Specyfikacja implementacyjna programu służącego do analizy grafów.

Filip Sosnowski, Krzysztof Tadeusiak

31.03.2022

Informacje ogólne

Program służy do analizy grafu wygenerowanego na podstawie podanych przez użytkownika argumentów. Umożliwia znalezienie najkrótszej drogi pomiędzy wybraną parą węzłów, pozwala sprawdzić spójność grafu, zapisać go do pliku wyjściowego, jak i odczytać z pliku wejściowego. Program uruchamia się w środowisku Linux. Wygenerowane dane są zapisywane do pliku tekstowego. W naszym projekcie wykorzystane jest działanie algorytmu Breadth-first Search (BFS) oraz algorytmu Dijkstry.

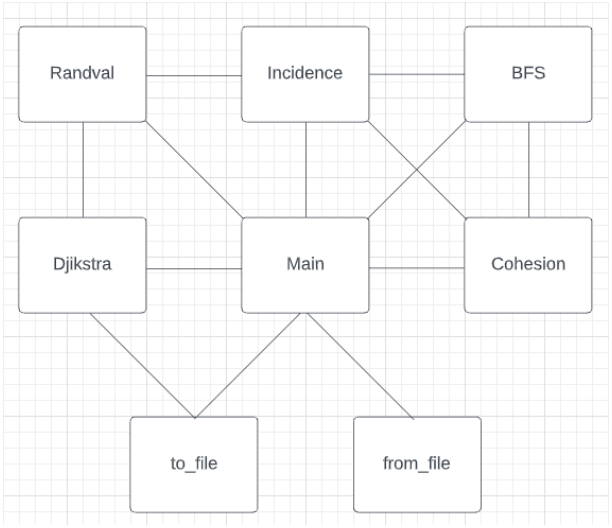
Opis modułów

Program będzie składać się z sześciu modułów:

- **main** - moduł główny, odpowiedzialny za łaczenie wszystkich poniższych modułów. Ma za zadanie umożliwić użytkownikowi wybrać odpowiednie argumenty wywołania i wartości parametrów, a także sprawdzić ich poprawność. Obsługuje funkcje z innych modułów.
- **incidence** - moduł odpowiedzialny za tworzenie macierzy incydencji o wymiarach $x \times y$, gdzie: x$\leq 10^8$, y $\leq 10^8$, $x \times y \leq 10^8$.
- **bfs** - moduł odpowiedzialny za przeszukiwanie grafu metodą breadth-first search (przeszukiwanie wszerz). Służy do ustalenia spójności grafu.
- **djikstra** - moduł odpowiedzialny za znalezienie najkrótszej drogi pomiędzy wybranymi przez użytkownika wierzchołkami.
- **randval** - moduł odpowiedzialny za wypełnienie macierzy incydencji wartościami pseudolosowymi od 0 do 1 lub zakresami podanymi przez użytkownika.
- **cohesion** - moduł odpowiedzialny za n-krotne podzielenie grafu gdzie: n ≥ 1, $n \leq \frac{x \times y}{4}$ oraz $n \in C$.

- **to_file** - moduł odpowiedzialny za zapisywanie parametrów wygenerowanego grafu do pliku.
- **from_file** - moduł odpowiedzialny za odczytywanie pliku z zapisanym grafem.

Wizualizacja zależności pomiędzy modułami:



Opis funkcji

Moduł main:

- plik main.c
- void main(int argc, char**argv)

Moduł `main` ma za zadanie scalać pozostałe moduły programu analizującego graf. W tym module został zaimplementowany wybór wartości określonych parametrów poprzez argumenty wywołania i odpowiednie flagi. Zawarte są w nim także odpowiednie warunki odczytywania flag i dopuszczalne zakresy podanych przez użytkownika argumentów.

