Eötvös Loránd Tudományegyetem

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Rendezési algoritmusok szemléltetése

Márföldi Péter Bence programtervező informatikus BSc

Veszprémi Anna mestertanár

Budapest, 2015

Tartalomjegyzék

1. Bevezetés 1
   1. [A feladat és annak értelmezése 1](#bookmark2)
   2. [Alkalmazott technológiák 2](#bookmark3)
      1. [Java 2](#bookmark4)
      2. [JavaFX 2](#bookmark5)
      3. [JUnit 3](#bookmark6)
2. [Felhasználói dokumentáció 4](#bookmark8)
   1. [Rendszerkövetelmények 4](#bookmark9)
3. [Minimális rendszerkövetelmények 4](#bookmark10)
4. [Ajánlott rendszerkövetelmények 5](#bookmark11)
5. [Telepítés és eltávolítás 5](#bookmark12)

[Telepítés natív telepítővel 5](#bookmark13)

Hagyományos telepítés 6

Eltávolítás 6

* 1. [Felhasználói felület bemutatása 7](#bookmark14)

1. [Főmenü 7](#bookmark15)

[Eszköztár 7](#bookmark16)

Központi panel 8

1. [Bemenet megadása panel 8](#bookmark17)

Manuális 8

[Generálás 9](#bookmark18)

[Fájlból beolvasás 10](#bookmark19)

1. [Főpanel 10](#bookmark20)

[Eszköztár 11](#bookmark21)

Panel választó 12

Panel 12

1. [Megfigyelés panel 12](#bookmark22)

Algoritmus lista 12

[Állapotjelző táblázat 13](#bookmark23)

[Gombok 13](#bookmark24)

[Állapotjelző felület 13](#bookmark25)

1. [Összehasonlítás panel 14](#bookmark26)

[Elemzés táblázat 15](#bookmark27)

[Elemzés diagram 15](#bookmark28)

[2,3, A vizsgált algoritmusok 15](#bookmark29)

1. [Buborékrendezés 15](#bookmark30)
2. [Beszúró rendezés 17](#bookmark35)
3. [Shell rendezés 19](#bookmark38)
4. [Gyorsrendezés 20](#bookmark44)
5. [Kupaerendezés 22](#bookmark50)
6. [Versenyrendezés 23](#bookmark56)
7. [Eadix "előre" 25](#bookmark61)
8. [Eadix "vissza" 27](#bookmark65)
9. [Fejlesztői dokumentáció 29](#bookmark72)
10. [Tervezés 29](#bookmark73)
11. [Alapelvek 29](#bookmark74)
12. [Használt fejlesztőeszközök 30](#bookmark75)
13. [Felhasználói felület 30](#bookmark76)

Képernyőtervek 30

Főmenü 30

Manuális bemenet 30

Bemenet generálása 31

Fájl beolvasása 31

Főpanelek 31

Megfigyelés panel 31

Összehasonlítás panel 31

Felületek közötti navigálási lehetőségek 31

1. [Használati esetek 32](#bookmark77)
2. [Csomagszerkezet 33](#bookmark78)

[Modell csomag 34](#bookmark79)

algorithm alesomag 35

algorithm.raw alesomag 35

algorithm,raw,test alesomag 35

info alesomag 35

[Megjelenítő csomag 35](#bookmark80)

graph alesomag 36

[Vezérlő csomag 36](#bookmark81)

1. [Osztályszerkezet 36](#bookmark82)

[alapcsomag osztálya 37](#bookmark83)

[controller csomag osztályai 37](#bookmark84)

[algorithm csomag osztályai 39](#bookmark85)

[alogirtmh.raw csomag osztályai 40](#bookmark86)

[info csomag osztályai 40](#bookmark87)

[3,2, Megvalósítás 41](#bookmark88)

1. [Az eredeti terv módosítása 41](#bookmark89)

[Szálkezelés 41](#bookmark90)

[A JavaFX kibővítése 41](#bookmark91)

1. [A megvalósítás menete 42](#bookmark92)
2. [FXML állományok 43](#bookmark93)

[Néhány fontosabb FXML állomány 43](#bookmark94)

BaseController 43

OverviewLavout 43

BenehmarkLayout 44

1. Nem forrásfájl állományok 44
2. Osztályok leírása 44

[Modell réteg osztályai 44](#bookmark95)

SortingAlgorithm absztrakt osztály 44

ChartAlgorithm absztrakt osztály 45

GraphAlgorithm absztrakt osztály 45

EadixAlgorithm absztrakt osztály 45

SortingAlgorithmFaetory osztály 46

CounterData osztály 46

ReeursiveParameter osztály 47

BubbleSort osztály 47

InsertionSort osztály 48

ShellSort osztály 48

Quicksort osztály 49

HeapSort osztály 49

TöurnamentSort osztály 50

ForwardRadix osztály 51

BaekwardRadix osztály 51

BenehmarkData osztály 52

SortingThread absztrakt osztály 52

BubbleSortThread osztály 53

InsertionSortThread osztály 53

ShellSortThread osztály 53

QuickSortThead osztály 53

HeapSortThread osztály 54

TournamentSortThread osztály 54

InfoFlyWeight osztály 54

InfoFlyWeightFactory osztály 55

[Kontroller réteg osztályai 55](#bookmark96)

MainApplication osztály 55

MenuController osztály 56

ManualInputController osztály 56

KandomlnputController osztály 57

FilelnputController osztály 57

BaseController osztály 58

OverviewController osztály 58

OverviewListController osztály 59

OverviewChartController osztály 60

OverviewDoubleChartController osztály 60

OverviewGraphController osztály 60

SortingThreadListener interfész 61

BenehmarkController osztály 61

BenehmarkChartController osztály 62

[Megjelenítési réteg osztályai 62](#bookmark97)

Vertex osztály 62

Edge osztály 63

Graph osztály 63

1. [Tesztelés 63](#bookmark98)

[Felhasználó felület tesztelése 63](#bookmark99)

Főmenü eszköztára 63

Főmenü gombjai 63

Számok manuális megadása panel 63

Számok generálása panel 64

Fájl betöltése párbeszédablak 64

Főpanel eszköztára 65

Megfigyelés panel 65

Összehasonlítás panel 65

[Egységtesztelés 65](#bookmark100)

zeroElementQ 66

oneElementQ 66

fiveConcretElementQ 66

thousandRandomElement() 66

millionRandomElement 66

[Strukturális tesztelés 66](#bookmark101)

1. [Összegzés 69](#bookmark103)
2. [Irodalomjegyzék 70](#bookmark106)
3. fejezet Bevezetés

Az eddigi egyetemen töltött éveim során az Algoritmusok és adatszerke­zetek kurzus foglalkozott mélyrehatóan a rendezési algoritmusokkal. Számomra a tananyagból talán ez a témakör volt a legnehezebben elsajátítható. Ez sarkallt arra, hogy a szakdolgozatom témáját adják ezek az eljárások, szerettem volna biztos tu­dással rendelkezni ezen a területen.

Az bizonyos, hogy minden informatikus - beleértve a leendőeket is - tanulmányaik kezdetén találkoztak ezen eljárásokkal. Nagyszerű terület arra, hogy megérthessük mi a műveletigény, hogy mi számít igazán sok adatnak, vagy, hogy mit értünk egy algoritmus stabilitásán,

* 1. A feladat és annak értelmezése

A feladat egy oktatóprogram létrehozása, melynek segítségével az érdeklődő meg­értheti néhány rendezési algoritmus működését egy egyszerű, letisztult felületen. Le­hetőséget kell adni az algoritmusok lépésenkénti vizsgálatára, továbbá a műveletek számának összehasonlítására, A felhasználó több módon megadhatja a rendezendő pozitív egész számokat:

* begépeléssel
* generálással
* fájlból betöltéssel

A bemutatott algoritmusok lépéseit, valamint a rendezendő értékeket minél szem­léletesebben kell megjeleníteni, így nem elegendő csak a számok egymás utáni kiíra­tása az egyes lépésekben.

* 1. Alkalmazott technológiák

A Következőkben röviden összefoglaljuk a Java\8], JavaFX\9] és JUnit[10\ jel­legzetességeit.

* + 1. Java

A Java egy általános célú, objektumorientált programozási nyelv, melyet 2009- ig a Sun Microsystems fejlesztett, ezt követően pedig az Oracle. A szakdolgozatban használt 1,8-as verziót már az Oracle adta ki 2014-ben. A Java nyelv a szintaxisát a C és C++ nyelvektől örökölte, azonban utóbbitól eltérően egyszerű objektummo- dellel rendelkezik.

A Java platformra készült programok túlnyomó többsége asztali alkalmazás. Ma­napság egyre több helyen találkozhatunk a Java nyelven írt programokkal, például mobil eszközökön, banki rendszereknél vagy akár egy szórakoztató elektronikai esz­közön. Nagy előnye, hogy sok nyelvvel ellentétben platformfüggetlen, azaz egy adott platformról egy program minimális változtatással átültethető egy másik platformra.

A Java legfontosabb része a Java virtuális gép (JVM). A JVM-et sokféle be­rendezés és szoftvercsomag tartalmazza, így a nyelv egyaránt platformként és kö­zépszintként is működik. Összefoglalva a Java program három fontos szerepet tölt be:

* programozási nyelv
* köztes réteg (middleware)
* platform
  + 1. JavaFX

Olyan szoftverplatform, amelynek célja, hogy gazdag internetes alkalmazást le­hessen készíteni és futtatni eszközök széles skáláján. Eredetileg a Swing könyvtárat váltotta volna fel, azonban jelenleg mindkettő része a Javé SE-nek.

A 2,0-ás verzióig a fejlesztők egy külön nyelvet használtak, amelyet JavaFX Script-nek neveznek. Azonban mivel ez szintén Java bájtkódot generál a későbbiek­ben megadatott a lehetőség, hogy a programozók Java kódot használjanak helyette. A JavaFX egyik legnagyobb előnye, hogy egy egyszerű XML struktúrában leírhatók a program graűkus felületének összetevői, melyhez ezt követően elegendő az egyes interakciókhoz tartozó funkciókat implementálni.

Az elterjedtebb operációs rendszerek mindegyikét támogatja. Ahogyan előnye, úgy hátránya is a Swing-hez képest az, hogy jelenleg is folyik a fejlesztése, ezért oly­kor csak hosszas utánajárást követően sikerül megoldást találni egy-egv problémára.

* + 1. JUnit

Egv egységteszt keretrendszer a Java programozási nyelvhez. Az egységtesztek karbantartására, és futtatására kínál szolgáltatást. Gyakran a verzió kiadási folya­mat részeként szokták beépíteni, azaz egy kiadás akkor hibátlan, ha ezen tesztek mindegyike hibátlanul lefut. Egy 2013-as felmérésben[12] tízezer Java technológiát használó GitHub projektet vizsgáltak. A projektek csaknem harmadánál használták a JUnit-oí, ezzel az egyik leggyakrabban használt függvénykönyvtár volt a felmé­résben résztvevő projektekben.

1. fejezet

Felhasználói dokumentáció

* 1. Rendszerkövetelmények

Az elkövetkezőkben ismertetésre kerülnek a minimális és az ajánlott rendszerkö­vetelmények,

* + 1. Minimális rendszerkövetelmények

Mivel a program Java nyelven íródott, ezért elengedhetetlen, hogy a felhasználó számítógépén lehetőség legyen Java alkalmazások futtatására. Az alábbi operációs rendszereken érhető el a Java Runtime Enviroment 8v45-ös verziója:

* Windows 8
* Windows 7
* Windows Vista SP2
* Windows Server 2008 R2 SP1 (64-bit)
* Windows Server 2012 (64-bit)
* Mac OS X 10,8,3 vagy újabb
* Suse Linux Enterprise Server 10 SP2+, 11.x
* Ubuntu Linux 12,04 vagy újabb
* Red Hat Enterprise Linux 5,5+, 6.x
* Oracle Linux 5,5+; 6.x; 7.x

Hardverkövetelmények tekintetében a Java JRE 8 futtatásához szükséges mi­nimum követelményei az irányadóak. Azonban a program bizonyos esetekben több erőforrást is igényelhet, ezért ajánlott nagyobb memóriával és erősebb processzor rendelkező rendszer használata, A követelmények a következő táblázatban találha­tóak:

|  |  |
| --- | --- |
| Memória | 128 MB |
| Szabad lemezteriilet | 124 (+2) MB |
| Processzor | Pentium 2 266 MHz |

* + 1. Ajánlott rendszerkövetelmények

A szoftver tökéletes működéséhez a legelterjedtebb operációs rendszer, a Win­dows ajánlott. Továbbá követelmény 16:9-es képaránnyal rendelkező monitor, és leg­alább 1366x768 képernyőfelbontás használata.

A rendszer fejlesztése a következő ajánlott konfiguráción történt:

|  |  |
| --- | --- |
| Operációs rendszer | Windows 7 |
| Memória | 4 GB |
| Processzor | Intel Core Í5-2467M, 2000 MHz |

* + 1. Telepítés és eltávolítás

Alapvetően elegendő a Java futtatási környezet telepítéséről gondoskodnia a fel­használónak, ezen felül más program telepítésére nincs szükség. Azonban, mivel első­sorban Windows operációs rendszeren történő futtatásra lett felkészítve a program, ezt az operációs rendszert használók választhatják a kényelmesebb, natív telepítéses megoldást.

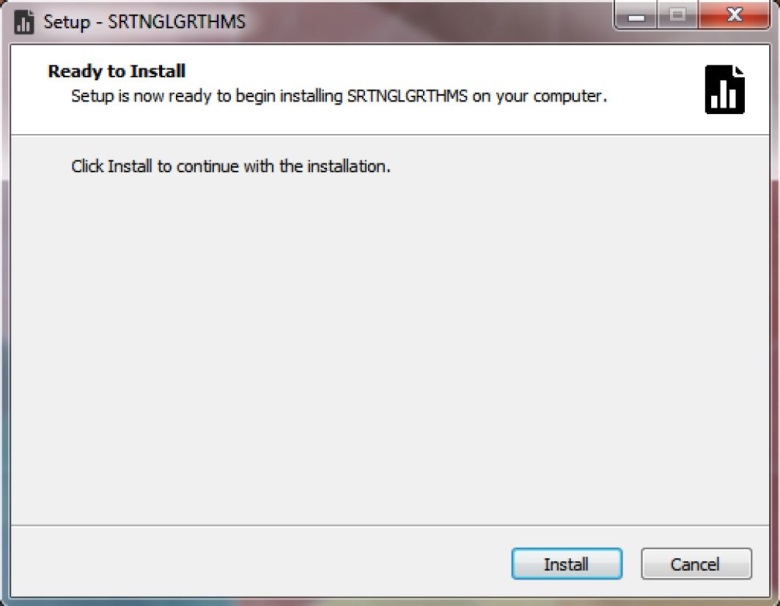
Elsőként a programhoz készült telepítővel történő konfigurálást vesszük végig, majd ezt követően a Java programokra inkább jellemzőbb, ám kissé körülményesebb telepítési mód kerül bemutatásra. Végül röviden összefoglaljuk a program eltávolí­tásához szükséges lépéseket.

Bármely módszert is szándékozik követni a felhasználó, először győződjön meg róla, hogy az előzőekben ismertetett rendszerkövetelményeknek megfelel a számító­gépe.

Telepítés natív telepítővel

Ez a telepítési mód csak a Windows-í használók számára érhető el.

Az első lépés a telepítési varázsló elindítása. A megjelenő párbeszédpanelon kat- tintsunk a Telepítés gombra. Ezt követően elindul a telepítés, ami körülbelül fél percet vesz igénybe.



2.1. ábra. A telepítési párbeszédpanel

A telepítési panel bezárása után máris megjelenik a program főmenüje, így meg­kezdheti a felhasználó a használatát. A későbbiekben történő futtatáshoz a következő könyvtárba szükséges navigálni: C:\Felhasználók\{Felhasználói név}\AppData \Local\SRTNGLGRTHMS\SRTNGLGRTHMS.exe

A fenti útvonal akkor érvényes, ha C:\ meghajtón található az operáeiős rend­szer, néhány rendszeren más lehet ennek a meghajtónak a betűjele. Továbbá egyes számítógépeken az AppData mappa rejtett lehet, így érdemes valamilyen fájlbön­gészőt, például Totál Commander-t használni.

Hagyományos telepítés

A most ismertetésre kerülő telepítési mód minden operáeiős rendszeren elérhető.

Az első feladat a Java virtuális gép telepítése, amely letölthető a következő web- eímről: http: //java. com/inc/BrowserRedirect 1. jsp

Fogadjuk el a lieeneszerződést, töltsük le a JRE telepítőfájlt. Ha frissíteni szeret­nénk a jelenlegi Java verziót a rendszerünkön, akkor előbb eélszerűbb eltávolítani a régi verziót. Mintán feltelepítettük a futtatási környezetet, készen áll a szoftver futta­tására a rendszerünk. A program könyvtárában található SRTNGLGRTHMS.jar fájl elindításával kezdhetjük meg a szoftver használatát.

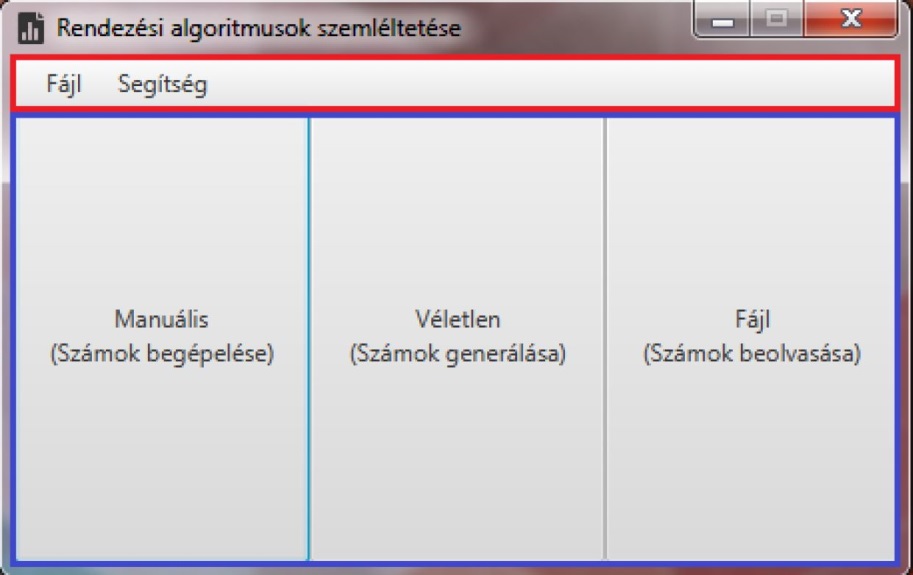
Eltávolítás

A program törlése az utóbb ismertetett telepítési mód esetében mindössze annyi­ból áll, hogy a SRTNGLGRTHMS.jar fájlt eltávolítjuk, valamint amennyiben a jövőben nines igény a Java futtatási környezet használatára, úgy a felhasználó eltá­volíthatja azt.

Window.s-on történő natív telepítést követően a program telepítési könyvtárában található unins000.exe fájlt futtatva, majd az Igen gombra kattintva a program törlődik a számítógépről.

* 1. Felhasználói felület bemutatása
     1. Főmenü

A program indítását követően megjelenik a főmenü, mely két komponensből áll.

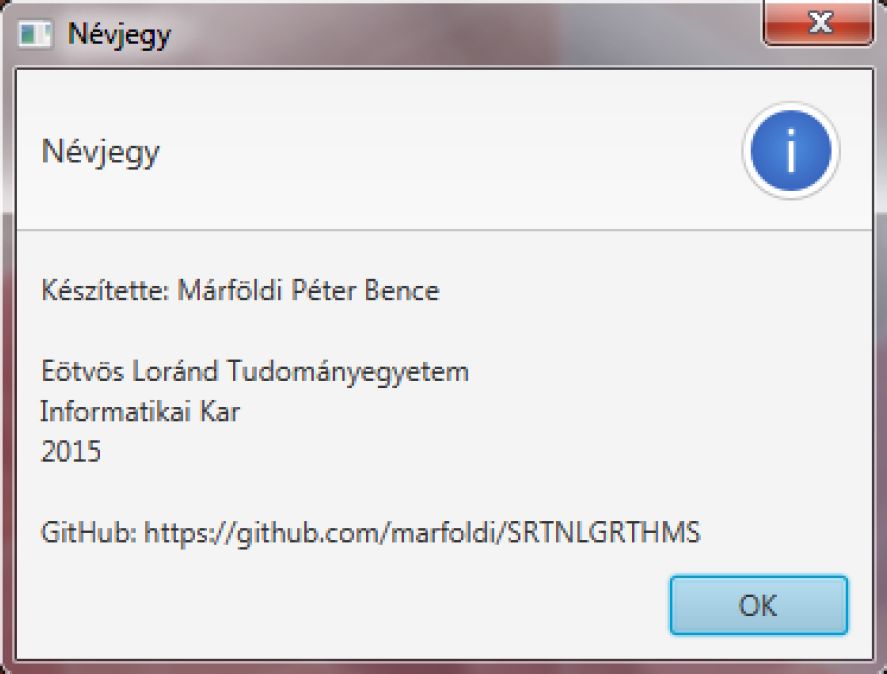


2.2. ábra. A program főmenüje

Az ablak felső részén található piros színnel jelölt rész az eszköztárat foglalja magában. A kék szín jelöli a főmenü központi paneljét, mely három gombból tevődik össze. Ezen gombokra történő kattintás után lehetőség nyílik a rendezendő számok megadására.

Eszköztár

Az eszköztárat két menüpont alkotja, Fájl és Segítség címszóval ellátva. Az előzőben a program bezárásának lehetősége kapott helyet, míg utóbbiban a szoftver névjegye tekinthető meg. Az Ok gomb lenyomásával bezárható a névjegy.



2.3. ábra. A program névjegye

Központi panel

A három gombból áll:

* Manuális (számok begépelése)
* Generálás (számok generálása)
* Fájl (számok beolvasása)

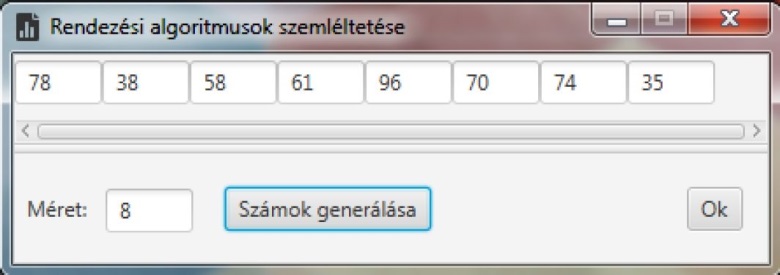
Ezek közül bármelyre kattintva átnavigálhatunk a bement megadását lehetővé tévő felületekre.

* + 1. Bemenet megadása panel

Az előzőekben említett három lehetőség közül választhat a felhasználó.

Manuális

A megjelenő panelen két gomb található, melyek kezdetben inaktívak. A "Mé­ret:" címke után található beviteli mezőbe egy pozitív egész szám megadásával és az ENTER billentyű leütésével megjelennek a számok bevitelére lehetőséget adó mezők. Ezt követően a panelen található két gomb már kattintható, a Számok generálása gombra kattintva 0 és 100 közötti véletlen számokkal töltődnek fel a mezők. Az Okra történő kattintás után megjelenik a program Főpanelje.



1. ábra. 8 elemű manuálisan megadott bemenet

Meg kell jegyezni, hogy ebben a módban legfeljebb száz érték adható meg, ennél nagyobb méretű bemenet manuális feltöltése túl körülményes lenne. Amennyiben több számot szándékozik megadni a felhasználó, válasszon a számok generálása vagy a fájlból történő beolvasás lehetőségek közül.

