Politechnika Warszawska
Wydział Elektryczny

Sprawozdanie „Arbitrage”

Autor:
Grzegorz Kopyt

28 listopada 2018

Spis treści

1 Cel powstania dokumentu

Dokument ma na celu podsumowanie pracy nad projektem „Arbitrage". Przedstawia problem, jaki miał zostać rozwiązany, opisuje zastosowany algorytm oraz wskazuje zmiany dokonane w stosunku do wcześniejszych specyfikacji. Dokument zawiera również wnioski z podjętych decyzji oraz refleksje nad zastosowanymi rozwiązaniami.

2 Opis problemu

Program został postawiony przed dwoma zadaniami:

- znajdowaniem najkorzystniejszej ścieżki wymiany waluty,
- znajdowaniem dowolnego arbitrażu.

Jako źródło danych otrzymywał specjalnie spreparowany plik, który zawierał definicje walut oraz możliwych wymian (z informacją o kursach i opłatach). Na podstawie tego pliku oraz kwoty wejściowej program miał znajdować dowolny arbitraż. Natomiast do znalezienia najkorzystniejszej ścieżki wymiany waluty otrzymywał oprócz pliku i kwoty wejściowej, także walutę wejściową oraz docelową.

3 Wysoko abstrakcyjny opis działania algorytmu

1. Algorytm pobiera dane od użytkownika (w zależności od zadania są to dane opisane w sekcji 2).
2. Na podstawie tych danych algorytm tworzy graf walut, gdzie każda waluta zawiera możliwości jej wymiany na inną walutę.
3. Algorytm przechodzi kilkukrotnie po wszystkich walutach i możliwych wymianach (według algorytmu Bellmana-Forda) dokonując obliczeń wymiany określonej kwoty.
4. Jeśli znajdzie korzystniejszą ścieżkę dotarcia do węzła to aktualizuje jego dane.
5. W każdej walucie pozostawia największą kwotę oraz ścieżkę wymiany, którą można uzyskać z dotarcia do tej waluty od waluty początkowej.
6. W zależności od zadania:
   (a) Wymiana

i. Algorytm pobiera z waluty docelowej ścieżkę oraz kwotę końcową.

(b) Arbitrage

i. Algorytm pobiera z waluty początkowej, która jest także walutą końcową, ścieżkę wymiany oraz kwotę końcową.

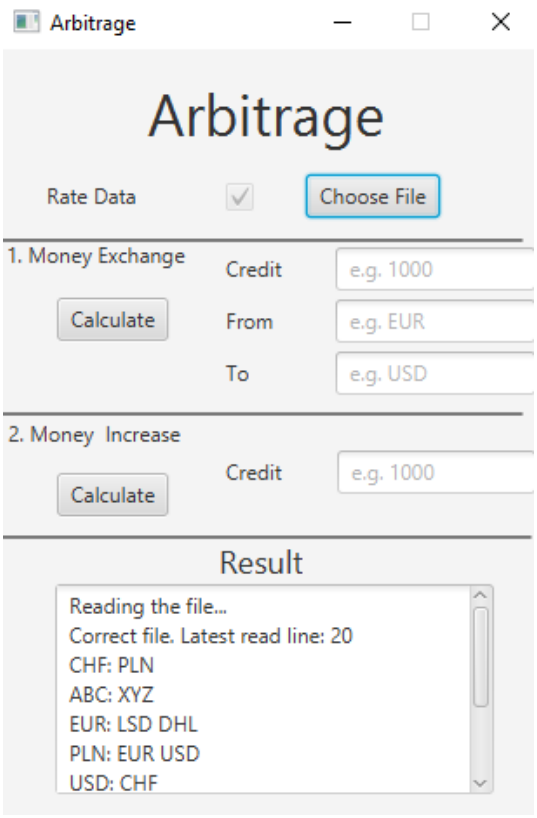
7. Algorytm wyświetla użytkownikowi wyniki swojej pracy.

