Specyfikacja funkcjonalna programu do obsługi grafów

Filip Budzyński, Piotr Rzymowski
Politechnika Warszawska Wydział Elektryczny
3.03.2022

1. Cel projektu

Program napisany w języku C umożliwia tworzenie, zapisywanie i odczytywanie grafu o strukturze siatki. Tworzy graf o zadanej liczbie wierszy i kolumn oraz nadanych wagach krawędzi (generowanych losowo w podanym zakresie). Za pomocą algorytmu BFS program sprawdza spójność grafu. Z wykorzystaniem algorytmu Dijkstry potrafi znaleźć najkrótsze ścieżki pomiędzy wybraną parą węzłów.

2. Funkcje, jakie program udostępnia

- **Tworzenie grafu -** Program tworzy graf o podanej przez użytkownika liczbie kolumn i wierzchołków.
- **Zapisywanie grafu -** Program zapisuje stworzony graf do pliku o podanej nazwie.
- **Odczytywanie grafu -** Program potrafi odczytać graf z zapisanego pliku.
- **Sprawdzenie czy graf jest spójny (algorytm BFS) -** Wykorzystując algorytm BFS (Breadth-First Search) program sprawdza, czy graf jest spójny, czyli czy istnieją połączenie pomiędzy wszystkimi węzłami.
- **Znalezienie najkrótszych ścieżek (algorytm Dijkstry) -** Wykorzystując algorytm Dijkstry, program znajduje najkrótszą ścieżkę pomiędzy dwoma wybranymi węzłami.

3. Argumenty wywołania programu

Argumenty wywołania programu przy tworzeniu grafu, bez zapisu do pliku:

- **Generowanie grafu** - *./generate <kolumny:wiersze> [waga]<od-do>*
- **Algorytm BFS** - *./generate <kolumny:wiersze> [waga]<od-do> <tryb> <węzeł początkowy>*
- **Algorytm Dijkstry** - *./generate <kolumny:wiersze> [waga]<od-do> <algorytm> <węzeł początkowy> <węzły końcowe>*

kolumny:wiersze - x:y	Liczby naturalne x, y x, y ∈ $\mathbb{N}$ oraz x, y ≥ 0
waga, od-do - x-y	Liczby rzeczywiste x, y x, y ∈ $\mathbb{R}$ przy czym x $\leq$ y
Tryb	„bfs” lub „dijkstra”
węzeł startowy - x	0 ≤ x ≤ n-1, gdzie x – numer węzła, x ∈ $\mathbb{N}$, n - liczba węzłów w grafie
węzły końcowe - y_1, y_2, y_3 …	0 ≤ y ≤ n-1, gdzie y – numer węzła, y ∈ $\mathbb{N}$, n - liczba węzłów w grafie

Argumenty wywołania programu przy tworzeniu grafu z zapisem do pliku:

- **Generowanie grafu** - *./graf <nazwa pliku> <kolumny:wiersze> [waga]<od-do>*

nazwa pliku	kolumny:wiersze x:y	waga, od-do x-y
Nazwa stworzonego pliku (format .txt)	Liczby naturalne x, y; x, y ∈ $\mathbb{N}$ oraz x, y ≥ 0	Liczby rzeczywiste x, y; x, y ∈ $\mathbb{R}$ przy czym x $\leq$ y

Argumenty wywołania programu przy odczytywaniu grafu z pliku:

- **Czytanie -** ./read <nazwa pliku>
- **Algorytm BFS -** ./bfs <nazwa pliku> <węzeł początkowy>
- **Algorytm Dijkstry -** ./dijkstra <nazwa pliku> <węzeł początkowy> <węzły końcowe>

nazwa pliku	węzeł początkowy x	węzły końcowe y_1, y_2, y_3 …
Nazwa istniejącego pliku (format .txt)	0 $\leq$ x $\leq$ n-1, gdzie x – numer węzła i x ∈ $\mathbb{N}$, n - liczba węzłów w grafie	0 $\leq$ y $\leq$ n-1, gdzie y – numer węzła i x ∈ $\mathbb{N}$, n - liczba węzłów w grafie

4. Przykłady argumentów wywołania

Przykład wywołania programu przy tworzeniu grafu:

- **Generowanie grafu bez zapisu do pliku** - ./generate 100:30 0-1
  Program wygeneruje graf 100 kolumn na 30 wierszy z wagami między generowanymi losowo wagami krawędzi z przedziału od 0 do 1.
- **Generowanie grafu z zapisem do pliku** - ./graf graf01.txt 50:20 1-7
  Program stworzy plik graf01.txt z wygenerowanym grafem 50 kolumn na 20 wierszy z generowanymi losowo wagami krawędzi z przedziału od 1 do 7).

