

# GridGraph - Specyfikacja

Skoczek Mateusz, Jędrzejewski Sebastian

2 kwietnia 2022

## Streszczenie

Dokument zawiera specyfikację funkcjonalną oraz implementacyjną dotyczącą projektu *Grid-Graph*

# Spis treści

<b>1</b>	<b>Specyfikacja funkcjonalna</b>	<b>3</b>
1.1	Cel projektu . . . . .	4
1.2	Opis funkcji . . . . .	5
1.3	Opis wywołania . . . . .	6
1.3.1	Tryb zapisu . . . . .	6
1.3.2	Tryb czytania . . . . .	7
1.4	Format danych wejściowych i wyjściowych . . . . .	9
1.5	Opis błędów . . . . .	10
<b>2</b>	<b>Specyfikacja implementacyjna</b>	<b>12</b>
2.1	Diagram dołączeń plików źródłowych . . . . .	13
2.2	Plik main.c . . . . .	14
2.2.1	Stałe *_o . . . . .	14
2.2.2	Stała help . . . . .	14
2.2.3	Funkcja write_init . . . . .	14
2.2.4	Funkcja read_init . . . . .	14
2.2.5	Funkcja main . . . . .	14
2.3	Pliki write.c i .h . . . . .	15
2.3.1	Stała ec_drawing_weight . . . . .	15
2.3.2	Funkcja gen_graph . . . . .	15
2.3.3	Funkcja write . . . . .	15
2.4	Pliki read.c i .h . . . . .	16
2.4.1	Funkcja path_fill . . . . .	16
2.4.2	Funkcja path_display . . . . .	16
2.4.3	Funkcja bfs_init . . . . .	16
2.4.4	Funkcja dijkstra_init . . . . .	16
2.4.5	Funkcja read_graph . . . . .	17
2.4.6	Funkcja read . . . . .	17
2.5	Pliki dijkstra.c i .h . . . . .	18
2.5.1	Struktura d_result . . . . .	18
2.5.2	Funkcja result_init . . . . .	18
2.5.3	Funkcja result_free . . . . .	18
2.5.4	Funkcja dijkstra . . . . .	18
2.6	Pliki bfs.c i .h . . . . .	19
2.6.1	Funkcja bfs . . . . .	19
2.7	Pliki graph.c i .h . . . . .	20
2.7.1	Struktura graph . . . . .	20
2.7.2	Struktura edge_list . . . . .	20
2.7.3	Funkcja graph_init . . . . .	20
2.7.4	Funkcja graph_free . . . . .	21
2.7.5	Funkcja edge_list_add . . . . .	21

2.7.6	Funkcja <code>edge_list_contains_vertex</code> . . . . .	21
2.7.7	Funkcja <code>edge_list_contains_vertex</code> . . . . .	21
2.8	Pliki <code>queue.c</code> i <code>.h</code> . . . . .	22
2.8.1	Struktura <code>queue</code> . . . . .	22
2.8.2	Funkcja <code>queue_enqueue</code> . . . . .	22
2.8.3	Funkcja <code>queue_enqueue</code> . . . . .	22
2.9	Pliki <code>helpers.c</code> i <code>.h</code> . . . . .	23
2.9.1	Funkcja <code>str_arr_get_index</code> . . . . .	23
2.9.2	Funkcja <code>str_is_int</code> . . . . .	23
2.9.3	Funkcja <code>str_is_double</code> . . . . .	23
2.10	Plik <code>makefile</code> . . . . .	24
2.10.1	Kompilacja ( <code>make</code> ) . . . . .	24
2.10.2	Czyszczenie ( <code>make clean</code> ) . . . . .	24
2.10.3	Test ( <code>make test</code> ) . . . . .	24

## Rozdział 1

# Specyfikacja funkcjonalna

## 1.1 Cel projektu

Program `GridGraph` ma na celu wygenerowanie oraz zapis do pliku (lub na standardowe wyjście) grafu siatkowego o podanych paramentrach lub wczytanie grafu z pliku (lub ze standardowego wejścia) i sprawdzenie wybranych jego parametrów. Program działa w trybie wsadowym. Grafy są przedstawiane w plikach w postaci listy sąsiedztwa.

## 1.2 Opis funkcji

Program może działać w dwóch trybach: zapisu (`write`) i czytania (`read`).

W trybie zapisu program generuje graf o określonej przez użytkownika szerokości (ilości kolumn) (`width`), wysokości (ilości wierszy) (`height`), minimalnej (`edge_weight_min`) i maksymalnej (`edge_weight_max`) wagi krawędzi oraz minimalnej (`edge_count_min`) i maksymalnej (`edge_count_max`) ilości krawędzi wychodzących z jednego wierzchołka, a następnie zapisuje go w formie listy sąsiedztwa do pliku określonego przez użytkownika (lub wypisuje na standardowe wyjście).

