Specyfikacja funkcjonalna aplikacji do podziału grafu

Emilia Urbanek i Mikołaj Frąckowiak

03.03.2025

1 Cel projektu

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. Program działa w trybie wsadowym i przyjmuje wszystkie parametry z linii poleceń. Wynikiem działania programu jest plik tekstowy lub binarny-zależnie od wyboru użytkownika - zawierający dane wejściowe i wynik podziału grafu.

2 Teoria

Program realizuje podział grafu w sposób rekurencyjny. W każdej iteracji sprawdzana jest możliwość podziału grafu (lub jego części) na dwa podgrafy, dla których różnica liczby wierzchołków mieści się w podanym marginesie procentowym. Jeśli podział jest możliwy, proces jest kontynuowany na uzyskanych podgrafach, aż do uzyskania zadanej liczby części. Kryterium decydującym o poprawnym podziale jest wzór:

$$\frac{|a-b|}{\left(\frac{a+b}{2}\right)} <= N \tag{1}$$

gdzie a i b oznaczają liczby wierzchołków w dwóch uzyskanych podgrafach, a N oznacza wskazany margines. Program minimalizuje liczbę przeciętych krawędzi, dążąc do podziału zgodnego z podanym marginesem.

3 Kompilacja programu

Program będzie kompilowany za pomocą Makefile, co zapewni automatyzację procesu budowy i optymalizację kompilacji poprzez rekompilację tylko zmienionych plików. Dzięki temu użytkownik może łatwo skompilować program jednym poleceniem (make), a także korzystać z dodatkowych funkcji, takich jak czyszczenie plików tymczasowych (make clean), czy przetestowanie określonej w Makefile funkcji(make test).

4 Dane wejściowe

Plik wejściowy- wskazany za pomocą flagi -i filename, zawiera opis grafu, który program będzie dzielił na części. Format tego pliku musi spełniać pewne wymagania, aby program mógł poprawnie przetworzyć dane. Plik wejściowy będzie miał następujący format:
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.

Opis formatu pliku .txt:

- **Sekcja 1: Mapa grafu** - przedstawia tabelę '0' i '1', które określają istnienie - '1' (lub nie - '0') węzła w danej komórce.
- **Sekcja 2: Połączenia w grafie** - w kolejnych wierszach zawiera istniejące krawędzie grafu.

Plik wyjściowy- wskazany za pomocą flagi -o filename, zawiera wynik podziału grafu na określoną liczbę części. Możliwe są dwa formaty:
Format tekstowy (domyślny) – W wyniku działania programu w pliku tekstowym zapisana jest w pierwszej linijce liczba udanych podziałów grafu, a następnie graf(opisany w taki sam sposób jak w pliku wejściowym).
Format binarny – Jeśli użytkownik poda flagę -b, wynik zostanie zapisany w formacie binarnym-wynikiem będzie jedynie podzielony graf. Format ten jest bardziej kompaktowy i nadaje się do przetwarzania przez inne aplikacje, które oczekują danych w takim formacie.
Struktura pliku binarnego:

Plik binarny zapisuje graf w zmodyfikowanym formacie **CSRRG**, bazującym na strukturze **CSRR-G** opisanej wcześniej. Kolejne sekcje pliku to:

1. `max_vertices` (liczba całkowita)
2. `col_index` (tablica indeksów kolumn)
3. `row_ptr` (tablica wskaźników początku grup)
4. `neighbors_out` (nowo wygenerowane grupy sąsiadów, z zachowaniem zasad: zapisywane są tylko krawędzie, dla których numer sąsiada jest większy od numeru wierzchołka; każda niepusta grupa zaczyna się od własnego indeksu wierzchołka)
5. `row_ptr_out` (tablica wskaźników dla `neighbors_out`)

Wszystkie dane są zapisane binarnie jako liczby typu `int` (4 bajty). Format nie zawiera separatorów ani znaków końca linii. //[0.5]

