Sprawozdanie Końcowe
Podział grafu

Anastasiia Dmytrenko, Arkadiusz Perko

08.05.2025

1 Użyteczność programu

Program do podziału grafu w języku C umożliwia użytkownikowi efektywne dzielenie grafu na określoną liczbę części, minimalizując liczbę krawędzi między grupami. Program jest uruchamiany z poziomu terminala i wymaga precyzyjnego określenia parametrów wejściowych oraz wyjściowych, które pozwalają na konfigurację działania programu.

1.1 Plik wejściowy

Program oczekuje, że dane grafu będą zapisane w pliku tekstowym w formacie listy sąsiedztwa. Format pliku wejściowego musi być zgodny z poniższym schematem:

- **Pierwsza linia:** Maksymalna możliwa liczba węzłów w wierszu (w grafie nie musi znajdować się wiersz o takiej liczbie węzłów).
- **Druga linia:** Indeksy węzłów w poszczególnych wierszach - liczba wszystkich indeksów odpowiada liczbie węzłów grafu.
- **Trzecia linia:** Wskaźniki na pierwsze indeksy węzłów w liście wierszy z punktu 2.
- **Czwarta linia:** Lista sąsiedztwa węzłów grafu.
- **Piąta linia (i każda kolejna w przypadku grafów niespójnych):** Wskaźniki na pierwsze węzły w poszczególnych grupach z linii 4.

Przykładowa zawartość pliku wejściowego:

```
4
0; 2; 3; 1; 1; 2
0; 0; 3; 4; 6
0; 1; 1; 2; 3; 2; 3; 5; 3; 4
0; 2; 5; 8; 10
```

1.2 Parametry uruchomieniowe

Program przyjmuje kilka parametrów wejściowych, które umożliwiają konfigurację procesu podziału grafu.
Poniżej jest przedstawiony format standardowego wywołania programu oraz opis dostępnych flag:

```
./graph_partition -i [input-file] -o [output-file] -r [format-output-file] -p [parts] -b [error-margin] -m [method(optional)] [other optional: -f, -g, -h]
```

- **-i, --input-file**
  Określa ścieżkę do pliku wejściowego z danymi grafu. Plik ten musi być zapisany w formacie opisanym powyżej.

- **-o, --output-file**
  Określa ścieżkę do pliku wyjściowego, w którym zostaną zapisane wyniki podziału grafu. Plik może być zapisany w formacie tekstowym (ASCII) lub binarnym.

- **-r, --format ascii / binary**
  Określa format pliku wyjściowego. Dostępne opcje to:
  - **ascii** – wynik w formacie tekstowym (czytelny dla użytkownika),
  - **binary** – wynik w formacie binarnym (bardziej efektywny pod względem rozmiaru).

- **-p, --parts**
  Określa liczbę części, na które ma zostać podzielony graf. Liczba ta musi być równa bądź mniejsza połowie liczby wszystkich wierzchołków i $>= 2$. Program automatycznie dąży do równomiernego podziału, starając się minimalizować liczbę krawędzi między grupami.

- **-b, --error-margin**
  Ustala dopuszczalny margines błędu dla równomierności rozmiarów części grafu. Podaje się go w procentach. Na przykład, jeśli margines błędu wynosi 10, to każda część grafu może mieć od 90% do 110% docelowej wielkości.

- **-m, --method**
  Określa metodę podziału grafu. Możliwe wartości:
  - **kl** – Algorytm Kernighan-Lin, który jest heurystycznym algorytmem służącym do podziału grafu na dwie grupy, minimalizując liczbę krawędzi między nimi.
  - **m** – Metoda spektralna oparta na macierzy Laplacjana grafu, pozwalająca na podział grafu na więcej niż dwie części.

- **-g, --graph-index**
  Jest to flaga opcjonalna. Wybiera konkretny graf z pliku, gdy w pliku znajduje się więcej niż jeden graf. Numer grafu (indeks) jest podawany jako argument. Jeśli w pliku jest tylko jeden graf, flaga jest niepotrzebna.

- **-f, --force**
  Jest to flaga opcjonalna. Wymusza podział grafu, nawet jeśli nie uda się spełnić podanego marginesu błędu. Może to prowadzić do nierównych części grafu, ale pozwala na kontynuację działania programu mimo problemów.

