Gymnázium, Praha 6, Arabská 14
předmět Programování, vyučující Daniel Kahoun

Animace řadících algoritmů

ročníkový projekt

Kristián Kunc, 1.E

květen 2023

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu

.....

Anotace

Tento program umožní uživateli jednoduchou a intuitivní vizualizaci řadících algoritmů. Uživatel má na výběr z 10 známých algoritmů a před spuštěním může specifikovat velikost náhodného pole a má také možnost zapnout zvukové efekty. Každý algoritmus je implementován tak, že se po každé změně pole na chvíli zastaví, aby bylo možné změny pozorovat lidským okem. Krom animací je také k dispozici krátký popis a výkon každého algoritmu.

Abstract

This program allows the user to have a simple and intuitive visualization of sorting algorithms. The user can choose from 10 well-known algorithms and before running them, they can specify the size of the random list and also have the option to turn on sound effects. Each algorithm is implemented in such a way that it stops for a moment after each change in the array so that the changes can be observed by the human eye. In addition to animations, a brief description and performance of each algorithm are also available.

Összefoglalás

Ennek a programnak a használatával a felhasználó egy szimpla és intuitív vizuális megjelenés kap a szortírozó algoritmusukról. A felhasználó választhat 10 ismert algoritmus közül, és lefuttatásuk előtt megadhatják a méretét a random listának illetve ki/bekapcsolhatják a hangeffektusokat. Mindegyik algoritmus úgy van implementálva, hogy megálljon egy pillanatra minden listaváltozató művelet után, hogy megtekinthető legyen az emberi szem által is. Animációkon kívül egy rövid leírás, illetve egy hatékonysági mértéket is kap a felhasználó

Özet

Bu program kullanıcıya sıralama algoritmalarının basit ve sezgisel görselleştirme-lerini sunar. Kullanıcı, 10 tanınmış algoritma arasından seçim yapabilir, çalıştır-ma öncesi rastgele dizinin büyüklüğünü belirleyebilir ve ses efektlerini etkinleştir-me seçeneğine sahip olabilir. Her algoritma, değişimler insan gözü tarafından gözlemlenebilsin diye her değişim sonrası bir an bekleyecek şekilde yazılmıştır. Animasyonlara ek olarak, her algoritmanın kısa açıklaması ve performansı da mevcuttur.

Obsah

1 Zadání **4**

2 Úvod **4**
2.1 Co je to řadící algoritmus 4
2.2 Měření výkonu algoritmu 4
2.3 Stabilita algoritmu 5
2.4 Jak algoritmus animovat? 6

3 Využité nástroje **7**

4 Důležité části implementace **7**
4.1 Konverze pole na matici 7
4.2 Zvukové efekty 8
4.3 Seznam algoritmů 9

5 Spuštění a ovládání **10**
5.1 Systémové požadavky 10
5.2 Spuštění jar souboru 10

6 Závěr **10**

7 Zdroje **11**

1 Zadání

Naprogramujte aplikaci, která bude vizualizovat různé řadicí algoritmy pro seřazení pole čísel. Aplikace by měla umožňovat uživateli vybrat z několika různých řadicích algoritmů, které se budou vizualizovat. Pro vizualizaci algoritmů použijte sloupce, jejichž výška reprezentuje hodnotu daného čísla. Aplikace by měla také umožňovat uživateli změnit velikost pole a zapnout zvukové efekty, které doprovází algoritmus a hrají frekvenci podle hodnoty změněného prvku. Pro měření výkonu algoritmu použijte asymptotickou složitost.