Przykład grafu w formacie CSRRg i binarnym

Rozważmy prosty graf składający się z dwóch wierzchołków (0 i 1) połączonych jedną krawędzią. Reprezentacja tego grafu w formacie CSRRg (Compressed Sparse Row for Regular Graphs) wygląda następująco:

Przykład minimalnego grafu w formacie CSRRg i binarnym

Rozważmy prosty graf składający się z dwóch wierzchołków (0 i 1) połączonych jedną krawędzią. Reprezentacja tego grafu w formacie CSRRg (Compressed Sparse Row for Regular Graphs) wygląda następująco:

```
2
1;0
0;1;2
0;1
0;2
```

- **Linia 1** — liczba wierzchołków: `2`
- **Linia 2** — kolumny sąsiedztwa (col_index): `1;0`
- **Linia 3** — wskaźniki wierszy (row_ptr): `0;1;2`
- **Linia 4** — sąsiedzi wierzchołków (neighbors): `0;1`
- **Linia 5** — wskaźniki grup sąsiadów (row_ptr dla neighbors): `0;2`

Format binarny:
Plik binarny zapisuje dane w kolejności dokładnie odpowiadającej pięciu sekcjom formatu CSRRg. Wszystkie wartości są zapisywane jako 4-bajtowe liczby całkowite typu `int` w formacie **little-endian**. W rzeczywistym pliku wszystkie znaki znajdować się będą w jednej linijce bez oddzielających linijek. Dla powyższego przykładu dane będą zapisane w binarnej postaci jako:

```
02 00 00 00    // max_vertices = 2

01 00 00 00    // col_index[0] = 1
00 00 00 00    // col_index[1] = 0

00 00 00 00    // row_ptr[0] = 0
01 00 00 00    // row_ptr[1] = 1
02 00 00 00    // row_ptr[2] = 2

00 00 00 00    // neighbors[0] = 0
01 00 00 00    // neighbors[1] = 1

00 00 00 00    // row_ptr_neighbors[0] = 0
02 00 00 00    // row_ptr_neighbors[1] = 2
```

Plik binarny nie zawiera żadnych separatorów między sekcjami ani długości segmentów – zakłada się, że dane są czytane w ustalonej kolejności zgodnie z wcześniej opisanym schematem CSRRg.
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%.

Dodatkowe parametry podawane są jako argumenty wywołania programu.

5 Argumenty wywołania programu

Program przyjmuje następujące argumenty w linii poleceń:

- `-i input_filename` - plik wejściowy zawierający opis grafu;
- `-o output_filename` - plik wyjściowy z podziałem grafu;
- `-p N` - liczba przecięć, aby podzielić graf (domyślnie 1);
- `-m M` - maksymalny procentowy margines różnicy liczby wierzchołków między częściami (domyślnie 10%);
- `-b` - jeśli podano, wynik zostanie zapisany w formacie binarnym(podzielony graf). Domyślnie zapisane będzie w formacie tekstowym-jeśli nie podano flagi
- `-t`-jeśli podano, wynik zostanie wypisany w terminalu(liczba udanych przecięć grafu, podzielony graf w taki samym formacie jak w pliku wejściowym)

6 Przykładowe wywołanie programu

```
./graph_partition -i graph.csrrg -o wynik -p 3 -t
```

Powyższe polecenie podzieli graf zapisany w pliku `graph.csrrg` 3 razy, zapewniając, że liczba wierzchołków w każdej części nie różni się o więcej niż 10%. Wypisze wynik w terminalu oraz zapisze do pliku `wynik.csrrg`.