Jeżeli graf zostanie pomyślnie zapisany do pliku (lub wypisany na standardowe wyjście), program zwróci 0. W przeciwnym wypadku zostanie wyświetlony komunikat błędu, a program zwróci 1.

W trybie czytania program wczytuje graf zapisany (w formie listy sąsiedztwa) w określonym przez użytkownika pliku (lub czyta ze standardowego wejścia), a następnie sprawdza określone przez użytkownika właściwości grafu:

- Spójność grafu (`connectivity`)
- Najkrótsza ścieżka z węzła A do innych węzłów (`shortest_path_a`) lub do określonego węzła B (`shortest_path_a` oraz `shortest_path_b`)

Jeżeli graf został wczytany oraz sprawdzony pomyślnie, zostanie wyświetlony wynik sprawdzania...

Przykład (graf spójny, ścieżka istnieje):

`Connectivity: connected`

`Shortest path from 0 to 10 (weight): 0-3-4-6-9-10 (0.778)`

Przykład (graf niespójny, ścieżka nie istnieje):

`Connectivity: disconnected`

`Shortest path from 0 to 10 (weight): path does not exist`

...a następnie program zwróci 0. W przeciwnym wypadku zostanie wyświetlony komunikat błędu, a program zwróci 1

## 1.3 Opis wywołania

Jeżeli nie zostanie wybrany tryb (tzn. nie zostanie przekazany argument `write` lub `read`) zostanie wyświetlona pomoc.

### 1.3.1 Tryb zapisu

Wywołanie:

```
./gridgraph --write/-w [argumenty]
```

Argumenty:

- `--width/-xw` (Szerokość grafu - liczba kolumn)  
**Typ:** Liczba naturalna  
**Zakres:**  $> 0$   
**Wymagany:** TAK
- `--height/-xh` (Wysokość grafu - liczba wierszy)  
**Typ:** Liczba naturalna  
**Zakres:**  $> 0$   
**Wymagany:** TAK
- `--edge_weight_min/-Wmin` (Minimalna waga pojedynczej krawędzi)  
**Typ:** Liczba rzeczywista  
**Zakres:**  $<0, \text{edge\_weight\_max}>$   
**Wymagany:** NIE (domyślnie: 0)
- `--edge_weight_max/-Wmax` (Maksymalna waga pojedynczej krawędzi)  
**Typ:** Liczba rzeczywista  
**Zakres:**  $<\text{edge\_weight\_min}, 1>$   
**Wymagany:** NIE (domyślnie: 1)
- `--edge_count_min/-Cmin` (Minimalna liczba krawędzi wychodzących z jednego wierzchołka)<sup>1</sup>  
**Typ:** Liczba naturalna  
**Zakres:**  $<0, \text{edge\_count\_max}>$   
**Wymagany:** NIE (domyślnie: 0)
- `--edge_count_max/-Cmax` (Maksymalna liczba krawędzi wychodzących z jednego wierzchołka)  
**Typ:** Liczba naturalna

---

<sup>1</sup>Program będzie dążył do utworzenia co najmniej `edge_count_min` krawędzi, ale nie może tego zagwarantować. Nie jest możliwe wygenerowanie więcej niż 2 krawędzi dla wierzchołków w narożnikach oraz więcej niż 3 dla wierzchołków bocznych. Nie jest możliwe także utworzenie krawędzi, jeżeli wszystkie wierzchołki wokół osiągnęły już swoją nominalną (wylosowaną z podanego przedziału) liczbę krawędzi.



**Zakres:** <edge\_count\_min, 4>

**Wymagany:** NIE (domyślnie: 4)

- **--file/-f** (Plik w którym ma zostać zapisany graf)

**Typ:** Ścieżka do pliku

**Zakres:** -

**Wymagany:** NIE (domyślnie: standardowe wyjście)

Przykład:

```
./gridgraph -w -xw 6 -xh 6 --Wmin 0.65 --Wmax 0.2 -Cmax 3 -f "/home/user/graph"
```

Powyższy przykład ilustruje wywołanie programu, który generuje graf o 6 kolumnach i 6 wierszach, z wagami krawędzi mieszczącymi się w przedziale od 0.2 do 0.65, gdzie minimalna ilość krawędzi wychodzących z wierzchołka to 0, a maksymalna ilość krawędzi to 3. Program zapisuje graf w odpowiednim formacie do pliku o nazwie `graph` znajdującego się w `/home/user`.