- **-h, --help**
  Jest to flaga opcjonalna. Wyświetla pomoc z instrukcjami dotyczącymi używania programu.

1.3 Przykłady użycia

Przykład 1: Podział grafu na 4 części z użyciem algorytmu Kernighan-Lin i marginesem błędu równym 5%. Wynik zostanie zapisany w pliku `partition.txt` w formacie ASCII:

```
./graph_partition -i data.txt -o partition.txt -r ascii -p 4 -b 5 -m kl
```

Przykład 2: Podział grafu o indeksie 1 z pliku na 3 części z użyciem algorytmu opartego na metodzie spektralnej, z marginesem błędu równym 10%.Wynik zostanie zapisany w formacie binarnym:

```
./graph_partition --input-file data.txt -o partition.txt -r binary -p 3
    -b 10 -m m -g 1 --force
```

2 Opis funkcjonalności

Program przyjmuje dane wejściowe w formacie tekstowym, które reprezentują graf w postaci listy sąsiedztwa. Użytkownik może określić:

- plik wejściowy i wyjściowy,
- format pliku wyjściowego (tekstowy lub binarny),
- metodę podziału (kl, m),
- liczbę części oraz margines błędu dla ich rozmiarów,
- wybór grafu do podziału, jeśli w pliku wejściowym znajduje się więcej niż jeden graf.

3 Kluczowe struktury danych

3.1 Struktura Vertex

Prototyp struktury:

```
typedef struct vertex {
    int edge_num;
    int *conn;
    int group;
    int fixed;
    int processed;
    int D;
```

```
8     int x;
9     int y;
10 } Vertex;
```

Opis struktury: Struktura `Vertex` jest kluczową częścią reprezentacji grafu, w którym przechowywane są informacje o każdym wierzchołku. Każdy wierzchołek zawiera dane dotyczące liczby krawędzi wychodzących z niego, połączeń z innymi wierzchołkami, przynależności do grupy w procesie podziału oraz dodatkowe informacje pomocnicze, jak poprawki w algorytmie Kernighan-Lin (KL).

Argumenty struktury:

- **edge_num:**
  - Typ: int
  - Opis: Liczba krawędzi wychodzących z danego wierzchołka. Określa, ile krawędzi łączy dany wierzchołek z innymi wierzchołkami w grafie.
- **conn:**
  - Typ: int* (wskaźnik do tablicy)
  - Opis: Tablica przechowująca numery wierzchołków, z którymi dany wierzchołek jest bezpośrednio połączony. Każdy element tej tablicy wskazuje na inny wierzchołek, z którym obecny wierzchołek jest połączony krawędzią.
- **group:**
  - Typ: int
  - Opis: Wartość tej zmiennej określa, do której grupy należy dany wierzchołek, gdy graf jest dzielony na mniejsze części.
- **fixed:**
  - Typ: int
  - Opis: Określa, czy wierzchołek był poprawiany w trakcie działania algorytmu Kernighan-Lin. Wartość 1 oznacza, że wierzchołek był już przetworzony i nie jest brany pod uwagę przy dalszym podziale, podczas gdy 0 oznacza, że wierzchołek jest nadal aktywny i może być przenoszony między grupami w dalszej iteracji algorytmu.
- **processed:**
  - Typ: int
  - Opis: Określa, czy wierzchołek został przeniesiony do innej grupy podczas podziału. Wartość 1 oznacza, że wierzchołek został przeniesiony do innej grupy, a wartość 0 wskazuje, że wierzchołek pozostał w tej samej grupie.
- **D:**
  - Typ: int

- Opis: Przechowuje różnicę między liczbą krawędzi przecinających grupy a liczbą krawędzi wewnątrz grupy. Jest to zmienna pomocnicza w algorytmie Kernighan-Lin, używana do oceny, czy zamiana wierzchołków między grupami poprawia podział grafu.

- **x, y:**
  - Typ: int
  - Opis: Współrzędne wierzchołka w przestrzeni (np. wykorzystywane w algorytmie spektralnym do wizualizacji grafu lub w optymalizacji podziału).

3.2 Struktura Swap

Prototyp struktury:

```
typedef struct swap {
    int a;
    int b;
    int gain;
} Swap;
```

Opis struktury: Struktura `Swap` jest używana w algorytmie Kernighan-Lin (KL) do przechowywania informacji o zamianach wierzchołków między dwoma grupami. Zamiany te są oceniane na podstawie zysku w liczbie krawędzi wewnętrznych, które pozostają w obrębie jednej grupy.

