JIMP2 Projekt 2025

Dokumentacja końcowa - Java

Michał Ludwiczak
GR3

10 czerwca 2025

Spis treści

1 Cel projektu **2**

2 Problem **3**

3 Algorytm **3**
3.1 Macierz Laplace'a 4
3.2 Obliczenie par własnych 4
3.3 Podział 4
3.3.1 Podział według wektora Fiedlera 4
3.3.2 Klasteryzacja 5
3.4 Wynik 5

4 Interfejs graficzny **5**

5 Struktura projektu **9**

6 Architektura MVC **10**

7 Uproszczony diagram klas UML **11**

8 Najważniejsze klasy i funkcje w programie **11**

9 Pliki wejściowe i wyjściowe dla trybu dzielenia **16**
9.1 Plik wejściowy (`.csrrg`) 16
9.2 Plik wyjściowy (`.csrrg2` / `.bin`) 17

10 Pliki wejściowe dla trybu wyświetlenia podzielonego grafu **18**
10.1 Plik wejściowy w formacie `.csrrg` 18
10.2 Plik wejściowy w formacie binarnym (`.bin`) 19

11 Obsługa błędów i wyjątków **19**

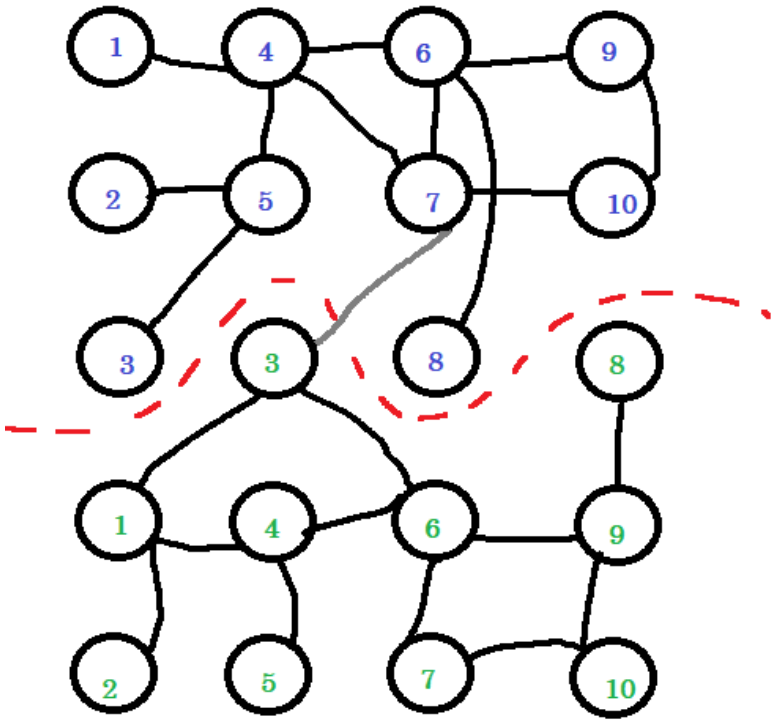
12 Przykłady użycia **20**
12.1 Wczytywanie grafu z pliku tekstowego 20
12.2 Podział grafu na `x` części z marginesem `y%` 20
12.3 Zapisanie wyniku do pliku tekstowego 20
12.4 Zapisanie wyniku do pliku binarnego 20
12.5 Przykładowe pliki 21

13 Wymagania niefunkcjonalne **21**

14 Podsumowanie **21**

1 Cel projektu

Cel projektu to stworzenie aplikacji w języku Java, umożliwiającej podział grafu na określoną przez użytkownika liczbę części, z zachowaniem określonego lub domyślnego 10-procentowego marginesu różnicy liczby wierzchołków pomiędzy częściami. Domyślnie graf dzielony jest na dwie części. Celem podziału jest także minimalizacja liczby przeciętych krawędzi pomiędzy powstałymi częściami grafu. Aplikacja ma być wyposażona w graficzny interfejs użytkownika wykonany w technologii Swing. Użytkownik będzie mógł wczytywać już podzielony graf z pliku tekstowego lub binarnego, wczytywać niepodzielone grafy, definiować liczbę części oraz margines podziału, a także zapisywać wynikowy graf wraz z danymi wejściowymi do pliku tekstowego lub binarnego. Każda część grafu będzie prezentowana w interfejsie graficznym w odrębnym kolorze.



Rysunek 1: Przykładowy graf podzielony na 2 równe części

2 Problem

Podział grafu na części w taki sposób, aby liczba przeciętych krawędzi była jak najmniejsza, nie jest problemem łatwym. Znalezienie optymalnego rozwiązania dla dużych grafów jest praktycznie niewykonalne w rozsądnym czasie, ponieważ liczba możliwych podziałów rośnie wykładniczo wraz z liczbą wierzchołków. W przypadku podziału grafu na dwie części istnieje $2^{n-1}-1$ możliwych sposobów podziału, gdzie n oznacza liczbę wierzchołków. Z tego powodu zamiast sprawdzania wszystkich możliwych podziałów stosuje się algorytmy przybliżone, heurystyki lub algorytmy zachłanne, które pozwalają na szybkie znalezienie dobrego, choć niekoniecznie optymalnego rozwiązania.

3 Algorytm

W programie zdecydowałem się użyć metody spektralnej dzielenia grafu, która wykorzystuje własności spektralne macierzy Laplace'a grafu. Na podstawie spektrum grafu, które

opisuje jego strukturę graf możemy podzielić w efektywny i satysfakcjonujący sposób, minimalizując liczbę przeciętych krawędzi.

3.1 Macierz Laplace'a

W podejściu spektralnym do podziału grafu kluczową rolę odgrywa macierz Laplace'a [2]. Wzór na Laplacian klasyczny definiuje się jako:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

gdzie:

- L to **macierz Laplace'a** grafu,
- D to **macierz stopni** to macierz diagonalna, w której elementy na diagonali d_{ii} odpowiadają stopniowi wierzchołka i, czyli liczbie krawędzi, które są z nim bezpośrednio połączone. W przypadku pętli, każda taka krawędź zwiększa stopień wierzchołka o 2, ponieważ jest traktowana jako dwie krawędzie incydentne z tym samym wierzchołkiem
- A to **macierz sąsiedztwa** grafu, gdzie elementy a_{ij} są równe 1, jeśli istnieje krawędź między wierzchołkami i i j, oraz 0 w przeciwnym przypadku.

3.2 Obliczenie par własnych

Następnym krokiem jest obliczenie $k = p - 1$ (gdzie p to liczba części, na które graf ma być podzielony) najmniejszych wektorów własnych, z których pierwszy zerowy zostanie pominięty. Dla bisekcji będzie to jeden wektor, tzw. drugi najmniejszy – wektor Fiedlera. Wektory własne macierzy Laplace'a grafu przechowują informacje o połączeniach pomiędzy wierzchołkami.

