`Specyfikacja implementacyjna`
Generacja grafu i odnalezienie najkrótszej ścieżki pomiędzy węzłami - `graph` - Java

Ulyana Petrashevich, Inga Maziarz

17.05.2022

Spis treści

Informacje ogólne

Program `graph` został napisany w języku `Java` w środowisku programistycznym `IntelliJ IDEA`. Graficzny interfejs użytkownika został zrealizowany przy użyciu technologii `JavaFX`. Współpraca i wersjonowanie odbywa się w repozytorium zdalnym w serwisie `GitHub`.

Po uruchomieniu programu wyświela się okno o wymiarach 850 x 800 przedstawiające główny interfejs aplikacji. Cały interfejs programu jest przedstawiony w trybie graficznym. Argumenty określające parametry generowanego grafu są przekazywane przez użytkownika z klawiatury do konkretnych pól edycyjnych lub poprzez przycisk opcji (w przypadku wyboru spójności). Reszta pól to przyciski wywołujące akcje, tj. zapisywanie, wczytywanie, generowanie grafu itd. Po podjęciu akcji przez użytkownika wygenerowany lub wczytany graf wyświetli się na czarnym polu na środku ekranu. Działanie wyboru węzłów i rysowania ścieżek odbywają się na głównym oknie interfejsu. Przyciski `save graph`, `save path`, `read` inicjują pojawienie się nowych okien z opcjami wyboru plików do zapisu/wczytania. Nowe okna są również inicjowane w przypadku wystąpienia komunikatów z ostrzeżeniem/błędem oraz przy wyborze palety kolorów używanych do kolorowania grafu. Wszystkie z okien można zamknąć przyciskiem `"x"` lub po zatwierdzeniu zmian, powracając w ten sposób do głównego okna.

Opis modułów, pakietów i narzędzi

Przy tworzeniu programu zostały użyte nastąpujące dodatki:

- Moduł javafx.controls

  Moduł zawiera sterowniki i kompozycję dla graficznego interfejsu JavaFX.

- Pakiet java.awt

  Zawiera klasy dla zbudowania interfejsu użytkownika i stworzenia grafik.

- Pakiet java.util

  Zawiera podstawowe klasy (m.in Arrays, ArrayList, Date).

- Pakiet java.io

  Dostarcza obsługę operacji wejścia/wyjścia.

