Podział systemu na moduły i wzorce projektowe

Emilia Urbanek i Mikołaj Frąckowiak

26.05.2025

1 Struktura projektu

Projekt jest zorganizowany w następujący sposób:

1.1 Główne katalogi źródłowe

- `org.example` - Główny pakiet aplikacji
- `algorithm` - Implementacje algorytmów
  - `GraphPartitioner` - Główna klasa odpowiedzialna za logikę podziału grafu
- `GraphVisualisation` - Komponenty do wizualizacji grafów
  - `GraphPostPartitionPane` - Panel wyświetlający graf po podziale
  - `GraphPrePartitionPanel` - Panel wyświetlający graf przed podziałem
- `io` - Operacje wejścia/wyjścia
  - `GraphLoaderBin` - Ładowanie grafów z plików binarnych
  - `GraphLoaderCsrrg` - Ładowanie grafów w formacie csrrg
- `model` - Główne klasy modelu danych
  - `Graph` - Reprezentacja struktury grafu
  - `GraphException` - Wyjątki związane z operacjami na grafach
  - `PartitionResult` - Wynik operacji podziału grafu
- `test_data` - Dane testowe
- `Main` - Główna klasa testowa
- `MainFrame` - Główna rama aplikacji testowej

1.2 Interfejs użytkownika

- `DetailsUI` - Okno ze szczegółowymi informacjami(listą sąsiedztwa grafu)
  - `DetailsUI` - Główna klasa okna
  - `DetailsUI.form` - Plik definicji formularza
- `MainUI` - Główne okno aplikacji
  - `MainUI` - Główna klasa okna
  - `MainUI.form` - Plik definicji formularza
- `PartitionUI` - Okno wizualizacji grafu po podziale
  - `PartitionUI` - Główna klasa okna
  - `PartitionUI.form` - Plik definicji formularza

2 Zastosowane wzorce projektowe

W kodzie można zidentyfikować kilka wzorców projektowych, które wspierają modularność, elastyczność oraz czytelność rozwiązania. Oto najważniejsze z nich:

- **Strategy (Strategia)** – Wzorzec ten jest widoczny w implementacji różnych algorytmów w klasie `GraphPartitioner` (np. `dfsMarkComponents`, `dijkstra`). Każda z tych metod reprezentuje inną strategię przetwarzania grafu, którą można wykorzystać w zależności od potrzeb.
- **Observer (Obserwator)** – Wzorzec ten znajduje zastosowanie w klasie `MainUI`, gdzie komponenty GUI (np. `buttonPodziel`) pełnią rolę obserwatorów zdarzeń użytkownika. Reagują one na akcje poprzez przypisane *listenery*.
- **Memento (Pamiątka)** – Wzorzec ten jest realizowany za pomocą zmiennych `savedPrePartitionPositions` oraz `savedPostPartitionPositions` w klasie `MainUI`, które przechowują stan pozycji wierzchołków przed i po podziale grafu.
- **Composite (Kompozyt)** – Wzorzec ten można zauważyć w strukturze interfejsu graficznego, gdzie komponenty takie jak `JPanel` zawierają inne komponenty (np. `JLabel`, `JSpinner`) i traktowane są jako pojedyncze jednostki złożone.
- **Factory Method (Metoda wytwórcza)** – Wzorzec ten występuje w metodzie `performPartitioning()` klasy `PartitionResult`, gdzie tworzone są obiekty `PartitionInfo` reprezentujące różne warianty wyników podziału.
- **Decorator (Dekorator)** – Wzorzec ten jest wykorzystywany w GUI poprzez opakowywanie komponentów (np. `JSpinner` z `SpinnerNumberModel`) w celu nadania im dodatkowych właściwości lub zachowań.
- **State (Stan)** – Wzorzec ten jest obecny w klasach `Graph` oraz `PartitionResult`, gdzie stan grafu (np. składowe, informacje o podziałach) determinuje dostępność i przebieg poszczególnych operacji.

Instrukcja użytkownika aplikacji do podziału grafu (Java Swing)

Emilia Urbanek i Mikołaj Frąckowiak

26.05.2025

1 Wstęp

Niniejsza instrukcja opisuje sposób obsługi graficznego interfejsu użytkownika aplikacji do podziału grafu, zaimplementowanego w technologii Java Swing. Aplikacja pozwala na wczytywanie grafów, definiowanie parametrów podziału oraz wizualizację wyników. Celem aplikacji jest dokonanie określonej liczby podziałów w taki sposób, aby przy każdym podziale liczba wierzchołków otrzymanych dwóch podgrafów była możliwie równa (z dopuszczalnym marginesem różnicy) oraz aby liczba przeciętych krawędzi była minimalna.

2 Parametry używane przy podziale grafu

N– Liczba przecięć grafu. Parametr ten określa,ile razy należy podzielić graf- np. dla wartości 1 zostanie podzielony jeden raz-więc powstaną 2 części. Domyślnie wartość N to 2, ale użytkownik może podać inną liczbę.
M – Maksymalny procentowy margines różnicy liczby wierzchołków między częściami. Określa dopuszczalną różnicę w liczbie wierzchołków między podzielonymi częściami. Domyślnie wartość M to 10%. Oznacza to, że różnica w liczbie wierzchołków w każdej części nie może przekroczyć 10%.

Odczyt z pliku- użytkownik wybiera w pasku menu, z którego pliku ma być odczytany graf(.csrrg lub .bin)

3 Obsługa danych wejściowych

Dane wejściowe mogą być wczytywane z pliku o rozszerzeniu `.csrrg`, zawierającego reprezentację grafu w formacie CSR.
Opis formatu pliku .csrrg:
Plik wejściowy zawiera reprezentację grafu w formacie CSR (Compressed Sparse Row), który umożliwia efektywne przechowywanie i przetwarzanie grafów rzadkich. Struktura pliku składa się z następujących sekcji:

- **Sekcja 1: Rozmiar grafu** - Pierwsza linia zawiera pojedynczą liczbę całkowitą n, określającą maksymalny wymiar grafu. Liczba określa szerokość oraz wysokość grafu(maksymalną w każdym wierszu/kolumnie).
- **Sekcja 2: Układ wierzchołków** - Kolejne liczby określają numer kolumny, w której znajduje się dany węzeł
- **Sekcja 3: Rozkład wierszy** - Określa kolejne nakładające się ze sobą pary wyznaczające zakres nastepnych wierszy. Np. 0,8,11-pierwszy wiersz obejmuje elementy z sekcji 2 od 0 do 8(zaczynając liczenie od liczby+1), a kolejny wiersz ma elementy od 9 do 11.
- **Sekcja 4: Lista grup połączonych wierzchołków** - Zawiera listę grup wierzchołków, które należą do wspólnych komponentów grafu. Każda grupa jest reprezentowana jako zbiór wierzchołków, które są wzajemnie połączone bezpośrednimi krawędziami.

- **Sekcja 5: Wskaźniki na pierwsze węzły w grupach** - Wskazuje, gdzie zaczynają się kolejne grupy połączonych węzłów opisane w sekcji 4. Może występowac wielokrotnie, jeśli plik zawiera więcej niż jeden graf.

Dane wejściowe mogą być również wczytywane z pliku o rozszerzeniu `.bin`, zawierającego reprezentację grafu w formacie binarnym.
Opis formatu pliku .bin:
Format binarny jest zbliżony do wewnętrznej reprezentacji danych grafu. Wszystkie liczby są zapisywane jako 32-bitowe liczby całkowite bez znaku (`uint32_t`) w kolejności **little-endian**. Listy są zapisywane poprzez najpierw zapisanie długości listy jako `uint32_t`, a następnie kolejno wszystkich elementów listy jako `uint32_t`.

Struktura pliku binarnego:

1. `max_vertices_in_row` (`uint32_t`)
2. `vertices_by_rows`:
   - `vertex_count` (`uint32_t`) - długość listy
   - Następnie `vertex_count` elementów listy (`uint32_t` każdy)
3. `row_indexes`:
   - `row_count` (`uint32_t`) - długość listy
   - Następnie `row_count` elementów listy (`uint32_t` każdy)
4. `graph_count` (`uint32_t`) - liczba grafów w kontenerze
5. Dla każdego z `graph_count` grafów:
   - `vertex_count` (`uint32_t`) - liczba wierzchołków w tym grafie
   - `edge_count` (`uint32_t`) - liczba krawędzi w tym grafie
   - `adjacencies`:
     - Długość listy (`uint32_t`), równa `graph->adjacency_indexes[graph->vertex_count]`
     - Następnie elementy listy `graph->adjacencies` (`uint32_t` każdy)
   - `adjacency_indexes`:
     - Długość listy (`uint32_t`), równa `graph->vertex_count + 1`
     - Następnie elementy listy `graph->adjacency_indexes` (`uint32_t` każdy)

Przykład struktury danych dla grafu z 3 wierzchołkami i 2 krawędziami (0-1, 1-2): `vertex_count` = 3, `edge_count` = 2 `adjacencies` = {1, 0, 2, 1} `adjacency_indexes` = {0, 1, 3, 4} (Wierzchołek 0 sąsiaduje z 1; wierzchołek 1 sąsiaduje z 0 i 2; wierzchołek 2 sąsiaduje z 1)

Zapis binarny tego grafu (fragment dotyczący pojedynczego grafu):

```
03 00 00 00  (vertex_count = 3)
02 00 00 00  (edge_count = 2)
04 00 00 00  (length of adjacencies = 4)
01 00 00 00  (adjacencies[0] = 1)
00 00 00 00  (adjacencies[1] = 0)
02 00 00 00  (adjacencies[2] = 2)
01 00 00 00  (adjacencies[3] = 1)
04 00 00 00  (length of adjacency_indexes = 4)
00 00 00 00  (adjacency_indexes[0] = 0)
01 00 00 00  (adjacency_indexes[1] = 1)
03 00 00 00  (adjacency_indexes[2] = 3)
04 00 00 00  (adjacency_indexes[3] = 4)
```

4 Algorytm podziału grafu

Proces podziału grafu realizowany jest w kilku krokach:

4.1 Wyznaczenie wierzchołka centralnego

Na potrzeby efektywnego podziału grafu wybierany jest wierzchołek centralny. W tym celu uruchamiany jest algorytm Dijkstry z losowego wierzchołka. Następnie jako punkt startowy przeszukiwania wybierany jest wierzchołek najbardziej oddalony od punktu początkowego, co pozwala objąć większą część grafu w pierwszej grupie.