Do obliczenia tych par własnych wykorzystano bibliotekę ARPACK (poprzez binding Java: `com.github.fommil.netlib.ARPACK`), która implementuje metodę Implicitly Restarted Lanczos Method (IRLM) – specjalizowaną wersję metody Arnoldiego dla macierzy symetrycznych. ARPACK wymaga jedynie implementacji mnożenia macierzy przez wektor, co pozwala na efektywne wykorzystanie pamięci proporcjonalnie do liczby niezerowych elementów macierzy (format CSR). Dzięki temu rozwiązanie dobrze sprawdza się przy analizie dużych, rzadkich grafów.

3.3 Podział

3.3.1 Podział według wektora Fiedlera

Podział grafu według wektora Fiedlera polega na wykorzystaniu drugiego najmniejszego wektora własnego macierzy Laplasjanu grafu, nazywanego wektorem Fiedlera. Ten wektor koduje informację o strukturze spójności grafu – wierzchołki o podobnych wartościach współrzędnych tego wektora leżą blisko siebie w grafie. W implementacji sortuje się wierzchołki według wartości wektora Fiedlera i dzieli na dwie grupy o równej liczności.

3.3.2 Klasteryzacja

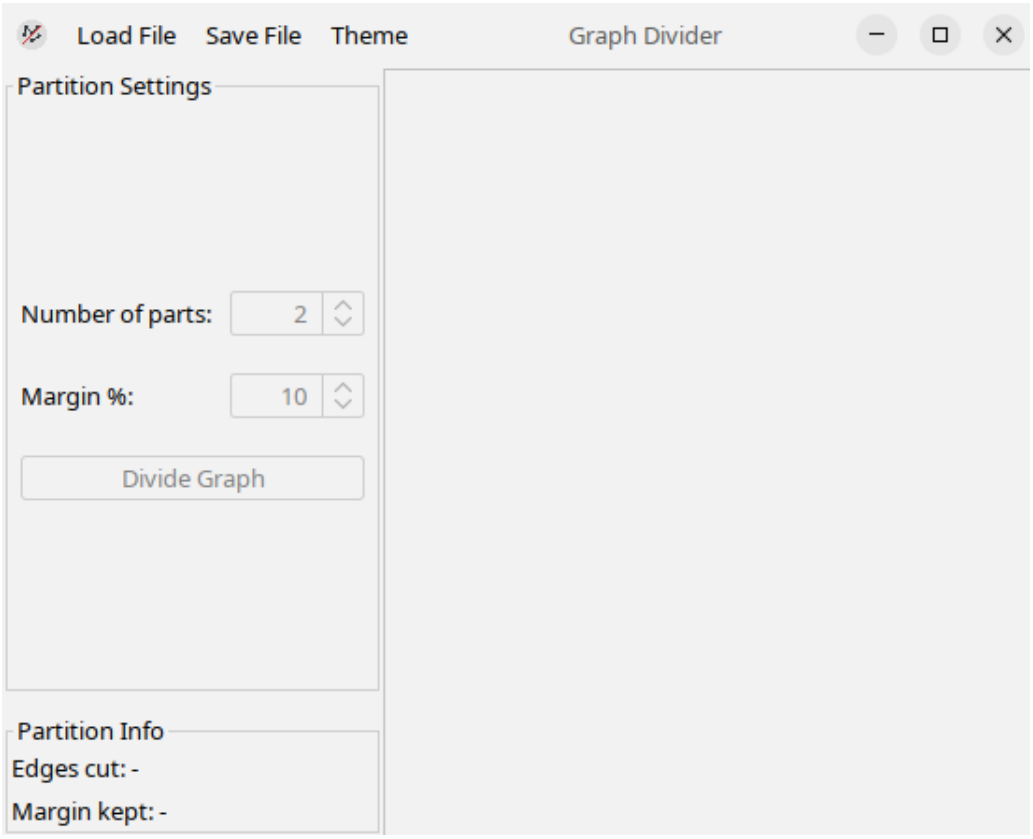
Po otrzymaniu macierzy zawierającej k wektorów własnych (gdzie $k = p$), każdy wierzchołek grafu jest reprezentowany jako punkt w k-wymiarowej przestrzeni. W celu podziału grafu na p części, zastosowano algorytm centroidów (k-means) [3]. Centroidy są inicjalizowane równomiernie wzdłuż głównych osi na podstawie zakresu wartości współrzędnych. Następnie iteracyjnie przypisuje się wierzchołki do najbliższych centroidów, z zachowaniem ograniczenia liczebności każdego klastra (tak, aby podziały były możliwie równe). Po każdej iteracji aktualizowane są położenia centroidów na podstawie średniej pozycji przypisanych punktów. Proces powtarzany jest do momentu osiągnięcia zbieżności (braku zmian centroidów) lub osiągnięcia maksymalnej liczby iteracji.

3.4 Wynik

Po otrzymaniu poszczególnych części z wierzchołkami modyfikuje się macierz sąsiedztwa A w formacie CSR usuwając krawędzie pomiędzy wierzchołkami przynależącymi do różnych grup. W ten sposób otrzymuje się nową macierz sąsiedztwa, która zawiera już podzielony graf. Macierz sąsiedztwa jest już gotowa do przetworzenia i wypisania na plik wyjściowy.

4 Interfejs graficzny

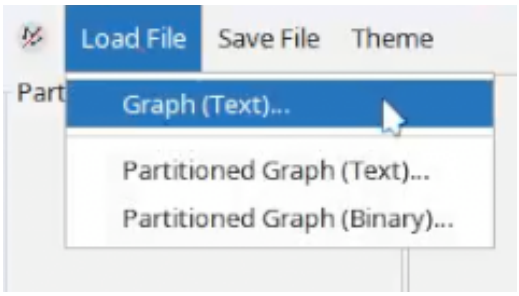
Aplikacja została zaprojektowana z wykorzystaniem biblioteki Swing i oferuje graficzny interfejs użytkownika, który umożliwia interaktywną obsługę procesu podziału grafu.