Argumenty struktury:

- **a:**
  - Typ: int
  - Opis: Indeks pierwszego wierzchołka, który ma zostać zamieniony z drugim wierzchołkiem.

- **b:**
  - Typ: int
  - Opis: Indeks drugiego wierzchołka, który ma zostać zamieniony z pierwszym wierzchołkiem.

- **gain:**
  - Typ: int
  - Opis: Zysk z wymiany, czyli liczba krawędzi wewnętrznych, które pozostaną w tej samej grupie po dokonaniu zamiany. Większy zysk wskazuje na bardziej optymalny podział.

3.3 Struktura Matrix

Prototyp struktury:

```
typedef struct matrix {
    int n;
    double **data;
} Matrix;
```

Opis struktury: Struktura `Matrix` jest używana do reprezentacji macierzy, szczególnie macierzy Laplacjana, w algorytmie spektralnym. Macierz ta jest używana do obliczeń własności spektralnych grafu, takich jak wartości własne i wektory własne.

Argumenty struktury:

- **n:**
  - Typ: int
  - Opis: Liczba wierszy (lub kolumn) macierzy, która jest równa liczbie wierzchołków w grafie.
- **data:**
  - Typ: double**
  - Opis: Tablica wskaźników, która przechowuje dane macierzy. Jest to dynamiczna struktura dwuwymiarowa, w której zapisywane są wartości związane z macierzą Laplacjana grafu.

3.4 Struktura Entry

Prototyp struktury:

```
typedef struct entry {
    int index;
    double value;
} Entry;
```

Opis struktury: Struktura `Entry` jest używana do przechowywania pary indeks-wartość. Jest wykorzystywana w analizach wektora własnego, gdzie każdemu wierzchołkowi przypisywana jest wartość własna.

Argumenty struktury:

- **index:**
  - Typ: int
  - Opis: Indeks wierzchołka w grafie, który jest powiązany z wartością własną.
- **value:**
  - Typ: double
  - Opis: Wartość własna powiązana z danym wierzchołkiem, wykorzystywana w obliczeniach spektralnych.

3.5 Struktura SHA256_CTX

Prototyp struktury:

```
typedef struct {
    BYTE data[64];
    WORD datalen;
    unsigned long long bitlen;
    WORD state[8];
} SHA256_CTX;
```

Opis struktury: Struktura `SHA256_CTX` przechowuje stan algorytmu SHA-256 podczas obliczania sumy kontrolnej pliku. Jest to kluczowa struktura w module odpowiedzialnym za obliczanie sumy kontrolnej.

Argumenty struktury:

- **data[64]:**
  - Typ: BYTE
  - Opis: Bufor danych, który przechowuje dane, które zostały przetworzone przez algorytm SHA-256.
- **datalen:**
  - Typ: WORD
  - Opis: Długość danych, które zostały przetworzone.
- **bitlen:**
  - Typ: unsigned long long
  - Opis: Łączna liczba przetworzonych bitów danych.
- **state[8]:**
  - Typ: WORD
  - Opis: Stan algorytmu SHA-256, przechowujący pośrednie wartości podczas obliczeń.

4 Podział na moduły

Program jest podzielony na moduły, z których każdy odpowiada za konkretną część działania aplikacji. Poniżej jest przedstawiony szczegółowy opis każdego modułu:

4.1 flags.c / flags.h

Moduł odpowiedzialny za obsługę argumentów wejściowych i flag. Zawiera:

- Funkcję `flags_error`, która sprawdza poprawność przekazanych argumentów wejściowych,
- Funkcję `flags`, która analizuje argumenty wejściowe i ustawia odpowiednie wartości,
- Deklarację funkcji w nagłówku `flags.h`.

4.2 graph_partition.c / graph_partition.h

Moduł implementujący logikę podziału grafu na grupy. Zawiera:

- Strukturę `Vertex`, która przechowuje informacje o wierzchołkach grafu, takie jak liczba krawędzi, połączenia, grupa, itp. (zdefiniowaną w `graph_partition.h`),

- Funkcję `graph_partioning`, która odpowiada za wybór metody podziału (Kernighan-Lin lub spektralna),
- Zmienną globalną `vertices`, która przechowuje tablicę wierzchołków,
- Deklarację funkcji i zmiennych globalnych w nagłówku `graph_partition.h`.

