.

5.3. Továbbfejlesztési lehetőségek

A program egyik lehetséges továbbfejlesztési módja, hogy a faépítés funkció használata közben legyen lehetőség visszafele is léptetni.
Demonstrációs program lévén egy másik kézenfekvő lehetőség az, hogy a szoftvert animációkkal tegyük látványosabbá.

6. Összegzés

Az alkalmazás elkészítése során fontosnak tartottam, hogy kezelése bárki számára nagyon könnyen elsajátítható legyen. Ne legyen a program felületén olyan részlet ami nem érthető. A műveletek végzése közben mindig csak azok a funkciók legyenek elérhetőek, amire feltétlenül szükség van a továbblépés érdekében. Alapállapotban minden funkció elérhető, attól függően melyik fül van kiválasztva. A program ablakmérete tetszőlegesen nagyítható, ezzel is segítve az AVL fa szerkezetének bemutatását, megértését.
Úgy gondolom, a szakdolgozatomként elkészített program, az elméleti résszel kiegészítve egy könnyen használható oktatási segédanyag.

7. 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] Niklaus Wirth : Algoritmusok + Adatstruktúrák = Programok, Műszaki Könyvkiadó, 1982, [345], ISBN: 9631038580

[4] http://en.wikipedia.org/wiki/AVL_tree (2012. május)

[5] http://people.inf.elte.hu/fekete/ (2012. május)