Rysunek 2: Interfejs programu GraphDivider

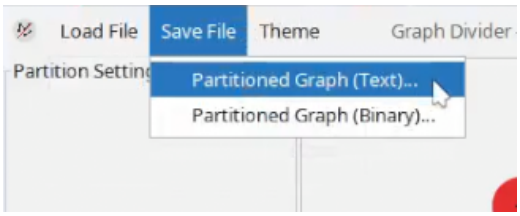
Główne okno aplikacji składa się z:

- **Paska menu**, który umożliwia:
  - wybranie pliku z grafem do wczytania:
    - niepodzielony graf w formacie tekstowym,
    - podzielony graf w formacie tekstowym,
    - podzielony graf w formacie binarnym



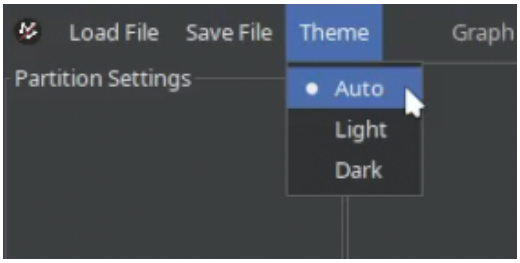
Rysunek 3: Opcja wczytania pliku

– zapisanie pliku z podzielonym grafem w formacie:
  * tekstowym,
  * binarnym



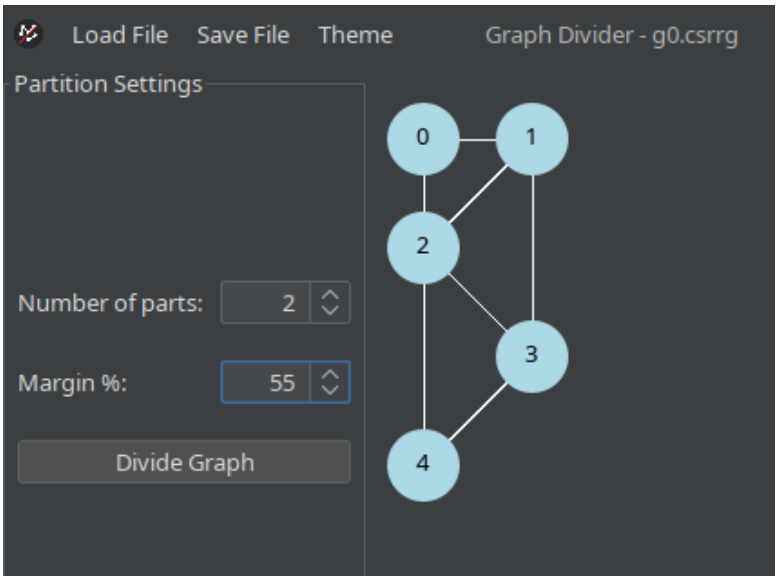
Rysunek 4: Opcja zapisu pliku

– zmianę motywu:
  * automatyczny / systemowy,
  * jasny,
  * ciemny



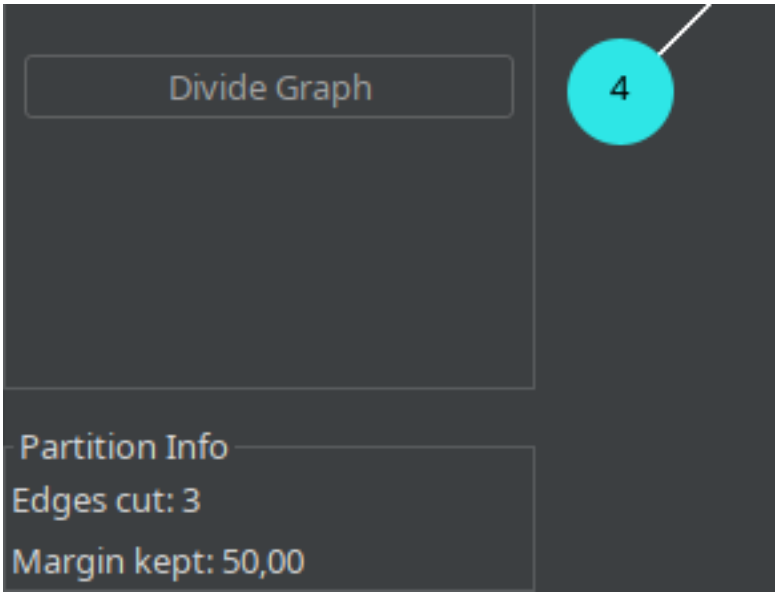
Rysunek 5: Opcja motywu

- **Bocznego panelu podziału**, który umożliwia wybranie liczby części oraz marginesu podziału, a także rozpoczęcie podziału.



Rysunek 6: Boczny panel podziału

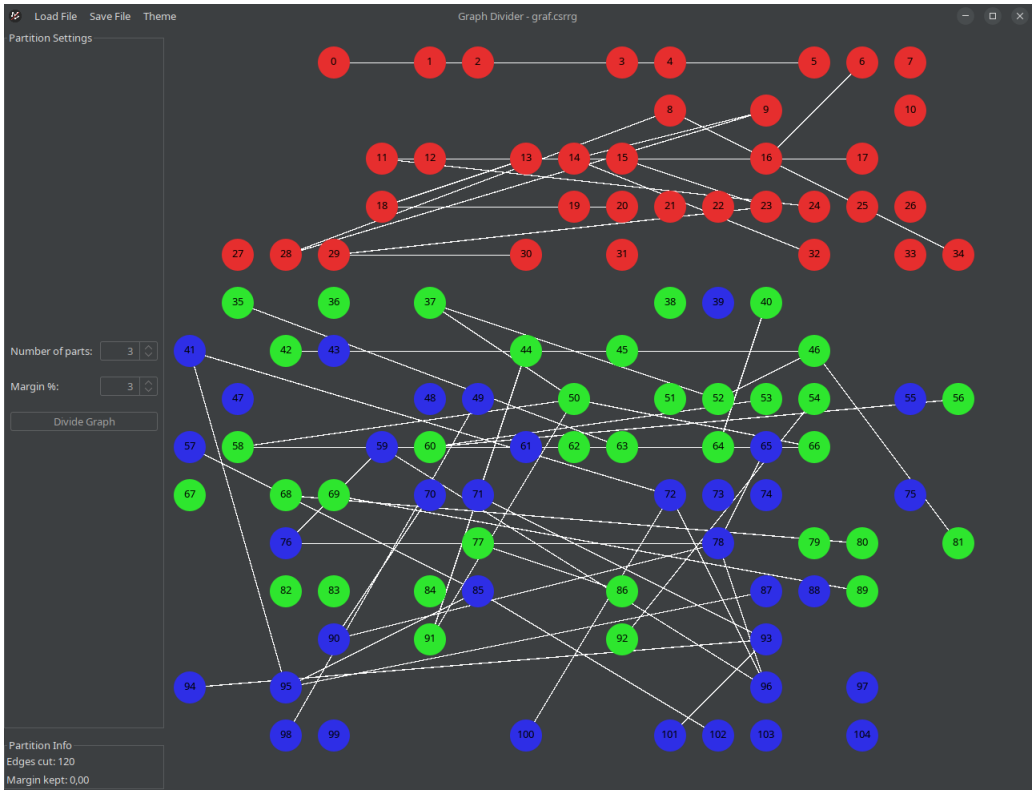
- **Panelu informacji**, który wyświetla liczbę przeciętych krawędzi i zachowany margines po podziale.