4.2 Podział na dwie grupy

Podział wierzchołków na dwie grupy realizowany jest z wykorzystaniem algorytmu DFS (Depth-First Search), który działa od wyznaczonego wierzchołka centralnego. Liczba wierzchołków w każdej grupie jest monitorowana, aby zapewnić możliwie równy rozkład (z tolerancją różnicy jednego wierzchołka).

4.3 Sprawdzenie spójności

Po zakończeniu podziału każda z grup poddawana jest niezależnemu sprawdzeniu spójności, również z wykorzystaniem algorytmu DFS. Weryfikowana jest liczba wierzchołków odwiedzonych podczas przeszukiwania – musi ona odpowiadać liczbie elementów w danej grupie.

4.4 Postępowanie z niespójną drugą grupą

Gdy po podziale okaże się, że druga grupa nie jest spójna, algorytm wykonuje następujące kroki:

1. **Identyfikacja komponentów spójnych**:
   - Dla niespójnej grupy drugiej wyznaczane są wszystkie jej składowe spójne za pomocą algorytmu DFS
   - Każda składowa otrzymuje tymczasowy identyfikator
2. **Wybór głównej składowej**:
   - Wybierana jest największa składowa spójna (o największej liczbie wierzchołków)
   - Pozostałe składowe są oznaczane do przeniesienia
3. **Przenoszenie wierzchołków**:
   - Wszystkie wierzchołki z mniejszych składowych są przenoszone do pierwszej grupy
   - W grupie drugiej pozostają tylko wierzchołki z głównej składowej
4. **Weryfikacja warunków**:
   - Sprawdzana jest spójność nowo utworzonej grupy pierwszej
   - Weryfikowana jest różnica liczby wierzchołków między grupami

$$|V_1| - |V_2| \leq \text{margin} \tag{1}$$

   - Jeśli warunki nie są spełnione, algorytm wraca do etapu podziału z nowymi grupami
5. **Obsługa przypadków skrajnych**:
   - Gdy przeniesienie wierzchołków narusza spójność grupy pierwszej, wybierany jest alternatywny podział
   - W przypadku niemożności spełnienia warunków, algorytm może:
     - Próbować podziału innej składowej spójnej
     - Zwrócić informację o niemożności podziału

4.5 Generowanie podgrafów

Dla każdej z dwóch grup tworzony jest osobny podgraf. Dane są przetwarzane do wewnętrznej reprezentacji listy sąsiedztwa, uwzględniając jedynie sąsiadów należących do tej samej grupy. W efekcie uzyskiwane są dwa spójne podgrafy, gotowe do zapisania w plikach wyjściowych.

5 Uruchamianie programu

Program można uruchomić poleceniem "run" po odpowiedniej konfiguracji pliku "Main.java". Po uruchomieniu pojawi się główne okno aplikacji.

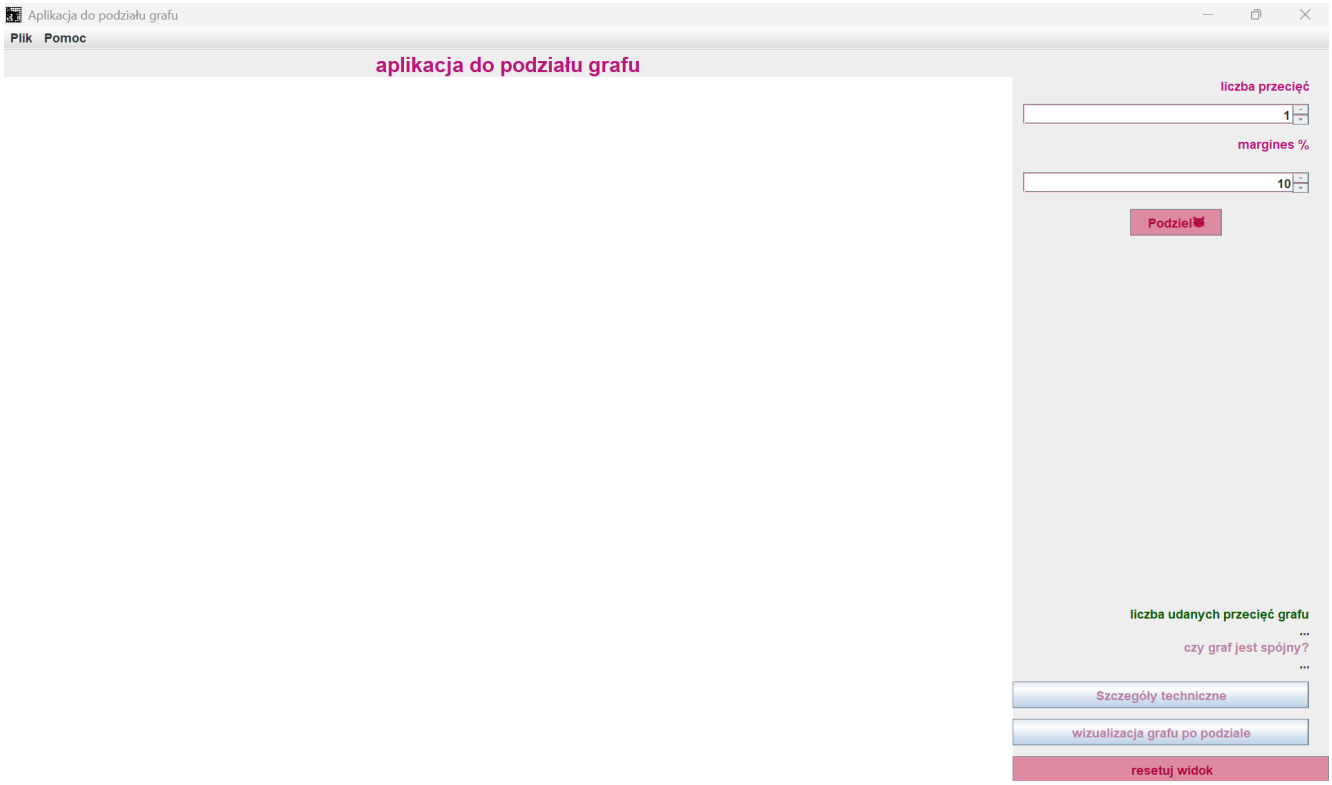

Figure 1: Główne okno aplikacji

6 Opis interfejsu strony głównej

Interfejs składa się z trzech głównych części:

6.1 Pasek menu

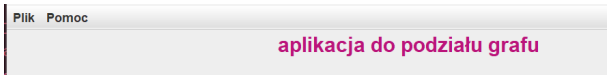

Figure 2: Pasek menu

Menu zawiera następujące opcje:

- `Plik > Otwórz tekstowy (.csrrg)` – wczytanie grafu z pliku tekstowego.
- `Plik > Otwórz binarny (.bin)` – wczytanie grafu z pliku binarnego.

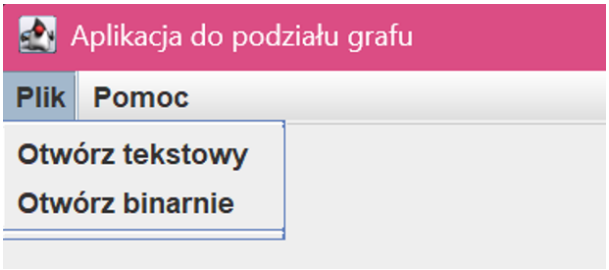

Figure 3: opcja Plik

Po wybraniu jednej z tych opcji wczytywania pliku pojawi się okno do zaimportowania pliku z systemu.

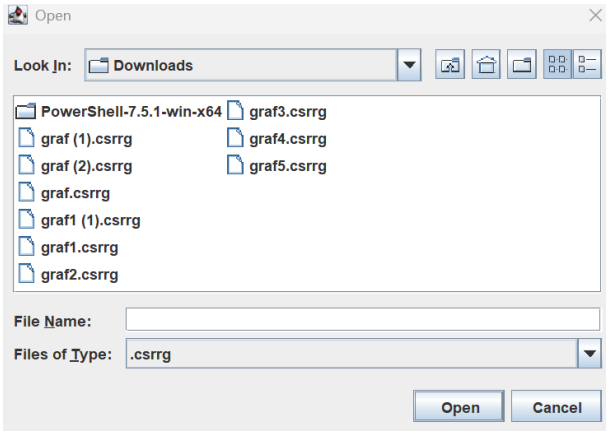

Figure 4: import pliku

Po wczytaniu pliku wyświetlany jest komunikat:

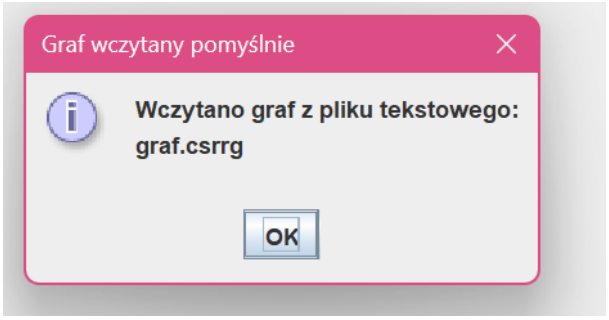

Figure 5: wczytano plik

Pomoc zawiera następujące opcje:

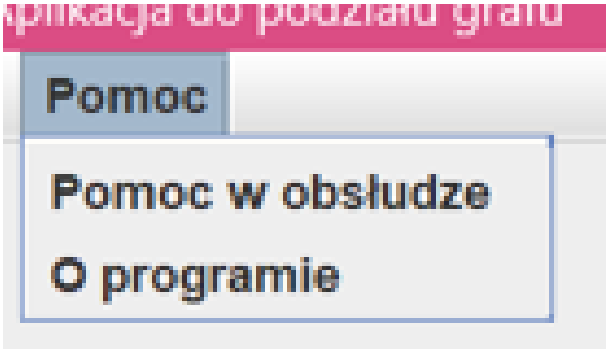

Figure 6: opcja Pomoc

- `Pomoc >Pomoc w obsłudze` – informacje o instrukcji obsługi. W przypadku wybrania opcji "Pomoc w obsłudze" wyświetli się następujący komunikat:

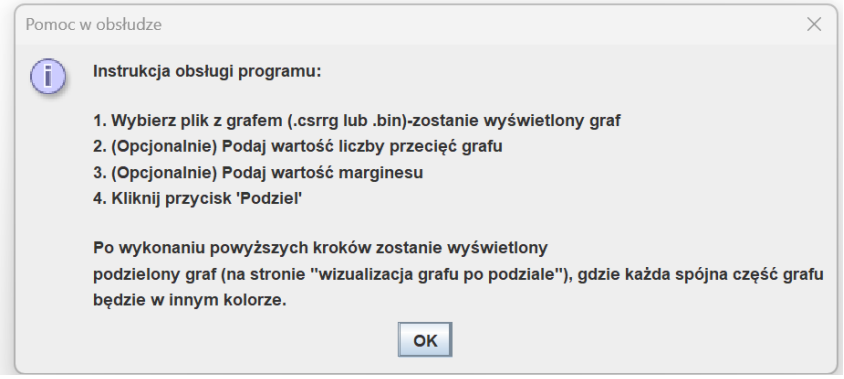

Figure 7: Pomoc w obsłudze

- `Pomoc > O programie` – informacje o sposobie działania aplikacji. W przypadku wybrania opcji "O programie" wyświetli się następujący komunikat:

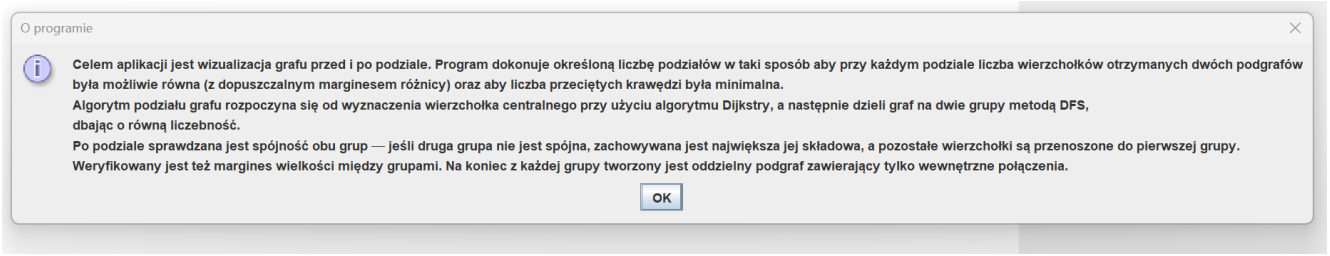

Figure 8: O programie

6.2 Panel graficzny grafu

W głównym obszarze okna znajduje się komponent Swing odpowiedzialny za rysowanie grafu. Po wczytaniu pliku graf wyświetlany jest jako zbiór węzłów i krawędzi:

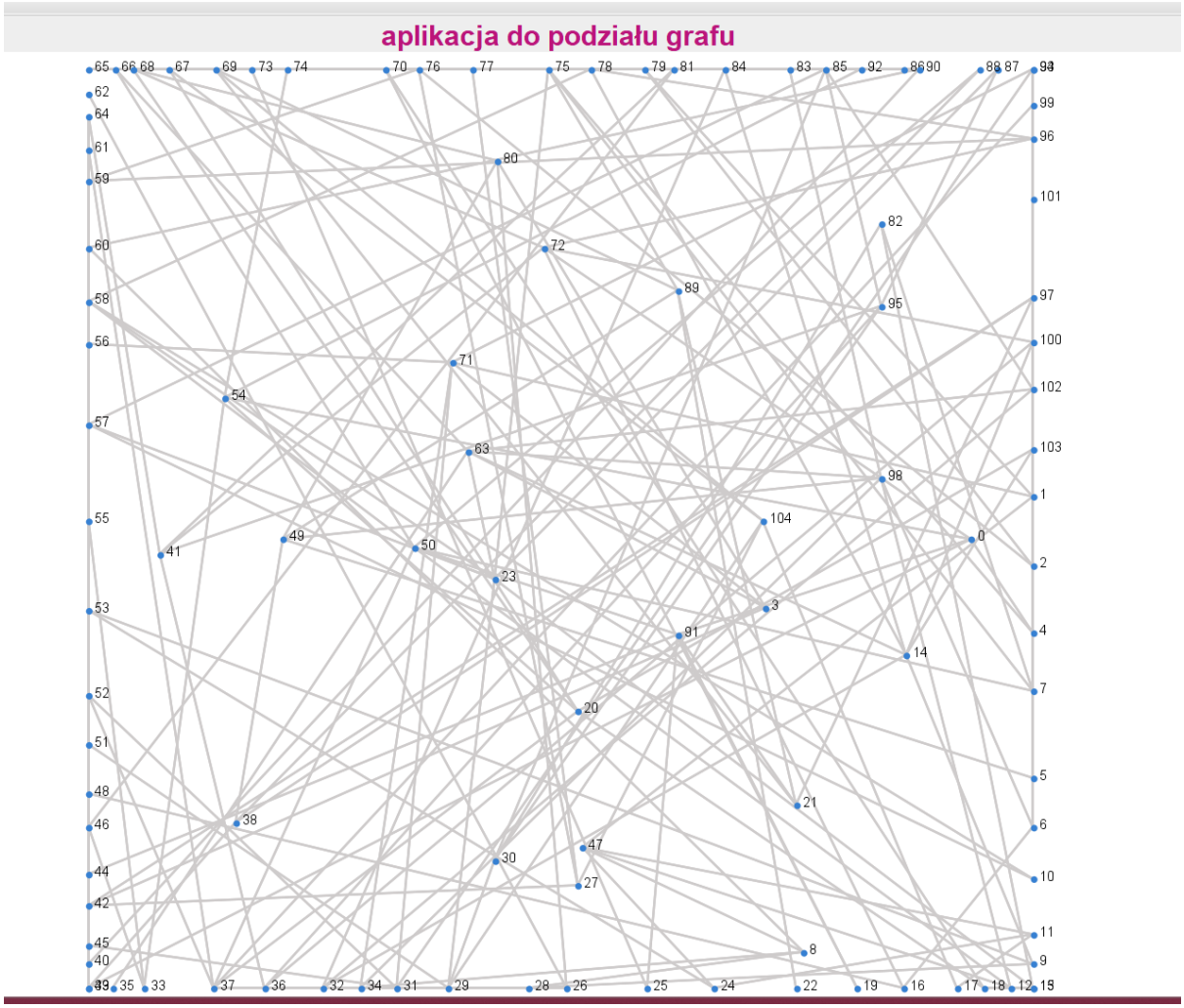

Figure 9: Graf przed podziałem

– Węzły rysowane są jako kółka z etykietami.
– Krawędzie łączą węzły liniami.
– Po podziale grafu fragmenty spójne grafu rysowane są w odrębnych kolorach.

Graf po podziale jest rysowany na odrębnej stronie dostępnej z poziomu głównej kliknięciem przycisku "wizualizacja grafu po podziale".

6.3 Panel narzędziowy

Po prawej stronie okna, w górnym rogu umieszczono panel narzędziowy, zawierający:

* Pole tekstowe `Liczba przecięć` – ustawienie liczby przecięć N (domyślnie 1).
* Pole tekstowe `Margines (%)` – ustawienie procentowego marginesu P (domyślnie 10%).
* Przycisk `Podziel` – uruchomienie procesu podziału grafu z zadanymi parametrami.
* Pole `czy graf jest spójny`– czerwone-gdy nie, zielone gdy spójny.
* Pole `liczba udanych przecięć grafu`-liczba przecięć grafu zakończonych sukcesem-odpowiada liczbe częsci grafu-1.
* Przycisk `Szczegóły techniczne` – pozwala zobaczyć listę sąsiedztwa grafu przed i po podziale.

* Przycisk `Wizualizacja grafu po podziale` – pozwala przejść do wizualizacji grafu po podziale.
* Przycisk `Resetuj widok` – resetuje wybrany plik z grafem, czyści dane dotyczącego tego grafu(czyli usuwa listę sąsiedztwa i narysowane grafy) oraz ustawia parametry z powrotem na ich domyślne wartości. Po klkinięciu inne przyciski zostają zdezaktywowane do czasu wybrania grafu.

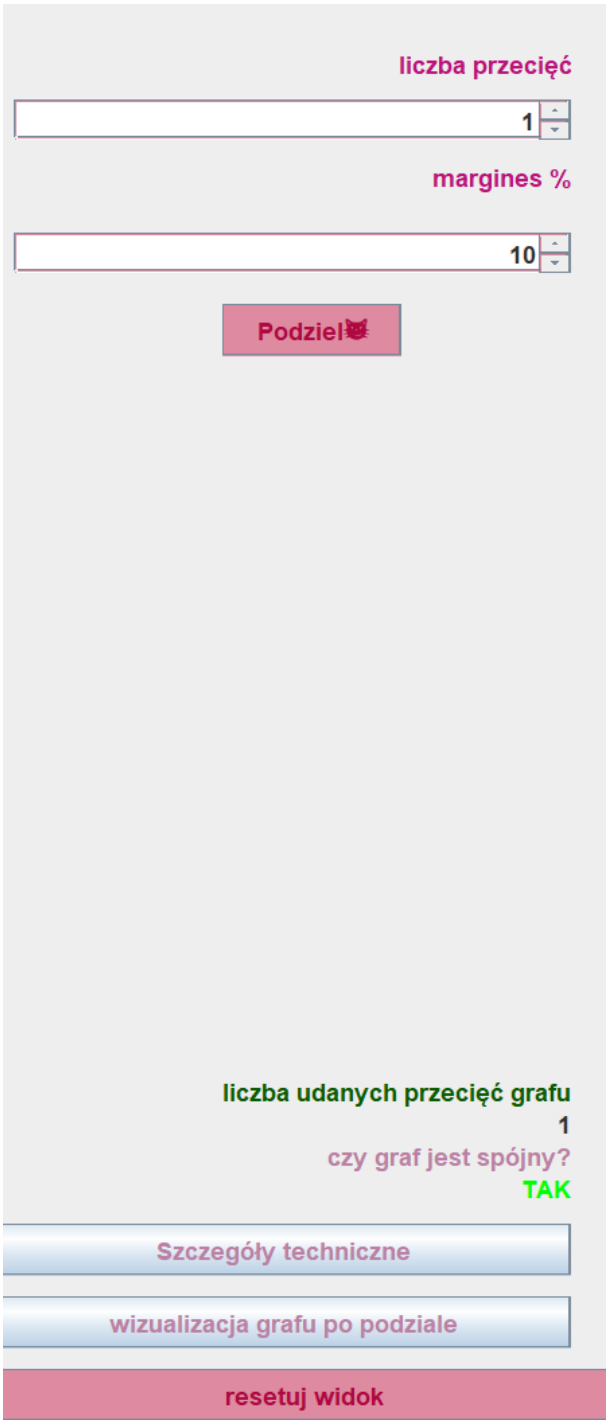

Figure 10: panel narzędziowy

7 Interfejs strony Szczegóły techniczne

Na stronie przedstawiona jest lista sąsiedztwa wczytanego grafu przed i po podziale. Dodatkowo, dzięki komponentowi do przesuwania, można przejść w dół listy, w przypadku gdy nie mieści się na stronie.

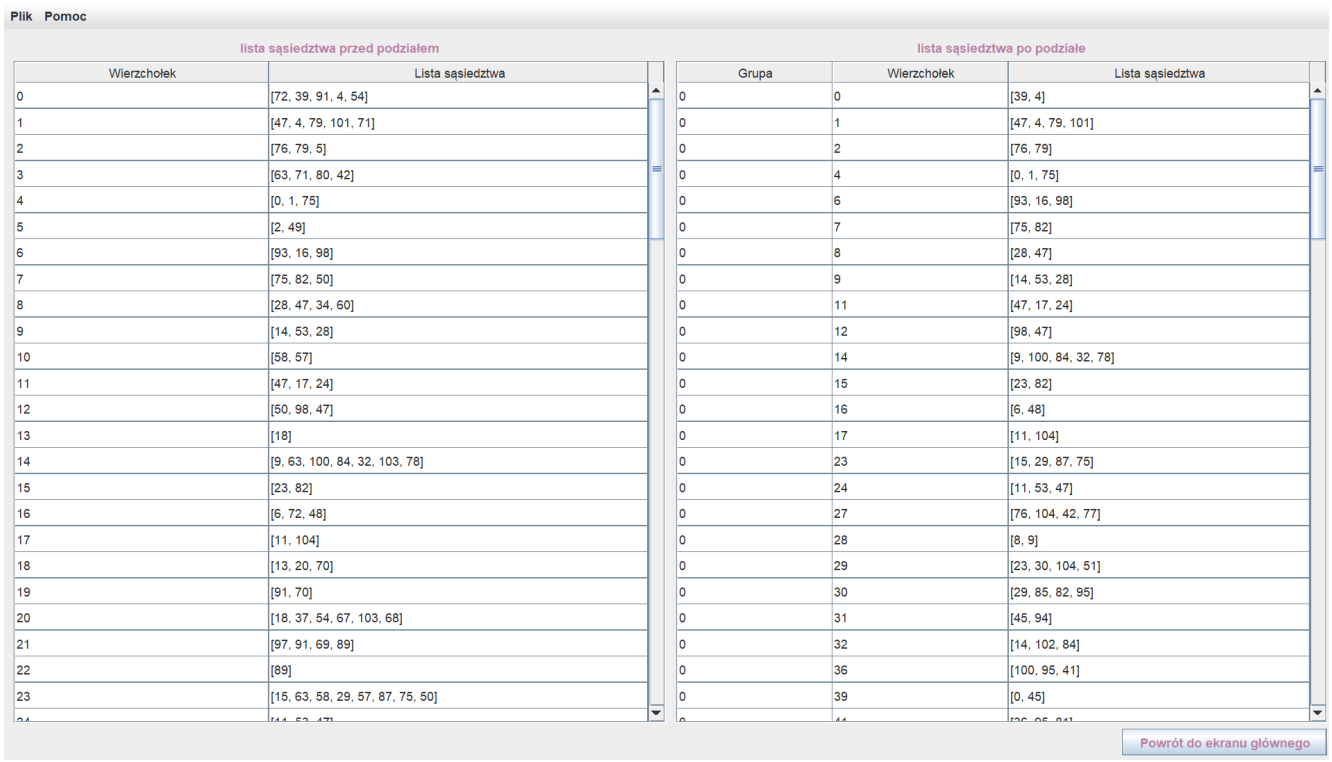

Figure 11: Lista sąsiedztwa

Zamieszczono również przycisk "Powrót do ekranu głównego", który umożliwia powrót do wcześniejszej strony.

8 Strona wizualizacja grafu po podziale

Na stronie przedstawiona jest wizualizacja grafu po podziale. Po podziale grafu fragmenty spójne grafu rysowane są w odrębnych kolorach. Została zaimplementowana dodatkowa funkcjonalność-śledzenie przybliżenia i aktualizowana na bieżąco liczba aktualnie widocznych wierzchołków zależnie od zoomu/przesunięcia. Dzięki temu można porównać wielkości danych części grafów po podziale.

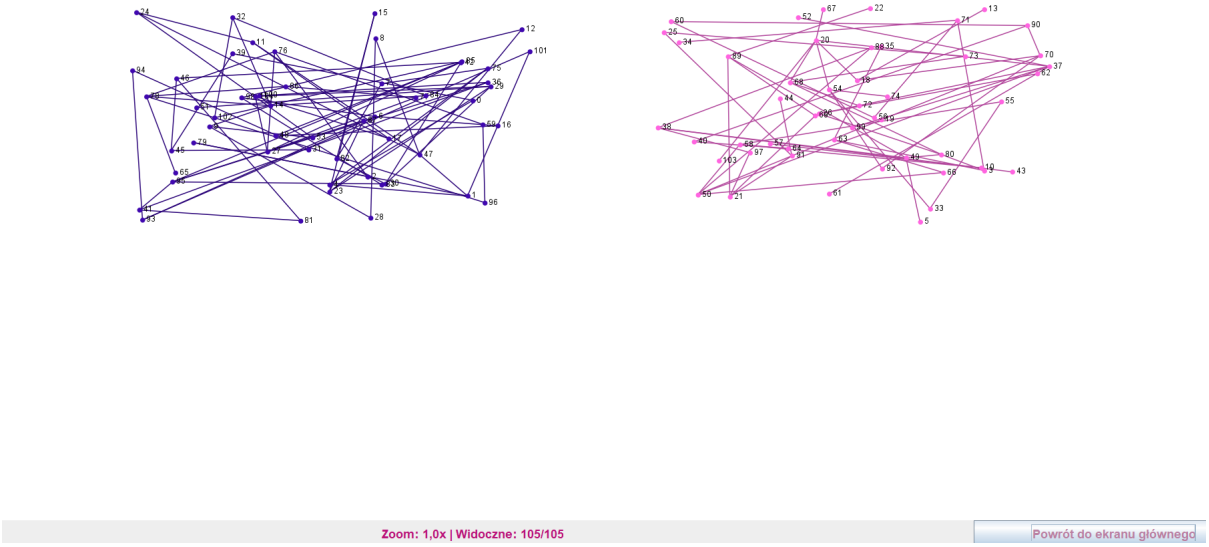

Figure 12: Graf po podziale

9 Obsługa błędów

W przypadkach:

- Nie wczytano grafu,
- Nieosiągalność grafu (niespełnienie warunków podziału),

aplikacja wyświetla okno dialogowe z opisem problemu i nie zmienia stanu głównego panelu.