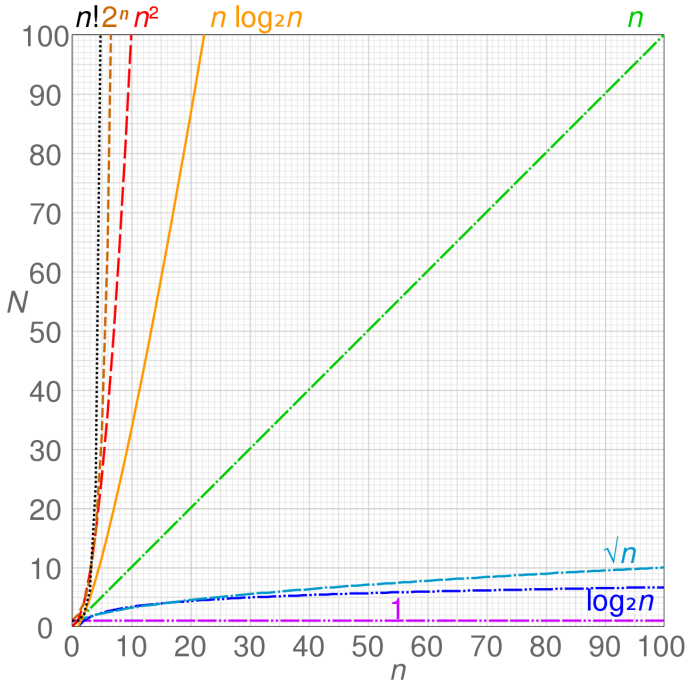
2 Úvod

2.1 Co je to řadící algoritmus

Řadící algoritmus je algoritmus který dokáže seřadit pole prvků do požadovaného pořadí podle specifikovaného kritéria. Jeden z častých příkladů využití řadícího algoritmu řazení množiny reálných čísel vzestupně, od nejnižšího po nejvyšší, kdy vstupní množina je zcela náhodně zamíchaná. Když je pole seřazené, je možné použít algoritmy jako například binární vyhledávání, které jsou mnohem rychlejší než klasické lineární vyhledávání.

2.2 Měření výkonu algoritmu

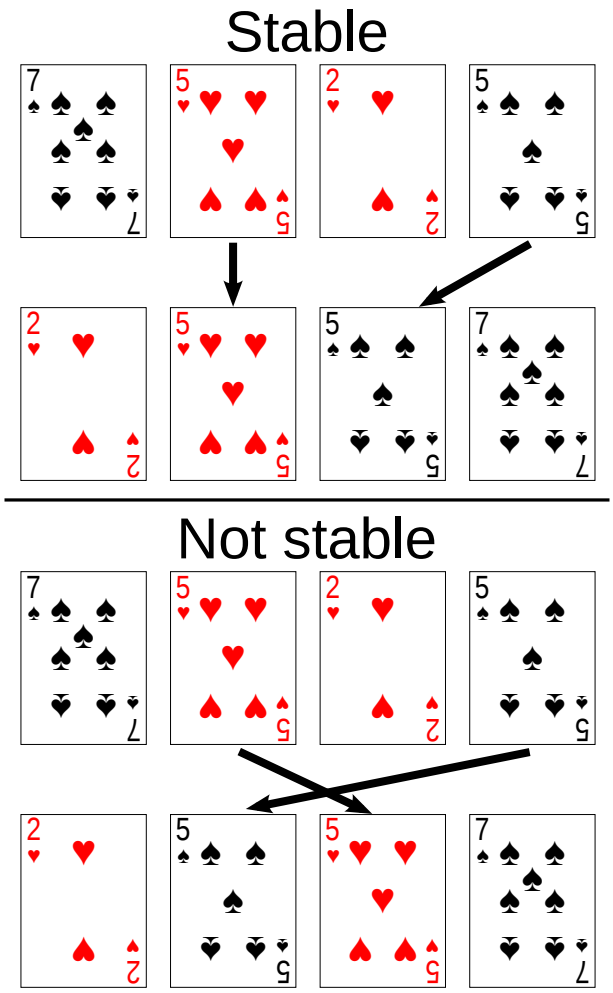
Pro měření výkonu algoritmu se běžné používá asymptotická složitost (v angličtině *big O notation*), která klasifikuje algoritmy podle růstu jejich doby běhu nebo využití paměti s rostoucí velikostí vstupu. V zápisu se často užívá proměnná n, která reprezentuje velikost vstupních dat. Například algoritmus s $O(n)$ bude rychlejší než algoritmus s $O(n^2)$. Na grafu můžeme sledovat jak rostou různé složitosti, kdy na osa x reprezentuje velikost vstupních dat a osa y čas.



