# Dokumentacja implementacyjna

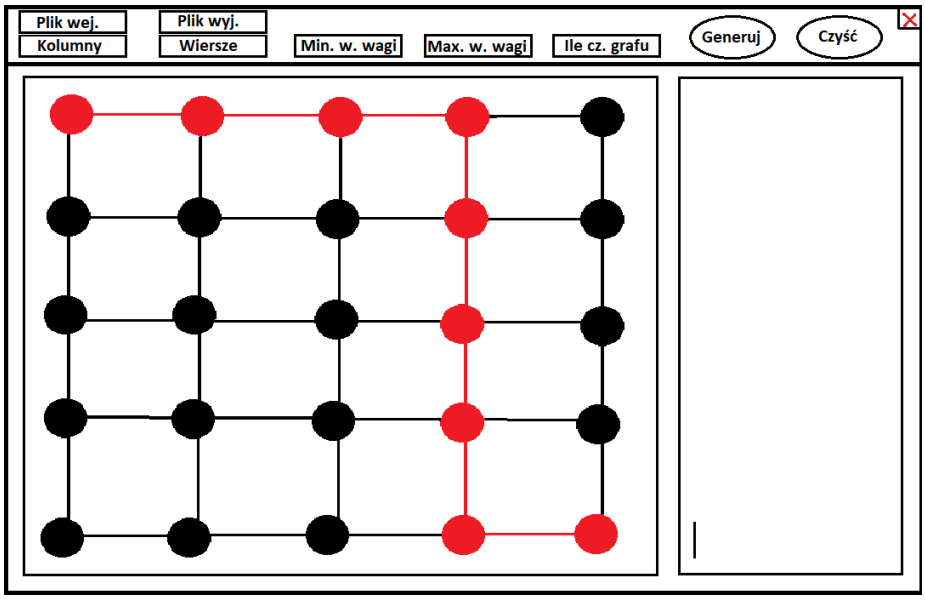
Mateusz Charczuk, Antoni Rosiecki

May 2022

## 1 Cel projektu

Celem projektu jest napisanie programu (zawierającego interfejs użytkownika) który wygeneruje graf o podanej liczbie kolumn i wierszy, posiadającego losowe wartości na krawędzi. Dodatkowo program sprawdza czy graf jest spójny, jest w stanie określić najkrótszą drogę pomiędzy jego dowolnymi wierzchołkami oraz zapisuje go do pliku o podanym formacie.

## 2 Interfejs graficzny




Rysunek 1: Interfejs graficzny

# 3 Zastosowane algorytmy

1. Algorytm Dijkstry - Algorytm służący do znajdowania najkrótszej ścieżki między wybranym węzłem a wszystkimi innymi, w grafie w którym ścieżki mają wartości nie ujemne. Przy odpowiedniej modyfikacji tego algorytmu oraz przy użyciu kolejki priorytetowej można sprawić, by wyliczał jedynie najkrótszą ścieżkę między dwoma punktami.
   Działanie algorytmu Dijkstry:
   1. Dla każdego węzła należy w zapisać w strukturze jego poprzednika
   2. Dodać do kolejki wagę połączenia.
   3. Z kolejki wybrać węzeł o najmniejszej wadze połączenia.
   4. Następnie z pozostałych węzłów wybrać ten o najmniejszej wadze połączenia. Kroki 3 i 4 powtarzać, do momentu aż kolejka będzie pusta.

2. BFS - Algorytm służący do przeszukiwaniu grafu w postaci drzewa. Przeszukanie polega na wybraniu węzła a następnie odwiedzeniu wszystkich sąsiadujących z nim wierzchołków.Do zastosowania tego algorytmu potrzebna jest implementacja kolejki priorytetowej.

   działanie algorytmu:
   1. Sprawdzanie czy węzeł jest tym który szukamy, jeżeli tak to zwrócenie go i wyjście z funkcji.
   2. Jeżeli sąsiadujący węzeł nie jest odwiedzony to oznacz go jako odwiedzony.
   Kroki 1 i 2 powtarzaj, do momentu gdy kolejka będzie pusta.

# 4 Struktura danych

1. Dane z pliku / wygenerowane za pomocą parametrów wywołania są początkowo przechowywane w tablicy struktur o liczbie elementów odpowiadającej iloczynowi kolumn i wierszy grafu. Struktura ta zawiera 4 elementową tablicę intów w której przechowywane są indeksy wierzchołków połączonych z wierzchołkiem któremu odpowiada indeks w tablicy struktur. Kolejnym elementem który przechowuje struktura jest 4 elementowa tablica intów zawierająca wagi połączeń między indeksem tablicy struktur i wartością tablicy 4 elementowej wierzchołków (w ramach tej samej struktury) o tych samych indeksach.
   Dokładny wygląd definicji struktury:

2. Kolejka – liniowa struktura danych, w której nowe dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania.(Domyślnie zaimplementowano w javie jako Queue)

3. Kolejki Deque - struktura danych rozszerzająca interfejs zwykłej kolejki

w Javie, zapewnia względem niej dodatkowe funkcje.(Domyślnie zaimplementowano w javie jako Deque)

4. Klasa ArrayList - jest implementacją listy, w taki sposób, że kolejne elementy znajdują się obok siebie w wewnętrznej tablicy. Element i oraz i+1 sąsiadują ze sobą nie tylko jako elementy kolekcji, ale również w pamięci operacyjnej komputera. Główną różnicą w stosunku do poznanych wcześniej tablic jest to, że rozmiar może zmieniać się dynamicznie w trakcie działania programu.

