JavaGridGraph - Sprawozdanie

Skoczek Mateusz, Jędrzejewski Sebastian

16 czerwca 2022

Streszczenie

Dokument zawiera specyfikację funkcjonalną i implementacyjną dotyczącą projektu *JavaGrid-Graph* oraz opis testów programu.

Spis treści

Rozdział 1

Specyfikacja funkcjonalna

1.1 Cel projektu

Program **JavaGridGraph** ma na celu umożliwiać wykonanie dwóch podstawowych zadań

- wygenerowanie grafu siatkowego o podanych paramentrach
- sprawdzenie wybranych parametrów dowolnego grafu siatkowego

Program posiada interfejs graficzny. Grafy są przedstawiane w plikach w postaci listy sąsiedztwa.

1.2 Opis funkcji

Program oferuje dwie główne funkcje: generowanie grafu oraz sprawdzanie grafu.

Program pozwala wygenerować graf o:

- określonej wysokości (ilości wierszy)
- określonej szerokości (ilości kolumn)
- określonej minimalnej i maksymalnej wadze krawędzi
- określonej minimalnej i maksymalnej ilości krawędzi wychodzących z pojedyńczego wierzchołka (dla grafu skierowanego)
- określonej minimalnej i maksymalnej ilości krawędzi wchodzących do pojedyńczego wierzchołka (dla grafu skierowanego)
- określonej minimalnej i maksymalnej ilości "sąsiadów" pojedyńczego wierzchołka (dla grafu nieskierowanego)
- niestandardowym ziarnie generatora liczb losowych

Program umożliwia zapis wygenerowanego grafu do pliku oraz/lub wczytanie grafu do sprawdzenia.

W ramach funkcji sprawdzania grafu, program pozwala na sprawdzenie następujących parametrów wczytanego grafu:

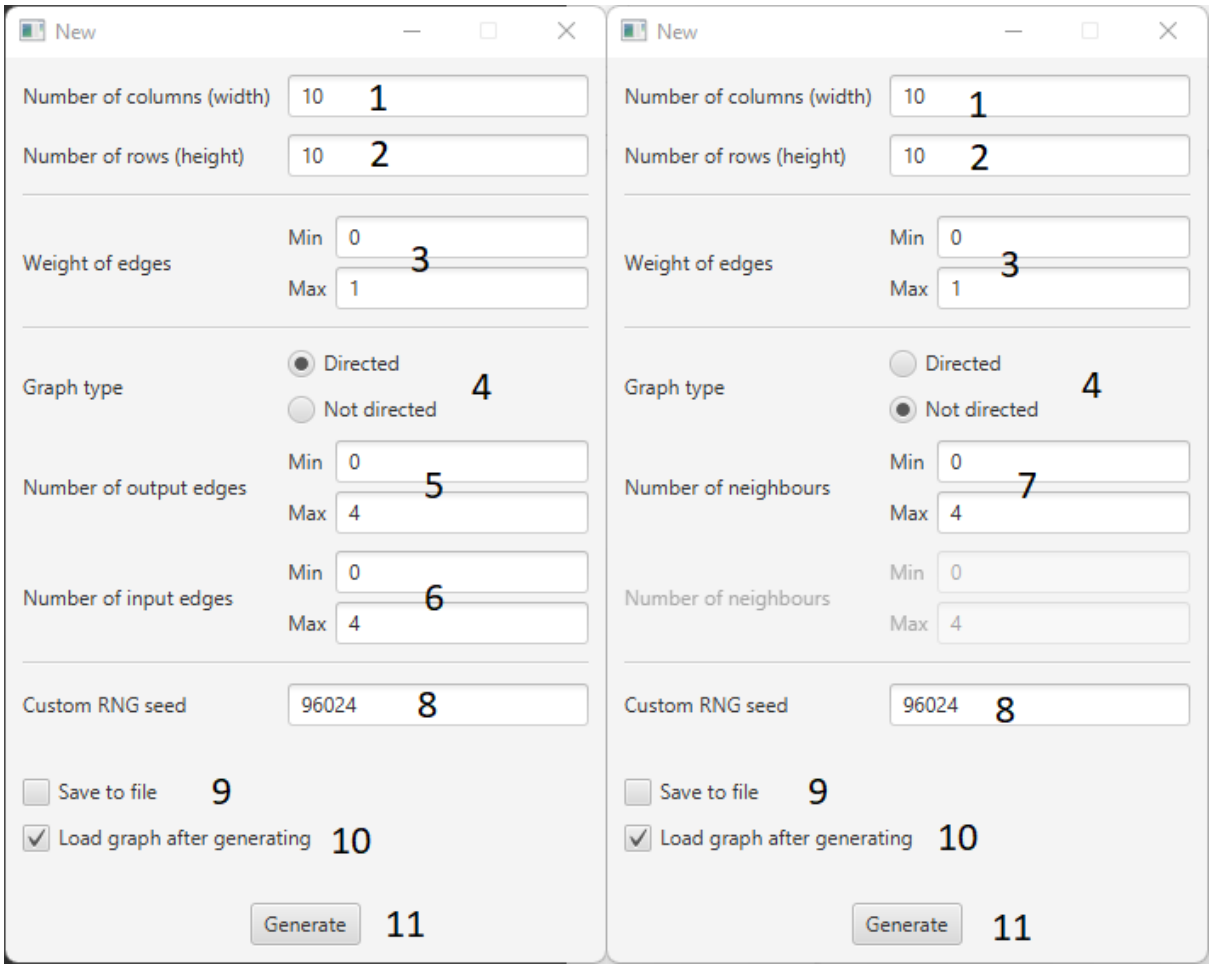
- spójność grafu
- najkrótsze ścieżki od wybranego wierzchołka A do wybranych wierzchołków B_n