Generálás

A "Méret:" eímke mellett megadva a bemeneti számok mennyiségét, és a legör­dülő menüből kiválasztva a generálás módját az OK gomb kattinthatővá válik.

|  |  |  |
| --- | --- | --- |
| Pj Rendezési algoritmusok szemléltetése | | |
|  | | [KI |
| Méret: 1000000 | Véletlen generált ▼ |
|  | Véletlen generált |
| Majdnem rendezett |

Fordított Néhány egyedi

1. ábra. Egymillió véletlen generált érték megadása

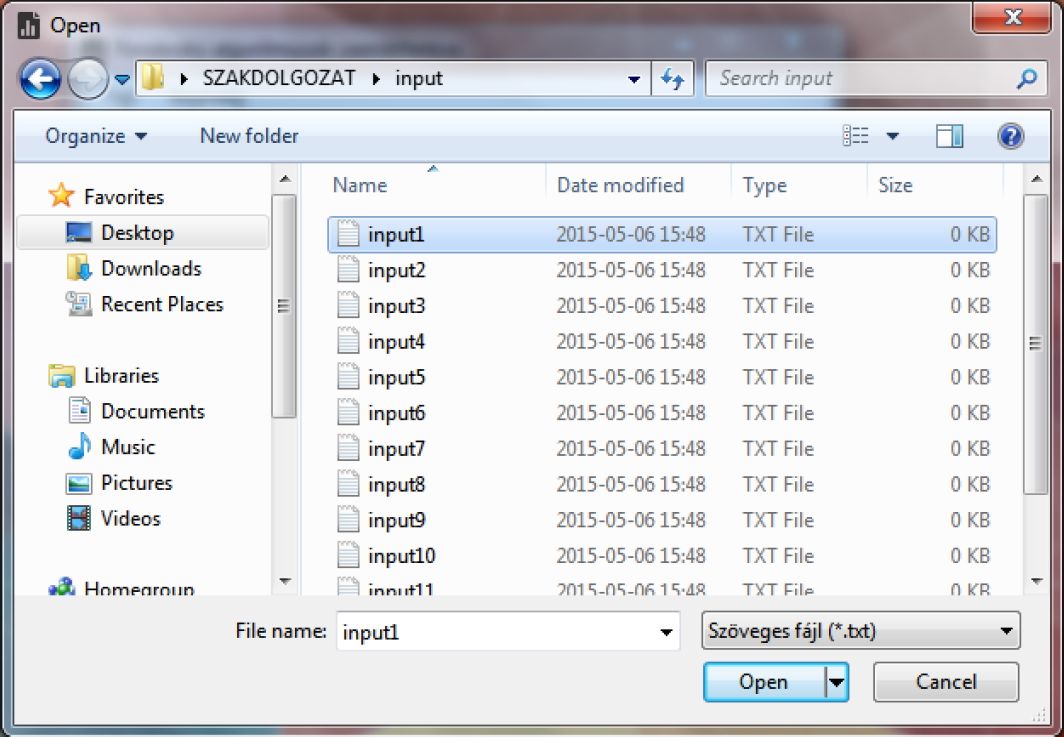
A legördülő menüből a következő négy típus közül lehet választani, azaz a gene­rált tömb legyen:

* Véletlen generált - véletlenszerűen választott számokból álljon
* Majdnem rendezett - a tömb 80%-a már rendezve legyen
* Fordított - a tömb legyen csökkenőleg rendezett
* Néhány egyedi - a tömb elemei között sok azonos érték szerepeljen

Itt megjegyzendő, hogy amennyiben az input mérete az 11,1001 intervallumban van, akkor a rendezendő számok a 0 és 100 közötti értékek közül kerülnek kiválasz­tásra és megjelenik a Megfigyelés panel. Ennek oka, hogy a túl nagy differencia az egyes értékek között sokat rontana az oszlopdiagramok megjelenésén. Ellenkező esetben a Java nyelv által definiált egész típus (Integer) maximális értékéig terjedhet a generált számok nagysága és csak a Összehasonlítás panel érhető el.

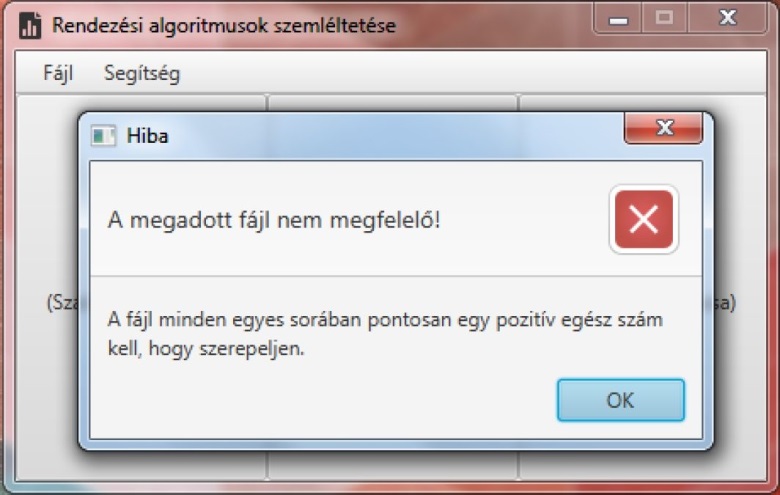
Fájlból beolvasás

A gombra történő kattintás után megjelenik egy fájltallózó. A tallózóban esak szöveges(txt kiterjesztésű) fájl választására van lehetőség.



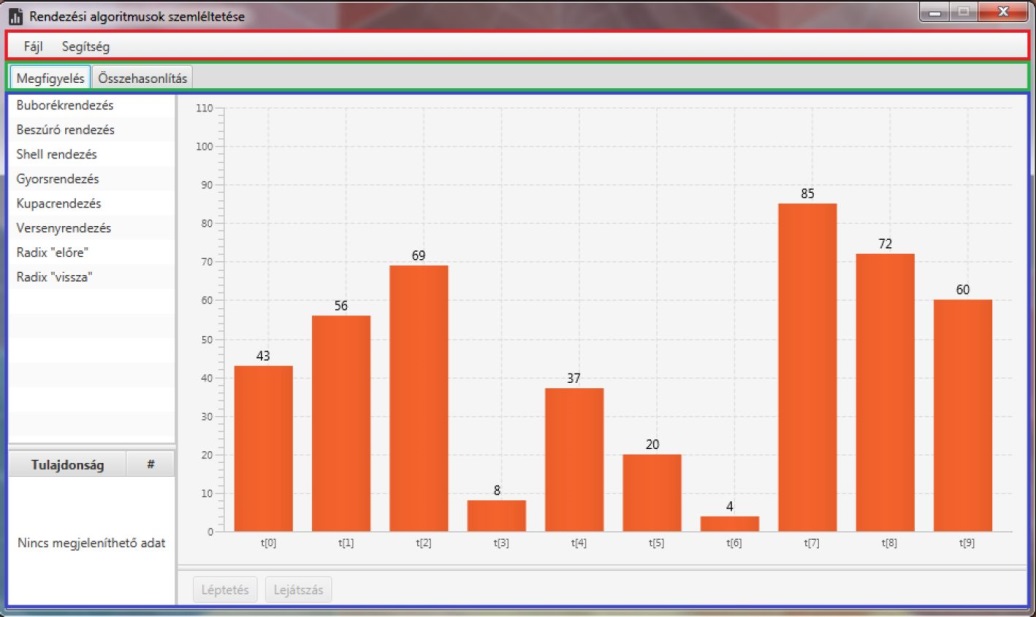
1. ábra. Fájltallózó

A fájl tartalmára történő megszorítások szigorúak. Minden sora legfeljebb egy pozitív egész számot tartalmazhat, ellenkező esetben a program hibaüzenet kísére­tében visszatér a főmenübe. Fontos, hogy az előzőek értelmében az üres sorok sem megengedettek.



1. ábra. Xem megfelelő fájl esetén a hibaüzenet
   * 1. Főpanel

A rendezendő számok sikeres megadása után megjelenik a program főpanelje, mely három logikai részből áll. A főpanel magában foglalja a Megfigyelés és Össze­hasonlítás paneleket, melyek összetettségükből fakadóan külön alfejezetekben ke­rülnek részletezésre.



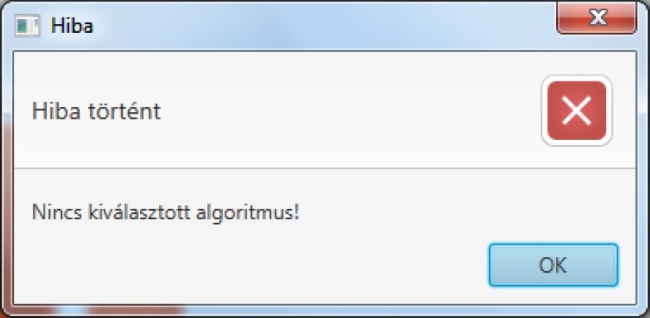
2.8. ábra. A program főpanelja

Eszköztár

A Főmenühöz hasonlóan itt is jelen van a piros színnel jelzett eszköztár sáv. Itt fontos kiemelni, hogy a Fájl és Segítség pontokon belül további alpontok is elérhetőek:

A Fájl menüpont bővül a Vissza a főmenübe lehetőséggel, melynek segítsé­gével a program újraindítása nélkül lehetőség van újabb rendezendő számsorozat megadására.

A Segítségre kattintva további lehetőségként választható Az algoritmusról pont. Itt elolvasható a rövid szöveges ismertetője a Megfigyelés panel listájából kiválasztott elemnek. Amennyiben pedig az Összehasonlítás panel aktív, akkor e panel táblázatából kiválasztott algoritmus leírása tekinthető meg. Az aktuális pane­len ha nem került kiválasztásra sor, akkor a menüpontra történő kattintás után fel­ugró ablak figyelmezteti a felhasználót arról, hogy e menüpont használatához előbb ki kell választani egy algoritmust.



2.9. ábra. Xines kiválasztott algoritmus hibaüzenet

Panelválasztó

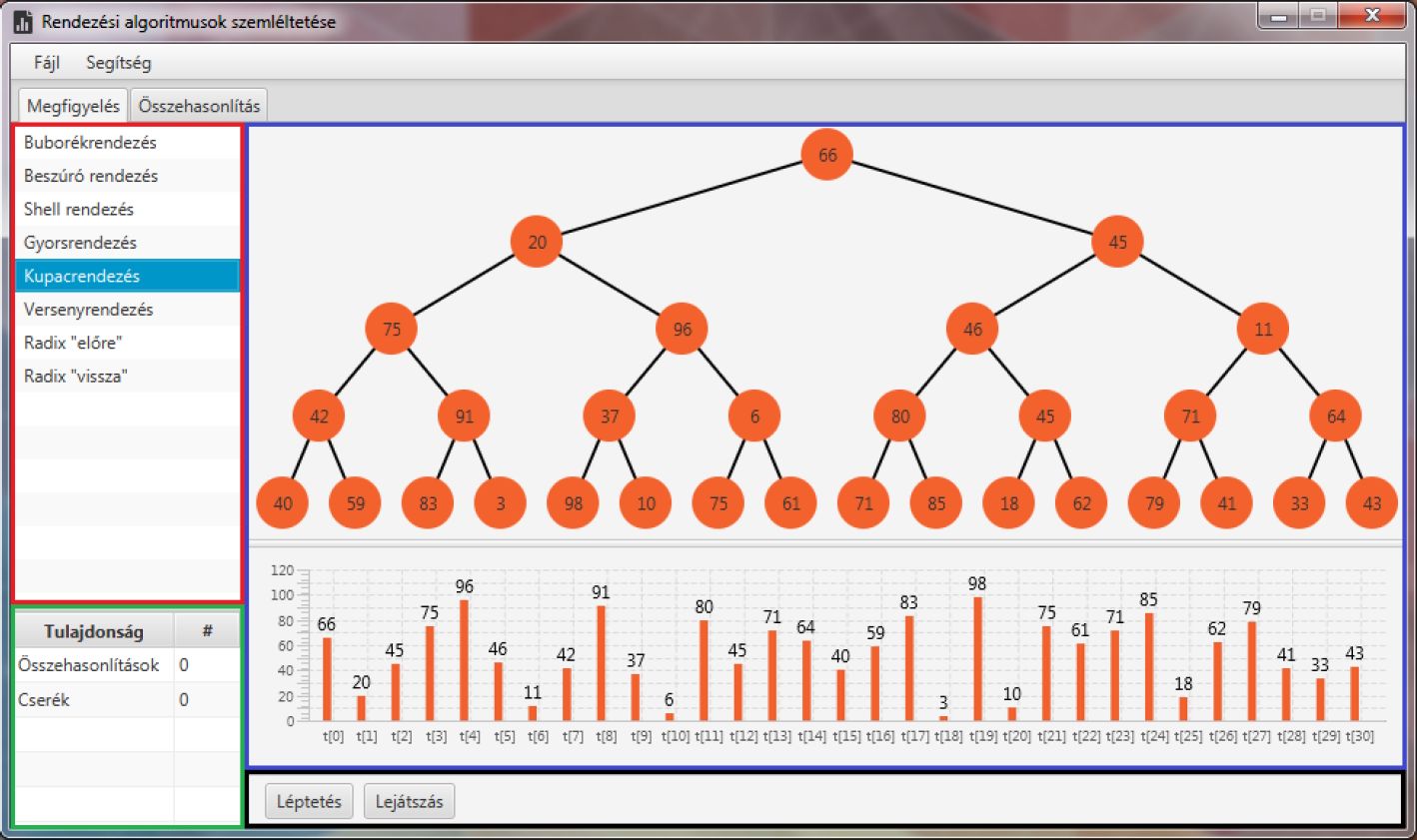
Amennyiben a bemenet megadásánál már ismertetett 11,100| intervallum magá­ban foglalja a bemenet hosszát, akkor két elem látható a zöld jelölt részen. így lehe­tősége van a felhasználónak navigálni a Megfigyelés és az Összehasonlítás panel között. Ellenkező esetben esak az utóbbi panel jelenik meg. Az aktuálisan megje­lenített felület neve a listában kék kerettel jelenik meg, valamint a másik elemhez képest világosabb szürke színnel.

Panel

A harmadik logikai egységet alkotják a panelek - kék színnel jelölve -, melyek közötti váltást a Panelválasztó teszi lehetővé. Mivel részletesebb leírást kíván a panelek ismertetése, ezért egy-egy külön alfejezetben kerülnek bemutatásra.

* + 1. Megfigyelés panel

A főpanel középső területén foglal helyet, négy komponensből tevődik össze. A felhasználónak itt nyílik lehetősége az egyes algoritmusok megfigyelésére, tanulmá- nvozására.



2.10. ábra. Megfigyelés panel

Algoritmus lista

Az ábrán piros színnel jelölt rész, melynek elemeire kattintva kiválasztható, hogy mely algoritmust szeretné a felhasználó vizsgálni. A kiválasztott elem kék háttérszínt kap, alapértelmezett esetben nines kijelölt elem.

Állapotjelző táblázat

A kiválasztott algoritmus aktuális állapotához tartozó információk jelennek meg a táblázatban. Az ábrán zöld kerettel van jelölve a komponens. Amennyiben nincs kiválasztott elem a Nincs megjeleníthető adat szöveg jelenik meg.

Két oszlopa a Tulajdonság és a #, mely utóbbi az értéket jelöli. Alapvetően az összehasonlításon alapuló algoritmusoknál megjelenő adatok az összehasonlítások és cserék vagy mozgatások száma. Az edényrendezéseknél pedig az aktuálisan vizs­gált bit indexe és a vizsgálatok száma jelenik meg. Egyes algoritmusokhoz a jobb megérhetőség miatt további állapotjelző értékek is tartoznak, melyek a következők:

|  |  |
| --- | --- |
| Algoritmus | Tulajdonság |
| Shell rendezés | Lépésköz |
| Gyorsrendezés | Vezérelem |
| Radix "előre" | Cserék száma |

Gombok

Alaphelyzetben a Léptetés és Lejátszás gombok inaktívak. A képen fekete keretet jelzi a helyüket. Amennyiben a felhasználó kiválaszt egy elemet az algoritmus listáról kattinthatóvá vállnak, A léptetéssel egy következő állapotot tekinthet meg a felhasználó. A Lejátszás gombra kattintva a program bemutatja az algoritmus működését. A felhasználó bármikor megállíthatja az animációt a Lejátszás gomb helyén található Megállítás gombra kattintva, majd ha kívánja innen folytathatja a vizsgálatot. Amennyiben a rendezés lezajlott, megjelenik az Újraindítás gomb, melynek megnyomásával az értékek visszakerülnek az eredeti helyükre.

Állapotjelző felület

A fenti ábrán kék szín jelöli ezt a területet. Az állapotjelző felületen minden esetben legalább egy oszlopdiagram foglal helyet. Ettől eltérően a Kupac- és Ver­senyrendezés kiválasztásakor egy gráf is helyet kap. Továbbá a Radix "vissza" rendezés második tömbjének reprezentálásához egy plusz oszlopdiagram is megje­lenik. Itt megjegyzendő, hogy a szemléltetéshez használt bináris fa legfeljebb 31 csúcsot tartalmazhat. Hosszabb bemenet megadásakor csak az első 31 elem vesz részt a megjelenítésben, erről felugró ablak tájékoztatja a felhasználót:



2.11. ábra. Megváltozott bemenetről a figyelmeztetőablakok

Az egyes műveleteket különböző színek is jelölik. Alapértelmezetten a rendezendő elemek színe a következőket jelentik:

Az alapértelmezett szín, nines kijelölve az elem.

® Az elem eserére vagy mozgatásra van kijelölve.

Az elemnek kitüntetett szerepe van.

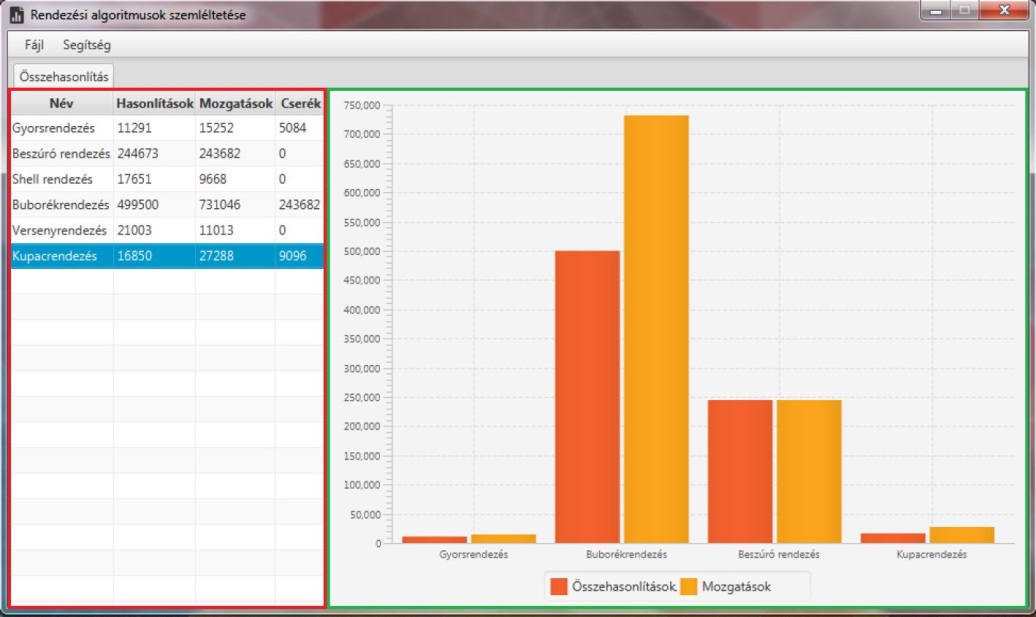
® Az elem már a végleges helyére került.

A • jelölés némi magyarázatra szorul. A kitüntetett szerepű elemnek számít például a gyorsrendezés vezéreleme, vagy a versenyrendezés fájának felépítésekor egy belső csúcsba kerülő elem.

Némely algoritmusnál a fentiektől eltérő lehet az egyes színek jelentése. A kö­vetkező fejezetben(2,3), az algoritmusok ismertetésénél jelölve van minden ilyesfajta különbség.

* + 1. Összehasonlítás panel

A Megfigyelés panellel megegyezően a főpanel középső részén található, két logikai egységből épül fel. Lehetőséged ad az összehasonlításon alapuló rendezések műveletigényeinek a vizsgálására.



2.12. ábra. Összehasonlítás panel

Elemzés táblázat

Piros kerettel jelölt rész az ábrán, mely táblázatnak négy oszlopa van:

* Név - az algoritmus neve
* Hasonlítások - a rendezés során végzett összehasonlítások szummája
* Mozgatások - az elemmozgatások számának összege
* Cserék - a végzett cserék szummája

A Mozgatások oszlop minden esetben kitöltésre kerül, még ha az algoritmus nem is mozgatásokat használ. Ekkor a mozgatások oszlopban a cserék számának három­szorosa jelenik meg. Ennél fogva egyszerűbb az algoritmusok vizsgálata.

Elemzés diagram

A táblázathoz képest balra helyezkedik el a zöld kerettel jelölt oszlopdiagram. Kezdetben teljesen üres, csak a jelölések jelentése látható. Az Elemzés táblázat egy során történő dupla kattintás következtében megjelenik az algoritmus összeha­sonlításainak és mozgatásainak a száma. Előbbi narancs- utóbbi citromsárga színnel. Amennyiben olyan sorra kattint a felhasználó, mely már látható a diagramon, azon algoritmus adatai eltűnnek a felületről.

* 1. A vizsgált algoritmusok
     1. Buborékrendezés

Leírás

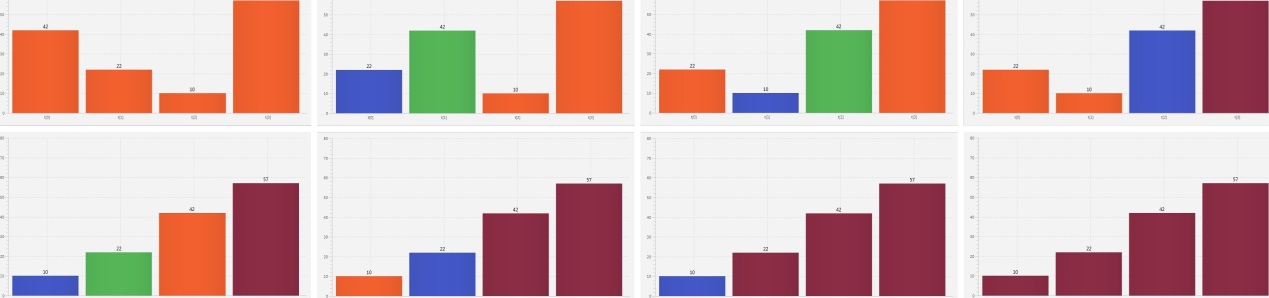
A legrégebbi és a legegyszerűbb rendezési algoritmus. Mindemellett a legtöbb esetben a leglassabb is. Már az 1965-ös évben megjelent egy teljes körű elemzése[4].

Működés

A rendezés minden egyes elemet összehasonlít a rákövetkező elemmel, és ha szük­séges megcseréli őket. Ez azt eredményezi, hogy lépésenként a maximális elem "bu­borék" szerűen a lista végére kerül, ezzel egyidejűleg a kisebb elemek "lesüllyednek" a tömb elejére. Amennyiben egy menetben a maximális eleme elérte a helyét vissza­vezetjük a problémát az eggyel "rövidebb" rendezési feladatra[l]. Az algoritmus javítható azzal, ha ügyeljük, hogy az egyes menetekben történt-e csere. Amennyi­ben egy olyan menet végére értünk, amelyben egy elem helye sem változott, akkor a tömb már rendezve van. A program az eredeti, nem javított verziót mutatja be.

Példa

A rendezendő számok: 42, 22,10, 57.

Az első ábrán szerepel a kezdeti állapot, a másodikon az első két értéket cseréje látható. A 42 ismét fentebb kerül egy pozícióval a harmadik ábrán. A negyedik ábrán csupán egy összehasonlítás történik, mivel az 57 nagyobb mint az őt megelőző érték. Az ötödik ábrán az előző állapot első két értékének(22 és 10) a cseréje látható. Az ezt követő összehasonlítás eredménye, hogy a második legnagyobb érték(42) a helyére került. Ezt követően már csak egy összehasonlítást történik, mely után az adatsor rendezve lesz.

2.13. ábra. Példa a buborékrendezésre

Műveletigény

Párosával haladunk végig az elemeken, és így vizsgáljuk őket. Mivel minden me­net végén visszavezetjük a problémát az eggyel rövidebb feladatra, ezért az összeha­sonlítások száma n hosszú bemenetre:

O(n) = (n — 1) + (n — 2) + ... + 1

n • (n — 1)  
2

2

n2 n

T — 2

0(n2)

A cserék száma már nem állandó, a bemenő adatok inverziószámával egyezik meg. Belátható, hogy minden egyes cserével egy inverzió szüntethető meg két elem között. A legtöbb cserét akkor szükséges eszközölni, ha a rendezendő elemek mind­egyike inverzióban áll a rákövetkező elemmel, azaz a tömb nagyság szerint csökkenő sorrendben rendezett. Ekkor a cserék száma:

MCs(n) = n ' (^ ^ = 0(n2)

Amennyiben a tömb elemei már rendezettek, akkor egyetlen cserét sem szükséges végrehajtani. Az átlagos csereszám a maximális cserék számának fele[l], ám nagy­ságrendileg még ez is 0(n2)

Jelölések az állapotjelző felületen

Kitüntetettnek ( színnel) jelöljük azt az elemet, amely egy csere következtében feljebb került. Kincs egyéb eltérés az eredeti jelölésekhez képest.

* + 1. Beszúró rendezés

Leírás

A legtöbb esetben akár kétszer hatékonyabb a bnborékrendezéshez képest) 1|, Az elve egyszerűen megérthető egy hétköznapi példán keresztül: A kezünkben tartunk kártyalapokat, majd egy pakliból húzva az új lapot beszúrjuk a kezünkben lévő, már rendezett lapok közé. Ennélfogva szokás kártyás rendezésnek is nevezni.