Rysunek 7: Panel informacji o podziale

- **Panelu grafu**, który wyświetla zarówno graf przed podzieleniem, jak i po podzie-

leniu kolorując poszczególne podgrafy.



Rysunek 8: Panel grafu (po podziale)

5 Struktura projektu

Projekt korzysta z systemu budowania Gradle. Wszystkie zależności oraz zadania budowania są zdefiniowane w pliku `build.gradle`. Do kompilacji i uruchamiania aplikacji wystarczy użyć poleceń `./gradlew build` / `gradle build` oraz `./gradlew run` / `gradle run`. Do poprawnej kompilacji i uruchomienia projektu wymagane jest również środowisko Java Development Kit (JDK) w wersji co najmniej 21.

Projekt został zorganizowany w następujący sposób:

- `build.gradle` – główny plik konfiguracyjny Gradle, definiuje zależności i zadania budowania projektu.
- `settings.gradle` – plik ustawień projektu Gradle.
- `gradlew`, `gradlew.bat`, `gradle/wrapper/` – skrypty i pliki wrappera Gradle, umożliwiające budowanie projektu bez konieczności instalowania Gradle globalnie.
- `build/` – katalog generowany automatycznie przez Gradle, zawiera artefakty kompilacji.

- `docs/` – dokumentacja projektu (funkcjonalna, implementacyjna, diagramy UML).
- `src/main/java/graphdivider/` – główny kod źródłowy aplikacji:
  - `Main.java` – klasa uruchamiająca aplikację.
  - `controller/` – logika sterująca aplikacją.
  - `io/` – obsługa wejścia/wyjścia (wczytywanie i zapisywanie plików).
  - `model/` – klasy modelujące strukturę grafu oraz algorytmy (m.in. macierz Laplace'a, obliczanie par własnych, klasteryzacja).
  - `view/` – klasy odpowiedzialne za interfejs graficzny (Swing).
- `src/main/resources/` – zasoby aplikacji:
  - `graphs/` – przykładowe pliki wejściowe z grafami w formacie .csrrg.
  - `divided_graphs/` – pliki wyjściowe z programu w C z podzielonymi grafami w formacie tekstowym i binarnym.
  - `icon/` – pliki graficzne, ikony.
  - `output/` – katalog na generowane pliki wyjściowe.

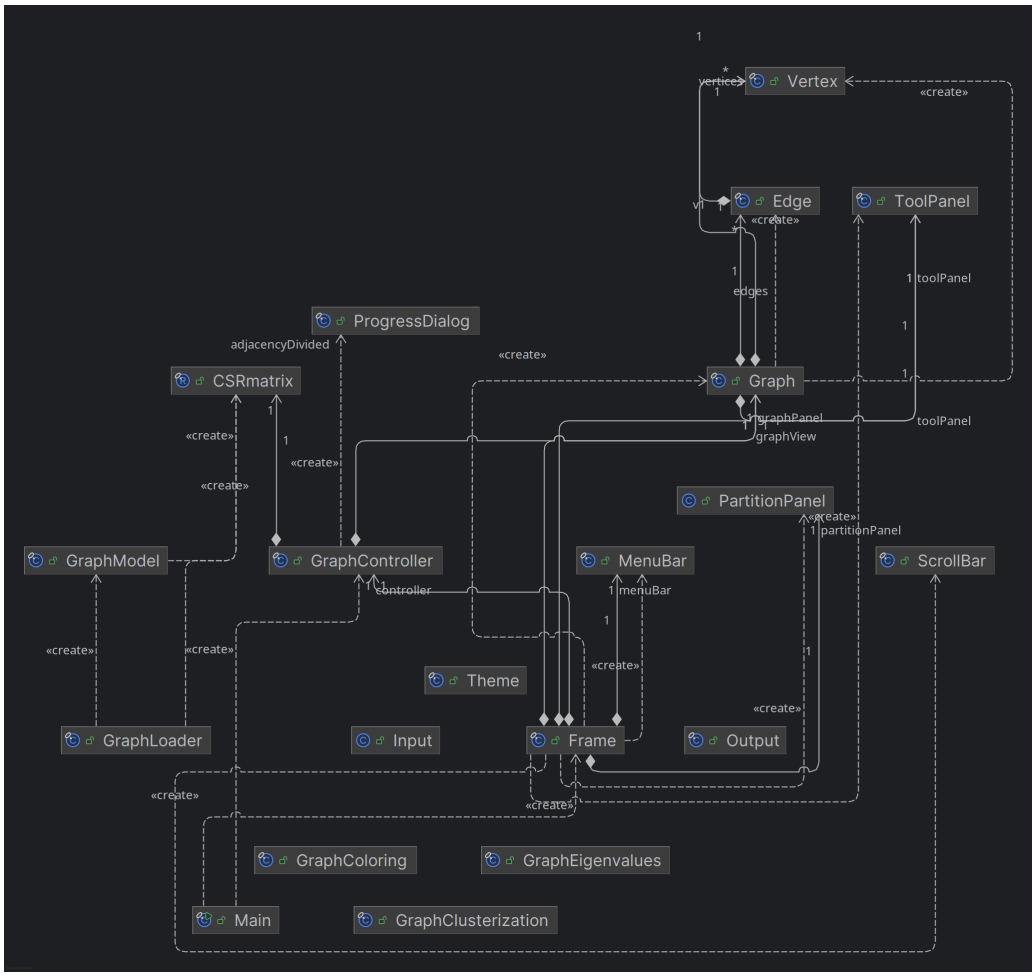
6 Architektura MVC

W projekcie zastosowano wzorzec architektoniczny **Model-View-Controller (MVC)**. Logika aplikacji została podzielona na trzy główne warstwy:

- **Model** – odpowiada za reprezentację danych oraz logikę związaną z przetwarzaniem grafów (np. klasy do przechowywania struktury grafu, obliczania wartości własnych, podziału na części).
- **View** – odpowiada za prezentację danych użytkownikowi oraz interakcję z użytkownikiem (np. okna, panele, menu).
- **Controller** – pośredniczy pomiędzy modelem a widokiem, obsługuje zdarzenia użytkownika i steruje przepływem danych w aplikacji.

Takie podejście ułatwia rozwój, testowanie oraz utrzymanie kodu, a także pozwala na łatwą rozbudowę interfejsu użytkownika bez ingerencji w logikę biznesową. Obsługę plików wejściowych i wyjściowych zaimpmlementowano w **Input/Output** (io).