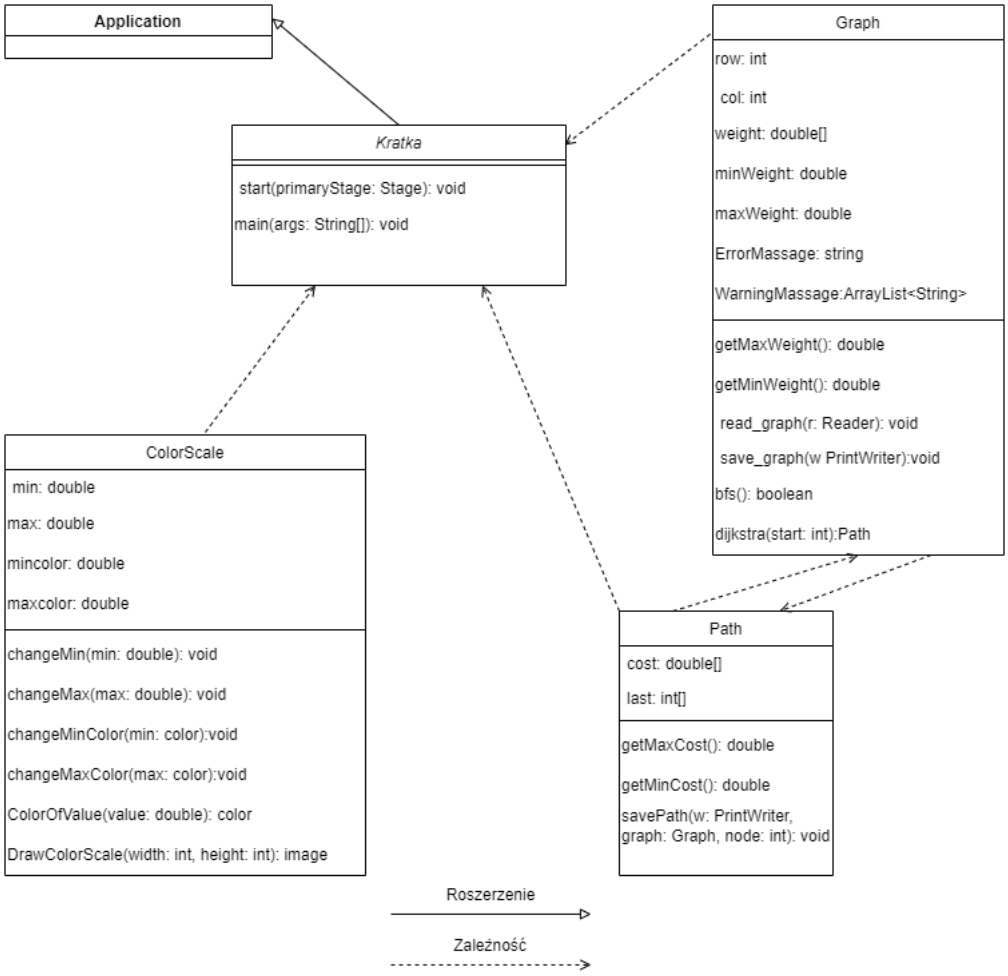
- Biblioteka JUnit

  Służy do przeprowadzania testów jednostkowych.

- Narzędzie Maven

  Umożliwia automatyzację procesu budowania aplikacji.

Diagram klas



Rys. 1: Diagram klas

Opis klas

Klasy zawarte są w jednym z dwóch pakietów:

Pakiet "Kratka":

- Kratka (dziedziczy po Application)

  Klasa odpowiada za wygląd i funkcjonowanie interfejsu graficznego. Zawiera funkcję main, uruchamiającą program.

Metody:

- start

  Metoda zastępuję metodę start, zdefiniowaną w klasie Application. W metodzie są zdefiniowane właściwości okna "Kratka", dodane etykiety, pola tekstowe, pole wyboru, przyciski z ustalonymi działaniami.

- main

  Metoda odpowiada za uruchomienie aplikacji.

- ColorScale

  Klasa odpowiada za zdefiniowanie i narysowanie skali kolorów.

  Metody:

  - void changeMin(double min)

    Ustawia min jako wartość minimalną na skali.

  - void changeMax(double max)

    Ustawia max jako wartość maksymalną na skali.

  - void changeMinColor(Color min)

    Metoda pozwala zmienić kolor wartości minimalnej na wybrany.

  - void changeMaxColor(Color max)

    Metoda pozwala zmienić kolor wartości maksymalnej na wybrany.

  - color ColorOfValue(double value)

    Zwraca kolor, odpowiadający wartości value zgodnie ze skalą.

  - image DrawColorScale(int width, int height)

    Tworzy i zwraca rysunek skali kolorów o szerokości width i wysokości height.

Pakiet "Graph":

- Graph

  Klasa definiuje postać przechowywania grafu i zawiera działania na grafie. Reprezentacją grafu w kodzie jest macierz sąsiedztwa. Metody:

  - double getMaxWeight()

    Metoda zwraca największą wagę krawędzi w grafie.

  - double getMinWeight()

    Metoda zwraca najmniejszą wagę krawędzi w grafie.

- void generateGraph(boolean connect)

  Metoda generuje krawędzie do grafu. Spójność jest określona przez flagę connect.

- void readGraph(Reader r)

  Metoda wykorzystuje plik otwarty w czytniku r, nadaje grafowi wczytane wartości liczb wierszy i kolumn i dodaje krawędzie.

- void saveGraph(PrintWriter w)

  Metoda zapisuje graf do pliku, otwartego w zapisywaczu w.