Működés

Két részre bontjuk a tömböt, az egyik részén - a tömb bal széle - a már rendezett, míg a másikon a még nem vizsgált elemek szerepelnek. Kezdetben a tömb első elemét tekintjük a rendezett résznek.

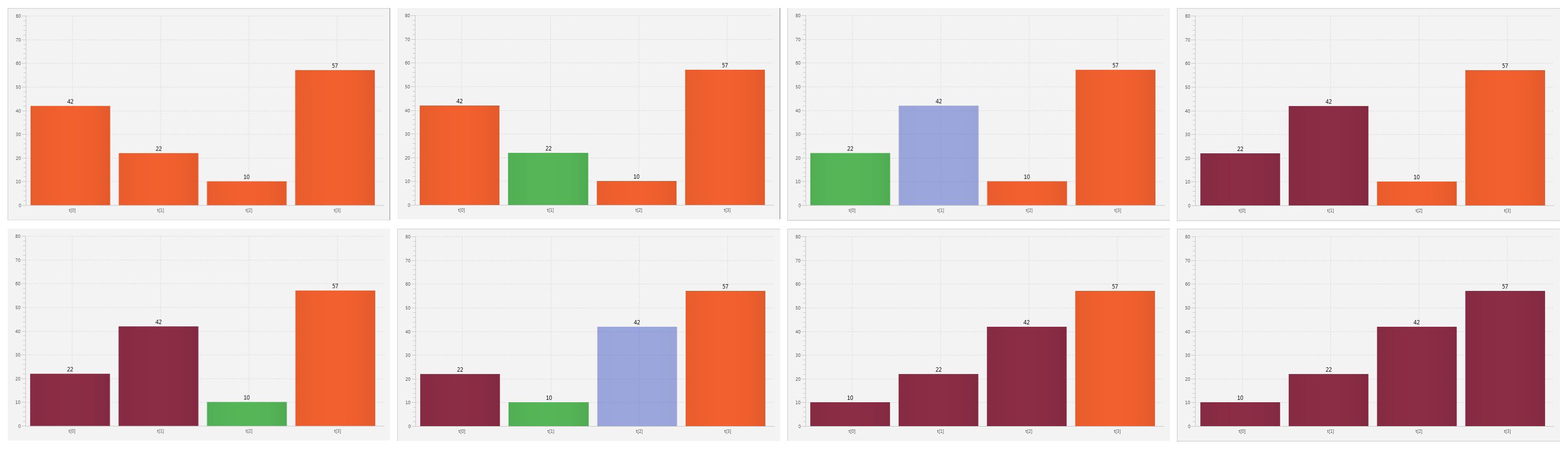
Lényege, hogy a soron következő elemet - a második elemtől kezdve - egy ideig­lenes változóba mentjük, és a rendezett tömbrész elemeit jobbra csúsztatjuk mind­addig, amíg a kiválasztott érték nem kerül a helyére. Ezt n-1 alkalommal ismételve megkapjuk a rendezett tömböt.

Példa

A rendezendő számok megegyeznek a bnborékrendezés bemutatásánál használ­takkal, azaz: 42, 22,10, 57.

Az első ábrán szerepel a kezdeti állapot. A következő képen jelöljük azt, hogy a második elemet fogjuk beszúrni. A harmadik ábrán felesúsztatjnk a 42 értéket, majd a negyedik képen már látható, hogy a tömb első két eleme alkotja rendezett tömbrészt. Ezt követően a harmadik elem kerül kiválasztásra. Az összehasonlítások eredményeképp a teljes rendezett tömbrészt jobbra tolódik, így az 7. képen már a tömb első három eleme megfelelő sorrendben van. Végül a tömb utolsó eleme kerül összehasonlításra az őt megelőzővel, mivel nagyobb tőle, így kész a rendezési feladat.

Műveletigény

****

2.14. ábra. Példa a beszúró rendezésre

Legjobb esetben elegendő a második elemtől mindegyik elemet az előtte található értékkel összehasonlítani, vagyis a tömb már eleve rendezett[3]. Ez pontosan n-1 összehasonlítást jelent, azaz:

mO(n) = (n — 1) = O(n)

Ebben az esetben pedig egyetlen elemet sem kell mozgatni.

A legrosszabb eset akkor áll fenn, ha a beszúrandó elem minden alkalommal kisebb a már beszúrt elemeknél, azaz a tömb elemei csökkenőleg rendezettek[l]. Ekkor az összehasonlítások száma:

n

MO(n) = £ i =

i= 1

(1 + (n — 1)) ■ (n — 1)

n ■ (” ~ 11 = O(n[[1]](#footnote-1))

Továbbá ekkor a mozgatások száma:

MM (n) = O(n2)

Jelölések az állapotjelző felületen

A már rendezett tömbrészt ® szín jelöli. Az aktuálisan beszúrandó elem pedig • jelölést kapja. A kiválasztott elemmel összehasonlított értékek • színt kapnak. Megjegyzendő, hogy a kitüntetett szerepű avagy beszúrandó elem nem szerepel tömbelemként az összehasonlításkor, csupán a felületen van jelölve a jobb megért- hetőség okán.

* + 1. Shell rendezés

Leírás

Donald Shell nevéhez fűződik, a legtöbb esetben a leggyorsabb négyzetes idejű algoritmus. Az elve az, hogy célszerű lehet előbb a "távolabb" lévő elemeket ha­sonlítani és mozgatni, mivel így az elemek hamarabb közel kerülhetnek a végleges helyükhöz.

Működés

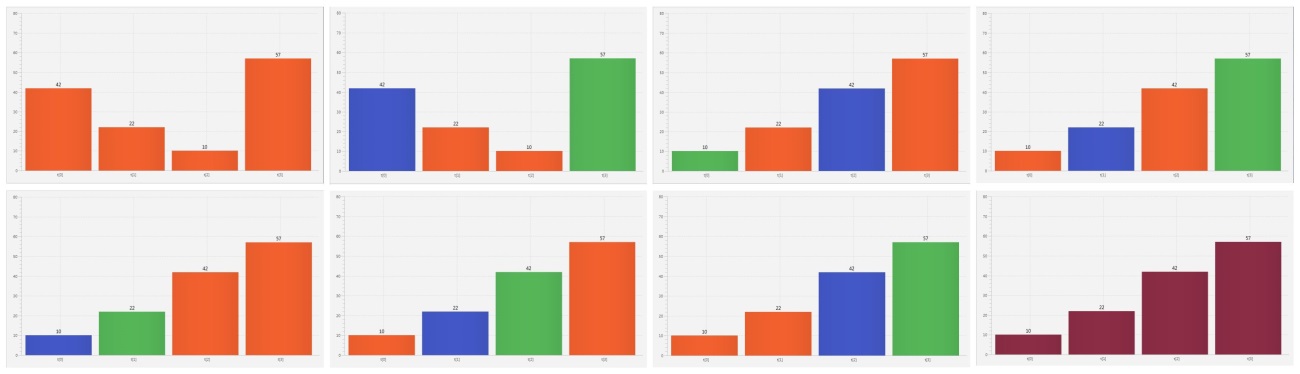
Többször vizsgálja a tömböt, és minden alkalommal egy részén beszúró rendezést hajt végre. Arra, hogy mekkora méretű résztömböt vizsgáljon az egyes lépésekben az algoritmus több javaslat is található. A teljesség igénye nélkül néhány ajánlás[5] erre vonatkozóan:

|  |  |  |  |
| --- | --- | --- | --- |
| Szabály (k=l...) | Konkrét értékek | Műveletigény | Szerző |
| \\_n/2k \ | n n i  |\_2 J , |\_4j , \* \* 1 | 0(n2) | Shell, 1959 |
| 2k - 1 | 1,3, 7, 11,\*\*\* | 0(n 2) | Hibbard, 1963 |
| 2p3q váltakozva | 1,2, 3, 4, \*\*\* | 0(n log2 n) | Pratt, 1971 |

Példa

Az előző példákban használt számokat kívánjuk rendezni ismét, azaz a bemenet: 42, 22,10, 57.

Az első képen látható a kezdeti állapot. A következő ábrán jelöljük azt, hogy a negyedik elemet fogjuk mozgatni, ekkor a lépésköz három. A harmadik képen már kettő a vizsgált elemek távolsága. A harmadik, kiválasztott elemet beszúrjuk az első elem helyére. A negyedik képen vesszük a következő elemet, továbbra is 2 lépésközzel, ekkor nem szükséges mozgatni. Végül az ötödik megjelenített lépéstől kezdve beszúró rendezést alkalmazunk.



2.15. ábra. Példa a Shell rendezésre

Műveletigény

A fentebbi táblázatból látható, hogy az algoritmus sebessége nagyban függ a lépésköz megválasztásától. A program a Vaughan Pratt által javasolt értékeket[6] használja, így legrosszabb esetben ©(n log2 n) a műveletek száma.

A legjobb eset akkor áll fenn, ha a tömb elemei már rendezettek, ekkor nincs szükség mozgatásra, az algoritmus futási ideje ©(n) nagyságrendű.

Jelölések az állapotjelző felületen

A felület bemutatásában szereplőkhöz képest nincs eltérés.

* + 1. Gyorsrendezés

Leírás

C.A.R. Hoare[7] alkotta meg 1965-ben. Az egyik leggyorsabb rendezési eljárás, ezért rendkívül gyakran alkalmazzák.

Működés

Helyben rendező, oszd meg és uralkodj [2] elven működő rekurzív algoritmus. A következő négy lépésre bontható fel az rendezés:

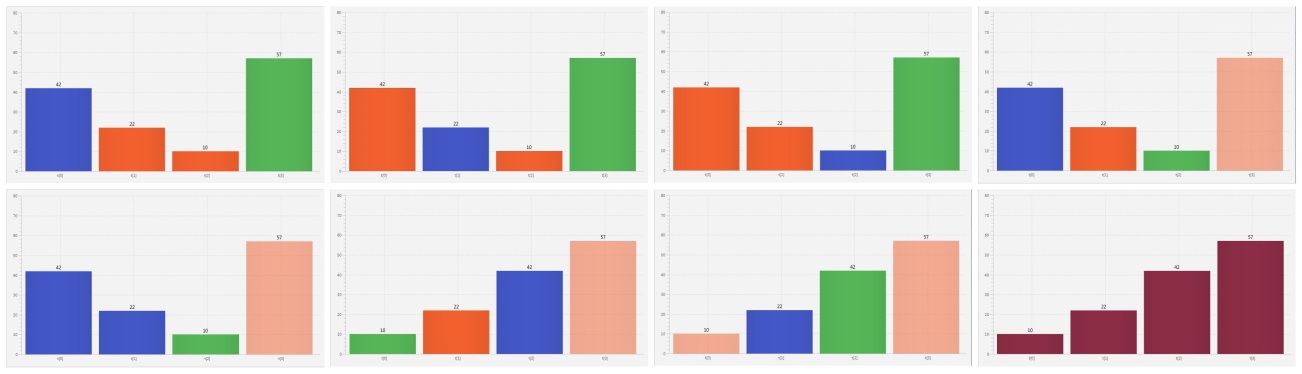
* Ha csak egy vagy nulla elemű az elemzett rész, akkor ne tegyünk semmit.
* Válasszunk egy vezérelemet (legjobb oldalibb elem).
* Osszuk két részre a rendezendő részt, mozgassuk a vezérelemtől kisebb eleme­ket, míg a nagyobbakat a tömb végébe[3].
* Rekurzívan ismételjük meg az előbbi lépéseket a résztömbökön.

Gyakorlatilag szétválasztást eszközölünk, melynek eredményeképp a kiválasztott elem­től kisebb értékek tőle balra, a nagyobbak pedig jobbra helyezkednek el. Az egysze­rűbb megérthetőséget szem előtt tartva a program mindig a legjobboldalibb elemet szelektálja.

Példa

Az eddigi példákkal megegyezően a bemenet: 42, 22,10, 57.

Az első képen látható a már kiválasztott vezérelem. A negyedik ábráig csak összehasonlítások történnek, mivel minden elem kisebb, mint a kiválasztott érték. A negyedik képen a három elemű résztömbön végezzük el a rendezést, a vezérelem ismét a legjobboldalibb érték. Az ötödik képen az első érték nagyobb mint a vezérelem, így ezt az elemet a hatodik képen megcseréljük a vezérelemmel. Ezt követően a középső két elem kerül összehasonlításra, mivel nincs szükség cserére így a rendezés befejeződött.



2.16. ábra. Példa a gyorsrendezésre

Műveletigény

A rendezés műveletigényét befolyásolja, hogy hogyan választjuk meg a vezér­elemet. Például a legrosszabb futási időt (0(n2)) eredményezi, ha mindig a legjob­boldalibb elemet választjuk vezérelemnek, és a tömb elemei csökkenő sorrendben vannak[2]. Éppen ezért a gyakorlatban javasolt ezen elem véletlenszerű megválasz­tása.

Ha feltételezzük azt, hogy minden rekurzív lépés felezi a tömböt, akkor egy n log(n) magasságú fával lehet ábrázolni a rekurziót. Továbbá, mivel minden szinten az elemek száma n, és ezek particionálásához használt lépésszám 0(n) így a legjobb esetben a futási idő 0(n • log(n)).

A gyorsrendezés a legtöbb esetben(közepes és nagy méretű bemenetre) a leg­megfelelőbb választás ha számít a rendezés sebessége, mivel az átlagos futási ideje - O(n • log(n)) - közel áll a legjobb futási idohöz[2].

Amennyiben a tömb elemei már eleve rendezettek vagy esetleg fordított sorrend­ben szerepelnek nem hatékony az eljárás.

Jelölések az állapotjelző felületen

A vezérelemet szín jelöli. Annak érdekében, hogy jobban átlátható legyen, hogy éppen melyik résztömbön folyik a vizsgálat az éppen nem vizsgált rész halová- nyabb sárga színnel van jelölve. A többi jelölés a felület bemutatásában leírtaknak megfelelően történik.

* + 1. Kupacrendezés

Leírás

A módszert J. W. J. Williams és R. W. Floyd javasolták 1964-ben|3|, Az O(nlog(n)) algoritmusok közül az egyik leglassabb, azonban előnye a gyorsrende­zéssel szemben, hogy nem erőteljesen rekurzív. Ennél fogva jól alkalmazható milliós nagyságrendű bemenetre. Ahogyan a neve is sugallja, a rendezéshez egy kupáé adat­szerkezetet használ. Az algoritmus ismertetése előtt definiáljuk a kupáé fogalmát|l|: Olyan bináris fa, amelyre a következők teljesülnek:

* Kizárólag a levelek szintjén hiányozhat csúcs, azaz "majdnem teljes".
* A levélszint csúcsai balra tömörítettek.
* Minden belső csúcs értéke nagyobb vagy egyenlő, mint a gyerekeinek értékei.

A második pont értelmében egyetlen olyan csúcs lehet, amelynek csak egy gyereke van, és az közvetlenül a levélszint felett kell, hogy elhelyezkedjen.

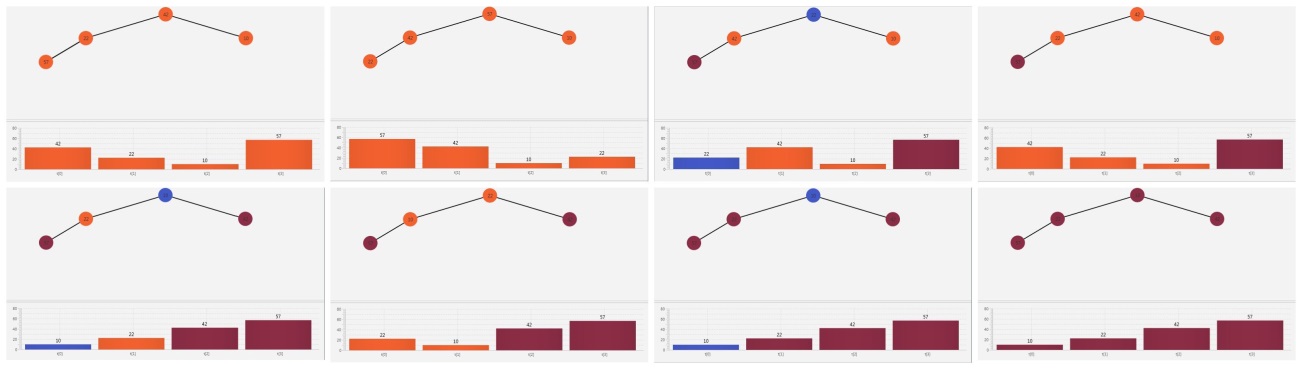
Működés

A rendezés az előbbi tulajdonságokra támaszkodik. Az algoritmus a bemeneti elemekből kupacot épít, majd a legfelsőbb elemét áthelyezi a kupac "végére". Ezt követően ellenőrzi a kupac tulajdonságokat, ahol szükséges cseréket hajt végre, hogy helyreálljon a kupac adatszerkezet, ekkor már az utolsó, legjobboldalibb levélelem nem vesz részt a kupacépítésben. A gyökérben található elemet a legjobboldalibb le­vélelem elé helyezi, és újraépíti a kupacot. Ezen lépések addig ismétlődnek, amíg már a maximális elem áthelyezése nem lehetséges, a kupac "végére" helyezési művelet elérte a gyökeret.

Példa

Az eddigi példákkal azonosan a rendezendő számsorozat: 42, 22,10, 57.

Az első kép a kezdeti állapotot mutatja. Ezt követően a bemeneti adatokból kupacot építünk, melynek eredménye a második ábra. A gyökérelemet a levélszint utolsó elemével megcseréljük a harmadik képen. Ellenőrizzük, hogy teljesülnek-e a kupac tulajdonságok, amennyiben nem cseréket hajtunk végre, ennek eredménye látható a 4. képen. Ismét a gyökérelemet lesüllyesztjük, majd ellenőrizzük a kupac tulajdonságot. Az előző két lépést megismételve rendezett tömböt kapunk.



2.17. ábra. Példa a kupacrendezésre

Műveletigény

A kupacrendezés egyik további, hogy míg a gyorsrendezésnek legrosszabb esetben a futási ideje O(n2), addig itt a futási idő továbbra is O(n• log(n)). Ennél fogva azon rendszereknél, ahol a négyzetes futási idő elfogadhatatlan inkább kupacrendezést alkalmaznak.

Jelölések az állapotjelző felületen

Az éppen összehasonlított értékeket • jelöli. Amennyiben csere történt a belső csúcsba kerülő érték háttérszíne • lesz. Ez alól kivétel, ha a gyökérbe került új érték, mivel ekkor már tudjuk az új legnagyobb értéket, ennek a színe ® lesz. Továbbá akkor is ezt a színt használjuk, ha már a tényleges, végleges helyére került egy elem.

* + 1. Versenyrendezés

Leírás

A maximum-kiválasztó rendezések közé tartozik, minden egyes menetben kivá­lasztja a legnagyobb elemet, kiírja és végül eltávolítja. A maximum kiválasztásnak a gyakorlati hátterét a sportesemények lebonyolítási rendje adja, azaz meghatározza az elemek között a "nyertestül]. A módszert n=2k inputhossz esetén érdemes al­kalmazni, mivel ettől értérő bemenetre sokkal kedvezőbb eredményt lehet elérni a kupacrendezéssel[l].

Működés

Az algoritmus által használt adatszerkezet egy teljes bináris fa. A bináris fa leveleiben szerepelnek a rendezendő elemek. Az első speciális menetben a fa belső pontjait kitöltjük, úgy, hogy a pontba a gyerekei közüli nagyobb érték kerül.

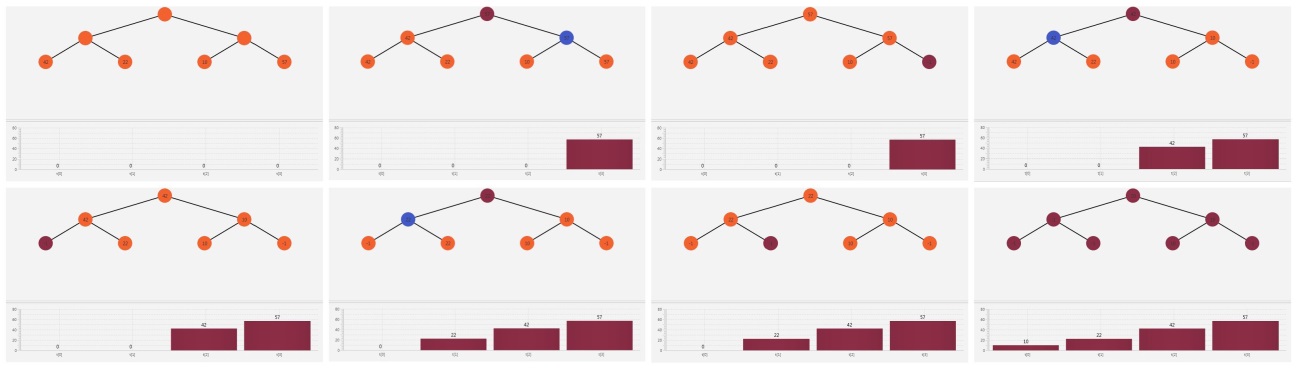
Ezt követően kerül sor az (n — 1) egyszerűbb menetre: A gyökérben található elemet keresve ,flefelé,f haladunk a bináris fában, majd megtalálva azt a levelet amelyben a gyökér értéke szerepel egy abszolút vesztest állítunk a helyére. Ez az érték a programban -1, mivel csak pozitív egészeket használunk a rendezések szem­léltetésére. Ezzel ellentétben a gyakorlatban ez az érték —ro. Majd ezen az "ágon" újrajátsszuk a mérkőzéseket.

Amennyiben a bemenet hossza nem kettő hatvány a program -1 értékekkel tölti fel a bináris fa további leveleit, amíg a bemenet hossza nem lesz megfelelő.

Példa

Az eddigi példákkal megegyezően a bemenet: 42, 22,10, 57.

Az első ábra a kezdőállapotot mutatja. A második kép a versenyfa kitöltését követő állapot, melynek eredményeképp megjelenik az abszolút maximum érték a diagramon. Ezt követően a harmadik képen látható a maximális értékhez tartozó levélelem megtalálásának állapota. A negyedik ábra szemlélteti az újrajátszás ered­ményét. Az előzőekhez hasonlóan szemléltetik a további képek a maximum érték megjelenítését, a levélelem megkeresését majd az újrajátszás eredményét.



2.18. ábra. Példa a versenyrendezésre

Műveletigény

A rendezés egyetlen hátránya a tárigénye, n szám esetén további n — 1 mezőre szükség van a versenyfa elkészítéséhez. Éppen emiatt a gyakorlatban nem sűrűn használt eljárás. Az első, speciális menet, a versenyfa kitöltése n — 1 összehasonlítást és mozgatást használ (n — 1a belső csúcsok száma). Minden további menetben a fán kétszer kell végigmenni, melynek magassága log2 (n). Egyszer a maximális levél megtalálásához, majd az újrajátszáshoz, így a ezen menetek 2log2(n) összehason­lítást végeznek. Mozgatás csak a második, újrajátszási művelethez tartozik. így a művelet igények:

O(n) = n — 1 + (n — 1) ■ 2 ■ log2(n) = 0(n ■ (log(n))  
M(n) = n — 1 + (n — 1) ■ log2(n) = 0(n ■ (log(n))

Jelölések az állapotjelző felületen

Az összehasonlításokat, valamint a maximális levélelem megkereséséhez bejárt utat • szín jelöli, A meccsek lejátszásakor a belső' csúcsba kerülő érték • jelölést kap. Ez alól kivétel, amikor a gyökérbe kerül egy meccs győztese, ekkor ® lesz az elem színe, továbbá akkor is ez a jelölés, amikor megtaláltuk a maximális levélcsú­csot,

* + 1. Radix "előre"

Leírás

Az előzőekben ismertetett algoritmusok mindegyike összehasonlításon alapuló rendezés, A radix rendezés viszont az edényrendezések közé tartozik. Ezen rendezé­sek nem hasonlítják össze az elemeket, hanem az elemek a rendezés során az érté­küknek megfelelő edényekbe kerülnek. Az edény rendezések eredményeként rendezett adatsorozatot kapunk lineáris időben|3|, A radix rendezés egy rekurzív algoritmus, melynek minden szintjén létrejönnek az edények.

Az általános edényrendezés egy speciális változata a radix előre rendezés, bináris, d hosszú számokra.