4.3 graph_utils.c / graph_utils.h

Moduł pomocniczy do manipulacji grafem i grupami wierzchołków. Zawiera:

- Funkcję `remove_cross_group_connections`, która usuwa połączenia między grupami,
- Funkcję `fix_group_connectivity`, która zapewnia spójność grup w grafie,
- Inne funkcje pomocnicze do manipulacji wierzchołkami i grupami, takie jak `find_swap_candidate`
- Deklarację funkcji w nagłówku `graph_utils.h`.

4.4 input_file.c / input_file.h

Moduł odpowiedzialny za wczytywanie danych wejściowych z pliku oraz walidację tych danych. Zawiera:

- Funkcję `read_file`, która wczytuje dane z pliku wejściowego,
- Funkcję `validate_graph_data`, która sprawdza poprawność danych wczytanych z pliku,
- Deklarację funkcji w nagłówku `input_file.h`.

4.5 kl_method.c / kl_method.h

Moduł zawierający implementację algorytmu Kernighan-Lin (KL). Zawiera:

- Funkcję `kernighan_lin_algorithm`, która implementuje algorytm Kernighan-Lin do podziału grafu na dwie grupy,
- Funkcje pomocnicze do wstępnego podziału grafu, obliczania funkcji `D` oraz obliczania `G`,
- Struktury `Swap` i inne pomocnicze dane, które przechowują informacje o zamianach wierzchołków,
- Deklarację funkcji i struktur w nagłówku `kl_method.h`.

4.6 spectral_method.c / spectral_method.h

Moduł zawierający implementację metody spektralnej. Zawiera:

- Funkcję `spectral_partitioning`, która implementuje algorytm podziału grafu przy użyciu metody spektralnej,
- Funkcję `power_iteration`, która oblicza drugi najmniejszy wektor własny macierzy Laplacjana grafu,
- Funkcje pomocnicze do tworzenia i manipulacji macierzą Laplacjana w funkcji `build_laplacian_matrix`,
- Deklarację funkcji i struktur w nagłówku `spectral_method.h`.

4.7 output_file.c / output_file.h

Moduł odpowiedzialny za zapis wyników podziału grafu do pliku wyjściowego. Zawiera:

- Funkcję `write_binary_output` i `write_ascii_output`, które zapisują dane o wierzchołkach w formacie binarnym i ASCII,
- Funkcję `calculate_sha256_checksum`, która oblicza sumę kontrolną SHA256 dla pliku wyjściowego,
- Deklarację funkcji w nagłówku `output_file.h`.

5 Specyfikacja wejścia

Program przyjmuje dane wejściowe w postaci pliku tekstowego o rozszerzeniu `.txt`, który opisuje graf w formie listy sąsiedztwa. Format pliku wejściowego musi być zgodny z poniższym opisem:

- **Pierwsza linia:** Maksymalna możliwa liczba węzłów w wierszu.
- **Druga linia:** Indeksy węzłów w poszczególnych wierszach – liczba wszystkich indeksów odpowiada liczbie węzłów grafu.
- **Trzecia linia:** Wskaźniki na pierwsze indeksy węzłów w liście wierszy z punktu 2.
- **Czwarta linia:** Lista sąsiedztwa węzłów grafu.
- **Piąta linia (i każda kolejna w przypadku grafów niespójnych):** Wskaźniki na pierwsze węzły w poszczególnych grupach z linii 4.

5.1 Przykładowa zawartość pliku wejściowego:

```
4
0; 2; 3; 1; 1; 2
0; 0; 3; 4; 6
0; 1; 1; 2; 3; 2; 3; 5; 3; 4
0; 2; 5; 8; 10
```

5.2 Opis:

- **Pierwsza linia**: Określa maksymalną liczbę wierzchołków w grafie.
- **Druga linia**: Wyszczególnia wszystkie węzły w grafie. Indeksy węzłów są zapisane oddzielone średnikami.
- **Trzecia linia**: Zawiera wskaźniki na pierwsze węzły w poszczególnych wierszach z listy sąsiedztwa.
- **Czwarta linia**: Lista sąsiedztwa węzłów. Każdy numer odpowiada sąsiadowi danego wierzchołka.
- **Piąta linia**: Wskaźniki na pierwsze węzły w grupach w przypadku grafów niespójnych.

