Dokumentacja finalna programu Visualizer

Wojciech Regulski, Kajetan Wójcik

12.06.2025

Spis treści

1 Cel projektu

Program `Visualizer` ma na celu rozwiązanie problemu `Sparsest Cut`. Polega on na podziale grafu na określoną liczbę części, zapewniając równomierność liczby wierzchołków oraz minimalizację liczby przeciętych krawędzi. Parametry określane są z interfejsu programu.

2 Instrukcja obsługi programu

- `Number of divisions` — pozwala wybrać, liczbę części, na które graf zostanie podzielony,
- `Margin` — określa margines błędu podczas podziału grafu, jeżeli nie jest on spełniony, program próbuje zwiększyć margines błędu informując o tym użytkownika,
- `File > Load from...` — wczytuje wybrany przez użytkownika graf w formie tekstowej lub binarnej,
- `File > Save graph` — zapisuje wybraną część grafu (lub wszystkie po wybraniu opcji `Save all`, w formie tekstowej lub binarnej, do wybranej przez użytkownika lokalizacji,
- `Divide` — główna funkcja programu, dzieli graf zgodnie z podanymi wcześniej parametrami,
- `Zoom` — przy użyciu scrolla można dowolnie przybliżać oraz oddalać graf,
- `Move` — po przytrzymaniu lewego przycisku myszy, ruszając myszką można przesuwać widok grafu.

3 Użyte technologie

Projekt został zrealizowany w języku Java z użyciem biblioteki Swing do tworzenia interfejsu graficznego. Algorytmy partycjonowania grafu zostały zaimplementowane od podstaw, a dane wejściowe i wyjściowe zapisano w specjalnym formacie `.csrrg` (tekstowym i binarnym).

4 Struktury danych

W implementacji programu Visualizer w języku Java wykorzystano zestaw klas reprezentujących strukturę grafu oraz interfejs graficzny aplikacji. Klasy te umożliwiają modelowanie wierzchołków, relacji między nimi, a także wyświetlanie grafu i interakcję użytkownika.

Vertex

Reprezentuje pojedynczy wierzchołek grafu.

- `int id` — identyfikator wierzchołka.
- `List<Vertex> neighbors` — lista sąsiadujących wierzchołków.

- `int groupId` — identyfikator partycji, do której należy dany wierzchołek.
- `int x, y` — współrzędne wierzchołka w przestrzeni 2D (używane do wizualizacji).

Każdy wierzchołek tworzony jest z przypisanym `id`, a jego sąsiedzi mogą być dynamicznie dodawani do listy.

Graph

Reprezentuje całą strukturę grafu.

- `Vertex[] vertexData` — tablica wszystkich wierzchołków w grafie.
- `int maxDim` — maksymalny wymiar grafu wykorzystywany do skalowania w widoku.

Konstruktor klasy `Graph` tworzy tablicę `Vertex` o zadanym rozmiarze i inicjalizuje każdy wierzchołek z odpowiednim identyfikatorem.

GraphPanel

Komponent odpowiedzialny za wizualizację grafu.

- Dziedziczy po `JPanel` i nadpisuje metodę `paintComponent(Graphics g)` w celu rysowania wierzchołków oraz ich połączeń.
- Obsługuje:
  - **Powiększanie i pomniejszanie** grafu za pomocą scrolla myszy.
  - **Przesuwanie widoku** poprzez przeciąganie myszą.
- Zawiera pola:
  - `Graph graf` — graf do narysowania.
  - `double scale`, `offsetX`, `offsetY` — parametry przeskalowania i przesunięcia widoku.
  - `int lastDragX, lastDragY` - współrzędne do śledzenia przeciągania.

`GraphPanel` umożliwia płynne i dynamiczne przeglądanie dużych struktur grafowych przez użytkownika.

GraphUI

Główne okno aplikacji zbudowane na bazie `JFrame`.

- Udostępnia graficzny interfejs użytkownika (GUI), umożliwiający:
  - Wczytanie grafu z pliku tekstowego lub binarnego,
  - Zapis grafu do pliku,
  - Uruchomienie partycjonowania,
  - Ustawienie liczby partycji i marginesu procentowego.