### 1.3.2 Tryb czytania

Wywołanie:

```
./gridgraph --read/-r [argumenty]
```

Argumenty:

- **--connectivity/-c** (Sprawdza czy graf jest spójny, używając algorytmu BFS)

**Typ:** -

**Zakres:** -

**Wymagany:** NIE<sup>2</sup>

- **--shortest\_path\_a/-Sa** (Znajduje najkrótszą ścieżkę od wierzchołka A do pozostałych wierzchołków, używając algorytmu Dijkstry)

**Typ:** Liczba naturalna

**Zakres:** <0, ilość wierzchołków grafu>

**Wymagany:** NIE<sup>2</sup>

- **--shortest\_path\_b/-Sb** (Znajduje najkrótszą ścieżkę od wierzchołka A do wierzchołka B, używając algorytmu Dijkstry)

**UWAGA:** `shortest_path_a` wymagane jeżeli `shortest_path_b` zostało podane

**Typ:** Liczba naturalna

**Zakres:** <0, ilość wierzchołków grafu> (nie licząc `shortest_path_a`)

**Wymagany:** NIE<sup>2</sup>

- **--file/-f** (Plik z którego ma zostać wczytany graf)

**Typ:** Ścieżka do pliku

**Zakres:** -

**Wymagany:** NIE (domyślnie: standardowe wejście)

Przykład:

```
./gridgraph -r -c -Sa 0 -Sb 10 -f \home/user/graph"
```

Powyższy przykład ilustruje wywołanie programu, który czyta plik ze strukturą grafu o nazwie `graph` znajdujący się w `/home/user`, a następnie sprawdza czy ten graf jest spójny oraz wyznacza najkrótszą ścieżkę pomiędzy węzłami numer 0 i 10.

---

<sup>2</sup>Wymagany przynajmniej jeden

## 1.4 Format danych wejściowych i wyjściowych

Dane wejściowe i wyjściowe przechowują graf w postaci listy sąsiedztwa. W pierwszej linii znajdują się dwie liczby, które oznaczają odpowiednio liczbę kolumn i wierszy danego grafu. Każda następna linijka reprezentuje jeden wierzchołek, przy czym wierzchołki numerujemy od 0 od lewej do prawej. Zatem druga linijka w pliku zawiera numery wierzchołków, z którymi połączony jest wierzchołek numer 0, kolejna dotyczy wierzchołka numer 1 itd. Przy każdym numerze wierzchołka po dwukropku podana jest waga krawędzi pomiędzy tymi dwoma wierzchołkami.

Przykład:

```
2 2
1 :0.54  2 :0.78
0 :0.54  3 :0.12
0 :0.78  3 :0.89
1 :0.12  2 :0.89
```

Powyżej przedstawiona jest przykładowa zawartość pliku przechowującego graf. W pierwszej linii można odczytać, że jest to graf o dwóch kolumnach i dwóch wierszach. W drugiej linii przedstawiona jest informacja o tym, że wierzchołek numer 0 połączony jest z wierzchołkiem numer 1, a krawędź ta ma wagę 0.54. Istnieje również krawędź pomiędzy wierzchołkiem 0 a 2 o wadze 0.78. W trzeciej linii znajdują się numery wierzchołków połączonych z wierzchołkiem numer 1 wraz z wagami itd.

## 1.5 Opis błędów

W przypadku błędu program wypisuje błąd na standardowy strumień błędów i zwraca 1. Komunikat błędów jest poprzedzony słowem **ERROR** oraz nazwą trybu w nawiasie (np. (**Write mode**)), jeżeli błąd dotyczy konkretnego trybu.

Poniżej przedstawione są komunikaty generowane przez program, gdy ten wykryje błąd, wraz z ich wyjaśnieniem:

**WIDTH\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **width**, ale nie została podana wartość lub wartość nie jest liczbą.

**HEIGHT\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **height**, ale nie została podana wartość lub wartość nie jest liczbą.

**EDGE\_WEIGHT\_MIN\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **edge\_weight\_min**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**EDGE\_WEIGHT\_MAX\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **edge\_weight\_max**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**EDGE\_COUNT\_MIN\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **edge\_count\_min**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**EDGE\_COUNT\_MAX\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **edge\_count\_max**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**WIDTH\_LOWER\_OR\_EQUAL\_TO\_ZERO** Wartość argumentu **width** jest mniejsza lub równa 0 (musi być większa od 0).