Przykład wywołania programu przy odczytywaniu grafu:

- **Algorytm BFS -** ./bfs graf01.txt
  Program odczyta plik graf01.txt i określi jego spójność przy użyciu algorytmu przeszukiwania wszerz (BFS - Breadth First Search).
- **Algorytm Dijkstry -** ./dijkstra graf01.txt 0 19 8
  Program odczyta plik graf01.txt i wyznaczy najkrótsze ścieżki od węzła 0 do węzła o numerze 19 i od 0 do węzła o numerze 8, przy użyciu algorytmu Dijkstry.

5. Przykład pliku z zapisanym grafem

```
7 4
        1 :0.8864916775696521  4 :0.2187532451857941
        5 :0.2637754478952221  2 :0.6445273453144537  0 :0.4630166785185348
        6 :0.8650384424149676  3 :0.42932761976709255  1 :0.6024952385895536
        7 :0.5702072705027322  2 :0.86456124269257
        8 :0.9452864187437506  0 :0.8961825862332892  5 :0.9299058855442358
        1 :0.5956443807073741  9 :0.31509645530519625  6 :0.40326574227480094  4 :0.44925728962449873
        10 :0.7910000224849713  7 :0.7017066711437372  2 :0.20056970253149548  5 :0.3551383541997829
        6 :0.9338390704123928  3 :0.797053444490967  11 :0.7191822139832875
        4 :0.7500681437013168  12 :0.5486221194511974  9 :0.25413610146892474
        13 :0.8647843756083231  5 :0.8896910556803207  8 :0.4952122733888106  10 :0.40183865613683645
        14 :0.5997502519024634  6 :0.5800735782304424  9 :0.7796297161425758  11 :0.3769093717781341
        15 :0.3166804339669712  10 :0.14817882621967496  7 :0.8363991936747263
        13 :0.5380334165340379  16 :0.8450927265651617  8 :0.5238810833905587
        17 :0.5983997022381085  9 :0.7870744571266874  12 :0.738310558943156  14 :0.45746700405234864
        10 :0.8801737147065481  15 :0.6153113201667844  18 :0.2663754517229303  13 :0.22588495147495308
        19 :0.9069409600272764  11 :0.7381164412958352  14 :0.5723418590602954
        20 :0.1541384547533948  17 :0.3985282545552262  12 :0.29468967639003735
        21 :0.7576872377752496  13 :0.4858285745038984  16 :0.28762266137392745  18 :0.6264588252010738
        17 :0.6628790185051667  22 :0.9203623808816617  14 :0.8394013782615275  19 :0.27514794195197545
        18 :0.6976948178131532  15 :0.4893608558927002  23 :0.5604145612239925
        24 :0.8901867253885717  21 :0.561967244435089  16 :0.35835658210649646
        17 :0.8438726714274797  20 :0.3311114339467634  25 :0.7968809594947989  22 :0.9281943906422196
        21 :0.6354858042070723  23 :0.33441278736675584  18 :0.43027465583738667  26 :0.3746522679684584
        27 :0.8914256412658524  22 :0.8708278171237049  19 :0.4478162295166256
        20 :0.35178269705930043  25 :0.2054048551310126
        21 :0.6830700124292063  24 :0.3148089827888376  26 :0.5449034876557145
        27 :0.2104213229517653  22 :0.8159939689806697  25 :0.4989269533310492
        26 :0.44272335750313074  23 :0.4353604625664018
```

W powyższym pliku zapisany jest graf o 7 kolumnach i 4 wierzchołkach (pierwsza linijka). W drugiej linii pliku podane są wagi krawędzi pomiędzy wierzchołkiem o numerze 0 i wierzchołkami 1 i 4. W każdej kolejnej linii analogicznie podane są wagi krawędzi pomiędzy wierzchołkiem o numerze o jeden większym i wierzchołkami o numerach wypisanych w danej linii przed znakiem „:”.

