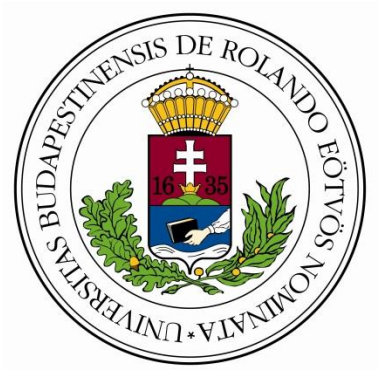


Eötvös Loránd Tudományegyetem
Informatikai Kar

2-3-áris fák műveleteinek grafikus szemléltetése

Témavezető:
dr. Ásványi Tibor
egyetemi docens

Készítette:
Balogh Péter
Programtervező Informatikus
BSc. szakos hallgató

Budapest, 2012

Tartalomjegyzék

1. Bevezetés

A számítógépek többsége elsődleges és másodlagos memóriát is tartalmaz. A technológiák különbségéből adódóan az elsődleges memória előállítási költsége 2 nagyságrenddel nagyobb, mint a másodlagosé, míg a tárolókapacitás tekintetében hasonló a különbség a másodlagos memória javára. A másodlagos tárhoz való „véletlenszerű” hozzáférés azonban mechanikus mozgást igényel, emiatt ez mintegy 5 nagyságrenddel lassabb, mint az elsődleges memória elérése. Így a jellemzően másodlagos memóriában tárolt adatokhoz való hozzáférést úgy tehetjük hatékonyabbá, ha minimalizáljuk a szükséges elérések számát. A cél egy olyan dinamikus adatszerkezet megalkotása, amelynek használatával a lehető legkevesebb lépésben elérhetjük a kívánt adatot. Ilyen adatszerkezetek a keresőfák. Különböző fájl- és adatbázisrendszerekben széles körben elterjedt a B-fák használata. A keresőfa gyökércsúcsát folyamatosan az elsődleges memóriában tartva egy tetszőleges rekord helyének meghatározása h-1 lemezolvasással megtörténik, ahol h a fa magassága. Természetesen a szintek száma a tárolt rekordok számával nő, azonban ezt igen lassan, logaritmikusan teszi. Egy tipikus B-fa használata esetén két lemezolvasással akár egymilliárd rekord közül is kiválasztható a megfelelő, a fa elágazási tényezőjétől függően[1]. A 2-3-áris fák gyakorlati alkalmazása ugyan nem mondható ilyen elterjedtnek, jelentőségüket az adja, hogy a B-fák elődjének tekinthetők, illetve felfoghatók olyan speciális B-fákként, amelyek elágazási tényezője 2 vagy 3. Viselkedésük megértése nagyban segítheti a látszólag bonyolultabb szerkezetű B-fák viselkedésének megértését is. Dolgozatommal ebben kívánok segítséget nyújtani a felhasználóknak, egy 2-3-áris fa különböző állapotainak és állapotátmeneteinek interaktív, grafikus megjelenítésén keresztül.

2. Elméleti háttér

Ebben a fejezetben röviden áttekintem a 2-3-áris fákra és műveleteikre jellemző fontosabb tulajdonságokat.

2.1. Alapfogalmak

Néhány alapfogalom, amelyeket a későbbiekben használni fogok:

Def. fa: olyan gráf, ami összefüggő, és körmentes [2].

Def. irányított fa: olyan fa, aminek van olyan csúcsa, amiből minden más csúcshoz vezet irányított út[2]. „Fa" alatt a továbbiakban irányított fát értek.

Def. gyökér: az irányított fának az az egyértelműen meghatározott csúcsa, amelyből minden csúcshoz vezet irányított út [2].

A fenti definíciókból következnek az alábbiak:

- a gyökér a fa egyetlen olyan csúcsa, amelybe nem vezet él
- minden más csúcsba pontosan egy él vezet
- minden csúcs egyértelműen elérhető a gyökérből

Def. „*r*-áris fa": az egyes csúcsokból kivezető élek száma minden fára korlátos ($r>0$). Ha a fa minden (nem levél) csúcsából pontosan r él vezet ki, akkor a fát r-áris fának nevezzük [3].

Def. gyermek: egy x csúcs gyermekei azok a csúcsok, melyekbe x-ből él vezet. Ha x-nek két gyermeke van, akkor ezek tipikus címkézése a „bal-" illetve „jobboldali" gyermek, illetve három gyermek esetén a bal- és jobboldali gyermek között megjelenő „középső" gyermek.

Def. szülő: egy y csúcs szülője az x csúcs, ha x-ből vezet él y-ba.

Def. belső csúcs: olyan csúcs, melyből vezet ki él.

Def. levél: olyan csúcs, amelyből nem vezet ki él.

Def. utolsó belső csúcs: olyan belső csúcs, amelynek minden gyereke levél.

Def. részfa: egy t fa részfája t' fa, ha t' gyökere t tetszőleges belső csúcsa.

Dolgozatomban egy belső csúcs „bal-, jobb-, középső részfája" alatt azt a részfát értem,

amelynek gyökere a vizsgált csúcs baloldali, jobboldali, illetve középső gyermeke. Részfa állhat akár egyetlen levél csúcsból is.

Def. fa magassága: a gyökértől a legtávolabbi levélig vezető út hossza.

Def. kulcs: levél csúcsban szereplő érték.

Def. keresőkulcs: belső csúcsban szereplő érték. Ahol nem okoz félreértést, a keresőkulcsot is kulcsnak hívom. Ha egy csúcsban 2 keresőkulcs is szerepelhet, akkor ezek tipikus címkézése „baloldali", és „jobboldali" kulcs.

Def. csúcs bejárása: tetszőleges művelet elvégzése a csúcsban található valamely kulcson, kulcsokon.

Def. fa bejárása: a fa valamely előre meghatározott csúcsából kiindulva bejárjuk az összes csúcsát, valamilyen meghatározott szabály szerinti sorrendben.

2.2. A 2-3-áris fa definíciója

A tárolt adatrekordok helyével és számával kapcsolatosan többféle értelmezés is elfogadott. Jelen dolgozatban olyan 2-3-áris fákkal foglalkozom, amelyek a rekordok kulcsait csak a levelekben tárolják (belső csúcsban nem), minden levélben pontosan egy rekord kulcsa szerepel, és a fában egy adott kulccsal legfeljebb egy levél fordulhat elő. A gyakorlatban a tárolt adatok összetett szerkezetűek, különböző típusú információelemekből állnak. Ezek közül azonban csak egy (vagy néhány) adja a tárolás és keresés alapját jelentő kulcsot. A fát karbantartó algoritmus szempontjából közömbös, hogy ezen a felhasznált kulcson kívül még milyen elemek alkotják a tárolt struktúrát, így az egyszerűség kedvéért feltehetjük, hogy csak ezt a kulcsot kell a levelekben tárolnunk [4].

Def. „gyenge" 2-3-áris fa: olyan fa, amelyre teljesülnek az alábbiak:

- minden belső csúcs rendelkezik legalább egy keresőkulccsal, ez a „baloldali" kulcs (megjegyzés: erre a kulcsra akkor is „baloldali kulcs"-ként fogok hivatkozni, ha a csúcsban nincs másik kulcs)
- bármely belső csúcs rendelkezhet egy második keresőkulccsal, ez a „jobboldali" kulcs
- minden belső csúcs ismeri a baloldali részfájának gyökerét, azaz a saját baloldali gyermekét, kivétel ez alól az egyetlen elemű fa gyökere, amelynek nincs

baloldali gyermeke [5]

- minden belső csúcs ismeri a jobboldali részfájának gyökerét, azaz saját jobboldali gyermekét [5]
- ha egy belső csúcsban van jobboldali kulcs, akkor (és csak akkor) a csúcsnak van középső részfája, melynek gyökerét a csúcs szintén ismeri
- minden levélcsúcs rendelkezik pontosan egy kulccsal
- minden csúcs ismeri a szülőjét, kivéve a gyökércsúcsot, melynek nincsen szülője

Def. „gyenge" 2-3-áris fa inorder bejárása: a fa gyökeréből kiindulva az összes csúcs rekurzív módon történő bejárása úgy, hogy minden csúcsnak először a baloldali részfáját járjuk be, majd elvégezzük az adott műveletet a csúcs baloldali kulcsán. Ezután bejárjuk a fa középső részfáját, és elvégezzük a műveletet a jobboldali kulcson – ha ezek léteznek. Végül bejárjuk a jobboldali részfát. (1. ábra)

```
inOrder(n)
   if (n = null)
      skip
   else
      inOrder(n.leftChild)
      process(n.leftKey)
      if (n.rightKey != null)
         inOrder(n.middleChild)
         process(n.rightKey)
      end if
      inOrder(n.rightChild)
   end if
end
```

1. ábra
2-3-áris fa inorder bejárásának pszeudo-kódja

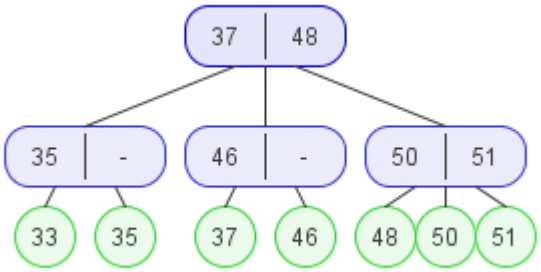
Def. 2-3-áris fa keresőfa-tulajdonsága: egy 2-3-áris fa teljesíti a keresőfa-tulajdonságot, ha végrehajtva rajta az inorder bejárást, a fában szereplő levelek kulcsainak szigorúan monoton növő sorozatát kapjuk, és igaz, hogy:

- minden belső csúcs baloldali részfájában szereplő összes kulcs kisebb, mint a csúcs baloldali keresőkulcsa
- ha egy belső csúcsban két keresőkulcs szerepel, akkor ezek közül a jobboldali mindig nagyobb, mint a baloldali, továbbá a baloldali keresőkulcs a középső részfában szereplő kulcsok minimuma, míg a jobboldali keresőkulcs a jobboldali

részfában szereplő kulcsok minimuma

– ha egy belső csúcsban egy keresőkulcs szerepel, akkor az a jobboldali részfájában szereplő kulcsok minimuma

A keresőfa-tulajdonság definíciójából következik, hogy a keresőfában egy adott elemet egyértelműen meg lehet keresni: A keresett értéket minden csúcsban összehasonlítjuk az ott levő keresőkulccsal, és a fennálló reláció alapján eldöntjük, hogy a csúcs melyik gyermekében kell a keresést folytatnunk [6]. Igaz továbbá, hogy minden levél kulcsa szerepel a fa valamelyik felsőbb szintjén, keresőkulcsként is (kivétel a bal szélső levél). (2. ábra)



2. ábra
Keresőkulcsok és levelek viszonya

Def. 2-3-áris fa: olyan „gyenge” 2-3-áris fa, melyre igazak a következők:

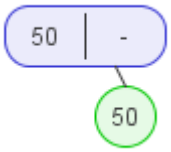
– kulcsértékei olyan típusból kerülnek ki, amely felett rendezés értelmezett [4]
– levelei egy szinten, azaz a gyökértől megegyező távolságra helyezkednek el [5]
– keresőfa-tulajdonság [4]

A fenti definícióból következik, hogy egy h magasságú fa $2^h \leq n \leq 3^h$ elem tárolására képes. Ugyanez az összefüggés megfordítva: $log_3 n \leq h \leq log_2 n$.

Speciális esetek

Mindössze két olyan eset van, amelyben egy 2-3-áris fa tulajdonságai eltérnek a megszokottól. Az egyik az egyetlen elemű fa, a másik az üres fa esete. Értelmezés kérdése, hogy ezeket hogyan ábrázoljuk. Egyelemű fa ábrázolására az egyik lehetőség egyetlen csúcs létrehozása, amely a fa gyökere és egyetlen levele is egyben. A másik

lehetőség olyan fa, amelynek van külön gyökere, és annak gyermeke a fa egyetlen levele. A definícióból adódóan a levél mindig jobboldali gyermeke a gyökérnek. (3. ábra) Az üres fát reprezentálhatja üres gyökér, vagy „semmi” (*NIL*). Dolgozatomban mindkét esetre a másodikként említett módot használom.



3. ábra
Egyelemű fa

2.3. 2-3-áris fán értelmezett műveletek

2.3.1. Keresés

A gyökérből indul, ahol összehasonlítjuk a keresett értéket a gyökér baloldali kulcsával. Ha nem kisebb, és a gyökérnek van jobboldali kulcsa is, akkor azzal is összehasonlítjuk.

```
Search (n, v)                          //n csúcsból indulva v értéket keressük.
                                       //kívülről a gyökérre hívjuk
  if (n = null)                        //ha a csúcs nem létezik (üres a fa),
    return -1                          //extremális elemet adunk vissza
  else
    if (n.isLeaf)                      //ha levélbe értünk, megnézzük, a kulcsa a keresett érték-e
      if (n.leftKey = v)               //ha igen, akkor visszaadjuk a levél azonosítóját
        return n.nodeId
      else
        return -1                      // ha nem, akkor extremális elemet adunk vissza
      end if
    else                               //egyéb esetben rekurzívan hívjuk a függvényt
      if (v < n.leftKey)               //a megfelelő részfára, amit az alábbiakban választunk ki
        return search(n.leftChild, v)
      else
        if (n.rightKey = null)
          return search(n.rightChild, v)
        else
          if (v < n.rightKey)
            return search(n.middleChild, v)
          else
            return search(n.rightChild, v)
          end if
        end if
      end if
    end if
  end if
end
```

4. ábra
Keresés pszeudo-kódja

Az eredmény alapján kiválasztjuk a gyökér valamelyik részfáját, majd addig ismételjük rekurzívan ezeket a lépéseket, amíg el nem jutunk egy levélbe. (4. ábra) Ha a keresett érték szerepel a fában, akkor az azt tartalmazó levélbe jutottunk. Ha a levél kulcsa nem a keresett érték, akkor azt a fa biztosan nem tartalmazza.

2.3.2. Beszúrás

A művelet célja egy új adatrekord kulcsának elhelyezése a fában úgy, hogy az a már bent levő rekordokhoz hasonló módon kereshető legyen.

Első lépésként végrehajtunk a fán egy keresést a beszúrni kívánt kulccsal. Ha ennek során kiderül, hogy a fa már tartalmaz ilyen kulcsú levelet, akkor nincs további teendőnk, hiszen kikötöttük, hogy a kulcsok egyediek. Ellenkező esetben az első feladat, hogy meghatározzuk a leendő új levél helyét, vagyis azt az utolsó belső csúcsot, amely majd az új levél szülője lesz. A szülő-jelölt keresése a „normális" kereséshez hasonló, a különbség csak annyi, hogy ha levélcsúcsba futottunk, akkor visszatérünk annak szülőjébe, és ez a csúcs lesz a keresés eredménye. A további lépéseket a beszúrást szenvedő csúcsban levő keresőkulcsok száma határozza meg.

a) Beszúrás egykulcsú csúcsba

Ha a csúcsnak csak egy kulcsa, azaz csak két levele van, akkor beilleszthetünk egy második kulcsot és egy harmadik levelet anélkül, hogy ezzel megsértenénk a fa invariáns tulajdonságait. A 3 levelet balról jobbra növekvő sorrendbe helyezzük, a két nagyobb kulcsot beírjuk a szülőbe, szintén ügyelve a sorrendjükre, és készen is vagyunk. (5. ábra)

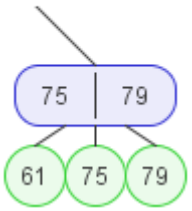
```
insert(n, v)                    //n csúcsba szúrjuk a v értéket
   m := new Node                //létrehozzuk az új levelet
   m.leftKey := v               //beállítjuk a kulcsát
   m.parent := n                //beállítjuk a szülőjét
   if (v < n.leftKey)           //eldöntjük, n-nek melyik gyermeke lesz az új levél és
                                //ennek megfelelően rendezzük a leveleket és a kulcsokat
      n.rightKey := n.leftKey
      if (v < n.leftChild.leftKey)
         n.leftKey := n.leftChild.leftKey
         n.middleChild := n.leftChild
         n.leftChild := m
      else
         n.leftKey := v
         n.middleChild := m
      end if
   else
      n.rightKey := v
      n.middleChild := n.rightChild
      n.rightChild := m
   end if
end
```

5. ábra

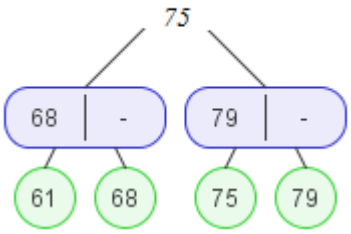
Egykulcsú csúcsba történő beszúrás pszeudo-kódja

b) Beszúrás kétkulcsú csúcsba, „csúcsvágás”

