

Sprawozdanie końcowe projektu 1

Sebastian Pietrykowski, Paweł Borkowski
Grupa projektowa nr 3

12 kwietnia 2022

1 Osiągnięcie celu projektu

Cel projektu będący wytworzeniem oprogramowania operującego na grafach uznaje się za osiągnięty.

Program tworzony w ramach projektu 1. znajduje najkrótszą możliwą ścieżkę pomiędzy dwoma wybranymi wierzchołkami oraz sprawdza, czy graf jest spójny. Potrafi on generować grafy o zadanej liczbie kolumn, wierszy oraz zakresie wartości, z którego będzie losowana waga krawędzi, podawanych przez użytkownika. Program dodatkowo wyposażony jest w możliwość zapisu wygenerowanego grafu do pliku oraz odczytu grafu z takiego pliku. W programie wykorzystywane są dwa algorytmy:

- algorytm Dijkstry – algorytm dzięki, któremu wyszukiwana jest najkrótsza ścieżka,
- algorytm BFS – algorytm umożliwiający sprawdzenie czy graf jest spójny.

Program działa w trybie wsadowym.

Udało się osiągnąć pełną zgodność ze specyfikacją funkcjonalną, część ustaleń ze specyfikacji implementacyjnej musiała zostać zmodyfikowana.

2 Co zostało zaimplementowane?

1. Moduł Graph - w tym module znajduje się kod odpowiedzialny za obsługę grafu.

- Struktury :
 - `graph_t` – Struktura odpowiedzialna za przechowywanie informacji na temat grafu.
 - * `int columns` – liczbę kolumn
 - * `int rows` – liczba rzędów
 - * `int no_vertexes` – liczba wierzchołków w grafie
 - * `double ** adj_mat` – macierz sąsiedztwa (ang. adjacency matrix) przechowująca krawędzie grafu; pierwszy wskaźnik wskazuje na numer wierzchołka, z którego zaczyna się krawędź, drugi - na wierzchołek, do którego biegnie krawędź; wartość danego elementu jest równa wadze odpowiadającej tej krawędzi jeżeli wartość jest większa od 0, jeżeli krawędź nie istnieje wartość elementu jest równa -1.
- Funkcje:
 - `read_graph` – wczytuje graf znajdujący się w pliku in; zwraca wczytany graf w przypadku sukcesu, NULL jeżeli wystąpił błąd
 - `make_graph` – tworzy nowy graf, wypełnia w nim wartościami zmienne `columns`, `rows` i `no_vertexes`, alokuje pamięć na wektor `adj_mat` o rozmiarze na `no_vertexes*no_vertexes` elementów
 - `does_have_all_edges` – sprawdzenie czy graf ma wszystkie możliwe krawędzie
 - `write_graph` – zapisywanie grafu do pliku wyjściowego
 - `neighbors` – zwraca listę wierzchołków
 - `print_graph` – funkcja używana do testowania generatorów, wypisująca dany graf

- `potential_neighbors` – zwraca tablicę wierzchołków sąsiadujących z danym wierzchołkiem
 - `free_graph` – dealokacja struktury graf
2. Moduł BFS - w tym module znajduje się kod odpowiedzialny za sprawdzenie czy graf jest spójny czy nie

- Struktury :
 - `FIFO_t` – Struktura odpowiedzialna za kolejkę FIFO w algorytmie BFS
 - * `int * vertexes` – tablica wierzchołków w kolejce FIFO
 - * `int head` – zmienna head określa indeks wierzchołka w kolejce dla którego będą szukani sąsiedzi
 - * `int back` – zmienna back określa indeks wierzchołka który został dodany do kolejki jako ostatni
- Funkcje:
 - `make_fifo` – alokacja oraz inicjalizacja kolejki
 - `FIFO_get` – usuwanie wierzchołka z kolejki w celu sprawdzenia jego sąsiadów
 - `FIFO_put` – wkładanie wierzchołka do kolejki
 - `bfs` – sterowanie całym modulem BFS, zwraca informację czy graf jest spójny
 - `write_graph` – zapisywanie grafu do pliku wyjściowego
 - `is_FIFO_empty` – sprawdza czy w kolejce są oczekujące wierzchołki
 - `free_bfs` – dealokacja struktury fifo oraz odwiedzonych wierzchołków

3. Struktura danych Set w module Data Structures

- Set – Struktura przechowująca zbiór, zawierający unikalne liczby całkowite.
 - `int * vertexes` – tablica dodanych wierzchołków
 - `no_elements` – liczba wierzchołków w tablicy `vertexes`.