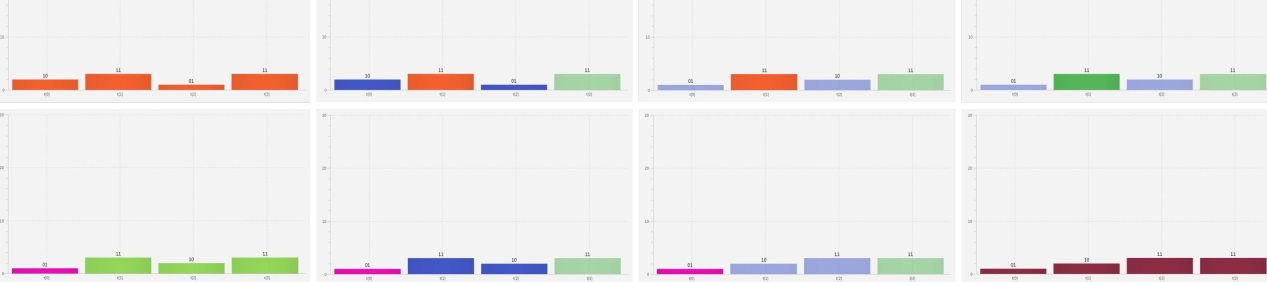
Működés

Az első menetben a rendezendő elemek első bitjét vizsgálja az algoritmus, A vizs­gálat két mutatóval történik, melyek a tömb két végéről indulnak, A tömb elején addig halad a mutató, amíg a vizsgált elem első jegye nem 1, ezzel párhuzamosan a tömb végén olyan elemet keres a másik, melynek első jegye 0, Amennyiben talált ilyen elemeket megcseréli őket. Ezt mindaddig folytatódik, amíg a két mutató nem találkozik. Ekkor kialakul két edény, az elsőben a 0-ás kezdőbittel rendelkező szá­mok, míg a másodikban az 1-essel kezdődő elemek foglalnak helyet. Ezt követően a második bit kerül vizsgálatra az "aledényekben", az előzővel azonos módon, A rendezés befejeződött, ha minden számjegy szerinti vizsgálat megtörtént, vagy ha mindegyik, a futás alatt kialakult edény már csak egy elemet tartalmaz.

Példa

Az rendezendő számok: 2, 3,1, 3, azaz binárisan 10,11, 01,11.

Az első ábra a kezdőállapotot mutatja. Mivel az első érték valószínűleg nem megfelelő helyen van, ezért a tömb végéről kezdve keresünk egy olyan értéket, amely 0-ás jeggyel kezdődik. A harmadik elem pont ilyen, a második képen ez a két érték cserére van kijelölve, majd a negyedik ábrán megtörténik a két elem cseréje. Az ötödik képen látható a menet során kialakult két edény. Mivel az első edény egy elemű, így nem szükséges a vizsgálata. A második edényben az 10 értéket szeretnénk cserélni, így az edény végétől haladunk előrefelé, amíg nem találunk olyan elemet, melynek második bitje egy. A 11 érték éppen ilyen, ezért megcseréljük őket. Az utolsó ábrán látható a rendezett sorozat.



2.19. ábra. Példa a radix ,felőre,f rendezésre

Műveletigény

A rendezés lineáris időben történik, a számjegyek számát jelölje d, ekkor belát­ható, hogy a legrosszabb esetben is, azaz ha az összes szám minden bitjét meg kell vizsgálnunk az algoritmus futási ideje:

T(n) = ©(d • n) = ©(n)

Jelölések az állapotjelző felületen

Az éppen vizsgált elem színnel van jelölve. Amennyiben két elemet fel kell cserélni ® háttérszínt kapnak. Továbbá a már vizsgált elemek háttérszíne fakóbb lesz. Amennyiben egy új edény keletkezik annak a színe véletlenszerűen választódik

ki.

1. Radix "vissza"

Leírás

Az algoritmus rövid ismertetője megegyezik a Radix "előre" rendezés leírásá­val.

Működés

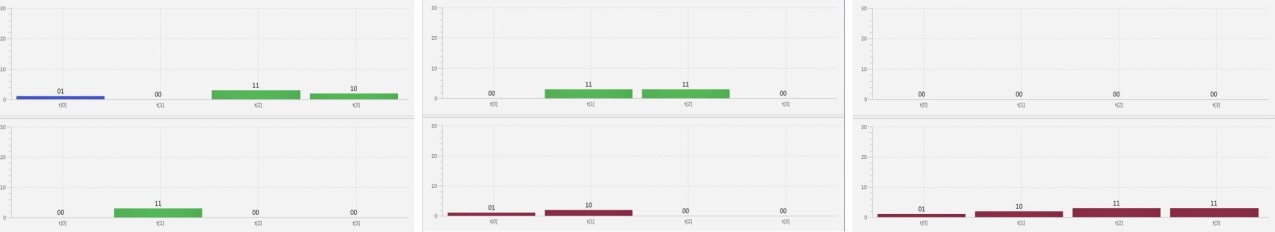
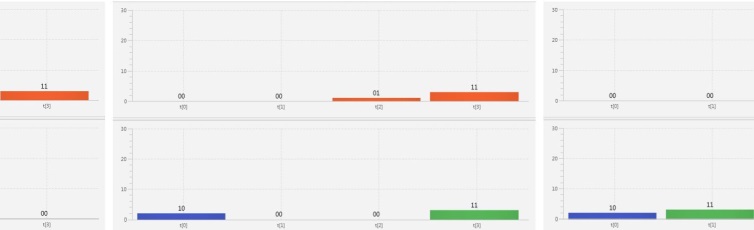
Az előző algoritmustól eltérően már nem helyben rendez, az eljárásnak két tömbre van szüksége. További különbség, hogy a kisebb helyiértéktől a nagyobb felé halad a vizsgálat.

Amennyiben az aktuálisan vizsgált bit értéke 0, akkor a "második" tömb elejére, ellenkező esetben a tömb végére töltjük át az aktuális bináris számot. Ezáltal minden egyes menetben két edény keletkezik: egy, melyben található számok aktuális bitje 0, s egy másik, melyek vizsgált jegye 1-es, Ezt követően a 0-ás edényt az elejéről olvasva feltöltjük újra az eredeti tömböt az előbb ismertetett módon, majd a 1-es edényt elemeit az utolsó elemtől visszafelé haladva töltjük át az értékeket. Amennyiben a rendezendő bináris számok d hosszúak, úgy d+1 áttöltést követően rendezett tömböt kapunk.

Példa

A bemenet megegyezik a radix "előre" példájában használt számokkal, azaz: 2, 3,1, 3, amelyek binárisan a 10,11, 01,11 értékek.

Az első ábra a kezdeti állapotot mutatja. A második képen a tömb első két eleme már áttöltésre került, a 10 az alsó tömb elejére, míg a 11 a végére. A harmadik ábrán látható a teljen áttöltött állapot. A következő ábrán visszatöltésre került az első érték, melynek első bitje 1, így az eredeti tömb végére került. Ekkor a második edény értékeit hátulról előrefelé vizsgáljuk, így a 10 érték kerül először az eredeti tömb elejére, majd a 11 a 10 előtti helyre. Végül az utolsó 11-es érték is bekerül az eredeti tömbbe. Ezt követően az 5. ábrán már csak átmásolásra kerülnek az értékek a megfelelő sorrendben, azaz az egyes edény tartalma előröl olvasva, míg a második edény értékei hátulról előrefelé haladva.

2.20. ábra. Példa a radix "vissza" rendezésre

Műveletigény

Lineáris idejű a rendezés, amennyiben d jegyűek a rendezendő számok az algo­ritmus futási ideje:

T(n) = 0(d • n) = 0(n)

Jelölések az állapotjelző felületen

Az aktuálisan vizsgált bit értékének megfelelően, amely bináris szám a 0-ás edénybe kerül míg amely az 1-es edénybe kerül az háttérszínt kap.

1. fejezet

Fejlesztői dokumentáció

* 1. Tervezés

A dolgozat fő célja egy olyan elsősorban hallgatóknak szánt program létrehozása, amellyel néhány rendezési algoritmus működése egy letisztult és egyszerű felhasználó felületen keresztül tanulmányozható,

A fejlesztés során több szempontot is űgyelembe kell venni, úgy mint: művelet­igény, memóriaigény, jó megjelenés, egyszerű kezelhetőség, és átlátható-, bővíthető kód készítése. Mivel ezen kritériumok közül több is csak egy másik rovására javít­ható, ezért a tervezés során kompromisszumokat kell kötni. Továbbá fel kell készülni arra, hogy az eredeti terven a fejlesztés során módosításokat kell végezni, mivel egv- egv probléma megoldása más megközelítést kívánhat,

* + 1. Alap elvek

A programnak három jól elkülönülő komponensből kell állnia: Egy logikai(modell) részből, ami gyakorlatilag a rendszer "motorja", itt kell, hogy történjen mindenféle számítási és adattárolási művelet. Egy megjelenítési rétegből, amely a logikai rész eredményeit jeleníti meg a felhasználó számára. Végül pedig egy kontroller szintből, amely kapcsolatot teremt a logikai- és a megjelenítési réteg között, A gyakorlatban ezt a fajta tagolást nevezik Modell-Nézet-Vezérlő (MVC) tervezési mintának.

Az elsődleges szempont az, hogy a felhasználó könnyedén tudja kezelni a progra­mot, és segítségével megértse az algoritmusok működését, így a felhasználói felület áttekinthetőségére és letisztultságára nagy hangsúlyt kell fektetni. Továbbá fontos az is, hogy a jövőben további rendezési eljárásokat is könnyedén meg lehessen jeleníteni a jelenlegiek mellett, így fontos a kód egyszerű bővíthetősége.

* + 1. Használt fejlesztőeszközök

A fejlesztés Edipse. SDK 4-4 fejlesztői környezet keretei között történik. A prog­ram grafikus fejlesztői felületet ad alkalmazások készítéséhez.

A kódolást segítő funkció volt a kódkiegészítés, továbbá az egyik beépített pro­jektmenedzsment eszköz (EGit).

A fejlesztéshez elengedhetetlen a Java SE 8u40 vagy magasabb verziójú szoft­ver. Továbbá a fejlesztést nagyban elősegíti a JavaFX Scene Builder 2.0, melynek segítségével egyszerűen megtervezhetővé válnak a grafikus felület komponensei.

Végül a telepítési környezet létrehozásához Ant és InnoSetup eszközök kerül­nek felhasználásra. Az egész projekt, beleértve e dokumentumot is megtalálható, és az egyes verziók visszakövethetők a GitHub-on: <https://github.com/marfoldi/> SRTNGLGRTHMS

A program fejlesztése során egyedüli külső függ vény könyvtár a JUnit, melynek segítségével egységtesztek készülnek.

* + 1. Felhasználói felület

Képernyőtervek

A program főmenüjében három gomb foglal helyet, melyekre kattintva megadha­tók a rendezendő számok. Továbbá az eszköztárban a Fájl és Segítség menüpontok foglalnak helyet, előbbire kattintva lehetőség nyílik a program bezárására, utóbbiban pedig megtekinthető a szoftver névjegye.



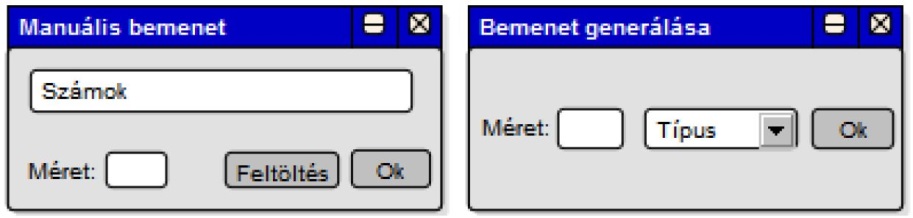
3.1. ábra. Főmenü látványterv

Főmenü.

Manuális bemenet. Az ablakon a Méret: címke mellett található mezővel defini­álható a bemenet hossza. A megadását követően megjelennek a hossznak megfelelően beviteli mezők jelennek meg a Számok címkével jelölt részen. A Feltöltés gomb segítségével véletlen számokkal töltődnek fel az előbb említett beviteli mezők. Az Ok gombra történő kattintás után megjelennek a Megfigyelés és Összehasonlí­tás panelek.

Bemenet generálása. A Méret címke után megadható a kívánt bemenet hossza. A Típus legördülő listából különböző generálási beállítások választhatóak ki. Az Ok gomb megnyomásával megjelennek a program főpaneljei.

Fájl beolvasása. A fájlból történő beolvasáshoz nem készült látványterv, mivel egy egyszerű fájltallózó segítségével van lehetősége a felhasználónak számokat ilyen módon megadni.



3.2. ábra. Bemenet megadása látványtervek

Főpanelek. A főpanelek eszköztárában megtalálhatóak a Főmenü pontjai, to­vábbá lehetőség van az előbb említett felületre történő visszatérésre. A Segítség menüpontban pedig lehetőség van az algoritmusok rövid leírásának megtekintésére. Valamint két gomb segítségével lehet navigálni a Megfigyelés- és Összehasonlítás panelek között.



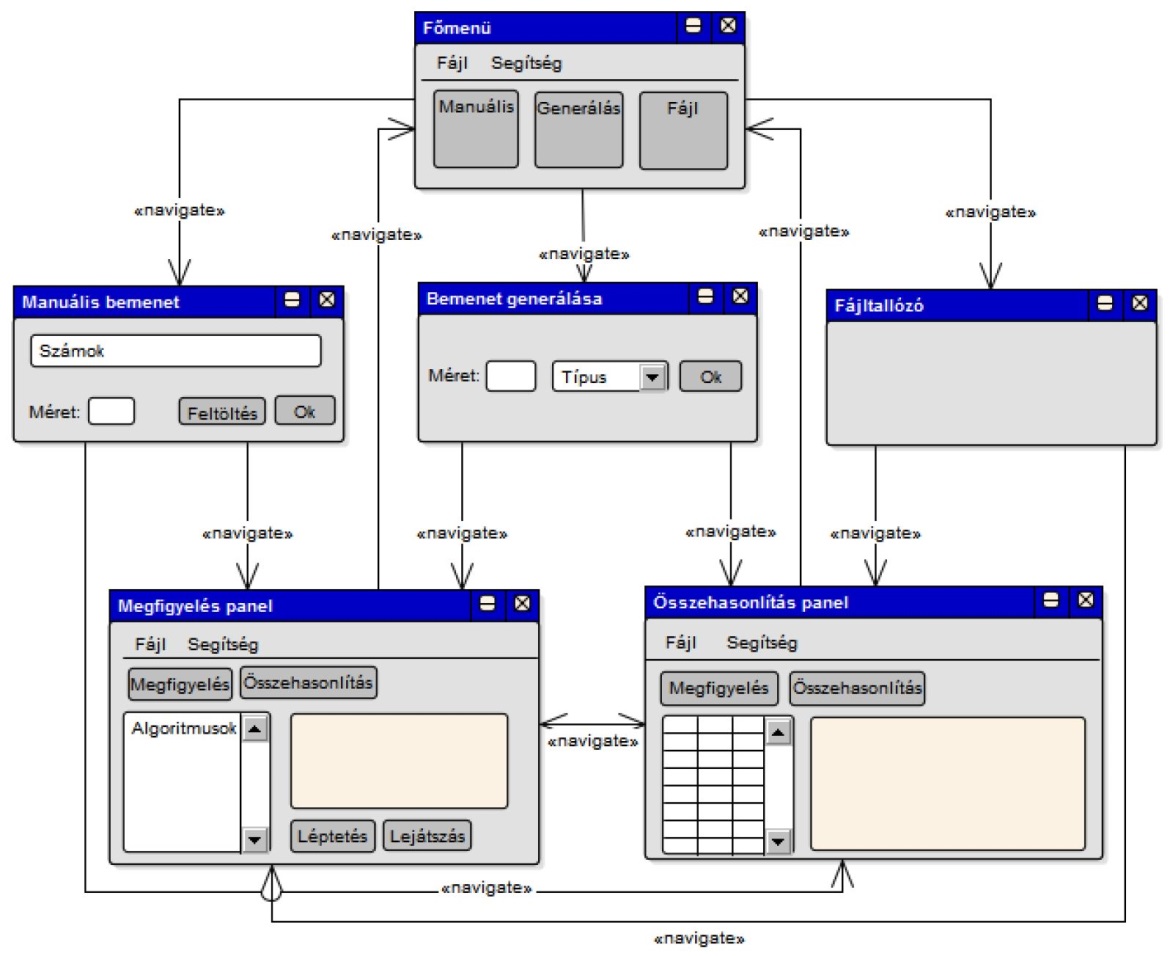
3.3. ábra. Főpanelek képernyőterve

Megfigyelés panel. A bal szélen egy algoritmus lista található, melyből kivá­lasztható a megfigyelni kívánt eljárás. A Léptetés és Lejátszás gombokkal a sárgás színnel jelzett diagramon az algoritmus egy újabb állapota ügyelhető meg.

Összehasonlítás panel. A bal oldalon található táblázatban szerepelnek az algo­ritmusok által végzett műveletek összegei. A sárga színnel jelölt diagramon megjelen­nek a kiválasztott algoritmus által végzett összehasonlítások és mozgatások összege.

Felületek közötti navigálási lehetőségek

A felületek közti navigálási irányokat a következő diagram szemlélteti:



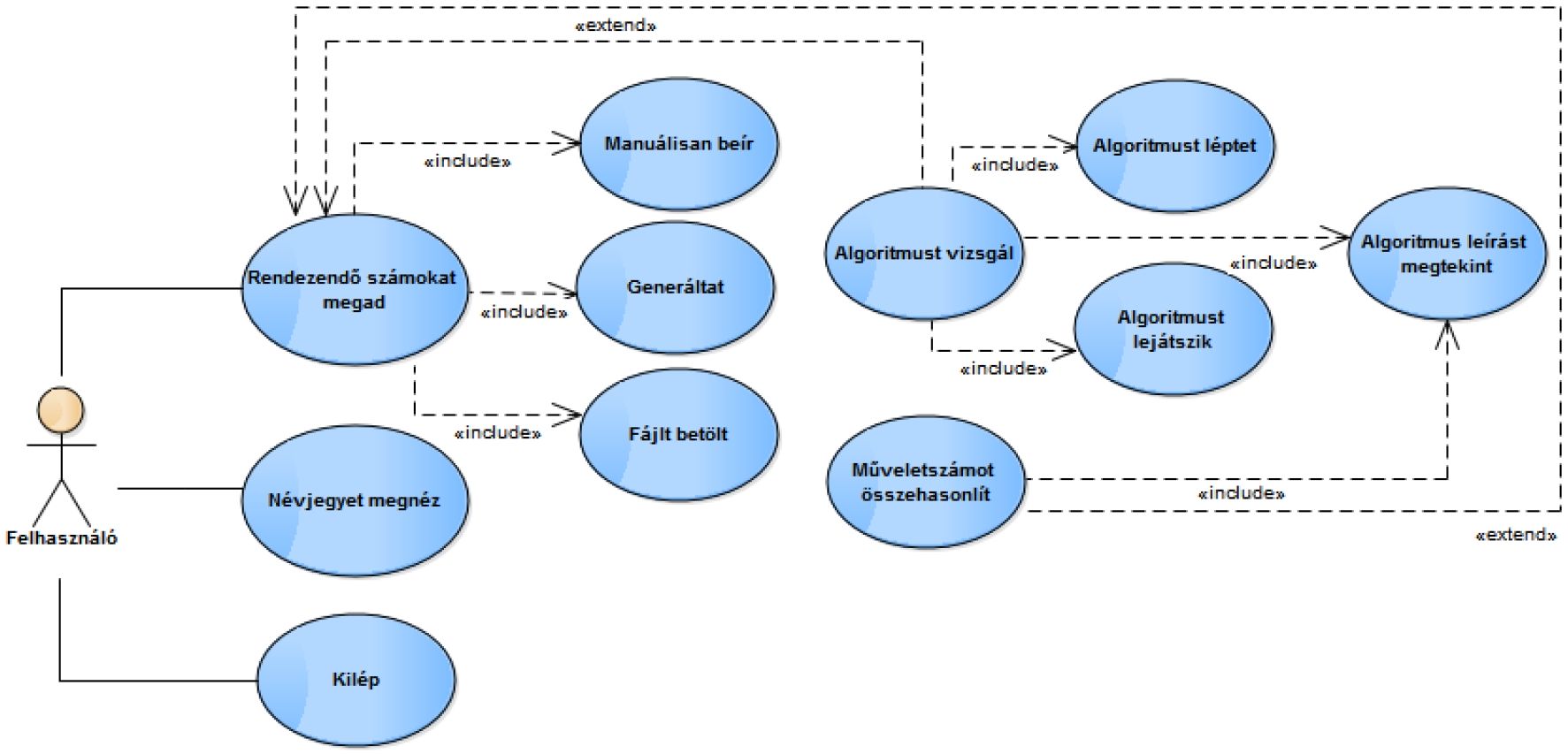
3.4. ábra. Navigálási lehetőségek

A főmenüből valamely gombra kattintva elérhetővé válik a kiválasztott típusú bemenet megadása panel. A számok megadásával megjelennek a Megfigyelés és Összehasonlítás panelek. A két panel közötti átjárás egy fiilön történő kattin­tással tehető meg. Ezen paneleken pedig az eszköztár segítségével lehetőség van a Főmenübe való visszatérésre.

* + 1. Használati esetek

A következőkben a főbb használati esetek kerülnek bemutatásra. A felhasználó­nak tudnia kell:

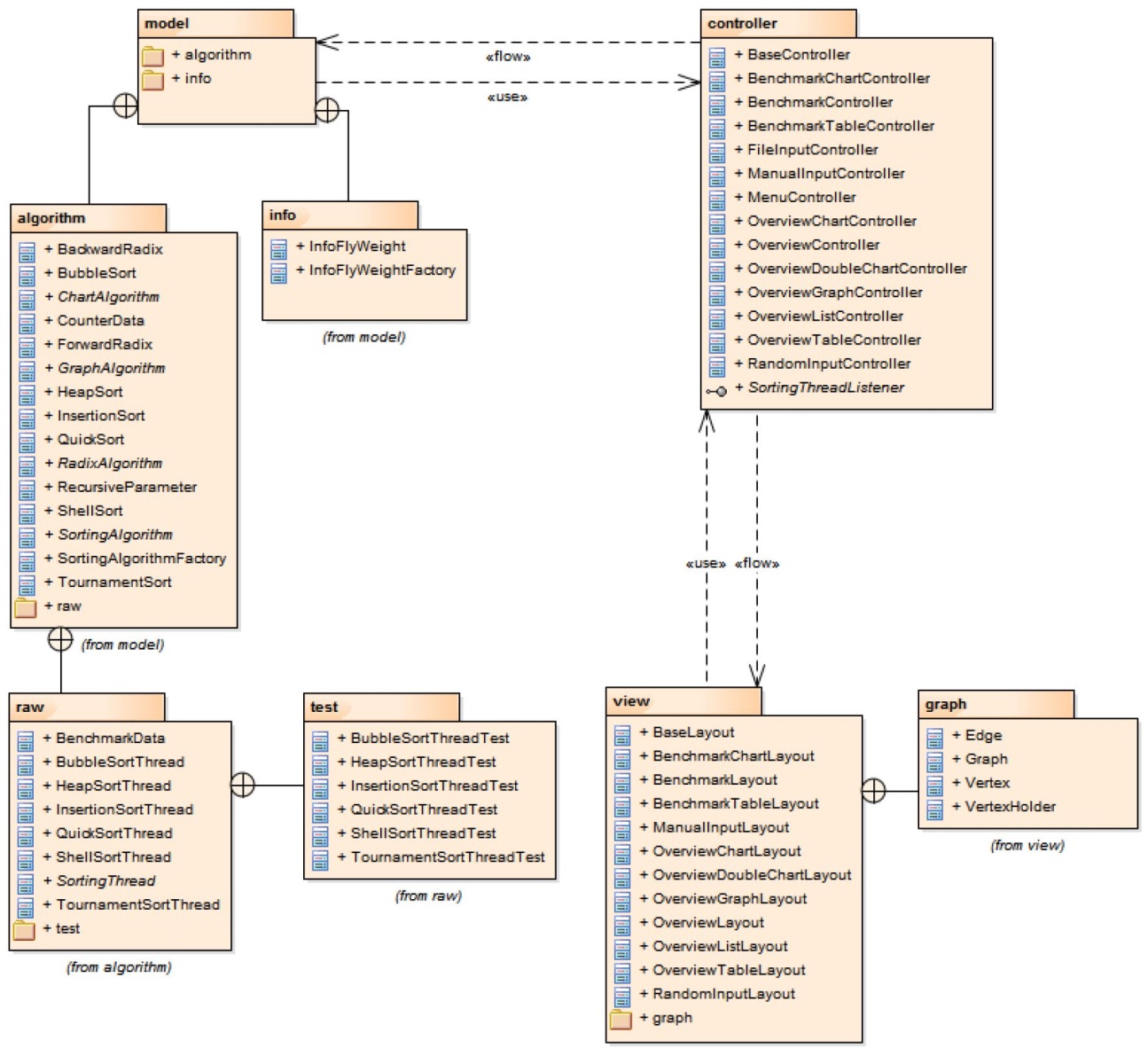
* rendezendő számok sorozatát megadni;
* algoritmust lejátszani;
* műveletszámokat összehasonlítani;
* rendezés leírását elolvasni;
* program névjegyét megtekinteni;
* programot bezárni;
* esetleges hibákról értesülni (például nem megfelelő fájl).