Obr. 1: Grafické porovnání různých tříd složitosti s ohledem na změnu velikosti vstupních dat. [1]

2.3 Stabilita algoritmu

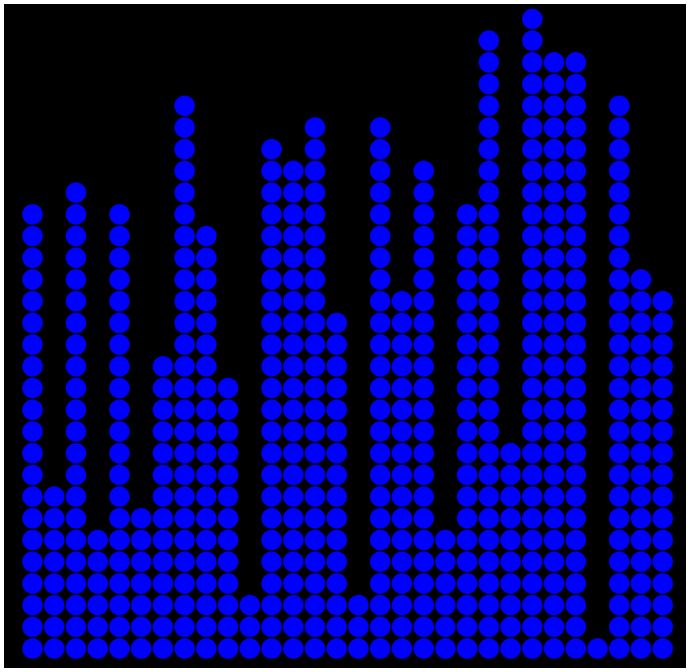
U algoritmu se také uvadí jestli je stabilní či ne což indikuje jestli algoritmus ponechá identické vstupní prvky ve stejném relativním pořadí, jako ve vstupních datech. Na příkladu je vidět algoritmus, který řadí karty podle číselné hodnoty vzestupně, ve vstupních datech jsou ale 2 karty se stejnou hodnotou a srdcová karta 5 stojí před pikovou katrou 5. V příkladu stabilního (*Stable*) algoritmu je toto relativní pořadí ponecháno a ve výstupu dodrženo. To neplatí pro nestabilní algoritmus (*Not stable*), který relativní pořadí těchto karet změní.



Obr. 2: Stabilita řadícího algoritmu na příkladu hracích karet. [2]

2.4 Jak algoritmus animovat?

Inspirace a způsob pro animaci a vizualizaci algoritmů vznikla převážně z GitHub repozitáře github.com/bingmann/sound-of-sorting, zveřejněný s licencí GPL v3.0. V tomto projektu jsou pro vizualizaci využité sloupce, jejichž výška reprezentuje hodnotu daného čísla. Tento nápad jsem převzal a implementoval pomocí třídy `Matrix` (*dále jen matice*) z knihovny `ArabTools`, která slouží pro vytvoření pole světel, která jsem následně použil pro vizualizaci. Z tohoto projektu jsem si také vzal inspiraci pro zvukové efekty, které si uživatel může zapnout a ty doprovází algoritmus a podle hodnoty změněného prvku zahrají danou frekvenci.



Obr. 3: Vizualizace náhodně zamíchaného pole čísel v matici

3 Využité nástroje

Jako vývojové prostředí jsem použil IntelliJ IDEA Ultimate od společnosti JetBrains, které je specializované na vývoj programů v Javě a nabízí mnoho funkcí pro zefektivnění vývoje jako například automatizované testování, profilování výkonu a podporu pro různé verzovací systémy. Další nástroj, který jsem využil při práci na mém projektu, je knihovna ArabTools. Tato knihovna nabízí různé funkce například pro pozastavení programu nebo samotnou matici.

4 Důležité části implementace

4.1 Konverze pole na matici

Pro konverzi pole čísel do výšky sloupců v matici je využit následující kód. V této implementaci se předpokládá že:

1. pole je stejně velké jako je matice široká
2. maximální hodnota prvků v poli nepřevyšuje výšku matice

