

JIMP2 Projekt 2025

Dokumentacja implementacyjna - C

Michał Ludwiczak
GR3

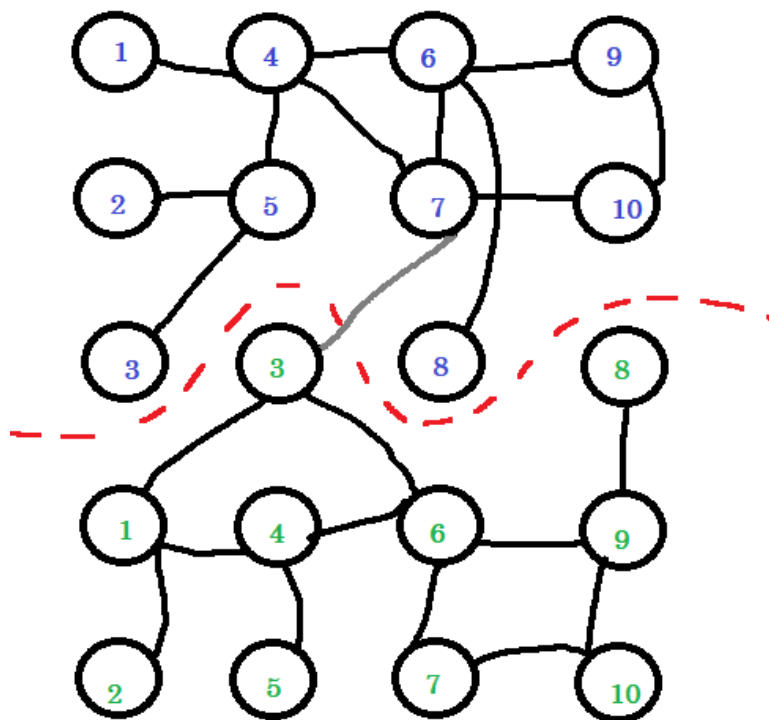
8 kwietnia 2025

Spis treści

1	Cel projektu	1
2	Algorytm	2
2.1	Macierz Laplace’a	3
2.2	Wektory własne	3
2.3	Klasteryzacja	3
2.4	Wynik	3
3	Flagi i argumenty	4
4	Szczegóły implementacyjne	4
4.1	Struktura plików	4
4.2	Moduły	5
4.3	Struktury	6
4.4	Najważniejsze funkcje	7
4.5	Pliki wejściowe i wyjściowe	11

1 Cel projektu

Celem projektu jest stworzenie aplikacji w języku C, dokonującej podziału grafu na określoną przez użytkownika lub domyślną liczbę 2 równych części z zachowaniem wybranego lub, ponownie, domyślnego 10-procentowego marginesu różnicy. Liczba wierzchołków w powstałych częściach grafu nie powinna się różnić o więcej niż zadany margines procentowy, a liczba przeciętych krawędzi pomiędzy wynikowymi częściami grafu powinna być jak najmniejsza. Wyjściem programu ma być plik tekstowy lub binarny. Użytkownik ma mieć możliwość wskazać wyjście programu, a więc plik tekstowy lub binarny, zawierający wynik działania programu, oraz dane wejściowe, które mogą być wykorzystane ponownie w kolejnym działaniu programu.



Rysunek 1: Przykładowy graf podzielony na 2 równe części

2 Algorytm

Niestety znalezienie optymalnego podziału grafu należy do klasy problemów NP-trudnych. Dla grafu o n wierzchołkach istnieje aż $2^{n-1} - 1$ możliwych sposobów na bisekcję (wyrażonego podziału na 2 grafy). [1] Przy tego typu problemie nie możemy po prostu sprawdzić wszystkich możliwości i wybrać najlepszej z nich - jest to problem optymalizacji kombinatorycznej. W związku z tym opracowano wiele algorytmów zachłannych, metod przybliżonych i heurystyk, które pozwalają na uzyskanie satysfakcjonujących rozwiązań w praktyce. W naszym programie zdecydowaliśmy się użyć algorytmu spektralnego, który wykorzystuje własności spektralne macierzy Laplace'a grafu. Na podstawie spektrum grafu, które opisuje strukturę jego strukturę, graf możemy podzielić w efektywny i satysfakcjonujący sposób, minimalizując liczbę przeciętych krawędzi.

