

SPRAWOZDANIE KOŃCOWE

Projekt zespołowy

Mateusz Smoliński

Łukasz Knigawka

16 stycznia 2019

Spis treści

1	Wstęp	2
2	Opis algorytmu	3
3	Działanie programu	3
4	Zmiany względem specyfikacji funkcjonalnej	4
4.1	Argumenty wywołania	4
4.2	Sytuacje wyjątkowe	5
5	Zmiany względem specyfikacji implementacyjnej	6
5.1	Klasa CurrencyMatrix	6
5.2	Klasa DataReader	6
5.2.1	Metoda readData	6
5.2.2	Metoda parseCurrency	6
5.2.3	Metoda parseRate	7
5.3	Klasa ExchangeSearcher	7
5.4	Klasa GroupOfVisionaires	7
5.4.1	Metoda checkArgs	7
5.4.2	Metoda main	8
6	Wnioski	8

1 Wstęp

Sprawozdanie powstało w celu podsumowania projektu „LUPA”. Zawiera ono opis zastosowanego rozwiązania i algorytmu rozwiązującego problem podziału mapy na obszary, jak i pomysłów i koncepcji testowanych we wcześniejszych fazach projektu. Opisane zostały także zmiany względem specyfikacji funkcjonalnej i implementacyjnej, które zostały podjęte już w trakcie implementacji projektu. Na końcu dokument zawiera wnioski wyciągnięte w trakcie i po ukończeniu projektu, a także podsumowanie zajęć laboratoryjnych i subiektywna opinia co do tematów zadań.

Projekt „Lupa” ma za zadanie wczytanie z pliku tekstowego mapy, podanej w postaci punktów konturu i punktów kluczowych, a następnie wydzielenie obszarów na mapie na podstawie tych punktów kluczowych. W pliku otrzymujemy także deklarację obiektów, jakie mają być umieszczone na mapie, żeby umożliwić podsumowanie obiektów na danych obszarach. Program ma umożliwiać także dodawanie i usuwanie punktów kluczowych i konturowych w trakcie trwania programu.

Plik tekstowy z danymi przyjmuje następującą postać:

```
# Kontury terenu (wymienione w kolejnosci laczenia): Lp. x y
1. 0 0
2. 0 20
3. 20 30.5
4. 40 20
5. 40 0

# Punkty kluczowe: Lp. x y Nazwa
1. 1 1 KOK Krawczyka
2. 1 19 KOK Kaczmarskiego
3. 30 10 KOK Lazarewicza

# Definicje obiektow: Lp. Typ_obiektu (Nazwa_zmiennej Typ_zmiennej)*
1. SZKOLA Nazwa String X double Y double
2. DOM X double Y double L_MIESZKANCOW int
3. NIEDZWIEDZ X double Y double

# Obiekty: Typ_obiektu (zgodnie z definicja)
1. SZKOLA "Szkoła robienia duzych pieniedzy" 4 1
2. DOM 4 3 100
3. DOM 4 17 120
4. DOM 4 18 80
5. NIEDZWIEDZ 20 20
6. NIEDZWIEDZ 40 1
7. NIEDZWIEDZ 39 1
8. NIEDZWIEDZ 39 2
```

Program ma postać aplikacji z graficznym interfejsem użytkownika, co umożliwi wypełnienie wszystkich założeń projektowych. Został napisany w języku C#, na sprzęcie komputerowym zgodnym z tym zadeklarowanym w specyfikacji implementacyjnej.

2 Opis algorytmu

Poszukiwanie algorytmu dzielącego mapę na obszary okazało się największym wyzwaniem w toku realizacji projektu. Można je podzielić na trzy części:

- próby wykorzystania algorytmu Fortune’a,
- próby napisania własnego algorytmu opartego na podziale symetralnych,
- próby rozwiązania problemu na podstawie analizy pikseli.

Pierwsza część, czyli eksperymenty z algorytmem Fortune’a, była pierwszym założeniem, przyjętym jeszcze w fazie planowania projektu. Algorytm ten został wybrany ze względu na najlepszą złożoność czasową $O(n \log n)$. Algorytm okazał się jednak dużo bardziej złożony niż początkowo zakładaliśmy, a jego samodzielna implementacja byłaby trudna i czasochłonna. Stąd też pojawił się pomysł rozwiązania problemu algorytmem alternatywnym, napisanym przez nas na potrzeby tego konkretnego projektu.

