Specyfikacja funkcjonalna programu rysującego grafy i ścieżki: *grafC*.

Kacper Karaś, Jakub Niewadzi

10.04.2022

1. Cel projektu

Program ma na celu generowanie, zapisywanie i czytanie grafów oraz powinien posiadać opcje sprawdzenia czy graf jest spójny za pomocą algorytmu BFS i wyznaczenia najkrótszej możliwej ścieżki pomiędzy wybraną parą węzłów, przy pomocy algorytmu Dijkstry.

2. Scenariusz działania programu

Program służy do badań grafu pod względem spójności wykorzystując algorytm BFS oraz znajduje w nim najkrótsze ścieżki za pomocą algorytmu Dijkstry. Program generuje samodzielnie graf lub pobiera dane z umieszczonego przez użytkownika pliku. Program wykonawczy uruchamiamy z odpowiednimi argumentami umożliwiającymi nam wykorzystanie programu w przewidzianym przez nas celu (więcej o formacie danych jak i rodzajach argumentów będzie mowa w dalszej części specyfikacji). Po wczytaniu argumentów program zaprojektuje badany graf w tablicy dwuwymiarowej i rozpocznie badanie go odpowiednimi algorytmami. Po zakończeniu działania algorytmów program wypisze wyniki swojego działania na wyjście główne lub do pliku, jeżeli wyrazimy taką chęć podczas uruchamiania programu.

3. Kompilacja programu

Wpisanie “make”, bądź “make grafC” w konsoli będzie skutkowało skompilowaniem programu. Kompilacja poszczególnych testów jest wykonywana za pomocą komend “make testBFS”, “make testCzytania”, “ make testDijkstra”, “make testGeneracji”.

4. Argumenty wywoływania programu

Program grafC akceptuje następujące wejścia:

- -f <plik> -> podajemy plik, w którym zapisane są wierzchołki naszego grafu i wagi połączeń między nimi. Jeżeli nie zostanie podany ten argument, program samodzielnie wygeneruje nam graf
- -b -> program sprawdza, czy graf jest spójny za pomocą algorytmu BFS i wypisuje rodzica każdego z wierzchołków
- -d <a b> -> program za pomocą algorytmu Dijkstry wyznacza możliwe ścieżki pomiędzy wybranym wierzchołkiem, a resztą wierzchołków w grafie. Przy ewentualnym podaniu 2

wierzchołka znajduję on najkrótszą trasę między oboma punktami. Litera “a” to wierzchołek, z którego będziemy wyznaczać trasy jest on konieczna dana do wywołania tej opcji, litera “b” to opcjonalny wierzchołek, między którym będziemy wyznaczać najkrótszą trasę

- -g <k w> -> Generacja grafu o podanej ilości kolumn i wierszy, gdzie “k“ to liczba kolumn, a “w” liczba wierszy. Liczba “k” i “w” są konieczne do uruchomienia tej opcji.
- -w <min max> -> Ustalenie przedziału wag, gdzie “min” to najniższa waga, a “max” maksymalna. Wagi są liczbami rzeczywistymi nieujemnymi
- -z <plik> -> Zapisuje wygenerowany graf do pliku o podanej nazwie.
- -h -> wyświetla instrukcję korzystania z programu.

Jeżeli program zostanie uruchomiony bez argumentów, to automatycznie wyświetli się instrukcja korzystania z programu.

Jeżeli jakikolwiek argument zostanie wypisany z nieprawidłową liczbą danych program zaznaczy ten błąd i zakończy działanie.

Przykładowe wywołanie programu:

```
./grafC -g 7 4 -b -w 0 20 –z punkty
```

Efektem takiego polecenia będzie uruchomienie programu grafC, który wygeneruje graf o 7 kolumnach i 4 wierszach z wagami w przedziale <0, 20> i zapisze go w odpowiedniej formie w pliku punkty. Następnie program sprawdzi, czy graf jest spójny metodą BFS.

```
./grafC -f test1 –b –d 0 7
```

Takie polecenie przeczyta graf z pliku test1 (zakładając, że jest poprawnie podany), sprawdzi jego spójność oraz obliczy najkrótszą ścieżkę z wierzchołka 0 do wierzchołka 7.

5. Dane wejściowe

Program zakłada możliwość przekazania własnego pliku przechowującego informacje o ilości kolumn i wierszy w grafie oraz opisuje ilość krawędzi w każdym punkcie oraz ich wagi. Wagi krawędzi, gdy nie zostanie podany argument wejściowy mówiący inaczej, to liczby rzeczywiste z przedziału 0 – 10.