- boolean bfs()

  Metoda służy do zastosowania algorytmu przeszukiwania grafu wszerz, aby sprawdzić, czy graf jest spójny. Zwróci wartość false, jeżeli graf będzie niespójny, wartość true - spójny.

  Algorytm BFS krok po kroku:

  * Zaczynamy od węzła początkowego. Zaznaczamy go jako odwiedzony i dodajemy do kolejki wszystkie węzły, z którymi jest powiązany, w kolejności od węzła z najmniejszym indeksem.
  * Odwiedzamy następny węzeł w kolejce. Dodajemy do kolejki wszystkie węzły z nim powiązane i jeszcze nieodwiedzone.
  * Powtarzamy poprzedni krok, aż kolejka będzie pusta. Jeżeli wszystkie węzły w grafie zostały odwiedzone, to można stwierdzić, że graf jest spójny.

- Path dijkstra(int st) Metoda służy do odnajdywania kosztów dojścia od podanego wierzchołka st do wszystkich innych w grafie za pomocą algorytmu Dijkstry.

  Algorytm Dijkstry krok po kroku:

  * Dla każdego węzła ustawiamy długość ścieżki na nieskończoność lub wartość, która do niej dąży; długości przy węźle początkowym nadajemy wartość 0.
  * Oznaczamy węzeł jako odwiedzony. Dla każdego węzła połączonego z początkowym, przypisujemy długość równą wadze krawędzi ich łączących.
  * Z nieodwiedzonych węzłów znajdujemy węzeł o najmniejszej przepisanej długości. Oznaczamy go jako odwiedzony. Dla każdego węzła sąsiadującego z obecnym liczymy wartość „długość przy obecnym węźle + waga krawędzi łączącej”. Jeżeli znaleziona wartość jest mniejsza niż przypisana do sąsiadującego węzła, podmieniamy ją.

    * Powtarzamy poprzedni krok, aż zostaną odwiedzone wszystkie węzły. Po zakończeniu każdy węzeł będzie miał przypisaną długość najkrótszej ścieżki od węzła początkowego. Samą ścieżkę możemy odtworzyć od końca, jeżeli przy każdym przypisaniu węzłowi nowej długości będziemy zapamiętywali numer poprzedniego węzła.

- Path

  Klasa przechowuje listę wyliczonych kosztów dojścia do wierzchołków i listę poprzedników.

  – double getMaxCost()

    Metoda zwraca największy koszt dojścia ze wszystkich.

  – double getMinCost()

    Metoda zwraca najmniejszy koszt dojścia ze wszystkich.

  – void savePath(PrintWriter w, Graph graph, int node)

    Metoda zapisuje ścieżkę do wierzchołka node do pliku, otwartego w zapisywaczu w.

Obsługa zdarzeń

- Pole "Size"

  Pole będzie pobierało od użytkownika rozmiar grafu za pomocą getText().

- Pole "Edge weight range"

  Pole będzie pobierało od użytkownika zakres wartości wag krawędzi za pomocą getText().

- Przycisk opcji "Connectivity"

  Element Radio Box będzie pobierał wybraną opcję spójności grafu (jedną na raz) za pomocą ButtonGroup.getSelection().getActionCommand().

- Przycisk "Generate"

  Przycisk będzie uruchamiał wczytywanie wartości z pól "Size", "Edge weight range" i przycisku opcji "Connectivity" i uruchamiał generator grafu. Wartości te będą przekazywane do generatora w klasie Graph.

- Przycisk "Read"

  Przycisk uruchomi wyświetlanie okienka do wyboru pliku do wczytania grafu, uruchomi jego wczytanie do programu i narysowanie w interfejsie graficznym. Jedynym akceptowanym formatem pliku jest format tekstowy

`*.txt` (niemożliwe jest wybranie pliku o innym formacie). Filtrowanie formatu odbywa się przy użyciu getExtensionFilters().addAll(). Sprawdzana będzie poprawność zawartości pliku, to znaczy jego zgodność z przyjętą konwencją zapisu grafu, która jest następująca:

W pierwszej linii pliku podana jest liczba wierszy i kolumn grafu. Parametry do jednego węzła są definiowane poniżej w jednej linii. Numer węzła docelowego jest oddzielany od wagi spacją i znakiem `:`. Parametry dla jednego węzła są oddzielane dwiema spacjami.