1.3 Opis interfejsu programu

1.3.1 Okno główne - widok startowy

1. Wynik sprawdzenia spójności grafu. W przypadku gdy graf nie został wczytany, zostanie pokazana informacja o tym że graf nie został wczytany.
2. Przycisk "Open" otwiera systemowe okno wyboru pliku w celu wybrania pliku zawierającego graf.
3. Przycisk "New" otwiera okno generowania grafu. Więcej informacji w punkcie "Okno generowania grafu".

1.3.2 Okno generowania grafu



1. Liczba kolumn (szerokość) grafu. Pole nie może być puste.
2. Liczba wierszy (wysokość) grafu. Pole nie może być puste.

3. Waga krawędzi. Wartość w pierwszym polu (min, domyślnie 0) musi być mniejsza bądź równa wartości w drugim polu (max, domyślnie 1).

4. Wybór między generowaniem grafu skierowanego (directed), generowaniem grafu nieskierowanego (not directed).

5. Liczba krawędzi wychodzących z pojedyńczego wierzchołka. Wartość w pierwszym polu (min, domyślnie 0) musi być mniejsza bądź równa wartości w drugim polu (max, domyślnie 4). [Tylko gdy zaznaczone generowanie grafu skierowanego][1]

6. Liczba krawędzi wchodzących do pojedyńczego wierzchołka. Wartość w pierwszym polu (min, domyślnie 0) musi być mniejsza bądź równa wartości w drugim polu (max, domyślnie 4). [Tylko gdy zaznaczone generowanie grafu skierowanego][1]

7. Liczba sąsiadów pojedyńczego wierzchołka. Wartość w pierwszym polu (min, domyślnie 0) musi być mniejsza bądź równa wartości w drugim polu (max, domyślnie 4). [Tylko gdy zaznaczone generowanie grafu nieskierowanego][1]

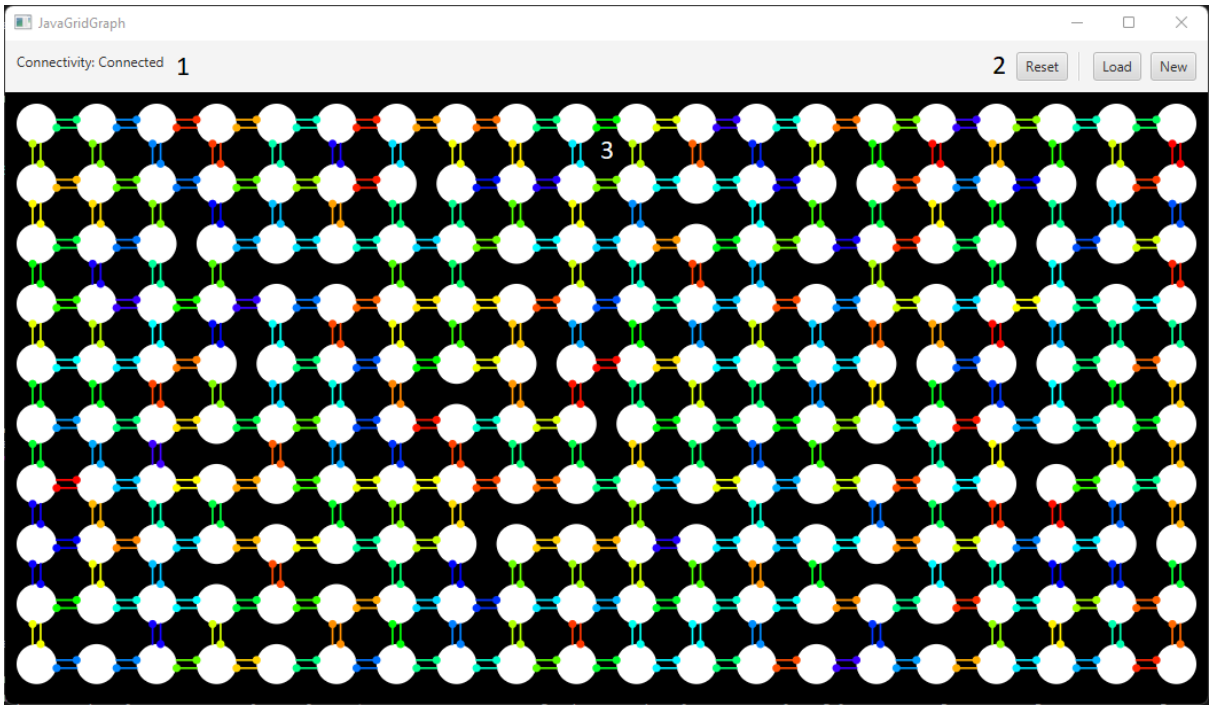
8. Niestandardowe ziarno generatora liczb losowych.