Plik ten powinien być sformatowany następująco:

- Dwoma pierwszymi liczbami powinno być liczba kolumn i wierszy grafu
- Następnie powinny pojawić się opisy wierzchołków, które występują w grafie (tu należy zaznaczyć, że numeracje zaczynamy od 0 i numerujemy w dół poszczególnych kolumn). Opisy wierzchołków powinny rozpoczynać się od pierwszego i kończyć na ostatnim, dlatego nie ma potrzeby zaznaczać jaki wierzchołek aktualnie opisujemy.
- Opis wierzchołka powinien składać się z co najwyżej 4 bloków zawierających numer sąsiadującego wierzchołka oraz wagę krawędzi między aktualnie wymienionym wierzchołkiem, a aktualnie opisywanym. Numer wierzchołka i waga powinny być od siebie oddzielone wyłącznie znakami spacji.

W przypadku wywołania programu bez żadnych argumentów program, poinformuje o tym użytkownika i samodzielnie wygeneruje plik z danymi. Będzie to graf 10 x 10.

Przykładowy prawidłowo sformatowany plik:

```
3 3
      1 :1.232          3 :2.233
      0 :1.232          2 :0.485          4 :7.583
      1 :0.485          5 :6.593
      0 :2.233          4 :2.690          6 :9.515
      1 :7.583          3 :2.690          5 :10.000   7 :3.467
      2 :6.593          4 :10.000   8 :3.234
      3 :9.515          7 :7.383
      4 :3.467          6 :7.383          8 :8.487
      5 :3.234          7 :8.487
```

6. Teoria

Graf jest to uporządkowana para {**W, K**}, gdzie W to zbiór n węzłów odpowiednio ponumerowanych np. 0, 1, 2, …, n-1. K jest to zbiór krawędzi, czyli relacji między dwoma węzłami, które w naszym przypadku posiadają wagę (liczbę rzeczywistą).

Graf spójny - graf, dla którego spełniony jest warunek, że dla każdej pary węzłów istnieje ścieżka, która je łączy.

7. Komunikaty błędów

Program *grafC* będzie kontynuował pracę po wpisaniu błędnych argumentów wtedy, kiedy jest to możliwe.

Oto lista komunikatów o błędach jakie mogą wystąpić podczas próby uruchomienia programu:

1) Program nie ma dostępu do pliku, z którego chcesz pobrać dane na temat opisu grafu, spróbuj podać inny plik.
2) Nieodpowiedni format lub ilość liczb w pierwszym wierszu pliku! Powinny być to 2 liczby naturalne określające liczbę wierszy i kolumn grafu.
3) Podczas wywoływania argumentu -f należy podać maksymalnie jeden parametr. Program kończy działanie.
4) Podano argument dla algorytmu BFS, który nie przyjmuje argumentów.
5) Podany argument dla -d nie jest liczbą całkowitą. Program kończy działanie.
6) Podczas wywoływania argumentu -d wymagane jest podanie początkowego wierzchołka wymaganego do analizy grafu za pomocą algorytmu Dijkstry. Nie można wywołać argumentu bez tego parametru.
7) Podczas wywoływania argumentu -d należy podać maksymalnie dwa parametry. Program kończy działanie.
8) Podany argument dla -g nie jest liczbą całkowitą. Program kończy działanie.

9) Podczas wywoływania argumentu -g należy podać maksymalnie dwa parametry. Program kończy działanie.
10) Podany argument dla -w nie jest liczbą rzeczywistą. Program kończy działanie.
11) Podczas wywoływania argumentu -w należy podać dwa parametry: wagę minimalną i wagę maksymalną. Nie można wywołać argumentu bez tych parametrów
12) Podczas wywoływania argumentu -z należy podać maksymalnie jeden parametr. Program kończy działanie
13) Podczas wywoływania programu nie podano żadnej metody analizy grafu! Jednym z argumentów wywołania powinien być -b lub -d lub oba.
14) Podano zły argument wywołania! Aby dowiedzieć się jakie argumenty przyjmuje program proszę wywołać go z flagą -h.
15) Podana przez użytkownika waga minimalna jest większa od wagi maksymalnej! Podczas wywoływania argumentu -w pierwszą zmienną powinna być waga minimalna, a potem maksymalna. Nieodwrotnie! Proszę poprawić kolejność wag lub podać inne ich wartości.
16) Początkowy węzeł podany przez użytkownika do analizy grafu algorytmem Dijkstry nie zawiera się w grafie! Proszę podać wierzchołek znajdujący się w przedziale od 0 do [ilość węzłów].
17) Końcowy węzeł podany przez użytkownika do analizy grafu algorytmem Dijkstry nie zawiera się w grafie! Proszę podać wierzchołek znajdujący się w przedziale od 0 do [ilość węzłów].
18) W pliku znajduje się wierzchołek, który wychodzi poza podany zakres. Program kończy działanie.