---

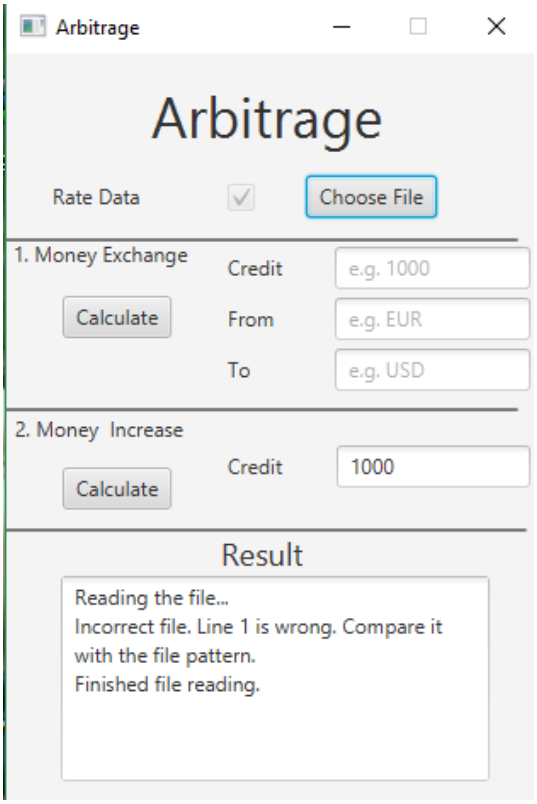
4 Efekty działania programu

W wyniku pracy programu pojawiają się następujące sytuacje:

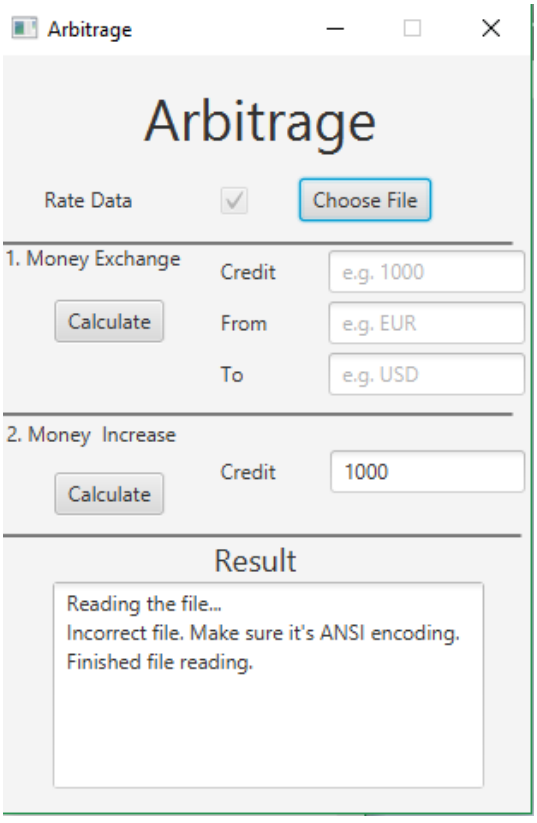
1. Poprawne wczytanie pliku.



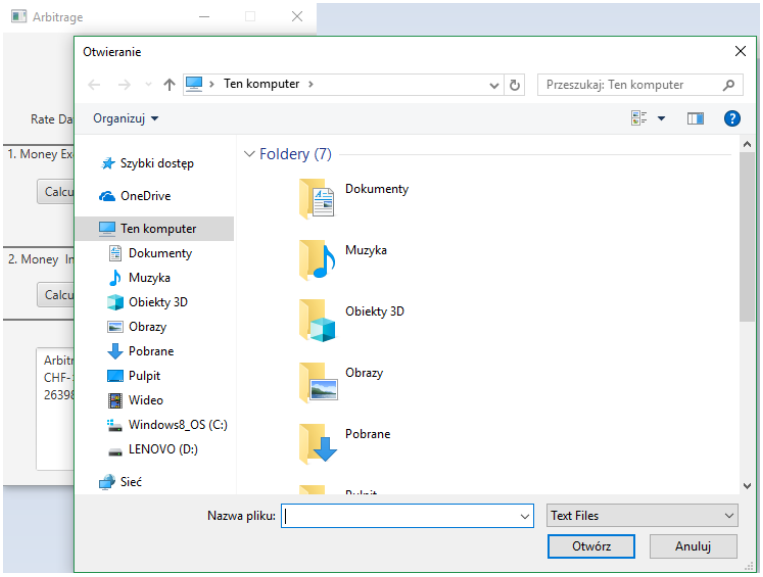
2. Niezgodność ze wzorem w pliku.