7 Uproszczony diagram klas UML



Rysunek 9: Uproszczony diagram klas UML

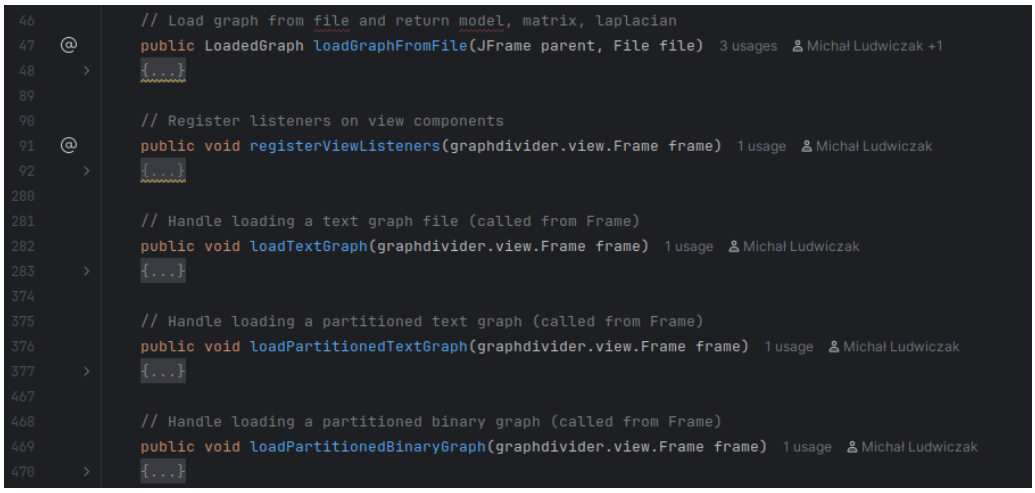
8 Najważniejsze klasy i funkcje w programie

1. **Frame**
   Klasa `graphdivider.view.Frame` odpowiada za główne okno aplikacji oraz integrację wszystkich paneli interfejsu użytkownika. Zarządza panelem narzędzi (`ToolPanel`), panelem podziału (`PartitionPanel`), panelem wizualizacji grafu (`Graph`) oraz paskiem menu (`MenuBar`). Udostępnia metody do aktualizacji interfejsu, obsługi zdarzeń oraz przekazywania poleceń do kontrolera.

```java
public Frame()  1 usage   Michał Ludwiczak +2
{
    setTitle("Graph Divider");

    // Setup menu bar
    menuBar = new MenuBar();
    setJMenuBar(menuBar);

    // Setup tool and partition panels (left side)
    toolPanel = new ToolPanel();
    partitionPanel = new PartitionPanel();
    Box leftBox = Box.createVerticalBox();
    leftBox.add(toolPanel);
    leftBox.add(Box.createVerticalStrut( height: 10));
    leftBox.add(partitionPanel);
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(leftBox, BorderLayout.WEST);

    // Setup graph panel with scrollbars (center)
    graphPanel = new Graph(toolPanel);
    JScrollPane scrollPane = new JScrollPane(graphPanel,
            JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
            JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    scrollPane.setVerticalScrollBar(new graphdivider.view.ui.ScrollBar(JScrollBar.VERTICAL));
    scrollPane.setHorizontalScrollBar(new graphdivider.view.ui.ScrollBar(JScrollBar.HORIZONTAL));
    scrollPane.setBorder(null);
    scrollPane.setPreferredSize(new Dimension( width: 1800,  height: 900));
    getContentPane().add(scrollPane, BorderLayout.CENTER);

    // Set partition panel to unknown at startup
    partitionPanel.setUnknown();
}
```

Rysunek 10: Konstruktor klasy Frame

2. **GraphController**
   Klasa `graphdivider.controller.GraphController` pełni rolę kontrolera w architekturze MVC. Odpowiada za obsługę logiki aplikacji: ładowanie grafów z plików, inicjalizację modelu, wywoływanie algorytmów podziału oraz aktualizację widoku. Rejestruje nasłuchiwacze zdarzeń dla interfejsu użytkownika i zarządza przepływem danych pomiędzy modelem a widokiem.



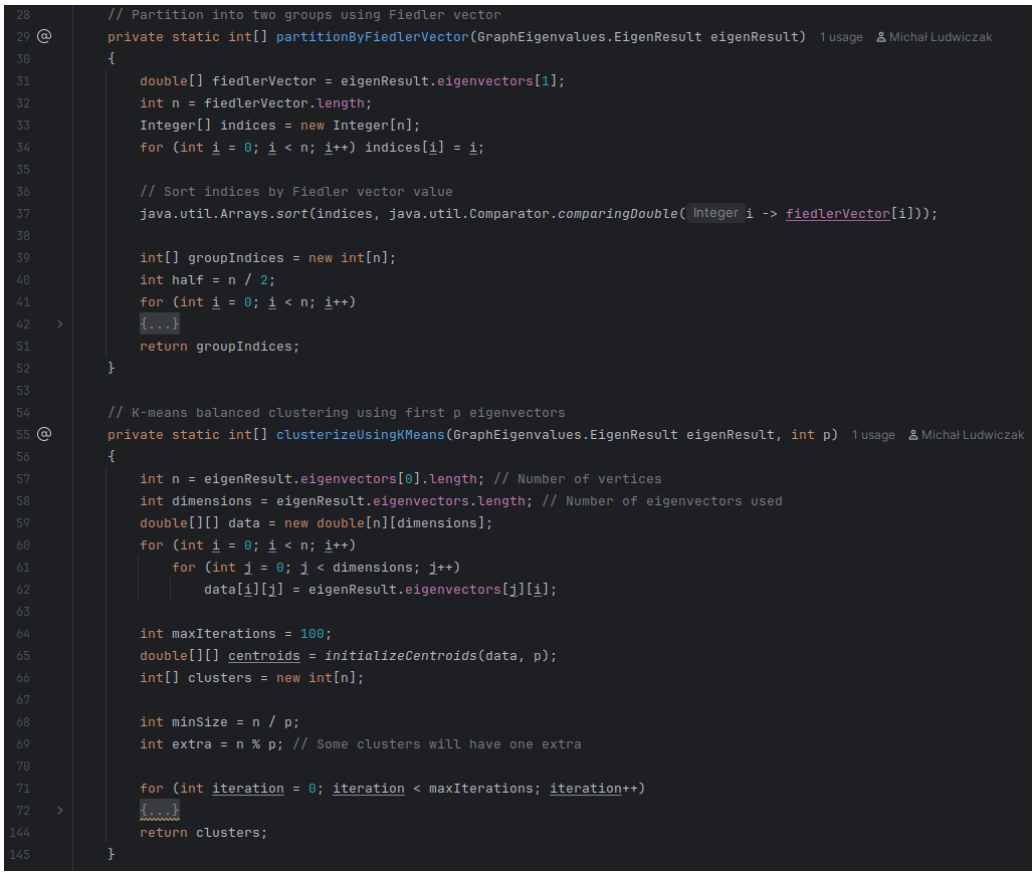
Rysunek 11: Interfejsy funkcji kontrolujących wczytywanie plików wejściowych (GraphController)