7 Komunikaty błędów

W przypadku błędnych danych wejściowych program wyświetla komunikaty diagnostyczne, np.:

- `Błąd: Brak wymaganych argumentów` -nie podano nazwy pliku wejściowego lub/i wyjściowego- następnie wyświetlana jest funkcja wypisująca wymagane argumenty;
- `Błąd: Niepoprawny format pliku wejściowego` - plik wejściowy nie spełnia wymagań;
- `Błąd: Liczba części przekracza liczbę wierzchołków` - użytkownik podał zbyt dużą liczbę przecięć;
- `Błąd: Nie można otworzyć pliku` - podany plik nie istnieje;
- `Błąd otwierania pliku do zapisu` - nie można otworzyć podanego pliku
- `Błąd wczytywania grafu`- nie udało się wczytać grafu.
- `Błąd: Margines musi być w zakresie 0-100%` -źle podany margines.
- `Błąd: Liczba przecięć musi być >= 1`- źle podana liczba przecięć.
- `Błąd: Nie udało się wykonać podziału x` - nie udało się wykonać x podziału.

8 Podsumowanie

Aplikacja zapewnia elastyczny podział grafu na części, umożliwiając użytkownikowi kontrolę nad liczbą części oraz różnicami w liczbie wierzchołków. Pliki wynikowe mogą być wykorzystane do dalszej analizy.

Dokumentacja implementacyjna aplikacji do podziału grafu

Emilia Urbanek i Mikołaj Frąckowiak

31.03.2025

1 Wstęp

Celem niniejszej dokumentacji jest przedstawienie szczegółów implementacyjnych programu realizującego podział nieskierowanego grafu na dwie spójne części. Aplikacja została zaimplementowana w języku C i działa w trybie wsadowym. Dokument zawiera opis sposobu operowania na danych wejściowych, przekształcenia formatu grafu, zastosowanych algorytmów oraz metody podziału zbioru wierzchołków.

2 Założenia projektowe

Podczas projektowania aplikacji przyjęto szereg założeń, które ułatwiły implementację i pozwoliły skupić się na efektywnym podziale grafu. Przyjęte ograniczenia oraz struktura danych determinują sposób działania algorytmów i wpływają na wydajność programu.

Struktura grafu

Założono, że graf wejściowy:

- jest grafem nieskierowanym,
- jest reprezentowany w formacie CSR (*Compressed Sparse Row*)
- nie zawiera informacji o wagach krawędzi – wszystkie połączenia są traktowane jako równoważne.

Podział grafu

Głównym celem aplikacji jest podział zbioru wierzchołków na dwie możliwie równe, spójne części. W związku z tym:

- dopuszczalna jest minimalna różnica liczby wierzchołków między grupami (domyślnie 10%),
- każda z wygenerowanych grup musi tworzyć graf spójny – tzn. dla każdego wierzchołka w grupie istnieje ścieżka do każdego innego wierzchołka w tej samej grupie,
- podział oparty jest na eksploracji grafu z punktu startowego wyznaczanego za pomocą algorytmu Dijkstry,
- przypisanie wierzchołków do grup realizowane jest metodą przeszukiwania w głąb (DFS), ze śledzeniem liczebności każdej grupy w czasie działania.

Zachowanie aplikacji

Aplikacja działa w trybie wsadowym i:

- oczekuje pliku wejściowego w ustalonym formacie,
- samodzielnie przetwarza dane, bez potrzeby interakcji z użytkownikiem w trakcie działania,
- oferuje zapis wyników w dwóch formatach – tekstowym lub binarnym ,

Założenia te pozwoliły uprościć projekt aplikacji i skupić się na kluczowej funkcjonalności, czyli efektywnym i poprawnym podziale grafu na dwie spójne części.

3 Obsługa danych wejściowych

Dane wejściowe wczytywane są 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.

Na podstawie danych wejściowych dokonywana jest konwersja do listy sąsiedztwa, co ułatwia dalsze operacje przeszukiwania oraz podziału grafu.

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 Zapis danych wyjściowych

Użytkownik ma możliwość wyboru formatu zapisu danych wyjściowych:

- **Plik tekstowy** – dane zapisywane są w postaci czytelnej dla człowieka, w formacie jak dane wejściowe(.csrrg)
- **Plik binarny** – dane są serializowane w sposób optymalny pod względem rozmiaru oraz czasu wczytywania w przyszłości. Składa się z dwóch głównych sekcji: nagłówka oraz listy wierzchołków w każdej części.