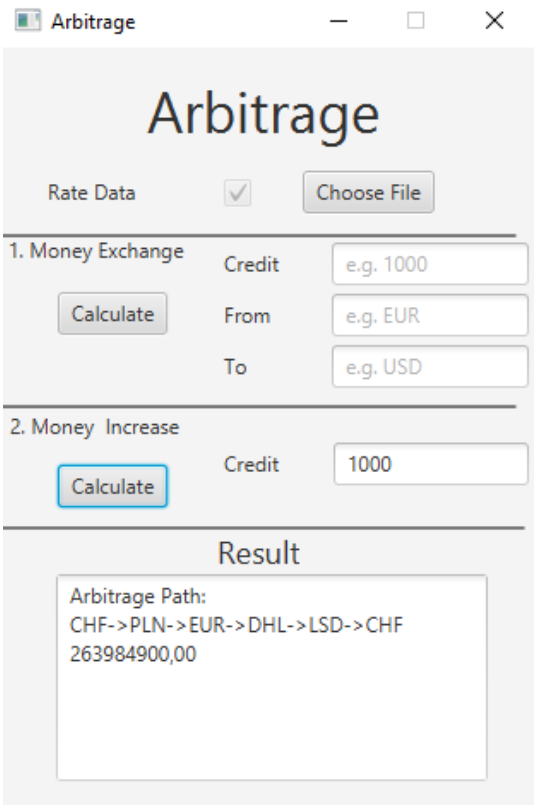
3. Niepoprawny format pliku.



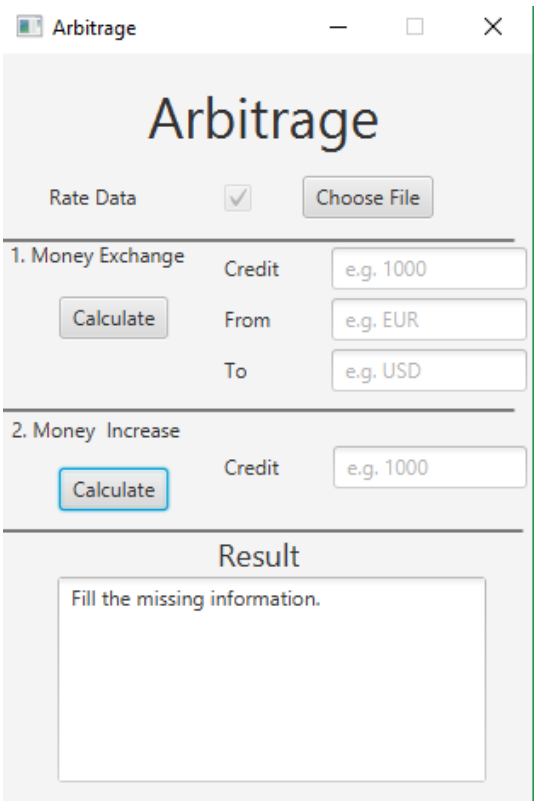
4. Wybór pliku po naciśnięciu *Choose File*.