6 Specyfikacja wyjścia

Po wykonaniu procesu podziału grafu, program generuje plik wyjściowy, który zawiera wynik podziału. Format pliku wyjściowego zależy od wybranego formatu (ASCII lub binarny), jak określono przez użytkownika podczas uruchamiania programu.

6.1 Format pliku: .bin

6.1.1 Opis formatu pliku

Plik binarny wyjściowy zawiera wyniki podziału grafu na różne podgrafy. Jego celem jest zapisanie w sposób efektywny informacji o przypisaniu wierzchołków do podgrafów, oraz o strukturze tych podgrafów. Plik jest zoptymalizowany pod kątem minimalizacji rozmiaru, przechowując takie niezbędne dane, jak współrzędne wierzchołków, numer podgrafu (grupy), liczba połączeń z innymi wierzchołkami, oraz lista numerów tych wierzchołków.

Plik binarny zawiera następujące sekcje:

- **Endianness** (uint8): 1 bajt – sposób zapisania bitów w pliku:
  - `0x01` – Little Endian
  - `0x00` – Big Endian
- **File ID** (uint32): 4 bajty – identyfikator pliku.
- **Checksum** (uint32): 4 bajty – suma kontrolna obliczona na podstawie danych pliku (SHA-256).
- **Współrzędne wierzchołka** (uint16): 2 bajty na współrzędną, w tym przypadku 2 współrzędne (x, y).
- **Numer podgrafu** (uint16): 2 bajty – numer podgrafu, do którego należy wierzchołek.
- **Liczba połączeń** (uint16): 2 bajty – liczba połączeń z innymi wierzchołkami.
- **Połączenia** (uint16): 2 bajty na każdy numer wierzchołka, z którym dany wierzchołek jest połączony.

Uwaga: numery wierzchołków w Połączeniach nie są dublowane!

Wszystkie dane są zapisane w porządku binarnym, gdzie każda wartość ma stałą długość. Struktura pliku jest zoptymalizowana pod kątem minimalnego rozmiaru oraz efektywnego przechowywania danych.

6.1.2 Struktura pliku

Plik binarny składa się z następujących pól:

Nazwa pola	Wielkość w bitach	Opis
Endianness	1	Sposób zapisania bitowy (1 - little endian, 0 - big endian)
File Id	32	File ID
Checksum	32	Suma kontrolna
X	16	Współrzędna X wierzchołka
Y	16	Współrzędna Y wierzchołka
Group Number	16	Numer podgrafu wierzchołka
Conn Number	16	Liczba połączeń wierzchołka
Connections	16 x n	Połączenia wierzchołka (n to liczba sąsiadów)

6.1.3 Przykład formatu zapisu

Załóżmy, że mamy graf z 3 wierzchołkami, podzielony na 2 podgrafy:
- Wierzchołek 1: współrzędne (2, 3), należy do podgrafu 1, ma 2 połączenia z wierzchołkami 2 i 3. - Wierzchołek 2: współrzędne (4, 5), należy do podgrafu 1, ma 1 połączenie z wierzchołkiem 3. - Wierzchołek 3: współrzędne (6, 7), należy do podgrafu 2, ma 1 połączenie z wierzchołkiem 1.

6.1.4 Przykładowy zapis w formacie binarnym

Nazwa pola	Dane	Opis
Endianness	0x01	Sposób zapisania bitowy (1 - little endian, 0 - big endian)
File Id	0x54554c12	File ID
Checksum	0xc40bb90d	Suma kontrolna
X	00 02	Współrzędna X wierzchołka 1
Y	00 03	Współrzędna Y wierzchołka 1
Group Number	00 01	Numer podgrafu wierzchołka 1
Conn Number	00 02	Liczba połączeń wierzchołka 1
Connections	00 02 00 03	Lista połączeń wierzchołka 1

6.2 Format pliku: .txt

6.2.1 Opis formatu pliku

Plik tekstowy zawiera dane o wierzchołkach grafu. Każdy wierzchołek wraz z informacjami zapisywany jest na oddzielnej linii. Format ten zawiera jedynie wartości liczbowe

typu int, oddzielone średnikami, co czyni go bardziej kompaktowym.