`Main` ma również na celu wyświetlić komunikaty błędów dotyczących nieprawidłowo określonych przez użytkownika argumentów wywołania. Po wprowadzeniu błędnych wartości, program wyświeli komunikat błędu informujący o złym wyborze wartości parametru oraz poinformuje o dozwolonych zakresach. Przykładem takiego komunikatu jest:

`"Nieprawidlowa wartosc 'x = -2'. (0 < x < 10`^{8}`)".`

`Komendą, która zostanie wykorzystana do uruchomienia programu, jest: ./graf [argumenty_wywołania]`

Nasz program przyjmuje poniżej wymienione argumenty wywołania:

- -x (liczba) określa liczbę kolumn w grafie;
  Jest to flaga obowiązkowa.
- -y (liczba) określa liczbę wierszy w grafie;
  Jest to flaga obowiązkowa.
- -n (liczba) określa liczbę grafów otrzymanych z pierwotnie wygenerowanego grafu;
  domyślnie n = 1;
- --min (liczba) lub -l (liczba) określa minimalną wartość wagi krawędzi między węzłami;
  domyślnie min = 0;
- --max (liczba) lub -g (liczba) określa maksymalną wartość wagi krawędzi między węzłami;
  domyślnie max = 1;
- --ps (numer_węzła) lub -s (numer_węzła) określa punkt startowy;
  domyślnie ps = 0;
- --pk (numer_węzła) lub -e (numer_węzła) określa punkt końcowy;
  domyślnie pk = $x \times y$-1 ;
- --out (nazwa_pliku) lub -o (nazwa_pliku) określa nazwę pliku, do którego zostaną zapisane dane;
  domyślnie out = mygraph;

Projekt ten może działać także w drugim trybie. Program potrafi przeczytać graf z pliku o ustalonym formacie. Służy do tego flaga:

- --in (nazwa_pliku) lub -i (nazwa_pliku) określa nazwę pliku, z którego zostaną odczytane dane;

W tym przypadku dopuszczalne są jedynie flagi -ps (numer_węzła) oraz -pk (numer_węzła) określające punkt startowy oraz końcowy, które posłużą do znalezienia najkrótszej drogi w odczytanym z pliku grafie.

Zakresy wartości parametrów:

- x > 0, x $\leq$ 10^8, x$\times y$ $\leq 10^8$ oraz $x \in C$ (x jest zmienną typu int);
- y > 0, y $\leq$ 10^8, x$\times y$ $\leq 10^8$ oraz $y \in C$ (y jest zmienną typu int);
- min $\geq 0, min \leq 1$ oraz min$\in R$ (min jest zmienną typu double);
- max $\geq 0, max \leq 1$ oraz max$\in R$ (max jest zmienną typu double);

- ps $\geq 0, ps \leq$ x$\times y - 1$ oraz $ps \in C$ (ps jest zmienną typu int);
- pk $\geq 0, pk \leq$ x$\times y - 1$ oraz $pk \in C$ (pk jest zmienną typu int);
- n $\geq 1, n \leq \frac{x \times y}{4}$ oraz $n \in C$ (n jest zmienną typu int);

Ważnym elementem wywoływania programu jest to, że argumenty x oraz y powinny być wywołane w pierwszej kolejności, gdyż bez nich nie mogłby być określone domyślne wartości pominiętych argumentów. W innym przypadku zostanie wypisany komunikat błędu o nastepującej (przykładowej) treści: "Przed podaniem wartosci 'ps', prosze okreslic parametr 'x' oraz 'y'".

Kolejnym możliwym błędem jest podanie zbyt długiej nazwy pliku. Domyślnie, długość ta ustalona jest na 32 znaki. W przypadku jej przekroczenia, wypisany zostanie komunikat błędu o przykładowej treści: "Zbyt dluga nazwa pliku wejsciowego: (długość nazwy). Maksymalna dlugosc to: 32".

Program poinformuje o braku możliwości odczytania podanego pliku. Wyświetli komunikat: "Nie mozna odczytac pliku (nazwa)".

W celu wyświetlenia instrukcji użycia programu, należy uruchomić program bez argumentów, bądź skorzystać z flagi -h lub --help.