- Kluczowe komponenty:
  - `GraphPanel graphPanel` — panel do rysowania grafu,
  - `JTextField divisionsField`, textttmarginField — pola wejściowe dla użytkownika,
  - Menu `File` z opcjami wczytywania/zapisu grafów,
  - Przycisk `Divide` do rozpoczęcia procesu partycjonowania.

`GraphUI` łączy logikę aplikacji z warstwą prezentacji, umożliwiając użytkownikowi prostą i intuicyjną obsługę programu.

5 Obsługa błędów

W implementacji programu Visualizer w języku Java zastosowano mechanizmy obsługi wyjątków, które umożliwiają bezpieczne reagowanie na błędy pojawiające się w czasie działania programu.
Główne założenia:

- Program nie powinien się zatrzymywać w przypadku błędu - zamiast tego powinien informować użytkownika o problemie w formie okienek dialogowych.
- Wykryte błędy są przechwytywane i obsługiwane za pomocą bloków `try-catch`.

Typowe przypadki obsługiwanych błędów:

- **Błędy podczas wczytywania pliku:**

```
1     try {
2         graph = TempGraphIO.loadGraph(fc.getSelectedFile().
              getAbsolutePath(), 1);
3         graphPanel.setGraph(graph); // Przekazujemy graf do
              panelu
4         repaint(); // Odswiezamy widok
5         } catch (Exception e) {
6             JOptionPane.showMessageDialog(this, "An␣error␣
                  occured␣while␣loading␣text␣file:␣" + e.
                  getMessage(), "Error", JOptionPane.
                  ERROR_MESSAGE);
7         }
```

- **Błędy zapisu do pliku:**

```
1     try {
2         if (saveAll) {
3             for (int i = 0; i <= maxGroup; i++) {
4                 String path = basePath + "_group_" + i + ".
                      csrrg";
5                 TempGraphIO.savePartition(graph, assignment, i
                      , path, isBinary);
6             }
```

```
7               JOptionPane.showMessageDialog(this, "All␣groups␣
                    saved␣successfully.");
8           } else {
9               String path = basePath;
10              if (!path.endsWith(".csrrg")) path += ".csrrg";
11              TempGraphIO.savePartition(graph, assignment,
                    selectedGroup, path, isBinary);
12              JOptionPane.showMessageDialog(this, "Group␣" +
                    selectedGroup + "␣saved␣to:\n" + path);
13          }
14      } catch (Exception ex) {
15          ex.printStackTrace();
16          JOptionPane.showMessageDialog(this, "Error␣saving:␣" +
                ex.getMessage());
17      }
```

Zastosowane klasy wyjątków:

- `IOException` - odczyt/zapis pliku,
- `NumberFormatException` - błędne dane wprowadzone przez użytkownika,
- Inne specyficzne wyjątki mogą być przechwytywane w zależności od potrzeb (np. `EOFException` lub `RuntimeException` wywołane w naszym kodzie w sytuacjach awaryjnych).

6 Algorytm partycjonowania grafu

W celu uzyskania spójnych i zrównoważonych podziałów grafu w programie Visualizer zastosowano dwa podejścia algorytmiczne: zmodyfikowany algorytm Dijkstry oraz klasyczny algorytm Kernighana-Lina. Każdy z nich odpowiada za inny etap procesu podziału - inicjalizację i optymalizację.

6.1 Zmodyfikowany algorytm Dijkstry - inicjalne przypisanie grup

Moduł ten odpowiada za utworzenie bazowego podziału grafu na podstawie odległości wierzchołków od tzw. "nasion"(punktów startowych). Stanowi pierwszy krok w procesie partycjonowania.
Funkcje pełnione przez algorytm:

- Oblicza najkrótsze ścieżki z wybranego wierzchołka do wszystkich pozostałych w grafie, wykorzystując klasyczny algorytm Dijkstry.
- Przeszukuje graf w celu znalezienia pary wierzchołków o największej wzajemnej odległości. Para ta stanowi dobre punkty startowe (tzw. nasienne) do dalszego podziału.
- Przypisuje każdy wierzchołek do jednej z dwóch grup w zależności od tego, który z punktów startowych znajduje się bliżej. Dzięki temu graf zostaje podzielony na dwie logiczne części.