**HEIGHT\_LOWER\_OR\_EQUAL\_TO\_ZERO** Wartość argumentu **height** jest mniejsza lub równa 0 (musi być większa od 0).

**EDGE\_WEIGHT\_MIN\_LOWER\_THAN\_ZERO** Wartość argumentu **edge\_weight\_min** jest mniejsza od 0 (musi być większa lub równa 0 i mniejsza lub równa **edge\_weight\_max**).

**EDGE\_WEIGHT\_MAX\_GREATER\_THAN\_ONE** Wartość argumentu **edge\_weight\_max** jest większa od 1 (musi być mniejsza lub równa 1 i większa lub równa **edge\_weight\_min**).

**EDGE\_WEIGHT\_MIN\_GREATER\_THAN\_EDGE\_WEIGHT\_MAX** Wartość argumentu **edge\_weight\_min** jest większa od **edge\_weight\_max** (musi być większa lub równa 0 i mniejsza lub równa **edge\_weight\_max**).

**EDGE\_COUNT\_MIN\_LOWER\_THAN\_ZERO** Wartość argumentu **edge\_count\_min** jest mniejsza od 0 (musi być większa lub równa 0 i mniejsza lub równa **edge\_count\_max**).

**EDGE\_COUNT\_MAX\_GREATER\_THAN\_FOUR** Wartość argumentu **edge\_count\_max** jest większa od 4 (musi być mniejsza lub równa 4 i większa lub równa **edge\_count\_min**).

**EDGE\_COUNT\_MIN\_GREATER\_THAN\_EDGE\_COUNT\_MAX** Wartość argumentu **edge\_count\_min** jest większa od **edge\_count\_max** (musi być większa lub równa 0 i mniejsza lub równa **edge\_count\_max**).

**SHORTEST\_PATH\_A\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **shortest\_path\_a**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**SHORTEST\_PATH\_B\_NOT\_POSITIVE\_NUMBER** Został wybrany argument **shortest\_path\_b**, ale nie została podana wartość lub wartość nie jest liczbą (nieujemną).

**SHORTEST\_PATH\_B\_WITHOUT\_SHORTEST\_PATH\_A\_SPECIFIED** Został wybrany argument `shortest_path_a`, ale nie został wybrany argument `shortest_path_b`.

**SHORTEST\_PATH\_B\_EQUAL\_TO\_SHORTEST\_PATH\_A** Argument `shortest_path_b` jest równy `shortest_path_a` (wartości argumentów muszą być różne od siebie).

**CHECKING\_OPTIONS\_NOT\_SPECIFIED** Nie została wybrana przynajmniej jedna opcja sprawdzająca (przynajmniej jedna wymagana).

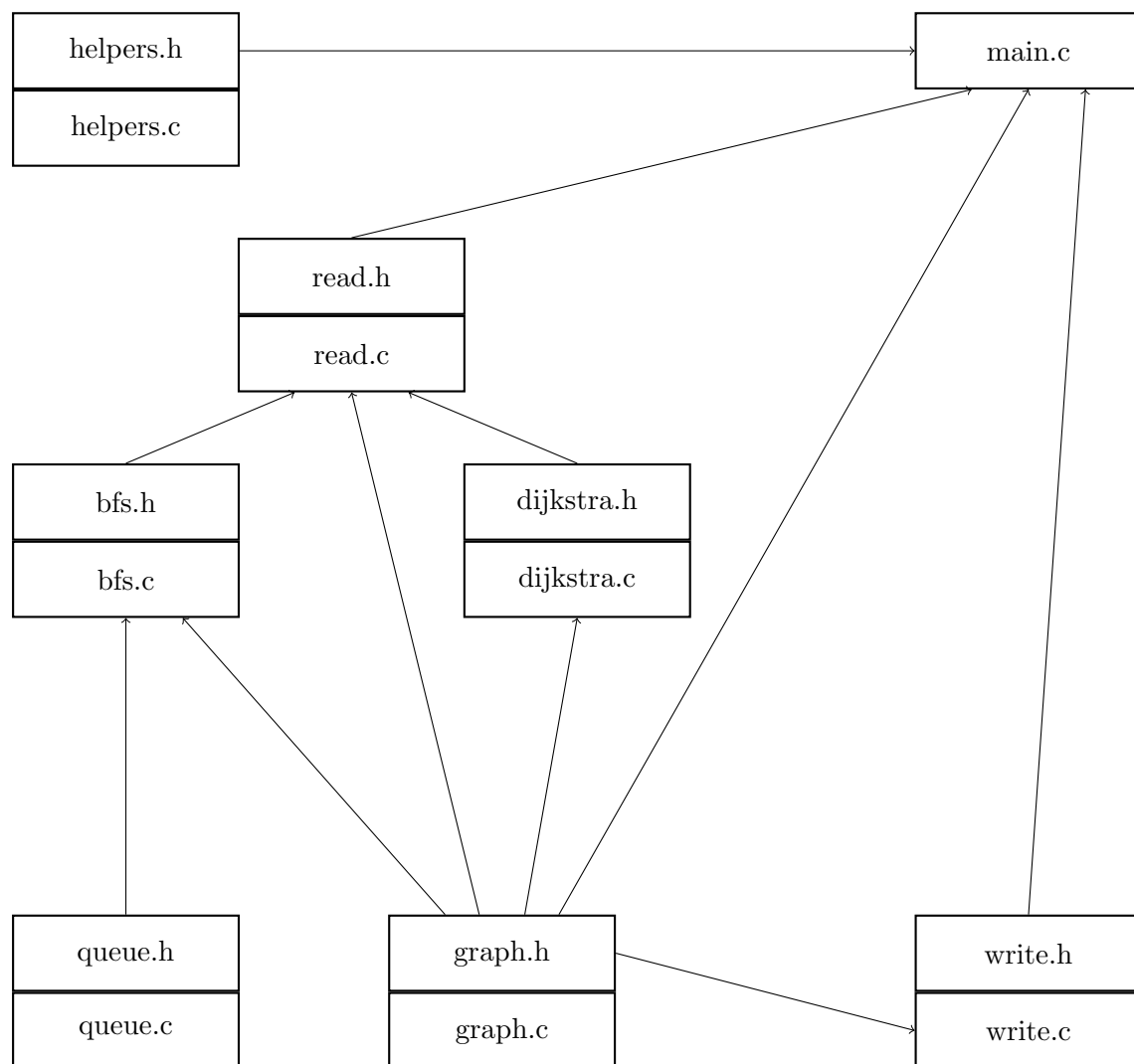
**SHORTEST\_PATH\_A\_GREATER\_THAN\_TOTAL\_NUMBER\_OF\_VERTICES** Wartość argumentu `shortest_path_a` jest większa niż całkowita liczba wierzchołków grafu.