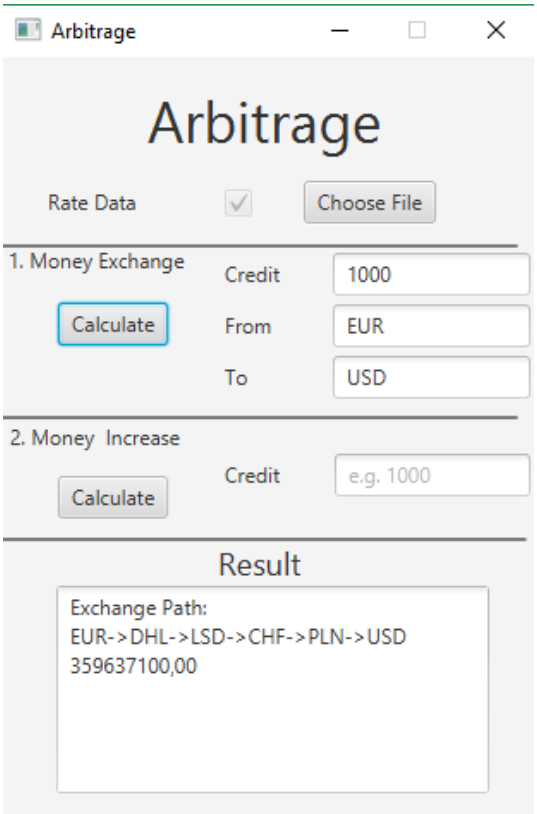
5. Znalezienie arbitrażu.



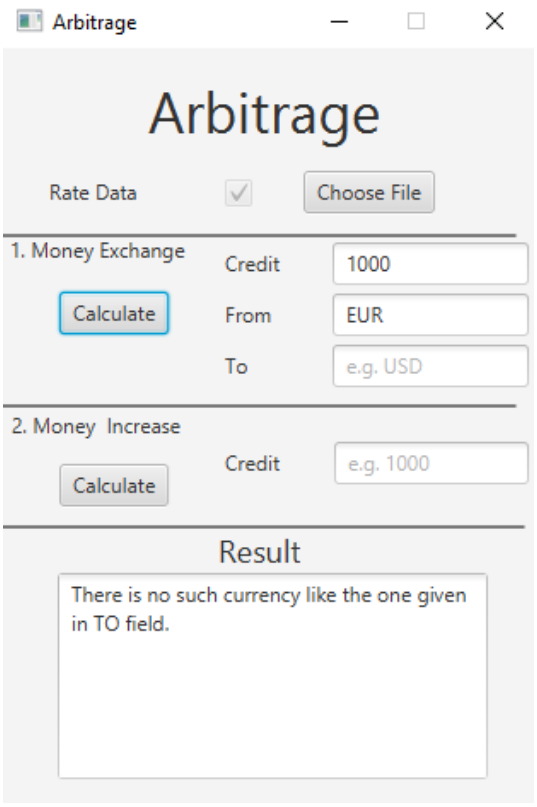
6. Nie podano danych do wymiany lub arbitrażu.



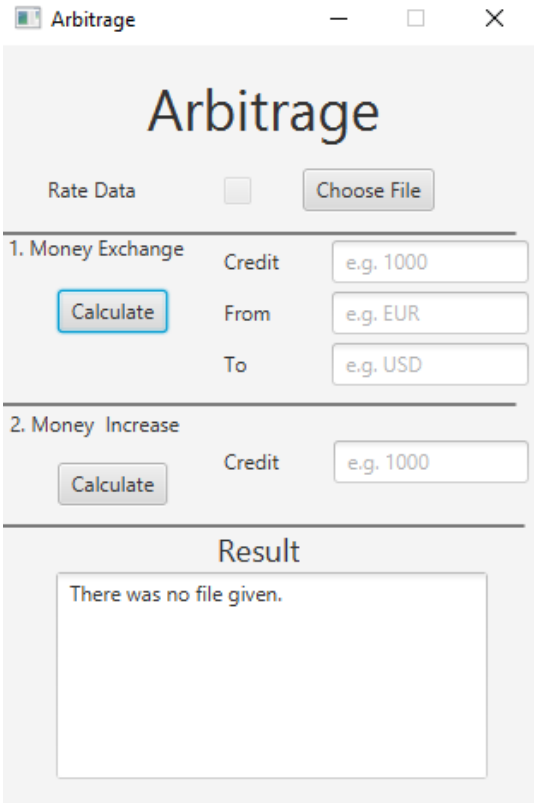
7. Znalezienie wymiany.