9. Zaznaczenie tego pola spowoduje zapisanie grafu w pliku wybranym w systemowym oknie zapisu pliku, które wyświetli się po naciśnięciu przycisku "Generate".

10. Zaznaczenie tego pola spowoduje wczytanie grafu do programu zaraz po jego wygenerowaniu.

11. Kliknięcie przycisku spowoduje wygenerowanie grafu o podanych parametrach.

1.3.3 Okno główne - po wczytaniu grafu



[1]Program będzie dążył do utworzenia co najmniej minimalnej liczby krawędzi (połączeń do sąsiadów), ale nie może tego zagwarantować. Nie jest możliwe wygenerowanie więcej niż 2 krawędzi dla wierzchołków w narożnikach oraz więcej niż 3 dla wierzchołków bocznych. Nie jest możliwe także utworzenie krawędzi, jeżeli wszystkie wierzchołki wokół osiągnęły już swoją nominalną (wylosowaną z podanego przedziału) liczbę krawędzi.

1. Wynik sprawdzenia spójności grafu. "Connected" - spójny, "Unconnected" - niespójny.
2. Kliknięcie przycisku spowoduje odznaczenie wszystkich zaznaczonych wierzchołków na grafie.
3. Pole w którym wyświetlany jest graf. Aby znaleźć najkrótsze ścieżki musimy wybrać wierzchołek A lewym przyciskiem myszy (zostanie zaznaczony na czerwono) oraz wierzchołki B_n prawym przyciskiem myszy (zostaną zaznaczone na zielono). Po wybraniu wierzchołka A oraz przynajmniej jednego wierzchołka B zostanie narysowana najkrótsza ścieżka od wierzchołka A do wierzchołka B_n.

1.4 Format danych wejściowych i wyjściowych

Dane wejściowe i wyjściowe przechowują graf w postaci listy sąsiedztwa. W pierwszej linijce znajdują się dwie liczby, które oznaczają odpowiednio liczbę kolumn i wierszy danego grafu. Każda następna linijka reprezentuje jeden wierzchołek, przy czym wierzchołki numerujemy od 0 od lewej do prawej. Zatem druga linijka w pliku zawiera numery wierzchołków, z którymi połączony jest wierzchołek numer 0, kolejna dotyczy wierzchołka numer 1 itd. Przy każdym numerze wierzchołka po dwukropku podana jest waga krawędzi pomiędzy tymi dwoma wierzchołkami.

Przykład:

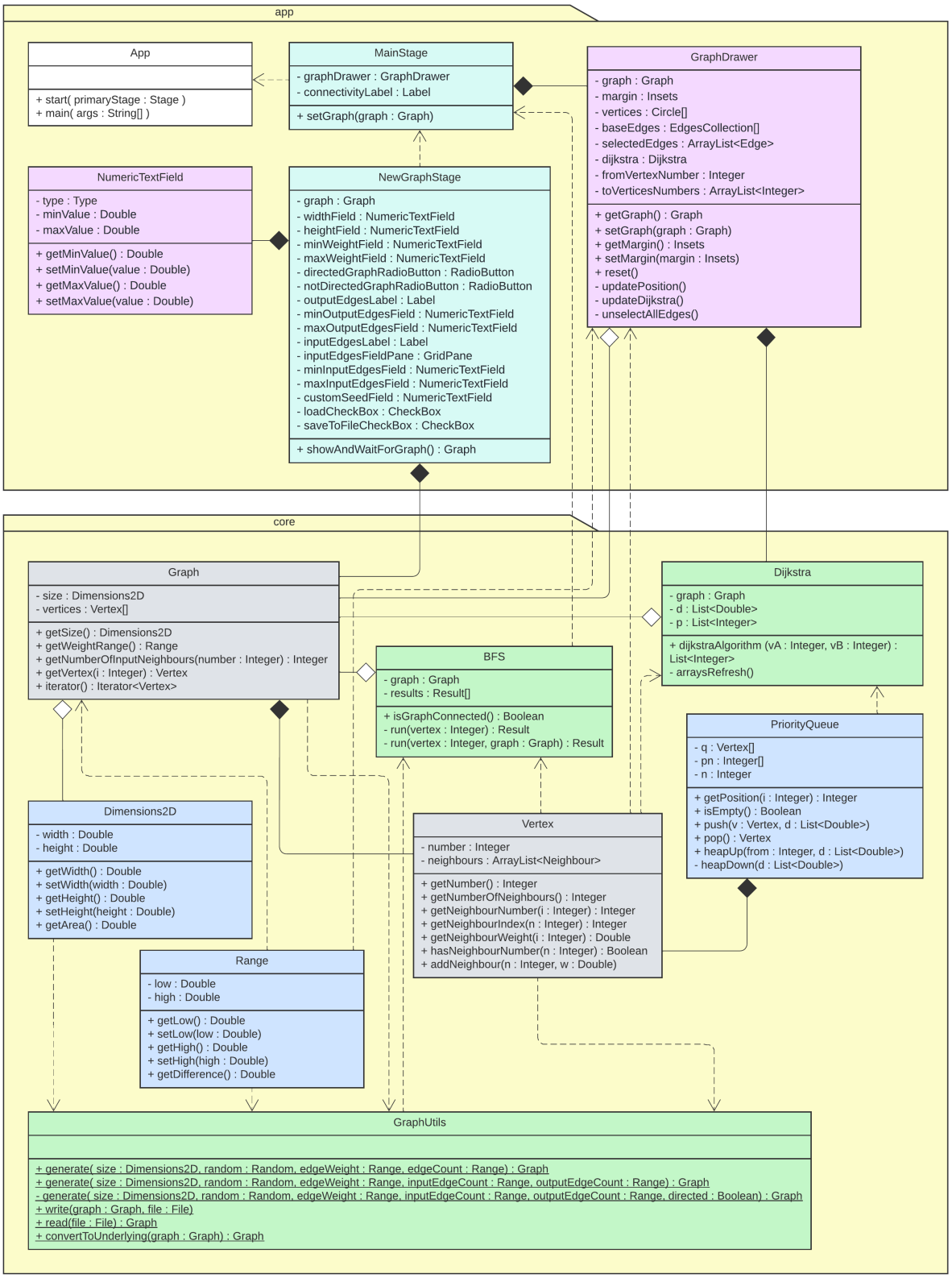
```
2 2
    1 :0.54 2 :0.78
    0 :0.54 3 :0.12
    0 :0.78 3 :0.89
    1 :0.12 2 :0.89
```

Powyżej przedstawiona jest przykładowa zawartość pliku przechowującego graf. W pierwszej linijce można odczytać, że jest to graf o dwóch kolumnach i dwóch wierszach. W drugiej linijce przedstawiona jest informacja o tym, że wierzchołek numer 0 połączony jest z wierzchołkiem numer 1, a krawędź ta ma wagę 0.54. Istnieje również krawędź pomiędzy wierzchołkiem 0 a 2 o wadze 0.78. W trzeciej linijce znajdują się numery wierzchołków połączonych z wierzchołkiem numer 1 wraz z wagami itd.

Rozdział 2

Specyfikacja implementacyjna

2.1 Diagram klas



2.2 Pakiet App (Interfejs)

Interfejs aplikacji został stworzony przy wykorzystaniu biblioteki JavaFX. Pakiet **App** składa się z dwóch podpakietów:

- **Stages** zawierającym widoki aplikacji
- **Controls** zawierającym niestandardowe kontrolki

2.2.1 App (App)

Klasa App jest główną klasą całej aplikacji. Dziedziczy ona po klasie `Application` należącej do pakietu `javafx.application`

Publiczne metody:

- `public void start(Stage primaryStage)` - Odpowiada za utworzenie głównego okna aplikacji (`MainStage`) i pokazanie go.
- `public static void main(String[] args)` - Jest entry pointem i odpowiada za zainicjowanie aplikacji.

2.2.2 MainStage (App.Stages)

Klasa MainStage jest odpowiedzialna za główne okno aplikacji. Dziedziczy ona po klasie `Stage` należącej do pakietu `javafx.stage`

Właściwości:

- `private final GraphDrawer graphDrawer`
- `private final Label connectivityLabel`

Konstruktory:

- `public MainStage()` - Główny konstruktor. W konstruktorze inicjowane jest okno (tworzone są kontrolki) i ustawiane są właściwości okna.

Publiczne metody:

- `public void setGraph(Graph graph)` - Metoda odpowiedzialna za ustawienie grafu w kontrolce `graphDrawer` oraz sprawdzenie i wyświetlenie infromacji (w kontrolce `connectivityLabel`) o spójności grafu.