Do struktury Set zostały napisane następujące funkcje:

- `Set_is_element_in` – sprawdź czy dana liczba znajduje się w zbiorze,
- `Set_is_empty` – sprawdź czy zbiór jest pusty,
- `Set_add` – dodaj daną liczbę całkowitą do zbioru,
- `Set_remove` – usuń daną liczbę całkowitą ze zbioru,
- `Set_pop` – zwróć losową liczbę całkowitą ze zbioru i usuń ją z tego zbioru.

4. Struktura danych PriorityQueue w module Data Structures

- PriorityQueue – Struktura symulująca kolejkę priorytetową. Każdej dodanej liczbie całkowitej odpowiada odległość wyrażona liczbą zmiennoprzecinkową.
 - `int * vertexes` – tablica wierzchołków dodanych do kolejki
 - `double * distances` – każdy element odpowiada elementowi o takim samym indeksie w `vertexes`, zawiera odległości do danych wierzchołków
 - `no_elements` – liczba wierzchołków w tablicy `vertexes`.

Do struktury PriorityQueue zostały napisane następujące funkcje:

- `PQ_get` – zwróć liczbę całkowitą zawartą w kolejce, której odpowiada najmniejsza odległość oraz usuń ją z kolejki,
- `PQ_put` – dodaj daną liczbę całkowitą do kolejki wraz z odpowiadającą jej odległością.

5. Algorytm Dijkstry - algorytm umożliwiający odnalezienie najkrótszej możliwej drogi pomiędzy dwoma punktami w grafie

- Funkcje:
 - `initiate_values_dijkstra` – alokacja tablicy poprzedników oraz tablicy wag

- `relax` – potencjalnie przypisuje nowy poprzednik do kolejnego wierzchołka
- `dijkstra` – implementacja algorytmu Dijkstry, zwraca tablicę wszystkich poprzedników w grafie
- `determine_path` – otrzymawszy tablicę poprzedników wszystkich wierzchołków w grafie, uzyskaną w wyniku działania funkcji `dijkstra`, zwraca tablicę kolejnych wierzchołków składających się na drogę w kolejności odwrotnej
- `print_path` – wyświetla na ekran drogę między dwoma wierzchołkami w grafie
- `find_path_dijkstra` – funkcja przeznaczona do użycia w `main`, łączy wszystkie funkcje składające się na znalezienie najkrótszej ścieżki między dwoma wierzchołkami

6. Generatory

- `generate_complete_graph` – funkcja generująca graf kompletny i zwracająca go.
 - Generuje graf o wszystkich możliwych krawędziach w grafie łączących wierzchołki sąsiadujące pionowo i poziomo. Uzyskany graf ma zadaną liczbę kolumn i wierszy oraz wagi krawędzi mieszczące się w zadanym zakresie.
- `generate_connected_graph` – funkcja generująca graf spójny i zwracająca go.
 - Generuje graf spójny. Uzyskany graf ma zadaną liczbę kolumn i wierszy oraz wagi krawędzi mieszczące się w zadanym zakresie.
- `generate_random_graph` – funkcja generująca losowy graf i zwracająca go.
 - Generuje graf o losowym umiejscowieniu krawędzi. Uzyskany graf ma zadaną liczbę kolumn i wierszy oraz wagi krawędzi mieszczące się w zadanym zakresie.

3 Różnice względem planowanej wersji

1. Moduł Data Structures

- Struktury danych `Set` i `PriorityQueue` zostały przeniesione do nowego modułu `Data Structures`, ponieważ okazały się przydatne również w module `Generator`, a nie tylko w `Dijkstra`.
- Funkcja `PQ_get` okazała się zbyt obszerna, dlatego część funkcjonalności została przeniesiona do nowej funkcji `PQ_get_many_elements_in_pq`
- Dodanie funkcji `Set_is_element_in` sprawdzającej czy dany element znajduje się w `Set`
- Dodanie funkcji `Set_is_empty` sprawdzającej czy `Set` jest pusty
- Dodanie funkcji `Set_remove` usuwającej z `Set` element o danej wartości
- Dodanie funkcji `Set_pop` (zwracającej losowy element z `Set` i usuwający go tam)

2. Moduł Dijkstra

- Planowana funkcja `print_path` okazała się zbyt duża, więc trzeba było dodać funkcję `determine_path`, która przetworzy tablicę poprzedników otrzymaną w wyniku działania funkcji `dijkstra`, a następnie jest używana przez funkcję `print_path`.
- Dodanie funkcji `find_path_dijkstra` przeznaczonej do uruchomienia w `main`, która wywołuje po kolei wszystkie funkcje powiązane z algorytmem Dijkstry