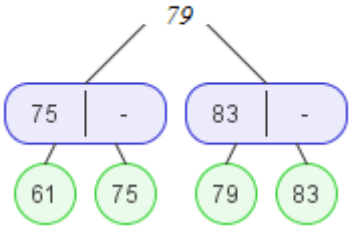
A beszúrást követően összesen 4 levelet kellene belső csúcs gyermekeiként elhelyeznünk. Ha szeretnénk megőrizni a fa definíció szerinti tulajdonságát, miszerint egy belső csúcsnak 2 vagy 3 gyermeke lehet, akkor a 4 levélhez két ilyen csúcsra lesz szükségünk az eddigi egy helyett. Az egyik csúcs az lesz, amely eddig is szerepelt a fában, nevezzük ezt *x*-nek. Mellette létre kell hoznunk egy újat, legyen ez *y*. Legyen továbbá *x* szülője *s*. Hogy az új csúcs melyik oldalán lesz a réginek, attól függ, hogy az éppen beszúrt kulcs értéke hogyan viszonyul az eddig is ott levőkhöz. Van tehát 4 levelünk, két belső csúcsunk, és 3 keresőkulcsunk, melyek közül kettő eddig is a fában (*x*-ben) volt, míg a harmadik az új levéllel együtt érkezik. Ahhoz, hogy a módosuló részfa keresőfa-tulajdonságát is megőrizzük, ezeknek az elemeknek a következőképpen kell elhelyezkedniük: a levelek balról jobbra olvasva növekvő sorrendben vannak. Az első kettő felett áll az egyik belső csúcs, az utolsó kettő felett a másik. Az elöl álló belső csúcsban a legkisebb értékű keresőkulcs kap helyet, a hátul állóban pedig a legnagyobb értékű. A nagyság szerint középső keresőkulcs egy szinttel feljebb kerül a fában. (6-8. ábra)



6. ábra
Két kulcsú csúcs beszúrás előtt…



7. ábra
68 beszúrása után



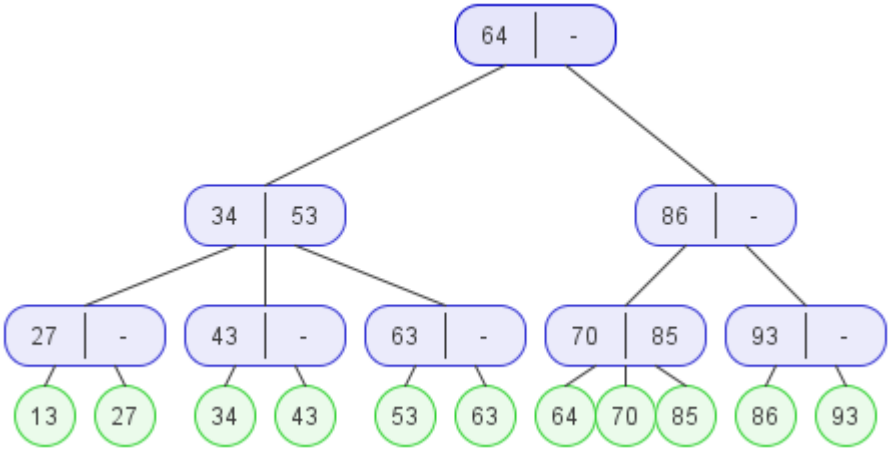
8. ábra
83 beszúrása után

Az „elvándorolt” keresőkulcs a fa eggyel magasabb szintjén jelenik meg „új”-ként, beszúrása *s*-be hasonló lesz a levél iménti beszúrásához: ha *s*-ben eddig egy kulcs volt, akkor mostantól kettő lesz, az új kulcstól balra levő mutató *x* és *y* közül a baloldalira fog mutatni, a jobboldali mutató pedig a jobboldalira. Ha *s*-ben már eddig is két kulcs volt, akkor *x*-hez hasonlóan *s* is kettéválik. Ezt a jelenséget nevezzük felgyűrűző csúcsvágásnak. A felgyűrűzés addig folytatódik, míg egy egykulcsú csúcs el nem nyeli az alatta levő szintről érkező kulcsot. Ha nincs ilyen csúcs, akkor a beszúrás hatása egészen a gyökérig gyűrűzik.

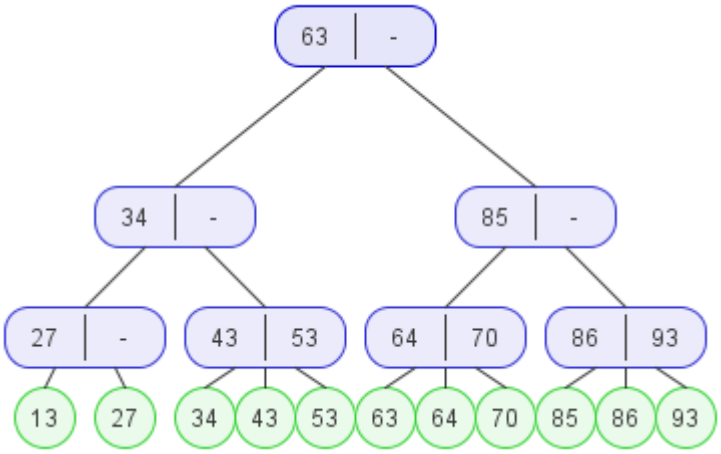
c) Beszúrás a gyökérbe

A gyökér, az összes többi belső csúcshoz hasonlóan tartalmazhat egy vagy két kulcsot. Ha csak egyet tartalmaz, akkor viselkedése megegyezik az egykulcsú egyéb csúcséval. A kétkulcsú gyökeret az eddig látott módon kettévágjuk, és a két egykulcsú csúcsnak létrehozunk egy szintén egykulcsú szülőt, melynek kulcsa a régi gyökér szintjéről érkező kulcs. A fa gyökere ettől kezdve az új csúcs, a magassága eggyel nőtt. Mivel ez a növekedés a fa tetején történt, továbbra is igaz, hogy minden levél azonos hosszúságú úton érhető el.

Fontos megjegyeznünk, hogy a 2-3-áris fák alakja nem determinisztikus. Ha ugyanazokat az értékeket különböző sorrendben szúrjuk be, akkor a beszúrás két esete is különböző sorrendben kerül végrehajtásra, így megegyező tartalmú, de különböző szerkezetű fák jönnek létre. (9-10. ábra)



9. ábra
A beszúrások sorrendje: *64, 86, 53, 34, 70, 93, 43 85, 63, 13, 27*



10. ábra
A beszúrások sorrendje: *27, 13, 63, 85, 43, 93, 70, 34, 53, 86, 64*

d) Speciális beszúrások

Az eddig tárgyaltaktól eltér az üres fába, valamint az egyelemű fába történő beszúrás. Előbbi esetben nemcsak levelet kell létrehoznunk a beszúrandó kulccsal, hanem a fa gyökerét is. A levél mindig a gyökér jobboldali gyermeke lesz. A második elem beszúrásakor már csak levelet hozunk létre. Ha ennek kulcsa kisebb, mint a gyökérben már szereplő keresőkulcs, akkor egyszerűen beillesztjük a bal gyermek helyére. Ha nagyobb, akkor a kulcsa felülírja a keresőkulcsot, az eddigi jobb gyermekből bal gyermek lesz, az új levél pedig a jobb gyermek pozíciójába kerül.

2.3.3. Törlés

A művelet célja, hogy egy szükségtelenné vált elemet eltávolítsunk a fából úgy, hogy a fa eddig meglévő tulajdonságai továbbra is teljesüljenek. Első lépésként végrehajtunk a

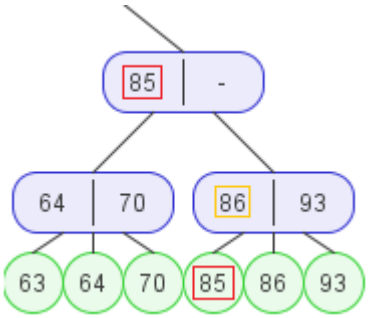
fán egy keresést a törölni kívánt kulccsal. Ha ennek során kiderül, hogy a fa nem tartalmaz ilyen kulcsú levelet, akkor nincs további teendőnk. Ellenkező esetben az első feladat, hogy meghatározzuk a törlendő elem helyét, valamint azt a belső csúcsot, mely a törlendő levél kulcsát keresőkulcsként tartalmazza. Tudjuk, hogy – hacsak nem a fa legkisebb kulcsú levelét töröljük – lesz ilyen belső csúcs. A további lépések a törlendő levél helyzetétől függnek.

a) Középső vagy jobb levél törlése kétkulcsú csúcsból

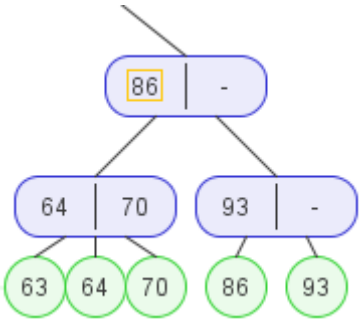
Amikor a törlendő levél szülőjének 3 gyermeke van, a levelet nyugodtan törölhetjük anélkül, hogy ezzel elrontanánk a fa bármely invariáns tulajdonságát. Ha a törlendő levél a szülőjének középső vagy jobb gyermeke, akkor a szülő tartalmazza a törölt levelével megegyező keresőkulcsot. Törlést követően a csúcsnak egy kulcsa és két levele marad, ezek megfelelő rendezése után készen vagyunk a törléssel.

b) Bal levél törlése kétkulcsú csúcsból

Az előző esethez képest a változás itt annyi, hogy a megfelelő értékű keresőkulcs nem a levél szülőjében van, hanem a fa valamelyik felsőbb szintjén. Az érintett levél törlése előtt ezt a keresőkulcsot az inorder bejárás szerinti rákövetkezőjével kell helyettesítenünk. A rákövetkező értékű keresőkulcs a törölt levél szülőjében található, onnan azt el kell távolítanunk. Így ismét előáll az a helyzet, hogy az utolsó belső csúcsnak csak egy keresőkulcsa és két levele van, ezeket megfelelően rendezve készen vagyunk a törléssel. (11-12. ábra)



11. ábra
85 törlése előtt



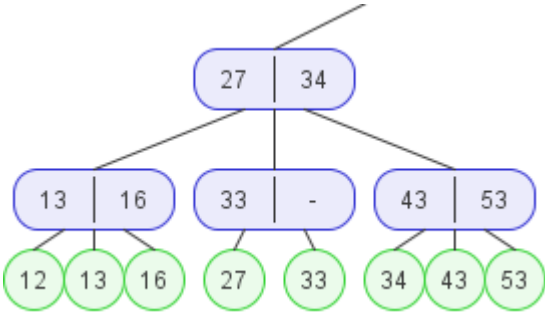
12. ábra
85 törlése után

c) Törlés egykulcsú csúcsból

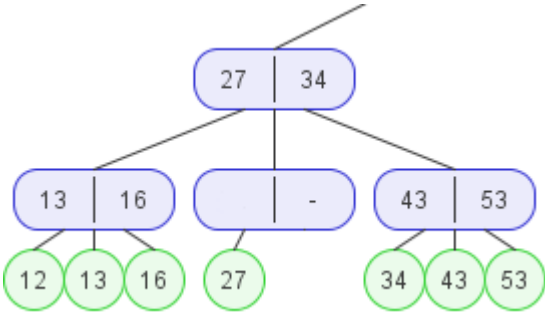
Az előző pontokhoz hasonlóan, ha a törölt kulcsú levél a szülője jobb gyermeke, akkor a szülője tartalmazza a megegyező értékű keresőkulcsot, ha bal gyermek, akkor valamelyik felsőbb csúcs. A levél és a keresőkulcs törlése után azonban sérül a fa invariáns tulajdonsága, hiszen a törölt levél szülőjének csak egy gyermeke maradt, és nincs már egyetlen keresőkulcsa sem. Az invariáns tulajdonság helyreállításának több módja van, melyek közül a törölt levelet tartalmazó, 2 magasságú részfa szerkezete alapján választhatjuk ki a megfelelőt. Jelölje a következőkben *n* azt a belső csúcsot, amely a levele törlése következtében sérti az invariánst.

d) Keresőkulcs áthelyezése testvérből

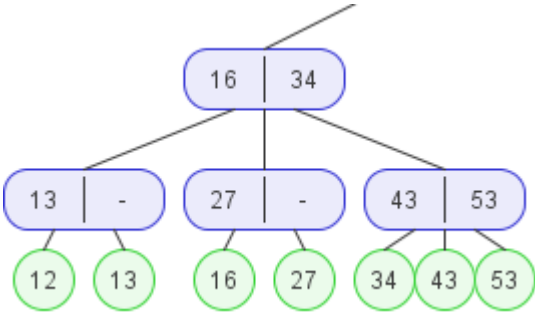
Ha n-nek van szomszédos két kulcsú testvére, akkor az az n-hez legközelebbi levelét közvetlenül átadja n-nek. Az n-hez közelebb eső keresőkulcsát szintén elveszíti, azonban nem azonnal n-be kerül át. Ha így lenne, akkor elveszne a keresőfa-tulajdonság, hiszen az átadott kulcs helye egyértelműen az, ahol eddig volt. A kulcsáthelyezés n-be valójában a szülőből történik, így a szülőben megüresedik egy hely. Ezt a helyet tölti fel a testvérből feljebb lépő kulcs. Az áthelyezések eredményeképpen két olyan csúcsot kaptunk, amelynek egy-egy kulcsa, és két-két levele van. A leveleket elhelyezve az értéküknek megfelelő pozícióba, a helyreállítással készen vagyunk. Az „adakozó” testvér lehet *n* baloldalán vagy jobb oldalán is. Sőt, ha *n* szülőjének 3 gyermeke van, melyek közül *n* a középső, akkor előfordulhat, hogy *n* mindkét oldali testvéréből lehetséges a kulcsáthelyezés. Ebben a helyzetben szabadon választhatjuk meg, hogy melyik csúcsot módosítjuk, de a választás eltérő szerkezetű fákat eredményez. (13. a-d. ábra)



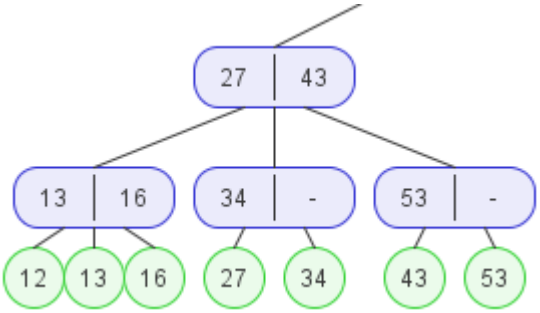
13a. ábra
33 törlése előtt



13b. ábra
33 törlése után, helyreállítás előtt



13c. ábra
Kulcsáthelyezés balról



13d. ábra
Kulcsáthelyezés jobbról

e) Csúcsösszevonás

- Ha *n*-nek egyik testvére sem kétkulcsú, de a szülője az, akkor a szülő egyik kulcsa lekerül *n* kulcsának helyére. Ezt követően végrehajtjuk a beszúrásnál látott csúcsvágással ellentétes műveletet, a csúcsösszevonást. Az *n* csúcsnak és (egyik) testvérének összesen két keresőkulcsa, és 3 gyermeke van, ami elegendő egy kétkulcsú csúcs létrehozásához. Az összevonás után az eddigi *n* szülőjének egy kulcsa és két gyermeke marad, vagyis az invariáns tulajdonságok továbbra is teljesülnek a fára. A csúcs, amivel *n* összevonásra kerül, lehet *n* bal vagy jobboldali gyermeke is, illetve az előző ponthoz hasonlóan az is előfordulhat, hogy választanunk kell a két testvér között.
- Ha *n* testvére, és a szülője sem kétkulcsú, akkor is a csúcsösszevonást kell alkalmaznunk. Az előző esethez hasonlóan *n* szülőjének a kulcsa kerül le *n*-be, majd *n* egyesül a testvérével. *(Megjegyzés: ha a törölt levél a szülőjének baloldali levele, akkor n szülőjének kulcsa a jelen fejezet b) és c) pontjában leírtaknak megfelelően felülíródhatott, mire a törlési folyamat ebbe a szakaszba*

jut) Ekkor ugyanaz a helyzet áll elő, mint a helyreállítás kezdetén, de már nem *n* az a csúcs, aminek csak egy gyermeke van, és nincs kulcsa, hanem *n* szülője. A csúcsösszevonás tehát, a csúcsvágáshoz hasonlóan felgyűrűzhet. A gyűrűzés addig folyatódik, míg a keresőút mentén előforduló kétkulcsú csúcsból fel nem tudjuk tölteni a hiányzó kulcs helyét. Ha nincs ilyen csúcs, akkor a törlés hatása elér egészen a fa gyökeréig.