**SHORTEST\_PATH\_B\_GREATER\_THAN\_TOTAL\_NUMBER\_OF\_VERTICES** Wartość argumentu `shortest_path_b` jest większa niż całkowita liczba wierzchołków grafu.

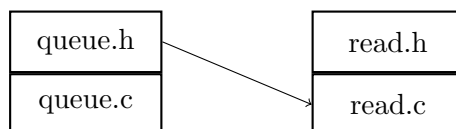
## Rozdział 2

# Specyfikacja implementacyjna

## 2.1 Diagram zależności plików źródłowych



Objaśnienie:



Plik **queue.h** jest dołączany do pliku **read.c** (oraz ewentualnie do **read.h**). To znaczy że w pliku **read.c** (oraz ewentualnie w **read.h**) znajduje się dyrektywa `#include "queue.h"`.

## 2.2 Plik main.c

Plik źródłowy (main.c) zawiera stałe \*\_o i help oraz definicje funkcji:

- write\_init
- read\_init
- main

### 2.2.1 Stałe \*\_o

```
const char* *_o[2] (np. const char* file_o[2])
```

Stale te przechowują nazwy argumentów wywołania, gdzie pierwszym elementem jest długa (normalna) nazwa argumentu, a drugim nazwa skrócona.

### 2.2.2 Stała help

```
const char* help
```

Stała ta przechowuje pomoc, zawierającą instrukcję wywołania i opisy wszystkich argumentów.

### 2.2.3 Funkcja write\_init

```
int write_init(int argc, char* argv[])
```

Funkcja ta sprawdza argumenty wywołania dla trybu zapisu i ich wartości, a następnie wywołuje funkcję odpowiedzialną za tryb zapisu. Funkcja zwraca wartości EXIT\_SUCCESS lub EXIT\_FAILURE.

### 2.2.4 Funkcja read\_init

```
int read_init(int argc, char* argv[])
```

Funkcja ta sprawdza argumenty wywołania dla trybu czytania i ich wartości, a następnie wywołuje funkcję odpowiedzialną za tryb czytania. Funkcja zwraca wartości EXIT\_SUCCESS lub EXIT\_FAILURE.

### 2.2.5 Funkcja main

```
int main(int argc, char* argv[])
```

Funkcja ta sprawdza, który tryb został wybrany, a następnie wywołuje funkcje write\_init lub read\_init (lub wyświetla pomoc jeżeli tryb nie został wybrany). Funkcja zwraca wartości EXIT\_SUCCESS lub EXIT\_FAILURE.