# 5 Opis modułów

1. MainAplication.java - rozszerza klasę Application, Służy jedynie do wywołania funkcji pozwalających uruchomić interfejs graficzny.

   Funkcje:

   - main() - służy do wywołania funkcji launch() zawartej w klasie nadrzędnej. Przyjmuje tablice zmiennych typu string, nic nie zwraca.
   - start() - Odpowiada częściowo za wygląd okna GUI, ładuje oraz wczytuje scenę z pliku FXML oraz podpina ją do Stage oraz wyświetla. Przyjmuje obiekt typu Stage oraz wyrzuca wyjątek IOException, nic nie zwraca.
   - quit() - Odpowiada za obsłużenie sytuacji w której użytkownik wciśnie przycisk x w prawym górnym rogu okna GUI. Przyjmuje obiekt typu Stage, nic nie zwraca.

2. dijkstra.java - Odpowiada za realizacje algorytmu Dijkstry w programie.

   Funkcje:

   - dijkstra - przymjmuje obiekt graph, int start(początek wyznaczony przez użytkownika) oraz int end(koniec wyznaczony przez użytkownika). Zwraca tablice wierzchołków przez które prowadzi najkrótsza droga.
   - getResult - nie przyjmuje żadnych parametrów, zwraca zmienną Dijkstra.result będacą double.

3. DijkstraResultToGraphPane.java - tworzy graficzne przedstawienie grafu na odpowiednim Pane w interfejsie użytkownika.

4. GenerationGrapPane.java - tworzy interfejs GUI.
   Funkcje:

- generationGraphPane - przyjmuje ScrollPane, int row(rzędy), int col(kolumny), obiekt Graph oraz ActionEvent e. Nie zwraca nic, bo generuje interfejs uzytkownika.

5. Bfs.java - Odpowiada za sprawdzenie spójności grafu. Klasa odpowiedzialna za obsługę algorytmu bfs.

   Funkcje:

   - breath_first_search - funkcja odpowiedzialna za odpowiednie przeprowadzenie algorytmu bfs według wskazanych powyżej zasad. Przyjmuje obiekt Graph. Zwraca wartość int 1 gdy graf jest spójny, 0 gdy graf jest nie spójny.

6. GraphGen.java - zawiera funkcje odpowiadające za generacje grafu z parametrów z interfejsu użytkownika oraz wpisuje je do obiektu Graph.

   Funkcje:

   - graphgen - generuje graf na podstawie prarmetrów z GUI. Przyjmuje obiekt Graph, int minwage(waga minimalna) oraz maxwage(waga maksymalna)
   - neighbourGen - odpowiada za odpowiednie wygenerowanie sasiadow wierzchołka. Przyjmuje int i, j, l zmienne z funkcji graphgen, int min(minimalna waga), int max(maksymalna waga), obiekt Graph. Funkcja nic nie zwraca.

7. Graph.java - Jest to klasa definiująca strukturę przechowywania grafu w programie. Zmienne:

   - int row - ilość wierszy,
   - int col - ilość kolumn,
   - int rowcol - liczba odpowiadająca iloczynowi wierszy i kolumn.
   - Neighbours[] graph - tablica Obiektów typu Neighbour która przechowuje cały graf.

   Konstruktory:

   - konstruktor1 - przyjmuje int row(liczba rzędów), int col(liczba kolumn), double minwage(minimalna wartość wagi), double maxwage(maksymalna wartość wagi). Wywołuje funkcje graphgen która generuje graf na podstawie podanych argumentów.
   - konstruktor2 - przyjmuje String fName(nazwa pliku z grafem), Label stdErr(Obiekt typu Label przjmujący rolę konsoli komunikatów w GUI). Wywołuje funckje graphFromFile która generuje graf z podanego pliku.