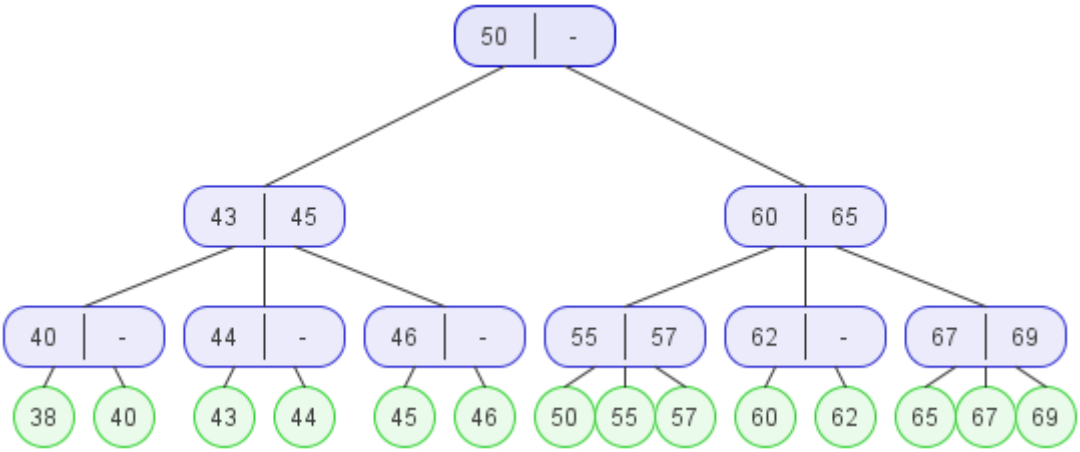
f) Törlés a gyökérből

Amikor a felgyűrűző törlési igény eléri a gyökeret, két dolog történhet: ha a gyökérnek ekkor még 3 gyermeke volt, akkor ugyanúgy viselkedik, mint bármely más kétkulcsú belső csúcs, elveszíti az egyik kulcsát, és az egyik gyermekét. Ha csak egy kulcsa van, akkor ez a kulcs lekerül az összevonásra kerülő gyermekébe, vagyis a gyökér kulcs nélkül marad, és egy gyermeke van. Ez az egy gyermek azonban az eddigi gyökér minden tulajdonságát teljesíti, vagyis kinevezhetjük őt gyökérnek, és az eddigi gyökeret törölhetjük a fából. Ily módon a fa magassága eggyel csökken.

g) Speciális törlések

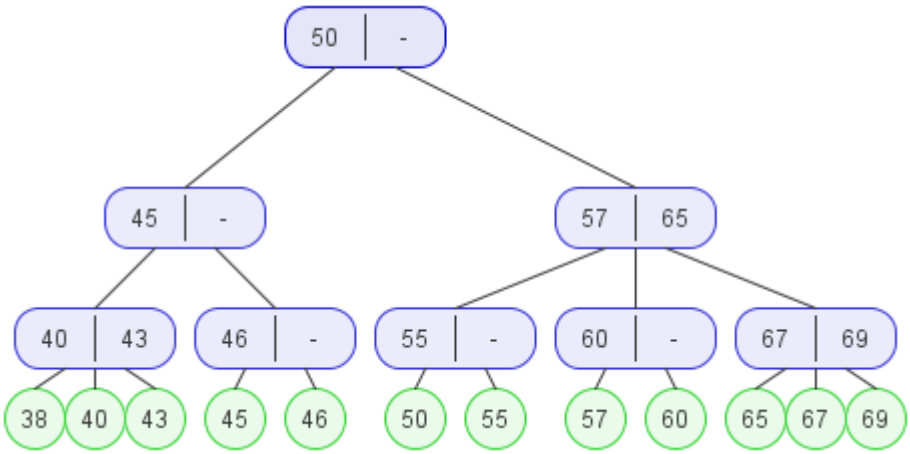
Két eset van, amikor a törlés nem az eddig leírtak alapján történik: az utolsó előtti, illetve az utolsó elem törlésekor. Amikor a fában már csak két elem van, azok a gyökér leveleiben vannak tárolva. Ha ezek közül a kisebbet töröljük, akkor a baloldali levél egyszerűen eltűnik. Ha a nagyobbat, akkor a jobboldali levél tűnik el, helyére átkerül az eddigi baloldali levél, és a gyökérben levő keresőkulcs felülíródik az új jobboldali levél kulcsával. Az utolsó elem törlésekor törlésre kerül a levél és a gyökér is, az egész fa *NIL*.

A beszúrásnál említettem, hogy a beszúrások sorrendje befolyásolja a kialakuló fa szerkezetét. Fokozottan igaz ez a törlésre, hiszen ekkor már nem csak a sorrend lehet befolyásoló tényező, hanem az is, hogy ahol erre lehetőség adódik, hogyan választjuk meg az összevonásra kerülő, illetve a kulcsáthelyezést elszenvedő csúcsokat. Tekintsük például a 14. ábrán látható fát.

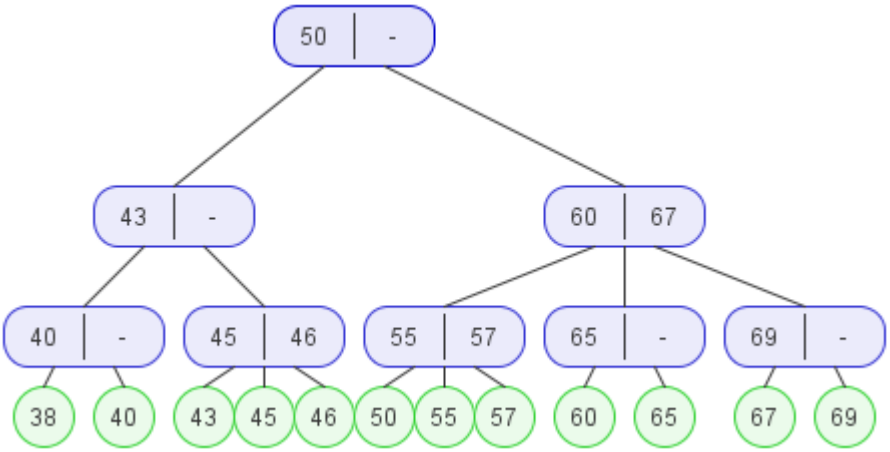


14. ábra
44 és *62* helye a fában, törlés előtt

A *44* törlésekor azt kell kiválasztanunk, hogy melyik testvérrel vonjuk össze a csúcsot, míg a *62* törlésekor azt, hogy melyik testvérből hozzunk át kulcsot. Ha mindkétszer a baloldali testvért választjuk, akkor a 15. ábrán látható fát kapjuk, míg ha mindkétszer a jobboldalit, akkor az eredmény a 16. ábrán látható.



15. ábra
Balkéz-szabály kétszeri alkalmazásának eredménye



16. ábra
Jobbkéz-szabály kétszeri alkalmazásának eredménye

2.4. A műveletek költsége

A 2-3-áris elágazási tényező alapján látható, hogy szintenként legfeljebb 2, azaz *h* magasságú fában legfeljebb *2h* összehasonlítással meghatározhatjuk egy adott értékű levél helyét, és még egy összehasonlítással eldönthetjük az ott talált levélről, hogy az a megfelelő kulcsot tartalmazza-e. $h \leq log_2 n$ összefüggést felhasználva felírható:

$T(n) \leq 2log_2 n + 1 = O(logn)$

ahol $T(n)$ az n elemet tartalmazó fában egy keresés költségét jelöli, $O(logn)$ pedig az $n \rightarrow 2log_2 n$ függvény aszimptotikus felső korlátja [7].

Beszúrás és törlés esetén ehhez a költséghez hozzáadódik ugyan a módosítások költsége, ám mindkét esetben csak a keresőút-menti csúcsokon, és azok közvetlen szomszédjain kell módosításokat végrehajtani, és ezeknek a száma mezőnként konstans. Azaz nagyságrendi változást e költségek nem jelentek, tehát az ***O(logn)*** továbbra is felső korlát.

3. Felhasználói dokumentáció

Ebben a fejezetben az alkalmazás futtatásához szükséges követelményekről, felhasználási lehetőségeiről és módjairól esik szó, valamint sor kerül a felhasználói felület részletes bemutatására.

3.1. A programról

A program célja, hogy segítse a felhasználót a 2-3-áris fák felépítésének és működésének megértésében, illetve betekintést nyújtson az egyes műveletek fizikai megvalósításába. Ezen kívül számos kényelmi funkcióval támogatja az elvégzett műveletek lefolyásának és eredményének akár többszöri elemzését.

3.1.1. Rendszerkövetelmény

a) Szoftverkövetelmény

- Windows XP vagy újabb operációs rendszer, vagy Linux operációs rendszer valamely, grafikus felülettel rendelkező disztribúciója
- Java Runtime Environment 1.6, vagy újabb verzió

b) Hardverkövetelmény

- legalább 1024x768 felbontásra alkalmas képernyő
- 512 MB memória

3.1.2. Telepítés

A program nem igényel telepítést, a szükséges rendszerkövetelmények megléte esetén, a CD mellékleten található TwoThreeTree.bat (Windows alatt) ill. TwoThreeTree.sh (Linux alatt) fájlra történő kattintással azonnal indítható.

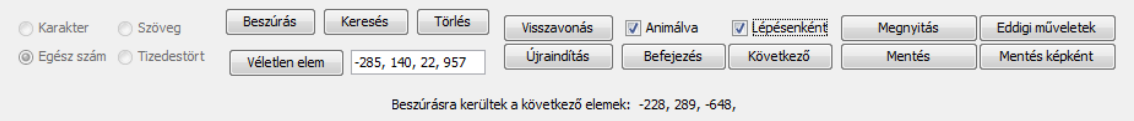
A Java Runtime Environment megfelelő verziója letölthető innen:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

3.2. Felhasználási útmutató

3.2.1. Programablak

A program egyablakos rendszerű. Az ablak függőlegesen ketté van osztva, a felső részben a vezérlőelemek láthatók (17. ábra), alatta pedig a fa aktuális állapotát reprezentáló grafika (18. ábra).



17. ábra
Kezelőfelület az alkalmazás egy tipikus állapotában

3.2.2. Vezérlőelemek működése

a) Kulcstípus kiválasztása

A 2-3-áris fa definíciója szerint a benne szereplő kulcsok bármilyen típus értékei közül kikerülhetnek, ha a típus felett rendezés értelmezett. E tulajdonságot demonstrálandó, az alkalmazás 4 különböző kulcstípust is kezel: karaktert, szöveget, egész és lebegőpontos számot. Indítás után a felhasználó első teendője egy típus kiválasztása, ameddig ez nem történik meg, minden más vezérlőelem inaktív marad. A kiválasztás a kezelőpanel bal szélén található rádiógomb-csoportban a megfelelő gombra kattintással történhet. Kattintásra a vezérlőelemek nagy része aktívvá válik, azonban másik kulcstípus kiválasztására is lehetőség van mindaddig, amíg a kiválasztott típusú fába az első beszúrás meg nem történik. A beszúrást követően a típusválasztó rádiógombok inaktívvá válnak, a kulcstípust csak az alkalmazás újraindítása után lehet megváltoztatni.

b) Fa manipulálása

A típusválasztó gomboktól jobbra elhelyezkedő újabb elemcsoport használatával lehetséges a fa tartalmának módosítása, illetve lekérdezése.

Kulcs megadása

A beszúrni, keresni, vagy törölni kívánt kulcsok megadására egy szöveges beviteli mező szolgál. A megadott kulcsnak meg kell felelnie a kiválasztott kulcstípusnak, ellenkező esetben az alkalmazás azt nem fogadja el érvényes kulcsként. Az egész számok

megadásának módja magától értetődő. A lebegőpontos szám tizedespontot használ (mivel a vessző elválasztóként funkcionál). Karakter típusú kulcs megadásakor nem szükséges azt idézőjelek vagy aposztrófok közé foglalni. Szöveg típusú kulcs esetén viszont szükség van az idézőjelekre, enélkül az alkalmazás a beírtakat nem fogja figyelembe venni. Karakter vagy szöveg típusú elemek összehasonlításakor az alkalmazás a Unicode karaktertáblát veszi alapul – természetesen a mező a kódtábla bármely elemét elfogadja.

Több kulcs egyidejű megadása

Ha az elvégezni kívánt művelet keresés, akkor a beviteli mező csak egy kulcsot fogad el. A másik két esetben azonban lehetőség van egyidejűleg több kulcs megadására. Ilyenkor a később kiválasztott művelet a megadás sorrendjében valamennyi (a megadott típusként értelmezhető) elemre meg fog történni. Ez a funkció lehetőséget ad arra, hogy ha a felhasználó egy konkrét fán szeretne valamit végrehajtani, akkor azt jóval gyorsabban, „felesleges" kattintások nélkül elő tudja állítani. Az elemek közti elválasztó karakter egész illetve lebegőpontos számok esetén vessző és/vagy szóköz lehet. Karakter típus esetén a vessző és a szóköz is elválasztó karakterként viselkedik, azonban egyik sem szükséges. A mező tartalmának feldolgozása karakterenként történik, és az előző kettő kivételével valamennyi karakter önálló kulcsként lesz értelmezve. Vessző és szóköz úgy adható meg kulcsként, hogy a beviteli mezőben // karakterek előzik meg őket. Ha a felhasználó egy / karaktert szeretne a fába szúrni, akkor ezt a /// karaktersorozat megadásával teheti meg. Szöveg típusú elemek megadása esetén sem szükséges elválasztó karakterek használata, az idézőjelek ellátják ezt a funkciót. (Illetve minden olyan karakter, ami nem idézőjelek között szerepel, elválasztó karakterként fog viselkedni.)

Véletlen elem gomb

A beviteli mező mellett található *Véletlen elem* gombbal a felhasználó a kiválasztott típusnak megfelelő véletlen elemeket adhat hozzá a listához, vagy akár csak ennek a gombnak a használatával is felépíthet tetszőleges elemszámú fát. Típustól függően a gomb által generált kulcsok:

- -1000 és +1000 közötti egész számok
- 0 és 100 közötti, egy tizedesjegy pontosságú lebegőpontos számok

- a görög ábécé betűi
- a latin ábécé betűiből, számjegyekből, és néhány speciális karakterből álló, 2-4 karakter hosszúságú karakterláncok,

amelyeket a felhasználó tetszés szerint kombinálhat az általa megadott kulcsokkal.

Beszúrás

A gomb megnyomásakor a program megkísérli a beviteli mezőben található értékeket a megfelelő típusú kulcsként értelmezni, és beilleszteni a fába, illetve figyelmeztetést, hibaüzenetet jelenít meg, ha:

- a mező üres volt
- a mezőben szereplő valamelyik értéket nem sikerült a kiválasztott típusú elemként értelmezni
- a mezőben szereplő értékek valamelyike korábban már beszúrásra került

A fa új állapota azonnal láthatóvá válik a kezelőpanel alatti területen, a sikeresen beszúrt elemek pedig megjelenítésre kerülnek a státuszinformációban.

Keresés

A gomb megnyomásakor a program megkeresi a fában a kapott kulcsú levelet, a keresés eredményéről a státuszinformációban tájékoztat, illetve figyelmeztetést, hibaüzenetet jelenít meg, ha:

- a mező üres volt
- a mezőben egynél több kulcsot talált
- a mezőben szereplő értéket nem sikerült az adott típusú elemként értelmeznie

Ha a keresés sikeres volt, a fa grafikus reprezentációjában kiemeli a talált levélhez vezető keresőutat.

Törlés

A gomb megnyomásakor a program megkísérli a beviteli mezőben található értékeket a megfelelő típusú kulcsként értelmezni, és törölni a fából, illetve figyelmeztetést, hibaüzenetet jelenít meg, ha

- a mező üres volt
- a mezőben szereplő valamelyik értéket nem sikerült a kiválasztott típusú elemként értelmezni
- a mezőben szereplő értékek valamelyike nem volt a fában

Ha egy elem törlése során több különböző szerkezetű fa is előállhat, arról felugró ablakban kap tájékoztatást a felhasználó (*lásd a „Bal- és jobbkéz-szabály c. pontban”*).

A fa új állapota azonnal láthatóvá válik a kezelőpanel alatti területen, a sikeresen törölt elemek pedig megjelenítésre kerülnek a státuszinformációban.

„Bal- és jobbkéz-szabály”

Bizonyos kulcsok törlése közben előfordulhat, hogy egy belső csúcsba annak bal vagy jobb testvéréből, illetve testvéréből vagy szülőjéből is áthelyezhető kulcs. Ekkor megjelenik egy párbeszédpanel, melyen a felhasználónak ki kell választania, melyik szabályt kívánja alkalmazni. Fontos megjegyezni, hogy ez a kérdés a Törlés gomb minden megnyomásakor legfeljebb egyszer jelenik meg. Ha a felhasználó egyszerre több olyan kulcsot töröl, amely rendelkezik a fenti tulajdonsággal, akkor minden kulcsra a választott szabály kerül alkalmazásra. Amennyiben a felhasználó pillanatnyi igényeinek ez nem megfelelő, úgy a megoldás a kulcsok egyenként történő törlése.

Művelet visszavonása

Ha a felhasználó egy művelet eredményeképpen nem azt a fát kapta, amire számított, rossz helyre kattintott, vagy nem emlékszik a kiinduló állapotra, a visszavonás gombbal kirajzoltathatja a fa legutóbbi műveletet megelőző állapotát. Ha ez a művelet több kulcs egyidejű beszúrása vagy törlése volt, akkor minden kulcs eltűnik, illetve visszakerül a fába. Egy alkalommal csak egy művelet hatása vonható vissza.

Újraindítás

A program indítását követően bármikor lehetséges azt újraindítani. Ekkor újra kiválasztható az elemtípus, viszont a fán végzett minden eddigi tevékenység – a fával együtt – visszavonhatatlanul törlésre kerül. Az újraindítást követően az alkalmazás állapota és viselkedése megegyezik az indítás utáni állapottal.

c) Megjelenítési lehetőségek

A következő gombcsoport elemeivel a fa megjelenítésének módja szabályozható.

Animálás

Az *Animálás* jelölőnégyzet bejelölésével a beszúrás és törlés műveleteknek nem csak az eredménye jelenik meg, hanem a két állapot közti átmenet részletei is. Ez különösen akkor lehet fontos, ha a művelet végrehajtása közben csúcs-vágásra, csúcs-összevonásra kerül sor, ami a végeredmény megértését lényegesen megnehezítheti. A további műveletek animálva fognak megjelenni mindaddig, amíg a felhasználó ki nem kapcsolja a funkciót, vagy újra nem indítja a programot.

Az animáció során a következő elemi lépések láthatóak:

- beszúrás esetén minden más lépés előtt, törlés esetén minden más lépés után az érintett csúcsok elfoglalják új helyüket
- új csúcsok illetve mutatók megjelenése (létrehozása)
- feleslegessé vált csúcsok illetve mutatók eltűnése (törlése)
- kulcs-áthelyezések
- mutató-áthelyezések

Animált megjelenítéskor a beszúrt, illetve törölt elemhez (vagy az elem helyéhez) vezető keresőút is kiemelve látható.