## 2.3 Pliki `write.c` i `.h`

Plik źródłowy (`write.c`) zawiera stałą `ec_drawing_weight` oraz definicje funkcji:

- `gen_graph`
- `write*`

Plik nagłówkowy (`write.h`) zawiera deklaracje zaznaczonych funkcji z pliku źródłowego (\*).

### 2.3.1 Stała `ec_drawing_weight`

```
const int ec_drawing_weight[5]
```

Stała ta przechowuje informację o wagach losowania dla poszczególnych liczb krawędzi. Na przykład wartości `{1,2,3,4,5}` oznaczają że wraz z liczbą krawędzi rośnie szansa na wylosowanie danej liczby krawędzi, gdyż tablica z której losowana będzie liczba krawędzi będzie wyglądać tak: `{0,1,1,1,2,2,2,3,3,3,3,3,4,4,4,4,4}`.

### 2.3.2 Funkcja `gen_graph`

```
graph gen_graph(int width, int height, double edge_weight_min, double edge_weight_max,  
                int edge_count_min, int edge_count_max)
```

Funkcja ta odpowiada za wygenerowanie grafu o szerokości (ilości kolumn) `width`, wysokości (ilości wierszy) `height`, gdzie każdy wierzchołek będzie miał maksymalnie `edge_count_max` i (w miarę możliwości) minimalnie `edge_count_min` krawędzi<sup>1</sup>, z których każda będzie miała wagę mieszczącą się w przedziale od `edge_weight_min` do `edge_weight_max` włącznie. Funkcja zwraca graf.

### 2.3.3 Funkcja `write`

```
int write(FILE* file, int width, int height, double edge_weight_min,  
          double edge_weight_max, int edge_count_min, int edge_count_max)
```

Funkcja ta wywołuje funkcję `gen_graph` (przekazując jej jednocześnie wszystkie swoje parametry poza `file`), a następnie zapisuje uzyskany graf do pliku `file`. Funkcja zwraca wartości `EXIT_SUCCESS` lub `EXIT_FAILURE` (funkcja nie przewiduje wystąpienia błędu, więc w praktyce jest to tylko `EXIT_SUCCESS`).

---

<sup>1</sup>liczba krawędzi jest losowana; patrz "Stała `ec.drawing.weight`"

## 2.4 Pliki `read.c` i `.h`

Plik źródłowy (`read.c`) zawiera definicje funkcji:

- `path_fill`
- `path_display`
- `bfs_init`
- `dijkstra_init`
- `read_graph`
- `read*`

Plik nagłówkowy (`read.h`) zawiera deklaracje zaznaczonych funkcji z pliku źródłowego (\*).

### 2.4.1 Funkcja `path_fill`

```
int path_fill(d_result result, int *nv, int from, int to)
```

Funkcja ta odpowiada za wypełnienie tablicy `nv` numerami wierzchołków, które tworzą ścieżkę od `from` do `to`, na podstawie tablicy poprzedników wygenerowanej przez algorytm Dijkstry i przechowywanej w strukturze `result`. Funkcja zwraca ilość dodanych wierzchołków do tablicy.

### 2.4.2 Funkcja `path_display`

```
void path_display(int *nv, int l)
```

Funkcja ta odpowiada za wyświetlenie ścieżki pomiędzy dwoma wierzchołkami przechowywanej w tablicy `nv` znając jej długość `l`.

### 2.4.3 Funkcja `bfs_init`

```
void bfs_init(graph g)
```

Funkcja ta odpowiada za wywołanie algorytmu BFS i wyświetlenie komunikatu o spójności grafu `g`.

### 2.4.4 Funkcja `dijkstra_init`

```
void dijkstra_init(graph g, int vertex_a, int vertex_b)
```

Funkcja ta odpowiada za wywołanie algorytmu Dijkstry i wyświetlenie najkrótszej ścieżki pomiędzy wierzchołkami `vertex_a` i `vertex_b` lub najkrótszych ścieżek pomiędzy wierzchołkiem `vertex_a` i pozostałymi wierzchołkami grafu `g`.

### 2.4.5 Funkcja `read_graph`

```
graph read_graph(FILE *f)
```

Funkcja ta odpowiada za odczytanie grafu z pliku `f`, tworzenie struktury grafu i uzupełnienie jej przeczytanymi wierzchołkami i wagami z pliku. Funkcja zwraca stworzoną strukturę grafu.