2.1 Macierz Laplace’a

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

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

gdzie:

- L to **macierz Laplace’a** grafu,
- D to **macierz stopni** to macierz diagonalna, w której elementy na diagonalu d_{ii} odpowiadają stopniowi wierzchołka i , czyli liczbie krawędzi, które są z nim bezpośrednio połączone (w przypadku pętli krawędź liczy się podwójnie)
- A to **macierz sąsiedztwa** grafu, gdzie elementy a_{ij} są równe 1, jeśli istnieje krawędź między wierzchołkami i i j , oraz 0 w przeciwnym przypadku.

2.2 Wektory własne

Następnym krokiem jest obliczenie $k = p - 1$ (gdzie p to liczba części, na które graf ma być podzielony) najmniejszych wektorów własnych, nie wliczając pierwszego zerowego. Dla bisekcji będzie to tylko jeden wektor, drugi najmniejszy - wektor Fiedlera. Wektory własne macierzy Laplace’a grafu przechowują informacje o połączeniach pomiędzy wierzchołkami. Istnieje metoda Lanczosa [3], którą można zastosować na macierzy Laplace’a, gdyż jest to macierz symetryczna, a więc również i hermitowska. Generuje ona ortonormalną bazę przestrzeni Kryłowa oraz przekształca macierz Laplace’a do macierzy trójdzielnej T . Z wartości i wektorów własnych tej uproszczonej macierzy i bazy Kryłowa można obliczyć przybliżenia wartości i wektorów własnych oryginalnej macierzy.

2.3 Klasteryzacja

Po otrzymaniu macierzy zawierającej wartości k wektorów własnych należy podzielić graf na p części. Te wartości są traktowane jako punkty w przestrzeni k -wymiarowej odpowiadające poszczególnym wierzchołkom. Stosuje się algorytm klasteryzacji, aby podzielić te wierzchołki na grupy, a tym samym podzielić graf na części. W naszym programie algorytmem klasteryzacji jest centroidów / k -średnich (k -means) [4]. Wierzchołki w tych samych częściach są ze sobą bardziej powiązane niż te, które są w innych częściach, co gwarantuje satysfakcjonujący podział.

2.4 Wynik

Po otrzymaniu poszczególnych części z wierzchołkami modyfikuje się macierz sąsiedztwa A wstawiając 0 w pola oznaczające krawędzie pomiędzy wierzchołkami przynależącymi do różnych grup. W ten sposób otrzymuje się nową macierz sąsiedztwa, która zawiera już podzielony graf. Macierz sąsiedztwa jest już gotowa do przetworzenia i wypisania na plik wyjściowy.

3 Flagi i argumenty

Program jest uruchamiany z linii poleceń i obsługuje następujące flagi:

<plik_wejściowy>

input

Ścieżka do pliku wejściowego zawierającego opis grafu.

Wymagane jako pierwszy argument

-p <liczba_części>

parts

Określa liczbę części, na które ma zostać podzielony graf.

Domyślnie: 2

-m <margines>

margin

Określa dopuszczalny margines procentowy różnicy w liczbie wierzchołków między częściami.

Domyślnie: 10%

-o <plik_wyjściowy>

output

Określa ścieżkę do pliku wyjściowego, w którym zostaną zapisane wyniki.

Domyślnie: output.txt lub output.bin (w zależności od formatu)

-f <format_pliku_wyjściowego>

format

Określa format pliku wyjściowego: **txt** dla pliku tekstowego lub **bin** dla pliku binarnego.