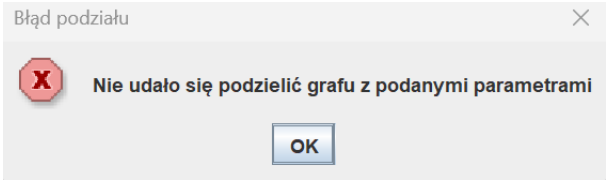

Figure 13: Komunikat o błędzie

10 Podsumowanie

Interfejs Swing zapewnia użytkownikowi intuicyjną obsługę: wczytywanie, wizualizację, definiowanie parametrów oraz wyświetlenie dodatkowych informacji o grafie. Modułowa budowa ułatwia dalszy rozwój i dostosowanie do zmieniających się wymagań.

Dokumentacja końcowa aplikacji do podziału grafu (Java Swing)

Emilia Urbanek i Mikołaj Frąckowiak

26.05.2025

1 Wstęp

Niniejsza dokumentacja opisuje aplikację do podziału grafu, zaimplementowaną w technologii Java Swing. Aplikacja pozwala na wczytywanie grafów, definiowanie parametrów podziału oraz wizualizację wyników. Celem aplikacji jest dokonanie określonej liczby podziałów w taki sposób, aby przy każdym podziale liczba wierzchołków otrzymanych dwóch podgrafów była możliwie równa (z dopuszczalnym marginesem różnicy) oraz aby liczba przeciętych krawędzi była minimalna.

2 Struktura projektu oraz diagramy

Projekt w języku Java został podzielony na logiczne pakiety i klasy, zgodnie z zasadami dobrej organizacji kodu oraz separacji odpowiedzialności. W skład projektu wchodzą następujące główne katalogi źródłowe:

2.1 1.1 Główne katalogi źródłowe

- **org.example** – Główny pakiet aplikacji uruchamiającej i testowej.
- **algorithm** – Pakiet zawierający implementację algorytmu podziału grafu.
  - `GraphPartitioner` – Główna klasa odpowiedzialna za logikę dzielenia grafu, implementująca algorytmy DFS i Dijkstry oraz balansowanie grup.

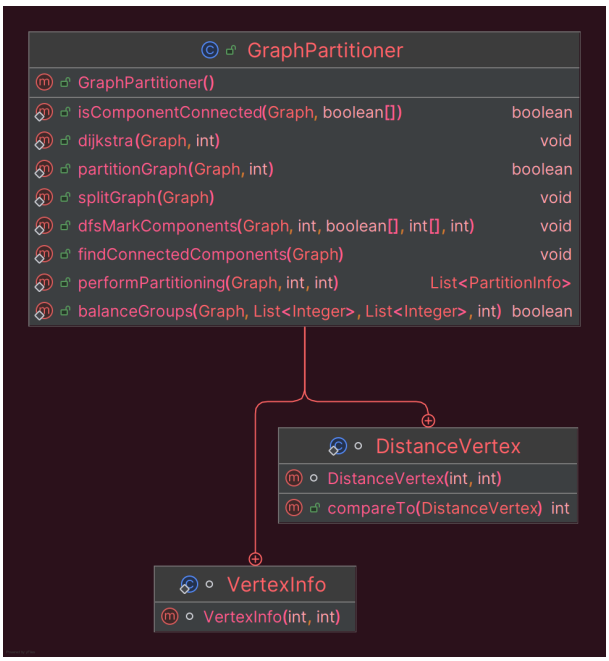

Figure 1: diagram algorithm

Najważniejsze metody:

- `dfsMarkComponents(Graph graph, int startVertex, boolean[] visited, int[] component, int currentComponent)`
  Wykonuje iteracyjną wersję algorytmu DFS i oznacza wszystkie wierzchołki należące do jednej składowej spójnej.
- `findConnectedComponents(Graph graph)`
  Znajduje wszystkie spójne składowe grafu i zapisuje ich liczbę oraz przynależność wierzchołków do odpowiednich komponentów.
- `dijkstra(Graph graph, int start)`
  Oblicza maksymalną odległość od danego wierzchołka do pozostałych w jego składowej spójnej przy użyciu algorytmu Dijkstry, zakładając wagę krawędzi równą 1.
- `isComponentConnected(Graph graph, boolean[] inComponent)`
  Sprawdza, czy podzbiór wierzchołków stanowi spójną składową.
- `balanceGroups(Graph graph, List<Integer> group1, List<Integer> group2, int margin)`
  Próbuje zrównoważyć liczebność dwóch grup poprzez przenoszenie wierzchołków o najmniejszej odległości maksymalnej, przy zachowaniu spójności każdej grupy.
- `partitionGraph(Graph graph, int marginPercent)`
  Główna metoda odpowiadająca za podział grafu. Dla każdej składowej spójnej wyznacza środkowy wierzchołek, tworzy dwie grupy oraz sprawdza i ewentualnie naprawia ich spójność i balans liczebności względem podanego marginesu procentowego.
- `splitGraph(Graph graph)`
  Przekształca podział na nowy komponent grafu. Usuwa krawędzie między grupami i aktualizuje numerację komponentów.

- **GraphVisualisation** – Komponenty odpowiadające za graficzne przedstawienie grafu przed i po podziale.
  - `GraphPrePartitionPanel` – Panel wyświetlający graf przed podziałem.
  - `GraphPostPartitionPane` – Panel pokazujący graf po podziale.

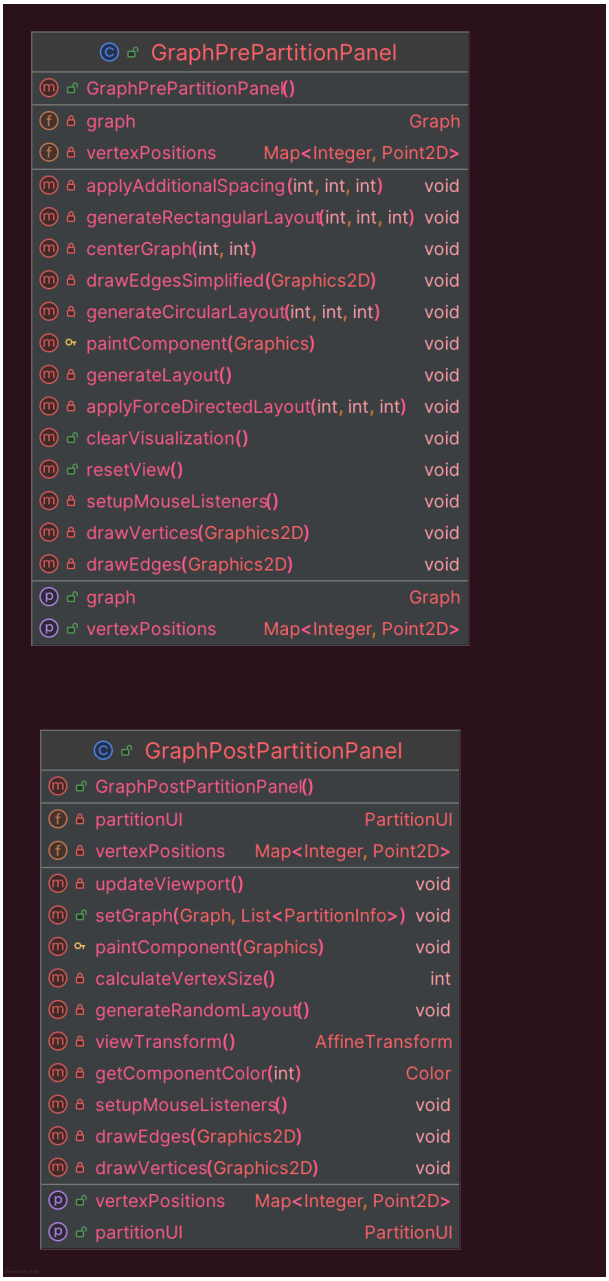

Figure 2: diagram GraphVisualisation

GraphPostPartitionPanel to komponent graficzny dziedziczący po `JPanel`, który służy do wizualizacji grafu po jego podziale na komponenty spójne. Klasa umożliwia interaktywną prezentację wyników podziału z wykorzystaniem kolorów dla poszczególnych komponentów, obsługuje zoomowanie, przesuwanie widoku oraz dynamiczne rysowanie wierzchołków i krawędzi.