W powyższym pliku zapisany jest graf o 7 kolumnach i 4 wierzchołkach (pierwsza linijka). W drugiej linii pliku podane są wagi krawędzi pomiędzy wierzchołkiem o numerze 0 i wierzchołkami 1 i 4. W każdej kolejnej linii analogicznie podane są wagi krawędzi pomiędzy wierzchołkiem o numerze o jeden większym i wierzchołkami o numerach wypisanych w danej linii przed znakiem „:”.

6. Obsługa sytuacji wyjątkowych

Brak nazwy pliku

W przypadku niepodania nazwy pliku program wyświetli komunikat: “nie podano nazwy pliku”.

Zły format pliku

W przypadku złego formatu pliku do odczytu, program poinformuje nas o tym, iż nie potrafi przeczytać pliku w takim formacie.

Wygenerowanie pliku o istniejącej nazwie

W przypadku generowania grafu i podania nazwy istniejącego już pliku, program poinformuje nas, że plik ten nie jest pusty i nie może zapisać pliku o takiej nazwie, gdyż plik taki już istnieje.

Podanie nieistniejących węzłów przy algorytmie Dijkstry i BFS

W przypadku wpisania nieistniejących węzłów, program poinformuje nas o błędnym ich podaniu.

Podanie nieistniejącej krawędzi przy edytowaniu grafu

Program poinformuje nas o tym że podane połączenie (krawędź) nie istnieje i nie zmieni grafu w pliku.

Brak argumentów wywołania

W przypadku braku argumentów wywołania, program podaje prostą instrukcję prawidłowego wywołania programu.

Brak pliku do zapisu

W przypadku niepodania ścieżki do pliku, w którym graf ma zostać zapisany, program wyświetla komunikat o błędzie: „Błąd! Nie podano ścieżki do zapisu grafu”.

Brak pliku do odczytu

W przypadku podania ścieżki do nieistniejącego pliku do odczytu program wyświetla komunikat: „Nie podano pliku do odczytu grafu” i generuje graf o wymiarach domyślnych (10x10).

Zły format liczby kolumn

Program wyświetla komunikat o błędzie: „Błąd! Nieprawidłowy parametr liczby kolumn”.

Zły format liczby wierszy

Program wyświetla komunikat o błędzie „Błąd! Nieprawidłowy parametr liczby wierszy”.

Zły format zakresu do losowania wag krawędzi

Program wyświetla komunikat o błędzie „Błąd! Nieprawidłowy parametr zakresu wag.

Specyfikacja implementacyjna programu do obsługi grafów

Filip Budzyński, Piotr Rzymowski
Politechnika Warszawska Wydział Elektryczny
3.03.2022

1. Struktura plików źródłowych

Program zbudowany jest na zasadzie modularnej. Oznacza to, że składa się z plików zawierających odpowiednie funkcje, każdy z nich odpowiada za inną funkcjonalność programu.

Graf

- **graf.c** - Zawiera funkcję niezbędne do tworzenia grafu tj. createGraph(), printGraph(), randomFrom(min, max) oraz struktury: Node i Graph (w formie listy sąsiedztwa).

Algorytmy

- **bfs.c, dijkstra.c** - Pliki zawierające implementację algorytmów BFS oraz Dijkstra.
- **queue.c** – Plik zawierający implementację kolejki FIFO.
- **mpq.c**– Plik zawierający implementację kolejki priorytetowej opartej na kopcu.

Main

- **main.c** - Plik zawierający obsługę argumentów wywołania podczas tworzenia grafu z zapisem do pliku.
- **generate.c –** Zawiera obsługę argumentów wywołania podczas generowania grafu bez zapisu do pliku.
- **readmain.c** – Plik zawierający obsługę argumentów wywołania podczas czytania grafu z pliku.
- **bfsmain.c –** Plik odpowiedzialny za obsługę argumentów wywołania przy użycia algorytmu BFS, odczytując graf z pliku.
- **dijkstramain.c -** Plik odpowiedzialny za obsługę argumentów wywołania przy użycia algorytmu BFS podczas odczytania grafu z pliku.