3. **GraphEigenvalues**
   Klasa `graphdivider.model.GraphEigenvalues` realizuje obliczanie wartości i wektorów własnych macierzy Laplace'a grafu. Kluczowa metoda `computeSmallestEigenpairs` wykorzystuje bibliotekę ARPACK do wydajnych obliczeń spektralnych, zwracając obiekt `EigenResult` z tablicą wartości własnych i odpowiadającymi im wektorami własnymi.

```
        String bmat = "I";   // Standard eigenvalue problem
        String which = "SM"; // Smallest magnitude

        double[] resid = new double[n];
        Arrays.fill(resid, val: 0.0); // Deterministic init
        double[] V = new double[n * ncv];
        int ldv = n;
        int[] iparam = new int[11];
        int[] ipntr = new int[11];
        double[] workd = new double[3 * n];
        int lworkl = 3 * ncv * (ncv + 2);
        double[] workl = new double[lworkl];

        // Setup iparam
        iparam[0] = 1;       // Exact shifts
        iparam[2] = maxIter; // Max iterations
        iparam[6] = 1;       // Mode 1: standard eigenproblem

        // Reverse communication loop
        while (ido.val != 99)
        {
            doubleW tolWrapper = new doubleW(tol);
            arpack.dsaupd(ido, bmat, n, which, p, tolWrapper, resid, ncv, V, ldv, iparam, ipntr, workd, workl, lworkl, info);

            if (ido.val == -1 || ido.val == 1)
            {
                double[] x = workd;
                double[] y = new double[n];
                int xOffset = ipntr[0] - 1;
                int yOffset = ipntr[1] - 1;

                // y = L * x
                for (int i = 0; i < n; i++) y[i] = 0.0;
                for (int i = 0; i < n; i++)
                {
                    for (int j = laplacian.getRowPtr()[i]; j < laplacian.getRowPtr()[i + 1]; j++)
                    {
                        y[i] += laplacian.getValues()[j] * x[xOffset + laplacian.getColInd()[j]];
                    }
                }
                System.arraycopy(y, srcPos: 0, workd, yOffset, n);
            }
        }
```

Rysunek 12: Fragment funkcji computeSmallestEigenpairs (argumenty + główna pętla, bez normalizacji)

4. **GraphClusterization**
   Klasa `graphdivider.model.GraphClusterization` odpowiada za podział grafu na części na podstawie wyników analizy spektralnej. Udostępnia metody do podziału według wektora Fiedlera (dla dwóch części) oraz zbalansowanego klasteryzowania k-średnich (dla większej liczby części). Zawiera także funkcje pomocnicze do obliczania marginesu i wypisywania wyników podziału.



```java
// Partition into two groups using Fiedler vector
private static int[] partitionByFiedlerVector(GraphEigenvalues.EigenResult eigenResult)  1 usage  Michał Ludwiczak
{
    double[] fiedlerVector = eigenResult.eigenvectors[1];
    int n = fiedlerVector.length;
    Integer[] indices = new Integer[n];
    for (int i = 0; i < n; i++) indices[i] = i;

    // Sort indices by Fiedler vector value
    java.util.Arrays.sort(indices, java.util.Comparator.comparingDouble( Integer i -> fiedlerVector[i]));

    int[] groupIndices = new int[n];
    int half = n / 2;
    for (int i = 0; i < n; i++)
    {...}
    return groupIndices;
}

// K-means balanced clustering using first p eigenvectors
private static int[] clusterizeUsingKMeans(GraphEigenvalues.EigenResult eigenResult, int p)  1 usage  Michał Ludwiczak
{
    int n = eigenResult.eigenvectors[0].length; // Number of vertices
    int dimensions = eigenResult.eigenvectors.length; // Number of eigenvectors used
    double[][] data = new double[n][dimensions];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < dimensions; j++)
            data[i][j] = eigenResult.eigenvectors[j][i];

    int maxIterations = 100;
    double[][] centroids = initializeCentroids(data, p);
    int[] clusters = new int[n];

    int minSize = n / p;
    int extra = n % p; // Some clusters will have one extra

    for (int iteration = 0; iteration < maxIterations; iteration++)
    {...}
    return clusters;
}
```

Rysunek 13: Funkcje odpowiadające za dzielenie grafu na podstawie obliczonych wartości wektorów własnych (względem wektora Fiedlera + zbalansowana klasteryzacja k-średnich

5. **Graph**
   Klasa `graphdivider.view.ui.Graph` odpowiada za wizualizację grafu w interfejsie graficznym. Pozwala na rysowanie wierzchołków i krawędzi, aktualizację kolorowania wierzchołków według przypisanych części oraz dynamiczne odświeżanie widoku po podziale grafu.

```java
// Panel for drawing the graph visualization
public final class Graph extends JPanel  6 usages  MichaelL200 +2
{
    // All vertices (as components)
    private final List<Vertex> vertices = new ArrayList<>();  6 usages
    // All edges (drawn manually)
    private final List<Edge> edges = new ArrayList<>();  7 usages
    // Map: vertex index -> Vertex component
    private java.util.Map<Integer, Vertex> vertexIndexToComponent;  7 usages
    // Reference to tool panel
    private final ToolPanel toolPanel;  2 usages
    // Cached preferred size
    private Dimension cachedPreferredSize = null;  7 usages

    // Setup panel, listen for theme changes
    public Graph(ToolPanel toolPanel)  1 usage  MichaelL200 +2
    {...}

    // Show graph from model, clear previous
    public void displayGraph(GraphModel model)  1 usage  MichaelL200 +1
    {
        clearGraph();
        setupVertices(model);
        setupEdges(model);
        updatePreferredSize();
        revalidate();
        repaint();
        toolPanel.setDivideButtonEnabled(true);
    }
```