### 2.4.6 Funkcja `read`

```
int read(FILE *file, int connectivity, int vertex_a, int vertex_b)
```

Funkcja odpowiada za zarządzanie trybem `read`. Przyjmuje wskaźnik na plik (bądź `stdin`), z którego ma być czytany graf, a także czy ma być sprawdzona jego spójność (`connectivity` przyjmuje wartość 1/0) oraz numery wierzchołków, między którymi ma być wyznaczona ścieżka (gdy `vertex_b` jest równy -1, wyznaczana jest ścieżka pomiędzy `vertex_a` i wszystkimi wierzchołkami).

## 2.5 Pliki `dijkstra.c` i `.h`

Plik źródłowy (`dijkstra.c`) zawiera definicje funkcji:

- `result_init*`
- `result_free*`
- `dijkstra*`

Plik nagłówkowy (`dijkstra.h`) poza deklaracjami zaznaczonych funkcji z pliku źródłowego (\*) zawiera także deklarację struktury wyniku generowanego przez algorytm Dijkstry.

### 2.5.1 Struktura `d_result`

```
typedef struct r
{
    double *d;
    int *p;
} *d_result;
```

Struktura przechowuje dwie tablice: `d` - zawierającą całkowitą wagę dojścia do danego wierzchołka z wierzchołka początkowego oraz `p` - zawierającą numery poprzedników.

### 2.5.2 Funkcja `result_init`

```
d_result result_init(int n, int vertex_a, int *visited)
```

Funkcja ta odpowiada za zainicjalizowanie struktury typu `d_result` i wypełnienie jej tablic w taki sposób jaki wymaga tego algorytm Dijkstry (tablica `d` wypełniona nieskończonościami oprócz elementu o indeksie wierzchołka początkowego; tablica `p` wypełniona wartościami -1). Oprócz tego uzupełnia tablicę `visited` zerami. Funkcja zwraca utworzoną strukturę.

### 2.5.3 Funkcja `result_free`

```
void result_free(d_result result)
```

Funkcja ta odpowiada za zwolnienie pamięci zarezerwowanej dla struktury `result`.

### 2.5.4 Funkcja `dijkstra`

```
d_result dijkstra(graph graph, int vertex_a)
```

Funkcja ta odpowiada za wykorzystanie algorytmu Dijkstry do uzupełnienia tablic struktury typu `d_result`. Funkcja zwraca utworzoną strukturę.

## 2.6 Pliki `bfs.c` i `.h`

Plik źródłowy (`bfs.c`) zawiera definicje funkcji:

- `bfs*`

Plik nagłówkowy (`bfs.h`) zawiera deklaracje zaznaczonych funkcji z pliku źródłowego (\*).

### 2.6.1 Funkcja `bfs`

```
int* bfs(graph graph, int vertex)
```

Funkcja ta jest implementacją algorytmu BFS (Breadth-first search) w uproszczonej wersji. Sprawdza czy w grafie `graph` od wierzchołka `vertex` istnieją ścieżki do wszystkich pozostałych wierzchołków. Funkcja zwraca tablicę liczb 0/1 (prawda/fałsz) o długości [ilość wierzchołków grafu], w której `index` oznacza numer wierzchołka a wartość wynik sprawdzenia dla wierzchołka o danym indeksie.

## 2.7 Pliki graph.c i .h

Plik źródłowy (`graph.c`) zawiera definicje funkcji:

- `graph_init*`
- `graph_free*`
- `edge_list_add*`
- `edge_list_contains_vertex*`
- `edge_list_length*`

Plik nagłówkowy (`graph.h`) poza deklaracjami zaznaczonych funkcji z pliku źródłowego (\*) zawiera także deklarację struktury grafu w formie listy sąsiedztwa (`graph`) oraz deklarację struktury listy wierzchołków (`edge_list`).

### 2.7.1 Struktura graph

```
typedef struct g
{
    int width;
    int height;
    edge_list **list;
} *graph;
```

Struktura przechowuje szerokość grafu (liczba kolumn) (`width`), wysokość grafu (liczba wierszy) (`height`) oraz tablicę list połączonych wierzchołków dla każdego wierzchołka (lista sąsiedztwa) (`list`).

### 2.7.2 Struktura edge\_list

```
typedef struct e
{
    int vertex;
    double weight;
    struct e *next;
} edge_list;
```

Struktura przechowuje numer połączanego wierzchołka (`vertex`), wagę krawędzi (`weight`) oraz wskaźnik na następny element (`next`) (lista jednokierunkowa).

### 2.7.3 Funkcja graph\_init

```
graph graph_init(int w, int h)
```

Funkcja ta odpowiada za zainicjalizowanie grafu o szerokości (ilości kolumn) `w` i wysokości (ilości wierszy) `h`. Funkcja zwraca graf

#### 2.7.4 Funkcja `graph_free`

```
void graph_free(graph g)
```

Funkcja ta odpowiada za zwolnienie pamięci zarezerwowanej dla grafu `g`.