3. Moduł Graph

- Macierz sąsiedztwa składa się z dwóch wskaźników – pierwszy wskazuje na wierzchołek, gdzie zaczyna się krawędź, a drugi – gdzie się kończy
- Dodanie funkcji `potential_neighbors` zwracającej tablicę wierzchołków sąsiadujących z danym wierzchołkiem
- Dodanie funkcji `neighbors` zwracającej tablicę wierzchołków, do których można dojść z danego wierzchołka
- Dodanie funkcji `print_graph` używanej do testów, aby sprawdzić, jak wygląda wygenerowany graf

- (e) Brak implementacji funkcji `is_connected` - to samo robi funkcja `bfs`

4. Moduł BFS

- (a) Dodanie funkcji `make_fifo` alokującej oraz inicjalizującej kolejkę FIFO
- (b) Dodanie funkcji `is_FIFO_empty` sprawdzającej czy w kolejce są wierzchołki oczekujące na bycie sprawdzonymi pod względem posiadania sąsiadów.
- (c) Planowana funkcja `bfs` miała zwracać tablicę poprzedników. Ostatecznie stała się ona główną funkcją sterującą działaniem algorytmu, która zwraca informację o tym czy graf jest spójny czy nie.
- (d) Brak implementacji funkcji `is_connected` oraz `does_exist_path`

5. Moduł Generator

- (a) Dodanie argumentów `int columns`, `int rows` w funkcjach generujących graf: `generate_complete_graph`, `generate_connected_graph`, `generate_random_graph`
- (b) dodanie funkcji `try_to_create_random_edge`, gdzie została przeniesiona część funkcjonalności z funkcji `generate_random_graph`
- (c) dodanie funkcji `add_edge_to_neighbor`, gdzie została przeniesiona część funkcjonalności z funkcji `generate_connected_graph`
- (d) dodanie funkcji `are_all_vertices_visited`

4 Testowanie

Testy jednostkowe zostaną napisane w języku C. Będą one uruchamiane używając osobnego pliku z metodą `main()`, w której zostaną wywołane funkcje porównujące zmienne z ich przewidywanymi prawidłowymi wartościami na dany moment. Planuje się następujące testy jednostkowe:

1. Moduł Graph – testy dla tego modułu zostały przeprowadzone na podstawie trzech różnych grafów
 - (a) `test_read_graph` – testuje funkcję `read_graph`
 - (b) `test_does_have_all_edges` – testuje funkcję `does_have_all_edges`
2. Moduł BFS – testy dla tego modułu zostały przeprowadzone na podstawie trzech różnych grafów
 - (a) `test_bfs` – testuje funkcję `bfs`
 - (b) testy funkcji `FIFO_get`, `FIFO_put` przeprowadzone są funkcji `bfs_test`
3. Moduł Dijkstra
 - (a) `test_Dijkstra` – testuje funkcje odpowiedzialne za algorytm Dijkstry
4. Moduł Generator
 - (a) `test_generate_complete_graph` testuje funkcję `generate_complete_graph`
 - (b) `test_generate_connected_graph` testuje funkcję `generate_connected_graph`
 - (c) `test_generate_random_graph` testuje funkcję `generate_random_graph`
5. Moduł Data Structures
 - (a) `test_Set` testuje funkcje związane ze strukturą `Set`
 - (b) `test_PQ` testuje funkcje związane ze strukturą `PriorityQueue`

5 Czego się dowiedzieliśmy?

- Wywołanie funkcji `srand(time(NULL))` zapewniającej większą losowość generowanych liczb pseudolosowych należy umieścić w kodzie tylko raz. W przypadku umieszczenia jej wielokrotnie, generowane liczby pseudolosowe będą zawsze takie same w ciągu jednej sekundy.

6 Co teraz zrobilibyśmy inaczej?

- Napisana przez nas funkcja generująca losowy graf spójny jest nieefektywna. Generowanie grafu o wielkości większej niż 50x50 zajmuje jej więcej czasu niż kilkanaście sekund, co wywołuje opóźnienie w całym programie. Z pozoru wydawało się, że współczesne komputery nie będą miały problemu z nieefektywną implementacją algorytmu, jednak efekt końcowy okazał się bardzo daleki od ideału. Następnym razem trzeba by skorzystać ze znanego algorytmu.
- Wykorzystana przez nas do przechowywania krawędzi macierz sąsiedztwa jest nieefektywna pamięciowo. Ogranicza ona liczbę krawędzi, którą program mógłby przechowywać. Następnym razem trzeba by skorzystać z bardziej efektywnej metody, jak np. lista sąsiedztwa.