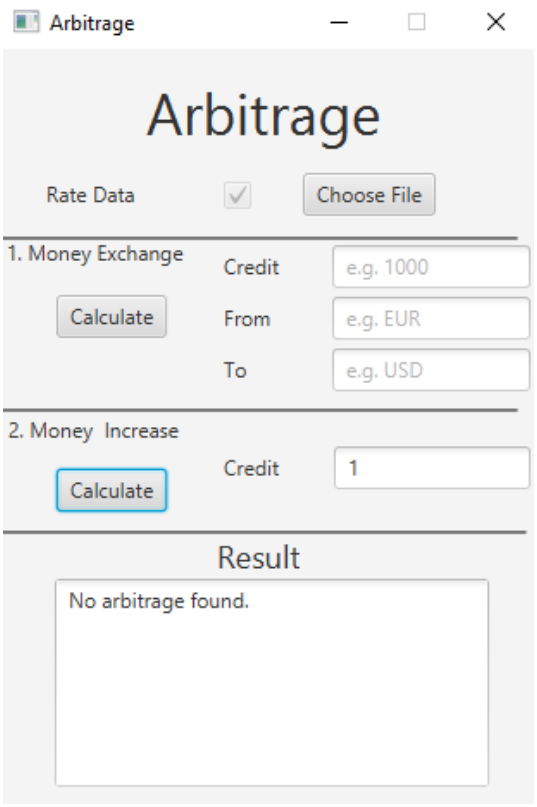
8. Podano za mało danych.



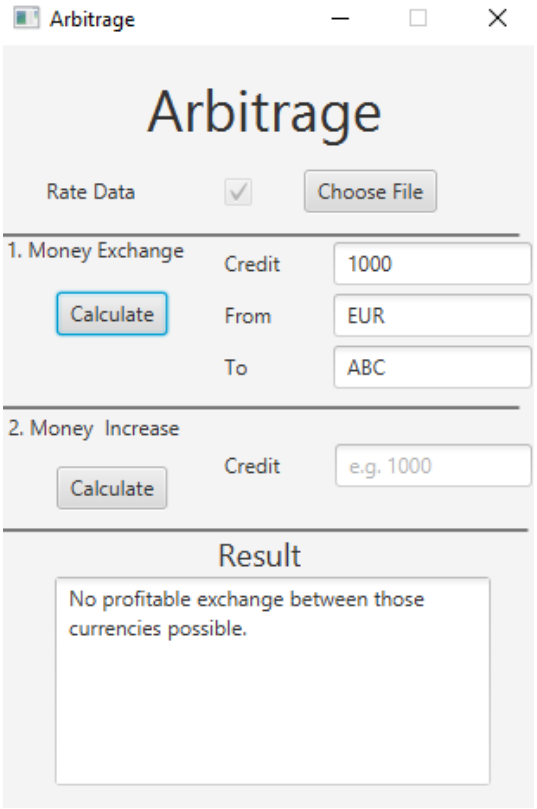
9. Nie podano pliku.