Funkcje:

- getRow() - zwraca zmienną int row, nie przyjmuje żadnych argumentów.
- getCol() - zwraca zmienną int col, nie przyjmuje żadnych argumentów.
- getRowcol() - zwraca zmienną int rowcol, nie przyjmuje żadnych argumentów.
- setRow() - przyjmuje zmienną int row, nic nie zwraca. Służy do ustawienia zmiennej danego obiektu na tą podaną jako argument.
- setCol() - przyjmuje zmienną int col, nic nie zwraca. Służy do ustawienia zmiennej danego obiektu na tą podaną jako argument.
- setRowcol() - przyjmuje zmienną int rowcol, nic nie zwraca. Służy do ustawienia zmiennej danego obiektu na tą podaną jako argument.
- weightGen() - przyjmuje zmienne double minwage(minimalna wartość wagi), double maxwage(maksymalna wartość wagi), int l(konkretny indeks tablicy graph). Funkcja Zapisuje losowo wygenerowaną liczbę rzeczywistą typu double (znajdującą się pomiędzy minwage oraz maxwage) do tablicy graph pod indeksem l.
- printToFile() - przyjmuje zmienną typu String zawierającą nazwę pliku do których dane mają zostać zapisane. Funkcja wyrzuca wyjątek IOException, nic nie zwraca. Służy ona do wypisania wygenerowanego grafu do pliku którego nazwa została podana.
- printToScreen() - funkcja nic nie przyjmuje oraz nic nie zwraca. Służy do wypisania struktury grafu na konsolę błędów.

8. GraphSlicing - Abstrakcyjna klasa obsługuje cięcie grafu.
   Funkcje:

   - graph_slice - funkcja przyjmuje Obiekt typu graf g oraz int n(ilość części na które podzieli się graf). Funkcja jest statyczna oraz nic nie zwraca. Zajmuje się cięciem grafu na części.

9. Neighbours - Kolejna klasa która przechowuje sąsiadów i wagi połączeń do nich w tablicach ArrayList, do konkretnych wierzchołków.
   Zmienne:
   private ArrayList¡Integer¿ neighbour - tablica typu ArrayList która przechowuje maksymalnie 4 wierzchołki typu Integer.
   private ArrayList¡Double¿ wage - tablica typu ArrayList która przechowuje maksymalnie 4 wagi typu Double.
   Funkcje:

   - addNeighbour - przyjmuje wartość typu int item (wartość która zostanei dodana ), funkcja nic nie zwraca. Funkcja dodaje podaną wartość do tablicy neighbour.

- getNeighboorsSize - funkcja nic nie przyjmuje, zwraca wielkość tablicy neighbours jako int.
- neighbourPopOnIndx - funkcja przyjmuje int indx - indeks z którego bierzemy wartość w tablicy neighbours, funkcja zwraca int
- neighbourRemoveOnIndx - robi to samo co funkcja wyżej tyle że zwraca i usuwa wartość z tablicy pod indeksem indx.
- wagePopOnIndx - robi to samo co neighbourPopOnIndx tyle że w tablicy wage i pobiera double i zwraca double.
- wageRemoveOnIndx - robi to samo co funkcja wyżej tyle że zwraca i usuwa wartość z tablicy pod indeksem indx.

# 6 Sytuacje wyjątkowe

- Błędy w pliku wejściowym:

  1. Źle skonstruowana pierwsza linijka pliku komunikat:
     Format pliku jest niepoprawny( 1 linijka )
  2. Enter postawiony w złym miejscu pliku uniemożliwiający jego poprawne odczytanie komunikat:
     Format pliku jest niepoprawny(entery)
  3. znak : postawiony w złym miejscu pliku uniemożliwiający jego poprawne odczytanie komunikat:
     Format pliku jest niepoprawny(brakuje : przed wagą )
  4. Wartości które powinny być liczbami w odpowiednich miejscach w pliku nimi nie są komunikat:
     Format pliku jest niepoprawny: wartości podane przez użytkownika nie są liczbami
  5. Liczby podane w pliku są ujemne komunikat:
     Format pliku jest niepoprawny: liczby podane przez użytkownika są ujemne
  6. Plik kończy się nie w tym miejscu co trzeba lub dotychczasowo wczytane wartości nie są wystarczające by zapisać graf o podanych rozmiarach komunikat:
     Format pliku jest niepoprawny(EOF)

- Błędy w parametrze wywołania:

  1. Gdy użytkownik poda nazwę pliku który nie istnieje bądź nie da się go otworzyć pojawi się komunikat:
     KRYTYCZNY BŁĄD: Podany plik wejściowy nie istnieje

2. Gdy użytkownik jako argument który powinien być liczbą poda nie liczbę wyświetli się komunikat:
   KRYTYCZNY BŁĄD: Wartość podana przez użytkownika nie jest liczbą
3. Gdy użytkownik jako argument który powinien być liczbą dodatnią poda nie liczbę dodatnią wyświetli się komunikat:
   KRYTYCZNY BŁĄD: Wartość podana przez użytkownika jest ujemna
4. gdy użytkownik wciśnie przycisk świadczący o chęci wygenerowania grafu przez program z argumentów linii polecenia a nie poda wystarczającej ich ilości pojawi się komunikat:
   Nie podano wystarczającej ilości argumentów aby wygenerować graf