Specyfikacja Funkcjonalna programu GraphSolver generującego oraz operującego na grafach

Autorzy: Wojciech Wasilewski i Wiktor Zaremba

I. Cel Projektu

Specyfikacja ta opisuje program do tworzenia grafów, zapisywania ich, czytania oraz wykonywania na nich funkcji przy użyciu algorytmów operujących nimi. Będzie on służył do sprawdzania spójności danego grafu i wyszukiwania najkrótszych ścieżek między danymi węzłami.

Głównymi zadaniami programu jest operowania na grafach za pomocą algorytmów BFS i Dijkstry. BFS, czyli algorytm służący do przeszukiwania grafu wszerz polega na odnalezieniu wszystkich możliwych dróg z podanego wierzchołka, umożliwia on sprawdzenie, czy graf jest spójny. Natomiast algorytm Dijkstry pozwala nam znaleźć najkrótszą ścieżkę między dwoma podanymi wierzchołkami w grafie (pod względem sumy wartości wag znajdujących się na krawędziach w grafie). Ścieżką nazywamy krawędzie, przez które trzeba „przejść" by z jednego wierzchołka przemieścić się na inny (przy okazji możliwie ze przechodząc przez inne wierzchołki).

W dodatku, program oferuje takie funkcjonalności jak generowanie grafu o podanym rozmiarze, jego odczyt oraz generowanie losowych wag na krawędziach danego grafu.

II. Opis Funkcji

Program potrafi wygenerować graf o podanych parametrach, ilości kolumn i wierszy węzłów, a także losowo dobierać wagi z podanego przez użytkownika zakresu. Tak wygenerowany graf, program może zapisać do pliku z rozszerzeniem .txt, jak również go odczytać.

Program akceptuje argumenty po skompilowaniu poleceniem *make* i wywołaniu *./a.out*:

- ***-generate x y a-z***, generuje graf o podanej w kolejności liczbie kolumn, wierszy i zakresie wag, dolny zakres wag nie może być mniejszy niż 0.
- ***-save filename.txt***, zapisuje graf o podanej nazwie do pliku o formacie .txt.
- ***-bfs s***, program odczytuje graf z pliku o formacie .txt, a następnie przeszukuje go z wierzchołka "s" algorytmem BFS. Uwaga! W przypadku braku argumentu "s" program poinformuje o błędzie.
- ***-dijkstra a b***, algorytm przeszukuje w grafie najkrótszą ścieżkę od zadanego wierzchołka. Uwaga! W przypadku braku jednego z argumentów program poinformuje o błędzie.
- ***-read filename.txt***, odczytuje graf o podanej nazwie, flaga może być zastosowana wraz z algorytmami BFS lub Dijkstra.

III. Format danych wejściowych i struktura plików

Program zostanie podzielony na pliki - moduły, z czego każdy z nich będzie zawierał inną funkcjonalność.

Głównym modułem będzie plik main.c, odpowiedzialny za sterowanie programem oraz poprawny odczyt argumentów podanych przez użytkownika w wierszu poleceń. Zostanie także zaimplementowana struktura wspomagająca odczytywanie flag. Moduł main.c będzie się odwoływał do wszystkich pozostałych plików zaimplementowanych w programie.

W pliku generator.c będzie znajdowała się funkcja, która pozwoli na generowanie oraz zapis grafu do pliku o podanej nazwie, wybranej ilości wierszy i kolumn oraz zakresie wag na krawędziach.

W module bfs.c będzie znajdować się funkcja, która będzie zwracała wartości typu int (1 lub 0) w przypadku gdy graf jest spójny bądź też nie.

W pliku dijkstra.c będzie znajdować się algorytm szukający najkrótszej ścieżki. Algorytm ten, docelowo będzie zwracał tablicę struktur z informacjami o pokonanych wierzchołkach, przebytej drodze.

Następnym modułem będzie reader.c, plik w którym program będzie odczytywał graf ze wskazanego przez użytkownika pliku, zaimplementowane zostanie także informowanie użytkownika o błędzie w odczytywaniu grafu z pliku.