Több művelet egyidejű animálása

Animált megjelenítés esetén is lehetséges egyszerre több kulcs beszúrása, vagy törlése. Azonban a csúcsok áthelyezése ekkor is egyszerre történik meg. Sok kulccsal kezdeményezett művelet animálása közben meglehetősen szokatlan alakú – bár a megkövetelt tulajdonságokat kielégítő – fák jelenhetnek meg. Ezért a több kulccsal történő animálás csak a fák viselkedésében jártas felhasználóknak ajánlott.

Animálás lépésenként

Az animált megjelenítés során az átmenetek egyes lépései automatikusan jelennek meg, rövid szünetekkel. Valószínű, hogy a felhasználók többsége nem tudja ilyen gyorsan követni az eseményeket, ezért lehetőség van azok egyenként történő megjelenítésére a *Lépésenként* jelölőnégyzet kiválasztásával. A helyet változtató csúcsok továbbra is egy lépésben jutnak el az új helyükre, azonban minden egyéb változás előtt az animáció megáll, és csak akkor kerül sor a következő lépés megjelenítésére, ha a felhasználó megnyomja a *Lépésenként* jelölőnégyzet alatt található *Következő* gombot. Az animáció futása közben a jelölőnégyzet állapotának megváltoztatása már nincs hatással a megjelenítés módjára.

Animáció befejezése

Különösen sok kulcs esetén az animáció viszonylag hosszú ideig tarthat, ám a *Befejezés* gomb megnyomásával bármikor megszakítható. Ekkor a fának az elvégzett műveletek utáni állapota lesz látható. A *Befejezés* gomb használható akkor is, ha az animáció nem fut le végig (pl. elrejtett részfán történő animáció esetén).

d) Adminisztrációs műveletek

Mentés fájlba / Beolvasás fájlból

Későbbi felhasználás céljából a fa aktuális állapotát reprezentáló fájl készíthető a *Mentés* gombbal. A keletkező `23t` típusú fájl a programnak szóló információkat tartalmaz, melyek alapján a fa egy beolvasást követően a jelenlegi állapotába visszaállítható. A beolvasás a *Megnyitás* gombbal történhet. A *Megnyitás* gomb közvetlenül a program elindítása után is használható, anélkül, hogy kulcstípust választanánk. A beolvasott fájl „tudja”, hogy milyen típusú fát tárol, és beolvasás közben automatikusan kiválasztja azt. Figyelem! Ha a beolvasást megelőzően a felhasználó már létrehozott egy fát, akkor az a beolvasással visszavonhatatlanul eltávolításra kerül.

Mentés képként

Ezzel a gombbal a fa aktuális állapotát reprezentáló ábra menthető `png` formátumban.

Eddigi műveletek

A gomb hatására a program egy felugró ablakban kilistázza, hogy milyen változások történtek a fában annak létrehozása óta. Ez a lista tulajdonképpen a mentésre kerülő fájl tartalma, áttekinthetőbb formában. Fájlból történő beolvasáskor ez a lista is elkészül az adott fához, így láthatóvá válik azon műveletek sorozata, melyek eredményeképpen a kirajzolt fa kialakult.

e) Állapotjelző üzenet

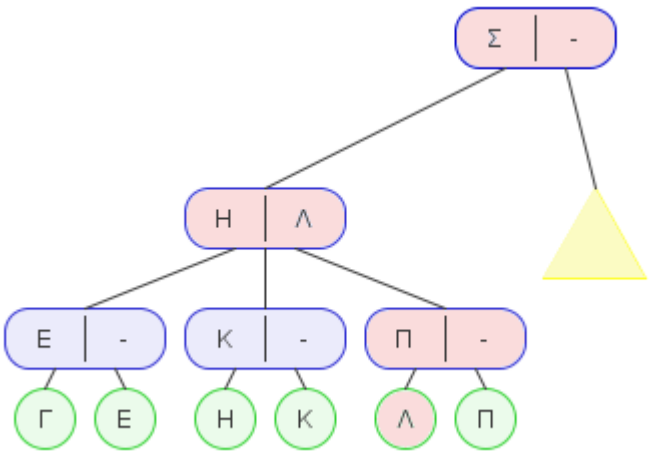
A fentebb részletezett vezérlőelemek alatt, de még a fát ábrázoló grafikus panel felett megjelenhet egy egysoros szöveg, mely a legutóbbi művelet eredményéről, illetve az esetleges további teendőkről tájékoztatja a felhasználót.

3.2.3. Grafika

a) A fa elemei és jelentésük

A fa belső csúcsait kék színű, lekerekített sarkú téglalapok, a leveleit pedig zöld színű körök jelölik.

A levelekben és belső csúcsokban szereplő szöveg a levél illetve belső csúcs kulcsait tartalmazza. A belső csúcsok közepén levő függőleges vonal a csúcs bal- és jobboldali kulcsának elválasztására szolgál. Ha egy belső csúcs valamelyik kulcsa hiányzik, azt egy '-' karakter jelöli. A fa természetéből adódóan a hiányzó kulcs legtöbbször a jobboldali, ám animáció közben esetenként látható olyan átmeneti állapot, amikor a csúcsnak nincs baloldali kulcsa. (18. ábra)



18. ábra: Belső csúcsok, levelek, keresőút, rejtett részfa

Ha a kulcs szövege hosszabb, mint ami a neki szánt területen megjeleníthető, akkor az alkalmazás csonkolja azt. A teljes kulcsok mindig megjeleníthetők az egér megfelelő csúcs fölé húzásával.

A csúcsokat összekötő egyenes szakaszok – a fa konzisztens állapotában - a szülő-gyermek kapcsolatokat reprezentálják. Animáció közben gyakran fellép olyan átmeneti állapot, amikor a fenti megállapítás nem igaz, előfordulhat, hogy egy csúcs több másik csúcsnak is gyermeke, vagy ugyanannak a csúcsnak többször is gyermeke.

b) Keresőút megjelenítése

A keresés művelet eredményeképpen létrejön egy keresőút. Ha a keresett csúcs megtalálható a fában, akkor a keresőút a megfelelő levelet is tartalmazza, ellenkező esetben abban az utolsó belső csúcsban ér véget, amelynek a keresett kulcsú levél a gyermeke lenne. A keresőút kiemelésre kerül abban az esetben, ha a felhasználó a keresés műveletet hívta meg, illetve ha a legutóbbi beszúrás vagy törlés animálva került megjelenítésre. Utóbbi két esetben a keresőút az utolsóként módosított levélhez vezet. A kiemelés piros színnel történik.

c) „Széles” fák megjelenítése

Az alkalmazás használatával elméletileg tetszőleges elemszámú fa felépíthető és megjeleníthető. Ám a fák szélessége az elemek számának növekedésével igen gyorsan nő. A megjelenítés tervezésekor törekedtem arra, hogy az aktuálisan megjelenített fa a lehető legkeskenyebb, így amíg ez lehetséges, egyben látható legyen. Minden igyekezet ellenére azonban már akár a negyedik szint szélessége is nagyobb lehet annál, mint ami egy átlagos méretű képernyőre kifér. További nehézséget okoznak a „nagyon ferde” fák, vagyis azok, amelyek kiterjedése a gyökértől balra illetve jobbra, a változó elágazási tényező miatt nagyon különböző.

A képernyő szélességét meghaladó fák esetén, animálás nélküli beszúráskor és törléskor a vízszintes görgetősáv középre igazodik, míg animált megjelenítéskor a(z utolsóként) módosított levél felé tolódik. Természetesen a fa magassága is meghaladhatja a megjeleníthetőség határait, amely esetben függőleges görgetősáv használatára is szükség van, de ehhez általában egy többszörös elemszámú fára van szükség, ami egy szemléltető célú program esetében kevésbé jellemző.

d) Részfák elrejtése és megjelenítése

Tovább csökkentendő a grafikus panel szélességét, az egyes csúcsok alá tartozó részfák elrejthetőek. Ez akkor lehet hasznos, ha egy művelet elvégzésekor sejtjük, hogy egy részfa nem fog változni, így annak elrejtésével nagyobb figyelmet (és képernyőterületet) fordíthatunk a megváltozó részfára. Egy részfa elrejtése a gyökerén történő kattintással érhető el. Ettől kezdve a részfát a gyökere helyén megjelenő sárga háromszög helyettesíti. A részfa újra megjeleníthető, ha a háromszög felső csúcsára kattintunk az egér jobb gombjával. Az elrejtett részfában is végrehajtódnak a szokásos műveletek, melyek eredménye a részfa újbóli megjelenítésekor láthatóvá válik. Pl. ha egy olyan levélre indítunk keresést, amely elrejtett részfa eleme, akkor a keresőút megjelenik a háromszöggel bezárólag, megjelenítés után pedig végig.

e) Animáció elrejtett részfán

Történhet a megjelenítés animált módban egy olyan fán, amelynek éppen van elrejtett részfája. Ha a változást szenvedő csúcsok egyike sem eleme rejtett részfának, akkor az animáció ez eddig leírtakhoz hasonlóan történik. Ha az animáció mégis ilyen csúcsot érint, akkor a hatása nem lenne látható a felhasználó számára. Ilyen esetben az állapotátmenet lépései pontosan addig kerülnek kirajzolásra, amíg nem érintenek elrejtett csúcsot. Ezt követően az animáció figyelmeztetéssel leáll, és a fa látszólag inkonzisztens állapotban marad. A végeredmény megjelenítéséhez az elrejtett részfa a szokásos módon megjeleníthető, illetve a *Befejezés* gomb megnyomására minden, eddig elrejtett részfa újra megjelenik.

4. Fejlesztői dokumentáció

A megoldandó feladat egyrészt egy olyan adatszerkezet megvalósítása, mely viselkedésében megegyezik a 2. fejezetben tárgyaltakkal, másrészt ennek interaktív megjelenítése, a felhasználó által könnyen áttekinthető formában, a lehető legkevesebb megszorítással élve a kezelt adatok tartalmát és mennyiségét illetően.

4.1. A választott technológia

A technológia (programozási nyelv) kiválasztásakor a következő szempontokat vettem figyelembe:

- a megjelenítéshez erős grafikus támogatásra van szükség
- lehetőleg platform-független
- integrált fejlesztőkörnyezet érhető el hozzá (melynek szolgáltatásaira támaszkodni szeretnék, pl. statikus kódelemzés, refaktorálási lehetőségek, kódkiegészítés)
- ingyenes
- tanulmányaim során már találkoztam vele
- elterjedt, így váratlan nehézség esetén nagy mennyiségű szakirodalom és online tartalom lehet a segítségemre

Ezek alapján a Java nyelvet találtam leginkább megfelelőnek, a fejlesztőkörnyezet az Eclipse Helios lett. A fejlesztés során használt JDK verzió 1.6, bár a tesztelés során lefordult, és hibamentesen futott a program 1.5-ös verzióval is.

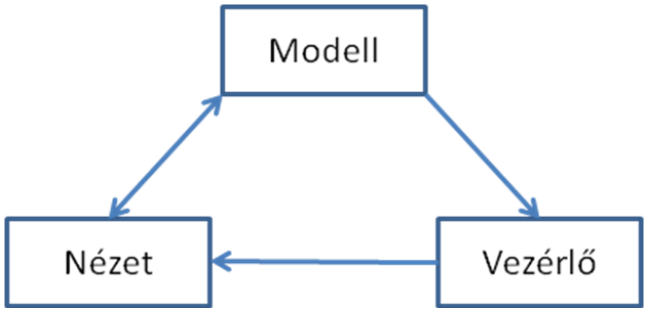
4.2. A tervezés során felmerült igények, problémák és megoldásaik

A felhasználói dokumentáció tartalma értelmezhető követelmény-specifikációként is, és mint ilyen, sok tekintetben befolyásolta a tervezési folyamatot. Az alábbiakban kitérek néhány „problémára”, melyre a tervezés során tekintettel kellett lennem, és röviden vázolom ezek megoldását is.

4.2.1. Tervezési minta

A program célja, hogy megismertesse a felhasználót a 2-3-áris fák viselkedésével. Ehhez elengedhetetlennek látszik egy ilyen fa egyes állapotainak valamilyen vizuális ábrázolása, illetve az egyes állapotátmenetek interaktív megjelenítése. E megjelenítés legcélravezetőbb módja sok tényezőtől függhet, például a felhasználók egyéni igényeitől, vagy a megjelenítésre használt eszköz felhasználási lehetőségeitől. Előnyös tulajdonság tehát, ha a megjelenítés módja viszonylag rugalmas, könnyen változtatható, továbbfejleszthető. Azonban bármilyen módját választjuk is a vizuális ábrázolásnak, az adatszerkezet felépítésének és műveletek fizikai megvalósításának változatlannak kell maradniuk. Az adatokat és a megjelenítést tehát, amennyire csak lehetséges, elkülönítve kell kezelnünk. E megközelítést támogató szerkezeti minta a Modell-Nézet-Vezérlő (Model-View-Controler)[8] minta (19. ábra), mely az elkülönítést 3 fő komponens segítségével valósítja meg:

modell: tartalmazza az adatokat, és képes azokkal műveleteket elvégezni,
nézet: megjeleníti a modellt a felhasználó számára értelmezhető formában,
vezérlő: kezeli a felhasználótól érkező bemenetet, általában valamilyen esemény alapú mechanizmussal, és a kérés végrehajtásához a nézet és a modell szolgáltatásait is használja. (19. ábra)



19. ábra
A Modell-Nézet-Vezérlő architekturális minta vázlata

Alkalmazva a mintát a megoldandó feladatra, a komponensekkel szemben támasztott követelmények az alábbiak szerint alakulnak:

- **modell:**
    - tárolja a fa egy állapotát
    - képes a fán elvégezhető műveletek előfeltételének ellenőrzésére
    - képes a műveletek végrehajtására
    - információt ad a végrehajtott műveletek sikerességéről
    - információt ad a fa aktuális állapotáról

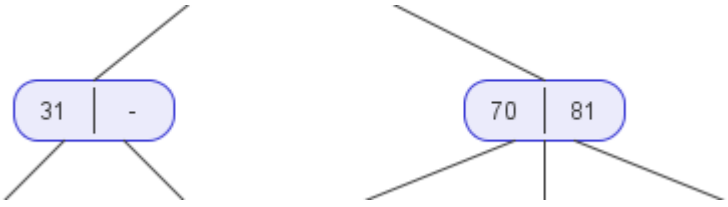
- **nézet:** jelenítse meg a fa egy vizuális reprezentációját, valamint a fa manipulálására alkalmas vezérlőelemeket.

- **vezérlő:** a felhasználói akciók nyomán műveletek vagy lekérdezések végrehajtását kezdeményezi a fán, a kapott adatokból pedig létrehozza a fa megjelenítésre alkalmas reprezentációját, amit aztán továbbad a nézetnek.

4.2.2. Csúcsok fizikai szerkezete

A csúcsoknak meg kell tudniuk valósítani a 2.2 pontban, a „gyenge” 2-3-áris fa definíciójában megfogalmazott követelményeket, melyhez a következőkre van szükség:

- a csúcs ismeri a szülőjét → minden csúcs rendelkezik egy mutatóval a szülőjére, a gyökér esetében a mutató null
- minden belső csúcs rendelkezik 1 vagy 2 keresőkulccsal, minden levél rendelkezik pontosan 1 kulccsal → minden csúcs képes két kulcs tárolására, levél esetében a jobboldali kulcs soha nem kap értéket, ám ennek biztosítása a fa feladata lesz
- minden belső csúcs ismeri a baloldali, jobboldali, és a középső gyermekét, ha az létezik → a csúcsnak rendelkeznie kell 3 mutatóval a gyermek csúcsokra. Ha az adott gyermek nem létezik, akkor a mutatója null, ennek kezelése a fa feladata lesz. (20.ábra)



20. ábra
Belső csúcsok fizikai szerkezete

4.2.3. „Hosszú” kulcsok megjelenítése

Adott egy 2-3-áris fa, melynek csúcsaiban tetszőleges típusú, így tetszőleges hosszúságú kulcsok tárolhatók. Mivel a fa szélessége várhatóan igen gyorsan nő, a megjelenített csúcsoknak *viszonylag* kicsiknek kell lenniük, és *viszonylag* sűrűn kell elhelyezkedniük. Ugyanakkor a kulcsértéket tartalmazó szövegnek könnyen olvashatónak kell maradnia, így annak mérete nem mehet egy bizonyos határ alá.

Ötlet

Szemléltető programról lévén szó, elképzelhető a kulcsok hosszúságának korlátozása, még akkor is, ha ez nem túl elegáns megoldása a problémának.

Ahhoz, hogy a kirajzolt fa kényelmesen használható legyen, legalább 3 belső szintet ki kell tudni rajzolni úgy, hogy az még görgetés nélkül látható maradjon. Egy teljesen kitöltött 3-áris fa esetén ez azt jelenti, hogy 9 egymás melletti belső csúcsot kell megjeleníteni, melyek között valamekkora hézagnak is maradnia kell. A vezérlő elemeket tartalmazó panel várható szélessége – a futtató környezettől függően – 800-900 képpont, amely meghatároz egy minimális ablak-szélességet. Így a belső csúcsok szélessége kb. 80 képpont lehet, melyben két kulcsnak kell megjelennie. Egy kulcsra tehát maximum 34-36 képpont jut, amire 10 pontos karakterméret esetén, átlagos szélességű karaktereket feltételezve 4 darab fér ki. Egy ilyen korlátozás bevezetésével pl. a *-10.5* kulcs már nem jeleníthető meg. Ez, figyelembe véve, hogy a modell generikusan lett megvalósítva, ami igen „látványos” kulcsok használatát is lehetővé tenné, drámai megszorítás lenne.