Najważniejsze metody

- `setGraph(Graph, List<PartitionInfo>)` – ustawia graf i listę podziałów oraz generuje pozycje wierzchołków.
- `generateRandomLayout()` – przypisuje pozycje wierzchołkom w ramach ich komponentów w sposób losowy, ale rozdzielony przestrzennie.
- `paintComponent(Graphics)` – główna metoda rysująca, przekształca widok i rysuje krawędzie i wierzchołki.
- `drawEdges(Graphics2D)` – rysuje krawędzie między wierzchołkami należącymi do komponentów.

– `drawVertices(Graphics2D)` – rysuje wierzchołki w kolorach przypisanych komponentom; wyświetla etykiety przy odpowiednim poziomie zoomu.
– `setupMouseListeners()` – konfiguruje obsługę myszy (przesuwanie widoku i zoomowanie kółkiem).
– `updateViewport()` – aktualizuje zestaw widocznych wierzchołków na podstawie aktualnego widoku.

Opis działania

Podczas inicjalizacji, komponent ustawia białe tło i aktywuje nasłuchy myszy. Po wywołaniu `setGraph`, generowany jest układ współrzędnych wierzchołków na panelu. Wierzchołki rozmieszczane są w obrębie oddzielnych stref przypisanych komponentom spójnym.

Podczas rysowania (metoda `paintComponent`) wykorzystywana jest transformacja macierzy, aby dostosować rysowanie do aktualnego poziomu zoomu i przesunięcia. Komponent dynamicznie oblicza widoczne wierzchołki i krawędzie, co pozwala na efektywne renderowanie dużych grafów. Kolory komponentów są przypisywane cyklicznie z tablicy `COMPONENT_COLORS`.

Interakcja z użytkownikiem
Użytkownik może:

– przytrzymać lewy przycisk myszy i przeciągnąć widok (panowanie),
– użyć scrolla myszy do powiększania i pomniejszania,
– obserwować zmieniającą się informację o zoomie i liczbie widocznych wierzchołków.

Współpraca z PartitionUI
Komponent może raportować do interfejsu `PartitionUI` informacje o aktualnym poziomie zoomu oraz liczbie widocznych wierzchołków, co pozwala na synchronizację informacji o stanie widoku z innymi elementami interfejsu użytkownika.

GraphPrePartitionPanel.java

Plik `GraphPrePartitionPanel.java` zawiera klasę odpowiedzialną za wizualizację grafu przed jego podziałem. Komponent ten umożliwia użytkownikowi przeglądanie struktury grafu, przybliżanie, oddalanie oraz przesuwanie widoku. Klasa rozszerza komponent `JPanel` z biblioteki Swing i implementuje metody rysowania grafu oraz generowania layoutu wierzchołków.

Najważniejsze metody

– `void setGraph(Graph graph)` – ustawia graf do wyświetlenia i, jeśli flaga `regenerateLayout` ma wartość `true`, wywołuje metodę `generateLayout()`.
– `void generateLayout()` – generuje rozmieszczenie wierzchołków na podstawie liczby wierzchołków w grafie:
  * dla grafów o liczbie wierzchołków mniejszej lub równej 1000 stosowany jest układ kołowy,
  * dla grafów większych – prostokątny układ losowy z opcjonalnym rozrzutem i miejscem na algorytm siłowy.
– `void paintComponent(Graphics g)` – metoda odpowiedzialna za rysowanie komponentu. Rysuje krawędzie jako linie oraz wierzchołki jako okręgi w ustalonych kolorach i pozycjach.
– `void setupMouseListeners()` – ustawia obsługę zdarzeń myszy, pozwalając użytkownikowi przesuwać oraz skalować widok grafu.

Działanie layoutu Metoda `generateLayout()` dostosowuje sposób rozmieszczenia wierzchołków w zależności od wielkości grafu:

– **Układ kołowy:** Wierzchołki są rozmieszczone równomiernie na okręgu.
– **Układ prostokątny:** Wierzchołki rozmieszczane są w losowych pozycjach na prostokącie. Dla grafów bardzo dużych dodawany jest dodatkowy rozrzut pozycji.
– **Centrowanie:** Wszystkie pozycje są przesuwane tak, by graf był wyśrodkowany względem panelu.

Przykładowe użycie Po ustawieniu grafu za pomocą `setGraph()`, komponent rysuje graf w panelu użytkownika. Użytkownik może następnie:

- przesuwać widok za pomocą myszy,
- zmieniać poziom powiększenia przy użyciu kółka myszy.

Uwagi implementacyjne

- Stałe kolory i rozmiary wierzchołków/krawędzi są ustalone z góry.
- W celu powtarzalności rozmieszczenia użyto generatora losowego z ustalonym ziarnem.
- Algorytm siłowy został przewidziany jako przyszłe rozszerzenie.

Kod implementuje wizualizację grafu używając algorytmu force-directed (siłowego).

Jak to działa:

1. Wierzchołki odpychają się od siebie (jak ładunki elektryczne)
2. Połączone wierzchołki przyciągają się (jak sprężyny)
3. Temperatura kontroluje maksymalny ruch wierzchołków
4. System stopniowo się stabilizuje dzięki ochładzaniu

Cel tego algorytmu to:

- Równomierne rozmieszczenie wierzchołków
- Minimalizacja przecięć krawędzi
- Wizualne wyeksponowanie struktury grafu
- Zachowanie czytelności dla dużych grafów

Temperatura i Siły

1. **Temperatura** w algorytmie force-directed:
   - Kontroluje maksymalną odległość, na jaką wierzchołki mogą się przemieścić w każdej iteracji
   - Zmniejsza się stopniowo (ochładzanie) w kolejnych iteracjach
   - Pomaga w stabilizacji układu grafu
2. **Siły** działające w układzie:
   - **Siła odpychania** (repulsion) - działa między wszystkimi parami wierzchołków
   - **Siła przyciągania** (attraction) - działa tylko między połączonymi wierzchołkami

- **io** – Pakiet zajmujący się operacjami wejścia/wyjścia.
  - `GraphLoaderBin` – Klasa do ładowania grafów z plików binarnych.
    GraphLoaderBin.java

    Klasa `GraphLoaderBin` odpowiada za wczytywanie grafu zapisanego w specjalnym formacie binarnym. Plik zawiera jeden lub więcej grafów zapisanych w formacie CSR (Compressed Sparse Row). Dane są wczytywane z uwzględnieniem kolejności bajtów typu little-endian.

    Podczas ładowania:

    * Pomijane są dane o strukturze siatki (`vertices_by_rows`, `row_indexes`), które nie są potrzebne do budowy listy sąsiedztwa.
    * Dla każdego grafu odczytywane są tablice sąsiedztwa (`adj`) i indeksy wierszy (`adjIndex`).
    * Wierzchołki z poszczególnych grafów są przesuwane względem siebie (przez offset), a następnie ich dane są scalane w jedną strukturę CSR.
    * Po scaleniu dane CSR są ustawiane w obiekcie klasy `Graph`, a następnie konwertowane do listy sąsiedztwa.

    Kluczowe metody:

    * `loadGraph(String filePath)` – główna metoda ładująca dane z pliku binarnego.

    - `convertCSRToNeighbors(Graph graph)` – konwertuje strukturę CSR na listę sąsiedztwa, zakładając nieskierowany graf.
    - `readUInt32(DataInputStream dis)` – odczytuje 4 bajty jako liczbę całkowitą bez znaku (unsigned int) w porządku little-endian.
  - `GraphLoaderCsrrg` – Klasa do wczytywania grafów w formacie .csrrg.
    GraphLoaderCsrrg.java

    Klasa `GraphLoaderCsrrg` odpowiada za wczytywanie grafu z pliku w formacie `.csrrg`, opartym na strukturze CSR (Compressed Sparse Row).

    Najważniejsze metody

    - `Graph loadGraph(String filePath)` – główna metoda wczytująca graf z pliku. Tworzy obiekt `Graph`, wczytuje dane, inicjalizuje sąsiadów oraz konwertuje format CSR do listy sąsiedztwa.
    - `int[] parseLineToIntArray(String line)` – pomocnicza metoda konwertująca linię tekstu oddzieloną średnikami na tablicę liczb całkowitych.
    - `void convertCSRToNeighbors(Graph graph)` – przekształca dane w formacie CSR na reprezentację listy sąsiedztwa. Tworzy połączenia między liderem grupy a pozostałymi członkami.

    Format wejściowy

    - **Sekcja 1: Rozmiar grafu** - Pierwsza linia zawiera pojedynczą liczbę całkowitą n, określającą maksymalny wymiar grafu. Liczba określa szerokość oraz wysokość grafu(maksymalną w każdym wierszu/kolumnie).
    - **Sekcja 2: Układ wierzchołków** - Kolejne liczby określają numer kolumny, w której znajduje się dany węzeł
    - **Sekcja 3: Rozkład wierszy** - Określa kolejne nakładające się ze sobą pary wyznaczające zakres nastepnych wierszy. Np. 0,8,11-pierwszy wiersz obejmuje elementy z sekcji 2 od 0 do 8(zaczynając liczenie od liczby+1), a kolejny wiersz ma elementy od 9 do 11.
    - **Sekcja 4: Lista grup połączonych wierzchołków** - Zawiera listę grup wierzchołków, które należą do wspólnych komponentów grafu. Każda grupa jest reprezentowana jako zbiór wierzchołków, które są wzajemnie połączone bezpośrednimi krawędziami.
    - **Sekcja 5: Wskaźniki na pierwsze węzły w grupach** - Wskazuje, gdzie zaczynają się kolejne grupy połączonych węzłów opisane w sekcji 4. Może występowac wielokrotnie, jeśli plik zawiera więcej niż jeden graf.

- **model** – Pakiet z głównymi strukturami danych i logiką modelu domenowego.
  - `Graph` – Klasa reprezentująca strukturę grafu.
  - `GraphException` – Klasa wyjątków dotyczących operacji na grafie.
  - `PartitionResult` – Klasa zawierająca wynik operacji podziału grafu.

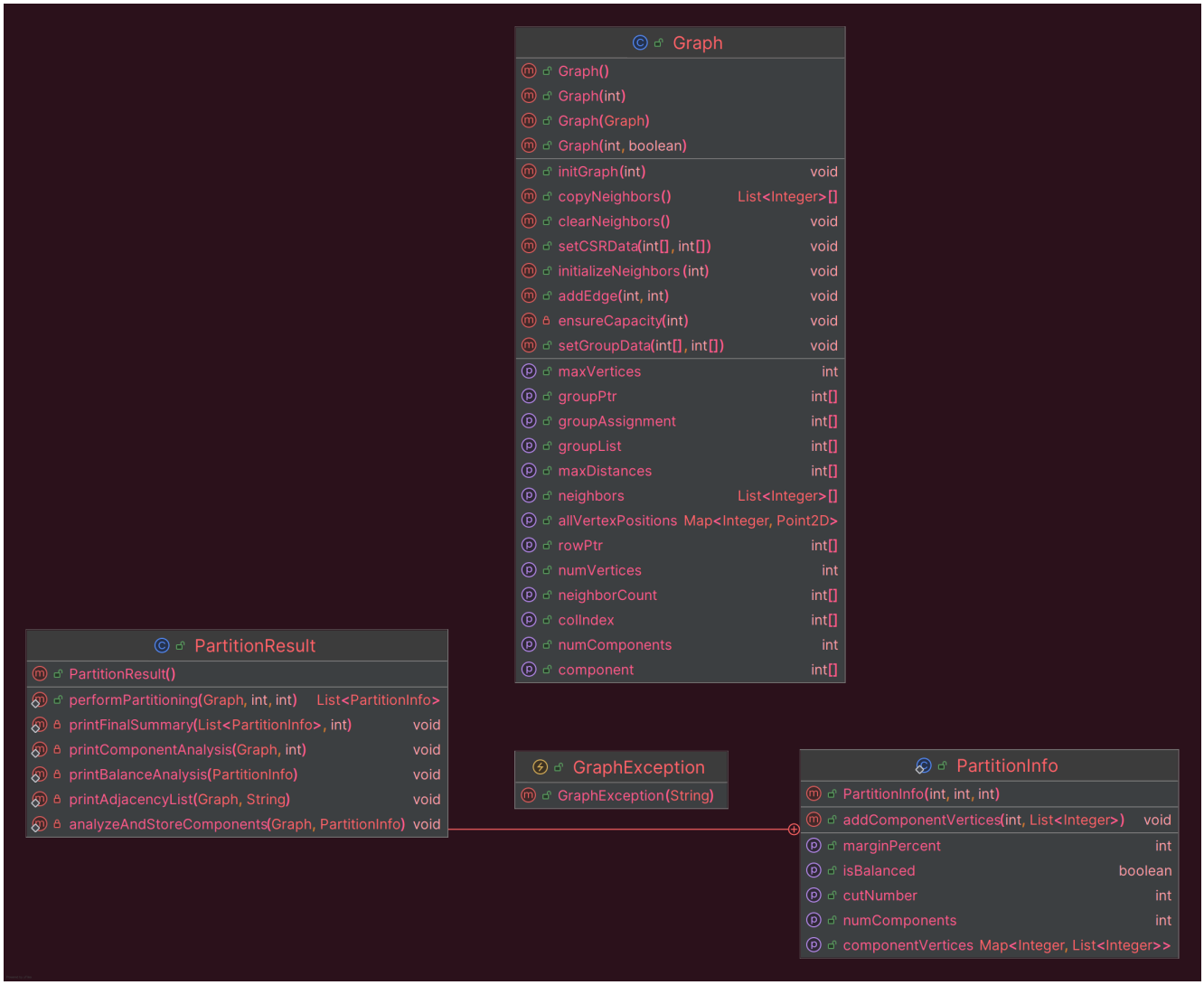

Figure 3: diagram model

Klasa `Graph`

Klasa **`Graph`** reprezentuje strukturę grafu i obsługuje zarówno dynamiczną reprezentację grafu w postaci list sąsiedztwa, jak i statyczną reprezentację w formacie CSR (Compressed Sparse Row).

- **Pola:**
  - `maxVertices`, `numVertices` – maksymalna oraz aktualna liczba wierzchołków.
  - `neighbors[]` – dynamiczne listy sąsiedztwa.
  - `neighborCount[]` – liczba sąsiadów dla każdego wierzchołka.
  - `groupAssignment[]` – przypisanie wierzchołków do grup.
  - `component[]` – przypisanie wierzchołków do komponentów spójności.
  - `colIndex[]`, `rowPtr[]` – tablice reprezentujące graf w formacie CSR.
  - `groupList[]`, `groupPtr[]` – dane grupowania w CSR.
  - `vertexPositions` – mapa położeń wierzchołków w przestrzeni 2D (opcjonalna).
- **Metody:**
  - `addEdge(int u, int v)` – dodaje nieskierowaną krawędź pomiędzy wierzchołkami.
  - `clearNeighbors()` – usuwa wszystkie połączenia w grafie.
  - `setCSRData(...)` – ustawia dane CSR.
  - `setGroupData(...)` – ustawia dane grup w CSR.

    * `copyNeighbors()` – zwraca kopię list sąsiedztwa.
    * `ensureCapacity(...)` – dynamicznie zwiększa pojemność tablic.

Klasa umożliwia elastyczne zarządzanie strukturą grafu, jego analizę oraz przetwarzanie danych wejściowych w formacie .csrrg.

Klasa `PartitionResult`

Klasa `PartitionResult` odpowiada za wykonanie iteracyjnego podziału grafu oraz analizę uzyskanych komponentów. Operuje na obiekcie `Graph` i wykorzystuje algorytmy zawarte w klasie `GraphPartitioner`. Każdy udany podział zapisywany jest w strukturze pomocniczej `PartitionInfo`.

Struktura `PartitionInfo`

Jest to wewnętrzna statyczna klasa służąca do przechowywania informacji o pojedynczym podziale grafu:

- `cutNumber` – numer wykonanej operacji podziału,
- `numComponents` – liczba komponentów po podziale,
- `marginPercent` – zadany margines procentowy dla balansu rozmiarów,
- `isBalanced` – flaga określająca, czy podział był zbalansowany,
- `componentVertices` – mapa zawierająca identyfikatory komponentów oraz przypisane do nich wierzchołki.

Metoda `performPartitioning`

Metoda `performPartitioning(Graph graph, int numCuts, int marginPercent)` jest punktem wejściowym całego procesu podziału:

1. Obliczana jest początkowa liczba komponentów w grafie.
2. W pętli do skutku (lub osiągnięcia maksymalnej liczby cięć):
    - wywoływana jest metoda `GraphPartitioner.partitionGraph()`,
    - po udanym cięciu aktualizowana jest liczba komponentów,
    - tworzony jest obiekt `PartitionInfo`, do którego zapisywana jest informacja o nowym stanie,
    - przeprowadzana jest analiza zbalansowania nowo powstałych komponentów,
    - dane są wypisywane w konsoli (dla celów diagnostycznych).
3. Na końcu wypisywane jest podsumowanie.

Analiza balansu

Dla każdego udanego podziału sprawdzana jest różnica w rozmiarach pomiędzy dwoma najmniejszymi komponentami. Jeżeli różnica jest mniejsza lub równa dopuszczalnemu marginesowi (wyrażonemu procentowo względem ich łącznego rozmiaru), to podział uznawany jest za zbalansowany.

Wybrane metody pomocnicze

- `analyzeAndStoreComponents()` – grupuje wierzchołki na podstawie identyfikatora komponentu i zapisuje do struktury `PartitionInfo`.
- `printAdjacencyList()` – wypisuje listę sąsiedztwa grafu w danym momencie.
- `printComponentAnalysis()` – wypisuje przydział wierzchołków do komponentów.
- `printBalanceAnalysis()` – analizuje rozmiary dwóch najmniejszych komponentów i porównuje z dopuszczalnym marginesem.
- `printFinalSummary()` – wypisuje podsumowanie wykonanych cięć.

- `test_data` - Dane testowe
- `Main` - Główna klasa testowa

- `MainFrame` - Główna rama aplikacji testowej
  Klasa `MainFrame`

  `MainFrame` to główna klasa okna aplikacji odpowiedzialna za zarządzanie interfejsem użytkownika oraz logiką przełączania widoków. Rozszerza klasę `JFrame` i wykorzystuje mechanizm `CardLayout` do przełączania pomiędzy trzema panelami:

  - `MainUI` – widok główny z przyciskami operacyjnymi,
  - `DetailsUI` – widok szczegółów grafu,
  - `PartitionUI` – widok wizualizacji grafu po podziale.