Przykładowy plik:

```
2 3
 1 :0.8864916775696521  2 :0.2187532451857941
 0 :0.2637754478952221  5 :0.5486221194511974
 3 :0.5380334165340379
 2 :0.5486221194511974  5 :0.25413610146892474
 3 :0.31509645530519625  0 :0.40326574227480094
 4 :0.44272335750313074
```

Jeżeli format będzie prawidłowy, wywołana zostanie metoda rysująca graf. W przypadku błędu pojawi się wyjątek i zostanie wyświetlony komunikat.

- Przycisk "Save graph"

  Przycisk wyświetli okienko do utworzenia pliku do zapisania obecnego grafu i uruchomi jego zapisywanie w formacie tekstowym `*.txt`. Konwencja zapisu grafu do pliku jest tożsama z przedstawioną powyżej (Przycisk Read).

- Przycisk "Save path"

  Przycisk wyświetli okienko do utworzenia pliku do zapisania wybranych ścieżek i uruchomi zapisywanie w formacie tekstowym `*.txt`. Przykładowa treść zapisywana do pliku wygląda następująco:

  ```
  Najkrótsza ścieżka:
      0 -0.1635472537235685- 3 -0.6524342534143475- 5
  Długość ścieżki jest równa 0.815981507137916
  ```

  Do pliku zapisywane będą wszystkie ścieżki na rysunku grafu obecne w momencie zapisywania.

- Przycisk "Clear"

  Przycisk usunie wszystkie wyznaczone ścieżki w grafie.

- Przycisk "Delete"

  Przycisk usunie rysunek grafu z interfejsu.

- Modify color range

Przycisk zainicjuje wywołanie okna z wyborem zakresu kolorów używanych do rysowania ścieżek w grafie. Możliwy będzie wybór koloru oznaczającego wierzchołek o maksymalnym i minimalnym koszcie dojścia, krawędź o największej i najmniejszej wadze oraz kolor używany do rysowania ścieżki.

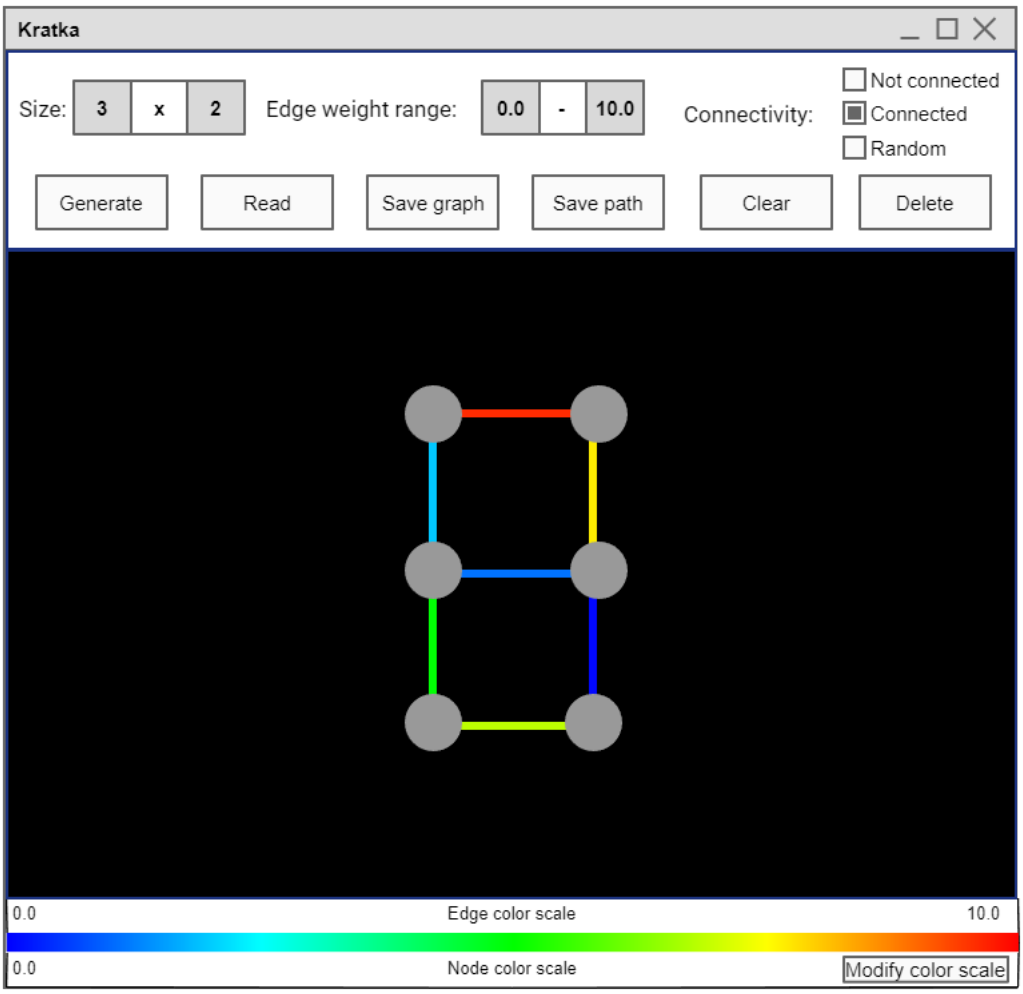
- Klikanie lewym klawiszem myszki na wierzchołek