Kolejny algorytm opierał się na własnościach diagramu Woronoja – w szczególności na tym, że dzieli on planszę na figury wypukłe. Logika algorytmu opierała się na analizie planszy z perspektywy pojedynczych punktów kluczowych, żeby wyznaczyć krawędzie ograniczające obszar wyznaczany przez dany punkt kluczowy. Analiza rozpoczynała się od matematycznego wyznaczenia najbliższej symetralnej spośród symetralnych pomiędzy tym punktem kluczowym a wszystkimi pozostałymi. Następnie program miał analizować wszystkie punkty przecięcia pozostałych symetralnych z wyznaczoną i wybierać najbliższe punkty, powtarzając tę czynność dla kolejnych symetralnych przecinających poprzednie aż do utworzenia zamkniętej figury lub napotkania konturu z obu stron. Ten algorytm jednakże również okazał się trudny do zaimplementowania i dopiero po kilku dniach możliwe było zobaczenie pierwszych rezultatów. Algorytm okazał się jednak nieskuteczny, czy to z powodu błędnej jego implementacji czy błędnych założeń matematycznych (algorytm nie przewidywał kilku przypadków) i choć działał poprawnie dla danych przykładowych, to przy większych planszach powodował błędy nie tylko w rysowaniu granic obszarów, ale także prowadząc do zapętlenia programu z przyczyn, których nie udało się ustalić.

Ostatnie podejście zakładało rozstrzygnięcie problemu najprostszą metodą, bardzo kosztowną czasowo dla programu, jednak nie wymagającą długiej implementacji. Tym razem rysowanie miało się odbywać na podstawie odległości dzielące punkty (piksele) na mapie, i na podstawie matematycznie wyznaczonej odległości od punktów kluczowych program miał przypisywać kolory do każdego punktu na mapie tym samym wyznaczając obszary w obrębie konturu. Ta metoda okazała się skuteczna i została zastosowana w ostatecznej wersji programu.

3 Działanie programu

Poniższe przykłady zostały wykonane na komputerze opisanym w specyfikacji implementacyjnej, uruchomione w cmd.exe.

Pierwszy rysunek ukazuje poprawne działanie programu dla pliku z danymi dane.txt zawierającego dokładnie te same dane, które zostały umieszczone we wstępie tego sprawozdania.

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../dane.txt 1000 EUR
USD
1000.0 EUR -> GBP -> USD -> 1137.23 USD
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 1: Uruchomienie programu z wyszukiwaniem ścieżki

Kolejny obraz przedstawia uruchomienie programu dla tych samych danych, ale z poszukiwaniem dowolnego arbitrażu.

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../dane.txt 1000
1000.0 EUR -> GBP -> USD -> EUR -> 1000.19 EUR
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 2: Uruchomienie programu – dowolny arbitraż

Na trzecim rysunku widzimy uruchomienie programu w trzeciej opcji – szukania arbitrażu dla konkretnej waluty.

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../dane.txt 1000 GBP
1000.0 GBP -> USD -> EUR -> GBP -> 1000.19 GBP
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 3: Uruchomienie programu – konkretny arbitraż

Kolejny obrazek to już niepoprawne uruchomienie programu – w tym wypadku użytkownik podał za mało danych, zabrakło przynajmniej podania kwoty wejściowej.

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../dane.txt
ERROR: program needs 2, 3 or 4 arguments to work properly
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 4: Uruchomienie programu – za mało danych

Piąty rysunek to próba uruchomienia programu dla pliku, który nie istnieje.

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../dan.txt 1000
ERROR: file dan.txt was not found. Make sure the file is located in proper place and try again.
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 5: Uruchomienie programu – niepoprawne dane

Dwa kolejne obrazki to przykłady uruchomienia programu dla plików, które nie zostały poprawnie zapisane. Są to pliki użyte do testowania programu, których treść zamieszczono poniżej. Obrazki ukazują reakcję programu na konkretne błędy w pliku tekstowym.

Pierwszy plik to data3.txt:

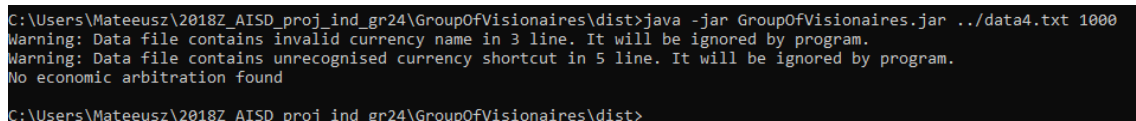
```
# Waluty (id | symbol | pelna nazwa)
0 EUR Euro
1 PLN Zloty
Kursy walut (id | symbol (waluta wejsciova)...)
0 EUR PLN 0.8889 PROC 0.0001
```

```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../data3.txt 1000
Warning: Data file contains invalid ID in 4 line. It will be ignored by program.
ERROR: Data file does not contain enough separating lines. Make sure that you have attached proper file and try again
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 6: Działanie programu z błędem krytycznym w pliku

Kolejny plik to data4.txt:

```
# Waluty (id | symbol | pelna nazwa)
0 EUR Euro
1 PLN
# Kursy walut (id | symbol (waluta wejscowa)...)
0 EUR PLN 0.8889 PROC 0.0001
```



```
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>java -jar GroupOfVisionaires.jar ../data4.txt 1000
Warning: Data file contains invalid currency name in 3 line. It will be ignored by program.
Warning: Data file contains unrecognised currency shortcut in 5 line. It will be ignored by program.
No economic arbitration found
C:\Users\Mateusz\2018Z_AISD_proj_ind_gr24\GroupOfVisionaires\dist>
```

Rysunek 7: Działanie programu z mniej ważnym błędem w pliku

4 Zmiany względem specyfikacji funkcjonalnej

4.1 Argumenty wywołania

Tak, jak zostało już wymienione w poprzedniej części sprawozdania, program może być teraz wywołany z tylko dwoma argumentami – w takim wypadku wyszukiwany jest dowolny arbitraż. W przypadku wywołania programu w taki sposób program wypisze jeden arbitraż dla pierwszej waluty, dla jakiej występuje – na przykład, jeśli w pliku z danymi znajduje się 6 walut i arbitraże występują dla waluty z numerami 2 i 4, program wypisze arbitraż dla waluty z ID równym 2.

4.2 Sytuacje wyjątkowe

W pierwotnej koncepcji w przypadku niewłaściwych danych w pojedynczej linii pliku z danymi program miał wypisać komunikat: „Warning: Data file contains invalid data in <number_of_line> line. It will be ignored by program.” Ze względu na zastosowanie wyjątków, które jest opisane szerzej w zmianach względem specyfikacji implementacyjnej, komunikaty wypisywane przez program są dokładniejsze. Na przykład, gdy w pliku z danymi wartość opłaty zawiera niewłaściwy znak, na przykład literę, program wypisuje komunikat: „Warning: Data file contains invalid fee value in <number_of_line> line. It will be ignored by program.” Może to ułatwić odnalezienie i poprawienie błędu użytkownikowi.

Specyfikacja funkcjonalna nie uwzględniała też przypadków niepożądanych, takich jak:

- opłata procentowa przekraczająca 1,
- kurs z waluty X na tą samą walutę X,
- kurs napisany dwukrotnie.

Ostateczna wersja programu ignoruje takie linie, również wypisując odpowiadające problemowi komunikaty.

5 Zmiany względem specyfikacji implementacyjnej

W strukturze całego programu nastąpiła jedna istotna zmiana – ze względu na wielkość pakietu *groupofvisionaires* przechowującego pierwotnie wszystkie klasy programu został wydzielony drugi pakiet, nazwany *filereader*, przechowujący klasę *DataReader* oraz wyjątki *ReadDataException* oraz *ParseLineException* używane przez program przy obsłudze błędów związanych właśnie z odczytem pliku. Ich dokładniejsze zastosowanie opisano poniżej, w sekcjach opisujących metody klasy *DataReader*.

5.1 Klasa *CurrencyMatrix*

Klasa odpowiedzialna za przechowywanie danych wczytanych z pliku nie uległa dużej zmianie względem specyfikacji implementacyjnej. Otrzymała jedynie jedno dodatkowe pole w postaci podawanej na początku istnienia klasy liczby *n*, odpowiadającej za ilość walut. Do metod dostępowych doszła zatem metoda *getN*, która wypisuje tę liczbę w innym miejscu kodu – w szczególności w trakcie iterowania po tablicy walut *currencies*, która jest prywatna i nie można dostać jej długości z zewnątrz.

5.2 Klasa *DataReader*

Klasa wczytująca dane z pliku została rozbudowana i ostatecznie składa się ona z trzech metod.

5.2.1 Metoda *readData*

- `public static CurrencyMatrix readData (File f)`
`throws ReadDataException`

Metoda *readData* zgodnie ze wcześniejszym założeniem zwraca przygotowany obiekt klasy *CurrencyMatrix* w przypadku powodzenia. Jeżeli w trakcie wczytywania wystąpi problem uniemożliwiający poprawne działanie programu, metoda wyrzuci wyjątek *ReadDataException*, który jest odczytywany w metodzie *main* klasy *GroupOfVisionaires*.

5.2.2 Metoda *parseCurrency*

- `private static Currency parseCurrency(String line ,`
`int lineNumber) throws ParseLineException`

Metoda *parseCurrency* powstała na późniejszym etapie projektu, w celu poprawy czytelności i usprawnienia obsługi błędów w klasie *DataReader*. Otrzymuje ona wczytaną linię z pliku z danymi oraz numer linii, która jest aktualnie wczytywana. W przypadku poprawnej analizy linii metoda zwraca obiekt klasy *Currency*, gotowy do użycia w dalszej części programu.