Struktura plików wyjściowych została zaprojektowana tak, aby możliwe było dalsze wykorzystanie danych w analizach lub innych narzędziach pracujących na grafach.

6 Podsumowanie

Aplikacja implementuje pełny cykl przetwarzania danych grafowych – od wczytania grafu w formacie CSR, poprzez jego przekształcenie, podział na dwie spójne części, aż po zapis wyników do pliku. Wykorzystanie algorytmu Dijkstry i DFS pozwala na zapewnienie efektywności i poprawności działania programu.

Dokumentacja końcowa aplikacji do podziału grafu

Emilia Urbanek i Mikołaj Frąckowiak

28.04.2025

1 Wprowadzenie

Dokumentacja implementacyjna opisuje podział projektu na pliki, funkcje oraz struktury danych użyte do realizacji aplikacji do podziału grafu. Program, zgodnie ze specyfikacją funkcjonalną, działa w trybie wsadowym, przyjmuje wszystkie parametry z linii poleceń (`-i`, `-o`, `-p`, `-m`, `-b`, `-t`) i zwraca wynik w postaci pliku tekstowego lub binarnego. Implementacja odbywa się w języku C, a podstawowym algorytmem podziału jest hybryda DFS (Depth-First Search) z Dijkstrą, gdzie Dijkstra służy do wyboru sensownego punktu startowego, a DFS zapewnia spójność części grafu.

2 Podział projektu na pliki

Projekt został podzielony na kilka modułów.

Pliki źródłowe:

- **main.c**
  Główna funkcja programu, która:
  - Parsuje argumenty linii poleceń zgodnie z funkcjonalną specyfikacją.
  - Inicjalizuje konfigurację aplikacji.
  - Wywołuje funkcje z modułów wejściowych, podziału i wyjściowych.
- **graf.c / graf.h**
  Moduł odpowiedzialny za reprezentację grafu oraz operacje na nim. Zawiera:
  - Definicję struktury `Graph`
  - Funkcję `load_graph_from_file()`, która wczytuje graf z pliku wejściowego (obsługa formatu `.csrrg` lub `.txt`).
  - Funkcje pomocnicze, np. `print_graph()` do debugowania.
- **partitions.c / partitions.h**
  Moduł implementujący algorytm podziału grafu z wykorzystaniem hybrydy Dijkstra + DFS. Zawiera:
  - Funkcję `find_center(Graph *graph)` – wykorzystuje algorytm Dijkstry do wyznaczenia centralnego wierzchołka (punktu startowego).
  - Funkcję `dfs()` – rekurencyjną implementację DFS, która przydziela wierzchołki do kolejnych części, zapewniając spójność.
  - Funkcję `dfs_partition(Graph *graph, int num_parts, int max_diff, int *partition)` – główną funkcję realizującą podział grafu, w której:
    * Najpierw wybierany jest wierzchołek startowy za pomocą Dijkstry.
    * Następnie wykonywany jest DFS od tego punktu, przydzielający wierzchołki do części.
    * Dodatkowo wykonywane są sprawdzenia spójności i ewentualne korekty podziału.
- **output.c / output.h**
  Moduł zajmujący się zapisem wyników podziału grafu. Zawiera:

– Funkcję `save_graph_to_csrrg(Graph *graph, const char *filename)` – zapis wyniku w formacie tekstowym.

– Funkcję `void create_neighbors_and_row_ptr_filtered(Graph *graph, int **neighbors_out, int **row_ptr_out);` -funkcja pomocnicza do zapisu tekstowego.

– Funkcję `save_to_binary(Graph *graph, int *partition, const char *filename)` – zapis wyniku w formacie binarnym.