Megoldás

Az 5* karakter hosszúságú kulcsok még megjelenítésre kerülnek, az ennél hosszabbakat azonban az alkalmazás a 4*. karakter után csonkolja, és ezt egy „...” karaktersorozattal jelzi. A csonkolás a `Graph` csomag *(lásd 4.3.2.)* `LogicalNode` osztályának konstruktorában történik. A csúcs természetesen ismeri a teljes kulcsait is, amit a csúcshoz tartozó tooltip-ben jeleníthet meg a felhasználó. A tooltip hozzárendelését a rajzolásért felelős `GraphicPanel` osztály végzi, a megjelenítés pedig a `MouseMotionListener` osztály segítségével történik minden grafikus csúcson.

**tesztelés közben szerzett tapasztalatok alapján a karakterszám végül 4-re, ill. 3-ra csökkent*

4.2.4. A „ritka” 2-3-áris fa esete

Adott egy 2-3-áris fa, amelyet áttekinthető módon meg szeretnénk jeleníteni a grafikus felületen. A csúcsok mérete már véglegesnek tekinthető *(lásd 4.2.1)*, most a pozíciójukat szeretnénk meghatározni. Az elsődleges feladat, hogy a fa minden csúcsát legyen hova megrajzolni, amelyre a legegyszerűbb megoldásnak az tűnik, ha minden csúcs alatt 3 gyermeknek hagyunk helyet. Azonban ez a módszer, túl azon, hogy nem várható tőle a fa esztétikus, átlátható megjelenése, nem is túlzottan hatékony. Egy „véletlenszerű” fa *sok* csúcsának csak két részfája lesz, míg a harmadik részfa helyét meghagyva nagy felület maradhat kihasználatlanul. Szélsőséges esetben az üres terület olyan nagy lehet, hogy lefed egy egész képernyőnyi területet, azaz, bár a fa létezik, és meg van jelenítve, valamely részfa közepére állva a felhasználó nem lát belőle semmit.

Ötlet

Minden beszúrás és törlés után az egész fát újrarajzoljuk, hiszen az érintett csúcson kívül további, előre nem látható változások is történhetnek a benne. Az újrarajzolás alkalmával minden csúcs pozícióját is újradefiniálhatnánk, ha tudnánk, milyen szélesek az alattuk levő részfák.

Megoldás

A művelet végrehajtását követően egy rekurzív fabejárással minden csúcs minden részfájáról megmondhatjuk, hogy a fa új állapotában mekkora a vízszintes kiterjedése. Ezt a feladatot a `tree` csomag *(lásd 4.3.1)* `GraphicSupport` osztálya látja el. Egy (rész)fa szélességét a benne szereplő levelek, vagy utolsó belső csúcsok száma határozza meg. A visszaadott értékekből a rajzolandó csúcsot előállító osztály – a `NodeListTransformator` – meg tudja határozni a csúcs koordinátáit úgy, hogy minden részfának elegendő hely jusson.

Megoldás javítása

A fentiekben vázolt eljárás nagy előrelépést jelentett a helykihasználás terén, ám rövid tesztelés után kiderült, hogy finomításra szorul. A részfában levő levelek száma helyesen határozza meg a részfa szélességét. Viszont ennek a szélességnek változó hányada foglal valóban helyet más részfa elől. Ha aszimmetrikus részfák kerültek egymás mellé, akkor ezek a szélesebb oldalukon belelóghattak a szomszédos részfába,

míg a kevésbé széles oldalukon felesleges hely maradt. A javításhoz szükségessé vált egy új változó kiszámítása, ez pedig egy részfa kiterjedése a szülő középvonalától valamelyik irányban. Pl. egy jobb oldali, aszimmetrikus részfa gyökerének helyét két tényező határozza meg: a részfa „baloldali kiterjedése”, és a szomszédos középső részfa „jobboldali kiterjedése”. Ha a részfa gyökere pontosan e két tényező összegével megegyező távolságra van a szülőjétől, akkor levelei nem érnek össze más részfa leveleivel, a helykihasználása pedig optimális. E kiterjedés kiszámítása is a `GraphicSupport` feladata lett. Természetesen a visszaadott értékek relatívak, a kiterjedést az adott területen ábrázolandó csúcsok számával reprezentálják, amikből a tényleges pozíció kiszámítása a csúcsok és hézagok méreteinek ismeretében, később kerül sor.

4.2.5. Elrejthető részfák

Az előzőekben felvázolt problémák megoldása olyan grafikát eredményezett, amely a rendelkezésére bocsátott képernyőterületet kellő hatékonysággal használta ki. Az animált megjelenítés módja ekkor még nem került átgondolásra, de sejthető volt, hogy ha a fa olyan nagy, hogy meghaladja a képernyő méretét, akkor az animáció a szélességhez képest csak egy kis sávot fog érinteni. A nem érintett részfák megjelenítése ekkor felesleges, hiszen nem történik bennük változás, és a felhasználó kiemelt figyelmére sem tarthatnak számot. Szerettem volna lehetőséget biztosítani a felhasználóknak arra, hogy ha egy részfát kevésbé tartanak érdekesnek, akkor azt elrejthessék, hogy az ne vonja el a figyelmüket, és ne foglaljon felesleges képernyőterületet.

Megoldás

Felhasználói szempontból a legkényelmesebb megoldásnak az tűnt, ha egy csúcsra kattintva a vele kezdődő részfa eltűnik, újra rákattintva újra megjelenik. Ehhez a GraphicPanel osztályban már jelen levő `MouseMotionListener` mellé egy `MouseListener` interfészt is meg kellett valósítani a kattintás kezelésére. Bizonyos számú csúcs felett a két listener valamilyen okból „összeakadt”, ami `StackOverFlowError`-t váltott ki. Ezután létrehoztam a `MouseInputAdapter`-ből származtatott `OnNodeMouseListener` osztályt, mely a szükséges metódusok felüldefiniálásával ellátja az említett két listener feladatait. *(Sajnos, amint az a teszt-*

eredményeknél látható lesz, ezzel nem oldódott meg a probléma, lásd 4.4.3)

Az elrejtett részfa ugyan nem jelenik meg, ennek ellenére változatlanul szerepel a modellben, és a vezérlő réteg csúcsai között is szerepelnie kell. A rajzolásban részt vevő valamennyi objektummal tudatni kell, hogy hogyan kezelje ezt a csúcsot. Pl. a `GraphicSupport` példányának nem kell figyelembe vennie a részfa méretét, a `GraphicPanel`-nek másképpen kell megrajzolnia a csúcsot, és nem szabad megrajzolnia a gyermekeit, más hatással lehet rá az animáció, stb. Az összezárt csúcsok azonosítóinak tárolásához egy egyke mintát megvalósító osztályt választottam, amely globális láthatóságú, így a rajzolás minden résztvevője le tudja kérdezni rajta keresztül egy adott csúcsról, hogy nem-e elrejtett részfa eleme.

4.2.6. Generikus fa, a Type Erasure hatása

A szemléltető célból ábrázolt adatszerkezetek általában leegyszerűsítik a tárolt adatok tulajdonságait, pl. csak egész számoknak egy intervallumát kezelik. Szerettem volna az effajta megszorításokat elkerülni, annak érdekében, hogy a modellem közelebb álljon egy „valódi”, gyakorlatban is használható adatszerkezethez. A modellben szereplő fa teendői minden elemtípus esetén ugyanazok lesznek, így célszerű azt generikus osztállyá tenni, így biztosítva az újrafelhasználhatóságot. A generikussá alakítás minimális változást jelent a modellen a rögzített típushoz képest, de annál összetettebb feladat a vezérlő rétegben.

A modell tehát generikus, többféle elemtípussal példányosítható. Egy bizonyos típusú fához további osztályoknak is kell tartozniuk, amelyek felelősek a felhasználó által megadott bemenet ellenőrzéséért, illetve a véletlen input generálásért. A Java Type Erasure következményeként egy generikus típust használó objektum típusváltozója fordítás után már nem kérdezhető le, és nem módosítható[9]. Ez két dolgot von maga után. Az egyik, hogy a modellt, és az említett osztályokat mindig újra kell példányosítanunk, mielőtt más típussal szeretnénk őket használni. A másik, hogy e példányok létrejöttekor a felhasználói felületnek már futnia kell – hiszen a felhasználó ott választja meg a típust -, tehát ezeknek szigorúan el kell különülniük a felülettől. Ezen tulajdonságok megvalósítása a `TreeInstance` osztály feladata lett. Ez az osztály is generikus, és mielőtt a felhasználó bármi egyebet tehetne az alkalmazással, először ezt kell példányosítania. A kapott típusparaméter alapján már létre tudja hozni a megfelelő típusú fát, és az ugyanilyen típusú elemző- és véletlen elemeket generáló

objektumokat.

A `parser` csomagban *(lásd 4.3.3)* definiálva van egy-egy interfész az utóbbi két objektumfajtához (`RandomGenerator` ill. `InputParser`). Ugyanitt vannak azok az osztályok, melyek típusonként eltérő módon megvalósítják ezeket az interfészeket. Szerepel még egy generikus interfész (`NodeTypeFactory`), és az azt megvalósító osztályok. Ez az interfész írja le, hogy amikor a felhasználó kiválaszt egy típust, a modell-, a véletlen-generátor- és az elemző osztálynak a megfelelő típussal létrejöjjenek a példányai.

4.2.7. Animációs lépések azonosítása, a csúcsok mozgatása

Az animáció során láthatóvá válnak az egyes műveletek megvalósításának elemi lépései. Mivel a megvalósítás részleteit csak a modell ismeri, az elvégzett munkát is ő dokumentálja, és később felhasználható formában továbbadja. Egy lépés tárolásának eszköze az `OperationLoggerElement` típus, melynek mezői tartalmazzák a lépés által érintett csúcsok azonosítóit, az új kulcsértéket, és egy lépésazonosítót. Lépések sorozatát egy ilyen objektumokból álló lista tárolja. A lista generálása nehézkes, ha nem a tényleges végrehajtással egy időben történik, hiszen minden megváltozott elem régi tulajdonságait meg kellene őriznünk ahhoz, hogy animáláskor az eredeti állapotot is láthatóvá tehessük.

A modell `TwoThreeTree` osztálya a fenti megfontolások alapján nem csak végrehajtja a műveletet a fán, hanem tárolja az elvégzett lépéseket, így a műveleteket megvalósító metódusok a lépéslistát is visszaadó függvényekké válnak.

Az animáció másik feladata a csúcsok új helyre mozgatása. A mozgatásnak folyamatosan kell történnie, és a kiinduló és végállapotot is tartalmaznia. Mivel a csúcsok pozícióját a modell nem tartalmazza, az átmeneti állapotok megalkotása kizárólag a vezérlő feladata. A mozgatás kezdőállapotában a „régi” fa összes csúcsa szerepel a „régi” pozíciójában, de a csúcsok már ismerik az új pozíciójukat. A köztes állapotok meghatározása a régi és az új pozíciók lépésenkénti közelítésével történik. A mozgatás és a lépéslista elemeinek végrehajtása az `AnimationControler` osztály feladata.

4.2.8. Több áthelyezhető kulcs kezelése

Szintén a modell viselkedését befolyásolja, ha egy kulcs törlése közben az *Elméleti háttér* c. fejezet *Csúcsösszevonás (lásd 2.2.3.e)* pontjában ismertetett helyzet alakul ki, amikor a felhasználónak választania kell az elmozdítható kulcsok közül. Amíg ilyen, „elágazást” okozó kulcsot nem akarunk törölni, a modell teendője egyértelmű. Amikor igen, akkor a felhasználó visszajelzéséig nem szabad végrehajtania a törlést. A megoldás az, hogy a törlésben részt vevő valamennyi metódus továbbítson egy paramétert, mely leírja, hogy melyik irányból kell kulcsot áthozni egy ilyen elágazás esetén. A paraméter alapértelmezett értéke semleges, és a metódusok nem is használják, amíg nincs rá szükségük. Amikor van, akkor egy kivétel, a `NotSelectedRuleException` váltódik ki, melynek hatására a vezérlő egy párbeszédablakot küld a felhasználónak. Az alkalmazandó szabály kiválasztása után a törlés újra lefut, ezúttal már a kiválasztott paraméternek megfelelően, és remélhetőleg sikeresen. A kivétel már a helyreállítási szakaszban váltódik ki, azaz a törlés egy része már lefutott fán. A megismételt törlés előtt vissza kell állítani a korábbi állapotot.

4.3. A program felépítése

4.3.1. Modell réteg – a `tree` csomag osztályai

`Node`

Az osztály példányai lesznek a fa csúcsai. A csúcs tárolja a saját kulcsait, saját azonosítóját, illetve a szülőjére, bal-, középső és jobboldali gyermekére mutató referenciát, ha vannak ilyenek. A kulcsokat tartalmazó változók a `Comparable` interfészt megvalósító generikus típusúak, a csúcs bármilyen összehasonlítható típus elemével példányosítható. (A példányosítást a fa végzi.) A csúcs azonosítója nem vesz részt a műveletek megvalósításában, szerepe, hogy a vezérlő rétegben létrehozott csúcsokat ez alapján lehet majd azonosítani a modell csúcsaival. (Ezt az állítást a `TwoThreeTree` osztály `searchPath` mezőjének ismertetésekor pontosítom.)

Bár a csúcs lehet belső csúcs, vagy levél szerepkörben, a fizikai megvalósításuk megegyezik. A levélre mindig igaz, hogy csak egy kulcsa van, és nincsen gyermeke,

míg a belső csúcsnak mindig van legalább egy gyermeke, így minden csúcsról egyértelműen eldönthető, hogy milyen típusú. A csúcs konstruktora csak a baloldali kulcsot, és a csúcs azonosítóját állítja be, azaz kezdetben minden csúcs levél, az esetleges további értékeket később kapja meg. Az osztály minden mezőjéhez tartozik egy *getter* és egy *setter* metódus, illetve egy `isLeaf()` metódus, amely a fent említett tulajdonságok ellenőrzése után megmondja egy csúcsról, hogy az levél-e.

`TwoThreeTree`

Az osztály példánya maga a fa, amelyen különböző műveletek végezhetők. Típusparamétere a `Comparable` interfészt megvalósító típus, melyet a felhasználó választhat ki a grafikus felületen, néhány megadott típus közül. A fa csúcsai ezzel a típussal lesznek majd példányosítva. Az osztály mezői:

- **`int`** `idOfNextNode` – a következőként létrehozott csúcs azonosítója, minden csúcs létrehozásakor inkrementálódik.
- `Node<T> root` – a gyökér csúcs, értéke *null*, ha a fa nem tartalmaz egyetlen elemet sem. Ha nem *null*, akkor a gyermekeire mutató referenciák használatával a fa bármely csúcsa rekurzívan elérhető, ezért egyetlen más csúcsot sem kell a fában folyamatosan ismernünk.
- `List<Integer> searchPath` – egy művelet végrehajtásakor az érintett levélbe vezető keresőúton szereplő csúcsok azonosítóit tartalmazza. Kereséskor a `search()` függvény minden csúcs bejárásakor beleteszi ebbe a listába az éppen bejárt csúcs azonosítóját, illetve egy extremális elemet, ha az utolsónak bejárt levélcsúcs kulcsa nem a keresett érték. A modellnek csak a lista utolsó elemére van szüksége az előfeltétel ellenőrzéséhez (ti. benne van-e már az adott kulcsú levél a fában), amelyet a `search()` visszaadhatna az út tárolása nélkül is, azonban a megjelenítéshez a vezérlőnek át kell tudnunk adni a teljes listát.
- `List<OperationLoggerElement> operationLog` – az egyes műveletek végrehajtása közben generálódó, a művelet végrehajtása során megtett elemi lépéseket tartalmazó lista. A modell ezt sem használja, a vezérlőnek lesz rá szüksége az animációs lépések megjelenítéséhez.