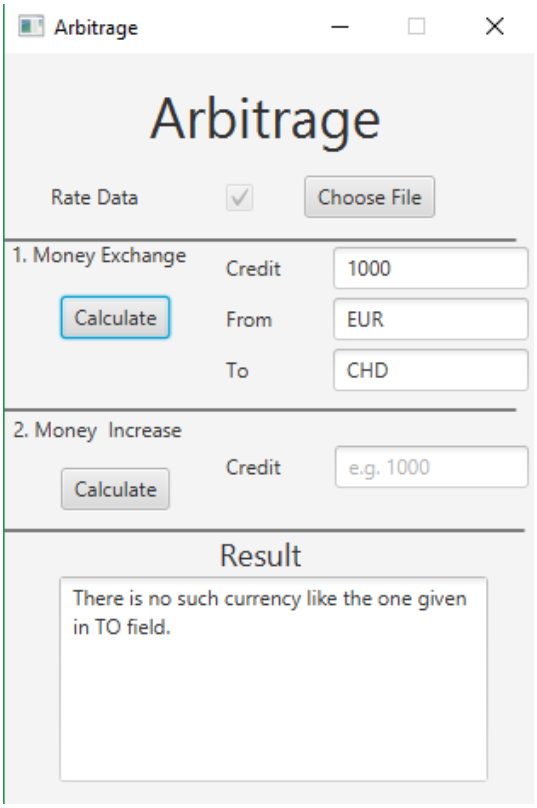
10. Nie znaleziono arbitrażu.



11. Nie znaleziono wymiany.



12. Nie ma takiej waluty.



5 Zmiany względem specyfikacji

1. Zmiana nazwy metody:
   (a) *calculateProfitPath* na *calculateArbitrage*,
   (b) *calculateExchangePath* na *calculateExchange*,
   (c) *checkOfferLine* na *checkRateLine*.

   . Zmiany te poprawiają czytelność kodu.

2. Zamiast jednej klasy *GUI* powstał pakiet dwóch klas: *Controller* i *Main*.

   Poprawia to czytelność kodu i wyodrębnia do klasy *Controller* obsługę okna aplikacji.

3. Funkcjonalności klas *FileChecker* i *FileProcessor* zostały połączone w jedną klasę *FileProcessor*.

   Zmiana ta pozwala uniknąć podwójnego czytania pliku. W trakcie jednego czytania algorytm sprawdza jego poprawność oraz przetwarza zawarte w nim dane na graf.

4. Do obliczeń najkorzystniejszej ścieżki wymiany została dołączona także kwota początkowa.

5. Możliwa kwota wejściowa została ograniczona do prawie 10 miliardów (9 999 999 999,99).

   Jest to arbitralnie przyjęta *wystarczająco duża kwota* nie mniejsza od zarobków przeciętnego obywatela i pozwalająca większości użytkowników swobodnie korzystać z funkcji programu.

6. Do *FileProcessor* dodano pole *repeatedCurrency*.

   Ułatwia to przekazywanie użytkownikowi informacji o powtórzonej deklaracji tej samej waluty.

7. Do klasy *Currency* dodano pole *fullName* reprezentujące pełną nazwę waluty.

   Pole to nie jest obecnie wykorzystywane, pozostaje w kodzie, ponieważ w przyszłości może okazać się przydatne do rozbudowy programu. Jest ono uzupełniane prawidłowymi wartościami z pliku wejściowego.

8. Została wydzielona klasa *Graph*.

   Powstała, aby uporządkować metody do operacji na mapie walut (*currencies*), co poprawia czytelność kodu oraz łatwość korzystania z tych metod.

9. Do klasy *Currency* dodano listę odwiedzonych walut.

   Pozwala to na równoczesne znajdywanie ścieżki wymiany oraz kwoty docelowej.

10. Dodano klase *LoopIterator*.

    Klasa ta poprawia czytelność iteracji po węzłach (walutach) w sposób zgodny z algorytmem Bellmana-Forda (kiedy iterator dochodzi do końca tablicy wskazuje na jej zerowy element, co pozwala iterować po tablicy *w obiegu zamkniętym*).

11. Z uwagi na dodanie listy odwiedzonych węzłów do klasy *Currency*, rola klasy *Path* została zredukowana do nośnika informacji o ścieżce i kwocie końcowej.

12. W trakcie implementacji klasa *FinancialAnalyst* okazała się niepotrzebnie wydzielona i została usunięta, ponieważ wszystkie jej domniemane funkcje spełnia klasa *Broker*.