  Kliknięcie na dany wierzchołek lewym przyciskiem myszy oznaczy go jako wierzchołek początkowy. Możliwe jest ponowne wybranie wierzchołka początkowego poprzez kliknięcie na inny wierzchołek. Dopóki nie zostanie wybrany wierzchołek końcowy, to nie rozpocznie się żadna akcja.

- Klikanie prawym klawiszem myszki na wierzchołek

  Kliknięcie na dany wierzchołek prawym przyciskiem myszy oznaczy go jako wierzchołek końcowy. Wybranie wierzchołka końcowego musi nastąpić po uprzednim wybraniu wierzchołka początkowego (kolejność wyboru nie może być zmieniona). Jeśli uprzednio żaden wierzchołek nie został wybrany jako początkowy, to nie rozpocznie się żadna akcja. W przypadku, gdy wierzchołki zostaną wybrane prawidłowo, uruchomiony zostanie algorytm BFS do sprawdzenia spójności grafu. Po jego pomyślnym zakończeniu zostanie uruchomiony algorytm Dijkstry. Nastąpi przypisanie kolorów do każdego z wierzchołków z zakresu Node Color Range. Wartości kolorów odwzorowują koszty dojścia do każdego z wierzchołków od wierzchołka początkowego. Najkrótsza ścieżka między wierzchołkiem końcowym a początkowym zostanie narysowana i pokolorowana (w przypadku domyślnym) na biało.

Budowa GUI



Rys. 2: Projekt głównego okna interfejsu

Testowanie

Testy są generowane przy pomocy biblioteki JUnit umożliwiającej przeprowadzanie testów jednostkowych. Testy przechowywane są w folderze `src/test/java`. Testowana jest większość obecnych w kodzie metod, a rodzaj asercji jest zależny od typu zwracanego przez metodę.

- assertTrue();

  używane do metod zwracających typ boolean, np. `boolean bfs()`. Przykładowo, algorytm bfs jest testowany na znanych grafach (zarówno spójnych i niespójnych); dla spójnego powinien zwracać wartość true, a dla niespójnego - false.

- assertEquals();

  używane do metod zwracających typ double, np. `double getMaxCost()`, `double getMaxWeight()`. W tych testach korzystamy ze znanych wartości długości wag i kosztów w danym grafie i przyrównujemy wartości zwrócone przez metody z oczekiwanymi.

- assertArrayEquals();

  używane do algorytmu Dijkstry w celu sprawdzenia poprawności wyznaczania najkrótszej ścieżki oraz jej przebiegu.

