Specyfikacja implementacyjna programu tworzącego i badającego grafy

Wojciech Majchrzak, Dawid Stereńczak

March 2022

1 Wstęp

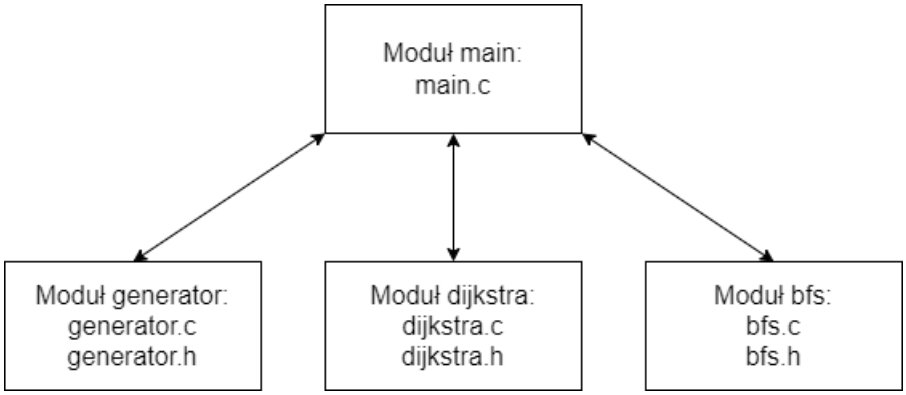
Dokument ten opisuje strukturę programu, podział na pliki nagłówkowe i źródłowe, oraz funkcje zawarte w tych plikach. Dodatkowo specyfikuje struktury danych którymi będzie operował program, algorytmy użyte w programie, oraz dodatkowe pliki i testowanie programu.
Do napisania programu zostaną użyte następujące technologie:

- Język programowania - `C`
- System kontroli wersji - `Git`

Program używany będzie w trybie wsadowym poprzez terminal `bash` w systemach z rodziny `Linux`.

2 Moduły programu

Program podzielony jest na moduły: `main, generator, dijkstra, bfs`.



Poglądowy schemat modułów, i komunikacji między nimi.

2.1 Modul main

Moduł main zawiera plik `main.c` i jest głównym modułem programu. Służy do obsługi argumentów wejściowych programu, oraz sterowania programem i wywoływania poszczególnych funkcji. W pierwszej kolejności moduł powinien za pomocą biblioteki `getopt` rozpoznawać argumenty wywołania programu, a następnie przetwarzać je do odpowiedniej formy i sprawdzać ich poprawność. Po wczytaniu argumentów, moduł na ich podstawie wywołuje odpowiednie funkcje i steruje działaniem programu.

2.2 Moduł generator

Moduł ten składa się z dwóch plików: `generator.c` oraz `generator.h`.
Plik `generator.h` zawiera definicje funkcji:

- `create_graph( int x, int y, char *nazwa_pliku, double początek_zakresu, double koniec_zakresu)` - która odpowiada za wygenerowanie grafu o podanych parametrach, i zapisanie go do podanego pliku. Jako argumenty funkcja przyjmuje wymiary grafu, następnie nazwę pliku do którego wygeneruje graf, oraz zakres wag krawędzi generowanego grafu.

W pliku `generator.c` znajduje się kod wyżej wymienionych funkcji.

2.3 Moduł dijkstra

Składa się z dwóch plików: `dijkstra.c` oraz `dijkstra.h`.
Moduł ten będzie odpowiadał za znalezienie najkrótszej drogi między węzłami grafu podanymi przez użytkownika. W tym celu będzie wykorzystany

algorytm Dijkstry, którego szczegółowy opis znajduje się w sekcji 3.3 tego dokumentu.
Plik `dijkstra.h` zawiera definicje funkcji:

- `dijkstra( char *nazwa_pliku, int x1, int y1, int x2, int y2)` - która odpowiada za znalezienie i obliczenie długości najkrótszej drogi pomiędzy wskazanymi wierzchołkami grafu. Za argumenty wywołania przyjmuje nazwę pliku w którym znajduje się graf, oraz współrzędne punktów między którymi liczymy drogę.

W pliku `dijkstra.c` znajdować się będzie implementacja funkcji wymienionych w pliku `dijkstra.h`.

2.4 Moduł bfs

Moduł ten składa się z dwóch plików: `bfs.c` oraz `bfs.h`.
Będzie w nim zaimplementowany algorytm BFS, którego szczegółowy opis znajduje się w sekcji 3.3 tego dokumentu.
Plik `bfs.h` zawiera definicje funkcji:

- `bfs( char *nazwa_pliku)` - która odpowiada za sprawdzenie spójności grafu za pomocą algorytmu BFS. Jako argument przyjmuje jedynie nazwę pliku w którym znajduje się badany graf.

Plik `bfs.c` zawierał będzie implementacje funkcji wymienionych w pliku nagłówkowym `bfs.h`.