– Funkcję `print_partition_terminal(Graph *graph, int successful_cuts)`-wypisanie wyniku na terminalu.

Makefile:

Plik Makefile służy do automatyzacji kompilacji projektu:

- Kompilacja wszystkich modułów za pomocą polecenia `make`.
- Obsługa poleceń `make clean` (usuwanie plików tymczasowych) oraz `make test` (test przykładowego małego grafu z określonymi parametrami wywołania).

3 Opis struktur i funkcji

3.1 Plik `graf.h`

Plik `graf.h` zawiera definicję głównej struktury danych służącej do reprezentacji grafu.

Struktury

- **Graph**
  Struktura reprezentująca graf, zawierająca:
  – `max_vertices` – maksymalna liczba wierzchołków w grafie,
  – `num_vertices` – aktualna liczba wierzchołków,
  – `neighbors` – dynamiczna tablica sąsiadów dla każdego wierzchołka (lista sąsiedztwa),
  – `neighbor_count` – tablica przechowująca liczbę sąsiadów dla każdego wierzchołka,
  – `max_distances` – tablica maksymalnych odległości (użyteczność opcjonalna),
  – `group_assignment` – przypisanie wierzchołków do grup,
  – `component` – identyfikacja przynależności wierzchołków do składowych spójnych,
  – `num_components` – liczba składowych spójnych w grafie,
  – `col_index` – kolumnowy indeks sąsiadów w reprezentacji CSR (Compressed Sparse Row),
  – `row_ptr` – wskaźniki na początki wierszy w reprezentacji CSR,
  – `group_list` – lista wierzchołków przypisanych do grup,
  – `group_ptr` – wskaźniki na początki grup w tablicy `group_list`.

3.2 Funkcje modułu wejściowego (graph.c)

- `void load_graph_from_file(Graph *graph, const char *filename);`
  Wczytuje graf z pliku zgodnie z opisem formatu wejściowego.
- `void print_graph(Graph *graph);`
  Debugowanie – wypisuje zawartość grafu.

3.3 Funkcje konwersji grafu na listę sąsiedztwa

- `void init_graph(Graph *graph;`
  inicjacja struktury grafu.
- `void add_edge(Graph *graph, int u, int v);`
  Dodanie krawędzi (graf nieskierowany) z dynamiczną alokacją listy.
- `void convert_csr_to_neighbors(Graph *graph);`
  Konwersja danych CSR do dynamicznej listy sąsiedztwa.
- `void dfs(const Graph *graph, int v, bool visited[]);`
  DFS na dynamicznej liście sąsiadów.
- `bool is_connected(const Graph *graph);`
  Funkcja pomocniczna-sprawdza połączenia.
- `void free_graph(Graph *graph);`
  Zwalnianie całej pamięci grafu.

3.4 Plik `partitions.c`

Plik `partitions.c` zawiera implementację funkcji związanych z analizą składowych spójnych grafu, algorytmem Dijkstry, sprawdzaniem spójności, dzieleniem grafu na grupy oraz balansowaniem tych grup.

Funkcje

- **dfs_mark(Graph *graph, int v, bool visited[], int component[], int current_component)**
  Rekurencyjna funkcja DFS oznaczająca wierzchołki należące do tej samej składowej spójnej.
- **find_connected_components(Graph *graph)**
  Wyszukuje wszystkie składowe spójne w grafie i przypisuje odpowiednie identyfikatory składowych do wierzchołków.
- **dijkstra(Graph *graph, int start)**
  Oblicza najkrótsze odległości od wierzchołka startowego do pozostałych wierzchołków przy pomocy algorytmu Dijkstry (dla nieważonych grafów).
- **is_component_connected(Graph *graph, const bool in_component[])**
  Sprawdza, czy zbiór wierzchołków tworzy spójną składową w grafie.
- **balance_groups(Graph *graph, int group1[], int group1_size, int group2[], int group2_size, int margin)**
  Balansuje rozmiary dwóch grup wierzchołków, starając się zachować ich spójność i minimalizować różnicę rozmiarów w granicach dopuszczalnego marginesu.
- **partition_graph(Graph *graph, int margin)**
  Główna funkcja odpowiedzialna za podział grafu na dwie grupy. Znajduje centrum składowej, przypisuje wierzchołki do grup oraz balansuje grupy w razie potrzeby.
- **split_graph(Graph *graph)**
  Aktualizuje strukturę grafu po podziale na grupy: przypisuje nowe składowe oraz usuwa krawędzie pomiędzy grupami.