Az osztály metódusai, függvényei:

- `List<OperationLoggerElement> insert(T value)`, és

- `List<OperationLoggerElement> remove(T value, int leftHandRule)` a megfelelő típusú paraméterrel inicializálják a műveleteket, azonosítják és kezelik a speciális eseteket, pl. az első kulcs beszúrását üres fába, vagy törlési kísérletet a fában nem szereplő kulcsra.
- `List<Integer> initSearch(T value)` a megfelelő típusú paraméterrel inicializája a keresést. A visszaadott lista a keresőút (`searchPath`). A fa gyökerén kezdi a megadott kulcs keresését.
- `void search(Node<T> n, T value)` - a paraméterében kapott csúcsból indítja a keresést (rekurzív).
- `void inOrder(Node<T> node)` - a paraméterben kapott csúcsból indítja az inorder bejárást, és a bejárt levélcsúcsot az `inorderTraversal` listához fűzi (rekurzív).
- `Node<T> lastInterior(Node<T> node, T value)` - meghatározza azt az utolsó belső csúcsot, melynek levele a paraméterben kapott kulcsú levél.
- `Node<T> findUpperKey(Node<T> n, T f)` - baloldali levél törlésekor meghatározza azt a felsőbb szinten levő csúcsot, amelyben a törölt kulcs keresőkulcsként szerepel.
- `void insert(Node<T> parentNode, Node<T> nodeToBeInserted, T value)` - a kapott paraméter ismeretében meghatározza, hogy milyen típusú beszúrás következik, (történhet egykulcsú vagy kétkulcsú csúcsba, vagy kétkulcsú gyökérbe) és meghívja a típusnak megfelelő metódust az alábbiak közül

  ```
  void insertToOneKeyNode(Node<T> parentNode, Node<T> nodeToBeInserted, T value)
  void insertToTwoKeyNode(Node<T> parentNode, Node<T> nodeToBeInserted, T value)
  void insertToTwoKeyRoot(Node<T> node, Node<T> nodeToBeInserted, T value)
  ```
- `T succ(T value)` - meghatározza a paraméterben kapott kulcs rákövetkezőjét. Mivel a rákövetkező meghatározására csak abban az esetben van szükség, ha baloldali levél kulcsértékét töröljük, a feladat leegyszerűsödik: a törölt kulcsú levél szülőjének baloldali kulcsa a rákövetkező. (Egy középső vagy jobboldali levél rákövetkezőjének meghatározására ez a függvény nem lenne alkalmas.)
- `void removeFromLeaf(Node<T> n, T value, int leftHandRule)` - a paraméterben kapott utolsó belső csúcsból törli a paraméterben kapott kulcsú

levelet, és ha szükséges, meghívja az eliminációt kiválasztó metódust

- `void eliminationControler(Node<T> hole, int leftHandRule)` - ha egy levél törlését követően egy utolsó belső csúcs kulcs nélkül marad, akkor a csúcs helyzete alapján meghívja rá a megfelelő eliminációs metódust az alábbiak közül: (mindegyikük az invariáns tulajdonságot sértő, kulcs nélkül maradt csúcsot, illetve a törlés irányát jelző számot kapja paraméterként)

```
eliminateLeftNodeWithOneKeyParentOneKeySibling(...)
eliminateRightNodeWithOneKeyParentOneKeySibling(...)
eliminateLeftNodeWithOneKeyParentTwoKeySibling(...)
eliminateRightNodeWithOneKeyParentTwoKeySibling(...)
eliminateLeftNodeWithTwoKeyParentOneKeySibling(...)
eliminateRightNodeWithTwoKeyParentOneKeySibling(...)
eliminateMedianNodeWithTwoKeyParentOneKeyLeftSibling(...)
eliminateMedianNodeWithTwoKeyParentOneKeyRightSibling(...)
eliminateLeftNodeWithTwoKeyParentTwoKeySibling(...)
eliminateRightNodeWithTwoKeyParentTwoKeyNode(...)
eliminateMedianNodeWithTwoKeyParentTwoKeyLeftSibling(...)
eliminateMedianNodeWithTwoKeyParentTwoKeyRightSibling(...)
```

`OperationLoggerElement`

A `TwoThreeTree.java` osztályban említett `operationLog` elemei ennek az osztálynak a példányai.

Az osztály mezői:

- **`int`** `nodeIdA`, `nodeIdB` – csúcsazonosítók, melyek az elemi műveletben érintett csúcsokat reprezentálják.
- `String value` – az új érték
- **`int`** `operationId` – magát az elemi műveletet reprezentáló azonosító

A vezérlő a mezők értékeiből tudja majd, hogy a következő animációs lépésben mely csúcs melyik mutatója vagy kulcsa változik és hogyan.

Pl. legyenek az osztály egy példányának mezőértékei rendre
(x, y, NOT_NEEDED, SET_LEFT_CHILD), illetve
(x, NOT_NEEDED, „kulcs”, SET_LEFT_KEY)
Az első sor eredményeképpen *x* baloldali gyermeke *y*-ra, a második sorban pedig *x*

baloldali kulcsa „*kulcs*”-ra változik a kirajzolt fán. Ha egy lépés végrehajtásához valamelyik mezőértékre nincs szükség, akkor azt a *NOT_NEEDED* konstans jelöli.

A mezők értéke a konstruktorban kerül meghatározásra, a további metódusok a hozzájuk tartozó *getter* függvények.

`GraphicSupport`

Az osztály tevékenysége a modellre nincs hatással, feladata, hogy egy fa szerkezetének ismeretében megadja a vezérlőnek a fa valamennyi részfájának horizontális kiterjedését. Ennek célját a *„Ritka” 2-3-áris fák (lásd 4.2.4)* ill. az *Elrejthető részfák (lásd 4.2.5)* c. részekben ismertettem.

`AlreadyContainsException`, `AlreadyRemovedException`

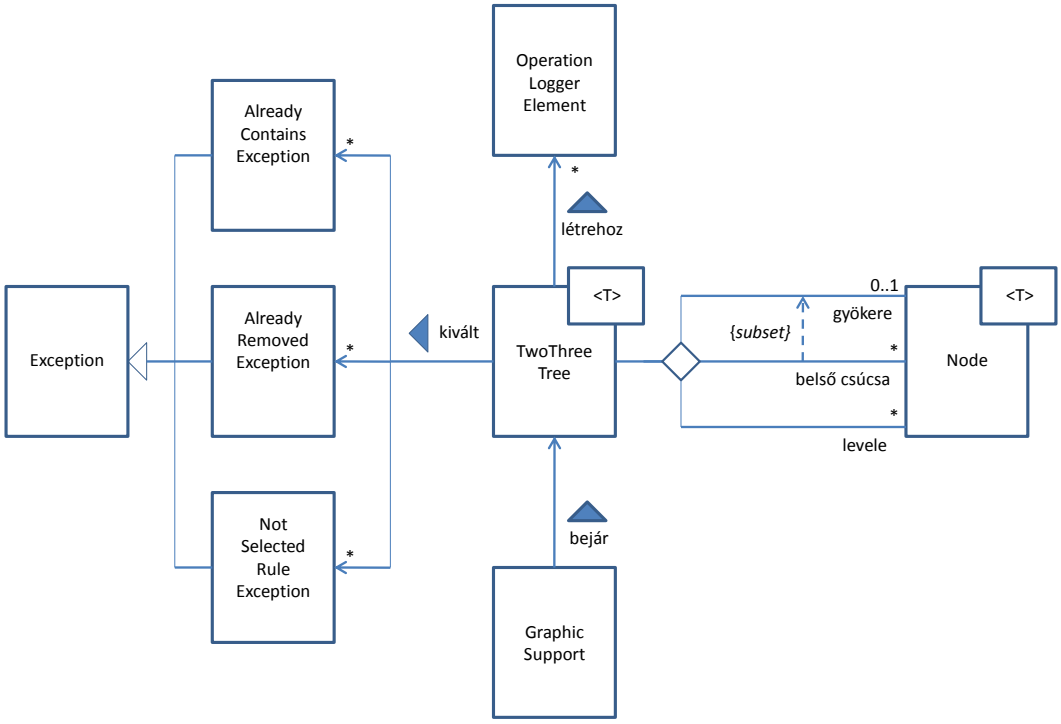
Ha a fába olyan elemet próbált beszúrni a felhasználó, amit az már tartalmazott, vagy olyan elemet próbált törölni, amelyet nem tartalmazott, akkor a modellnek nincs teendője, amiről a vezérlőt ezen kivételek kiváltásával tájékoztatja.

`NotSelectedRuleException`

A „*Több átheleyzhető kulcs kezelése*” *(lásd 4.2.4)* c. pontban részletezett célt szolgáló kivétel.

Az osztály **`boolean`** `oneKeyOnLeft` és **`boolean`** `oneKeyOnRight` mezői a konstruktorban megkapják a környező csúcsok tulajdonságait, ezek alapján tudja majd a vezérlő, hogy pontosan milyen lehetőségeket kell a felhasználónak felajánlania. Az osztály `String key` mezője azt a konkrét kulcsértéket tárolja, amelynek törlése során a kivétel kiváltódott, ám ezt csak a vezérlő réteg állítja majd be, amikor elkapja a kivételt.

A modell osztálydiagramja[10] a 21. ábrán látható.



21. ábra
A modell réteg osztálydiagramja

4.3.2. Vezérlő réteg – a `graph` csomag osztályai

`LogicalNode`

Az osztály a modell `Node` típusának megfelelője a vezérlő rétegben, mezői és metódusai a rétegek eltérő feladata miatt némileg eltérnek. Mezői az alábbiak:

- `String leftKey, rightKey, fullKeys` – a csúcs kulcsainak szöveges ábrázolása a *„Hosszú” kulcsok megjelenítése (lásd 4.2.3)* c. részben tárgyaltak szerint. Utóbbi a teljes bal- és jobboldali kulcsot, és a kettő közti elválasztó karaktert tartalmaz. A bal- és jobboldali kulcsnak – ha a hossza indokolja – csak a csonkolt alakja lesz a `leftKey`, `rightKey` változókban tárolva.
- **`boolean`** `isLeaf` – tárolja, hogy a megjelenítendő csúcs levél-e, vagy akként viselkedik-e. Utóbbi a helyzet, ha a felhasználó elrejtett egy részfát. A `GraphicPanel` osztály példánya ebből a mezőből tudja majd, hogy az elrejtett csúcsnak nem kell kirajzolnia a gyermekeit, akkor sem ha azok léteznek.
- **`float`** `coordVerticalStart, coordHorizontalStart` – a fa szerkezete alapján ezek a mezők tárolják az adott csúcs pozíciójának koordinátáit a grafikus

panelen. Ha nem animálva történik a megjelenítés, akkor a csúcsok ténylegesen itt kerülnek kirajzolásra.

- **float** `coordCurrentVerticalStart`, `coordCurrentHorizontalStart` – annak a pontnak a koordinátái, ahol a csúcs ténylegesen megjelenik az adott pillanatban. Animált megjelenítés közben egy csúcs pillanatnyi helye eltérhet attól, ahol ténylegesen lennie kéne, az új helyét animáció közben foglalja el.
- **int** `width`, `height` – az egyes grafikus csúcsok méretei
- **int**`[] childList` – a csúcs gyermekeinek azonosítóiból álló tömb
- **boolean** `isCollapsed` – tárolja, hogy a csúcsból kiinduló részfát a felhasználó elrejtette-e

Az osztály metódusai:

Levél és nem levélcsúcs előállítására külön konstruktorok szolgálnak. A mezők többségéhez tartozik *getter* függvény és *setter* metódus is, ezen kívül a `String createKeyStub(String key)` függvény a kulcsok csonkolását végzi el, ha szükséges.

NodeListTransformator

Az osztály feladata, hogy a modell aktuális állapota alapján elkészítse a logikai csúcsok olyan listáját, melyben minden csúcs megkapja a kulcsait, gyermekeinek azonosítóit, és a megjelenítéshez használandó koordinátákat. A koordináták meghatározásához a modell `GraphicSupport` osztályának példányát használja, melytől megkapja az ábrázolandó szintek számát, és minden részfa kiterjedését. A kiterjedések összegeként alakul ki az egyes grafikus csúcsok végleges helye a gyökérhez képest. A gyökér függőleges elhelyezkedése állandó a grafikus panel tetején, míg a vízszintes elhelyezkedéshez ismernünk kell a panel méretét. Ha az egész fa vízszintes kiterjedése nem haladja meg az aktuális ablakméretet, akkor a panel mérete megegyezik az ablakéval, és a gyökér ennek közepére kerül, ellenkező esetben az **int** `getNeededWidth()` függvény kiszámítja a teljes fa megjelenítéséhez szükséges panel szélességét, és a gyökér ennek lesz a közepén. Az osztály által létrehozott csúcslista lényegében a modell `TwoThreeTree` osztályának megfelelője a vezérlő rétegben.

GraphicPanel

A `JPanel`-ből származó osztály, amelyre a megjelenítendő fa kirajzolásra kerül. Mezői:

- `List<LogicalNode> listOfNodesToDraw` – a kirajzolandó logikai csúcsok listája, amit a `NodeListTransformator` osztály példánya állított elő
- `Map<Integer, int[]> childLists` – az egyes csúcsazonosítókhoz tartalmazza a megfelelő csúcs gyermekeinek listáját.
- `Map<Integer, NodeLocator> mapOfLocations` – az egyes csúcsazonosítókhoz tartalmazza a `NodeLocator` osztály egy példányát
- **`int`** `imageWidth`, `imageHeight` – a teljes csúcslista megjelenítéséhez szükséges panelméretek
- `MainWindow mainWindow` – az alkalmazás főablakának példánya
- `List<Integer> searchPath` – csúcsazonosítókat tartalmazó lista, az esetlegesen kiemelendő keresőúton szereplő csúcsokkal

A konstruktor paramétereiből a felsorolt mezők többsége értéket kap, melyeket a `JPanel` felüldefiniált `paintComponent()` metódusa használ majd. A `paintComponent()` létrehoz egy `BufferedImage` típusú objektumot, amelyen a kirajzolást végző további metódusok elvégzik a teendőiket, majd elhelyezi azt a panelen.

A rajzolás menete az alábbi: A `paintComponent()` bejárja a kirajzolandó logikai csúcsok listáját, és minden csúcsra meghívja a `drawNodesOnImage()` metódust. Itt a csúcs már ismertetett mezői, valamint a `searchPath` lista alapján a metódus meg tudja határozni a csúcs színét, alakját, helyét, szövegeinek szélességét. Ezen kívül minden kirajzolt csúcshoz létrehoz egy `NodeLocator` típusú, és egy `OnNodeMouseListener` típusú objektumot. Az elrejtett részfát jelző háromszög egy, a `Polygon` osztályból származó `CollapsedNode` osztály példánya, ez az osztály a `GraphicPanel`-en belül van definiálva.

Miután a `paintComponent()` végigjárta a csúcslistát, meghívja a `drawConnectionsOnImage()` metódust, amely elhelyezi a képen a csúcsokat összekötő éleket. Ez a `childList` alapján bejárja azokat a csúcsokat, amelyeknek van kirajzolt gyermekük, azaz az el nem rejtett belső csúcsokat. A tömbben szereplő azonosítójú csúcsokat megkeresi a `mapOfLocations`-ben, és azok körvonalának megfelelő pontjához egyenest rajzol az aktuális csúcsból.

NodeLocator

Az awt csomag `Rectangle` típusából származik, kiegészítve azt egy `selected` logikai változóval. Az osztály elemeit a `GraphicPanel` példánya hozza létre rajzolás közben, minden kirajzolt csúcshoz egyet. A téglalap befoglalja azt a grafikus csúcsot, amivel együtt jött létre, meghatározva ezzel annak körvonalait. A körvonalra az egér-eseményeket figyelő `OnNodeMouseListener` osztálynak van szüksége, annak eldöntésére, hogy egy esemény melyik csúcson történt. A `selected` mező értéke aszerint állítódik be, hogy a befoglalt csúcs kiemelt csúcs-e, azaz szerepel-e kiemelendő keresőútban. Ezt az értéket használja a `GraphicPanel` osztály `drawConnectionsOnImage` metódusa annak eldöntésére, hogy egy egyenes kiemelt csúcsokat köt-e össze. Ha igen, akkor az egyenes színe követi a kiemelt csúcsok színét.

OnNodeMouseListener

Az osztály a `MouseInputAdapter` kiterjesztése, pontosabban annak két metódusát definiálja:

- a **`void`** `mouseMoved(MouseEvent e)` metódus egy csúcs fölé húzva megjeleníti a hozzá tartozó tooltip-et, benne a csúcsban szereplő kulcsok teljes szövegével
- a **`void`** `mouseClicked(MouseEvent click)` megvizsgálja, hogy a kattintás az egér bal vagy jobb gombjával történt-e, illetve hogy valamely belső csúcs területén kattintott-e a felhasználó, és ennek megfelelően kezdeményezi az érintett részfa elrejtését, vagy újbóli megjelenítését.

Az osztály mezőértékeit a konstruktor paramétereiben kapja. Ezek:

- az alkalmazás főablaka, amely a csúcsok elrejtéséről, megjelenítéséről gondoskodik.
- `NodeLocator` objektum, amely meghatározza egy csúcs területét
- a grafikus panel, melyen a tooltip-ek megjelennek
- `LogicalNode` objektum, melyből a tooltip megjelenítéséhez kinyerhető a szükséges szöveg

CollapsedList

Az *Elrejthető részfák (lásd 4.2.5.)* c. részben tárgyalt lista

AnimationControler

Az *Animációs lépések azonosítása, a csúcsok mozgatása (lásd 4.2.7.)* c. részben ismertetett feladatokat ellátó osztály. `SetNewNodeList` metódusa megkapja azt a fát, amelyen az animálandó műveletek már el vannak végezve, ebből egy `NodeListTransformator` példány segítségével előállítja azt a csúcslistát, melyet az animáció végállapotában ki kell majd rajzolni. Bár a kiinduló állapotot leíró listát is ez az osztály készíti el, sem ezt, sem pedig a későbbi átmeneti állapotokat nem ismeri, minden egyes metódusa paraméterben kapja majd meg.