3 Dodatkowe elementy programu

3.1 Makefile

Program będzie dodatkowo zawierał plik `Makefile`, w którym zawarte będą trzy reguły:

- make - standardowa reguła, służąca do kompilowania gotowego programu
- debug - służy do kompilowania z dodatkowymi opcjami do debugownaia programu
- clear - służy do czyszczenia folderów projektu, w tym usuwania plików tymczasowych

3.2 Struktury danych

Graf będzie w programie przechowywany przy pomocy tablicy wskaźników na listy liniowe. Każdy element listy będzie posiadał dwie wartości - numer wierzchołka do którego można z niego przejść oraz wagę tego przejścia. Lista będzie wyglądała w następujący sposób:

```
struct node {
     int dest;
     int weight;
     struct node *next;
};
```

gdzie:
`dest` - numer wierzchołka sąsiadującego;
`weight` - waga przejścia między node a dest;
`struct node *next` - wskaźnik na następny element listy;

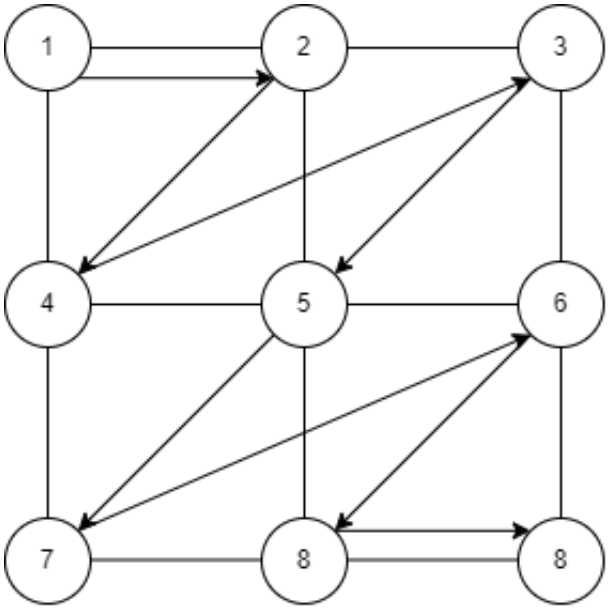
Tablica zaś dla ułatwienia również zostanie zapisana jako struktura w postaci:

```
struct graph {
     struct node *head[];
};
```

3.3 Algorytmy w programie

W tym akapicie zostaną opisane algorytmy które są używane w programie:

- Algorytm BFS - służy do sprawdzania spójności grafu. Do działania potrzebuje dwóch kolejek FIFO(kolejka First In, First Out). Algorytm zaczyna od wpisania pierwszego wierzchołka grafu do pierwszej kolejki. Następnie wpisuje do pierwszej kolejki wszystkie wierzchołki sąsiadujące z pierwszym wierzchołkiem. Po tej operacji pierwszy wierzchołek trafia do drugiej kolejki, a do pierwszej kolejki trafiają kolejne wierzchołki sąsiadujące z tymi znajdującymi się w pierwszej kolejce. Gdy do pierwszej kolejki zostaną dodane wszystkie wierzchołki sąsiadujące z danym wierzchołkiem jest on przenoszony do drugiej kolejki. W ten sposób należy postąpić ze wszystkimi węzłami. Jeżeli w drugiej kolejce znajdują się wszystkie wierzchołki grafu to znaczy że jest on spójny. W innym przypadku mamy doczynienia z grafem niespójnym.



Poglądowy schemat kolejności dodawania wierzchołków do kolejek

- Algorytm Dijkstry - służy do wyznaczania najkrótszych ścieżek w grafie między wybranym wierzchołkiem początkowym a pozostałymi wierzchołkami grafu. Algorytm wymaga, aby wagi krawędzi grafu nie były ujemne. Do każdego wierzchołka zostają przypisane 2 wartości - koszt dotarcia do niego oraz poprzedni wierzchołek na ścieżce. Początkowo każdy wierzchołek ma dystans nieskończony a poprzednik jest nieokreślony. Jedynie dla wierzchołka początkowego dystans wynosi 0.

  Do implementacji algorytmu wykorzystana jest kolejka priorytetowa. Początkowo w kolejce znajdują się wszystkie wierzchołki. Następnie, dopóki kolejka nie jest pusta, w każdej iteracji algorytm usuwa z kolejki wierzchołek o najmniejszym koszcie dotarcia i bada dla niego wszystkie połączone wierzchołki. Jeśli droga do sąsiadującego wierzchołka jest krótsza niż dotychczasowa, to należy zaktualizować dystans oraz poprzednika.

4 Przeprowadzanie testów

Program obsługuje wiele rodzajów błędów - wszystkie zostały opisane w specyfikacji funkcjonalnej. Testy zostaną przeprowadzone poprzez uruchomienie programu z odpowiednio przygotowanymi niepoprawnymi argumentami wywołania oraz błędnie sformatowanymi plikami wejściowymi tak, aby uzyskać

wszystkie możliwe komunikaty błędów z osobna. Z wyników testów zostaną utworzone raporty w postaci plików .txt, które zostaną udostępnione zespołowi za pośrednictwem gita.