



Eötvös Loránd Tudományegyetem

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Rendezési algoritmusok szemléltetése

Veszprémi Anna
mestertanár

Márföldi Péter Bence
programtervező informatikus BSc

Budapest, 2015

Tartalomjegyzék

1. Bevezetés	1
1.1. A feladat és annak értelmezése	1
1.2. Definíciók és jelölések	1
1.2.1. Oszd meg és uralkodj elv	1
1.3. Alkalmazott technológiák	1
1.3.1. A Java-ról röviden	1
1.3.2. JavaFX	2
2. Felhasználói dokumentáció	3
2.1. A vizsgált algoritmusok	3
2.1.1. Buborékrendezés	3
2.1.2. Beszűrő rendezés	3
2.1.3. Shell rendezés	3
2.1.4. Gyorsrendezés	4
2.1.5. Versenyrendezés	4
3. Fejlesztői dokumentáció	5
3.1. Tervezés és megvalósítás	5
3.1.1. Tervezés	5
3.1.2. Megvalósítás	6
3.1.3. Használt fejlesztőeszközök	6

1. fejezet

Bevezetés

Az bizonyos, hogy minden informatikus - beleértve a leendőket is - tanulmányaik kezdetén találkoztak a rendezési algoritmusokkal. Nagyszerű terület arra, hogy megérthessük a műveletigény kérdését, azt hogy mi számít igazán sok adatnak, vagy éppen, hogy mit értünk egy algoritmus stabilitásán.

1.1. A feladat és annak értelmezése

1.2. Definíciók és jelölések

1.2.1. Oszd meg és uralkodj elv

Egy algoritmus tervezési stratégia. Az elv[5] a következőképp old meg egy problémát:

- Az eredeti problémát felosztja kisebb az eredetihez hasonló típusú feladatokra.
- Rekurzívan megoldja ezeket a kisebb problémákat.
- A kisebb részek megoldásait összevonva, megoldást ad az eredeti feladatra.

1.3. Alkalmazott technológiák

A Következőkben röviden összefoglaljuk a Java[1] és a JavaFX[2] jellegzetességeit.

1.3.1. A Java-ról röviden

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 utóbbtól eltérően egyszerű objektummodellel rendelkezik.

A Java platformra készült programok túlnyomó többsége asztali alkalmazás. Ma-napság egyre több helyen találkozhatunk a Java nyelven írt programokkal, például mobil eszközökön, banki rendszereknél vagy akár egy szórakoztató elektronikai esz-közön. 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 be-rendezés és szoftvercsomag tartalmazza, így a nyelv egyaránt platformként és kö-zépszintként is működik. Összefoglalva a Java program három fontos szerepet tölt be:

- programozási nyelv
- köztes réteg (middleware)
- platform

1.3.2. JavaFX

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

A 2.0-ás verzióig a fejlesztők egy külön nyelvet használtak, amelyet *JavaFX Script*-nek neveznek. Azonban mivel ez szintén Java bájt-kódot generál a későbbiek-ben megadatott a lehetőség, hogy a programozók Java kódot használjanak helyette. A JavaFX egyik legnagyobb előnye, hogy egy egyszerű *XML* struktúrában leírhatók a program grafikus 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 oly-kor csak hosszas utánajárást követően sikerül megoldást találni egy-egy problémára.

2. fejezet

Felhasználói dokumentáció

2.1. A vizsgált algoritmusok

2.1.1. Buborékredezés

A legrégebbi és a legegyszerűbb rendezési algoritmus. Mindemellett a legtöbb esetben a leglassabb is. Már az 1965-ös évben megjelent egy teljes körű elemzése[3].

A rendezés minden egyes elemet összehasonlít a rákövetkező elemmel, és ha szükséges megcseréli őket. Mindezt addig, amíg nincs egy olyan menet, amelyben egyetlen elem sem cserél helyet. Ez azt eredményezi, hogy lépésenként a maximális elem "buborék" szerűen a lista végére kerül, ezzel egyidejűleg a kisebb elemek "lesüllyednek" a tömb elejére. Az algoritmus javítható azzal, hogy nem vizsgáljuk meg minden menetben a tömb összes elemét, hanem amennyiben egy maximális elem elérte a helyét visszavezetjük a problémát az eggyel "rövidebb" rendezési feladatra[4].

2.1.2. Beszűrő rendezés

A nevéből egyszerűen kikövetkeztethető az eljárás: beszűrja az elemeket a megfelelő helyükre a végleges tömbbe. Lényege, hogy a soron következő elemet egy ideiglenes változóba mentjük, és a rendezett tömb elemeit jobbra csúsztatjuk, mindaddig amíg a kiválasztott érték nem kerül a helyére. Kezdetben a tömb első elemét tekintjük rendezettnek. A legtöbb esetben akár kétszer hatékonyabb a Buborékredezéshez képest.[4] Továbbá kis (néhány száz) elemszámú bemenetre az egyik leghatékonyabb algoritmus.

2.1.3. Shell rendezés

Donald Shell nevéhez fűződik, a legtöbb esetben a leggyorsabb négyzetes idejű algoritmus. Többször vizsgálja a tömböt, és minden alkalommal egy részén beszűrő

rendezést hajt végre. Arra, hogy mekkora méretű résztömböt vizsgáljon az egyes lépésekben az algoritmus több javaslat is található. Az algoritmus nevét is adó Donald Shell $\lfloor N/2^k \rfloor$ ($k \geq 1$) lépésközt javasol. Azonban így az algoritmus időkomplexitása legrosszabb esetben $\Theta(N^2)$. Az algoritmus sebessége nagyban függ a lépésköz megválasztásától.