A kiinduló listát az `updateFixNodePositions` függvény készíti. A lista a fának pontosan a művelet végrehajtása előtti csúcsait tartalmazza, de minden csúcs koordinátája a végrehajtás utáni állapothoz igazodik. Ezen a ponton fog eltérni egymástól a `LogicalNode` oszály példányainak `coordHorizontalStart` és `coordCurrentHorizontalStart`, illetve `coordVerticalStart` és `coordCurrentVerticalStart` mezőinek értéke. A csúcsok mozgatásáért az alábbi függvények felelnek:

- **boolean** `movedVertically(LogicalNode node)` és
- **boolean** `movedHorizontally(LogicalNode node)` jelzik, ha a paraméterként kapott csúcs nincs a végleges helyén
- `isFinalPositionReached(List<LogicalNode> source)` az előző függvényeket a csúcslista minden elemére végrehajtja, így megállapítja, hogy az összes csúcs a helyére került-e már.
- **void** `approximatePositions(List<LogicalNode> source)` ha van olyan csúcs, ami nem a helyén jelent meg, akkor azt egységnyivel közelebb „teszi” a helyéhez

Az állapotátmenet további lépéseiben megjelenhetnek vagy eltűnhetnek csúcsok, vagy megváltozhatnak a kulcsaik és mutatóik. Egy-egy ilyen lépéssel az alábbi metódusok frissítik az átmeneti listát:

- **void** `addNewLeaf(int nodeAId, List<LogicalNode> internList)`
- **void** `addNewInternalNode(int nodeAId, List<LogicalNode> internList)`
- **void** `killANode(int nodeAId, List<LogicalNode> internList)`
- **void** `setAMidChild(int nodeAId, int nodeBId, List<LogicalNode> internList)`

- `void setALeftChild(int nodeAId, int nodeBId, List<LogicalNode> internList)`
- `void setARightChild(int nodeAId, int nodeBId, List<LogicalNode> internList)`
- `void removeMidChild(int nodeAId, List<LogicalNode> internList)`
- `void setLeftKeyFromString(int nodeAId, String s, List<LogicalNode> internList)`
- `void setRightKeyFromString(int nodeAId, String s, List<LogicalNode> internList)`
- `void removeLeftChild(int nodeAId, List<LogicalNode> internList)`
- `void removeRightChild(int nodeAId, List<LogicalNode> internList)`

Animator

Absztrakt osztály, melynek leszármazottja lesz a tényleges animációt végző `InsertionAnimator`, illetve `RemoveAnimator` osztály. Implementálja az `ActionListener` interfészt, hogy reagálni tudjon a felhasználói felületen a *Következő* gomb megnyomására, illetve a `timer` által kiváltott eseményre. Fontosabb adattagjai:

- `Timer timer` – az animációs lépések időzítésére
- `TwoThreeTree<?> oldTree, finalTree, List<LogicalNode> oldList, newList` – az animáció kezdeti és végállapotában megjelenítendő fák, illetve az ezek logikai csúcsait tartalmazó listák
- `List<LogicalNode> internList` – a csúcsokat az átmeneti állapotokban tartalmazó lista, amin az `AnimationControler` metódusai alakítanak lépésről lépésre.
- `List<List<OperationLoggerElement>> listOfLogs` – a „belső” lista az egyes műveletek során megtett elemi lépéseket tartalmazó lista. Ha egyszerre több operandussal végrehajtott műveletet kell animálni, akkor minden operandushoz külön lépéslista tartozik.
- `boolean stepByStep` – a megjelenítés módja, igaz, ha a felhasználó a lépéseket egyenként kívánja megtekinteni, hamis, ha a teljes animáció a timer használatával fut le

Fontosabb metódusok:

a konstruktor paraméterben kapja a kiinduló állapotban levő fát, amelyből a `transformator` elkészíti az ennek megfelelő csúcslistát.

- **void** `startAnimation()` – absztrakt metódus, amit a leszármazottak definiálnak majd
- **void** `operationIdentifier(OperationLoggerElement operation)` – az elvégzett lépések listáján végighaladva a műveletazonosítónak megfelelően meghívja az `internList`-re az `AnimationController` kívánt függvényét.

`InsertionAnimator`, `RemoveAnimator`

A két osztály közötti fő különbség, hogy beszúráskor először az új helyükre kerülnek a csúcsok, majd megjelenik a többi változtatás, míg törléskor fordított a sorrend. Ez eltérést jelent az eseménykezelő működésében, és a `timer` inicializálásában.
Különbözik továbbá a kezdeti állapot megjelenítésének mikéntje. Beszúrás esetén megfelelő, ha a régi fát az új fának megfelelő méretű panelen jelenítjük meg, hiszen az új panel az eddiginél nem lehet kisebb. Törlés esetén viszont a régi méretet kell használnunk, mert az új, kisebb méretű panelen a régi fa szélei már nem jelennének meg.

`CollapsedAnimationException`

Ezt a kivételt az `operationIdentifier` metódus váltja ki abban az esetben, ha az animáció rejtett részfa elemét érinti.

`parser` csomag

A *Generikus fa, Type Erasure (lásd 4.2.6.)* c. részben tárgyalt interfészeket, osztályokat taralmazza.
A `NodeTypeFactory` interfészt megvalósító osztályok:

- `IntegerNodeTypeFactory`
- `FloatNodeTypeFactory`
- `CharacterNodeTypeFactory`
- `StringNodeTypeFactory`

Metódusaik a megfelelő típusparaméterrel példányosítják a modellt, az `InputParser`-t és a `RandomGenerator`-t, és ezeket az objektumokat adják vissza a hívónak.
A `NodeType` objektum már a nézet réteget szolgálja ki a kért típusú elemeket kezelő példányokkal.

Bemeneti lista készítése:

Az `InputParser` interfészt megvalósító osztályok (`IntegerInputParser`, `FloatInputParser`, `CharacterInputParser`, `StringInputParser`) feladata hogy egy szöveget – mely a felhasználói felülettől érkezik – a kívánt típusú értékek listájává alakítsanak, illetve jelezzék, ha az átalakítás nem lehetséges (pl. `NumberFormatException`-t dobnak). Először az esetlegesen többszörözött elválasztó karakterek szűrik ki, majd a kapott szöveget karakterenként feldolgozzák. Egy elválasztó karaktert követően megkeresik a következőt, és a kettő közt szereplő szöveget megpróbálják a megadott típusú elemként értelmezni. Ha ez sikerül, akkor az elemet a listába helyezik, és továbbmennek a következő elemre.

Véletlen elemek generálása:

Valamennyi `RandomGenerator`-t megvalósító osztály metódusa egy karakterláncot kap paraméterként, amelyhez az általa generált új elemet hozzá kell fűznie.

Véletlen szám generálásához a `java.util.Random` osztályt használom.

Egész szám előállításakor (`IntegerRandomGenerator`) valójában két egész szám különbségét adjuk a vissza, annak érdekében, hogy a visszaadott szám negatív is lehessen.

Lebegőpontos szám előállításakor (`FloatRandomGenerator`) egy véletlen számot megszoroz 0.1-el, így érve el, hogy a visszaadott szám egy tizedesjegyet tartalmazzon. Ennek csak a kényelmes megjelenítés szempontjából van jelentősége.

Karakter előállítása (`CharacterRandomGenerator`) egy előre megadott konstans ábécéből történik, az eredmény az ábécé „véletlenedik” eleme.

Karakterlánc előállítása (`StringRandomGenerator`) annyiban tér el a karakterétől, hogy az ábécéből véletlen sok elemet összefűzünk. Ez jelenleg 2-4 karakter lehet, melyet a lebegőpontos számokhoz hasonlóan kényelmi szempontok indokolnak.

`TreeInstance`

Amint a felhasználó kiválasztja a fa elemtípusát, létrejön ennek az osztálynak a példánya, amely tartalmazza majd a fát, és a használatához szükséges egyéb komponenseket: a `RandomGenerator` és az `InputParser` megfelelő példányát, a fán eddig elvégzett műveleteket leíró szöveget, és ugyanennek a szövegnek egy korábbi verzióját, mely a fa legutóbbi művelet végrehajtása előtti állapotot írja le. Ez az osztály

kezdeményezi a felhasználó által adott bemenet ellenőrzését, és ha ez sikeres, akkor a modellen a kiválasztott műveletek végrehajtását. Az operandusokat szétválogatja aszerint, hogy a beszúrásuk vagy törlésük sikeres volt-e, megkönnyítve ezzel a felhasználói felületen a visszajelzés megjelenítését.

Minden művelet végrehajtásakor frissül a fa „életútját” dokumentáló karakterlánc. Ennek legelső karaktere mindig a kiválasztott elemtípust tartalmazza, a további részében egy-egy műveletet azonosító karakter, a hozzá tartozó operandus, és egy elválasztó karakter ismétlődik.
Fájlba mentéskor a keletkező fájl ezt a karakterláncot tartalmazza, a nézeten megjeleníthető műveletlista-ablak elemei is ebből készülnek. Mentett fájl beolvasásakor a szöveget karakterenként feldolgozó `buildTreeFromString` metódus azonosítja a műveleteket, és azok meghívásával felépíti a fát. Hasonló a helyzet, ha a felhasználó a visszavonás gombra kattint. Bár az osztály megvalósítja a `Cloneable` interfészt, klónozáskor csak a fa „történetéről” készül másolat, és az előbbi metódus ezúttal ebből építi újra a fát.

4.3.3. Nézet réteg – a `gui` csomag osztályai

A nézet a kezelőfelületből, illetve a grafikus panel megjelenítéséből áll.

A grafikus kezelőfelület részeit leíró, `JPanel`-ből származtatott osztályok:

- `TypeSelectorPanel` – a fa elemtípusát kiválasztó rádiógomb-csoport
- `LowerInputButton` – a véletlen elem gombot és a beviteli mezőt tartalmazó panel
- a beszúrás, keresés és törlés műveleteket kezdeményező gombokat tartalmazó panel
- `InputPanel` – az előző két panelt magába foglaló panel
- `UndoResetPanel` – A *Visszavonás* és az *Újraindítás* gombokat tartalmazó panel
- `DisplaySelectorPanel` – a kirajzolt fa megjelenítésének módját szabályzó elemeket tartalmazó panel
- `FileStreamPanel` – az adminisztrációs műveleteket kezdeményező elemeket tartalmazó panel.

Ezen osztályok létrehozzák és megjelenítik az elemeiket, beállítják kezdőállapotukat, illetve rendelkeznek a szükséges *getter* és *setter* metódusokkal. A `FileStreamPanel` tartalmazza a fájlműveleteket, illetve a mentéssel/megnyitással kapcsolatos párbeszédablakokat kezelő metódusokat is.

A vezérlőelemek paneljeit a `ControlPanel` osztály fogja össze.

A kezelőelemek állapota viszonylag gyakran változik, és egy-egy ilyen változás egyszerre több elemet érint. Az ilyen változások végrehajtására készült statikus metódusokat a `GuiHelper` osztály tartalmazza.

A `MainPanel` tartalmazza a `ControlPanelt`-t, illetve kiegészíti azt a felhasználónak szóló státuszüzenet címkéjével, létrehozza a rádiógombokhoz az eseményfigyelőt, és megvalósítja az eseménykezelés azon részét, mely egy, valamely kezelőelemet ért felhasználói beavatkozást követően frissíti a többi elem állapotát, a `GuiHelper` szükséges metódusainak hívásával.

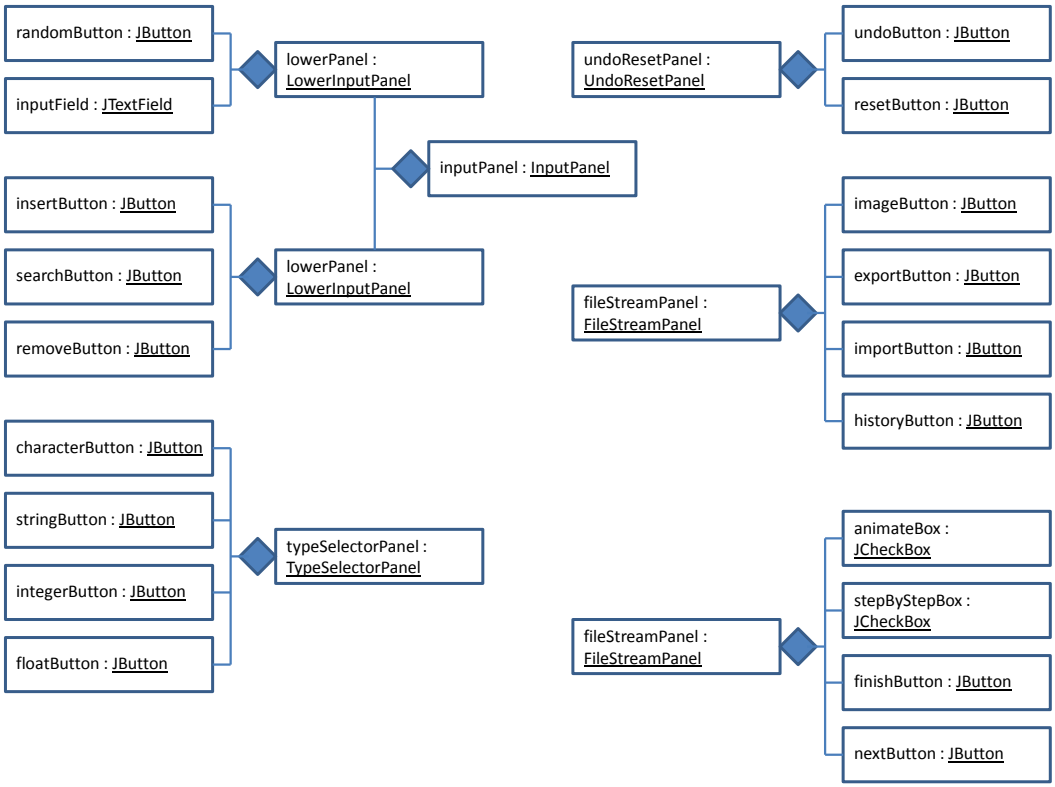
A `HistoryWindow` a `JFrame` leszármazottja, benne jelenik meg az eddig elvégzett műveletek listája.

A `RuleSelectorDialog` egy párbeszédpanel, ami akkor jelenik meg, ha a fa egy törlendő eleme többféleképpen is törölhető.

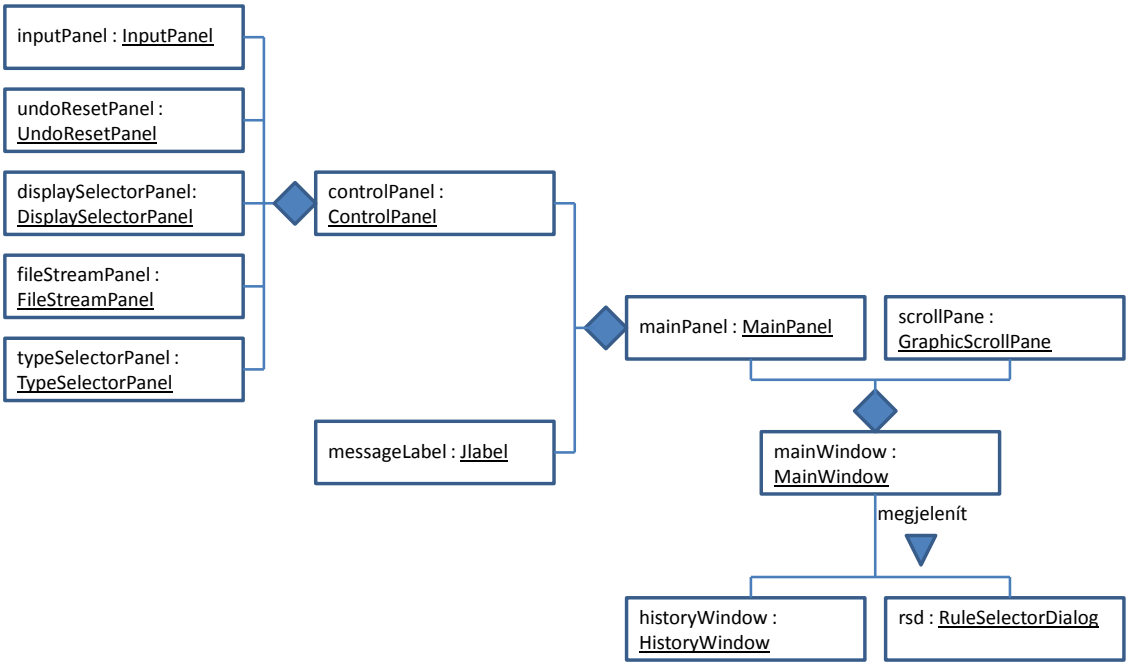
A `JScrollPane`-ből származtatott `GraphicScrollPane` példánya jeleníti meg a kirajzolt fát tartalmazó grafikus panelt. Ez az objektum hívja a `GraphicPanel` rajzoló metódusát, így a rajzoláshoz szükséges minden paramétert is ismernie kell. Ezeket a konstruktor paramétereiként kapja meg. A grafikus panel létrehozásán kívül kezelnie kell a görgetősávokat, hogy azok csak akkor jelenjenek meg, mikor szükség van rájuk, illetve a grafika frissítésekor, különösen animáció közben a fa ne „ugráljon”, és mindig látható legyen az a része, amelyen változás következik be.

A `MainWindow` szintén a `JFrame` leszármazottja, feladata a `TreeInstance` példányosítása, az eseménykezelés fennmaradó része, a felhasználónak szóló üzenetek

kezelése, valamint az elrejtendő részfákat tartalmazó lista karbantartása.



22. ábra
A kezelőfelület komponenseinek objektumdiagramja[10]



23. ábra
A kezelőfelület objektumdiagramja[10]

A kezelőfelület objektum-diagramját a 22. és 23. ábrák tartalmazzák.

4.3.4. Az alkalmazás belépési pontja – `main()` metódus

Az alkalmazás ikonjára kattintva lekérdezi a futtató környezet grafikus felületének tulajdonságait, alkalmazza azokat a főablakra, majd elindítja a főablakot.

4.4. Tesztelés

4.4.1. Statikus tesztelés

A program tervezése során került végrehajtásra, a követelményspecifikációban megadottak átgondolásával. A statikus kódelemzést a fejlesztőkörnyezet automatikusan végezte.

4.4.2. Dinamikus tesztelés

Fejlesztés közben sor került a komponensek unit- és integrációs tesztjeire, ezek azonban nem kerültek dokumentálásra. A Java nyelv támogatja automatikus unit tesztek írását és futtatását, ami nem valósult meg, ám a program esetleges továbbfejlesztése során ezek elkészítése nagy segítséget jelentene.

A tesztelés során a legnagyobb hangsúlyt a rendszertesztek tervezésére és végrehajtására fektettem.
A teljes rendszerteszt leírását terjedelmi okokból mellőzöm. Az alábbi táblázatokban egy *smoke* teszt esetei szerepelnek, funkció-csoportonkénti bontásban. (A smoke teszt rendszer-szintű tesztesetek olyan listája, mely lefedi az alkalmazás főbb funkcióit és tipikus felhasználási eseteit. Jellemzően regressziós jellegű tesztelésre használják.)