- assertThrows();

  używane do metod zwracających typ void, np. `void readGraph(Reader r)`, `void generateGraph(boolean connect)`. Testy mają na celu wyłapanie błędów dotyczących nieprawidłowo podanych parametrów przy tworzeniu obiektu grafu, niezgodności formatu wczytywanego pliku.

Przykładowe testy jednostkowe:

Poniższy test ma na celu sprawdzenie, czy metoda `boolean bfs()` zwróci wartość `true` w przypadku podania grafu spójnego.

```
@Test
public void ConnectedGraphTrue1(){

    Graph graph = new Graph(2, 3, new double[]{0.00, 1.00, 0.00, 4.00, 0.00,
    0.00, 1.00, 0.00, 3.00, 0.00, 0.00, 0.00, 0.00, 3.00, 0.00,
    0.00, 0.00, 7.00, 4.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
    0.00, 0.00, 0.00, 0.00, 2.00, 0.00, 0.00, 7.00, 0.00, 2.00, 0.00});

    assertTrue(graph.bfs());
    }
```

Poniższy test ma na celu sprawdzenie, czy metoda `double getMaxWeight()` zwróci największą długość wagi w grafie.

```
@Test
public void MaxWeight7(){

    Graph graph = new Graph(2, 3, new double[]{0.00, 1.00, 0.00, 4.00, 0.00,
    0.00, 1.00, 0.00, 3.00, 0.00, 0.00, 0.00, 0.00, 3.00, 0.00,
    0.00, 0.00, 7.00, 4.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
    0.00, 0.00, 0.00, 0.00, 2.00, 0.00, 0.00, 7.00, 0.00, 2.00, 0.00});
```

```
assertEquals(7, graph.getMaxWeight());
}
```

Poniższy test ma na celu sprawdzenie poprawności generowania spójnego grafu przez metodę `void generateGraph(boolean connect)`. Będzie także przeprowadzony test na generowanie grafu niespójnego z podobnym algorytmem.

```
@Test
void CheckGenerationOfConnectedGraph() {
    Graph graph = new Graph(3, 2);
    graph.generateGraph(true);
    assertTrue(graph.bfs());
}
```

Poniższy test ma na celu sprawdzenie poprawności metod zapisywania i odczytywania pliku `void saveGraph(PrintWriter w)` i `void readGraph(Reader r)`.

```
@Test
void CheckReadingAndSaving(){
    Graph graph = new Graph(3,2);
    graph.generateGraph(true);
    File write;
    write = new File("filename.txt");
    try {
        PrintWriter w = new PrintWriter (write);
        graph.saveGraph(w);
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
    Graph graph1 = new Graph();
    Reader r;
    try {
        r = new FileReader(write);
        graph1.readGraph(r);
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
    assertEquals(graph.row, graph1.row);
    assertEquals(graph.col, graph1.col);
    assertArrayEquals(graph.weights, graph1.weights);
}
```

Następny test służy do sprawdzania poprawnego działania algorytmu Dijkstry.

```
    @Test
void checkDijkstra() {
    Graph graph = new Graph(2, 3, new double[]{0.00, 1.00, 0.00, 4.00, 0.00, 0.00,
            1.00, 0.00, 3.00, 0.00, 0.00, 0.00,
```

```
                0.00, 3.00, 0.00, 0.00, 0.00, 7.00,
                4.00, 0.00, 0.00, 0.00, 0.00, 0.00,
                0.00, 0.00, 0.00, 0.00, 0.00, 2.00,
                0.00, 0.00, 7.00, 0.00, 2.00, 0.00});
        Path path = graph.dijkstra(0);
        assertArrayEquals(path.cost, new double[]{0.0, 1.0, 4.0, 4.0, 13.0, 11.0});
        assertArrayEquals(path.last, new int [] {-1,0,1,0,5,2});
    }
}
```

Reszta testowania odbywa się w środowisku graficznym, poprzez korzystanie z opcji interfejsu i sprawdzanie, czy wyniki działania są zgodne z oczekiwanymi.