3.5. ábra. Használati esetek

* + 1. Csomagszerkezet

Az egyik alapelv, hogy a grafikus megjelenítés működés szintjén minél inkább különüljön el a rendszerlogikától. A grafikus felület már esak a modell programész által számított, a vezérlő programrésznek átadott adatokat jelenítse meg. Továbbá lehetőséget biztosítson a felhasználó számára, hogy vezérelje a program rendszer­logikáját a kontrolién rétegen keresztül. Ez által, hogy a rendszerlogika lényegében független a grafikus interfésztől, így a szoftver magasabb fokú bővíthetőséget, a kőd pedig jobb átláthatóságot nyer.



3.6. ábra. A program csomagszerkezete

Ennek tekintetében a programot három fő csomagra kell bontani: A modell cso­mag, melyben a logikai osztályok találhatóak, a view, melyben a graűkai interfészt leíró FXML fájlok és osztályok foglalnak helyet, végül a controller csomag, melynek osztályai kapcsolatot teremtenek a logikai és a megjelenítési réteg között.

Modell csomag

A modell osztályok a modell csomag foglalja magában. A pontosabb rendsze­rezést szem előtt tartva további alesomagok kerülnek létrehozásra, így jobban át­tekinthető az osztályhierarchia. Összességében ebben a csomagban foglalnak helyet azon osztályok, amelyek a rendszer adattároló, rendszerező és számítási feladatait ellátják. Az alábbi táblázat ad áttekintést az előzőleg említett alesomagokről:

|  |  |
| --- | --- |
| Név | Leírás |
| algorithm | A lejátszható algoritmusok implementációját tartalmazza |
| algorithm.raw | Az "egyszerű" rendezést megvalósító osztályok csomagja |
| algorithm.raw.test | A raw csomag osztályaihoz tartozó egységtesztek |
| info | "Az algoritmusról" felugróablakhoz tartozó adatbetöltő |

algorithm alcsomag. A csomag osztályainak többsége azon algoritmusimplemen­tációkat tartalmazza, melyek a Megfigyelés panelen megjelennek, azaz számon tart­ják az algoritmus aktuális állapotát. A csomag további osztályainak a fő funkciója, hogy azon műveleteket, amelyeket több algoritmus is használ legyenek kiemelve, ezáltal megakadályozhatóvá válik a kódismétlés.

algorithm.raw alcsomag. Azon algoritmusok megvalósítását tartalmazza, ame­lyekben nem kerül elmentésre a rendezés aktuális állapota. Az általuk szolgáltatott információ mindössze a rendezési feladat megoldása során végzett műveletek száma. Továbbá egy olyan osztályt tartalmaz a csomag, amely ezen adatokat rendezett for­mában át tudja adni a kontroller rétegnek, melynek segítségével megjelennek az egyes értékek a felhasználói felületen.

algorithm.raw.test alcsomag. Az előbbi csomag osztályaihoz tartozó egység­tesztek osztályait tartalmazza. Néhány egyszerűbb eset kerül tesztelésre, például: üres- és egy elemű tömb, száz; illetve egymillió véletlen generált szám rendezésének eredménye.

info alcsomag. A felületen egy algoritmust kiválasztva megtekinthető az eljárás rövid leírása. A leírások betöltése fájlból történik. A már lekérdezett szövegeket eltá­roljuk, így ha a felhasználó ismét lekérdezi az algoritmus leírását már nem szükséges újból beolvasni a fájlt. A csomag e funkciót megvalósító osztályokat tartalmazza.

Megjelenítő csomag

A grafikai interfészt leíró FXML fájlokat tartalmazza. Ezen fájlokban a felület elemei kerülnek definiálásra, azonosítókkal ellátva, melyek segítségével a Vezérlő csomag osztályai tudnak az egyes komponensekre hivatkozni. Ezen fájlok mindegyi­kének neve a Layout szuffixet kapta, ezáltal kikövetkeztethető, hogy egy komponens elrendezésének az "tervét" tartalmazzák.

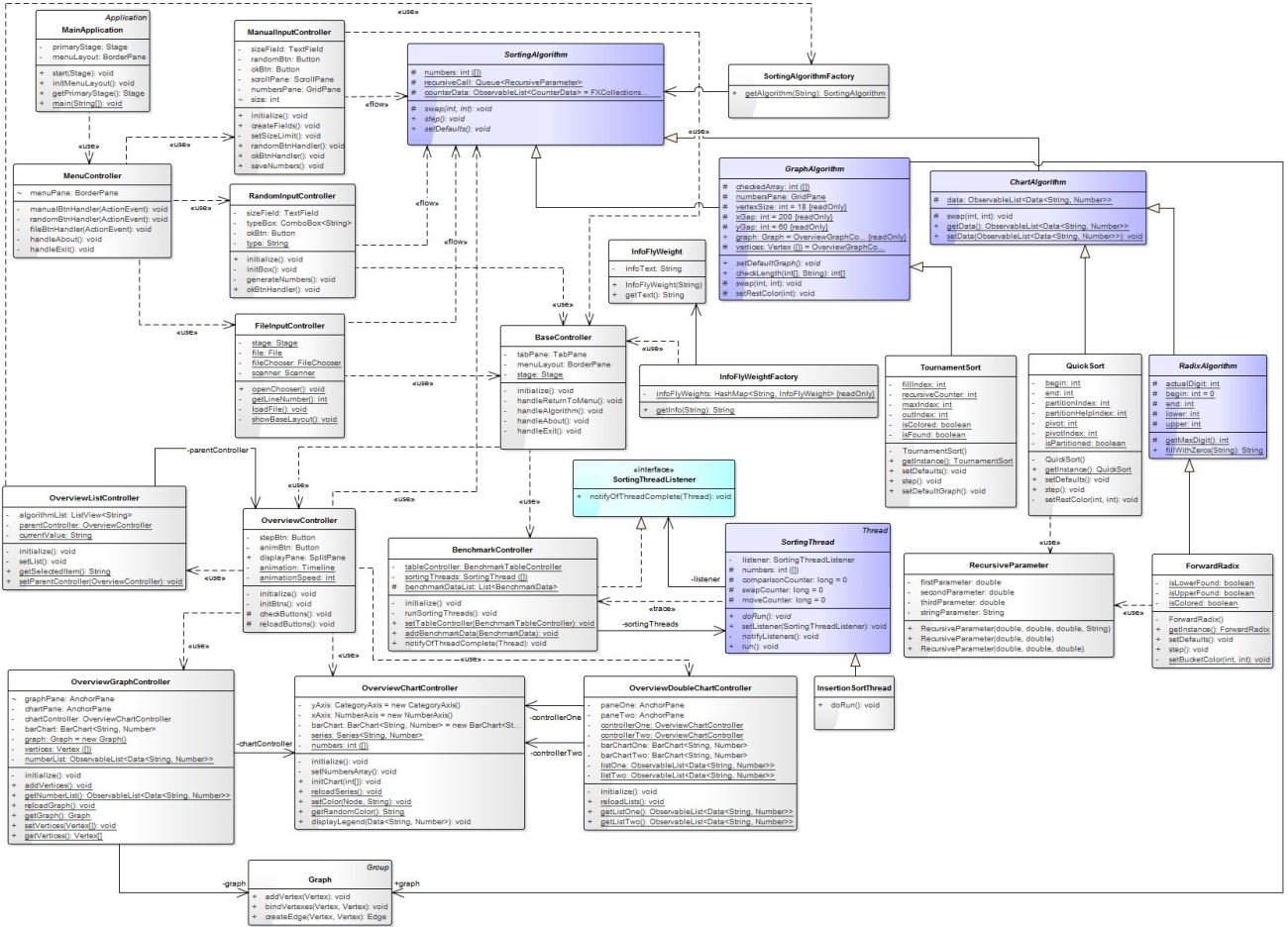
graph alcsomag. A gráf graűkus implementációját tartalmazó csomag.

Vezérlő csomag

Osztályai főleg a megjelenítő csomag komponenseihez tartoznak. Ezt az osztályok neve is jelölik, mivel csak a végződésükben(Controller) térnek el az FXML fájlok nevétől. Felhasználói interakcióra a modell osztály különböző metódusait hívja meg. így kapcsolatot teremt az előbb deűniált két csomag között. Továbbá tartalmaz egy interfészt, melynek felhasználásával az alrogithm.raw osztály algoritmusai miután befejezték a futásukat figyelmeztetik az interfészt megvalósító osztályt, ami jelen esetben a Összehasonlítás panel táblázatának vezérlő osztálya.

* + 1. Osztályszerkezet

A következőkben bemutatásra kerülnek a csomagok főbb osztályai, és ezek fonto­sabb eljárásai. A bemutatásban szereplő osztályok sorrendje többé-kevésbé megegye­zik a példánvosulásuk sorrendjével. Az ábrán kék színnel vannak jelölve az absztrakt osztályok, míg zöld hátteret kapott az egyik fontosabb interfész.



1. ábra. A főbb osztályok kapcsolata és kommunikációjuk

A megjelenítési réteg fájljai nem kerülnek bemutatásra, mivel ezek szerkezetüket tekintve azonosak, csak a grafikus interfész elemeit definiálják. Amennyiben egy fájl meghívásra kerül a kontroliéi\* réteg egyik osztályában az ismertetve lesz.

alapcsomag osztálya

MainApplication. A strnglgrthms csomag egyetlen osztálya, melyben a prog­ram main függvénye található. Létrehozza az ablakot, betölti a menü elrendezését tartalmazó FXML fájlt (MenuLayout.fxml).

controller csomag osztályai

A felhasználói interakciók hatását ez a réteg érvényesíti, figyeli a felület változá­sait és adatokat jelenít meg a megjelenítési rétegen.

MenuController. Az osztályban található metódusok kezelik a menü eszköztárán való kattintást (handleAboutQ és handleExitQ függvények). Továbbá a gombo­kon történő kattintás következtében betöltik a bemenet megadására lehetőséget adó panelekhez tartozó FXML fájlokat.

ManuallnputController. A menüben a Manuális gombra történő kattintás után példányosul az osztály. A sikeres bemenet megadását követően átadja a rendezendő elemeket a SortingAlgorithm osztályának. Végül betölti a program főpaneljeit tartalmazó BaseLayout.fxml fájlt.

RandomlnputController. A Véletlen gombra kattintás következtében példá­nyosul. A kontroller osztály végzi el a számok generálását, mely ezt követően az előbbivel azonosan átadásra kerül a SortingAlgorithm osztálynak. Szintén betölti a főpaneleket tartalmazó FXML fájlt.

FilelnputController. A harmadik gombra kattintás követően keletkezik belőle objektum. Beállítja a fájlkiterjesztés szűrőt és megjeleníti a fájltallózót. Továbbá itt történik meg a fájl tartalmának az ellenőrzése. Az előző két osztállyal azonosan végzi el a bemenő adatok átadását, valamint a főpanelek betöltését.

BaseController. A főpanelekhez tartozó kontroller osztály. Figyeli, hogy 100 elem­nél hosszabb-e a bemenet, amennyiben igen eltünteti az Megfigyelés panelt. Vala­mint metódusai kezelik az eszköztár egy elemén való kattintást.

OverviewController. A Megfigyelés panel gombjait figyeli és módosítja, pél­dául ha befejeződött egy algoritmus, akkor az Újraindítás gombot jeleníti meg a felületen. Továbbá az ezeken történő kattintást kezeli. A Léptetés vagy a Leját­szás gomb megnyomásakor lekérdezi a kiválasztott algoritmust a OverviewList­Controller osztálytól, majd a modell réteg SortingAlgorithmFactory osztályán keresztül elkéri a lejátszandó algoritmus példányát. Végül meghívja a megfelelő al­gorithm csomagban lévő osztály step() metódusát.

OverviewListController. A vizsgálható algoritmusok listáját tartalmazza, amennyi­ben a felületen módosítás történik a kiválasztott értéken újratölti a megfigyelés pa­nelen látható diagramot, és meghívja a kiválasztott elem setDefaultsQ függvényét, amellyel az aktuális algoritmus állapotát tároló változók felveszik az alapértéküket.

OverviewChartController. A Megfigyelés panelen található diagram kontrol­ler osztálya. A SortingAlgorithm osztály által eltárolt számokat betölti egy lis­tába, majd ezeket megjeleníti az oszlopdiagramon. Az oszlopok színének megváltoz­tatását is ez az osztály végzi el.

OverviewDoubleChartController. Az előbb említett osztálynak két példányát tartalmazza, a Radix "vissza" rendezésnél tölti fel az első példány listáját a ren­dezendő számokkal, míg a másodikét nullákkal.

OverviewGraphController. Az előző két osztállyal megegyező a funkciója. To­vábbá "karbantartja" a megjelenített gráfot.

BenchmarkController. Példányosulását követően elindítja a algorithm.raw cso­mag algoritmusait. Megvalósítja a SortingThreadListener interfészt, ezáltal fi­gyelmeztetést tudnak küldeni neki a SortingThread osztályok ha befejezték futá­sukat. Feladata az Összehasonlítás felületen a táblázat frissítése, ha egy rendezési feladat megoldódott.

BenchmarkChartController. Az OverviewChartController osztályhoz ha­sonlóan egy diagram kontroller osztálya. A Összehasonlítás panel táblázatának egy elemén történő kattintást kezeli, amennyiben a kiválasztott elem még nem jele­nik meg a grafikonon hozzáadja, ellenkező esetben eltávolítja a grafikonról.

algorithm csomag osztályai

Főként a lejátszható algoritmusokat tartalmazza. Ezen osztályok mindegyike egyke, azaz a futás során csak egyetlen példány keletkezik belőlük.

TournamentSort. Mivel mindegyik rendezést megvalósító osztály kevéssé tér el egymástól, ezért csak egy kerül rövid ismertetésre.

Alapértelmezetten mindegyik osztály tartalmaz egy step() metódust, amely egy következő állapotba lépteti a rendezést. Minden osztályhoz a rendezést megvalósító változókon felül tartoznak olyan logikai változók, amelyekben az aktuális állapot egy-egv tulajdonsága elmentésre kerül. Ilyen változóban tárolódik például az, hogy a Versenyrendezés bináris fájában megtaláltuk-e a maximális elemhez tartozó levélelemet.

SortingAlgorithm. A rendezést megvalósító osztályok őse a SortingAlgorithm absztrakt osztály. Néhány olyan metódust valósíst meg, amelyek minden rendezésnél azonosak.

ChartAlgorithm. Az előbb említett osztály absztrakt "gyereke", továbbá azon rendezést megvalósító osztályok ősosztálya, amelyek megjelenítéskor csak oszlopdi­agramokat használnak.

RadixAlgorithm. A Chart Algorithm osztály absztrakt leszármazottja. A ra­dix rendezések ősosztálya. Néhány olyan metódust tartalmaz, amelyeket mindkét rendezés használ.

GraphAlgorithm. Szintén a SortingAlgorithm absztrakt "gyereke", és azon osztályok őse, melyek a grafikus megjelenítéskor oszlopdiagramot és gráfot is hasz­nálnak.

SortingAlgorithmFactory. A SortingAlgorithm osztályok gyártó osztálya, az egyetlen getAlgoritlim(String algorithmName) metódusa a paraméter tartal­mától függően visszaadja a kért algoritmus példányát.

RecursiveParameter. Néhány adat eltárolására használt osztály. Például a gyors­rendezés megvalósításakor a rekurzív hívások elmentése ilyen objektumokat tartal­mazó listába történik.

alogirtmh.raw csomag osztályai

Azon rendezések implementációját tartalmazza, amelyek megjelennek az Össze­hasonlítás panelen. Ezen osztályok mindegyike egy külön szálat hoz létre, amely szálak befejeződésüket követően figyelmeztetik a már ismertetett BenchmarkCont­roller osztályt.

InsertionSortThread. Az előző csomag osztályaihoz hasonlóan, mivel ezen osz­tályok minimálisan térnek el felépítésüket tekintve csak egy osztály kerül rövid jel­lemzésre. A sort() metódus hívására egy programszálat indít el, amely elvégzi a rendezést majd átadja a műveletszámot a BenchmarkController osztálynak.

SortingThread. Az előbb említett osztálytípusok őse. Néhány olyan metódust valósíst meg, amelyek minden rendezésnél azonosak. Továbbá tartalmazza a műve­letek összegzéséhez használt változókat.

BenchmarkData. Egy rendezés a befejeződését követően példánvosít belőle egy objektumot. Eltárolja az algoritmus által végzett összehasonlítások és mozgatások vagy cserék számát. Ilyen példány kerül átadásra a BenchmarkController osztály számára, amely ezekből listát készít.

info csomag osztályai

A két osztálya megvalósítja a pehelysúlyú programtervezési mintát. A minta használatának oka az, hogy a felhasználó egy adott algoritmushoz többször is lekér­heti a leírást. Ekkor felesleges ismételten fájlból beolvasni a szöveget, mivel az nem változott. Célszerű ilyenkor a már lekért adatokat elmenteni, és azt visszaadni ha újra hivatkozás történik rá.

InfoFlyWeight. Az osztály egyetlen adattagja egy String, Ebben a változóban tárolódik egy algoritmus leírásának a szövege.

InfoFlyWeightFactory. Az osztály eltárolja a már betöltött szövegeket. A Ba­seController osztály handleAboutQ függvénye az algoritmus listából aktuálisan kiválasztott elem szöveges reprezentációjával meghívja az osztály getlnfo(String algorithmName) függvényét. A függvény megnézi, hogy már tárolásra került-e az algoritmushoz tartozó szöveg. Amennyiben igen, azt az objektumot adja vissza. Ellenkező esetben beolvassa a fájlt, és elmenti a szöveget egy InfoFlyWeight objek­tumba. Ezzel egyidejűleg az "új" algoritmus szövege is tárolásra kerül.

* 1. Megvalósítás

3.2.1. Az eredeti terv módosítása

A megvalósítás során néhány ponton módosítást kellett eszközölni az eredeti terveken.

Szálkezelés

Eredetileg egy-egv külön szálon futottak volna az algoritmusok, és a felhasználói interakció hatására ezek állapota változott volna. Azonban a JavaFX szálkezelése jelentősen eltér az szokványos szálkezelésétől, ezért járhatóbb útnak bizonyult az ha kétszer kerüljenek implementálásra az algoritmusok. így került kialakításra a

algorithm.raw csomag.

Az egyik implementációban elmentjük az interakciót követő állapotot, és ez jele­nik meg a felhasználói felületen. A másik megvalósításban pedig a rendezések azonnal lezajlanak, így képet kaphatunk arról, hogy mennyi műveletre volt szükség az egyes eljárások során. Ezen utóbbi implementációk mindegyike külön szálon fut, és ahogy valamelyik befejeződik figyelmezteti a főprogramot, hogy jelenítse meg a műveletek számát.

A JavaFX kibővítése

A szoftver erősen épít a JavaFX adta felületi komponensekre. Azonban akad olyan összetevő, amelyet ki kell bővíteni, vagy létre kell hozni az áttekinthetőség érdekében.

Ilyen például az oszlopdiagram, amely ugyan beépített komponens, de alapértel­mezetten nem jelennek meg a diagramon az oszlopokhoz tartozó értékek számsze­rűen, csak a tengelyszámokból lehet kikövetkeztetni őket. Ez nyilvánvalóan így nem megfelelő, az oszlopok értékét valahogy jelölni a megjelenítéskor. A legegyszerűbb megoldás az, ha mindegyik oszlopérték felé egy feliratot(Lahel-t) helyezünk el. Fi­gyelni kell a szöveghez tartozó oszlop méretét. Amint megváltozik az diagram egy elemének a mérete módosítani kell a felette lévő szövegdobozban tárolt számot is.

Továbbá a JavaFX egyelőre nem kínál lehetőséget gráfok ábrázolására, itt meg­oldás lehet az, hogy egy külső függvénykönvvtárat felhasználva - például az yFiles fór JavaFX-et - ábrázoljuk a bináris fákat. Azonban ezáltal a programnak több függősége lenne, és a könyvtárnak csupán néhány elemére van szükség a gráf meg­jelenítéséhez. Ezért célszerűbb egy saját implementációt készíteni, amely csupán azokat a műveleteket tartalmazza, amiket a szoftver használ.

3.2.2. A megvalósítás menete

Az első lépés volt a rendezési algoritmusok implementálása. Itt az előző alfeje- zetben leírtaknak megfelelően kétszer kell elkészíteni a rendezési eljárásokat. Ezzel egyidejűleg minden "egyszerű" rendezést megvalósító szálhoz egységteszteket kell írni.

Ezt követhette egyszerűbb felhasználói felület létrehozása. Kezdetben elegendő, ha csak egy graűkon jelenik meg, amely reprezentálja a tömbben található számokat.

Két algoritmushoz szükséges a gráfos megjelenítés, így a következő lépés egy gráf graűkus implementálása. Ezt követően a cél, hogy néhány "beégetett" elemre a rendezések lejátszhatóak legyenek, és az aktuális állapota a tömbnek szinkronban legyen a diagrammal valamint a gráffal. Később az egyes lépésekben történő össze- hasonlításokat/vizsgálatokat, mozgatásokat, cseréket kell különböző színekkel jelölni az állapotjelző felületeken és számon tartani ezen műveletek összegeit.

Miután a program alapjai elkészültek kezdetét veheti a a felhasználói felület részletes kialakítása. Elsőként a diagram elhelyezése egy panelen, amely tartalmaz továbbá egy listát a választható algoritmusokról. Illetve egy táblázatot, melyben szerepelnek az aktuális állapot egyes tulajdonságai.

A programnak egy fontos szolgáltatása az, hogy a felhasználó különböző adat­beviteli mód közül választhat. A logikai réteget ki kell bővíteni ezekkel az esetekkel, továbbá a felhasználói felületen lehetőséget adni ezen módok kiválasztására.

Ezután az eszköztár kerül a helyére, mellyel párhuzamosan megtörténik az egyes műveletekhez tartozó eljárások implementálása.

Az utolsó felületi módosítás egy rendezések összehasonlítására lehetőséget adó panel létrehozása, táblázattal, benne az algoritmusok műveletigényével. Továbbá egy diagram hozzáadása, amin megjelenik a táblázatból kiválasztott sor összehasonlítá­sainak és mozgatásainak a száma.

Végül, hogy a program egyszerűen használható legyen Windows környezetben egy telepítő fájl készítése, amellyel az előbb említett operációs rendszert használók­nak nem szükséges külön Java-1 telepíteniük.

1. FXML állományok

Az FXML az Oracle által fejlesztett dekleratív XML alapú nyelv[ll]. Ezen fájlok definiálják a grafikus felület komponenseit. Egy ilyen fájl a következőképp épül fel:

* Az első sor mindig az XML tulajdonságokat definiáló sor
* A következő sorokban található azon könyvtárak importálása, amelyekben megtalálhatók a felhasznált komponensek
* Ezt követi az ablak elrendezési módjának a megadása, ekkor több beépített megvalósítás közül lehet választani, többek között:

o BorderPane - 5 pozícióba helyezhetőek elemek o GridPane - táblázatszerűén jelennek meg az elemei o AnchorPane - egv "szimpla" felület, nincs megkötés

Továbbá ebben a sorban kell megadni az FXML állományhoz tartozó kontroller osztály elérési útvonalát. Az vezérlőosztály tud hivatkozni az egyes összetevőkre az azonosítójukon keresztül.

* Végül a komponensek definiálása: méretük, azonosítójuk megadásával.

Néhány fontosabb FXML állomány

A következőkben bemutatásra kerülnek a projekt főbb FXML fájljai.

BaseController. Az egyik "összefogó" FXML fájl. A rendezendő számok meg­adása után minden esetben ez az állomány töltődik be. Alapvetően a fejlécet és egy TabPane komponenst tartalmaz, ami a két panel közötti navigálást teszi lehetővé. Ezeken felül importálva van két további állomány is: az OverviewLayout és a BenchmarkLayout. Ennek következtében, amikor a BaseController betöltődik vele együtt az előbb említett két fájl is beolvasásra kerül minden további kompo­nensükkel.

OverviewLayout. Szintén több komponenst foglal magában, a Megfigyelés pa­nelt gombjait és a más fájlokból betöltött komponensek elhelyezkedését defini­álja. Az általa importált további állományok: OverviewListLayout, Overview- TableLayout, továbbá az aktuálisan kiválasztott algoritmus típusától függően a

OverviewChartLayout, OverviewDoubleChartLayout vagy OverviewGra- pliLayout. valamelyike.