Obsługa zdarzeń:

- `LoadGraphButtonClicked` - Obsługuje zdarzenie naciśnięcia przycisku ładowania grafu z pliku. Otwiera systemowe okno otwierania pliku. Po wybraniu pliku wywoływana jest metoda `setGraph`.
- `ResetGraphDrawerButtonClicked` - Obsługuje zdarzenie naciśnięcia przycisku resetowania kontrolki wyświetlającej graf. Czyści zaznaczone ścieżki i wierzchołki (wywołuje metodę `graphDrawer.reset()`).
- `NewGraphButtonClicked` - Obsługuje zdarzenie naciśnięcia przycisku tworzenia nowego grafu. Tworzy okno tworzenia grafu `NewGraphStage`, wyświetla je oraz wywołuje metodę `setGraph` dla zwróconego grafu.

2.2.3 NewGraphStage (App.Stages)

Klasa NewGraphStage jest odpowiedzialna za okno tworzenia grafu. Dziedziczy ona po klasie `Stage` należącej do pakietu `javafx.stage`

Właściwości:

- `private Graph graph` - Graf który zostanie zwrócony w przypadku zamknięcia okna.
- `private final NumericTextField widthField`
- `private final NumericTextField heightField`
- `private final NumericTextField minWeightField`
- `private final NumericTextField maxWeightField`
- `private final RadioButton directedGraphRadioButton`
- `private final RadioButton notDirectedGraphRadioButton`
- `private final Label outputEdgesLabel`
- `private final NumericTextField minOutputEdgesField`
- `private final NumericTextField maxOutputEdgesField`
- `private final Label inputEdgesLabel`
- `private final GridPane inputEdgesFieldPane`
- `private final NumericTextField minInputEdgesField`
- `private final NumericTextField maxInputEdgesField`
- `private final NumericTextField customSeedField`
- `private final CheckBox loadCheckBox`
- `private final CheckBox saveToFileCheckBox`

Konstruktory:

- `public NewGraphStage()` - Główny konstruktor. W konstruktorze inicjowane jest okno (tworzone są kontrolki) i ustawiane są właściwości okna.

Publiczne metody:

- `public Graph showAndWaitForGraph()` - Metoda odpowiedzialna za wyświetlenie okna, a po jego zamknięciu - zwrócenie grafu.

Obsługa zdarzeń:

- `GenerateButtonClicked` - Obsługuje zdarzenie naciśnięcia przycisku generowania grafu. Sprawdza podane parametry grafu, a następnie generuje graf. W przypadku gdy została wybrana opcja zapisu grafu w pliku (`saveToFileCheckBox`), otwiera systemowe okno zapisu pliku i zapisuje w nim wygenerowany graf. Na koniec zamyka okno.
- `DirectedGraphRadioButtonToggleGroupToggleChanged` - Obsługuje zdarzenie zmiany zaznaczenia przycisków (RadioButton) "Directed" (`directedGraphRadioButton`) i "Not directed" (`notDirectedGraphRadioButton`).

2.2.4 GraphDrawer (App.Controls)

Klasa GraphDrawer jest odpowiedzialna za kontrolkę wyświetlającą graf. Dziedziczy ona po klasie `AnchorPane` należącej do pakietu `javafx.scene.layout`

Właściwości:

- `private Graph graph` - Aktualnie ustawiony w kontrolce graf
- `private Insets margin` - Margines kontrolki
- `private Circle[] vertices` - Tablica zawierająca reprezentację wierzchołków
- `private EdgesCollection[] baseEdges` - Tablica zawierająca kolekcję krawędzi wychodzących z wierzchołka o indeksie odpowiadającym indeksowi w tablicy
- `private ArrayList<Edge> selectedEdges` - Lista zawierająca zaznaczone krawędzie.
- `private Dijkstra dijkstra` - Algorytm Dijkstra dla aktualnie ustawionego grafu.
- `private int fromVertexNumber` - Aktualnie zaznaczony wierzchołek "od" (zaznaczony na grafie kolorem czerwonym)
- `private ArrayList<Integer> toVerticesNumbers` - Lista aktualnie zaznaczonych wierzchołków "do" (zaznaczone na grafie kolorem zielonym)

Konstruktory:

- `public GraphDrawer()` - Główny konstruktor. W konstruktorze ustawiane są domyślne własciwości kontrolki.

Publiczne metody:

- `public Graph getGraph()` - Metoda zwraca aktualnie ustawiony graf
- `public void setGraph(Graph graph)` - Metoda ustawia podany graf w kontrolce.
- `public Insets getMargin()` - Metoda zwraca margines kontrolki
- `public void setMargin(Insets margin)` - Metoda ustawia podany margine kontrolki.
- `public void reset()` - Metoda czyści zaznaczenia wierzchołków i ścieżek

Prywatne metody:

- `public void updatePosition()` - Metoda uaktualnia pozycje wierzchołków i krawędzi.
- `public void updateDijkstra()` - Metoda uaktualnia zaznaczone krawędzie
- `public void unselectAllEdges()` - Metoda usuwa zaznaczenia ze wszystkich krawędzi.

