Specyfikacja implementacyjna projektu w języku Java

Kamil Fryszkowski, Oskar Biwejnis

11 maja 2022

1 Wstęp

Celem projektu jest stworzenie programu w języku Java (wersja 1.8), posiadającego graficzny interfejs użytkownika, który będzie w stanie znajdować najkrótszą ścieżkę w pomiędzy dwoma wierzchołkami w danym grafie ważonym skierowanym. Użyte zostaną do tego algorytmy BFS oraz Dijkstry. Środowiskiem programistycznym będzie maszyna wirtualna Javy, kod będzie pisany przy użyciu IntelliJ Idea (wersja 2022.1) wraz z pomocą narzędzia automatyzującego budowę oprogramowania Maven (wersja 3.8.5) i technologą JavaFX pozwalającą na wygodną pracę z graficznym interfejsem użytkownika. Wykorzystanym sposobem współpracy i wersjonowania jest system kontroli wersji GIT w domenie `projektor.ee.pw.edu.pl`.

2 Struktury danych

Do poprawnego funkcjonowania programu niezbędne będzie użycie następujących struktur danych:

1. Do przechowywania grafu użyta zostanie struktura danych `Graph` zaimplementowana w bibliotekach Javy.
2. Kolejka FIFO użyta zostanie przy działaniu algorytmu BFS. W kolejce tej, pierwszy wprowadzony element będzie pierwszym wyjętym. Aplikacja korzystać będzie z kolejki zaimplementowanej w bibliotekach Javy.
3. Kolejka priorytetowa niezbędna będzie do poprawenego działania algorytmu Dijkstry. Priorytem według które będą wyjmowane elementy z kolejki będzie wartość `d[v]`, mówiąca o długości najkrótszej ścieżki do danego wierzchołka z wierzchołka startowego w danym momencie działania algorytmu. Aplikacja korzystać będzie z kolejki zaimplementowanej w bibliotekach Javy.

3 Tryby pracy programu

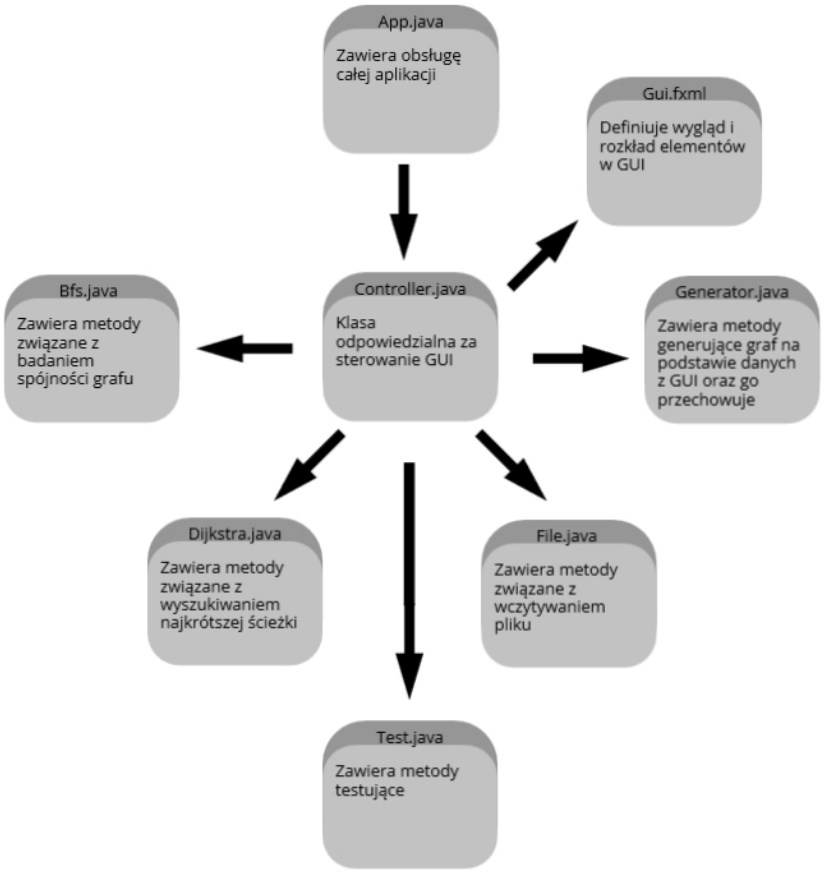
Program uruchamia się w jednym i głównym trybie działania, który pozwala na kliknięcie odpowiednich przycisków pozwalających na działanie w określonym zakresie. Dostępne przyciski to:

1. Tryby działania generatora
   - RandWeightMode - pozwala na stworzenie grafu, w którym istnieją wszystkie krawędzie zawierające się w danym przedziale wierzy i kolumn z wygenerowanymi losowo z danego przedziału wagami
   - AllRandMode - pozwala na stworzenie grafu, w którym istnieje jedynie określona szansa na powstanie danej krawędzi
   - ConMode - działa jak AllRandMode, jednak graf generuje się tak długo, aż nie będzie spójny.
2. Wyświetlanie grafu
   - Koloruj wagi - pozwala na graficzne odwzorowanie wag według koloru (najmniejsze wagi - najjaśniejszy kolor)
   - Wyswietl jedynie krawędzie ścieżki - wyłącza wyświetlanie tych krawędzi, przez które nie prowadzi ścieżka
3. Obsługa piliku

- Zapisz do pliku - zapisuje graf do pliku
- Wczytaj z pliku - wczytuje graf z pliku

4. Wyszukiwanie ścieżki i badanie spójności
   - Szukaj - po podaniu dwóch wierzchołków znajduje najkrótszą ścieżkę między nimi
   - Sprawdź spójność - sprawdza czy graf jest spójny

4 Modularna struktura programu



Rysunek 1: Graficzna reprezentacja struktury programu (opracowanie własne)

- App.java:
  - `void main(String[] args)` – odpowiada za całe sterowanie programem
- Generator.java:
  - `Graph graph` - struktura przechowująca graf
  - `void findNeighbours (Graph graph, int vertex, int rows, int cols, double minWeight, double maxWeight)` - odpowiada za szukanie sąsiadów poszczególnych wierzchołków
  - `Graph generateRandWeightMode (int rows, int cols, double minWeight, double maxWeight)` - odpowiada za generowanie grafu typu "kartka w kratkę"
  - `Graph generateAllRandMode (int rows, int cols, double minWeight, double maxWeight)` - odpowiada za generowanie losowego grafu

  - `Graph generateConMode (int rows, int cols, double minWeight, double maxWeight)` - odpowiada za generowanie spójnego grafu

- Bfs.java:
  - `Queue<Integer> queue` – struktura kolejki dla algorytmu BFS
  - `int BFS(Graph graph, int n, int startingVertex)` – odpowiada za działanie algorytmu BFS
  - `String isConst(Graph graph, int rows, int columns)` – odpowiada za sprawdzenie, czy graf jest spójny oraz zwraca odpowiedni komunikat
- Controller.java:
  - `void gui()` – odpowiada za obsługę graficznego interfejsu użytkownika
  - `void printComunicate(String)` – odpowiada za wypisanie podanego tekstu w okienku komunikatów
- Dijkstra.java:
  - `PriorityQueue<Double> pQueue` – struktura kolejki priorytetowej dla algorytmu Dijkstry
  - `LinkedList<Integer> path` – struktura listy liniowej dla najkrótszej ścieżki
  - `String printPath(Graph graph, LinkedList<Integer> path, boolean doPrintWeights)` – zwraca tekst komunikatu zawierającego wypisaną najkrótszą ścieżkę
  - `LinkedList<Integer> dijkstra (Graph graph, int start, int destination, int rows , int cols)` – odpowiada za działanie algorytmu dijkstry, zwraca listę liniową z kolejnymi wierzchołkami w najkrótszej ścieżce
- File.java:
  - `Graph readFile (String filePath)` - odpowiada za czytanie grafu z pliku
  - `void printGraphToFile (Graph graph, int rows, int cols)` - odpowiada za zapisywanie grafu do pliku
- Test.java - zawiera w sobie testy opisane szerzej w punkcie 7.
- Gui.fxml - definiuje wygląd, funkcjonalności i rozmieszczenie elementów w graficznym interfejsie użytkownika