Megjegyzés: A dinamikus tesztek tervezésekor és futtatásakor szokásos elvárás, hogy a tesztelő a kódtól független személy legyen. A szakdolgozatírás természetéből fakadóan ez az elvárás nem teljesül, ami a tesztelés minőségét negatívan befolyásolhatja[11].

Kezelőfelület állapotátmenetei

Teszteset rövid leírása és elvárt eredménye	Tényleges eredménye
Minden vezérlőelem, a grafikus fa és a görgetősávok is megjelennek különböző operációs rendszer, felbontás és színmélység használata mellett: Windows XP SP3 op. rendszer, 1280x800 felbontás, 24 bit színmélység Windows 7 SP1 op. rendszer, 1366x768 felbontás, 32 bit színmélység Ubuntu op. rendszer (12.04 kernel), 1366x768 felbontás, 32 bit színmélység	ok
A program indítása utáni állapotban csak a típusválasztó rádiógombok, és a Megnyitás gomb aktív, minden más elem inaktív.	ok
Bármelyik típus kiválasztása után aktív a Véletlen elem gomb, a beviteli mező, a fa-műveletek gombjai, a Megnyitás és az Eddigi műveletek gomb, és a típusválasztó rádiógombok. Minden más elem inaktív.	ok
Egyszerre egy típus választható ki. Típus kiválasztása után a Véletlen elem gomb a kiválasztott típusú elemet helyez a beviteli mezőbe. Az Eddigi műveletek gomb hatására felugró ablakban a kiválasztott típus helyesen jelenik meg.	ok
Típus kiválasztása után a beviteli mező szerkeszthető.	ok
Írjunk a beviteli mezőbe a kiválasztott típusnak megfelelő adatot, majd nyomjuk meg a Beszúrás gombot. A státuszinformációban megjelenik az elem sikeres beszúrása. Az Animálva jelölőnégyzet aktiválódik	ok
Írjuk a mezőbe az előbbi adatot, majd nyomjuk meg a törlés gombot. A státuszinformációban megjelenik a sikeres törlés. A vezérlőelemek a törlést megelőző állapotban maradnak.	ok
Írjunk a beviteli mezőbe a kiválasztott típusnak megfelelő adatot, majd nyomjuk meg a Beszúrás gombot. Írjuk a beviteli mezőbe ugyanezt az adatot, majd nyomjuk meg ismét a Beszúrás gombot. A státuszüzenetben megjelenik, hogy az elem már szerepelt a fában.	ok
Töröljünk olyan elemet, mely nem szerepel a fában. A státuszüzenetben megjelenik, hogy a törölni kívánt elem nem szerepel a fában.	ok
Szúrjunk be néhány elemet a fába, majd keressünk olyan elemre, ami szerepel benne. A státuszüzenetben megjelenik, hogy a keresés sikeres volt.	ok
Szúrjunk be néhány elemet a fába, majd keressünk olyan elemre, ami nem szerepel benne. A státuszüzenetben megjelenik, hogy a keresett elem nem található.	ok
Írjunk egyszerre több elemet a beviteli mezőbe, vesszővel és/vagy szóközzel elválasztva, legyenek köztük olyanok, amik szerepelnek, és amik nem szerepelnek a fában. Nyomjuk meg a beszúrás gombot. A státuszüzenet tartalmazza a sikeresen beszúrt elemeket, és a már tartalmazott elemeket, egymástól elkülönítve.	ok

Ismételjük meg az előző pontot törléssel.	ok
Egy beszúrás vagy törlés után kattintsunk a Visszavonás gombra. A visszavonás gomb inaktívvá válik, minden más elem változatlan marad.	ok
Sok beszúrás és törlés után kattintsunk az Eddigi műveletek gombra. A felugró ablakban megjelenő lista a kiválasztott elemtípussal kezdődik, majd a sikeresen végrehajtott műveleteket tartalmazza. Ha elég sok művelet történt, akkor a lista görgethető.	ok
Néhány művelet elvégzése után kattintsunk az Újraindítás gombra. Minden vezérlőelem állapota az indítás utáni állapotnak felel meg.	ok
Néhány elem beszúrása után kattintsunk a Mentés gombra. Megjelenik az operációsrendszer párbeszédablaka. Adjunk meg fájlnevet, és mentsünk. A mentés helyén megjelenik egy `23t` kiterjesztésű fájl.	ok
Néhány elem beszúrása után a Megnyitás gombbal olvassuk be az előzőleg mentett fájlt. A sikeres beolvasásról a státuszüzenet tájékoztat, a megjelenített fa a korábban mentett. Az Eddigi műveletek lista is a mentett fának megfelelő.	ok
Néhány elem beszúrása után nyomjuk meg a Mentés képként gombot. Megjelenik az operációs rendszer párbeszédablaka. Mentsünk, majd ellenőrizzük a mentés helyén megjelenő `png` fájlt.	ok
Aktiváljuk az Animálva jelölőnégyzetet. Kattinthatóvá válik a Lépésenként jelölőnégyzet, és a Befejezés gomb.	ok
Aktiváljuk a Lépésenként jelölőnégyzetet. Kattinthatóvá válik a Következő gomb.	ok
Deaktiváljuk az Animálva jelölőnégyzetet a Lépésenként jelölőnégyzet mindkét állapota mellett. A Lépésenként nem lesz kiválasztva, inaktívvá válik. A Befejezés és a Következő gomb inaktívvá válik.	ok
Hajtsunk végre egy műveletet animálva.	ok
Hajtsunk végre egy műveletet animálva, lépésenként. A Következő gomb megnyomásáig az animáció szünetel	ok
Animáció közben nyomjuk meg a befejezés gombot. Az elkészült fa kirajzolásra kerül.	ok
Az alkalmazás kezdőállapotában nyissunk meg egy fájlt. A tárolt fának megfelelő elemtípus automatikusan kiválasztódik.	ok

Grafikus ábrázolás

Teszteset rövid leírása és elvárt eredménye	**Tényleges eredménye**
Elemtípus kiválasztása után, az első elem beszúrása előtt nem jelenik meg grafika.	ok
Első elem beszúrása után megjelenik az egyelemű fa.	ok
Újabb elem beszúrása után megjelenik a kételemű fa.	ok

Több elem beszúrása esetén azok mindegyike megjelenik a fában.	ok
Törölt elem eltűnik a fából.	ok
Egyszerre több törölt elem mindegyike eltűnik a fából	ok
Animált beszúrás esetén a csúcsok a helyükre mozognak, majd további beavatkozás nélkül lefut az animáció.	ok
Animált törlés esetén további beavatkozás nélkül lefut az animáció, majd a csúcsok a helyükre mozognak.	ok
Lépésenkénti beszúrás esetén a csúcsok a helyükre mozognak, megtörténik az animáció első lépése, minden további lépés megvárja a Következő gomb megnyomását.	ok
Lépésenkénti törlés esetén az animáció lépései megvárják a Következő gomb újbóli lenyomását, az utolsó lépés után a csúcsok automatikusan a helyükre mozognak.	ok
Animáció közben nyomjuk meg a Befejezés gombot. Az elkészült fa teljesen megjelenik, nincs benne kiemelt keresőút	ok
Lépésenkénti animáció közben nyomjuk meg a Befejezés gombot. Az elkészült fa teljesen megjelenik, nincs benne kiemelt keresőút.	ok
Keresés végrehajtása után, ha a keresett levél szerepel a fában, akkor a keresőút csúcsai pirosak, és a levelet is tartalmazzák. Ha a levél nincs a fában, akkor is van keresőút, de levelet nem tartalmaz.	ok
Belső csúcsra bal gombbal kattintva a részfa eltűnik, sárga háromszög helyettesíti.	ok
Elrejtett részfára jobb gombbal kattintva a részfa megjelenik.	ok
Ha nincs a fában rejtett részfa, akkor tetszőleges alakú fa esetén az utolsó belső csúcsok egyenlő távolságban vannak a szomszédjaiktól.	ok
Ha van rejtett részfa, és a gyökere nem utolsó belső csúcs, akkor a szomszédos részfák elfoglalják a helyét, vagy a háromszög a szülője középvonala felé mozdul.	ok
Az egeret a csúcsokra húzva megjelenik a teljes kulcspárt tartalmazó tooltip.	ok
A képernyőnél szélesebb fa vízszintesen görgethető, újabb beszúráskor vagy törléskor a görgetősáv úgy állítódik be, hogy a fa gyökere látható.	ok
A képernyőnél szélesebb fán végzett animáció közben a görgetősáv úgy állítódik be, hogy az érintett csúcsok láthatók.	ok
A képernyőnél magasabb fa esetén megjelenik a függőleges görgetősáv.	ok
Hajtsunk végre egy műveletet animálva.	ok

Fa helyessége – az alábbi esetek lefedik a beszúrás és törlés által bejárt végrehajtási ágakat. Minden esetnél elvárás, hogy a megjelenített fa teljesítse az Elméleti háttér c. fejezetben tárgyalt követelményeket. Az elemtípus tetszőleges, ugyanazon ág

végrehajtása különböző típusú elemek esetén egy ekvivalencia-partícióba tartozik [11].

Teszteset rövid leírása	**Tényleges eredménye**
Szúrjunk egy elemet a fába.	ok
Szúrjuk egyelemű fába a benne levőnél kisebb elemet.	ok
Szúrjuk egyelemű fába a benne levőnél nagyobb elemet.	ok
Szúrjunk kételemű fába mindkét benne levőnél kisebb elemet.	ok
Szúrjunk kételemű fába mindkét benne levőnél nagyobb elemet.	ok
Szúrjunk kételemű fába olyan elemet, mely értéke a két bent levő érték közötti.	ok
Szúrjunk háromelemű fába újabb elemet.	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 20	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 35	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 45	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 55	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 65	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 20, 15	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 20, 25	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 20, 35	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 20, 45	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 70, 55	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 70, 65	ok
Szúrjuk be üres fába a következő egész számokat: 50, 40, 60, 30, 70, 75	ok
Szúrjunk a fába már benne levő elemet. Nem történik változás.	ok
Töröljünk üres fából. Nem történik változás.	ok
Töröljük egyelemű fa levelét.	ok
Töröljük kételemű fa baloldali levelét.	ok
Töröljük kételemű fa jobboldali levelét.	ok
Töröljük háromelemű fa baloldali levelét.	ok
Töröljük háromelemű fa jobboldali levelét.	ok
Töröljük háromelemű fa középső levelét.	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40. Töröljük ezt: 30	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40. Töröljük ezt: 40	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40. Töröljük ezt: 50	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40. Töröljük ezt: 60	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 70, 10. Töröljük a következőt: 20	ok

Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 70, 10. Töröljük a következőt: 40. Az alkalmazás megkérdezi, melyik irányból szeretnénk kulcsot áthelyezni. Próbáljuk ki mindkét irányból.	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 70, 10. Töröljük a következőt: 60	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 10, 0. Töröljük a következőt: 10	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 10, 0. Töröljük a következőt: 30. Az alkalmazás megkérdezi, melyik irányból szeretnénk kulcsot áthelyezni. Próbáljuk ki mindkét irányból.	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 10, 0. Töröljük a következőt: 60	ok
Szúrjuk a fába a következő egész számokat: 50, 60, 30, 40, 20, 10, 0, 70. Töröljük a következőt: 30. Az alkalmazás megkérdezi, melyik irányból szeretnénk kulcsot áthelyezni. Próbáljuk ki mindkét irányból.	ok
Töröljünk a fában nem szereplő elemet. Nem történik változás.	ok

4.4.3. Feltárt hibák

A program beadásra szánt verzióján a fenti smoke teszt nem futott hibára. A fejlesztés során, illetve a tesztelés egyéb szakaszaiban azonban jelentkeztek kisebb rendellenességek:

- Ha a felhasználó a képernyőjénél nagyobb méretű grafikát a görgetősáv húzásával mozgat, a megjelenés nem folyamatos, a grafika „villog”. Ennek okozója a `GraphicScrollPane` osztályban a görgetősávokra meghívott `repaint()` metódus, amire a JDK egy dokumentált hibájának [12] megkerüléséhez van szükség. A hiba következményeként a megjelenítés sikeressége az alkalmazott színmélységen, illetve a videokártya típusán múlhat
- A kezelőfelület elemeinek rendezéséhez `BoxLayout` elrendezéskezelőt használtam, amelyben a státuszüzenetet tartalmazó `JLabel` középre igazítása közvetlenül nem lehetséges[13], ezért az üzenet – az ajánlásnak megfelelően – egy `JPanel` objektumra kerül, és ezt jeleníti meg az elrendezéskezelő
- *Sok* (ezres nagyságrendű) csúcsot tartalmazó grafika esetén a grafikus panelen elhelyezett listener-ek valamelyike `StackOverflowError`-t, rosszabb esetben `Exception`-t dob, melyre nem sikerült magyarázatot találnom
- *Sok* (ezres nagyságrendű) csúcsot tartalmazó grafika animációja során a csúcsok mozgatásának erőforrásigénye olyan mértékűre nő, hogy az a mozgatás

érzékelhető lassulását okozhatja

- futó animáció közben a fa-műveletek gombjai nem kerülnek letiltásra. Ha a felhasználó ekkor újabb elemet szúr be vagy töröl, akkor az animálást végző osztálynak egyszerre több példánya indul el, melyek felváltva jelenítik meg a saját tevékenységüket, illetve ha több animáció érinti ugyanazt a csúcsot, akkor `NullPointerException` váltódik ki. Ilyen esetben az animáció megszakad, de a *Befejezés* gomb megnyomásával a fa konzisztens állapota megjeleníthető, ezt követően pedig újabb műveletek hajthatók végre rajta. Mivel a hiba nem blokkolja az alkalmazás funkcióit, és nem okozza az alkalmazás leállását, prioritása alacsony, javítására (jelen verzióban) nem kerül sor.

5. Összefoglalás

A végeredmény

A program célja a 2-3-áris fák műveleteinek szemléltetése, amit az alkalmazás sikeresen megvalósít, anélkül, hogy a méretre vonatkozó megszorításokat tenne, miközben nagy szabadságot ad a felhasználónak a kezelni kívánt értékek kiválasztásában. Ezen kívül további, az adatszerkezettől független, mégis igen hasznos szolgáltatásokkal egészül ki, ilyen pl. a grafika képként való mentésének lehetősége, vagy egyszerre több értékkel való műveletvégzés. Mindazonáltal az adatszerkezet viselkedésének megértéséhez önmagában kevés, az elméleti háttérnek való utánajárást nem váltja ki.

Ráadásul a modell jelenlegi formájában csak szemléltetésre alkalmas, hiszen éltem azzal az egyszerűsítéssel, hogy a levelek csak kulcsot tárolnak, valódi struktúrát nem.

Továbbfejlesztési lehetőségek

Mind a modell, mint a vezérlő réteg az egyes csúcsok kulcsait „bal-” és „jobboldali” kulcsként, gyermekeit pedig „jobboldali-”, „baloldali-” és „középső” gyermekként kezeli. Ha ezek valamilyen gyűjtemény (vektor, lista) elemeiként lennének definiálva, akkor – a megfelelő metódusok analóg átalakításával – olyan fa készíthető, amelynek elágazási tényezője nem csak 2 vagy 3 lehetne, hanem tetszőlegesen sok, azaz az alkalmazás B-fák szemléltetésére is használható lenne.

A vezérlő réteg a fa kirajzolásakor csak a rajzolandó csúcsokat ismeri. Ez a függetlenség lehetővé teszi másfajta keresőfák kirajzolását is, ha ahhoz egy megfelelő modellt alkotunk. Ehhez érdemes lehet a modell és a vezérlő réteg közé olyan interfészt készíteni, amely garantálja, hogy a modellen csak értelmezhető művelet legyenek meghívhatók, illetve az csak a vezérlő számára értelmezhető eredményeket adjon vissza.

Hivatkozások

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Új Algoritmusok, Scolar, 2003, [992], ISBN: 0-262-03293-7

[2] Járai Antal: Bevezetés a matematikába, Eötvös Kiadó, 2005, [242], ISBN-963-463-729-9

[3] Dr. Fekete István: Bináris fák, http://people.inf.elte.hu/fekete/docs_1/binaris_fak.pdf, 2012-12-15

[4] Ivanyos Gábor, Szabó Réka, Rónyai Lajos: Algoritmusok, Typotex, 2008, [350] ISBN: 978-963-2790-14-5

[5] Dr. Fekete István: A 2-3-fák és B-fák, http://people.inf.elte.hu/fekete/docs_1/2-3fak_B-fak.pdf, 2012-12-15

[6] Dr. Fekete István: Keresések, http://people.inf.elte.hu/fekete/docs_1/binaris_kereso_fak.pdf, 2012-12-15

[7] Dr. Fekete István: Algoritmusok műveletigénye http://people.inf.elte.hu/fekete/docs_1/muveletigeny.pdf, 2012-12-15

[8] Sike Sándor: Tervezés és elemzés elmélete http://people.inf.elte.hu/sike/TervElem/tervelem.pdf, 2012-12-15

[9] Oracle dokumentáció http://docs.oracle.com/javase/1.5.0/docs/guide/language/generics.html, 2012-12-15

[10] Sike Sándor, Varga László: Szoftvertechnológia és UML, Eötvös Kiadó, 2003, [350], ISBN-963-463-587-3

[11] ISTQB Certified Tester Foundation Level – Course Material

[12] Oracle hibajegyzék http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4763448, 2012-12-15

[13] Oracle hibajegyzék http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4275005, 2012-12-15