Uplně na záčátku se vstupní pole zkopíruje do posledního aplikované pole případě, že je prázdne/neexistuje (funkci spouštíme poprvé). Poté se projde každý sloupec matice a nastaví se barva světla na modrou popřípadně červenou, pokud se změnila hodnota v poli na této pozici od posledního aplikování. Na konci se vstupní pole zkopíruje do posledního aplikovaného pole.

```java
public static void arrayToMatrix(Matrix matrix, int[] array) {
    if (lastAppliedArray == null) { // if this is the first time the array is
    ↪  being applied, set lastAppliedArray to the array
        lastAppliedArray = array.clone();
    }

    for (int i = 0; i < matrix.getWidth(); i++) { // apply the array to the
    ↪  matrix
        LightColor color = LightColor.BLUE;

        if (array[i] != lastAppliedArray[i]) {
            color = LightColor.RED;
        }
        for (int j = 0; j < matrix.getHeight(); j++) {
            if (j < array[i]) { // if the height is less than the value of the
            ↪  array at the index, turn on the light
                matrix.setColor(i, j, color); // set the color of the light
            } else {
                matrix.setOff(i, j);
            }
        }
    }
    lastAppliedArray = array.clone(); // set lastAppliedArray to the array
}
```

4.2 Zvukové efekty

Funkce na spouštení zvukových efektů má 2 parametry: value a max. Value je hodnota, která se má převést na frekvenci a max je maximální hodnota, která se může vyskytnout. Frekvence se vypočítá pomoci vzorce: 120 + (value / max) * (1212 - 120), která převede hodnotu na rozsah 120Hz-1212Hz za pomocí specifikované maximální hodnoty. Pak se vytvoří pole bajtů s délkou 1, který se použije pro zápis do sdl (`SourceDataLine` z knihovny `javax.sound.sampled`). Sdl se následné otevře a spustí. V cyklu se vypočítá úhel, který se použije pro výpočet sinu abychom dostali sinusovou křivku. Poté se vypočítá sinus úhlu a vynásobí se 100, aby byl hlasitější. Nakonec se zapišou data do sdl. Tato implememntace lehce zpomaluje animace vzhledem k tomu, že běží na stejném vláknu a vyústí ve zvukové artefakty, které mohou být nepříjemné, proto je zvuk v základním nastavení vypnutý.

```java
    if (isMuted) return; // if the sound player is muted, return

    // https://stackoverflow.com/a/929107/14762088
    double freq = 120 + (value / (double) max) * (1212 - 120); // converts to
    ↪  range 120hz-1212hz

    byte[] buf = new byte[1];
    try {
        sdl.open(af); // try to open the audio line
    } catch (Exception e) {
```

```
            e.printStackTrace();
        }

        sdl.start();
        for (int i = 0; i < 1000; i++) {
            double angle = i / (44100.0 / freq) * 2.0 * Math.PI; // 44100 is the
            ↪  sample rate, 2.0 is to make it louder, Math.PI is to make it a
            ↪  sine wave
            buf[0] = (byte)(Math.sin(angle) * 100); // Get the sine of the angle,
            ↪  multiply by 100 to make it louder
            sdl.write(buf, 0, 1); // write to audio line
        }
    }
```

4.3 Seznam algoritmů

Všechny algoritmy jsou uložené ve speciální třídě typu `Enum`, která obsahuje mnoho užitečných metod pro získaní specifického algoritmu jak už podle jeho `ID`, `jména` nebo rovnou získat všechny v jednom poli. Každý algoritmus má mnoho atributů jako například `prettyName` pro pěkné jméno užité v textech, `description` pro stručný popis, nebo `isStable` proměná datového typu `boolean` reprezentující fakt, zda je algoritmus stabilní či nikoliv.

```
public enum Algorithm {
    ...
    public static String[] getNames() { // get all the algorithm names
        String[] names = new String[Algorithm.values().length];
        for (int i = 0; i < Algorithm.values().length; i++) {
            names[i] = Algorithm.values()[i].prettyName;
        }
        return names;
    }

    public static Algorithm[] getAlgorithms() { // get all the algorithms
        return Algorithm.values();
    }

    public static Algorithm getById(int id) { // get an algorithm by id
        if (id < 0 || id >= Algorithm.values().length) {
            return null;
        }

        return Algorithm.values()[id];
    }

    public static Algorithm getByName(String name) { // get an algorithm by
    ↪  name, case insensitive, spaces are ignored
        for (Algorithm algorithm : Algorithm.values()) {
            if (algorithm.prettyName.replace(" ",
            ↪  "").equalsIgnoreCase(name.replaceAll(" ", ""))) {
```

```
                return algorithm;
            }
        }
        return null;
    }
}
```