Klasy zagnieżdzone:

- `private class EdgesCollection` - Przechowuje reprezentacje krawędzi wychodzących z pojedyńćzego wierzchołka we wszystkich kierunkach.
- `private class Edge extends Group` - Zespół kontrolek Circle i Line, będących reprezentacją krawędzi. Dziedziczy po klasie `Group` należącej do pakietu `javafx.scene`

2.2.5 NumericTextField (App.Controls)

Klasa NumericTextField jest modyfikacją kontrolki `TextField` pozwalającą na wpisywanie tylko liczb. Dziedziczy ona po klasie `TextField` należącej do pakietu `javafx.scene.control`

Właściwości:

- `private final Type type` - Typ liczb wpisywanych w pole.
- `private double minValue` - Dolny zakres liczb wpisywanych w pole.
- `private double maxValue` - Górny zakres liczb wpisywanych w pole.

Konstruktory:

- `public NumericTextField(Type type)` - Konstruktor wywołujący główny konstruktor. Jako wartość początkową w polu ustawia pusty napis.
- `public NumericTextField(Type type, String text)` - Główny konstruktor.

Publiczne metody:

- `public double getMinValue()` - Metoda zwraca dolny zakres liczb wpisywanych w pole
- `public void setMinValue(double value)` - Metoda ustawia dolny zakres liczb wpisywanych w pole
- `public double getMaxValue()` - Metoda zwraca górny zakres liczb wpisywanych w pole
- `public void setMaxValue(double value)` - Metoda ustawia górny zakres liczb wpisywanych w pole

Obsługa zdarzeń:

- `TextChanged` - Obsługuje zdarzenie zmiany tekstu w polu. Sprawdza wprowadzoną liczbę pod kątem ustawionych własciwości kontrolki.

Wyliczenia:

- `Type` - Typ wprowadzanych w kontrolce liczb

2.3 Pakiet Core

Pakiet **Core** składa się z dwóch podpakietów:

- **GraphAlgorithms** zawierającym algorytmy grafowe
- **Helpers** zawierającym klasy pomocnicze.

2.3.1 Graph (Core)

Klasa Graph jest odpowiedzialna za przechowywanie grafu siatkowego w formie listy sąsiedztwa. Implementuje ona interfejs `Iterable`.

Właściwości:

- `private final Dimensions2D size` - Rozmiar grafu.
- `private final Vertex[] vertices` - Tablica wierzchołków.

Konstruktory:

- `public Graph(Dimensions2D size)` - Główny konstruktor.

Publiczne metody:

- `public Dimensions2D getSize()` - Metoda zwraca rozmiar grafu
- `public Range getWeightRange()` - Metoda zwraca dolną i górną granicę (najmniejszą i największą wagę) wszystkich wag krawędzi.
- `public int getNumberOfInputNeighbours(int number)` - Metoda zwraca liczbę krawędzi wchodzących do danego wierzchołka
- `public int getVertex(int i)` - Metoda zwraca wierzchołek o podanym indeksie

2.3.2 Vertex (Core)

Klasa Vertex jest odpowiedzialna za informacji o wierzchołku grafu.

Właściwości:

- `private final int number` - Numer wierzchołka.
- `private final ArrayList<Neighbour> neighbours` - Lista krawędzi wychodzących z wierzchołka (sąsiadów)

Konstruktory:

- `public Vertex(int n)` - Główny konstruktor.

Publiczne metody:

- `public int getNumber()` - Metoda zwraca numer wierzchołka
- `public int getNumberOfNeighbours()` - Metoda zwraca ilość krawędzi wychodzących z wierzchołka (sąsiadów)
- `public int getNeighbourNumber(int i)` - Metoda zwraca numer sąsiada w grafie, na podstawie jego indeksu w tablicy sąsiadów
- `public int getNeighbourIndex(int n)` - Metoda zwraca index sąsiada w tablicy sąsiadów, na podstawie jego numeru w grafie
- `public double getNeighbourWeight(int i)` - Metoda zwraca wagę krawędzi połączonej z sąsiadem o podanym indeksie w tablicy sąsiadów
- `public boolean hasNeighbourNumber(int n)` - Metoda zwraca wartość prawda/fałsz zapytania czy w tablcy sąsiadów znajduje się wierzchołek o podanym numerze w grafie
- `public void addNeighbour(int n, double w)` - Metoda dodaje nowego sąsiada do tablicy sąsiadów

Klasy zagnieżdzone:

- `static class Neighbour` - Przechowuje informacje (numer sąsiada i wagę) o krawędzi

2.3.3 GraphUtils (Core.GraphAlgorithms)

Klasa GraphUtils przechowuje metody pomocnicze dla grafów

Metody:

- `public static Graph generate(Dimensions2D size, Random random, Range edgeWeight, Range edgeCount)` - Metoda wywołuje główną metodę `generate` z opcją generowania grafu nieskierowanego
- `public static Graph generate(Dimensions2D size, Random random, Range edgeWeight, Range inputEdgeCount, Range outputEdgeCount)` - Metoda wywołuje główną metodę `generate` z opcją generowania grafu skierowanego
- `private static Graph generate(Dimensions2D size, Random random, Range edgeWeight, Range inputEdgeCount, Range outputEdgeCount, boolean directed)` - Metoda generuje graf o podanych parametrach

- `public static void write(Graph graph, File file)` - Metoda zapisuje graf w podanym pliku
- `public static Graph read(File file)` - Metoda czyta graf z pliku
- `public static Graph convertToUnderlying(Graph graph)` - Metoda generuje graf podstawowy podanego grafu.

2.3.4 BFS (Core.GraphAlgorithms)

Klasa BFS jest odpowiedzialna za zarządzanie algorytmem BFS dla podanego grafu oraz za zwracanie własciwości grafu na podstawie algorytmu BFS.

Właściwości:

- `private final Graph graph` - Graf.
- `private final Result[] results` - Wyniki działania algorytmu dla każdego wierzchołka grafu

Konstruktory:

- `public BFS(Graph graph)` - Główny konstruktor.

Publiczne metody:

- `public boolean isGraphConnected()` - Metoda sprawdza czy graf jest spójny.

Prywatne metody:

- `private Result run(int vertex)` - Metoda wywołuje algorytm BFS dla domyślnego grafu i podanego wierzchołka
- `private Result run(int vertex, Graph graph)` - Metoda wywołuje algorytm BFS dla podanego grafu i podanego wierzchołka.

Klasy zagnieżdzone:

- `private class Result` - Przechowuje wynik działąnia algorytmu BFS

2.3.5 Dijkstra (Core.GraphAlgorithms)

Klasa Dijkstra jest odpowiedzialna za zarządzanie algorytmem Dijkstry dla podanego grafu.

Właściwości:

- `private final Graph graph` - Graf.
- `private final List<Double> d`
- `private final List<Integer> p`

Konstruktory:

- `public Dijkstra(Graph graph)` - Główny konstruktor.

Publiczne metody:

- `dijkstraAlgorithm(int vA, int vB)` - Metoda wywołuje algorytm Dijkstry dla pary wierzchołków

Prywatne metody:

- `arraysRefresh()` - Metoda resetuje zawartość list

2.3.6 Dimensions2D (Core.Helpers)

Klasa Dimensions2D jest odpowiedzialna za przechowywanie informacji o rozmiarze dwuwymiarowego prostokątnego elementu (w aplikacji używany głównie do przechowywania informacji o rozmiarze grafu).

Właściwości:

- `private double width` - Szerokość elementu
- `private double height` - Wysokość elementu

Konstruktory:

- `public Dimensions2D(double width, double height)` - Główny konstruktor.

Publiczne metody:

- `public double getWidth()` - Metoda zwraca szerokość elementu

- `public void setWidth(double width)` - Metoda ustawia szerokość elementu
- `public double getHeight()` - Metoda zwraca wysokość elementu
- `public void setHeight(double height)` - Metoda ustawia wysokość elementu
- `public double getArea()` - Metoda zwraca pole powierzchni elementu

2.3.7 Range (Core.Helpers)

Klasa Range jest odpowiedzialna za przechowywanie informacji o zakresie wartości

Właściwości:

- `private double low` - Dolny zakres
- `private double high` - Górny zakres

Konstruktory:

- `public Range(double low, double high)` - Główny konstruktor.

Publiczne metody:

- `public double getLow()` - Metoda zwraca dolny zakres
- `public void setLow(double low)` - Metoda ustawia dolny zakres
- `public double getHigh()` - Metoda zwraca górny zakres
- `public void setHigh(double high)` - Metoda ustawia górny zakres
- `public double getDifference()` - Metoda zwraca różnicę między wartością maksymalną, a minimalną

2.3.8 PriorityQueue (Core.Helpers)

Klasa PriorityQueue jest implementacją kolejki priorytetowej dla wierzchołków (`Vertex`)

Właściwości:

- `private final Vertex[] q` - Tablica wierzchołków
- `private final int [] pn` - Pozycja wierzchołków w tablicy
- `private int n` - Liczba wierzchołków w tablicy

Konstruktory:

- `public PriorityQueue(int n)` - Główny konstruktor.

Publiczne metody:

- `public int getPosition(int i)` - Metoda zwraca pozycję wierzchołka w tablicy
- `public boolean isEmpty()` - Metoda sprawdza czy tablica jest pusta (true/false)
- `public void push(Vertex v, List<Double> d)` - Metoda dodaje element do kolejki
- `public Vertex pop(List<Double> d)` - Metoda wyciąga element z kolejki
- `public void heapUp(int from, List<Double> d)` - Przesiewanie w górę

Prywatne metody:

- `private void heapDown(List<Double> d)` - Przesiewanie w dół

Rozdział 3

Testy

3.1 Testy klasy GraphUtils

Klasa `GraphUtils` została przetestowana pod kątem poprawności generowania oraz czytania grafu.