Specyfikacja implementacyjna programu *grafC.*

Pliki źródłowe programu:

Poniżej znajduje się opis programów jakie będą używane w projekcie wraz z krótkim jakie funkcje w sobie zawierają orz jakie argumenty przyjmują (jeżeli jakieś przyjmują):

- **Main.c** - zawiera wykonanie wszystkich innych plików źródłowych oraz funkcje wykonujące flagi, jako argumenty wejścia.

- **BFS.c, BFS.h** - zawiera algorytm sprawdzający, czy graf jest spójny, posiada funkcję BFS, strukturę typu bfs_t, która zawiera w sobie 3 tablice: kolor wierzchołka, odległość od wierzchołka początkowego i rodzica wierzchołka. Funkcja przyjmuje argumenty: tablica z sąsiednimi wierzchołkami (int **), numer wierzchołka początkowego (int), ilość wierzchołków (int), tablicę z ilością sąsiadów dla poszczególnych wierzchołków (int *) oraz strukturę bfs_t, której dane zostaną ustawione podczas wykonywania kodu. Plik zawiera również funkcję czyszczenieBFS, której zadaniem jest zwolnienie pamięci po alokacji pamięci na strukturę BFS. Jako argument przyjmuje wskaźnik na strukturę (BFS_t *). Dodatkowo w pliku BFS.h następuje deklaracja struktury BFS.

- **czytanie.c, czytanie.h** - zawiera funkcję grafWTablice, której zadaniem jest odczytanie i interpretacja pliku zawierającego opis grafu. Funkcja ta zwracającą liczbę całkowitą (0 w przypadku prawidłowego odczytania prawidłowo sformatowanego pliku, oraz 1 w przypadku wszelkich problemów). Dodatkowo generuje ona 2 tablice: sasiedzi i macierzSasiedztwa. Tablica sasiedzi jest częścią struktury tablice, a macierzSasiedztwa znajduje się w strukturze graf. Obie struktury zostaną bardziej szczegółowo omówione w dalszej części dokumentu. Funkcja zamieszczona w pliku przyjmuje za argumenty: wskaźnik do struktury graf (graf_t *), wskaźnik na strukturę tablice (tablice_t*) oraz nazwę pliku zawierającego opis grafu (char *). Plik zawiera funkcję czyszczenieTablic, której zadaniem jest zwolnienie pamięci po alokacji pamięci na strukturę tablice. Jako argument przyjmuje wskaźnik na strukturę (tablice_t *). Ostatnią funkcją jest czyszczenieGrafu, której zadaniem jest zwolnienie pamięci po alokacji pamięci na strukturę graf. Jako argument przyjmuje wskaźnik na strukturę (graf_t *). Dodatkowo w pliku nagłówkowym czytanie.h są zapisane deklaracje 2 struktur wykorzystywanych w programie: graf oraz tablice.

- **Dijkstra.c, Dijkstra.h** - zawiera algorytm szukający najkrótszej możliwej drogi z jednego wierzchołka do drugiego. Posiada funkcję Dijkstra typu void. Funkcja przyjmuje następujące argumenty: wskaźnik na strukturę graf (graf_t *), numer wierzchołka wedle którego algorytm ma szukać ścieżek (int), ilość wierzchołków analizowanego grafu (int) oraz jednowymiarowa tablica, w której będą zapisywane drogi od poszczególnych wierzchołków (*double).

- **generator.c, generator.h** -zawiera funkcje wygenerujGraf, która ma za zadanie wygenerowanie i zapisanie w strukturze graf grafu o odpowiedniej ilości wierszy i kolumn z podanymi wagami połączeń między punktami. Funkcja jest typu void zatem nie zwraca żadnych wartości. Funkcja do prawidłowego funkcjonowania potrzebuje następujących argumentów: wskaźnika na strukturę graf (graf_t *), wskaźnik na strukturę tablice (tablice_t*) oraz nazwę pliku, w którym program ma zapisać wygenerowany graf (char *). Plik zawiera również funkcję losuj umożliwiającą generację liczb pseudolosowych z podanego zakresu. Funkcja przyjmuje argumenty: dolny zakres losowania (double) i górny zakres losowania (double).