Moduł Makefile:

- plik Makefile

Makefile ma za zadanie zautomatyzować komilację oraz pracę programu do analizy grafów. Jego podstawowym założeniem jest ułatwienie kompilowania wzajemnie zależnych od siebie plików i otrzymanie jednego piku końcowego. Dodatkowa funkcja to "make clean", która usuwa wcześniej stworzone pliki. Kolejną funkcją jest "make test". Komenda ta spowoduje wytworzenie kilku testów, które mają potwierdzić zgodność i poprawność działania całego programu.

Moduł incidence:

- plik incidence.c
- double *incidence(int x, int y)

Funkcja incidence ma za zadanie stworzyć dwuwymiarową tablice wypełnioną wartościami 0 lub 1. Wartość "1" odwzorowuje istniejący węzeł pomiędzy wierzchołkami, co za tym idzie, możliwość poruszania się między nimi. Wartość "0" odwzorowuje brak węzła, co oznacza niemożność poruszania się miedzy określoną parą wierzchołków. W celu implementacji skorzystaliśmy z podwójnej pętli for, w której iterowaliśmy po wierszach oraz kolumnach macierzy. W zależności od położenia węzłów, należało zapisać odpowiednie warunki, by

generacja macierzy sąsiedztwa była poprawna. Na podstawie tych warunków, wartości "1" były wpisywane w odpowiednie miejsce tablicy dwuwymiarowej. Moduł ten zwraca macierz incydencji.

Moduł bfs:

- plik bfs.c
- struct kolejka

  Struktura kolejki jest niezbędnym elementem wykorzystywanym w algorytmie BFS. Umożliwia on poruszanie się między węzłami i zapamiętywanie odwiedzone wczesniej węzły. W naszym programie wykorzystywać będziemy kolejkę typu FIFO. Początkowo kolejka składa się tylko z wierzchołka startowego. Dokładani są później do niej sąsiedzi pierwszego wierzchołka. Dopóki kolejka nie jest pusta, pobieramy pierwszej element (usuwając go z kolejki). Następnie sprawdzamy, który z jego sąsiadów nie ma ustawionego poprzednika. Dla każdego nieodwiedzonego dotychczas sąsiada należy ustawić odległość od wybranego elementu z kolejki. Następnie należy ustawić poprzednika na aktualnie wybrany element. Algorytm kończy się gdy kolejka będzie pusta.

- int kpusta(struct kolejka *k)

  Funkcja kpusta sprawdza, czy kolejka jest pusta. Służy to zakończeniu funkcjonowania algorytmu BFS, gdy nie ma już elementów w kolejce funkcja zwróci wartości 1. Wykorzystuje ona wskaźnik na strukturę kolejka.

- int kpelna(struct kolejka *k)

  Funkcja kpelna sprawdza, czy kolejka jest pełna. Służy to zachowaniu początkowej zdefinowanej długości kolejki oraz powiekszaniu jej rozmiaru wedle potrzeby użytkownika. Jeśli nie ma miejsca w kolejce, funkcja zwróci wartości 1. Wykorzystuje ona wskaźnik na strukturę kolejka.

- void wstaw(struct kolejka *k, int w)

  Funkcja wstaw ma za zadanie powiekszyć rozmiar kolejki oraz wstawić do niej nowo odczytaną przez funkcje BFS wartość.

- int zwolnij(struct kolejka *k)

  Funkcja zwolnij ma za zadanie zwolnić z kolejki numer napotkanego wierzchołka i przydzielić go do sekcji odwiedzonych.

Moduł ten zwróci komunikat na wyjście standardowe o tym, czy graf jest spójny. Informacja ta zostanie zapisana również do pliku.

Moduł djikstra:

- plik djikstra.c
- void djikstra(double *arr, int x, int y, int ps, int pk)

Funkcja djikstra ma za zadanie odnaleźć najkrótszą drogą w grafie poprzez poruszanie się po jego macierzy incydencji z wagami. Jej parametrami jest punkt startowy(ps), punkt końcowy(pk), szerokość grafu (x), wysokość grafu (y) oraz wcześniej wspomniana macierz incydencji z wagami (arr).