Domyślnie: txt

4 Szczegóły implementacyjne

4.1 Struktura plików

W projekcie obowiązuje poniższa struktura plików:

JIMP2-Projekt-2025-C/

```
├── src/
├── include/
├── input/
├── output/
├── docs/
├── obj/
├── Makefile
└── .gitignore
```

- **src/**

Przechowywanie plików źródłowych (.c).

- **include/**
Przechowywanie plików nagłówkowych (.h).
- **input/**
Przechowywanie plików wejściowych (.csrrg).
- **output/**
Przechowywanie plików wyjściowych (.txt, .bin).
- **docs/**
Przechowywanie dokumentacji.
- **obj/**
Przechowywanie plików wynikowych kompilacji typu object (.o). Folder ten jest tworzony i kasowany przez Makefile.
- **Makefile**
Organizacja kompilacji programu.
- **.gitignore**
Przechowywanie listy wyjątków - plików/folderów, które git powinien zignorować.

4.2 Moduły

- **main.c - plik główny**
Wywołanie modułów w odpowiedniej kolejności.
Parsowanie argumentów, wczytanie grafu z pliku wejściowego, utworzenie macierzy sąsiedztwa i macierzy Laplace'a, obliczenie wektorów własnych, klasteryzacja i podział grafu, wypisanie wyniku do pliku wyjściowego. Parsowanie argumentów, Wczytanie grafu, Przeprowadzenie podziału, Zapis wyniku.
- **config.h / config.c**
Obsługa argumentów.
- **input.h / input.c**
Wczytywanie grafu z pliku, reprezentacja grafu - macierz sąsiedztwa.
- **mat_vec.h / mat_vec.c**
Odpowiedzialny za wektory, macierze oraz operacje na nich, w tym za utworzenie macierzy Laplace'a.
- **eigenvalues.h / eigenvalues.c**
Implementacja algorytmu Lanczosa i obliczanie wartości wektorów własnych macierzy Laplace'a.
- **clusterization.h / clusterization.c**
Implementacja algorytmu k-means do podziału grafu. Podział grafu.
- **output.h / output.c**
Zapisanie wyniku w pliku wyjściowym w odpowiednim formacie.

4.3 Struktury

- Config

Przechowywanie informacji podanych przez użytkownika wsadowo.

```
4 // Struktura do przechowywania ustawień
5 typedef struct
6 {
7     int parts;
8     int margin;
9     char *input_file;
10    char *output_file;
11    char *format;
12 } Config;
```

- Input

Przechowywanie informacji pobranych z pliku wejściowego. Informacje te są później wykorzystywane m.in. do utworzenia macierzy sąsiedztwa.

```
6 // Struktura do przechowywania danych z pliku wejściowego
7 typedef struct
8 {
9     FILE *in;
10    int max_vertices;
11    int *vertices_groups;
12    size_t g_count;
13    int *vertices_ptrs;
14    size_t p_count;
15    int v_count;
16    size_t len;
17 } Input;
```

- LanczosEigenV

Przechowywanie zmiennych do metody Lanczosa oraz do obliczania wartości własnych i przybliżonych wektorów własnych macierzy Laplace'a grafu.

```
4 // Struktura do przechowywania zmiennych potrzebnych do metody Lanczosa oraz do
  // obliczenia przybliżeń wektorów własnych macierzy Laplace'a grafu
5 typedef struct
6 {
7     // rozmiar macierzy Laplace'a
8     int n;
9     // liczba iteracji
10    int m;
11    // baza ortonormalna Q (przybliżenia wektorów własnych)
12    double* Q;
13    // macierz trójdzielna T
14    double* alpha;
15    double* beta;
16    // wartości własne macierzy T
17    double* theta;
18    // wektory własne macierzy T (macierz)
19    double* y;
20    // przybliżone wektory własne macierzy L (macierz)
21    double* x;
22 } LanczosEigenV;
23 // Nazwy takie same jak na Wikipedii - Lanczos algorithm
24 // https://en.wikipedia.org/wiki/Lanczos\_algorithm
```