- **Makefile –** jest to plik funkcyjny, który ma ułatwić kompilacje wszystkich innych plików koniecznych do uruchomienia programu. Umożliwia skompilowanie programu oraz poszczególnych testów.

- **pliki testujące**: testCzytanie.c, testGenerator.c, testBFS.c, testDijkstra.c. Są to pliki do testowania odpowiednich funkcjonalności głównego programu.

- **pliki z danymi do testów:** test1, test2, test3. Są to pliki zawierające przykładowe dane do wywołania programu. Mają one umożliwić możliwość testowanie programu na różnych zestawach danych. Znajdą się również pliki z błędnymi danymi, aby można było przetestować zachowanie programu w przypadku napotkania błędu.

W pisanym przez nas programie będziemy wykorzystywać strukturę o nazwie graf, która będzie przechowywać następujące wartości:

- **int kolumny** – liczba kolumn grafu
- **int wiersze** – liczba wierszy w grafie
- **int wagaMin –** minimalna wartości wagi jaką może przyjmować połączenie między dwoma punktami grafu
- **int wagaMax** – maksymalna wartości wagi jaką może przyjmować połączenie między dwoma punktami grafu
- **double ** macierzSasiedztwa –** tablica dwuwymiarowa przechowująca wagi sąsiadów poszczególnych wierzchołków, w przypadku braku wierzchołka przechowuje stała DBL_MIN

Implementacja w kodzie w pliku czytanie.h:

```
typedef struct graf {
    int kolumny;
    int wiersze;
    double wagaMin;
    double wagaMax;
    double **macierzSasiedztwa;
} graf_t;
```

Następną wykorzystywaną strukturą jest struktura tabele przechowująca następujące informacje:

- **double **grafBFS –** tablica dwuwymiarowa przechowująca sąsiadów poszczególnych wierzchołków
- **int *sasiedzi –** tablica jednowymiarowa która ma rozmiar równy ilości wszystkich wierzchołków i na pozycji odpowiadającej odpowiedniemu wierzchołkowi przypisuje
- **int iloscWezlow** – przechowuje informacje o ilości węzłów w badanym grafie

Implementacja w kodzie w pliku czytanie.h:

```
typedef struct tablice {
    int **grafBFS;
    int *sasiedzi;
    int iloscWezlow;
} tablice_t;
```

Ostatnia wykorzystywaną strukturą jest struktura BFS przechowująca informacje konieczne do wykonania algorytmu BFS:

- **int *odleglosc** –tablica jednowymiarowa przechowująca odległości od wierzchołka 0 do kolejnych wierzchołków grafu.
- **int* rodzic** – tablica jednowymiarowa przechowuje kolejnych poprzedników wierzchołków w grafie
- **int *kolor** – tablica jednowymiarowa zawierająca wartości od 0 do 2, symbolizujące kolory poszczególnych wierzchołków przypisywanie im podczas badania grafu algorytmem BFS. 0 symbolizuje kolor biały, 1 szary i 2 czarny.

Implementacja w kodzie w pliku BFS.h:

```
typedef struct bfs{
  int* odleglosc;
  int* rodzic;
  int* kolor;
}bfs_t;
```

Opis testów

1) testCzytania.c

Jest to program testujący funkcje odczytującą graf z pliku. Program przyjmuje 2 argumenty podczas wywoływania. Pierwszym argumentem powinna być nazwa pliku zawierającego zapis grafu. Na podstawie tego pliku będziemy testować funkcję czytającą. Kolejnymi 2 argumentami powinny być liczby kolumn i wierszy grafu zapisanego we wcześniej podanym pliku. Obie te liczby powinny być naturalne. Ostatnie 2 argumenty powinny być to wartości najmniejszej i największej wagi połączeń pomiędzy wierzchołkami grafu. Mogą być to liczby zmiennoprzecinkowe.

Test polega na uruchomieniu funkcji grafWTablice oraz sprawdzeniu czy zmienna ilośćWezlow jest równa liczbie podanych przez użytkownika kolumn pomnożona przez liczbę wierszy podanych przez użytkownika. Jeżeli liczba ta jest taka sama test kontynuuje działanie w przeciwnym wypadku kończy działanie. Następnym obiektem, który jest testowany jest macierzSasiedztwa. Sprawdzany jest każdy element tablicy pod względem wag, które są tam zapisane. Gdy, któraś z wag nie znajduje się w przedziale podanym przez użytkownika program komunikuje to i kończy działanie. W przeciwnym wypadku test komunikuje użytkownikowi, że funkcja czytająca graf działa prawidłowo.