Funkcja faworyzuje wagi o mniejszej wartości(chętniej je wybieraja i poruszaja się nimi). Funkcja ta zamienia na początku w przekazanej macierzy incydencji zera na liczby niewykorzystane przez nas do odwzorowania grafu (np. 2), odbierając tym samym możliwość poruszania się po nich do punktu końcowego. Dla każdego wierzchołka przypisywane są dwie wartości: dystans oraz poprzednik. Początkowo każdy wierzchołek ma dystans nieskończony, a poprzednik jest nieznany. Wyjątkiem jest wierzchołek początkowy dla którego dystans wynosi 0.

Dalszy etap polega na utworzeniu kolejki wszystkich wierzchołków. Nastpnie dopóki kolejka nie jest pusta należy usunąć wierzchołek o najmniejszym kluczu dystansie i zbadać dla niego wszystkie połączone wierzchołki. Jeśli połączenie z wybranego wierzchołka do sąsiada ma łącznie mniejszą wartość niż docelowy wierzchołek to należy zaktualizować dystans oraz sąsiada.

Wynikiem działania tego modułu będzie wypisanie najkrótszej drogi na strumień wyjściowy stdout i do pliku.

Moduł randval:

- plik randval.c
- double *randval(double *arr, int x, int y, double min, double max)

Funkcja randval ma na celu wypełnić podaną macierz wartościami losowymi. Funkcja pobiera wskaźnik na tablicę arr, a następnie podczas iteracji w pętli przy napotkaniu wartości równej 1, nadpisuje ją pseudolosową wartością z przedziału od **min** do max, a także tą samą wartość przejścia przypisuje w odpowiednim miejscu w macierzy (odwzorowującym tę samą krawędź). Funkcja zwraca nową macierz, dla której zostały przypisane nowe, pseudolosowe wartości przejść.

Moduł cohesion:

- plik cohesion.c

- double *cohesion(double *arr, int start)

Funkcja cohesion ma za zadanie podzielić graf (reprezentowany przez macierz incydencji) tyle razy, ile określił użytkownik poprzez podanie parametru n. Dokonane jest to poprzez wybór początkowego punktu podziału położonego na krańcu grafu, następnie poprzez losowe wybranie przejścia po wierzchołkach i utworzenie przecięcia. Przecięcie kończy się po dotarciu do wierzchołka na jednej z krawędzi grafu. Opisany powyżej podział reprezentowany jest poprzez zastąpienie wartości 1 z macierzy incydencji wartościami 0, które odwzorowują brak możliwości dostania się z jednego wierzchołka do drugiego. Podział nie może zakończyć sie na wierzchołku, z którego wychodzą 4 krawędzie. Oznaczałoby to wtedy niepoprawność działania programu i brak podziału grafu. Moduł zwraca macierz.

Moduł to_file:

- plik to_file.c
- void to_file(int x, int y, double *arr, char** output)

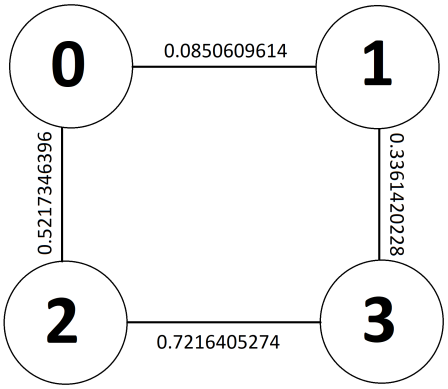
Funkcja to_file ma za zadanie zapisać dane o grafie do odpowiedniego pliku o nazwie output. Przy wykorzystaniu macierzy arr, funkcja sczytuje odpowiednie wartości oznaczające konkretną wartość wagi krawędzi oraz zapisze je do pliku wyjściowego, tak, by zachować odpowiedni format pliku z informacjami o grafie. W pierwszej linii zapisane zostaną wymiary x oraz y grafu, a następnie w nowych liniach dla kolejnych węzłów odpowiednie sąsiadujące z nimi węzły. Po każdym węźle po dwukropku wypisana zostanie wartość przejścia.