Jeśli w trakcie wczytywania nastąpi problem, metoda wyrzuci wyjątek *ParseLineException*, który jest odbierany przez metodę *readData*. W przeciwieństwie do wyjątku *ReadDataException* nie powoduje on zakończenia próby wczytania pliku, a jedynie zignorowanie błędnej linii i przejście do kolejnego etapu programu.

5.2.3 Metoda `parseRate`

- `private static void parseRate(String line ,
int lineNumber , CurrencyMatrix currencyMatrix)
throws ParseException`

Metoda *parseRate* powstała na późniejszym etapie projektu, podobnie jak poprzednia metoda, w celu oddzielenia obszernej procedury analizy linii z wymianą dwóch walut. W przeciwieństwie do metody *parseCurrency* nie zwraca ona przygotowanego obiektu, gdyż wybór indeksów w macierzy *CurrencyMatrix* jest bezpośrednio powiązane z odczytem tej linii oraz obsługą błędów. Otrzymuje ona wczytaną linię z pliku z danymi, numer linii, która jest aktualnie wczytywana oraz macierz *CurrencyMatrix*.

Jeśli w trakcie wczytywania nastąpi problem, metoda wyrzuca wyjątek *ParseException*, który jest odbierany przez metodę *readData*. Podobnie jak w przypadku metody *parseCurrency* nie powoduje on zakończenia próby wczytania pliku, a jedynie zignorowanie błędnej linii i przejście do kolejnego etapu programu.

5.3 Klasa `ExchangeSearcher`

- `public String SearchForArbitrage(double amount ,
CurrencyMatrix cm, Currency currency)`
- `public String SearchForBestExchange(double amount ,
CurrencyMatrix cm, Currency inputCurrency ,
Currency outputCurrency)`

Metody znajdujące się w tej klasie nie uległy dużej zmianie. Wykonują one dokładnie takie same czynności, jakie były przewidziane na wcześniejszym etapie projektu. Jediną różnicą jest to, że metody nie wypisują na standardowym wyjściu odnalezionych ścieżek (lub komunikatów), ale zwracają je do metody *main* w postaci obiektu klasy `String`.

5.4 Klasa `GroupOfVisionaires`

Również klasa główna projektu przeszła pewne zmiany w stosunku do planów ze specyfikacji implementacyjnej. Podstawową różnicą jest to, że zyskała ona pola odpowiadające danym odbieranym od użytkownika – zarówno tych z argumentów wywołania, jak i odczytywanych z pliku, co ułatwiło przekazywanie tych danych w programie.

5.4.1 Metoda `checkArgs`

- `public boolean checkArgs(String [] args)`

Metoda sprawdzająca argumenty wywołania została przygotowana tak, żeby realizować wykonanie programu także dla dwóch argumentów – dla wyszukiwania dowolnego arbitrażu. Ze względu na testy jednostkowe w programie w finalnej wersji programu metoda ta jest dostępna publicznie.

5.4.2 Metoda main

- `public static void main(String [] args)`

Metoda *main* nie zmieniała się znacząco, poza obsługą wyszukiwania dowolnego arbitrażu dla dwóch podanych przez użytkownika argumentów. Występuje w niej obiekt klasy *GroupOfVisionaires*, który funkcjonuje jak kontener dla danych i to na nim uruchamiana jest metoda *checkArgs*.

6 Wnioski

1. Zagadnienie diagramu Woronoja pomimo prostych założeń nie było oczywiste do rozwiązania w programie.
2. Popularny w rozwiązaniach zagadnień związanych z tym diagramem algorytm Fortune'a mógł być najlepszym rozwiązaniem pomimo swojej matematycznej złożoności. Zastosowanie go zdecydowanie poprawiłoby wydajność programu.
3. W przeciwieństwie do projektu indywidualnego, opracowanie algorytmu okazało się kluczowym problemem w trakcie projektu i pochłonęło ono zdecydowanie najwięcej czasu.
4. Tworzenie algorytmu od podstaw było ciekawym i kształcącym doświadczeniem, zarówno z perspektywy programistycznej, jak i algorytmicznej. Okazało się ono jednak ryzykowne i w związku z niepowodzeniem wymusiło na nas szybkie przygotowanie alternatywnego rozwiązania problemu.
5. Projekt zespołowy okazał się doskonałą okazją do rozwijania umiejętności korzystania z systemu kontroli wersji. Git posiada wiele narzędzi, które pomogły rozwiązać nam konflikty, przydatne okazało się także cofanie do wcześniejszych wersji kodu.

7 Komentarz do zajęć laboratoryjnych

bez komentarza