3.1.1 Generowanie dwóch grafów 500x500 ze stałym ziarnem generatora liczb losowych

Test polega na wygenerowaniu dwóch grafów nieskierowanych o rozmiarach 500 na 500, z ziarnem generatora równym 0 i z domyślnymi parametrami oraz zapisaniu ich odpowiednio do plików „test1.txt" i „test2.txt". Następnie sprawdzane jest czy oba pliki mają tę samą zawartość, tzn. 250002 linii każdy. W pierwszej linii powinny być zapisane rozmiary grafu, 250000 następnych zawiera listy sąsiedztwa, a ostatnia jest pusta. Na końcu sprawdzana jest też zgodność tych plików z oczekiwanym grafem zapisanym w pliku „expectedTest1.txt". Wyniki testów znajdują się w folderze `testResults`.

3.1.2 Czytanie wygenerowanego grafu

Test polega na czytaniu grafu wygenerowanego w poprzednim teście, który znajduje się w pliku „test1.txt" w folderze `testResults`. Następnie przeczytany graf jest zapisywany do nowego pliku o nazwie „test3.txt". Powinien on być taki sam jak dwa poprzednio wygenerowane grafy i ten oczekiwany. Na końcu porównywana jest zawartość nowopowstałego pliku z oczekiwanym.

3.2 Testy klasy BFS

W klasie `BFS` przetestowana została poprawność algorytmu BFS sprawdzającego spójność zarówno grafów skierowanych, jak i nieskierowanych.

3.2.1 Generowanie i sprawdzenie spójności grafu nieskierowanego na pewno niespójnego (granica liczby krawędzi od 0 do 1)

Test polega na wygenerowaniu grafu nieskierowanego, który na pewno jest niespójny oraz sprawdzeniu jego spójności algorytmem BFS. W tym teście metoda `isGraphConnected` obiektu klasy `BFS` powinna zwrócić `false`, aby test zakończył się pomyślnie.

3.2.2 Generowanie i sprawdzenie spójności grafu nieskierowanego na pewno spójnego (granica liczby krawędzi od 4 do 4)

Test polega na wygenerowaniu grafu nieskierowanego, który na pewno jest spójny oraz sprawdzeniu jego spójności algorytmem BFS. W tym teście metoda `isGraphConnected` obiektu klasy `BFS` powinna zwrócić `true`, aby test zakończył się pomyślnie.

3.2.3 Generowanie i sprawdzenie spójności grafu skierowanego na pewno niespójnego (granica liczby krawędzi wchodzących i wychodzących od 0 do 1)

Test polega na wygenerowaniu grafu skierowanego, który na pewno jest niespójny oraz sprawdzeniu jego spójności algorytmem BFS, który najpierw przekształci ten graf na podstawowy. W tym teście metoda `isGraphConnected` obiektu klasy `BFS` powinna zwrócić `false`, aby test zakończył się pomyślnie.

3.2.4 Generowanie i sprawdzenie spójności grafu skierowanego na pewno spójnego (granica liczby krawędzi wchodzących i wychodzących od 4 do 4)

Test polega na wygenerowaniu grafu skierowanego, który na pewno jest spójny oraz sprawdzeniu jego spójności algorytmem BFS, który najpierw przekształci ten graf na podstawowy. W tym teście metoda `isGraphConnected` obiektu klasy `BFS` powinna zwrócić `true`, aby test zakończył się pomyślnie.

3.3 Test klasy PriorityQueue

Test klasy PriorityQueue polega na sprawdzeniu poprawności działania kolejki priorytetowej. Sprawdzenie to odbywa się poprzez włożenie do kolejki priorytetowej 1000 wierzchołków o losowej odległości od wierzchołka źródłowego (odległość ta to priorytet), przy czym jest ona losowana od 0 do 1. Powinny one być włożone w ten sposób, żeby na początku kolejki znajdował się wierzchołek o najmniejszej odległości.

Następnie odbywa się wyjmowanie wierzchołków i reorganizacja kolejki tak, aby wierzchołek o największym priorytecie był na jej początku. Sprawdzane jest czy odległość wierzchołka zdjętego z kolejki jest mniejsza od odległości wierzchołka wcześniej zdjętego. Jeżeli dla każdego zdjętego wierzchołka jest to spełnione, to kolejka działa poprawnie i test jest zakończony pomyślnie.

3.4 Wyniki testów

Wszystkie powyższe testy zostały napisane w oparciu o bibliotekę `JUnit4`. Ich uruchomienie powoduje automatyczne sprawdzenie, czy zakończyły się one pomyślnie. Napisane testy udanie przeszły tę próbę: