Języki i metody programowania
Specyfikacja implementacyjna

Mateusz Czarnecki, Paweł Jędrzejczyk

30 marca 2022

1 Informacje ogólne

Program generujący graf węzłów o zadanej liczbie kolumn i wierszy oraz wypisujący najkrótsze ścieżki pomiędzy daną parą węzłów w całości napisany jest w języku C. Program napisany został wyłącznie przy pomocy funkcji i makr należących do biblioteki standardowej. Program działa w tekstowym interfejsie graficznym. W celu uruchomienia programu sugerowane jest używanie terminala.

2 Opis modułów/pakietów

Program podzielony jest na następujące moduły:

main:
Moduł główny programu: Inicjalizuje graf, odpowiada za parsowanie parametrów wywołania. Wywołuje funkcje odpowiedzialne za wyszukanie najkrótszej drogi, sprawdzenie, czy graf jest spójny oraz funkcje odpowiedzialne za współpracę z plikami. Używając odpowiednich kodów błędów wskazuje błędy występujące przy pisaniu kodu, a także ewentualne błędy wywołane przez użytkownika.

generate:
Moduł odpowiedzialny za generowanie połączeń oraz wag krawędzi w grafie w sposób opisany w specyfikacji.

search:
Moduł zawierający implementację algorytmu Dijkstry. Odpowiada za znajdywanie najkrótszej drogi między dwoma wybranymi wierzchołkami.

bfs:
Moduł odpowiedzialny za określenie spójności grafu. Zawiera implementację algorytmu Breadth First Search oraz strukturę kolejki wraz z funkcjami ją obsługującymi

graph:
Moduł zawierający implementację ważonego grafu. Stanowi on parę zbiorów (W, K) gdzie:
W to zbiór ponumerowanych liczb naturalnych całkowitych wraz z zerem. (0,1,2...)
K to zbiór krawędzi: każda krawędź stanowi relację pomiędzy dwoma węzłami oraz posiada wagę określoną jako liczba rzeczywista z zakresu podanego przez użytkownika

inout:
Moduł odpowiedzialny za obsługę plików zawierający dane wejściowe i wyjściowe.

makefile:
Moduł Make wykorzystujemy do ułatwienia kompilacji wszystkich modułów programu. W programie przewidujemy następujące metody wywołania makefile:

make program - Kompiluje wszystkie moduły programu, generuje plik programu gotowego do uruchomienia
make test - Kompiluje moduły programu przeznaczone do przetestowania programu. Generuje plik programu zawierającego szczegółowe informacje wykorzystywane przy testowaniu programu (wypisywanie na wyjście standardowe wartości w odpowiednich miejscach)
make clean - usuwa skompilowany plik programu

3 Opis głównych funkcji i struktur

3.1 int genGraph(graph_t graph, double chance)

- moduł: generate
- argumenty: wskaźnik na strukturę przechowującą graf, szansa na brak połączenia między wierzchołkami (double)
- Opis: Funkcja generuje graf według opisu przedstawionego w punkcie czwartym.
- Funkcja zwraca 0 w przypadku poprawnego wygenerowania grafu i kod błędu w przypadku wystąpienia błędu.

3.2 int saveGraph(graph_t graph, char *outFileName)

- Moduł: inout
- Argumenty: wskaźnik na strukturę przechowującą graf, nazwa pliku(char*).
- Opis: Funkcja zapisuje graf do pliku w odpowiednim formacie i podanej nazwie.
- Funkcja zwraca 0 w przypadku poprawnego zapisu kod błędu w przypadku wystąpienia błędu

3.3 int readGraph(graph_t graph, char *inFileName)

- Moduł: inout
- Argumenty: wskaźnik na strukturę przechowującą graf, nazwa pliku (char*)
- Opis: Funkcja wczytuje graf z pliku o odpowiednim formacie i podanej nazwie.
- Funkcja zwraca 0 w przypadku poprawnego wczytania grafu i 1 w przypadku wystąpienia błędu.

3.4 int bfs(graph_t graph)

- Moduł: bfs
- Argumenty: wskaźnik na strukturę przechowującą graf
- Opis: Funkcja posługuje się algorytmem bfs i sprawdza czy graf jest spójny.
- Funkcja zwraca wartość typu (int). 0 jeżeli graf nie jest spójny i 1 jeżeli graf jest spójny.

3.5 dijkstra(graph_t graph, int startVertex, int endVertex)

- Moduł: search
- Argumenty: wskaźnik na strukturę przechowującą graf, wierzchołek początkowy (int), wierzchołek końcowy (int)
- Opis: Funkcja posługuje się algorytmem dijkstry, zapisuje w tablicy ścieżkę między dwoma wybranymi wierzchołkami, zapamiętuje długość najkrótszej drogi
- Funkcja zwraca strukturę przechowującą tablicę odwiedzonych wierzchołków oraz długość drogi między wierzchołkiem początkowym a końcowym (suma wag krawędzi)

3.6 struct graph

- Moduł: graph
- Zawiera: liczbę kolumn (int), liczbę wierszy (int), tablicę z w węzłami(double**)
- Opis: Struktura stanowi implementację grafu opisanego w punkcie 4.

3.7 struct queue

- Moduł: dfs
- Zawiera: tablicę o długości równej liczbie wierzchołków (int*), indeks początku kolejki (int), indeks końca kolejki(int)
- Opis: Struktura stanowi reprezentację kolejki poisanej w punkcie 6.1.

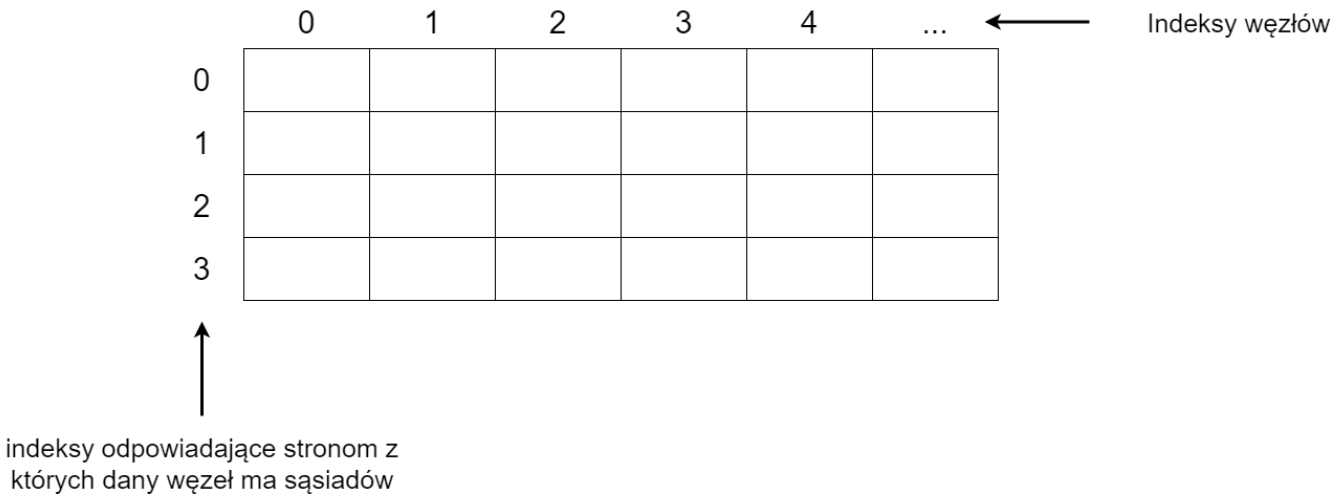
4 Sposób przechowywania i generowania grafu

Jako sposób przechowywania ważonego grafu wykorzystamy strukturę zawierającą liczbę kolumn i wierszy oraz tablicę dwuwymiarową o wymiarach n x 4, gdzie n oznacza liczbę wierzchołków, a liczba 4 stanowi maksymalną liczbę wierzchołków sąsiadujących z danym węzłem. Wagom poszczególnych krawędzi odpowiadają wartości znajdujące się w tablicy, w przypadku braku połączenia wartość wynosi 0.

Kolejne indeksy wierszy tablicy zawierają wagi połączeń węzła z odpowiednimi wierzchołkami:

[0] Wierzchołek nad węzłem
[1] Wierzchołek pod węzłem
[2] Wierzchołek po prawej stronie węzła
[3] Wierzchołek po lewej stronie węzła

Prosty schemat struktury przechowującej graf:



Program generuje graf zaczynając od wierzchołka zerowego. Losuje wagi na połączeniu do wierzchołka znajdującego się pod nim (wierzchołka o indeksie większym o ilość kolumn) oraz do wierzchołka znajdującego się po jego prawej stronie (wierzchołka o indeksie o 1 większym), uwzględniając szansę na to, że dana krawędź nie istnieje. Następnie przechodzi do kolejnego wierzchołka i powtarza całą procedurę. W przypadku ostatniej kolumny waga jest generowana jedynie na połączeniu do wierzchołka znajdującego się pod spodem, natomiast w ostatnim wierszu waga losowana jest tylko dla połączenia z wierzchołkiem po prawej stronie. Gdy zostanie osiągnięty ostatni wierzchołek, funkcja naturalnie nie generuje żadnych nowych połączeń.

5 Parametry wywołania programu

Sugerowana procedura uruchomienia programu następuje poprzez wywołanie go używając tekstowego terminala.

Akceptowane parametry wywołania programu:

-gen - Ustawia tryb generatora grafów którego parametry można przekazać za pomocą następujących flag:

-nc c (int) - Gdzie c jest całkowitą liczbą stanowiącą liczbę kolumn labiryntu

-nr r (int) - Gdzie r jest liczbą stanowiącą liczbę wierszy labiryntu

-minw k (double) - Gdzie k to liczba stanowiąca minimalną wartość wygenerowanej wagi.

-maxw n (double) - Gdzie n to liczba większa od 0, stanowiąca maksymalną wartość wygenerowanej wagi.

-s p (double) - Gdzie p to liczba w przedziale $<0; 100>$ oznaczająca prawdopodobieństwo w procentach na to, że sąsiadujące wierzchołki nie są ze sobą połączone.

-file plik.1 (char *) - Gdzie plik.1 to nazwa pliku do którego zostanie zapisany graf

-search - Ustawia tryb analizy grafu którego dane przekazujemy za pomocą następujących flag:

-start a (int) - Gdzie a jest numerem wierzchołka startowego, a musi zawierać się w zbiorze wierzchołków

-end b (int) - Gdzie b jest numerem wierzchołka końcowego, b musi zawierać się w zbiorze wierzchołków.

-in plik.2 (char *) - Gdzie plik.2 jest plikiem z danymi wejściowymi przedstawiającymi graf w odpowiednim formacie.

-out plik.3 (char *) - Gdzie plik.3 jest plikiem do którego zostaną zapisane wyniki.

-h - Powoduje wyświetlenie się instrukcji używania programu

Jako separator dziesiętny liczb **k,n,p** przyjmowana jest kropka (.)

przykładowe wywołania programu:

```
./program -gen -nc 1000 -nr 100 -minw 2.5 -maxw 234.34 -s 20 -file graf.txt
```

```
./program -search -start 3 -end 49 -in graf.1 -out wyniki.txt
```

```
./program -gen -search -nc 1000 -nr 100 -minw 2.5 -maxw 234.34 -s 20 -start 3 -end 49
-out wyniki.txt
```

6 Opis algorytmów

W ramach działania programu, wykorzystane zostaną dwa powszechnie znane algorytmy: BFS (Breadth First Search) oraz algorytm Dijkstry.

6.1 Breadth First Search

Algorytm służący do przeszukiwania grafu. Przechodzenie rozpoczyna się od danego wierzchołka i polega na odwiedzeniu wszystkich osiągalnych z niego wierzchołków, a następnie wykonanie tej samej operacji, dla kolejnych wierzchołków. Działanie algorytmu opiera się na strukturze kolejki.

Krótki schemat działania algorytmu:

- Dodajemy wierzchołek początkowy do kolejki.
- Odwiedzamy wszystkich sąsiadów wierzchołka i dodajemy je do kolejki.
- Z kolejki usuwamy wierzchołek początkowy, gdyż wszyscy jego sąsiedzi zostali odwiedzeni.
- Powtarzamy kroki dla kolejnych wierzchołków znajdujących się w kolejce, aż odwiedzimy wszystkie wierzchołki należące do grafu

W programie, algorytm BFS posłuży nam do określenia, czy wygenerowany graf jest spójny, to znaczy, czy dla każdych dwóch jego wierzchołków istnieje ścieżka, która je ze sobą łączy, a także odpowie na pytanie, czy istnieje połączenie dla dwóch wybranych przez użytkownika wierzchołków.

6.2 Algorytm Dijkstry

Algorytm służący do znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie, którego krawędzi posiadają wagi. Oryginalny algorytm szuka najkrótszych ścieżek do wszystkich pozostałych wierzchołków w grafie. W naszym przypadku, interesuje nas jedynie najkrótsza droga między dwoma wierzchołkami - startowym i końcowym.

Krótki schemat działania algorytmu:

- Wybieramy wierzchołek początkowy i ustalamy długość do niego samego jako 0 oraz długości do wszystkich innych wierzchołków jako INF (nieskończoność)
- Sprawdzamy odległości między wierzchołkiem początkowym, a nieodwiedzonymi wierzchołkami sąsiednimi. W przypadku, gdy obliczona odległość między wierzchołkiem a sąsiadem jest mniejsza od przydzielonej wcześniej odległości (inf), ustawiamy odległość na mniejszą.
- Ustawiamy poprzednio odwiedzony wierzchołek dla każdego wierzchołka, któremu zmieniliśmy odległość.
- Dodajemy wierzchołek początkowy do tablicy odwiedzonych wierzchołków.
- Odwiedzamy wierzchołek, do którego istnieje najkrótsza odległość z początkowego wierzchołka. Powtarzamy kroki od drugiego, aż nie odwiedzimy wszystkich wierzchołków w grafie.

7 Format danych

7.1 Dane wejściowe

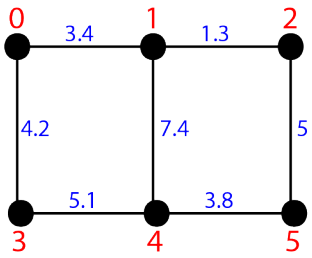
Program generuje dane wejściowe w pliku o formacie .txt. zawierające dane dotyczące rozmiaru grafu, a także krawędzi i odpowiadających im wag. W pierwszej linii danych zawarte są informacje o ilości wierszy i kolumn, a kolejne numery linii odpowiadają kolejnym wierzchołkom. Każdemu numerowi linii przyporządkowane są numery wierzchołków, z którymi tworzą krawędź oraz odpowiadające tym krawędziom wagi.

liczbaWierszy = n liczbaKolumn = m
nr_linii W1 :waga Wn :waga
nr_linii W0 :waga W2 :waga Wn+1 :waga
nr_linii W1 :waga W3 :waga Wn+2 :waga
.
.
.
nr_linii Wn*m-2 :waga W(m-2)*(n-1) :waga

na przykład:

```
2 3
1 :3.4 3 :4.2
0 :3.4 4 :7.4 2 :1.3
1 :1.3 5 :5
0 :4.2 4 :5.1
1 :7.4 3 :5.1 5 :3.8
2 :5 4 :3.8
```

Powyższe dane reprezentują poniższy graf



7.2 Dane wyjściowe

Program w zależności od wybranej opcji wypisuje do pliku lub na wyjście standardowe najkrótsze ścieżki w grafie, a także informację o tym, czy graf jest spójny.
na przykład:

```
Graf jest spójny, a długość najkrótszej drogi z wierzchołka 0 do wierzchołka 33
wynosi 8766 i przechodzi przez następujące wierzchołki:
0− > 1− > 2− > 12− > 13− > 23− > 33
```

8 Obsługa błędów

8.1 Błędy po stronie użytkownika

W przypadku braku podania żadnego z argumentów określających tryb pracy programu (-gen, -search), program wypisze stosowny komunikat i zaleci jego prawidłowe uruchomienie.

W przypadku braku argumentów odpowiadających za zakres przyjmowanych wag, program ustawi domyślne wartości:

Waga_minimalna = 1;
Waga_maksymalna = 100.

Jeżeli podane przez użytkownika argumenty wywołania nie stanowią liczby większej od zera (np. są napisem) lub Waga_minimalna będzie większa od Waga_maksymalna to program wypisze stosowny komunikat i zaleci właściwe uruchomienie.

W przypadku niepodania argumentów odpowiadających za liczbę kolumn oraz liczbę wierszy, program ustawi domyślne wartości:

Liczba_kolumn = 100;
Liczba_wierszy = 100.

W przypadku niepodania argumentów odpowiadających za prawdopodobieństwo na to że między sąsiadującymi wierzchołkami nie ma połączenia zostanie ustawiona domyślna wartość:

Szansa_Na_Brak_Połączenia = 10

Jeżeli użytkownik nie poda nazwy pliku do którego ma zostać zapisany graf, to zostanie on zapisany w pliku **graf_dane**

Jeżeli użytkownik nie poda nazwy pliku z którego ma zostać wczytany graf, to program przeczyta graf z pliku o nazwie **graf_dane**

Jeżeli użytkownik nie poda pliku do którego mają zostać zapisane dane wyjściowe, to program wypisze je na wyjście standardowe.

W przypadku podania tylko jednego z wierzchołków między którymi ma być wyszukana droga program wypisze stosowny komunikat i zaleci właściwe uruchomienie, gdy natomiast żaden z wierzchołków nie zostanie podany, program pominie wyszukiwanie najkrótszej drogi i wypisze jedynie informację dotyczącą spójności grafu.

8.2 Kody błędów

Podczas pracy z programem, w zależności od napotkanego błędu, obowiązywać będą następujące kody błędów (wartości zwracane przez funkcje):

0 - Prawidłowe działanie

1 - Nieudana alokacja pamięci

2 - Nieprawidłowy typ danych

3 - Nieudany odczyt z pliku

4 - Nieudany zapis do pliku

5 - Niedozwolona operacja matematyczna

6 - Nieprawidłowe argumenty wywołania

7 - Nieprawidłowa nazwa pliku

8 - Nieprawidłowa wartość (np. zła wartość wagi/numeru wierzchołka)

99 - Inny błąd