3.5 Funkcje modułu wyjściowego (output.c)

- `void save_graph_to_csrrg(Graph *graph, int *partition, const char *filename);`
  Zapisuje wynik podziału w formacie tekstowym – zgodnie z wejściowym formatem grafu.
- `void save_to_binary(Graph *graph, int *partition, const char *filename);`
  Zapisuje wynik podziału w formacie binarnym – zgodnie z opisem w specyfikacji.
- `print_partition_terminal(Graph *graph, int successful_cuts)`-wypisuje wynik,poprzedzony liczbą poprawnych przecięć, w formacie tesktowym(jak w pliku wejściowym) na terminalu.

4 Podejście do rozwiązania problemu podziału grafu

1. **Wczytanie danych:**
   Program uruchamiany z linii poleceń pobiera nazwę pliku wejściowego (np. `-i graph.csrrg`) oraz inne parametry. Moduł `graph.c` przetwarza plik i tworzy strukturę `Graph`.

2. **Podział oryginalnego grafu na składowe:**
   Wykorzystujemy funkcje zawarte w pliku `partitions.c`, aby podzielić graf na spójne *kompartmenty*.

3. **Przecięcia:**
   Wywołujemy w pętli funkcję podziału zwracającą sukces bądź porażkę i w zależności od wyniku kontynuujemy dalsze przecięcia.

4. **Algorytm podziału:**

   (a) **Wybór centralnego wierzchołka:**
   Za pomocą algorytmu Dijkstry wybieramy centralny wierzchołek danej składowej. Następnie, zaczynając od niego i korzystając z DFS-a faworyzującego najbardziej oddalonych sąsiadów, wybieramy pierwszą grupę wierzchołków. Druga grupa składa się z pozostałych wierzchołków.

   (b) **Bilans grup:**
   Rozpoczynamy od sprawdzenia spójności drugiej grupy. W przypadku niespełnienia tego warunku znajdujemy największą spójną składową, a pozostałe wierzchołki przenosimy do grupy pierwszej. Jeśli margines nie jest spełniony, rozpoczynamy proces pojedynczego sprawdzania. Zgodnie z zastosowanym algorytmem analizujemy, czy możemy przenieść dany wierzchołek do grupy drugiej, aż do momentu osiągnięcia marginesu lub zakończenia procesu niepowodzeniem.

   (c) **Zapis zmian:**
   W przypadku uzyskania podziału spełniającego wymagania zapisujemy zmiany. W przeciwnym wypadku kontynuujemy z następnym kompartmentem.

5. **Zapis wyniku:**
   Po uzyskaniu ostatecznego podziału grafu, moduł `output.c` zapisuje wynik do pliku wyjściowego w formacie tekstowym lub binarnym, zgodnie z parametrem `-b`.

5 Podsumowanie

Zastosowana metoda pozwala na sprawne i głównie niezawodne podejście do podziału grafu. Utworzony algorytm pozwala na znaczne ograniczenie pola poszukiwań, jednak w niektórych specyficznych przypadkach może prowadzić do nieodnalezienia rozwiązania. Dodatkowym problemem jaki może się ukazać w bardzo dużych i gęstych grafach jest brak limitu interakcji w części szukania możliwych do przeżucenia wierzchołków. W wyjątkowo niefortunnych przypadkach może to prowadzić do długiego wykonywania programu.

6 Repozytorum GitHub

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