13. Zrezygnowano z obiektów *CheckBox* w sekcji 1 oraz 2 okna aplikacji.

    Obiekty te miały wskazywać, czy podano wartości do pól *Credit*, *To*, *From*. Okazało się to niepotrzebne, ponieważ użytkownik jest w stanie zorientować się, czy pole zostało wypełnione, po samej zawartości pola.

14. Po poprawnym wczytaniu pliku program wyświetla informacje o walutach i ich możliwych wymianach.

    Opcja ta została dodana, aby użytkownik mógł w łatwiejszy sposób sprawdzić jakie wymiany między walutami są dostępne.

15. Program dostosowany jest do czytania plików o kodowaniu ANSI.

16. Zostało przyjęte, że nazwy skrócone muszą być pisane wielkimi literami i składać się z 3 liter, dlatego, że jest to ogólnie przyjętym standardem.

6 Podsumowanie i wnioski

Na implementacje algorytmu, wyszukującego arbitraż lub najkorzystniejsze wymiany walut miałem kilka pomysłów. Najbardziej oczywistym rozwiązaniem wydawało się zastosowanie algorytmu BFS albo DFS w celu odnalezienia wszystkich możliwych dróg w grafie między zadanymi walutami, a następnie wybranie tej najkorzystniejszej. Przed zastosowaniem jednego z tych algorytmów powstrzymywał mnie fakt, że znajdywanie wszystkich możliwych ścieżek w grafie wydawało mi się skrajnie nieefektywne, czasochłonne i niepotrzebne.

Zdecydowałem się na algorytm Bellmana-Forda, ponieważ bardzo spodobał mi się fakt, że po jednym jego wykonaniu wszystkie węzły zawierają informacje o koszcie dotarcia do nich. Dzięki temu użytkownik po ustaleniu waluty początkowej, może następnie dowolną ilość razy zmieniać walutę docelową, a algorytm i tak wykona operacje poszukiwania (ustawiania kosztów w grafie) tylko jeden raz. Przy każdym kolejnym żądaniu (dla tej samej waluty początkowej), będzie tylko pobierał z węzłów (walut) określone już ścieżki i kwoty. Decyzja ta oparta była również na moim osobistym przekonaniu, że przeciętny użytkownik jest w posiadaniu mniejszej ilości walut niż ilość tych walut, na które potencjalnie chciałby wymienić swoje pieniądze, czyli częściej będzie zmieniał walutę docelową niż wejściową.

W swojej implementacji tego algorytmu dokonałem zmiany polegającej na tym, że poszukiwany jest największy koszt dotarcia do węzła po ścieżce bez powtórzeń.

Algorytm prawidłowo wyznaczył ścieżki, we wszystkich przypadkach układu danych jakie testowałem. Poradził sobie zarówno w przypadku zapętlonych wymian jak i braku połączenia między walutami. Prawidłowo wyznaczył ścieżki w przypadku zróżnicowanych kursów i opłat jak i dla przypadku stałych opłat oraz kursów. Podołał w przypadku gęstych połączeń między węzłami jak i rzadkich. Arbitraż znalazł za każdym razem, gdy

było to możliwe. Na podstawie przeprowadzonych przeze mnie testów działania algorytmu stwierdzam, że program działa prawidłowo.

Wnioski:

- algorytm Bellmana-Forda okazał się bardzo przydatny w realizacji tego zadania, pozwala zaoszczędzić wielu operacji;
- ograniczenie nazw skróconych walut, nie pozwoli użytkownikowi na odejście od ogólnie przyjętych standardów oraz program przestanie być aktualny, jeśli standardy te ulegną zmianom;
- algorytm jest uzależniony od osobliwego wzoru pliku, wartościową opcją rozwoju byłoby dodanie mu możliwość czytania plików w popularnym formacie jak np. JSON;
- jednoczesne czytanie pliku oraz tworzenie grafu pozwala zaoszczędzić czas pracy programu, jednak utrudnia wymianę części programu odpowiedzialnej za tworzenie grafu bądź tej zajmującej się walidacją pliku.