BenchmarkLayout. Az Összehasonlítás panel részeinek elhelyezkedését írja le. A következő FXML fájlokat hívja meg betöltésekor: BenchmarkTableLayout, BenchmarkChartLayout.

1. Nem forrásfájl állományok

A program által használt nem forrás- vagy FXML fájlok a resources mappában találhatóak. Ezen belül az images könyvtárban található meg a szoftver ikonja. A text mappán belül találhatóak azon fájlok, amelyeket az InfoFlyWeight osztály betölt. Azaz ezekben az állományokban található meg az egyes algoritmusok rövid ismertetése. A harmadik input mappában a fájlból történő bemenet megadásához található néhány példaadat.

1. Osztályok leírása

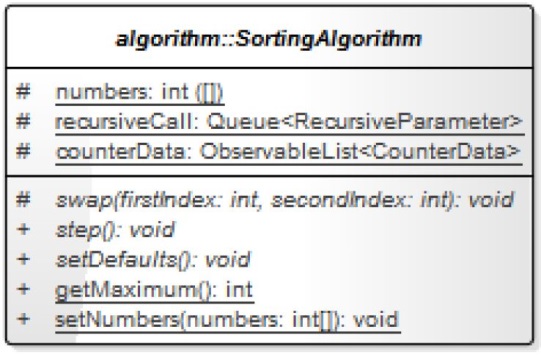
Az előző fejezetben(3,l) ismertetésre kerültek a főbb osztályok és azok fontosabb eljárásai, a következőkben nagyobb részletességgel kerülnek jellemzésre a szoftver osztályai.

Modell réteg osztályai

Elsőként a algorithm esomag főbb osztályai kerülnek ismertetésre.

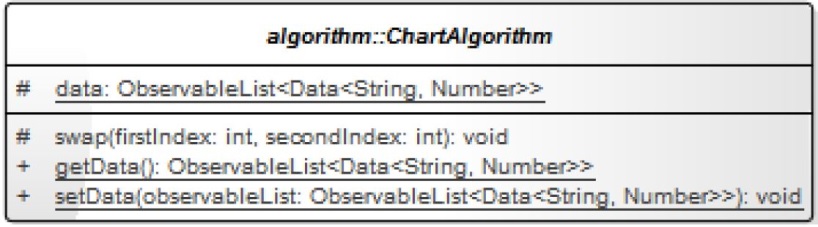
SortingAlgorithm absztrakt osztály. A Megfigyelés panelen megjelenő al­goritmusok osztályainak őse. Az osztály numbers statikus adattagjában kerül el­tárolásra a bemenetkor megadott számsorozat. További két adattagja egy rekurzív hívások mentésére szolgáló- és egy CounterData objektumokat tartalmazó lista.

A getMaximum() eljárás visszaadja a numbers tömb maximális elemét. A függvénynek a megjelenítéshez használt oszlopdiagram lépésközének deűniálásánál van szerepe. Továbbá tartalmaz három absztrakt eljárást, amelyeket a leszármazott osztályai valósítanak meg.



1. ábra. A SortingAlgorithm osztálydiagramja

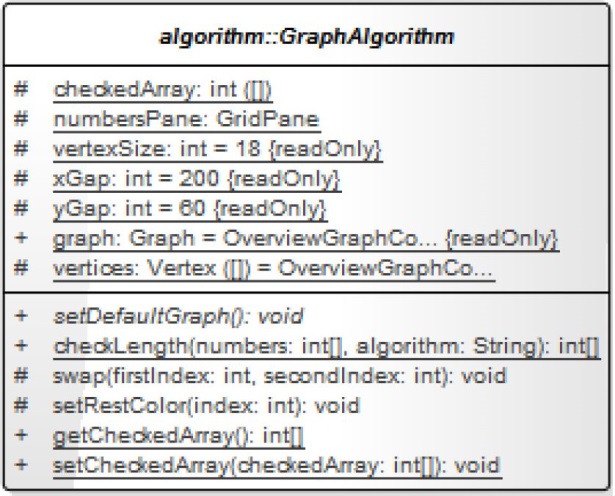
ChartAlgorithm absztrakt osztály. Az előző osztály leszármazottja, megvaló­sítja a swap(int firstlndex, int secondlndex) metódusát. A diagram kontroller osztálya a rendezendő számokat egy ObservableList adatszerkezetben tárolja, így ennek megfelelően került az eljárás implementálásra. A kontrolién réteg előbb emlí­tett listáját statikus változóként elmenti, így a leszármazott osztályai egyszerűbben tudnak hivatkozni az egyes értékekre.



1. ábra. A ChartAlgorithm osztálydiagramja

GraphAlgorithm absztrakt osztály. Szintén a SortingAlgorithm leszárma­zott absztrakt osztálya. A GraphController osztály gráfját és csúcsait tárolja, így a leszármazott osztályai egyszerűbb elérik ezeket az adattagokat.

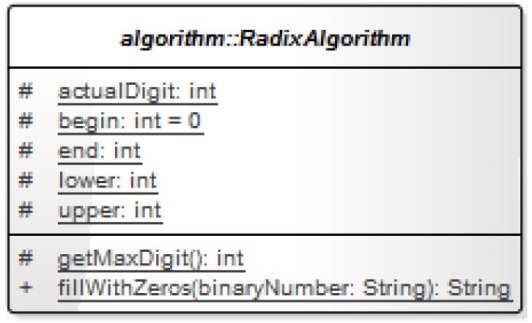
Továbbá ellenőrzi a bemenet hosszát, és ha szükséges "csonkolja" vagy megnö­veli az eredeti tömb hosszát a checkLength(int[] numbers, String algorithm) metódusa. Megvalósítja az előző osztályhoz hasonlóan a swap(int firstlndex, int secondlndex) metódust. Valamint a setRestColor(int index) metódusa a para­méterül kapott indexig visszaállítja az eredeti színűre a Megfigyelés panelen meg­jelenített gráfcsúcsokat, valamint a oszlopokat.



1. ábra. A GraphAlgorithm osztály diagramja

RadixAlgorithm absztrakt osztály. A ChartAlgorithm leszármazott abszt­rakt osztálya. Továbbá a radix eljárásokat megvalósító osztályok őse. A két származ­tatottjának adattagjai már itt definiálásra kerülnek, azaz négy "pointer", valamint az aktuálisan vizsgált számjegy indexe(actualDigit).

A getMaxDigit() metódusa visszaadja a leghosszabb bináris számot, amelyet a másik flllWithZeros(String binaryNumber) metódus fel is használ, mivel ennek megfelelően a rövidebb számokat nullákkal konkatenálja.



1. ábra. A GraphAlgorithm osztály diagramja

SortingAlgorithmFactory osztály. Létrehozásakor a gyártó metódus program­tervezési minta lett alapul véve. Ezen az osztályon keresztül kérhetőek el az algorit­musok példányai, függetlenül attól, hogy az aktuális algoritmus őse a ChartAlgo- rithm vagy a GraphAlgorithm osztály.

Egyetlen metódusának paramétere egy algoritmus szöveges reprezentáeiőja, pél­dául: "Gyorsrendezés". Az eljárás visszaadja a rendezést megvalósító osztály példá­nyát. Elsősorban a kontroller réteg AlgorithmList osztálya használja.

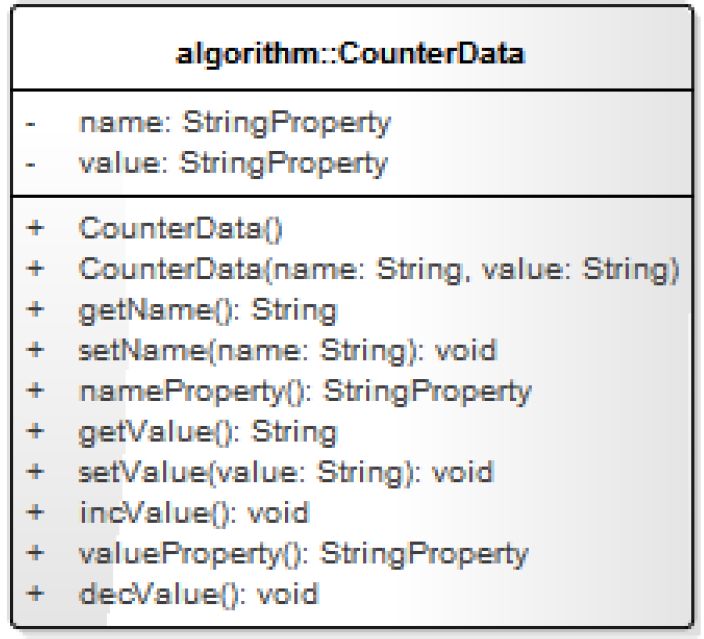
a I g Dritirm:: Sorti n gAl g orith m Fa ctory

■+ •gEtAI-g&rithm;al-gprithmNam&: String): ScrtingAIgcrithm

1. ábra. A SortingAlgorithmFactory osztálydiagramja

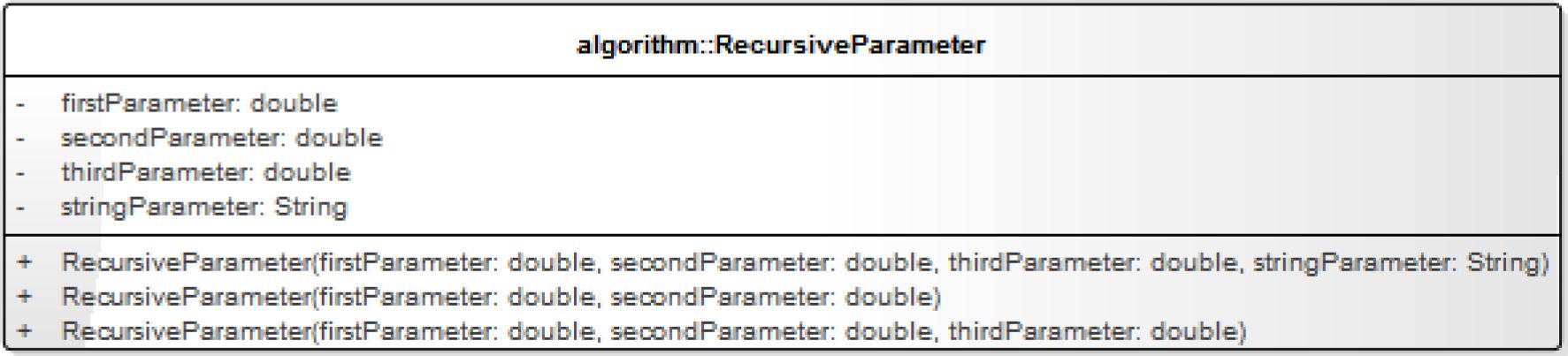
CounterData osztály. Az osztály segítségével adatok tárolhatóak kulcs-érték párban. Az implementációval kapcsolatban felmerülhet a kérdés, hogy miért nem a Java nyelv egy beépített adatszerkezete került felhasználásra. Ennek oka, hogy a felhasználói felületen lévő táblázatok csak olyan adatszerkezetet tudnak megje­leníteni, amelynek adattagjainak a típusa Property végződésű, és implementálva vannak ezekhez a mezőkhöz a getter metódusok.

Az algoritmusokat megvalósító osztályok ilyen adatokból álló listában tárolják az egyes "tulajdonságaikat", és adják át a vezérlőrétegnek. A vezérlőréteg pedig a Megfigyelés panel táblázatában jeleníti meg az értékeket.



1. ábra. A CounterData osztálydiagramja

RecursiveParameter osztály. Funkciója hasonló az előbb ismertetett osztályé­hoz, azaz adatok tárolására lett létrehozva. Legfeljebb négy érték elmentésére al­kalmas. Azon algoritmusok, amelyek rekurziót használnak ezen objektumokból álló listába mentik el a rekurzív hívások paramétereit. A konstruktora túlterhelt, azaz legalább kettő, maximum négy paraméterrel hívható.



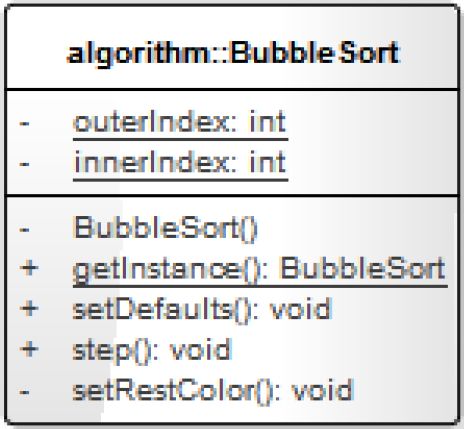
1. ábra. A RecursiveParameter osztálydiagramja

A következőkben az algoritmusok lejátszását megvalósító konkrét osztályokat jellemezzük, melyek szintén az algorithm csomagban találhatóak. Az osztályok mindegyike egyke, azaz a program futása során egyetlen példány keletkezik belőlük.

BubbleSort osztály. A buborékrendezést megvalósító egyke osztály. Ősosztálya a ChartAlgorithm Két változója megfelel az algoritmus két eiklusváltozőjának, Konstruktora privát, a getlnstance() metódusán keresztül kérhető le a példánya. Amennyiben nincs még létrehozva példány elkészíti, ellenkező esetben visszaadja a már létrehozott objektumot.

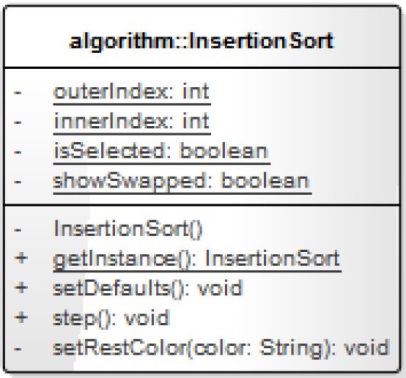
Az egyke tulajdonságból következően azokat a változókat melyekben tárolódik a rendezés aktuális állapota minden futtatás után vissza kell állítani az eredeti ér­tékre, ezt a setDefaults() függvény teszi meg. Továbbá ez a metódus hozza létre a tulajdonságokat tartalmazó CounterData objektumokat.

A step() örökölt eljárás egy rendezési lépést hajt végre. A setDefaults() me­tódus pedig visszaállítja a Megfigyelés panelen látható diagramok színét az alap­értékre.



1. ábra. A BubbleSort osztálydiagramja

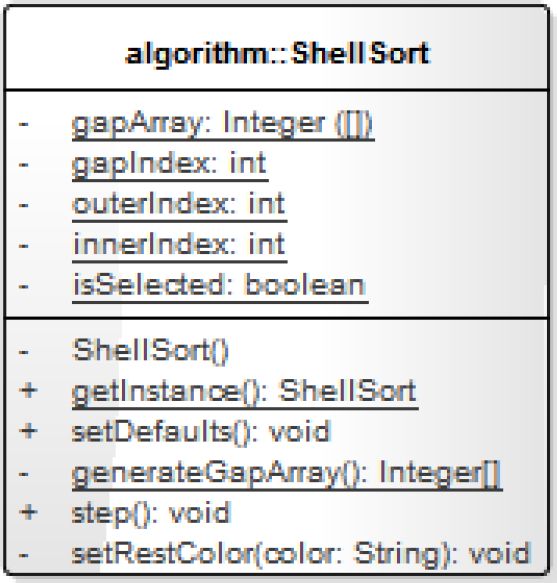
InsertionSort osztály. A beszúró rendezést megvalósító osztály. Alapvetően fel­építését és függvényeit tekintve megegyezik a BubbleSort osztálynál leírtakkal. Különbség, hogy két további logikai változóval rendelkezik. Az isSelected mezőben tárolódik az az informáeiő, hogy már kiválasztásra került-e egy elem, mivel a step() metódus ennek megfelelően hajtja végre a következő lépést. Továbbá a showSwap- ped változó értéke adja meg, hogy a felületen megjelenítésre került-e egy mozgatás.



1. ábra. Az InsertionSort osztálydiagramja

ShellSort osztály. A Shell rendezést megvalósító osztály. Őse, akáresak az eddigi osztályoknak a ChartAlgorithm, A BubbleSort-nál leírtakkal lehet jellemezni, viszont tartalmaz néhány további változót és eljárást. A gapArray tömbben táro­lódnak a lépésközök, az aktuális lépésköz a gaplndex változó segítségével kérhető le a tömbből. Az InsertionSort-nál leírtakkal megegyező szerepet tölt be a isSe­lected változó.

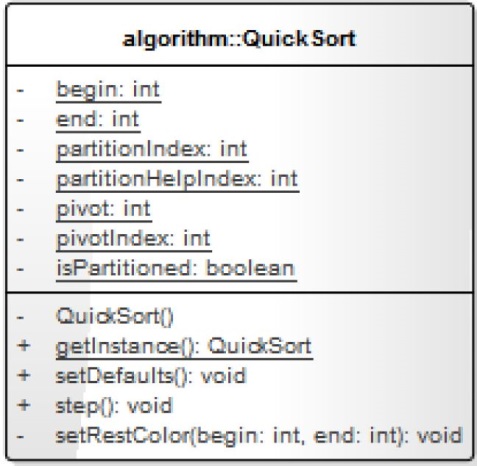
A generateGapArray() eljárás feltölti a lépésközöket tároló tömböt a bemenet méretétől függően a Vaughan Pratt által javasolt értékekkel|6|. Az implementáci­óban a legutóbbi tömbhöz adott számot kettővel és hárommal szorozzuk, és számon tartjuk, hogy egy lépésben melyik szorzást végeztük el. Amikor elértük azt az ér- tókot, amely nagyobb vagy egyenlő a tömb hosszával, akkor befejezzük a számok generálását, az utolsó érték nem lesz eleme a tömbnek.



1. ábra. A ShellSort osztálydiagramja

QuickSort osztály. A gyorsrendezés implementációja. Őse szintén a ChartAl- gorithm osztály. A begin és end változói jelölik az aktuálisan rendezendő tömb­rész első és utolsó elemét. A partitionlndex és partitionHelpIndex mezőknek a tömb felosztásánál van szerepük. A pivot változóban tárolódik a vezérelem értéke, melynek indexét a pivotlndex tartalmazza. A isPartitioned logikai változó értéke mutatja, hogy megtörtént-e a tömb felosztása. Az osztály használja a SortingAlgo- rithm ősének a recursiveCall listáját, ide menti el a rekurzív hívásokhoz szükséges adatokat.

A setDefaults() metódus működése megegyezik az eddigiekkel. A setRest- Color() eljárás viszont két egész számot vár paraméterként, és a két szám által meghatározott intervallumon kívül eső részen az oszlop diagramok színét fakóra ál­lítja át a vezérlő rétegen keresztül.

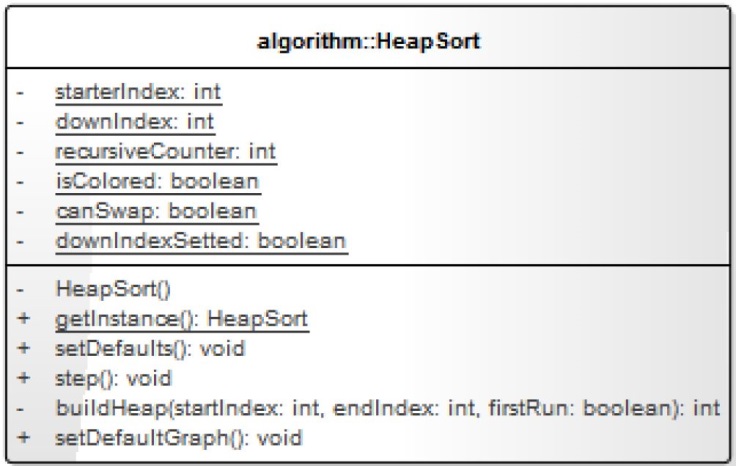


1. ábra. A QuickSort osztálydiagramja

HeapSort osztály. Őse a GraphAlgorithm osztály, a kupacrendezést valósítja meg.

A starterlndex változó tárolja azon elemét indexét, melynek vizsgáljuk a testvér­szülő "viszonyát". A downlndex segédváltozóban tárolódik a kupac tulajdonság ellenőrzésekor azon elem indexe, amelyet következőleg vizsgálni kell. A recursive- Counter megfeleltethető a kupacrendezés eljárás ciklusváltozójának. Továbbá há­rom logikai mezővel rendelkezik az osztály, melyek közül a isColored jelöli, hogy már a diagramon megváltozott-e a vizsgált elemek színe. A canSwap változó ér­tékétől függően hajtja végre az algoritmus a cseréket, mivel ez csak azt követően eszközölhető, ha már kijelölődtek a cserélendő elemek a Megfigyelés panelen. A downlndexSetted segédváltozó amennyiben igaz, akkor már megtörtént a kupac felépítése az első menetben, innentől kezdve a gyökérelemtől kell vizsgálni a kupac­tulajdonságot, így ennek megfelelően átállít ódik a downlndex értéke 0-ra.

Az átláthatóságot szem előtt tartva a step() metódusból kiemelésre került a kupac(újra)építés eljárása, melyet a buildHeap(int startlndex, int endlndex, boolean firstRun) metódus valósít meg. A harmadik paramétere határozza meg, hogy alapkupac építése-e a feladat vagy a kupactulajdonság ellenőrzése. Az eljárás visszatérési értéke a következő vizsgálandó elem. A setDefaultGraph() függvény építi fel a bináris fát a rendezendő számokból, majd adja át a vezérlőrétegnek meg­jelenítésre.



1. ábra. A HeapSort osztály diagramja

TournamentSort osztály. A versenyrendezést megvalósító egyke osztály, mely­nek őse az előző osztályhoz hasonlóan a Graph Algorithm.

A filllndex a versenyfa felépítésekor tárolja azon elem indexét, amelynek gyere­keit "versenyeztetjük". Akárcsak az előbb jellemzett osztálynál, itt is megfeleltethető a recursiveCounter változó az algoritmus ciklusváltozójának. A maxlndex mező tárolja a maximális levélelem keresésekor az egyik megtalált belső csúcsot. Az out­Index segítségével pedig elérjük egy maximális elem megtalálásakor a Megfigyelés panel oszlopdiagramjának egy adott oszlopát.

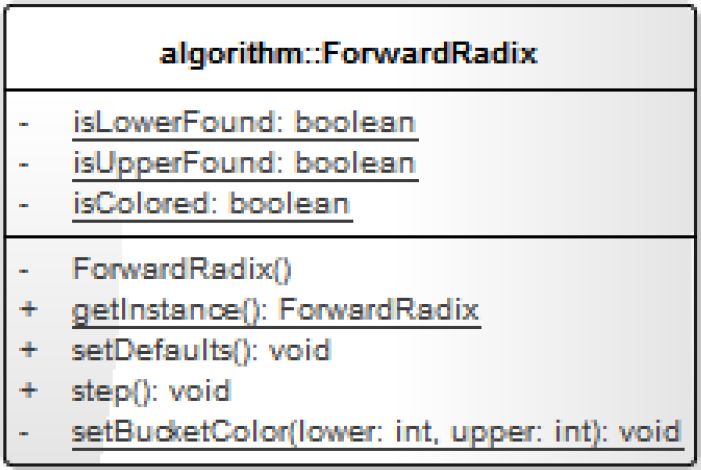
A setDefaultGraphQ és step() eljárás funkciója megegyezik a HeapSort-nál leírtakkal.

|  |  |
| --- | --- |
| a Igori ttim : :Tou ma mentSort | |
| - | fillIndEx: int |
| - | recursiveCountEr: int |
| - | maxindEx: int |
| - | outi nd ex: int |
| - | isColorEd: boolEan |
| - | iaFlwnd: boolean |
| - | Tourna mantSortQ |
| + | flEtlnstanoeí): ToirnamEntSdt |
| + | SEtDEfaultal): void |
| + | stEp(): void |
| + | SEtDEfa u ItG ra p b ! ): void |

1. ábra. A TournamentSort osztálydiagramja

ForwardRadix osztály. Őse a RadixAlgorithm osztály, a radix "előre" rende­zést valósítja meg. Az isLowerFound és isUpperFound változók tárolják, hogy egy adott lépésben talált-e az algoritmus olyan értéket, amely nem a megfelelő he­lyen van. Továbbá az isColored logikai mező értéke megadja, hogy egy cserélendő elem színe megváltozott-e már a felületen.