5 Algorytm BFS

Działanie algorytmu BFS, czyli algorytmu przeszukiwania grafu wszerz, można streścić następująco:

Wybieramy wierzchołek startowy. Następnie odwiedzamy po kolei wszystkie wierzchołki sąsiadujące z wierzchołkiem startowym. Następnie odwiedzamy po kolei wszystkie nieodwiedzone wierzchołki sąsiadujące z wierzchołkami sąsiadującymi z wierzchołkiem startowym i tak dalej...
Niezbędna do funkcjonowania BFSa jest kolejka FIFO (patrz pkt 2.) oraz tablica która dla każdego wierzchołka będzie posiadała informację o jego stanie:

0 – nieodwiedzony (kolor biały)
1 – dodany do kolejki lecz nieodwiedzony (kolor szary)
2 - odwiedzony (kolor czarny)

Złożoność pamięciowa
Dla wykorzystanej w tym programie listy sąsiedztwa złożoność pamięciowa opisana jest wzorem O(V+E) gdzie: V - liczba wierzchołków, E - liczba krawędzi.

Złożoność czasowa
Pesymistyczny przypadek to taki, gdy algorytm musi odwiedzić wszystkie wierzchołki oraz krawędzie, wtedy złożoność czasowa wynosi O(V+E), gdzie V - liczba wierzchołków, E - liczba krawędzi.

6 Algorytm Dijkstry

Algorytm dijkstry służy do znajdowania **najkrótszej ścieżki** w grafie ważonym. Przez **ścieżkę** pomiędzy dwoma wierzchołkami rozumiemy kolejno uporządkowane wierzchołki, które należy odwiedzić aby dostać się z wierzchołka startowego do końcowego. Z kolei **najkrótszą** ścieżką jest ta, której suma wag krawędzi jest najniższa. Algorytm Dijkstry, podczas działania, znajduje najkrótsze ścieżki do każdego wierzchołka do którego można dojść z wierzchołka startowego

Niezbędne do poprawnego działania będzie utworzenie:

- Tablicy d[V], gdzie V oznacza liczbę wszystkich wierchołków w grafie. Przechowywać będzie ona informację o długości aktualnej najkrótszej ścieżki z wierzchołka startowego do danego wierzchołka. Dla wierzchołka startowego s, d[s] przyjmuje początkowo wartość 0, a dla każdego innego nieskończoność.
- Tablicy p[V], gdzie V oznacza liczbę wszystkich wierchołków w grafie. Przechowywać będzie ona informację o poprzednim wierzchołku w najkrótszej ścieżce do danego wierzchołka. Początkowo jest pusta.
- Kolejki priorytetowej opartej na strukturze kopca. W kolejce tej początkowo będą wszystkie wierzchołki, a priorytetem wyciągania elementów z niej będzie najniższa wartość d[v] dla danego wierzchołka v.

7 Testowanie

Poprawność działania programu należy sprawdzić kolejno testami, oraz wyniki porównać z wcześniej przygotowanymi danymi:

- `int testAllRandModeGenerator()` - funkcja sprawdzająca poprawność działania generator w trybie AllRandMode
- `int testRandWeightModeGenerator()` - funkcja sprawdzająca poprawność działania generator w trybie RandWeightMode
- `int testConModeGenerator()` - funkcja sprawdzająca poprawność działania generator w trybie ConMode
- `int testBFS()` - funkcja sprawdzająca poprawność działania algorytmu BFS
- `int testDijkstra()` - funkcja sprawdzająca poprawność działania algorytmu Dijkstry
- `int testRead()` - funkcja sprawdzająca poprawność działania wczytywania grafu z pliku
- `int testSave()` - funkcja sprawdzająca poprawność działania zapisywania grafu do pliku

8 Źródła

1. https://pl.wikipedia.org