Utils

- **util.c –** Zawiera funkcje pomocnicze używane przez inne moduły: *Graf, Main, Algorytmy.*

Tests

- **graph_test.c, queue_test.c, bfs_test.c, mpq_test.c, dijkstra_test.c** – Pliki zawierające testy jednostkowe funkcji zdefiniowanych w podanych powyżej plikach.

2. Opis zawartości plików źródłowych

Graf

- **graf.c** - plik zawierający funkcję do tworzenia grafu: createGraph(), printGraph() oraz funkcję pomocniczą randomFrom(min, max) i addToList().
- **graf.h** - plik nagłówkowy zawierający prototypu funkcji z graf.c oraz struktury: Node, Graph.
- **Struktury dla *graf.c***

```
typedef struct Node{
```

```
  int dest;
  double weight;
  struct Node* next;
} Node_t;

typedef struct Graph{
  int k, w;
  struct Node** adjlist;
} Graph_t;
```

ALGORYTMY

BFS

- **bfs.c** - plik zawierający implementację algorytmu BFS do określenia spójności grafu. Algorytm oparty jest o kolejkę FIFO (First in first out).
- **queue.c** – plik z implementacją kolejki FIFO stworzoną za pomocą listy powiązań(and. Linked list).
- **queue.h –** plik nagłówkowy zawierający prototypy funkcji kolejki FIFO z pliku „queue.c" oraz struktury „Qnode" i „Queue".
- **Struktury dla *queue.c***

```
typedef struct Qnode{
    int data;
    struct Qnode* next;
} Qnode_t;

typedef struct Queue{
    struct Qnode* front;
    struct Qnode* rear;
} Queue_t;
```

Dijkstra

- **dijkstra.c** - plik zawierający implementację algorytmu Dijkstry, do znajdowania najkrótszych ścieżek między wybranymi parami węzłów. Oparty o kolejkę priorytetową na kopcu.
- **mpq.c –** plik z implementacją kolejki priorytetowej z wykorzystaniem kopca (ang. Heap).
- **mpq.h –** plik nagłówkowy zawierający prototypy funkcji kolejki priorytetowej z pliku „mpq.c" oraz struktury „**hn** – heap node" i „**pq** – priority queue".
- **Struktury dla *mpq.c***

```
typedef struct hn{
  double data;
  int v;
} *hn;

typedef struct pq{
  hn *arr;
  int *heapplace;
  int size;
} *pq;
```

MAIN

- **main.c** - Plik zawierający implementację obsługi argumentów wywołania. Sprawdza ich poprawność a także czy istnieje podana nazwa pliku (czy można plik o takiej nazwie

wygenerować bądź czy takowy już istnieje). Wykorzystuje pliki i funkcje z modułu *Graf* w celu poprawnego wygenerowania grafu.

- **generate.c** – Plik obsługujący argumenty wywołania, sprawdza ich poprawność. Wywołuje funkcję z modułu *Algorytmy* w celu użycia wybranej funkcjonalności programu.
- **readmain.c** - Plik do obsługi argumentów podczas czytania grafu. Czyta graf z podanego pliku sprawdzając przy tym poprawność argumentów oraz czy plik o podanej nazwie istnieje i można go otworzyć do pisania.
- **bfsmain.c** - Plik obsługujący i sprawdzający poprawność argumentów wywołania podczas uruchomienia programu z funkcją określania spójności grafu poprzez algorytm bfs. Posługuje się funkcjami zdefiniowanymi w „bfs.c".
- **djikstramain.c** – Plik obsługujący i sprawdzający poprawność argumentów wywoałania podczas uruchomienia programu do znajdowania najkrótszej ścieżki pomiędzy parą węzłów. Posługuje się funkcjami zdefiniowanymi w „dijkstra.c".
- **utils.c** - Plik zawierający prostą instrukcję prawidłowego wywołania programu w przypadku, gdy użytkownik nie poda argumentów wywołania.

Utils

- **utils.c** – Plik zawiera funkcje pomocnicze:
  - *isNumber()* – Sprawdza czy podany do funkcji *char* jest liczbą.
  - *randfrom()* – Wybiera losową wartość z podanych do funkcji przedziałów (min – max).
  - *helper()* – Funkcja wypisująca prostą instrukcję używania programu.