Plik wyjściowy ma przyjmować następujący format:

```
2 2
        1: 0.0850609614 2: 0.5217346396
        0: 0.0850609614 3: 0.3361420228
        0: 0.5217346396 3: 0.7216405274
        1: 0.3361420228 2: 0.7216405274
```

W pierwszej linijce określona jest liczba kolumn oraz wierszy. Kolejne wiersze opisują wartości przejść pomiędzy konkretnymi węzłami w rozważanym grafie, zaczynając od pierwszego węzła i kończąc na węźle ostatnim.

Jest to odwzorowaniem poniższego grafu przykładowego:



Moduł from_file:

- plik from_file.c
- void from_file(char** input)

Funkcja from_file ma na celu odczytać graf z pliku o określonym formacie (opisanym powyżej) oraz podanej nazwie: input. Funkcja przeanalizuje tekst, odczyta z pierwszej linii tekstu wymiary grafu, a sąsiedztwo węzłów oraz rozpoznane wartości wag zapisze do dynamicznie alokowanej tablicy dwuwymiarowej, która posłuży w dalszym funkcjonowaniu programu.

W przypadku wykrycia złego formatu pliku wejściowego, program wypisze komunikat błędu: "Zły format pliku wejściowego (nazwa pliku)".

Testowanie

Program ten zostanie przetestowany pod kątem poprawności generowanych plików, odczytu zewnętrznych plików o określonym formacie oraz prawidłowości działania algorytmów analizy grafu.

Po pierwsze utworzone zostaną dwa pliki, które posłużą do ocenienia działania generacji wartości losowych krawędzi przejścia. W tym celu skorzystamy z komendy diff [OPCJE]... PLIKI, by porównać ich zawartość. Jeżeli będą się od siebie różnić, oznacza to, że generacja pseudolosowych wartości wag działa prawidłowo (nie zostanie wygenerowana ta sama sekwencja liczb).

Dla grafów o wymiarach $x \times y \leq 25$ dokonamy analizy manualnej, to znaczy osobiście przestudiujemy wygenerowany graf oraz określimy spodziewane rezultaty. Określimy najkrótszą drogę pomiędzy wybranymi punktami, a także określimy spójność tego grafu. Test zostanie zakończony sukcesem, jeżeli wyniki te będą równe rozwiązaniom

uzyskanym za pomocą naszego programu. Pozwoli to ocenić poprawność zaimplementowanego działania algorytmu BFS oraz Dijkstry.

By przetestować prawidłowość działania modułu from_file.c (czytacz), utworzymy plik o złym formacie wprowadzonych danych. Jeśli funkcja ta zwróci błąd, bedzie to wskazywać, że działa poprawnie, a test ten zostanie zrealizowany pozytywnie.

Przeprowadzimy także test wprowadzenia nieprawidłowych wartości poszczególnych parametrów oraz złego podania argumentów wywołania. Program powinien poinformować użytkownika o niepoprawnie wprowadzonych danych poprzez wyświetlenie adekwatnego komunikatu błędu oraz wypisanie instrukcji użycia programu.

W celu sprawdzenia prawidłowości modułu to_file.c, przy użyciu naszego programu zostanie wygenerowany plik z grafem oraz otrzymamy rezulat uzyskany przy pomocy algorytmu Dijkstry oraz algorytmu BFS. Następnie uruchomimy program ponownie przy użyciu wygenerowanego właśnie pliku. Jeśli wynik działania programu będzie identyczny z poprzednimi rezultatami, to test zostanie uznany za pozytywny.

Dodatkowo, z pomocą pliku Makefile, zautomatyzujemy sprawdzenie działania programu poprzez przeprowadzenie kilku testów (w tym wyżej opisanych), które pomogą ustalić poprawność kluczowych funkcji naszego projektu.