4.4 Najważniejsze funkcje

- `parse_args`

Obsługa argumentów. Zapisanie konfiguracji do obiektu struktury `Config`.

```
8 // Parsowanie argumentów, zwrócenie skonfigurowanej struktury
9 Config parse_args(int argc, char **argv) {
10     Config c;
11
12     // Ustawienia domyślne
13     c.parts = 2;
14     c.margin = 10;
15     c.input_file = NULL;
16     c.output_file = "output.txt";
17     c.format = "txt";
18
19     // Parsowanie argumentów
20     int opt;
21     while ((opt = getopt(argc, argv, "p:m:o:f:")) != -1) {
22         switch (opt) {
23             // Parts
24             case 'p':
25                 c.parts = atoi(optarg);
26                 break;
27             // Margin
28             case 'm':
29                 c.margin = atoi(optarg);
30                 break;
31             // Output
32             case 'o':
33                 c.output_file = optarg;
34                 break;
35             // Format
36             case 'f':
37                 c.format = optarg;
38                 break;
39             // Nieznana flaga
40             case '?':
41                 fprintf(stderr, "Nieznana flaga: -%c\n", optopt);
42                 exit(EXIT_FAILURE);
43         }
44     }
45     return c;
46 }
```

- `read_input`

Wczytanie danych z pliku wejściowego do obiektu struktury `Input`.

```

24 // Czytanie pliku linia po linii
25 void read_input(Input *i)
26 {
27     ssize_t read;
28     char *line = NULL;
29     int line_number = 0;
30
31     while((read = getline(&line, &i->len, i->in)) != -1)
32     {
33         line_number++;
34
35         // Odczytanie maksymalnej liczby wierzchołków w wierszu
36         if(line_number == 1)
37         {
38             i->max_vertices = atoi(line);
39         }
40         // Pominięcie linii dla interfejsu graficznego
41         else if(line_number == 2 || line_number == 3)
42         {
43             continue;
44         }
45         // Odczytanie grafu (krawędzi pomiędzy wierzchołkami i ich numerów)
46 > else if(line_number == 4 || line_number == 5)...
102     }
103
104     // Zwolnienie pamięci dla odczytu linii
105     free(line);
106
107     // Zamknięcie pliku wejściowego
108     fclose(i->in);
109 }

```

- `get_adjacency_matrix`
Zapisanie grafu w formie macierzy sąsiedzywa.


```

36 // Wczytanie grafu do macierzy sąsiedztwa A
37 int* get_adjacency_matrix(Input *i)
38 {
39     printf("\n\tPołączenia dodane do macierzy sąsiedztwa:");
40     int *A = calloc(i->v_count * i->v_count, sizeof(int));
41     int p = 0;
42     int v = 0;
43     for(int it = 0; it < (int)i->g_count; it++)
44     {
45         // Zaktualizuj wierzchołek
46         if(p < (int)i->p_count && it == i->vertices_ptrs[p])
47         {
48             v = i->vertices_groups[it];
49             printf("\n\t%d - %d: ", it, v);
50             p++;
51         }
52         else
53         {
54             printf("\t%d", i->vertices_groups[it]);
55             add_edge(A, v, i->vertices_groups[it], i->v_count);
56         }
57     }
58     printf("\n");
59
60     // Test
61     //printf("\n\t%d\n", getv(A, 93, 90, v_count));
62
63     // Wyświetlenie macierzy sąsiedztwa
64     if(i->v_count < 30)
65     {
66         printf("\n");
67         printv(A, i->v_count * i->v_count, i->v_count);
68     }
69
70     return A;
71 }

```

- calc_laplacian
Obliczenie macierzy Laplace'a grafu.

```

111 // Obliczenie macierzy Laplace'a grafu L
112 int* calc_laplacian(int* A, int* D, int n)
113 {
114     // = 0
115     int* L = calloc(n * n, sizeof(int));
116     if(L == NULL)
117     {
118         fprintf(stderr, "Błąd alokacji pamięci!\n");
119         free(A);
120         free(D);
121     }
122
123     // + D
124     for(int i = 0; i < n; i++)
125     {
126         L[i * n + i] = D[i];
127     }
128
129     // - A
130     for(int i = 0; i < n; i++)
131     {
132         for(int j = 0; j < n; j++)
133         {
134             L[i * n + j] -= A[i * n + j];
135         }
136     }
137
138     // Wyświetlenie macierzy Laplace'a
139     if(n < 30)
140     {
141         printf("\n");
142         printv(L, n * n, n);
143     }
144
145     return L;
146 }