3. Definicje struktur danych

- **Graf (**nieskierowany lub skierowany w obie strony) - To struktura danych łącząca ze sobą węzły w odpowiedni sposób za pomocą krawędzi. Węzły połączone są w tzw. "kratkę". To oznacza, że węzły znajdujące się na rogach mają tylko dwie krawędzie (czyli także dwa sąsiadujące węzły), węzły tworzące "ramkę" mają trzy połączenia, a węzły znajdujące się w środku cztery. Krawędzie mają losowo wyznaczoną wagę. Przykładowo krawędź 0 do 1 ma tą samą wagę (bądź różną przy grafie skierowanym) jak krawędź 1 do 0.
- **Kopiec (Min heap)** – Wykorzystywany przy algorytmie Dijkstry do zaimplementowania kolejki priorytetowej. To struktura danych umożliwiająca dostanie najmniejszego elementu(względem określonego warunku, u nas wagi krawędzi) w czasie O(1), gdyż znajduje się on zawsze na szczycie kopca, więc mamy do niego natychmiastowy dostęp. Węzły wychodzące od mniejszego wartością węzła(ojca) nazywamy jego synami i mają zawsze wagę większą niż ojciec. Ojciec posiada maksymalnie dwóch synów, a każdy z nich staje się ojcem dla własnych następników.
- **Kolejka FIFO (First in first out)** – Abstrakcyjna struktura danych pozwalająca wydobywać elementy w uporządkowany sposób. Pierwszy który został dodany, zostanie jako pierwszy z niej wyciągnięty.
- **Kolejka priorytetowa** – Abstrakcyjna struktura danych wykorzystująca kopiec. Element o najmniejszej wartości zawsze stoi na początku kolejki. Reszta elementów nie jest posortowana. Dodając i odejmując z kolejki priorytetowej mamy pewność, że zawsze na jej początku znajdzie się element z najmniejszą wartością.

4. Testy

Funkcje testów jednostkowych wybranych funkcji są zakończone dopiskiem „_test.c” w nazwie pliku. Przed wprowadzeniem do funkcji wartości niepożądanych chronią je obsługi argumentów wywołania. Każdy test jednostkowy składa się z trzech fragmentów:

Przeprowadzenie testów

- Pierwszy blok testu to przygotowanie do użycia funkcji np. Stworzenie grafu, wybranie węzła startowego, nadanie wartości wykorzystywanych w kolejce FIFO lub kolejce priorytetowej.
- Drugim fragmentem jest wywołanie pojedynczej funkcji. Funkcje wywoływane są w takiej kolejności, by w następnych funkcjach wykorzystywać tylko te które zostały już przetestowane.
- Trzecia część testu to sprawdzenie, czy funkcja wykonała to czego się spodziewaliśmy (bądź czy zwróciła nam przewidywane, pożądane dane).

Pliki z testami

- **graph_test.c** – Testuje funkcję zaimplementowane w pliku *graph.c*. Dla przykładu funkcja *creategraph_test()* sprawdza czy poprawnie zostanie utworzona lista sąsiedztwa, oraz czy wartość węzłów znajdują się w zadanym przedziale.
- **queue_test.c** – Plik testując funkcje znajdujące się w pliku *queue.c*. Sprawdza poprawność działa funkji *enqueue()* oraz *dequeue()*. Jeżeli przy pierwszym wyjęciu z kolejki, dostaniemy wartość włożoną na samym początku to funkcja działa poprawnie.
- **mpq_test.c** – Plik testuje funkcje przeznaczone dla kolejki priorytetowej. Sprawdza poprawność funkcji *push()* i *pop()*. Jeżeli dodając do kolejki za pomocą funkcji *push()*, wartość na pierwszym miejscu kolejki jest najmniejszą z dodanych to funkcje działają poprawnie.
- **bfs_test.c** – Sprawdza czy funkcja *bfs()* zwraca poprawną tablicę „kolorów” czyli odwiedzonych węzłów. Jeżeli cała tablica zawiera kolor czarny, to osiągnięty został sukces.
- **dijkstra_test.c** – Testuje funkcję *dijkstra()* sprawdzając czy zwrócone przez nią dwie tablice (dystansów oraz ojców) są wypełnione poprawnie. Jeżeli tablica dystansów zawiera wartości inne niż *INF* oraz tablica ojców zawiera wartości różne od „-1” (oprócz wartości dla węzła startowego), to funkcja działa poprawnie, tj. dostarcza nam pożądane i przewidywane wartości.