2.1.4. Gyorsrendezés

C.A.R. Hoare[6] alkotta meg 1965-ben. Az egyik leggyorsabb rendezési eljárás, ezért rendkívül gyakran alkalmazzák.

Helyben rendező, oszd meg és uralkodj[5] 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).
- A rendezendő részt vágjuk ketté, az egyik oldalára a vezérelemtől kisebb, míg a másikra a nagyobb elemek kerüljenek.
- Rekurzívan ismételjük meg az előbbi lépéseket a résztömbökön.

A rendezés műveletigényét befolyásolja, hogy hogyan választjuk meg a vezérelemet. Például a legnagyobb műveletigényt ($\mathcal{O}(n^2)$) eredményezi, ha mindig a legjobboldalibb elemet választjuk vezérelemnek, és a tömb elemei csökkenő sorrendben vannak. Éppen ezért a gyakorlatban javasolt ezen elem véletlenszerű megválasztása. A gyorsrendezés a legtöbb esetben(közepes és nagy méretű bemenetre) a legjobb választás ha számít a rendezés sebessége. Azonban ha a tömb elemei már eleve rendezettek vagy esetleg fordított sorrendben szerepelnek sajnos nem hatékony. Ezekben az esetekben ajánlatos másik algoritmust használni.

2.1.5. Versenyrendezés

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 "nyertest". A módszert $n=2^k$ inputhossz esetén érdemes alkalmazni, mivel ettől értérő bemenetre sokkal kedvezőbb eredményt lehet elérni a kupacrendezéssel. Az algoritmus által használt adatszerkezet egy teljes bináris fa.

3. fejezet

Fejlesztői dokumentáció

3.1. Tervezés és megvalósítás

A fejlesztés során több szempontot is figyelembe kell venni, úgy mint: művelet-igény, 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 egy-egy probléma megoldása más megközelítést kívánhat.

3.1.1. 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 programnak három jól elkülönülő részből 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 több rendezési eljárást is könnyedén meg lehessen jeleníteni a jelenlegiek mellett, így fontos szempont a kód egyszerű bővíthetősége.

3.1.2. Megvalósítás

Az első lépés a rendezési algoritmusok implementálása. Ezt követhette egyszerűbb felhasználói felület létrehozása. Kezdetben elegendő, ha csak egy grafikon jelenik meg, amely reprezentálja a tömbben található számokat.

Két algoritmushoz szükséges a gráfok reprezentáció, így a következő lépés egy gráf 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 összehasonlításokat/vizsgálatokat, mozgásokat, cseréket kell különböző színekkel jelölni az állapotjelző felületeken és számon tartani ezen műveletek összegeit.

Ezen a ponton az módosítás történt a projekt tervein. Eredetileg egy-egy 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, hogy kétszer kerüljenek implementálásra az algoritmusok. Az egyik implementációban számon tartjuk az éppen aktuális állapotot, és ez jelenik meg a felhasználói felületen. A másik megvalósításban pedig az algoritmusok azonnal lejátszódnak, így képet kaphatunk arról, hogy mennyi műveletre volt szükség a rendezés során. Ezen utóbbi implementációk mindegyike külön szálon fut, és ahogy valamelyik befejeződik figyelmezteti a főprogramot, hogy jelenítse meg a műveletek számát.

Miután a program alapjai elkészültek kezdetét veheti a felhasználói felület részletes kialakítása. Elsőként a diagram elhelyezése egy panelen, amely tartalmaz továbbá egy listát a választható algoritmusokról.

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.

Végül a rendezések összehasonlítására lehetőséget adó panel létrehozása, egy táblázattal, benne az algoritmusok műveletigényével. Továbbá egy diagrammal, amin megjelenik a táblázatból kiválasztott sor összehasonlításainak és mozgásainak a száma.

3.1.3. Használt fejlesztőeszközök

A fejlesztés *Eclipse SDK 4.4* fejlesztői környezet keretei között történt. A program grafikus fejlesztői felületet ad alkalmazások készítéséhez.

A program elkészítése során 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 elősegítette a *JavaFX Scene Builder 2.0*, melynek segítségével egyszerűen megtervezhetővé váltak a grafikus felület komponensei.

Az egész projekt megtalálható, és az egyes verziói visszakövethetők a GitHub-on:
<https://github.com/marfoldi/SRTNLGRTHMS>

A program fejlesztése során nem került sor külső függvénykönyvtár használatára.

Irodalomjegyzék

- [1] *Java (programming language)*, Wikipedia the free encyclopedia. [ONLINE] [Hivatkozva: 2015.04.21] [http://en.wikipedia.org/wiki/Java_\(programming_language\)/](http://en.wikipedia.org/wiki/Java_(programming_language)/)
- [2] *JavaFX*, Wikipedia the free encyclopedia. [ONLINE] [Hivatkozva: 2015.04.21] <http://en.wikipedia.org/wiki/JavaFX/>
- [3] Demuth, H.: *Electronic Data Sorting*, PhD thesis, Stanford University, 1956, [184]
- [4] Dr. Fekete István: *Algoritmusok és adatszerkezetek I. jegyzet*, [ONLINE] [Hivatkozva: 2015.04.20] http://people.inf.elte.hu/fekete/algoritmusok_bsc/alg_1_jegyzet/
- [5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Új algoritmusok*, Sclolar kiadó, 2003, [992], 9789639193901
- [6] C.A.R. Hoare: *Algorithm 64: Quicksort* Communications of the ACM, 4, 7, 1961 [321]