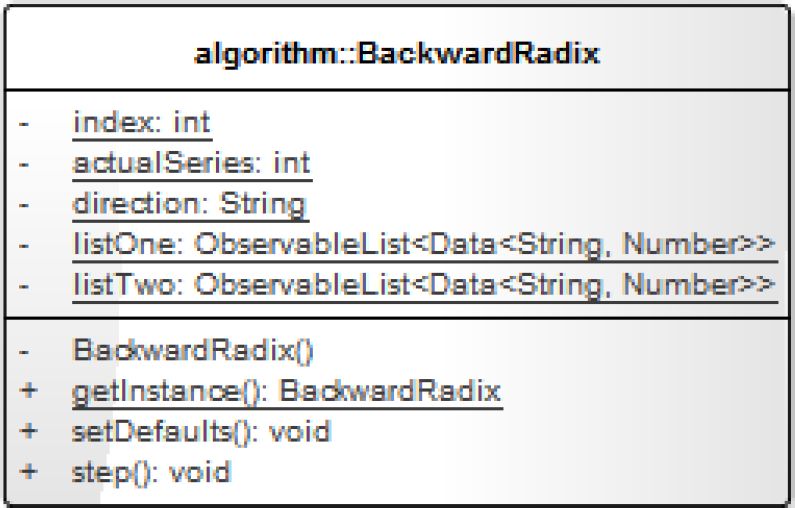
A setBucketColor(int lower, int upper) eljárás a két paraméterül kapott intervallumon beállíttatja a kontroller rétegben az oszlopok színét egységesre, így kijelöl egy edényt. A step() metódus használja az örökölt recursiveCall listát.



1. ábra. A ForwardRadix osztály diagramja

BackwardRadix osztály. A radix "vissza" rendezést valósítja meg, ősosztálya a RadixAlgorithm. Az index változó tárolja az aktuálisan vizsgált elem indexét. Az actualSeries azt a tömböt, amelyben vizsgálni kell az elemeket. A direction változó megadja, hogy az aktuális résztömbön mely irányból kell végezni a vizsgála­tot. A listOne és listTwo változók megfeleltethetőek az algoritmus által használt két tömbnek.

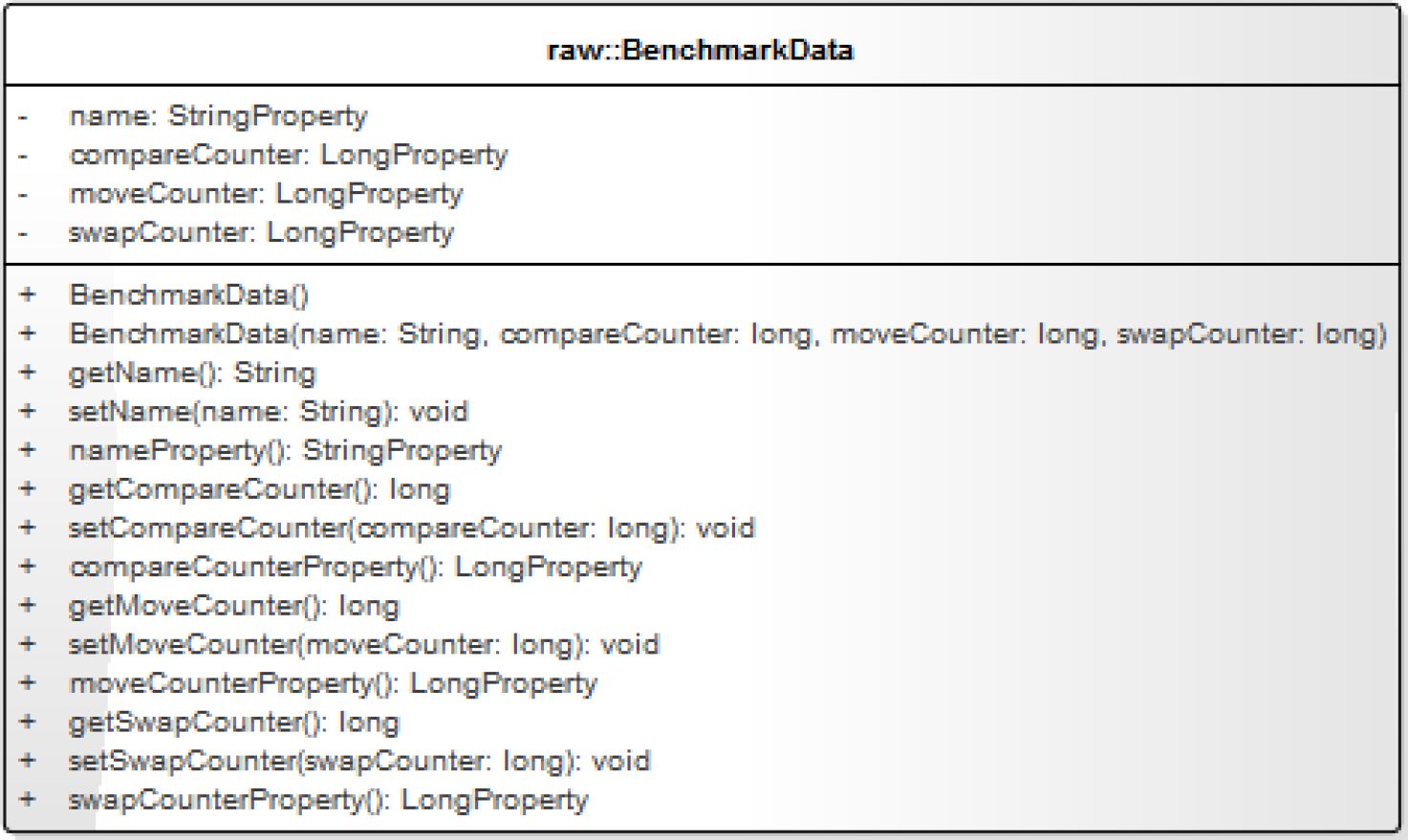
A step() metódus használja az örökölt recursiveCall listát.



1. ábra. A BaekwardRadix osztálydiagramja

Most az algorithm.raw csomag főbb osztályai kerülnek jellemzésre.

BenchmarkData osztály. A CounterData osztályhoz hasonló a funkciója és felépítése. Az algoritmus nevét és a rendezés során végzett műveletek összegét tá­rolja. A kontroller réteg használja fel a Összehasonlítás panel táblázatának a kitöl­téséhez. Minden rendezést végző programszál létrehoz belőle egy objektumot, majd hozzáadja egy listához, amit a kontroller réteg lekér egy szál befejeződésekor.



1. ábra. A BenchmarkData osztálydiagramja

SortingThread absztrakt osztály. A csomagban található rendezést megvaló­sító osztályok szülője. Őse a Thread osztály. A SortingThreadListener mezője tárolja azt az objektnmot(BenchmarkController), amelyet a szál befejeződése után figyelmeztet. Továbbá adattagjai tárolják a rendezés során végzett műveletek számát. A leszármazott osztályai a doRun() metódusát valósítják meg, amely a programszál elindításakor hívódik meg.

|  |  |  |
| --- | --- | --- |
|  | raw: .Sorting Thread | i bread |
| - | lËàener: SortingThrEadUstEnEr |  |
| \* | numbers: int{[|) |  |
| = | □omparisonCountEr: long =0 |  |
| # | swepCounter: long = 0 |  |
| # | mpvEOpuntEr: long = Q |  |
| + | doRunQ: void |  |
| + | SEtListener{l isten er: Sorting Thread Listen erji: void | |
| - | nptifyüstEnerj{): void |  |
| + | run{): void |  |

1. ábra. A SortingThread osztálydiagramja

A következőkben az Összehasonlítás panelen megjelenő informáeiőkhoz adato­kat szolgáltató konkrét osztályok kerülnek jellemzésre.

BubbleSortThread osztály. Megvalósítja az előbb említett doRun() metódust. Elvégzi a rendezést, majd a feladat befejeztével a kontrolién réteg Benchmark­Controller osztályában található BenchmarkDataList listához hozzáfűz egy új BenchmarkData objektumot.

raw::

Bu bble SortTh read

■+ doRuinQ: void

1. ábra. A BubbleSortThread osztály diagramja

InsertionSortThread osztály. Őse a SortingThread osztály. Felépítése telje­sen azonos a BubbleSortThread jellemzésénél leírtakkal. Ennélfogva az osztálydi­agramban sincs eltérés.

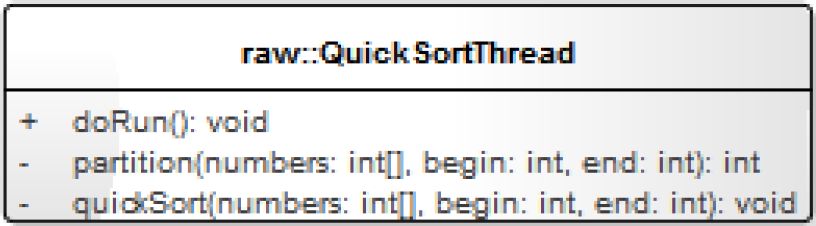
ShellSortThread osztály. Az egyetlen különbség az előző osztályokhoz képest, hogy rendelkezik egy további metódussal. A generateGapArray() metódus meg­egyezik a ShellSort osztály jellemzésében szereplő azonos nevű függvénnyel.

raw: : Shel I SortTh read

+ doRun{): void - flEnEratE0apAr,3Yi): IntepErf

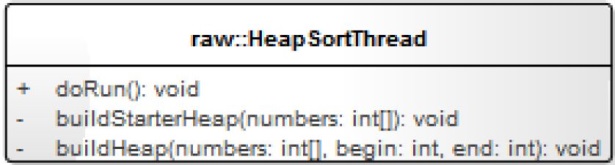
1. ábra. A ShellSortThread osztálydiagramja

QuickSortThead osztály. Két további metódussal rendelkezik, ezektől eltekintve megegyezik az BubbleSortThead felépítésével. A doRunQ metódusból kiemelésre került a tömb felosztását végző fiiggvény(partition(int[] numbers, int begin, int end)), valamint a rekurzív hívásokat végző quickSort(int[] numbers, int begin, int end) eljárás.



1. ábra. A QuiekSortThread osztálydiagramja

HeapSortThread osztály. A QuickSortThread-hez hasonlóan két metódus ke­rült kiemelésre a doRun() metódusból. Az egyik a buildStarterHeap(int[] num­bers) eljárás, amely felépíti a kezdőkupaeot. Továbbá a buildHeap(int[] num­bers, int begin, int end) függvény, amely ellenőrzi a kupaetulajdonságot.



1. ábra. A HeapSortThread osztálydiagramja

TournamentSortThread osztály. Felépítését tekintve a BubbleSortThread-

nál leírtakon felül egy egész számokat tartalmazó tömb adattagja van. Ez a tesztelés miatt került kiemelésre, mivel az algoritmus amennyiben nem kettő hatvány a be­menet hossza további (-1) értékekkel tölti fel a tömböt, amíg nem lesz megfelelő a mérete.

raw::

Tou ma ment SortTh reá d

- finalNumbeis: int (QJ ■+ doRun{): void

1. ábra. A TournamentSortThread osztály diagramja

Az info alesomag osztályai kerülnek bemutatásra

InfoFlyWeight osztály. Egyetlen adattagja egy szöveges mező. Ebben a válto­zóban tárolódik egy algoritmushoz tartozó leírás.

i nfo: :l nfoFI jWeighit

- i nfoT Ext: S-tri ng

+ InfcFIyWe-ightíinfsTElít: Sliing) + gEtText{): Sliing

1. ábra. Az InfoFlyWeight osztálydiagramja

InfoFlyWeightFactory osztály. Az előző osztály közreműködésével megvaló­sítja a pehelysúlyú programtervezési mintát. Mivel a felhasználó többször is lekér­heti egy algoritmus leírását, ezért célszerű egy objektumban tárolni ezt az adatot, így nem szükséges minden lekérésnél újra betölteni a leírást tartalmazó fájlt. Egy HashMap adatszerkezetben kulcs-érték párokként tárolódnak a már betöltött al­goritmusleírások.

A String getInfo(String algorithmName) metódusa ellenőrzi, hogy a para­méterként kapott algoritmus név (például: Gyorsrendezés) kulcs szerepel-e a Hash- Map-ben. Amennyiben igen, visszaadja a kulcshoz tartozó InfoFlyWeight objek­tumot. Ellenkező esetben betölti az algoritmus leírását, példányosít egy új InfoFly- Weight-et, majd hozzáfűzi a HashMap-hez,

info:: I nfoFIyWeig h tFa ctory

- i nfs F Iv'vVe ig hts: H as-h M 3 p < St-i rg I rf s F lyvVEig ht> {rEa d On ly}

+ gEtlnfslalgcithmNamE: String): String

3.3E ábra. Az InfoFlyWeightFactory osztálydiagramja

Kontroller réteg osztályai

Main Application osztály. Ugyan az alapcsomag része, funkcionalitását tekintve mégis a kontroller osztályok közé sorolható. Az osztály tartalmazza a program main() metódusát. A szoftver indításakor ez a metódus hívódik meg.

A start() függvény létrehoz egy "színpadot"(Stage), amely az ablaknak felel meg. Beállítja az ablak címsorát, betölti az alkalmazás ikonját. Majd meghívja a InitMenuLayout metódust.

Az imént említett eljárás betölti a főmenü elrendezését tartalmazó FXML fájlt, ciZclZ cl MenuLayout .fxml-t.

*Application*

5 rtng lg rttim 5:: Ma i n Appl ¡cat ion

* pri m □ ryStag e: StagE
* mEnuLayaut: BcrdErPanE

+ start(primaiyStagE: StagE): void + ntMenuLayout!J: vaid + mainjargs: String fi: vaid

1. ábra. A MainApplieation osztálydiagramja

MenuController osztály. A MenuLayout.fxml fájl betöltésével egyidejűleg példányosul. A felület eszköztárán a Segítség -> Névjegy pontjára kattintva fut le a handleAbout() metódus. Megjelenít egy alertBoxot, benne a program névje­gyének szövegével. Továbbá a Fájl -> Bezárás eszköztár elemet kiválasztva meghí- vódik a szoftver leállását eredményező handleExit() eljárás. A további függvényei a MenuLayout.fxml fájlban definiált gombokon történő kattintást kezelik, és be­töltik a kiválasztott bemenettípushoz tartozó FXML fájlt.

controller: MenuController

* mEnuPanE: BcrdErPanE
* manualBtnHandler(EVEnt: Acti on Event): void
* randomBtnHandlerfEvent: Acti on Event): void
* filEBtn Hand lEr| Event: Acti an Event): void
* h a nd leAbout{): void
* hendlEExit{): void

1. ábra. A MenuController osztálydiagramja

ManualInputController osztály. A ManualInputLayout.fxml fájl betölté­sével példányosul, ekkor lefut az initialize() metódus, mely letiltja a kattintást a felület két gombján.

A setSizeLimit() metódus figyeli a méret mező tartalmát, amennyiben nem pozitív egész szám a gombok továbbra is tiltva maradnak. Továbbá, ha a megadott hossz nagyobb száznál, akkor automatikusan átírja az értéket 100-ra. Amennyiben a méret megadása sikeres meghívődik a createFields() eljárás, mely létrehozza a számok beírására lehetőséged adó mezőket.

A randomBtnHandler() metódus a Feltöltés gombon történő kattintást köve­tően fut le. Ekkor feltölti a létrehozott mezőket 0 és 100 közötti véletlen számokkal. A saveNumbers() metódus átadja a rendezendő számokat a SortingAlgorithm osz­tályának. Ezt követően betölti a program főpaneljeit tartalmazó BaseLayout.fxml fájlt.

|  |  |
| --- | --- |
| controller::  ManualInputController | |
| - | sizEFÍEld: T ExtFiEld |
| - | randomBtn: Button |
| - | ofcBtn: Button |
| - | scroll Pa nE: Scroll Pa nE |
| - | numb-ErsPane: Gfid Pane |
| ' | size: int |
| + | initial íze!): void |
| + | createFielcfe{): voi-d |
| - | SEítSizeLimit{): void |
| + | randa m Btn H a n d 1 eî{): voit |
| + | orBtnHandleri}: void |
| + | ssueN u m bersQ: void |

1. ábra. A ManualInputController osztálydiagramja

RandomlnputController osztály. A RandomlnputLayout.fxml fájl betöl­tésekor jön létre belőle egy példány. Ekkor az initialize() metódus eljárás letiltja az Ok gombon való kattintást, továbbá meghívja az initBox() függvényt.

Az imént említett eljárás feltölti a generálás módjára lehetőséget adó legördülő listát az opeiőkkal. Az okBtnHandler() kezeli az Ok-on történő kattintást. Itt történik meg a bemenet méretét tartalmazó mező ellenőrzése.

Amennyiben megfelelő a hossz formátuma, azaz egy pozitív egész szám meghí- vődik a generateNumbers() eljárás. A typeBox változóban tárolt legördülőlista kiválasztott elemétől függően kerülnek a számok generálásra. A kigenerált tömböt átadja SortingAlgorithm osztályának, majd betölti a program főpaneljeit tartal­mazó BaseLayout.fxml fájlt.

controller::

Ra n d Dm I n putC on trol ler

* sizEFiEld: TEKtFiEld
* typ-EB ex: Com boB ox<Stri ng>
* okBtn: Button
* typ-E: String

+ initialize!}: void

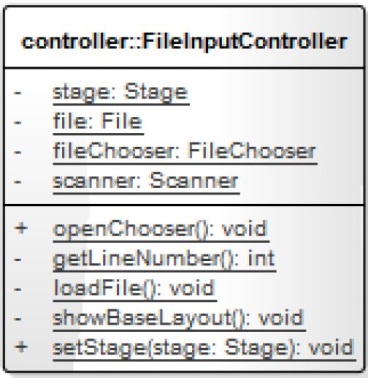
* inUBomO: void
* genersteN u mbefsQ: void + okBtnHandlet{): void

1. ábra. A RandomlnputController osztálydiagramja

FilelnputController osztály. A FilelnputLayout.fxml betöltésekor példányo­sul. Az openChooser() eljárás beállítja a formátumszűrőt(\*.txt), valamint megje­leníti a fájltallózó párbeszédpanelt.

A fájl kiválasztását követően meghívja a loadFile() függvényt.Kezdetben lekéri a fájl sorainak a számát a getLineNumber() eljárás segítségével, majd létrehoz egy ennek megfelelő nagyságú tömböt. Soronként végigolvassa a fájlt, amennyiben egy nem megfelelő sort talál (azaz egy sor üres vagy nem egy pozitív egész számot tartalmaz) befejezi a beolvasást, és hibaüzenetet jelenít meg.

Amennyiben a beolvasás sikeres volt átadja a tömböt a SortingAlgorithm osz­tálynak, és megnyitja a főpaneleket tartalmazó FXML fájlt.



1. ábra. A FilelnputController osztálydiagramja

BaseController osztály. A BaseLayout.fxml-hez tartozó kontroller osztály. Az initalize() metódusa ellenőrzi a bemenet hosszát, amennyiben 100 elemnél több került megadásra a tabPane panel választóról leveszi a Megfigyelés lehetőséget.

A további metódusai kezelik az eszköztár elemein való kattintást. A handleE- xit() eljárás a Fájl -> Bezárás kiválasztását követően fut le. A MenuControl- ler-hez hasonlóan a handleAbout() megjeleníti a program névjegyét.

A handleAlgorithm() metódus lekéri a megjelenített panel listájából vagy táb­lázatából a kiválasztott algoritmust. Ezt az információt felhasználva a InfoFly- WeightFactory osztálytól elkéri a rendezés leírását, majd létrehoz egy ablakot, amelyben megjeleníti az algoritmus jellemzését.

control ler:: Ba seC ontrDl ler

tabFane: TabFanE mEnuLayout: BordErFanE stage. Stags

initialize!): void handlEREtumToMEnu{): void handlEAIgorithm{): void handleAbout{): void handlEExitf): void

1. ábra. A BaseController osztálydiagramja

OverviewController osztály. Az OverviewLayout.fxml betöltésével példá­nyosul. Az initalize() letiltja a Léptetés és Lejátszás gombokon való kattintást.

Az animation adattagja egy TimeLine típusú objektum, melynek segítségével definiálható egy a felületen végrehajtandó műveletsorozat. Jelen esetben az aktuális

algoritmus step() műveletét ismétli amíg nem kerül megnyomásra a Leállítás gomb vagy nem fejeződött be a rendezés.

Az osztály további metódusai kezelik a panel gombjain történő kattintást, az initBtns() eljárás hozzárendel egy-egy eseménykezelőt a gombokhoz. A checkBut- tons() ügyeli, hogy befejeződött-e már a rendezési feladat, amennyiben igen, akkor az Újraindítás gombot jeleníti meg.

(lantról ler::  
Overvie-wC ontrol ler

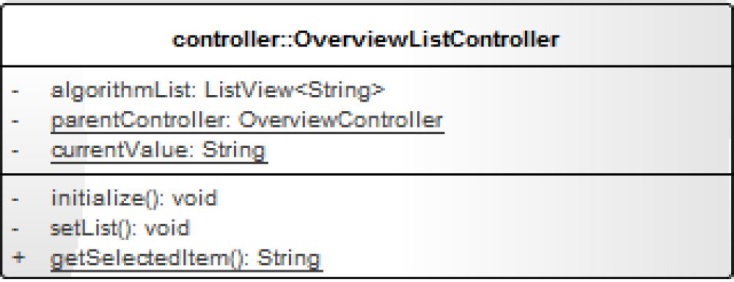
* stEp-Btn: Buttsn
* animBtn: Buttnn

+ displayPsnE: BplitFanE

* arirraticr: Tin'ElirE
* animatiiirSpEEd: int
* iniíialize(): veid
* rttBtnsQ: void
* SEtAn i m ati on{}: void # chEotButtonsO: void = relaadButtoní{): void

1. ábra. A OverViewController osztálydiagramja

OverviewListController osztály. A OverviewListLayout.fxml fájl betölté­sekor jön létre belőle egy példány. A setList() eljárás feltölti az Összegzés panelen látható listát a választható algoritmusok nevével. Az initialize() függvényben hoz­zárendelődik egy "figyelő" a listához. Amennyiben új érték kerül kiválasztásra a listában lefut a kiválasztott algoritmushoz tartozó setDefaults() metódus, továbbá betöltődik az Állapotjelző felület megfelelő elrendezése.



3.39. ábra. A OverviewListController osztálydiagramja

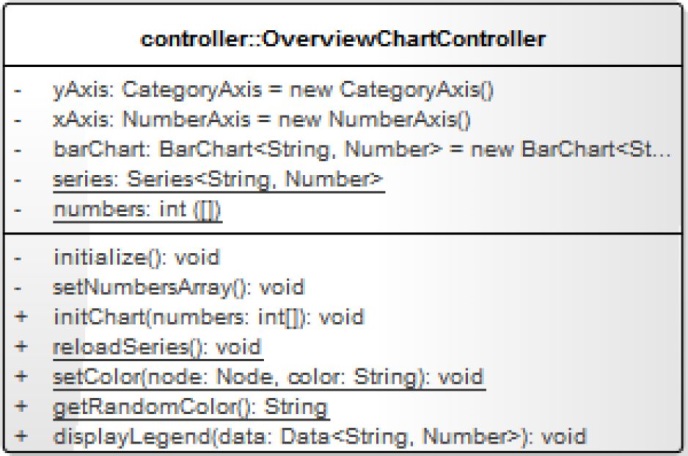
&DntrDller::OverviewLÍ5tCDntrDller

* algo'ithmUst: ListVÍE.v<51ring>
* parEntC-ontrollEr: OvefviewOonliollEr
* currEntValuE: St'ing
* ntializeQ: void
* SEtList{): void

+ qEtSelEctEdltEm;;i: Biting

OverviewChartController osztály. A OverviewChartLayout.fxml kontrol­ler osztálya. Feltölti az Megfigyelés panel diagramját a SortingAlgorithm által tárolt tömb értékeivel. Az osztályon keresztül módosíthatók az oszlopok színei, eh­hez a setColor(Node node, String color) metódusát szükséges meghívni. A getRandomColor() eljárást a Radix "előre" algoritmus használja az edények kijelöléséhez.

A tervezéskor már kiderült, hogy a JavaFX alapértelmezetten nem jeleníti meg az egyes oszlopok konkrét értékét. így szükséges volt a displayLegend(Data<String, Number> data) eljárás létrehozása, amely egy oszlop felett elhelyez egy szöveg­dobozt, benne a pontos értékkel.



1. ábra. A OverviewChartController osztálydiagramja

OverviewDoubleChartController osztály. Az előbb ismertetett osztálynak két példányát tartalmazza. A Radix "vissza" algoritmus kiválasztásakor kerül betöl­tésre a hozzátartozó FXML fájl, ekkor készül belőle egy objektum.