2) testGeneracji.c

Jest to program testujący funkcje generujący graf. Program przyjmuje 5 opcjonalnych argumentów. Pierwszymi dwoma argumentami powinna być liczba kolumn i wierszy grafu, którego chcemy wygenerować. Obie liczby powinny być liczbami naturalnymi. W przypadku ich niepodania test wygeneruje graf 10 x 10. Kolejne 2 argumenty powinny być to wartości najmniejszej i największej wagi połączeń pomiędzy wierzchołkami grafu. Mogą być to liczby zmiennoprzecinkowe. W przypadku niepodania tych wartości test przyporządkuje wartości z zakresu 0-10. Ostatnim argumentem jest nazwa pliku do jakiego chcielibyśmy zapisać wygenerowany graf. W przypadku niepodania takiego argumentu test wypiszę graf na wejście główne.

Test polega na uruchomieniu funkcji wygenerujGraf oraz sprawdzeniu odpowiednich parametrów, aby zdeterminować, czy funkcja działa prawidłowo. Na początku sprawdzamy czy zmienna ilośćWezlow jest równa liczbie podanych przez użytkownika kolumn pomnożona przez liczbę wierszy podanych przez użytkownika. Jeżeli liczba ta jest taka sama test kontynuuje działanie w przeciwnym wypadku kończy działanie. Następnym obiektem, który jest testowany jest macierzSasiedztwa. Sprawdzany jest każdy element tablicy pod względem wag, które są tam zapisane. Gdy, któraś z wag nie znajduje się w przedziale podanym przez użytkownika program komunikuje to i kończy działanie. W przeciwnym wypadku test komunikuje użytkownikowi, że funkcja generująca graf działa prawidłowo.

3) testBFS.c

Jest to program testujący funkcje analizującą graf algorytmem BFS. Program testujący przyjmuje 2 argumenty. Pierwszym z nich jest nazwa pliku zawierającego zapis grafu. Drugim jest liczba 0 lub 1. 0 oznacza, że w pliku jest zapisany graf spójny, 1 zaś oznacza, że graf nie jest spójny.

Test polega na sprawdzeniu czy funkcja BFS również dowiedzie, że graf znajdujący się w pliku podanym przez użytkownika jest spójny/niespójny. Aby tego dowieść na początku graf jest odczytywany z pliku. W przypadku podania pliku, do którego program nie ma dostępu test wyświetli odpowiedni komunikat oraz awaryjnie zakończy działanie. Gdy uda nam się zapisać graf w tablicySasiedztwa program przystępuje do uruchomienia funkcji BFS, a następnie analizuje wyniki przez nią wygenerowane. Jeżeli się, że funkcja zwróciła ten sam poziom spójności co podany przez użytkownika test informuje o prawidłowym działaniu funkcji BFS, w przeciwnym przypadku program wyświetla komunikat o nieprawidłowym działaniu badanej funkcji.

4) testDijkstra.c

Jest to program testujący funkcje analizującą graf algorytmem Dijkstry. Program testujący przyjmuje 4 argumenty. Pierwszym z nich jest nazwa pliku zawierającego zapis grafu. Drugim jest początkowy wierzchołek, od którego funkcja ma rozpocząć analizować graf. Trzecim argumentem jest wierzchołek końcowy, do którego funkcja ma wyznaczyć trasę. Czwartym argumentem jest długość najkrótszej ścieżki z wierzchołka początkowego do końcowego.

Test polega na sprawdzeniu czy funkcja dijkstra zwróci taką samą wartość najkrótszej ścieżki między podanymi przez użytkownika punktami. Aby tego dowieść na początku graf jest odczytywany z pliku. W przypadku podania pliku, do którego program nie ma dostępu test wyświetli odpowiedni komunikat oraz awaryjnie zakończy działanie. Gdy uda nam się zapisać graf w tablicySasiedztwa program przystępuje do uruchomienia funkcji dijkstra. Po jej wywołaniu sprawdzamy czy wyznaczona przez program droga zgadza się z podaną przez użytkownika. Jeżeli obie wartości są sobie równe program informuje nas o prawidłowym działaniu funkcji dijkstra, w przeciwnym przypadku program wyświetla komunikat o nieprawidłowym działaniu badanej funkcji.