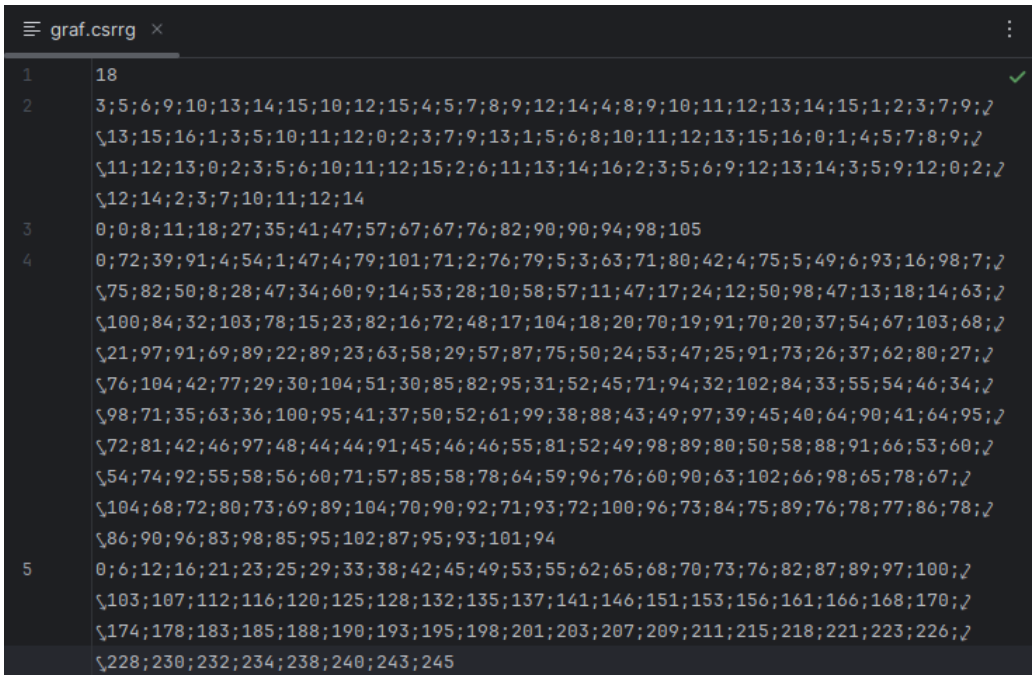
Rysunek 14: Fragment klasy Graph, funkcja displayGraph do wyświetlania grafu w panelu

9 Pliki wejściowe i wyjściowe dla trybu dzielenia

9.1 Plik wejściowy (`.csrrg`)

Format pliku składa się z pięciu sekcji zapisanych w kolejnych liniach:

1. Maksymalna liczba wierzchołków w dowolnym wierszu macierzy sąsiedztwa.
2. Lista sąsiadów wszystkich wierzchołków zapisana sekwencyjnie.
3. Wskaźniki (indeksy) na początki list sąsiedztwa dla poszczególnych wierzchołków.
4. Lista grup wierzchołków połączonych krawędziami (reprezentacja krawędzi).
5. Wskaźniki na początki grup węzłów z poprzedniej listy.



Rysunek 15: Przykładowy plik wejściowy (graph.csrrg)

9.2 Plik wyjściowy (`.csrrg2` / `.bin`)

Po przetworzeniu danych program generuje plik wyjściowy w jednym z dwóch formatów:

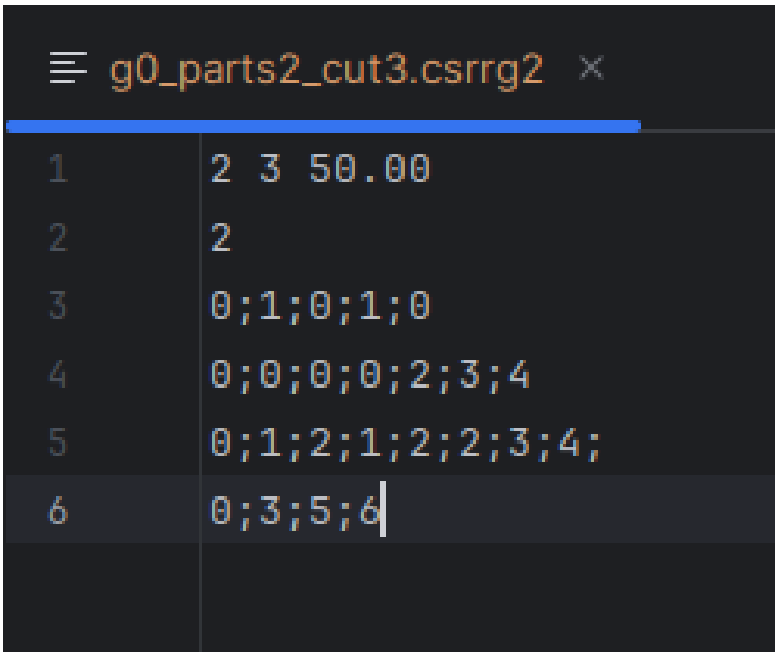
- `.csrrg2` — tekstowa forma pliku wyjściowego, wzorowana na formacie wejściowym .csrrg, ale zawierająca dodatkową linię nagłówkową z wynikiem działania programu.
- `.bin` — binarna wersja pliku tekstowego `.csrrg2`.

W pierwszej linii pliku wyjściowego zapisywany jest rezultat działania programu w formacie:

```
<liczba_części> <liczba_przecięć> <zachowany_margines>
```

Przykład:

```
3 14026 0,00
```



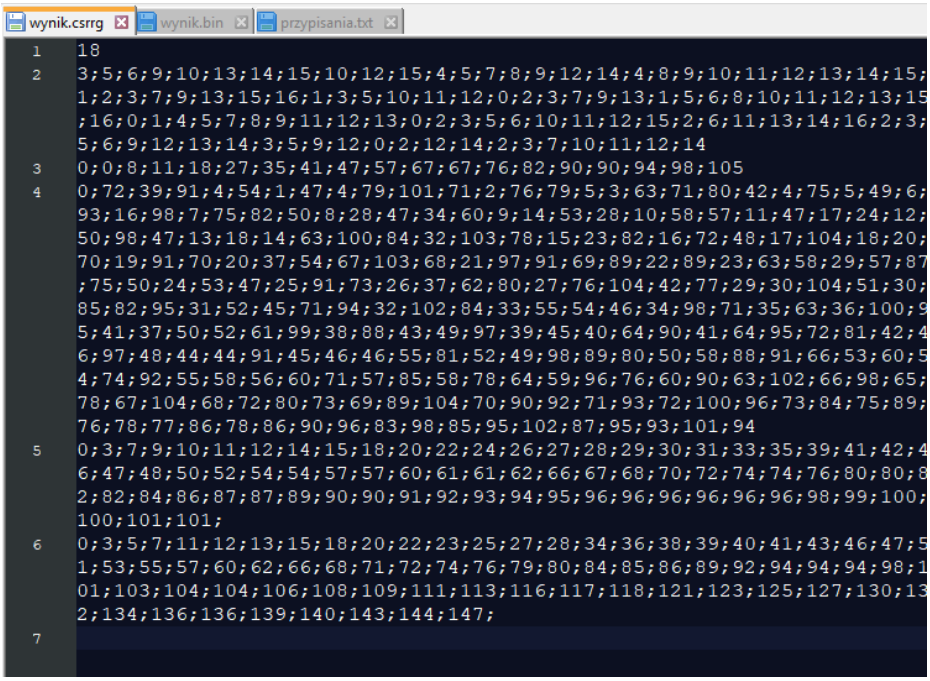
Rysunek 16: Przykładowy plik wyjściowy o domyślnej nazwie `g0_parts2_cut3.csrrg2`, powstały w wyniku podziału grafu `g0.csrrg` na 2 części z przecięciem 3 krawędzi

10 Pliki wejściowe dla trybu wyświetlenia podzielonego grafu

(pliki wyjściowe programu w C)

10.1 Plik wejściowy w formacie `.csrrg`

Plik przyjmuje postać taką jak plik wejściowy w formacie .csrrg z tym, że linijki po 5 linijce zawierają wierzchołki należące do danych kolejnych podgrup.



Rysunek 17: Przykładowy plik wejściowy dla trybu wyświetlenia

10.2 Plik wejściowy w formacie binarnym (`.bin`)

Zapis taki sam jak w formacie .csrrg, ale w formacie binarnym z wypisaniem ilości elementów w linijkach.

11 Obsługa błędów i wyjątków

W aplikacji zaimplementowano obsługę najważniejszych błędów i wyjątków, które mogą wystąpić podczas pracy programu. Dzięki temu użytkownik otrzymuje czytelne komunikaty o problemach, a aplikacja nie kończy działania w sposób niekontrolowany.

- **Błędy podczas wczytywania plików**
  Podczas próby wczytania grafu z pliku tekstowego lub binarnego obsługiwane są wyjątki związane z brakiem pliku, błędnym formatem lub błędami wejścia/wyjścia (`IOException`). W przypadku wykrycia błędu użytkownik otrzymuje komunikat w oknie dialogowym, a operacja jest przerywana.

- **Nieprawidłowe dane wejściowe**
  Program sprawdza poprawność parametrów podziału grafu (liczba części, margines). Jeżeli użytkownik wprowadzi nieprawidłowe wartości (np. liczbę części większą niż liczba wierzchołków, margines spoza zakresu), wyświetlany jest komunikat o błędzie i operacja nie jest wykonywana.

- **Błędy podczas obliczeń spektralnych**
  Podczas obliczania wartości własnych i wektorów własnych macierzy Laplace'a mogą wystąpić wyjątki związane z nieprawidłową macierzą lub brakiem zbieżności algorytmu ARPACK. Takie wyjątki są przechwytywane, a użytkownik informowany jest o niepowodzeniu operacji.

- **Błędy podczas zapisu wyników**
  W przypadku problemów z zapisem wyników do pliku (np. brak uprawnień, błąd dysku) obsługiwane są wyjątki wejścia/wyjścia. Użytkownik otrzymuje stosowny komunikat, a program nie kończy działania.

- **Błędy podczas wizualizacji**
  Jeżeli podczas rysowania grafu lub aktualizacji widoku wystąpi błąd (np. brak danych do wyświetlenia), aplikacja wyświetla komunikat i nie próbuje wykonać niepoprawnej operacji.

Wszystkie powyższe sytuacje są obsługiwane w kodzie poprzez instrukcje `try-catch`, a użytkownik jest informowany o problemach za pomocą okien dialogowych lub komunikatów w interfejsie graficznym.

12 Przykłady użycia

12.1 Wczytywanie grafu z pliku tekstowego

Aby wczytać graf z pliku tekstowego (np. `src/main/resources/graphs/graf.csrrg`), wybierz opcję `Load File -> Graph (Text)...` z paska menu. Program odczyta dane grafu w formacie CSR i wyświetli jego strukturę w głównym obszarze aplikacji.

12.2 Podział grafu na `x` części z marginesem `y%`

Po załadowaniu grafu ustaw w panelu narzędziowym liczbę części (`x`) oraz margines tolerancji wielkości części (`y%`). Następnie kliknij przycisk `Divide Graph`. Program dokona podziału grafu na `x` części, minimalizując liczbę przeciętych krawędzi i zachowując zadany margines liczebności wierzchołków w każdej części.

12.3 Zapisanie wyniku do pliku tekstowego

Po zakończeniu podziału wybierz opcję `Save File -> Partitioned Graph (Text)...` z menu. Wynik zostanie zapisany w katalogu `src/main/resources/divided_graphs/` w formacie `.csrrg2`, zawierającym dodatkową linię nagłówkową z informacją o wyniku działania programu oraz przypisania wierzchołków do części.

12.4 Zapisanie wyniku do pliku binarnego

Aby zapisać wynik w formacie binarnym, wybierz opcję `ZSave File -> Partitioned Graph (Binary` z menu. Plik zostanie zapisany w katalogu `src/main/resources/divided_graphs/` z rozszerzeniem `.bin`, co umożliwia szybkie wczytanie podzielonego grafu w przyszłości.

12.5 Przykładowe pliki

13 Wymagania niefunkcjonalne

Poniżej przedstawiono kluczowe wymagania niefunkcjonalne dla aplikacji:

- **Wydajność:** Czas podziału grafu powinien być jak najkrótszy dla danej wielkości grafu. Dla grafów zawierających do 1000 wierzchołków, czas przetwarzania nie powinien przekraczać minuty na standardowym sprzęcie klasy PC.
- **Użyteczność:** Interfejs użytkownika powinien być intuicyjny i zgodny z ogólnie przyjętymi standardami projektowania GUI.
- **Skalowalność:** Aplikacja powinna umożliwiać obsługę grafów o różnej wielkości, od małych do dużych.
- **Przenośność:** Aplikacja powinna działać poprawnie na różnych systemach operacyjnych wspierających środowisko Java.
- **Utrzymywalność:** Kod źródłowy aplikacji powinien być czytelny i dobrze udokumentowany, aby ułatwić przyszłe modyfikacje i rozwój.

14 Podsumowanie

Program umożliwia szybki i efektywny podział grafów dzięki wykorzystaniu biblioteki ARPACK, która pozwala na błyskawiczne obliczanie najmniejszych par własnych macierzy Laplace'a nawet dla bardzo dużych grafów. Zastosowanie zbalansowanej klasteryzacji k-średnich (k-means) sprawia, że margines nierówności podziału jest minimalny, co przekłada się na wysoką jakość wyników. Intuicyjny interfejs graficzny umożliwia łatwe wprowadzanie parametrów podziału oraz dynamiczne dostosowywanie zakresów kontrolek w zależności od wczytanego grafu. Wizualizacja grafu jest czytelna i pozwala na szybkie rozróżnienie poszczególnych części dzięki kolorowaniu wierzchołków według przypisania do części. Całość rozwiązania zapewnia wysoką wydajność, wygodę użytkowania oraz przejrzystość prezentowanych wyników.

Literatura

[1] Leonid Zhukov, *Lecture 7. Graph partitioning algorithms.*, YouTube, 24 luty 2021, Dostępny na 11 maja 2025 w: `https://youtu.be/zZae_C2BU_4`

[2] *Laplacian matrix*, Wikipedia, Dostępne na 11 maja 2025 w: `https://en.wikipedia.org/wiki/Laplacian_matrix`

[3] *Algorytm centroidów*, Wikipedia, Dostępne na 11 maja 2025 w: `https://pl.wikipedia.org/wiki/Algorytm_centroid%C3%B3w`