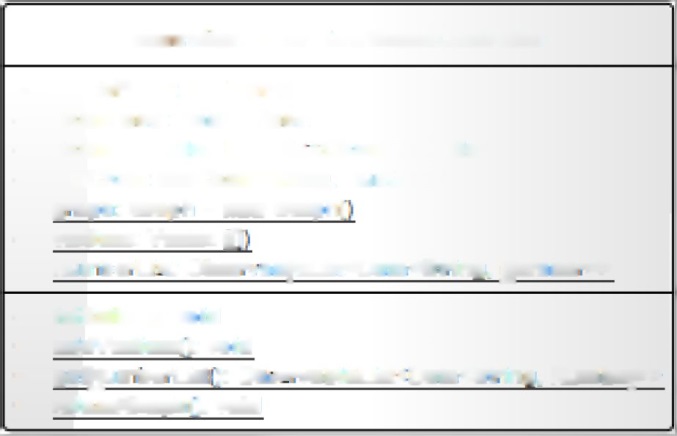
GontrDl ler:: Ov-erviewDou bleC ha rtC ontrol ler

* paneQn-e: AnchofPanE
* panaTwo: AnchofFana
* cc ntral I ErC n e: OvErvi E .v^ h a IC c ntra 11 e r
* ccntrslIarT'Va: OvEtviE,v£ha d" : ntrz 11 ar
* barChartOnE: BarCharKStíing. Numkei>
* bar£hartTwo: Bar£hart<Stíing, Nunnkei>
* listOnE: ObsgrvablEÜst<Data<5trinq. NumbEr»
* listTyvo: ObsErvablELisKData<51iinq. NumbEf»
* initializE^: void

+ relsas Listai): veid

1. ábra. A OverviewDoubleChartController osztálydiagramja

OverviewGraphController osztály. Az algoritmus listából bináris fát használó eljárás kiválasztását követően példányosul. "Karbantartja" a Állapotjelző felület gráfját. Az addVertices() függvény hívását követően hozzáadja a panelhez a fát. Az oszlop diagramját egy OverviewChartController-en keresztül lehet vezérelni.

control ler:: Ov&rviewG ra phC ontrol ler

~ graph Pa ne: AnchorPane

* chartPane: Anchc-rPane
* chartController: Overvie»,.,ChartControl ler
* barChart: BsirChsrKS-tring, Number>
* graph: Graph = na> ¡3raph;
* vertices: V&rtEx ;[í
* nl n~ i:erL¡ =-t: Ofass r-.-abjeü 5-t<D a ta ■=: S tring, jjurrbe r>>
* initializ-ef): veid

+ addVErtices!): veid

+ getjjLn'HtrLiEl ): Qt:>rvabIeLi;■£ =:Da ta ■=:Strirí;, Nurrb^[>> + rel sad Graph!): void

1. ábra. A OverviewGraphController osztály diagramja

SortingThreadListener interfész. Az interfészt megvalósító osztály értesíthető egy programszál befejeződését követően a notifyOfThreadComplete(final Th­read thread) eljáráson keresztül. A BenchmarkController valósítja meg. Ezáltal értesül róla, ha egy rendezési feladat befejeződött.

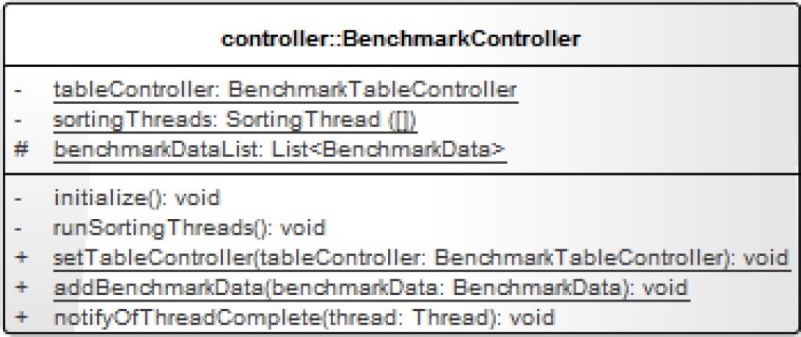
«intErfaoe»

control ler:: Sorti ngTh re a d L i siener  
+ notifyOfThrEadComplEtEithrEad: Thread): void

1. ábra. A SortingThreadListener interfészdiagramja

BenchmarkController osztály. Az előbb bemutatott interfészt valósítja meg. A sortingThreads tömbben tárolja a futtatandó programszálak egy példányát. Valamint a benchmarkDataList által tárolt BenchmarkData objektumokban találhatóak meg az algoritmusok által végzett műveletek száma.

A runSortingThreads() eljárás a sortingThreads tömb minden elemére meg­hívja a start() függvényt, ezáltal elindulnak a programszálak. Amennyiben egy rendezési eljárás befejeződött hozzáadja az benchmarkDataList listához a vég­zett műveletek számát, és értesíti az osztályt, hogy olvasson egy elemet a listából. A notifyOfThreadComplete() eljárás hozzáadja az új elemet a BenchmarkTab- leController-en keresztül a felület táblázatához.



BenchmarkChartController osztály. Egy listában tárolja azon elemeket, ame­lyekre már a felület táblázatán történt kattintás. Amennyiben olyan elem kerül kivá­lasztásra, amely még nem eleme a diagramnak hozzáadja a addElement(BenchmarkData data) metódus hívásával. Ebben a metódusban kerül ellenőrzésre, hogy már eleme-e a diagramnak az adott algoritmus. Amennyiben igen eltávolításra kerül.

control 1er: : Benchma rkC ha rtControl 1er

* barChart: B a rCh a rt<Stri ng, Long>
* ccn^F3rECcL^tE,: SE,Í£j<St,ing. Long>
* moveCountEr: Seri ES<Btii ng, Long>
* loaded ItEms: List<String>
* initialize!): void + initChart{): void

+ addElEmEnt;dgtg: B e n eh marrData ;r void

3.46. ábra. A BenchmarkChartController osztály diagramja

A OverviewTableController és BenchmarkTableController osztályok nem kerültek bemutatásra, mivel egyetlen funkciójuk, hogy rajtuk keresztül érhetőek el a grafikus interfész táblázatai.

Megjelenítési réteg osztályai

A megjelenítésért felelős komponensek főként FXML fájlokban kerülnek táro­lásra. A gráf implementációja azonban ilyen módon nem készíthető el. Ezért került kialakításra a view.graph csomag. Röviden most e csomag osztályai kerülnek jel­lemzésre.

Vertex osztály. A gráf egy csúcsát megvalósító osztály. Adattagja két JavaFX komponens, egy címke(Label) és egy kör(Circle), Konstruktora megkapja a kör X és Y pozícióját az ablakon, továbbá opcionálisan a címke vagy szín értékét.

A getGraphics() metódus visszaadja a csúcs grafikus reprezentációját. A set- Color(String color) metódus segítségével megváltoztatható a kör adattag színe.

*Group*

g ra ph : : Verte»

* aide Circle {readOn ly}
* number: int

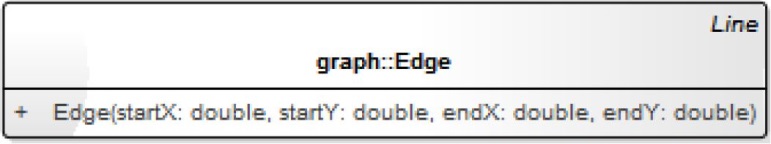
+ VeftEx(centefX: doublE, centefY: doublE, radius: doublE, color Btiing, numbEr: int) + VeftEx[centefX: doublE, centefY: doublE, radius: doublE, numbEr: int)

+ VeftEx(centErX: doublE, centErY: doublE, radius: doublE, color Btiing)

+ Vertex(centErX: doublE, centErY: doublE, radius: doublE)

+ setColof{oolof: Btiing): void + getG ra F h I cs{): LabEl

Edge osztály. Konstruktora a paraméterül kapott koordináták felhasználásával létrehoz egy vonal(Line) objektumot.



3.48. ábra. Az Edge osztálydiagramja

Graph osztály. Őse a Group osztály, ezáltal hozzáadható a felhasználó felület­hez. Az addVertex(Vertex vertex) metódusával új csúcs adható hozzá a gráfhoz. A bindVertexes(Vertex vertexA, Vertex vertexB) eljárás a két paraméterül kapott csúcsot köti össze a createEdge(Vertex vertexA, Vertex vertexB) me­tódus segítségével.

*Group*

g ra ph: :G ra pb

+ addVeiteK{veftex: VErtEx): void

+ bindVErtEXEs(vErtExA: VErtEX, VErtExB: VErtEx): void + CTEHtEEdgE{vertExA: VeitEX, vertexB: VEriex): EdgE

1. ábra. A Graph osztálydiagramja
2. Tesztelés

Felhasználó felület tesztelése

A felhasználó felület tesztelése manuálisan történik. A használati eseteket alapul véve a következő komponenseket szükséges tesztelni:

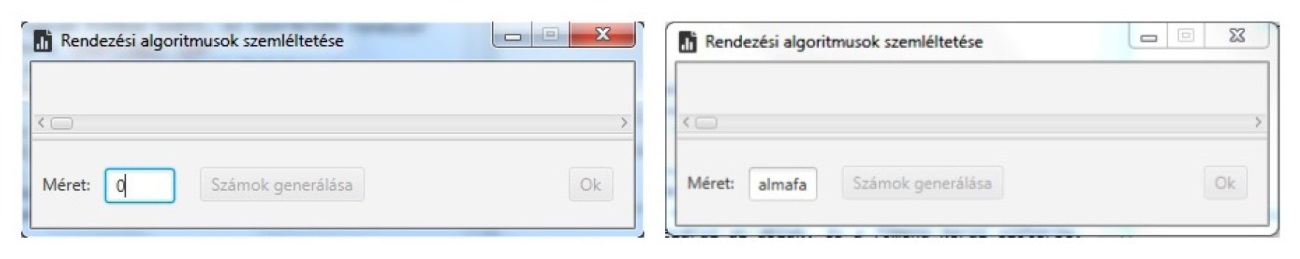
Főmenü eszköztára. Fájl -> Bezárásra kattintva a program futása leáll, az operációs rendszer Feladatkezelőjében sem fut tovább a JRE.

Segítség -> Névjegyre kattintva megjelenik a szoftver leírása. Az Ok gombra kattintást követően bezárul az ablak, és a főmenü kerül előtérbe.

Főmenü gombjai. Az egyes gombokra kattintva megjelennek a bemenet meg­adására alkalmas panelek.

Számok manuális megadása panel. Csúszka ellenőrzése, amikor nem férnek ki a beviteli mezők.

Kezdetben nem kattintható egyik gomb sem. A "Méret:" mezőt kitöltve például az almafa szöveggel vagy 0 értékkel, majd leütve az ENTER billentyűt továbbra sem kattinthatőak a gombok.



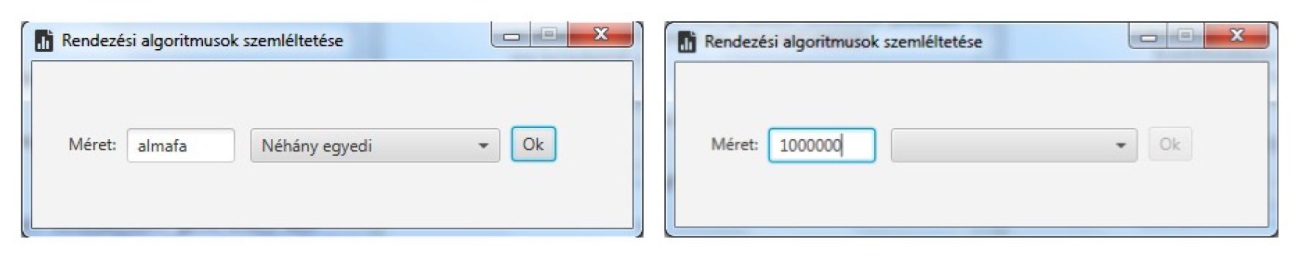
1. ábra. Xem pozitív egész szám megadása

Megfelelő elemhossz megadása után, majd átírva nem megfelelőre ismét inaktívvá válnak a gombok.

A sikeres hossz és rendezendő értékek megadását követően megjelenik a program főpanelje.

Számok generálása panel. Legördülő lista elemeinek az ellenőrzése.

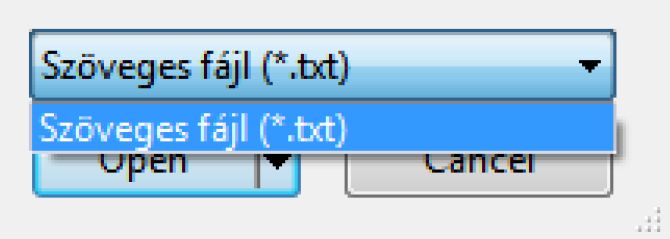
Amennyiben esak a méret került megadásra, és még nines kiválasztott generálási mód az Ok gomb nem kattintható. Amennyiben a módszer kiválasztásra került, de a méret nem megfelelő vagy nines kitöltve az Ok-ra történő kattintásnak nines hatása, továbbra is a panel látszik.



1. ábra. Xem pozitív egész szám és módszer megadása

A sikeres méret és generálási módszer megadását követően megjelenik a program főpanelje.

Fájl betöltése párbeszédablak. Fájlkiterjesztés szűrő ellenőrzése, esak a " Szöveges fájl (\*.txt)" opciót tartalmazhatja, ennek megfelelően a tallózóban esak az ilyen kiterjesztésű fájlok láthatók.



1. ábra. Fájlkiterjesztés legördülő lista

Xem megfelelő fájl megadását követően hibaüzenetet jelenít meg a program. Ilyen "hibás" fájlokat találunk a resources\input könyvtárban:

* üres fájl (input\_empty.txt)
* a fájl egy karaktersorozatot tartalmaz (input\_almafa.txt)
* pozitív egész számok az elemei, de van üres sora (input\_oneemptyline.txt)

Megfelelő fájl tallózásakor (például: input^ok) megjelenik a főpanel.

Főpanel eszköztára. Szintén elvégezendők a Főmenü eszköztárnál leírt tesz­tek, annyi különbséggel, hogy a Névjegy bezárást követően a főpanelt kell látnunk.

Továbbá tesztelendő funkció a Segítség -> Az algoritmusról menüpont. A Megfigyelés panel az aktív, és a listában nincs kiválasztott elem, akkor hibaüzenet jelenik meg. Ugyanez érvényes arra az esetre, ha a Összehasonlítás panel táblá­zatában nincs kiválasztott sor. Egy elem (például: "Kupaerendezés") kiválasztását követően megjelenik az algoritmus leírása, majd az Ok gombra kattintva bezárul az ablak, és a főpanel látható.

További funkció a Fájl -> Vissza a főmenübe lehetőség, ezt kiválasztva a főmenü jelenik meg.

Megfigyelés panel. A Léptetés és Lejátszás gombok inaktívak amíg nincs kivá­lasztott algoritmus. Például a "Gyorsrendezés"t kiválasztva már aktívak. A Leját­szásra kattintva a gomb szövege Megállításra változik. A rendezés befejeződését követően az Újraindítás gomb látható.

A Léptetés gombra kattintva egy következő állapot látható az Állapotjelző felületen. Ha egy elemű a bemenet, és például a buborékrendezés van kiválasztva, akkor a gombon történő kattintást követően az Újraindítás gomb jelenik meg. Több elem nem jelölhető ki egyszerre az algoritmus listában.

Összehasonlítás panel. A "Buborékrendezés" sorára kattintva hozzáadódik a diagramhoz az összehasonlítások és mozgatások összege. Ismételt dupla kattintást követően pedig eltűnik ez az információ.

Az oszlopok szélessége állítható, így szükséges tesztelni a táblázat alján lévő csúszkát.

Egységtesztelés

Az algoritmusok és műveletszámuk helyességének ellenőrzéséhez egységtesztek készültek. Ezáltal nem feltétlenül szükséges fehérdoboz megközelítéssel ellenőrizni a rendezési eljárásokat.

Mivel az algorithm csomagban található eljárások nagy része megegyezik a al- gorithm.raw csomag algoritmusaival, ezért célszerűbb az utóbbi osztályaihoz tesz­teket készíteni, egyszerűbb felépítésük és kisebb mértékű függőségük miatt.

Az algortihm.raw.test tartalmazza az egysógtesztosztályokat. Felépítésüket te­kintve azonosak, mindegyikben öt eset kerül ellenőrzésre. A tesztelést végző metó­dusok, azaz a tesztesetek a következők:

zeroElement(). Üres tömböt ad át rendezésre. Ekkor a rendezés után ellenőrzi, hogy szintén egy üres tömböt kapott-e vissza valamint, hogy a műveletek száma nulla-e.

oneElement(). A 42 értékkel hívja meg a rendezési eljárásokat. Ezt követően ellenőrzi, hogy ugyanezt az egy elemet tartalmazó tömböt kapta-e vissza a rendezést követően, valamint, hogy a műveletjelző értékek nullák-e.

flveConcretElement(). A [42, 20, 7,18,100] tömbön indítja el a rendezéseket. A rendezés befejeztével meghívja a Java beépített rendező algoritmusát(Arrays.sort), és ellenőrzi, hogy a rendezés befejeztekor tárolt tömb azonos-e az előbb említett függvényhívás eredményével. Továbbá ellenőrzi az algoritmus által végzett műveletek összegét.

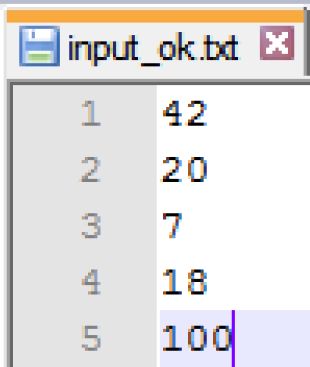
thousandRandomElementQ. Száz véletlen választott értéken hívja meg a ren­dezéseket. A műveletek száma nem kerül ellenőrzésre, esak az előbb említett módon a tömb tartalma.

millionRandomElement. Az előző eljárással azonos, mindössze a tömb mérete különbözik. Ez feleltethető meg a "nagy" bemenetre történő ellenőrzésnek, mivel esak ezek a rendezések hívódnak meg száz elemnél hosszabb bemenet megadásakor.

Strukturális tesztelés

Fehérdoboz tesztnek vetjük alá a programot, mely során vizsgáljuk, hogy a le­hetséges futási út során a megfelelő eredményeket produkálja-e a szoftver.

Miután elindult a program válasszuk a Fájlból betöltés lehetőséget. Adjuk meg a megjelenő fájltallózóban a resources\input könyvtárban található input\_ok.txt fájlt.



A betöltést követően megjelenik a program főpanelje. Látható, hogy a Megfi­gyelés panel az alapértelmezett, mivel a bemenet kevesebb, mint száz elemet tar­talmaz. A legtöbb algoritmus lejátszására már nines szükség, mivel működésük a Egységtesztelés során már ellenőrzésre került. Azonban a radix rendezések mivel nem jelennek meg a Összegzés panelen így ellenőrizendők.

Válasszuk ki az algoritmus listából a Radix "előre" rendezést. Ekkor az osz­lopok felett az értékek binárisan vannak ábrázolva. A Lejátszás gombra kattintást követően várjuk ki, amíg befejeződik a rendezés. Ekkor látható, hogy az összes oszlop színe • és megjelent a Újraindítás gomb. Az Állapotjelző táblázatban pedig a következő értékek szerepelnek:

|  |  |
| --- | --- |
| Tulajdonság | # |
| Vizsgálatok | 14 |
| Cserék | 3 |
| Aktuálsi bit | 5 |
|  | |
|  |  |

3.54. ábra. Radix "előre" rendezést követően az állapotjelzők

Amint látható a rendezésnek nem volt szüksége az összes bitet megvizsgálni, mivel 14 elemvizsgálatot követően már minden edény hossza egy volt.

Most ellenőrizzük a Radix "vissza" algoritmus helyességét. A listából kivá­lasztva megjelenik két oszlopdiagram egymás felett. A felső diagram tartalmazza a rendezendő értékeket. A Lejátszásra kattintva megfigyelhetjük, hogy hogyan törté­nik az értékek átmásolása egyik "tömbből" a másikba. A rendezés befejeztével a felső diagramon szerepelnek az értékek növekvő sorrendben, ekkor minden oszlop ® színű és látható a Újraindítás gomb. Az Állapotjelző táblázat értékei a következők:

|  |  |
| --- | --- |
| Tulajdonság | # |
| Vizsgálatok | 35 |
| Aktuálsi bit | 1 |
|  |  |
|  | |
|  |  |

3.55. ábra. Radix "vissza" rendezést követően az állapotjelzők

A vizsgálatok száma azonos a bitek számának és a tömb méretének szorzatával, azaz 7 ■ 5 = 35. Az aktuális bitet jelző szám a rendezés befejeztével nyilván 1.

Kérjük le a Radix "vissza" rendezés leírását az eszköztár Segítség -> Az al­goritmusról pontján való kattintással, majd zárjuk be. Ekkor továbbra is a Meg­figyelés panelt kell látnunk a befejeződött rendezési feladattal.

Navigáljunk át a Összehasonlítás panelre. Láthatjuk a kitöltött Összegzés táblázatot, valamint az üres diagramot. Adjuk hozzá a Gyorsrendezést és Bu­borékrendezést a diagramhoz, azaz kattintásunk kétszer ezeken a sorokon. Ekkor a következőt látjuk:



3.56. ábra. Összegzés panel

Ismételten kattintva a Gyorsrendezésen eltűnik a hozzá tartozó oszlop, már esak a Buborékrendezés műveletszáma látható a diagramon.

Válasszuk ki az eszköztár Segítség menüjének Névjegy menüpontját, aminek a hatására azonnal látható lesz a program névjegye. Zárjuk be a névjegyet, majd a Fájl menü Bezárás menüpontjával állítsuk le a szoftver futását.

1. fejezet

• •

Összegzés

Remélem, hogy a dolgozat segítséget nyújt azoknak, akiket érdekelnek a rendezési algoritmusok. Próbáltam az eljárásokat a lehető legszemléletesebben bemutatni, úgy, hogy az befogadható legyen a hallgatók számára és elősegítse tanulmányaikat.

Az alkalmazás fejlesztése során sokat tanultam. Mélyrehatóbban megismertem a program által bemutatott algoritmusokat, sikeresen használtam néhány program­tervezési mintát. Továbbá többet megtudtam a Java nyelv eszközeiről, valamint technológiai szempontból talán a legjelentősebb, hogy megismertem a JavaFX plat­formot és felépítését,

A szoftver továbbfejlesztése

A program a rendezési algoritmusok közül csak néhányat szemléltet, így ésszerű eél további eljárások implementálása. Mivel nagyjából minden rendezési algoritmus által használt adatstruktúra már adott, így egyetlen feladat ezen eljárások létreho­zása és "bedrótozása".

Továbbá elegánsabb megoldás lenne, ha minden algoritmushoz csak egy imple­mentáció tartozna, ez azonban ahogy a tervezés során kiderült nem igazán megvaló­sítható a JavaFX szálkezeléséből adódóan. Ez a fejlesztés még inkább csökkentené a kód redundanciáját.

Végszóként szeretném megköszönni Veszprémi Anna tanárnőnek, hogy elvállalta a projekt témavezetői szerepét és támogatott a dolgozat elkészítése során.

1. fejezet

Irodalomjegyzék

1. Dr, Fekete István: Algoritmusok és adatszerkezetek I. jegyzet, [ONLINE] [Hivatkozva: 2015,04,20] <http://people.inf.elte.hu/> fekete/algoritmusok\_bsc/alg\_l\_j egyzet/
2. Thomas H. Cormen - Charles E. Leiserson - Ronald L. Rivest - Clif­ford Stein: Új algoritmusok, Seolar kiadó, 2003, [992], 9789639193901
3. Rónyai Lajos - Ivanvos Gábor - Szabó Réka: Algoritmusok, Typotex Elektronikus Kiadó Kft., 2000, [350], 9789632790145
4. Demuth, H,: Electronic Data Sorting, PhD thesis, Stanford Univer­sity, 1956, [184]
5. Shellsort, Wikipedia the free encyclopedia, [ONLINE] [Hivatkozva: 2015,04,25] <http://en.wikipedia.org/wiki/Shellsort/>
6. Vaughan Ronald Pratt: Shellsort and Sorting Networks (Outstanding Dissertations in the Computer Sciences) Dissertations-G, 1980, [59], 0824044061
7. C.A.R, Hoare: Algorithm 6j: Quicksort Communications of the ACM, 4, 7, 1961
8. Java (programming language), Wikipedia the free encyclopedia, [ON­LINE] [Hivatkozva: 2015,04,21] <http://en.wikipedia.org/wiki/> Java\_(programming\_language)/
9. JavaFX, Wikipedia the free encyclopedia, [ONLINE] [Hivatkozva: 2015,04,21] http: //en. wikipedia. org/wiki/JavaFX/
10. JUnit, Wikipedia the free encyclopedia, [ONLINE] [Hivatkozva: 2015,05,01] http: //en. wikipedia. org/wiki/JUnit
11. FXML, Wikipedia the free encyclopedia, [ONLINE] [Hivatkozva: 2015,05,10] <http://en.wikipedia.org/wiki/FXML/>
12. The Takipi Blog: We Analyzed 30,000 GitHub Projects - Here

Are The Top 100 Libraries in Java, JS and Ruby, [ON­LINE] [Hivatkozva: 2015,04,30] <http://blog.takipi.com/>

we-analyzed-30000-github-projects-here-are-the-top-100

-libraries-in-java-js-and-ruby/

1. [↑](#footnote-ref-1)