Będą to:

- ***main.c*** – moduł sterujący całym programem i wywołujący odpowiednie funkcje na podstawie argumentów wywołania
- ***generator.c*** – moduł generujący oraz zapisujący graf o zadanej liczbie kolumn i wierszy
- ***bfs.c*** – moduł zawierający implementację algorytmu BFS do przeszukiwania grafu w celu ustalenia jego spójności
- ***dijkstra.c*** – moduł zawierający implementację algorytmu Dijkstry do szukania najkrótszej ścieżki między dwoma węzłami
- ***reader.c*** - moduł pozwalający na odczytanie grafu

Wszystkie potrzebne informacje o danym wierzchołku będą przechowywane w strukturze pokazanej niżej:

```
typedef struct {
        int x;          /* numer wierzcholka */
        int n;          /* ilosc polaczen z tego wierzcholka */
        int *w;         /* numery wierzcholkow polaczonych */
        double *drogi;  /* drogi do wierzcholkow polaczonych zgodnie z kolejnoscia wyzej */
        int p;          /* numer poprzednika w najkrotszej drodze (Dijkstra) */
        int stan;       /* stan potrzebny do obu algorytmow */
} wierzcholek;
```

Natomiast wszystkie wierzchołki będą przechowywane w tablicy wskaźników na takie struktury (wierzchołek nr. 0 na miejscu 0., wierzchołek nr. 1 na miejscu 1. Itd.)

W ramach danych wejściowych będzie można (przy użyciu odpowiedniego argumentu wywołania) podać plik z zapisanym grafem w odpowiednim formacie. Program będzie mógł również generować taki plik, informację o spójności danego grafu lub najkrótszą ścieżkę między danymi węzłami.

Program będzie generował plik o formacie .txt z zapisanym w nim grafem lub wypisze na standardowe wyjście efekt działania algorytmu BFS (informację, czy zadany graf jest spójny, czy nie), lub najkrótszą ścieżkę między dwoma podanymi wierzchołkami oraz jej długość.

IV. Scenariusz działania programu

W celu uruchomienia programu należy przejść do katalogu, w którym znajdują się pliki programu, a następnie skompilować program poleceniem “make” i go wywołać komendą “./a.out”. Po tych czynnościach nasz program jest gotowy do działania i interakcji z użytkownikiem

Program poinformuje o następujących rodzajach błędów:

- Informacja o niespójności grafu. Kod błędu: 1
- Wywołanie programu bez wymaganych flag, ***-save*** lub ***-read.*** Kod błędu: 2
- Zły format pliku wejściowego. Kod błędu: 3
- Program nie może otworzyć/zlokalizować pliku wejściowego. Kod błędu: 4
- Podanie przez użytkownika sprzecznych flag. Kod błędu: 5
- Wybranie przez użytkownika wierzchołka początkowego lub końcowego, który nie należy do wygenerowanego grafu. Kod błędu: 6

W wymienionych przypadkach program będzie informował o kodzie błędu oraz o tym, co spowodowało wystąpienie tego błędu. Będą także wyświetlane zalecane działania, w celu pozbycia się błędu.

V. Testowanie

Każdy z modułów wyszczególnionych w specyfikacji zostanie poddany osobnym, niezależnym od reszty modułów testom, w celu wykrycia występowania ewentualnych błędów. W szczególności testowi zostanie poddany:

- Moduł generatora
- Moduł odpowiedzialny za algorytmy BFS i Dijkstra
- Moduł zapisu i odczytu grafu

W celu sprawdzenia programu pod kątem wycieków pamięci oraz debugowania aplikacji, zostanie użyte narzędzie Valgrind. Jeżeli każdy z modułów pomyślnie przejdzie testy, oznaczać to będzie, że program działa poprawnie. W przypadku błędnego działania którejś z funkcjonalności programu, testy pojedynczych modułów pozwolą na szybkie zlokalizowanie błędu.