- Przymuje liczbę grup i przypisuje wierzchołki na podstawie ich odległości do wybranych nasion, zapewniając wstępnie logiczny i przestrzenny podział.

6.2 Algorytm Kernighana-Lina - optymalizacja podziału

Drugi etap procesu partycjonowania opiera się na algorytmie Kernighana-Lina, który iteracyjnie poprawia jakość podziału poprzez minimalizację liczby przeciętych krawędzi pomiędzy grupami.
Funkcja pełniona przez algorytm:

- Wykonuje iteracyjną optymalizację przypisań, poszukując par wierzchołków, których zamiana pomiędzy grupami prowadzi do zmniejszenia liczby przecięć. Proces ten powtarzany jest tak długo, jak długo przynosi poprawę jakości cięcia.

6.3 Podsumowanie

Zastosowane podejście hybrydowe - wykorzystanie heurystycznej inicjalizacji przez Dijkstrę oraz optymalizacji lokalnej przez Kernighana-Lina - pozwala na uzyskanie partycji grafu, które są zarówno spójnie topologicznie, jak i optymalnie rozdzielone względem liczby przecięć. Taka konstrukcja algorytmiczna łączy szybkość działania z wysoką jakością wyniku końcowego.

7 Formaty plików wejściowych i wyjściowych

Program przyjmuje plik o rozszerzeniu `csrrg`, który jest typem tekstowym lub plik typu `bin`, który został wygenerowany przez program Visualizer.

Tekstowy

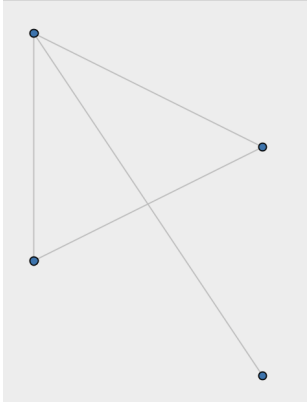
Plik tekstowy `csrrg` służy do przechowywania danych opisujących strukturę grafu i składa się z następujących sekcji zapisywanych kolejno w liniach:

1. **Linia 1:** Zawiera wartość określającą maksymalną liczbę węzłów, które mogą wystąpić w jednym wierszu.
2. **Linia 2:** Przechowuje indeksy wszystkich węzłów, podzielonych na wiersze, według ustalonego porządku.
3. **Linia 3:** Zawiera wskaźniki do pierwszych indeksów każdego wiersza znajdującego się w drugiej linii, umożliwiając łatwe odczytanie podziału na wiersze.
4. **Linia 4:** Opisuje grupy węzłów połączonych krawędziami, tworząc spójne fragmenty grafu.
5. **Linia 5 i kolejne:** Zawierają wskaźniki do pierwszych węzłów w każdej z grup opisanych w linii 4. Ta część pliku może występować wielokrotnie, co wskazuje, że dany plik zawiera więcej niż jeden graf.

Wynikiem działania programu jest plik zapisany w tym formacie. W zależności od wyboru użytkownika plik może być zapisany też binarnie.

Przykład

```
4
0;2;0;2
0;1;2;3;4
0;1;2;3;1;2
0;4
```



Rysunek 1: Wizualizacja grafu dla przykładu

Format binarny wejściowy

Dla zwiększenia wydajności i zmniejszenia rozmiaru pliku zastosowano binarną wersję formatu `csrrg`, zachowującą tę samą strukturę logiczną. Dane zapisywane są jako surowe bajty.

Na początku pliku umieszczana jest liczba wierzchołków, zakodowana w formacie **vByte**. Następnie zapisywane są posortowane listy sąsiedztwa – po jednej dla każdego wierzchołka. W każdej liście pierwszy sąsiad zapisywany jest w całości, a kolejne jako różnice względem poprzedniego (kodowanie delt). Wszystkie liczby kodowane są przy użyciu **vByte**.