#### 2.7.5 Funkcja `edge_list_add`

```
edge_list* edge_list_add(edge_list *l, int v, double wt)
```

Funkcja ta odpowiada za dodanie na koniec listy połączonych wierzchołów `l` nowego wierzchołka o numerze `v` i wadze krawędzi `wt`. Funkcja zwraca wskaźnik na listę połączonych wierzchołów.

#### 2.7.6 Funkcja `edge_list_contains_vertex`

```
int edge_list_contains_vertex(edge_list* l, int v)
```

Funkcja ta odpowiada za sprawdzenie czy na liście połączonych wierzchołów `l` znajduje się wierzchołek o numerze `v`. Funkcja zwraca wartości 1/0 (prawda/fałsz).

#### 2.7.7 Funkcja `edge_list_length`

```
int edge_list_length(edge_list* l)
```

Funkcja ta odpowiada za sprawdzenie długości listy połączonych wierzchołów `l`. Funkcja zwraca długość listy.

## 2.8 Pliki queue.c i .h

Plik źródłowy (queue.c) zawiera definicje funkcji:

- queue\_enqueue\*
- queue\_dequeue\*

Plik nagłówkowy (queue.h) poza deklaracjami zaznaczonych funkcji z pliku źródłowego (\*) zawiera także deklarację struktury kolejki FIFO (First in, first out) (queue).

### 2.8.1 Struktura queue

```
typedef struct q
{
    int value;
    struct q* next;
} queue;
```

Struktura przechowuje wartość pojedynczego elementu (value) oraz wskaźnik na następny element (next) (lista jednokierunkowa).

### 2.8.2 Funkcja queue\_enqueue

```
void queue_enqueue(queue** q, int value)
```

Funkcja ta odpowiada za dodanie na początku kolejki q (listy) nowego elementu o wartości value.

### 2.8.3 Funkcja queue\_dequeue

```
int queue_dequeue(queue** q)
```

Funkcja ta odpowiada za pobranie wartości elementu z końca kolejki q (listy), usunięcie ostatniego elementu z kolejki, a następnie zwrócenie pobranej wartości.



## 2.9 Pliki `helpers.c` i `.h`

Plik źródłowy (`helpers.c`) zawiera definicje funkcji:

- `str_arr_get_index*`
- `str_is_int*`
- `str_is_float*`

Plik nagłówkowy (`helpers.h`) zawiera deklaracje zaznaczonych funkcji z pliku źródłowego (\*).

### 2.9.1 Funkcja `str_arr_get_index`

```
int str_arr_get_index(char* element, const char* array[], int length)
```

Funkcja ta odpowiada za znalezienie w tablicy napisów `array` o długości `length` konkretnego napisu `element`. Funkcja zwraca indeks tego napisu w tablicy.

### 2.9.2 Funkcja `str_is_int`

```
int str_is_int(char* string)
```

Funkcja ta odpowiada za sprawdzenie czy napis `string` jest liczbą całkowitą nieujemną (tzn. napis zawiera tylko cyfry). Funkcja zwraca wartości 1/0 (prawda/fałsz).

### 2.9.3 Funkcja `str_is_double`

```
int str_is_double(char* string)
```

Funkcja ta odpowiada za sprawdzenie czy napis `string` jest liczbą rzeczywistą nieujemną (tzn. napis zawiera tylko cyfry i jedną kropkę). Funkcja zwraca wartości 1/0 (prawda/fałsz).

## 2.10 Plik makefile

Plik `makefile` przechowuje instrukcje kompilacji programu oraz instrukcję czyszczenia folderu (`clean`) i instrukcję testu (`test`).

### 2.10.1 Kompilacja (`make`)

Instrukcja kompilacji składa się z czterech etapów:

1. wygenerowania pliku zawierającego zależności  
(`cc -MM [pliki źródłowe] > [plik zaw. zależności]`)
2. wczytania go (`-include [plik zaw. zależności]`)
3. skompilowania programu (`cc -o gridgraph [pliki .o]`)
4. czyszczenia pokompilacyjnego (`rm [pliki .o] [plik zaw. zależności]`)

### 2.10.2 Czyszczenie (`make clean`)

Instrukcja czyszczenia usuwa pliki (wywoływana jest komenda `rm [pliki]`):

- plik programu
- pliki `.o`
- plik zawierający zależności

### 2.10.3 Test (`make test`)

Instrukcja testu w pierwszej kolejności wywołuje instrukcję kompilacji, a następnie