Plik zawiera następujące informacje o grafie i dla każdego wierzchołka:

- **Liczba wierzchołków**: Całkowita liczba wierzchołków w grafie.
- **Liczba podgrafów**: Całkowita liczba podgrafów w grafie.
- **Współrzędne wierzchołka**: Współrzędne (X, Y) wierzchołka.
- **Numer podgrafu**: Numer podgrafu, do którego należy wierzchołek.
- **Liczba połączeń**: Liczba połączeń wierzchołka z innymi wierzchołkami.
- **Połączenia**: Numery wierzchołków, z którymi dany wierzchołek jest połączony.

Uwaga: numery wierzchołków w Połączeniach są dublowane!

6.2.2 Struktura pliku

Plik tekstowy ma następującą strukturę:

```
1 [Liczba wierzcholkow]
2 [Liczba podgrafow]
3 [Wspolrzedna X wierzcholka 1];[Wspolrzedna Y wierzcholka 1];[Numer
     podgrafu];[Liczba polaczen wierzcholka 1];[Polaczenia]
4 [Wspolrzedna X wierzcholka 2];[Wspolrzedna Y wierzcholka 2];[Numer
     podgrafu];[Liczba polaczen wirzcholka 2];[Polaczenia]
5 ...
```

6.2.3 Przykład formatu zapisu

Załóżmy, że mamy graf z 3 wierzchołkami, podzielony na 2 podgrafy:
- Wierzchołek 1: współrzędne (2, 3), należy do podgrafu 1, ma 2 połączenia z wierzchołkami 2 i 3.
- Wierzchołek 2: współrzędne (4, 5), należy do podgrafu 1, ma 1 połączenie z wierzchołkiem 3.
- Wierzchołek 3: współrzędne (6, 7), należy do podgrafu 2, ma 1 połączenie z wierzchołkiem 1.

6.2.4 Przykładowy zapis w formacie tekstowym:

```
1 3
2 2
3 2;3;1;2;2;3
4 4;5;1;1;3
5 6;7;2;1;1
```

7 Obsługa błędów

Program wykrywa i obsługuje następujące błędy:

1. **Liczba części na które użytkownik chce podzielić graf jest mniejsza niż 2.**
   Komunikat: **"Błąd: liczba części musi być większa lub równa 2."**
   Działanie: Program prosi o ponowne wpisanie liczby części na wartość >=2. W przypadku podania liczby mniejszej niż 2, podział grafu jest niemożliwy.
   Kod błędu: **10**

2. **Brakujący parametr wywołania**
   Komunikat: **"Błąd: parametry wywołania są niewystarczające, aby uruchomić program."**
   Działanie: Program przerywa działanie. Użytkownik musi ponownie uruchomić program, podając wszystkie wymagane parametry wywołania, zgodnie z dokumentacją.
   Kod błędu: **11**

3. **Nieistniejący parametr**
   Komunikat: **"Błąd: nieznany parametr."**
   Działanie: Program przerywa działanie. Użytkownik musi sprawdzić dostępne parametry w dokumentacji i dostosować je w ponownym uruchomieniu programu.
   Kod błędu: **12**

4. **Niepoprawny format pliku wejściowego**
   Komunikat: **"Błąd: Niepoprawny format pliku wejściowego."**
   Działanie: Program przerywa działanie, gdy plik wejściowy nie spełnia określonych wymagań strukturalnych, takich jak:

   - Niepoprawna liczba wierszy,
   - Nieodpowiednia zawartość w wierszach (np. błędne wartości w pierwszej linii),
   - Nieprawidłowe współrzędne lub indeksy.