Dla ułatwienia parsowania, bloki danych oddzielane są 8-bajtowym separatorem `0xDEADBEEFCAFEBABE` (w zapisie *little-endian*: `BE BA FE CA EF BE AD DE`).

Kodowanie vByte

vByte dzieli liczbę na 7-bitowe fragmenty zapisane w bajtach. MSB (najstarszy bit) każdego bajtu to flaga:

- **1** – oznacza, że liczba jest kontynuowana w kolejnym bajcie,
- **0** – to ostatni bajt danej liczby.

Fragmenty zapisywane są od najmłodszego (little-endian). Przykład: liczba `300` to `0xAC 0x02`, co odpowiada (0x02«7)+0x2C=300(0x02«7)+0x2C=300.

Format binarny wyjściowy

Istnieje możliwość zapisu danych w formie binarnej. Format binarny zachowuje strukturę logiczną formatu `csrrg`, jednak dane są skompresowane w celu zmniejszenia potrzebnego miejsca do zapisu.

Skompresowany plik zapisuje liczby w formacie 16-bitowym Little Endian . Oznacza to, że każda liczba zajmuje dokładnie 16 bitów, a kolejne wartości występują bezpośrednio po sobie.

Liczby są zapisane używając kodowania delt, pierwsza liczba linii jest zapisana normalnie natomiast każda kolejna jest różnicą względem poprzedniej.

Aby odróżnić linie od siebie wykorzystujemy prefiks długości linii, który pojawia się na początku pliku i każdej kolejnej linii, informuje ile wartości 16-bitowych występuje w danej linii pliku.

8 Struktura katalogów

Projekt Visualizer został zorganizowany zgodnie z typową strukturą projektu języka Java, jednak bez podziału na podkatalogi pakietowe. Wszystkie pliki źródłowe Java zostały umieszczone bezpośrednio w katalogu `src/main/java/`. Pozwala to na szybką kompilację i dostęp do wszystkich klas w jednym miejscu, co bywa wygodne w prostych projektach. Struktura katalogów wygląda następująco:

```
GraphCut_Java/
    src/
        main/
            java/
                Dijkstra.java
                Graph.java
                GraphPanel.java
                GraphUI.java
                KernighanLin.java
                Main.java
                Partition.java
                TempGraphIO.java
                Utils.java
                Vertex.java
    data/
        testGraph.csrrg
        testGraphBinary.csrrg
    docs/
        dokumentacja.pdf
        dokumentacja.tex
```

Opis katalogów:

- `src/main/java/`
  Zawiera wszystkie klasy źródłowe programu, w tym:
  - logikę grafową (`Graph`, `Vertex`),
  - komponenty GUI (`GraphUI`, `GraphPanel`),
  - algorytmy (`Dijkstra`, `KernighanLin`),

  - moduł odpowiedzialny za obsługę plików (`TempGraphIO`),
  - funckje pomocnicze (`Partition`, `Utils`),
  - główną klasę startową (`Main`).

- `data/`
  Przykładowe pliki wejściowe w formacie `.csrrg` (tekstowym i binarnym).

- `docs/`
  Dokumentacja programu w formacie LaTeX i PDF.

Uwagi:

- Przy braku podziału na pakiety wszystkie klasy należą do domyślnego pakietu. Jest to dopuszczalne w małych projektach, jednak przy większej rozbudowie zaleca się wprowadzenie pakietów logicznych np. (`graph`, `ui`, `algorithm`).

- W przypadku wprowadzenia testów jednostkowych zalecane jest utworzenie katalogu `src/test/java/`.

9 Podsumowanie

Projekt `Visualizer` spełnia zakładane cele poprzez połączenie funkcjonalnego interfejsu graficznego z zaawansowanym mechanizmem przetwarzania i partycjonowania grafów. Dzięki zastosowaniu dwóch uzupełniających się algorytmów, użytkownik otrzymuje rozwiązanie dokładne, szybkie i czytelne wizualnie. Struktura projektu oraz jakość implementacji umożliwia jego dalszy rozwój, np. przez dodanie innych heurystyk lub integrację z zewnętrznymi źródłami danych.