5 Spuštění a ovládání

5.1 Systémové požadavky

Pro spuštění je potřeba mít nainstalovaný Java Runtime Environment (JRE) <= 17. Pro ověření přítomnosti požadopvané verze na Vašem počítači můžete použít `--version` parametr.

```
$ java --version
```

Pokud ve výstupu příkazu vidíme nižší verzi nebo příkaz nebyl nalezen, stáhneme nejnovější verzi pomocí instrukcí na java.com/en/download. Je nutné mít na paměti, že animace běží na jednom vlákně procesoru takže komplexnější animace mohou být na pomalejších procesorech nestabilní. Velikost matice $>= 50^2$ by měla bez větších potíží běžet plynule na běžném procesoru, je ale možné spustit i animaci na matici o rozměrech 300^2 na výkonějších procesorech.

5.2 Spuštění jar souboru

Jakmile jsme ujištěni, že máme požadovanou verzi, otevřeme si příkazový řádek v adresáří kde se nechází výše zmíněný `main.jar` a spustíme ho pomocí -jar parametru.

```
$ java -jar main.jar
```

Tento příkaz spustí intuitivní uživatelské rozhraní a všechny potřebné instrukce jsou vytištěné v příkazovém řádku. Postupně odpovídáme na dotazy programu (výběr algoritmu, velikost matice, ...) zadáváním čísel či jiných hodnot. Pokud chceme spouštět animace rychle nebo hromadně můžeme je specifikovat v parametrech programu samotného. Pro zobrazení návodů k příkazům použijeme parametr help.

```
$ java -jar main.jar help
```

6 Závěr

Celkově jsem velmi spokojen s výsledkem své práce, ale na druhou stranu bych určitěl poukázal na mnohé nedostatky, jako například špatný výkon při vykreslování větší matice na méně výkonných procesorech a nepříjemné zvukové artefakty. Práce ovšem splnila, ne-li překonala, má očekávání a rozhodně splnila účel - vizualizovat řadící algoritmy. Věřím, že může pomoci odtajnit uživatelům mnohdy neznámé či nepředstavitelné algoritmy schované za funkcemi `.sort()`. Určitě je možné práci dál posouvat například přidáním dalších algoritmů nebo více uživatelských vstupů a možností personalizace rozhraní.

7 Zdroje

1. Java dokumentace -
   `https://docs.oracle.com/en/java/`
2. ArabTools dokumentace -
   `https://vyuka.gyarab.cz/kahoun/download/ArabTools/javadoc/`
3. Řadící algoritmy, Wikipedia -
   `https://en.wikipedia.org/wiki/Sorting_algorithm`
4. Řadící algoritmy, GeeksForGeeks
   `https://www.geeksforgeeks.org/sorting-algorithms/`
5. Asymptotická složitost, Wikipedia
   `https://en.wikipedia.org/wiki/Big_O_notation`
6. The sound of sorting, GitHub
   `https://github.com/bingmann/sound-of-sorting`
7. Převod číslá na jiný rozsah, StackOverflow
   `https://stackoverflow.com/a/929107/14762088`
8. Vymazat příkazový řádek, StackOverflow
   `https://stackoverflow.com/a/32295974/14762088`
9. Maďarská anotace, Gábe
   `https://github.com/gabe56f`
10. Turecká anotace, geniusjam
    `https://github.com/geniusjam`
11. Vytvoření sinusoidy v Javě, GT's Blog
    `http://blog.gtiwari333.com/2011/12/java-sound-generate-play-sine-wave.html`

Reference

[1] Wikimedia Commons User:Cmglee. Grafické porovnání různých tříd složitosti s ohledem na změnu velikosti vstupních dat., 2007. File: Comparison computational complexity.svg |*online*|, 2013 [cit. 10.5.2023] Dostupné z `https://en.wikipedia.org`.

[2] Wikimedia Commons User:Dcoetzee, User:WDGraham. Stabilita řadícího algoritmu na příkladu hracích karet, 2013. File: Sorting stability playing cards.svg |*online*|, 2013 [cit. 10.5.2023], Dostupné z `https://en.wikipedia.org`.