   Użytkownik musi poprawić plik, zgodnie z wymaganiami programu.
   Kod błędu: **13**

5. **Błędne dane wejściowe**
   Komunikat: **"Błąd: Bledne dane wejściowe."**
   Działanie: Program przerywa działanie w przypadku wykrycia błędów w danych wejściowych, takich jak błędne wartości flag lub niezgodność z wymaganym formatem. Należy sprawdzić poprawność danych.
   Kod błędu: **14**

6. **Błąd pamięci**
   Komunikat: **"Błąd: pomyłka pamięci."**
   Działanie: Program przerywa działanie z powodu problemów związanych z alokacją pamięci. Użytkownik powinien upewnić się, że system ma wystarczająco dużo dostępnej pamięci.
   Kod błędu: **15**

7. Wierzchołek został bez połączeń

Komunikat: **"Błąd: Wierzchołek %d został bez połączeń (nie można usunąć wszystkich połączeń)."**

Działanie: Po podziale grafu, jeśli wierzchołek nie ma połączeń, traktowany jest jako oddzielny graf. W takim przypadku użytkownik powinien spróbować użyć flagi '–force'. Jeśli to nie pomoże, podział grafu jest niemożliwy.

Kod błędu: **16**

8. Metoda KL wspiera tylko podział na 2 grupy

Komunikat: **"Błąd: Metoda KL wspiera tylko podział na 2 grupy."**

Działanie: Program przerywa działanie, jeśli wybrana metoda Kernighan-Lin jest używana do podziału grafu na więcej niż 2 grupy. W takim przypadku należy wybrać inną metodę (np. spektralną).

Kod błędu: **17**

9. Graf jest zbyt mały, aby go podzielić

Komunikat: **"Błąd: Graf jest zbyt mały, aby można go było podzielić."**

Działanie: Program wymaga minimum 4 wierzchołków w grafie, aby wykonać podział. Jeśli graf zawiera mniej niż 4 wierzchołki, program zakończy działanie z tym błędem.

Kod błędu: **18**

10. Błąd podczas obliczania wartości i wektorów własnych

Komunikat: **"Błąd podczas obliczania wartości i wektorów własnych."**

Działanie: Program przerywa działanie, jeśli algorytm nie może obliczyć wartości własnych lub wektorów własnych macierzy Laplacjana grafu.

Kod błędu: **19**

11. Zbyt duża liczba podgrafów

Komunikat: **"Błąd: Zbyt duża liczba podgrafów."**

Działanie: Program wspiera podział grafu na połowę lub mniej części liczby wierzchołków. Jeśli liczba podgrafów jest zbyt duża, należy zmniejszyć jej wartość.

Kod błędu: **20**

12. Podział nie może być wykonany bez marginesu błędu

Komunikat: **"Błąd: Podział nie może być wykonany bez marginesu błędu. Zbyt mała liczba wierzchołków na podgraf."**

Działanie: Jeśli margines błędu jest ustawiony na 0, program sprawdzi, czy podział jest możliwy. W przypadku zbyt małej liczby wierzchołków w grafie, podział nie będzie możliwy bez większego marginesu błędu.

Kod błędu: **21**

13. Za mały margines błędu

Komunikat: **"Błąd: Za mały margines błędu, by dokonać prawidłowego podziału."**

Działanie: Program zakończy działanie, jeśli margines błędu jest zbyt mały, aby prawidłowo podzielić graf. Użytkownik może zwiększyć margines błędu lub użyć flagi '–force'.

Kod błędu: **22**

14. Nie udało się wybrać danego grafu

Komunikat: **"Błąd: Nie udało się wybrać danego grafu."**

Działanie: Jeśli plik wejściowy zawiera wiele grafów, użytkownik musi wskazać, który z nich ma zostać wybrany. Flaga ‘–graph_index‘ może pomóc w tym przypadku.

Kod błędu: **23**

15. Wykryto większą ilość grafów w pliku

Komunikat: **"Błąd: Wykryto większą ilość grafów w pliku. Zdefiniuj, z którego korzystasz."**

Działanie: Jeśli plik wejściowy zawiera więcej niż jeden graf, użytkownik musi określić, który z nich ma zostać wybrany, używając flagi ‘–graph_index‘.

Kod błędu: **24**

16. Nie można wybrać grafu

Komunikat: **"Błąd: Nie można wybrać grafu."**

Działanie: Jeśli plik wejściowy zawiera tylko jeden graf, flaga ‘–graph_index‘ jest niepotrzebna i powinna zostać usunięta.

Kod błędu: **25**

17. Program nie wykrył takiego grafu

Komunikat: **"Błąd: Program nie wykrył takiego grafu."**

Działanie: Jeśli plik wejściowy zawiera kilka grafów, użytkownik powinien upewnić się, że podał poprawny indeks grafu za pomocą flagi ‘–graph_index‘.

Kod błędu: **26**