  Główne zadania klasy to:

  - Obsługa wczytywania grafu z pliku tekstowego (`.csrrg`) lub binarnego (`.bin`),
  - Przechowywanie oryginalnego grafu i jego kopii (z zachowaniem pozycji wierzchołków),
  - Przechowywanie wyników podziału w postaci listy `PartitionInfo`,
  - Obsługa przycisków przełączających widoki aplikacji,
  - Tworzenie i konfiguracja menu aplikacji,
  - Przekazywanie danych pomiędzy komponentami graficznymi.

  Klasa dba również o aktualizację danych wyświetlanych po wykonaniu podziału grafu, zapewniając poprawne odświeżenie wizualizacji oraz szczegółów grafu. Obsługuje błędy wczytywania plików i informuje użytkownika za pomocą okien dialogowych.

  Przykładowe metody pomocnicze:

  - `updatePartitionResult()` – aktualizuje dane po wykonaniu podziału,
  - `copyGraphWithPositions()` – kopiuje pozycje wierzchołków między grafami,
  - `createMenu()` – tworzy menu górne aplikacji z opcją wczytywania i pomocą.

2.2 1.2 Interfejs użytkownika

- **DetailsUI** – Okno zawierające szczegółowe informacje o grafie w postaci listy sąsiedztwa.

  - `DetailsUI` – Główna klasa odpowiedzialna za logikę i prezentację okna.
  - `DetailsUI.form` – Plik definiujący układ komponentów GUI.

  Klasa `DetailsUI` odpowiada za wyświetlanie technicznych szczegółów dotyczących podziału grafu w interfejsie użytkownika. Główne funkcjonalności:

  - **Inicjalizacja tabel:**
    - `initializeTables()` - inicjalizuje obie tabele (przed i po podziale)
    - `showOriginalGraphTable()` - wyświetla listę sąsiedztwa oryginalnego grafu
    - `showPartitionedGraphTable()` - prezentuje graf po podziale z podziałem na grupy
  - **Obsługa danych:**
    - `setPartitionResults()` - ustawia wyniki podziału do wyświetlenia
    - `setOriginalNeighbors()` - zapisuje oryginalną listę sąsiedztwa
  - **Konfiguracja interfejsu:**
    - `configureTable()` - dostosowuje wygląd tabel (wysokość wierszy, szerokość kolumn)
    - Metody dostępowe do komponentów GUI (`getPanel()`, `getBackButton()`)

- **MainUI** – Główne okno aplikacji umożliwiające ładowanie grafu, wybór parametrów i rozpoczęcie procesu podziału.

  - `MainUI` – Główna klasa okna.
  - `MainUI.form` – Definicja wizualna okna.

  Klasa `MainUI` stanowi główny interfejs użytkownika aplikacji do podziału grafu. Główne funkcjonalności:

  - **Inicjalizacja interfejsu:**
    * Konstruktor konfiguruje spinnery i przyciski
    * Metoda `$$$setupUI$$$()` generuje układ GUI
  - **Obsługa akcji użytkownika:**
    * `onPodzielButtonClick()` - wykonuje podział grafu z podanymi parametrami
    * `onResetujWidokButtonClick()` - resetuje interfejs do stanu początkowego
  - **Zarządzanie grafem:**
    * `setGraph()` - ustawia graf do wizualizacji
    * `restoreOriginalGraph()` - przywraca oryginalny graf
    * `saveCurrentPositions()` - zapisuje pozycje wierzchołków
  - **Elementy GUI:**
    * Panel `GraphPrePartitionPanel` do wizualizacji grafu
    * Przyciski sterujące (`buttonPodziel`, `resetujWidokButton`)
    * Spinnery do ustawiania parametrów (`spinnerNumCuts`, `spinnerMargines`)
    * Etykiety informacyjne o stanie grafu

    Klasa integruje komponenty wizualne z logiką podziału grafu, zapewniając intuicyjną interakcję z użytkownikiem.

- **PartitionUI** – Okno aplikacji odpowiedzialne za wyświetlanie wyników podziału grafu.
  - `PartitionUI` – Główna klasa panelu po podziale grafu.
  - `PartitionUI.form` – Definicja wizualna panelu.

  Klasa `PartitionUI` zarządza interfejsem prezentującym podzielony graf oraz umożliwia jego ponowne wyświetlenie lub wyczyszczenie. Główne funkcjonalności:

  - **Inicjalizacja interfejsu:**
    * Konstruktor tworzy i osadza panel `GraphPostPartitionPanel` w miejscu placeholdera
    * Metoda `$$$setupUI$$$()` generuje układ graficzny komponentów
  - **Zarządzanie grafem:**
    * `setGraph(Graph, List<PartitionInfo>)` – ustawia graf oraz wyniki podziału do wizualizacji i odtwarza poprzednie pozycje wierzchołków
    * `clearVisualization()` – czyści wizualizację i zapisane pozycje wierzchołków
  - **Informacje o widoku:**
    * `updateViewZoomInfo()` – aktualizuje informacje o poziomie powiększenia i liczbie widocznych wierzchołków
  - **Elementy GUI:**
    * `GraphPostPartitionPanel` – komponent rysujący graf po podziale
    * `partitionBackButton` – przycisk powrotu do ekranu głównego
    * `viewZoomInfo` – etykieta informacyjna o stanie widoku

  Klasa łączy komponenty wizualne z logiką prezentacji wyników podziału grafu, umożliwiając użytkownikowi przeglądanie i analizę wygenerowanych partycji.

2.3 Panel narzędziowy

Panel narzędziowy dostępny w oknie głównym zawiera:

- Pola tekstowe do ustawienia parametrów N i M
- Przycisk `Podziel` uruchamiający algorytm podziału
- Przycisk `Resetuj widok`, który czyści dane i przywraca domyślne ustawienia
- Przycisk `Szczegóły techniczne` do otwarcia `DetailsUI`
- Przycisk `Wizualizacja grafu po podziale` do otwarcia `PartitionUI`
- Pola informacyjne o liczbie udanych przecięć oraz spójności grafu

2.4 Obsługa błędów

W przypadku błędów (brak wczytanego pliku, niespełnienie warunków podziału) aplikacja wyświetla komunikaty błędów w postaci okien dialogowych bez modyfikowania aktualnego stanu aplikacji.

3 Algorytm podziału grafu

Proces podziału grafu realizowany jest w kilku krokach:

3.1 Wyznaczenie wierzchołka centralnego

Na potrzeby efektywnego podziału grafu wybierany jest wierzchołek centralny. W tym celu uruchamiany jest algorytm Dijkstry z losowego wierzchołka. Następnie jako punkt startowy przeszukiwania wybierany jest wierzchołek najbardziej oddalony od punktu początkowego, co pozwala objąć większą część grafu w pierwszej grupie.

3.2 Podział na dwie grupy

Podział wierzchołków na dwie grupy realizowany jest z wykorzystaniem algorytmu DFS (Depth-First Search), który działa od wyznaczonego wierzchołka centralnego. Liczba wierzchołków w każdej grupie jest monitorowana, aby zapewnić możliwie równy rozkład (z tolerancją różnicy jednego wierzchołka).

3.3 Sprawdzenie spójności

Po zakończeniu podziału każda z grup poddawana jest niezależnemu sprawdzeniu spójności, również z wykorzystaniem algorytmu DFS. Weryfikowana jest liczba wierzchołków odwiedzonych podczas przeszukiwania – musi ona odpowiadać liczbie elementów w danej grupie.

3.4 Postępowanie z niespójną drugą grupą

Gdy po podziale okaże się, że druga grupa nie jest spójna, algorytm wykonuje następujące kroki:

1. **Identyfikacja komponentów spójnych**:
   - Dla niespójnej grupy drugiej wyznaczane są wszystkie jej składowe spójne za pomocą algorytmu DFS
   - Każda składowa otrzymuje tymczasowy identyfikator
2. **Wybór głównej składowej**:

- Wybierana jest największa składowa spójna (o największej liczbie wierzchołków)
- Pozostałe składowe są oznaczane do przeniesienia

3. **Przenoszenie wierzchołków**:
   - Wszystkie wierzchołki z mniejszych składowych są przenoszone do pierwszej grupy
   - W grupie drugiej pozostają tylko wierzchołki z głównej składowej
4. **Weryfikacja warunków**:
   - Sprawdzana jest spójność nowo utworzonej grupy pierwszej
   - Weryfikowana jest różnica liczby wierzchołków między grupami

$$|V_1| - |V_2| \leq \text{margin} \tag{1}$$

   - Jeśli warunki nie są spełnione, algorytm wraca do etapu podziału z nowymi grupami
5. **Obsługa przypadków skrajnych**:
   - Gdy przeniesienie wierzchołków narusza spójność grupy pierwszej, wybierany jest alternatywny podział
   - W przypadku niemożności spełnienia warunków, algorytm może:
     * Próbować podziału innej składowej spójnej
     * Zwrócić informację o niemożności podziału

3.5 Generowanie podgrafów

Dla każdej z dwóch grup tworzony jest osobny podgraf. Dane są przetwarzane do wewnętrznej reprezentacji listy sąsiedztwa, uwzględniając jedynie sąsiadów należących do tej samej grupy. W efekcie uzyskiwane są dwa spójne podgrafy, gotowe do zapisania w plikach wyjściowych.

4 Repozytorum GitHub

Cały kod źródłowy projektu dostępny jest w repozytorium: podzial grafu.