

Specyfikacja funkcjonalna

Marcin Kowalski i Jakub Orzełowski

03.03.2022

Streszczenie

Dokument stanowi opis projektu BFS i Algorytm Dijkstry, oraz związany z nim zakres prac a także funkcje programu.

Spis treści

Cel projektu	4
Dane wejściowe	4
Argumenty wywołania	4
Uruchomienie programu.....	5
Wynik działania	5
Scenariusz działania programu	5
Tryb podstawowy	5
Z flagą -d:.....	5
Z flagą -g:.....	5
Z flagami -g i -w:	5
Rozszerzony scenariusz działania programu	6

Cel projektu

Program generuje oraz zapisuje do pliku graf o krawędziach z określonymi wagami. Potrafi również wczytać informacje o tak wygenerowanym grafie z pliku tekstowego. Sprawdza spójność wczytanego grafu za pomocą algorytmu BFS. Znajduje najkrótszą drogę (z uwzględnieniem wag) pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi za pomocą algorytmu Dijkstry.

Argumenty wywołania

Niezbędne argumenty:

ścieżka do pliku - z tego pliku program będzie czytać dane o grafie lub je zapisywać, zależnie od wybranego trybu działania. W trybie czytania plik ten musi być sformatowany w sposób opisany poniżej.

-d - użycie tej flagi będzie wymagało podania numeru wierzchołka. Program znajdzie najkrótszą drogę z wybranego wierzchołka do wszystkich innych wierzchołków.

-p – dodatkowa flaga opcjonalna, której można użyć jedynie przy uruchomieniu algorytmu dijkstry. Program wywołany z tą flagą wypisze przebieg znalezionych ścieżek (domyślnie wypisze jedynie ich długości).

-g - użycie tej flagi sprawi, że program wygeneruje nowy graf i zapisze go do pliku podanego jako pierwszy argument wywołania. Jeśli taki plik nie istnieje, zostanie on utworzony. Po wpisaniu tej flagi należy również podać wymiary grafu jaki chcemy wygenerować.

-w - dodatkowy argument wywołania przy poleceniu -g. Pozwala na podanie przedziału wartości wag na węzłach. Użycie tej flagi wymaga podanie po niej dwóch liczb, odpowiadających końcom przedziału (włącznie).

-h – program wywołany z tym argumentem wyświetli podręczną pomoc. Takie samo działanie będzie miało również wywołanie programu bez żadnych argumentów

Uruchomienie programu

Program został przygotowany do uruchomienia w systemie Linux.

Aby uruchomić program należy, w terminalu Linuxa wpisać nazwę programu oraz argumenty wywołania:

```
./graf [ścieżka_do_pliku]
```

Przykład:

```
./graf dane/graf1 -g 4 5
```

Tak wywołany program wygeneruje graf o wymiarach 4x5 i zapisze go do pliku dane/graf1

Scenariusz działania programu

Tryb podstawowy:

Wywołanie:

```
./graf [ścieżka_do_pliku]
```

Wynik:

Sprawdzenie spójności grafu z pliku wejściowego.

Z flagą -d:

Wywołanie:

```
./graf [ścieżka_do_pliku] -d [nr_węzła_początkowego]
```

Wynik:

Znajduje wszystkie najkrótsze ścieżki pomiędzy węzłem początkowym a pozostałymi, wypisuje ich długości. Warunkiem działania są dodatnie wagi krawędzi.

Z flagami -d i -p:

Wywołanie:

```
./graf [ścieżka_do_pliku] -d [nr_węzła_początkowego] -p
```

Wynik:

Znajduje wszystkie najkrótsze ścieżki pomiędzy węzłem początkowym a pozostałymi, wypisuje przebiegi tych ścieżek oraz ich długości. Warunkiem działania są dodatnie wagi krawędzi.

Z flagą -g:

```
./graf [ścieżka_do_pliku] -g [liczba_wierszy] [liczba_kolumn]
```

Wynik:

Utworzenie nowego grafu o zadanych wymiarach i zapisanie ich do pliku źródłowego.

Z flagami -g i -w:

```
./graf [ścieżka_do_pliku] -g [liczba_wierszy] [liczba_kolumn]  
-w [początek_przedziału] [koniec_przedziału]
```

Wynik:

Utworzenie nowego grafu o zadanych wymiarach i wagach z przedziału podanego w argumentach wywołania, i zapisanie go do pliku źródłowego

Wynik działania

Wynikiem działania programu jest, w zależności od wybranego trybu, w trybie podstawowym, sprawdzenie spójności grafu lub znalezienie najkrótszych ścieżek od jednego wybranego przez użytkownika wierzchołka do wszystkich pozostałych, jeżeli uruchomimy program z flagą -d. Przy wywołaniu z flagą -d dodatkowe użycie flagi -p determinuje, czy zostaną wypisane przebiegi najkrótszych ścieżek. Uruchomienie programu z flagą -g spowoduje wygenerowanie nowego grafu o zadanych wymiarach i zapisanie go do pliku, natomiast dodatkowe użycie flagi -w pozwoli na modyfikację przedziału wartości wag w generowanym grafie.

Dane wejściowe

Program wymaga informacji o grafie, czyli informacji o połączeniach między węzłami i wagach tych połączeń. Plik wejściowy powinien zawierać $x*y+1$ linii, gdzie x jest liczbą wierszy, a y liczbą kolumn. Powinien być sformatowany następująco:

1: $x\ y$

2: $[n1\ :w1\ n2\ :w2\ n3\ :w3\ n4\ :w4]$

...

x*y: [n1 :w1 n2 :w2 n3 :w3 n4 :w4]

gdzie:

(nr_linii - 2) - numer węzła, którego dotyczą dane

n - numer sąsiadującego węzła

w - waga połączenia pomiędzy węzłami, liczba rzeczywista, domyślnie z przedziału (0, 1) lub innego, określonego w argumentach wywołania programu

(n1 :w1) to para (numer węzła :waga pomiędzy tym węzłem a węzłem źródłowym określonym numerem linii). Wartości tych par są opcjonalne, ponieważ węzeł może być połączony 0, 1, 2, 3 lub 4 węzłami sąsiadującymi.

Przykład:

1: 2 2

2: 1 :0.132 2 :0.8542

3: 0 :0.132

4: 0 :0.8542 3 :0.428

5: 2 :0.428

Rozszerzony scenariusz działania programu

Lista możliwych komunikatów kończących działanie programu

Komunikat	Opis
UNKNOWN_FLAG	Użycie niepoprawnej flagi przy wywołaniu programu
ARG_ERROR	Niepoprawna ilość lub kolejność argumentów wywołania programu
FILE_ERROR	Nie udało się otworzyć pliku do odczytu
INPUT_FORMAT_ERROR	Niepoprawny format danych w pliku wejściowym

Specyfikacja Implementacyjna

Marcin Kowalski i Jakub Orzełowski

12.03.2022

Streszczenie

Dokument stanowi szczegółowy opis projektu "Grafy". W dokumencie opisane zostały pliki źródłowe programu oraz użyte struktury danych.

Opis plików źródłowych:

gen.c i gen.h: Zawiera funkcje odpowiedzialne za generowanie grafu o zadanej wielkości i z podanymi wagami i zapis tak wygenerowanego grafu bezpośrednio do pliku.

bfs.c i bfs.h: Zawiera funkcje odpowiedzialne za działanie algorytmu przeszukiwania wszerz (BFS). Zostały tu również zawarte funkcje odpowiedzialne za prawidłowe działanie kolejki FIFO, niezbędnej do prawidłowego działania algorytmu BFS.

dijkstra.c i dijkstra.h: Zawiera funkcje odpowiedzialne za działanie algorytmu Dijkstry. Zawiera również implementację kolejki priorytetowej niezbędnej do działania algorytmu.

files.c i files.h: Odpowiada za odczyt danych z pliku oraz sprawdzenie ich poprawności. Odczytany graf zapisuje w postaci listy sąsiedztwa korzystając ze struktur zaimplementowanych w pliku list.h.

list.c i list.h: Zawiera implementację listy liniowej niezbędnej do utworzenia reprezentacji grafu w formie listy sąsiedztwa. Zawiera również funkcję odpowiedzialną za zwalnianie pamięci zarezerwowanej na przechowywanie grafu.

error.c i error.h: Moduł ułatwia obsługę błędów w programie, które mogą wystąpić w wielu z plików (main.c, files.c, dijkstra.c). Zawiera również funkcję drukującą opis poprawnego wywołania programu.

main.c: Odpowiada za działanie programu zgodnie z poleceniem użytkownika. W zależności od wybranego trybu wykorzystuje pliki: gen.c, bfs.c lub dijkstra.c. Zawsze wykorzystuje plik files.c. W tym pliku jest również zawarta obsługa większości błędów.

Opis struktury katalogów:

Wszystkie pliki projektu zostały umieszczone w trzech katalogach:

Kod: W tym katalogu zostały umieszczone wszystkie pliki źródłowe. W tym katalogu następuje również kompilacja programu i są tu zapisywane pliki zawierające informacje o wygenerowanym grafie.

Dokumentacja: W tym katalogu znajduje się pełna dokumentacja projektu, czyli specyfikacja funkcjonalna, implementacyjna oraz opis wykonanych testów.

Testy: Ten katalog zawiera pliki zawierające spreparowane dane o grafach, które zostały użyte w czasie testów programu.

Opis struktur wykorzystanych w programie:

Lista liniowa: struktura bazowa pozwalająca na implementację innych bardziej skomplikowanych struktur danych. W programie została wykorzystana przy implementacji listy sąsiedztwa i kolejki FIFO.

Kolejka FIFO: kolejka typu first in first out, wykorzystywana w algorytmie BFS

Kolejka priorytetowa: struktura, która przechowuje numery wszystkich wierzchołków. Priorytetem kolejki jest wyliczana dynamicznie odległość od wybranego wierzchołka. Do jej implementacji wykorzystany został kopiec. Jest ona niezbędnym elementem działania algorytmu Dijkstry.

Lista sąsiedztwa: tablica przechowująca listy liniowe, w których zawarte są informacje o sąsiadujących węzłach i wagach między nimi. Numerem węzła początkowego jest indeks w tablicy, numer węzła końcowego jest podany w liście.

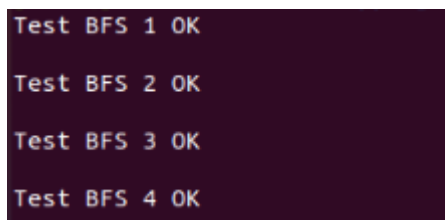
Graf: struktura przechowująca listę sąsiedztwa, liczbę kolumn i wierszy. Wczytywane są do niego dane z pliku wejściowego. Wykonuje się na nim wszystkie algorytmy.

Opis testów:

Testy zostały wykonane na specjalnie spreparowanych plikach z danymi. Został utworzony osobny plik maintest.c, który uruchamia wszystkie testy.

Testy algorytmu BFS:

Algorytm BFS wraz z funkcją czytającą dane z pliku zostały przetestowane na dwóch plikach z grafem spójnym i dwóch plikach z grafem niespójnym. Została zweryfikowana poprawność ich działania. Pierwsze dwa grafy niespójne miały kolejno wymiary: 5x6 i 14x16. Dwa kolejne grafy spójne miały wymiary kolejno: 14x16 i 213x160. Algorytm był testowany za pomocą funkcji assert. Pozytywnemu wynikowi testu towarzyszył odpowiedni komunikat.



```
Test BFS 1 OK
Test BFS 2 OK
Test BFS 3 OK
Test BFS 4 OK
```

Testy algorytmu Dijkstry:

Wykonano 5 testów których głównym zadaniem było zmierzenie czasu pracy algorytmu Dijkstry. Wykorzystano 5 grafów o rozmiarach odpowiednio:

- 10 00 węzłów z wagami z przedziału $<0,1>$
- 10 000 węzłów z wagami z przedziału $<12,43>$
- 100 000 węzłów z wagami z przedziału $<0,1>$
- 1 000 000 węzłów z wagami z przedziału $<0,1>$
- 10 000 węzłów z wagami z przedziału $<-2,3>$

W rezultacie tych testów otrzymaliśmy następujące wyniki:

```
Plik o rozmiarze: 10000, został wczytany w czasie: 0.031250  
Czas działania algorytmu Disjkstry: 0.000000, dla: 10000 węzłów  
  
Plik o rozmiarze: 10000, został wczytany w czasie: 0.031250  
Czas działania algorytmu Disjkstry: 0.000000, dla: 10000 węzłów  
  
Plik o rozmiarze: 100000, został wczytany w czasie: 0.218750  
Czas działania algorytmu Disjkstry: 0.078125, dla: 100000 węzłów  
  
Plik o rozmiarze: 1000000, został wczytany w czasie: 2.390625  
Czas działania algorytmu Disjkstry: 1.500000, dla: 1000000 węzłów  
  
INPUT_FORMAT_ERROR
```

Na pierwszy rzut oka widać, że wagi nie mają dla algorytmu żadnego znaczenia ponieważ czas wykonania dla testu 1 i 2 jest taki sam. W wyniku testu 5 otrzymaliśmy błąd formatu pliku, ponieważ algorytm Dijkstra nie działa dla wartości ujemnych.