```

- lanczos

Algorytm Lanczosa - obliczenie k wektorów własnych i zapisanie ich do macierzy.

```

48 // Metoda Lanczosa
49 void lanczos(LanczosEigenV *l);
50

```

- clusterization

Algorytm klasteryzacji k-means (k-średnich) - podział wierzchołków na grupy.

```
53 // Metoda klasteryzacji k-means
54 void clusterization(double* V);
```

- **write_output**
Wypisanie pliku wyjściowego z wynikiem i grafem.

```
123 // Zapisanie danych do pliku wyjściowego
124 void write_output(Input i, int* A);
```

4.5 Pliki wejściowe i wyjściowe

- **Format plików wejściowych**

Program akceptuje pliki wejściowe z rozszerzeniem .csrrg. Format danych wejściowych jest zdefiniowany za pomocą 5 linijek, z których każda odpowiada za zapis informacji.

1. Maksymalna możliwa liczba węzłów w wierszu (w grafie mogą być wiersze o ich mniejszej liczbie, ale nie o większej).
2. Indeksy węzłów w poszczególnych wierszach – liczba wszystkich indeksów odpowiada liczbie węzłów grafu.
3. Wskaźniki na pierwsze indeksy węzłów w liście wierszy z poprzedniego punktu.
4. Grupy węzłów połączone przy pomocy krawędzi.
5. Wskaźniki na pierwsze węzły w poszczególnych grupach z poprzedniego punktu.
Ta sekcja może występować w pliku wielokrotnie, co oznacza, że plik zawiera więcej niż jeden graf.

Przykład:

```
input > graf.csrrg
1 18
2 3;5;6;9;10;13;14;15;10;12;15;4;5;7;8;9;12;14;4;8;9;10;11;12;13;14;15;1;2;3;7;9;13;15;16;1;3;
3 0;0;8;11;18;27;35;41;47;57;67;67;76;82;90;90;94;98;105
4 0;72;39;91;4;54;1;47;4;79;101;71;2;76;79;5;3;63;71;80;42;4;75;5;49;6;93;16;98;7;75;82;50;8;2
5 0;6;12;16;21;23;25;29;33;38;42;45;49;53;55;62;65;68;70;73;76;82;87;89;97;100;103;107;112;116
```

- **Format plików wyjściowych**

Pierwsza linijka pliku wyjściowego zawiera wynik działania programu w postaci:

<wynik (S - pomyślnie, F - nieudanie)> <liczba_części> <zachowany_margines>

Przykład: S 2 0.045

Kolejne linijki są w takim samym formacie jak plik wejściowy i zawierają podział grafu.

Wersja binarna zawiera te same informacje co tekstowa, ale w formie binarnej (struktury danych zapisane za pomocą `fwrite`).

Literatura

- [1] Leonid Zhukov, *Lecture 7. Graph partitioning algorithms.*, YouTube, 24 luty 2021, Dostępny na 1 kwietnia 2025 w: https://youtu.be/zZae_C2BU_4
- [2] *Laplacian matrix*, Wikipedia, Dostępne na 1 kwietnia 2025 w: https://en.wikipedia.org/wiki/Laplacian_matrix
- [3] *Lanczos algorithm*, Wikipedia, Dostępne na 1 kwietnia 2025 w: https://en.wikipedia.org/wiki/Lanczos_algorithm
- [4] *Algorytm centroidów*, Wikipedia, Dostępne na 1 kwietnia 2025 w: https://pl.wikipedia.org/wiki/Algorytm_centroid%C3%B3w