Specyfikacja funkcjonalna programu ***grafexe***

v2.0

Konrad Skarżyński

Krystian Sereda

Cel projektu

Program *grafexe* wyznacza pomiędzy określonymi parami węzłów najkrótszą ścieżkę wykorzystując algorytm Dijkstry. Program działa w trybie wsadowym i umożliwia generowanie lub wczytanie grafu, sprawdzanie spójności grafu, poprzez wykorzystanie algorytmu BFS, oraz wyznaczanie najkrótszej ścieżki pomiędzy dowolną liczbą par węzłów.

Scenariusze działania

1. Generowanie grafu;
2. Wczytywanie grafu;
3. Sprawdzanie spójności grafu
4. Znajdowanie najkrótszej ścieżki między węzłami;
5. Wyświetlenie informacji o działaniu programu;

Użytkownik może łączyć przedstawione scenariusze.

Dane wejściowe

Do poprawnego działania programu niezbędne będą dane dotyczące wykorzystywanego przez program grafu. W zależności od sposobu dostarczenia tej informacji podawane będą odmienne dane wejściowe.

- Nazwa generowanego przez program grafu lub istniejącego, użytkownik nadaje ją jako jednoczłonowy wyraz, w którym występować mogą symbole z alfabetu łacińskiego, cyfry oraz symbole specjalne. Długość nazwy zawiera się w 64 znakach. Odczytywany plik z –grafem powinien zawierać w kolejnych liniach, które odpowiadają numerom węzłów (pierwsza linia w pliku odpowiada węzłowi numer zero) oddzielone znakiem białym numer węzła, z którym następuje połączenie ścieżką z aktualnego węzła oraz waga tej ścieżki, w przedstawionej kolejności. W jednej linii znajdują się odpowiednio wprowadzone wszystkie ścieżki poprowadzone z aktualnego węzła.

- Nazwa podawanego przez użytkownika pliku z danymi do określania najkrótszych ścieżek nadawana jako jednoczłonowy wyraz, w którym występować mogą symbole z alfabetu łacińskiego, cyfry oraz symbole specjalne. Długość nazwy zawiera się w 64 znakach. . Każda

następna linia powinna składać się z trzech dwóch liczb określających kolejno pierwszy i drugi węzeł z przedziału <0,w*k-1> oraz wartość przejścia.

- Liczba węzłów (w): podawana jako liczba całkowita dodatnia, większa od jeden liczba węzłów jaką będzie miał generowany graf.

- Przedział, z którego losowane będą wartości wag przejść podawane poprzez liczbę będącą początkiem (f) oraz końcem (t) tego przedziału.

- Liczba (n) tworzonych najkrótszych ścieżek między parami węzłów określonych kolejno w załączonym pliku. Najkrótszą drogą określamy przejście między węzłami, których suma wag pojedynczych przejść jest najmniejsza.(n- liczba całkowita dodatnia)

Argumenty

Program ***grafexe*** przyjmuje następujące argumenty:

- **`-g graf`** powoduje wygenerowanie grafu do pliku o określonej nazwie.
- **`-p x1`** określa liczbę kolumn generowanego grafu.
- **`-q x2`** określa liczbę wierszy generowanego grafu.
- **`-f k`** określa początek przedziału, z którego losowane są wagi ścieżek.
- **`-t l`** określa koniec przedziału, z którego losowane są wagi ścieżek.
- **`-r plikr`** powoduje wczytanie grafu znajdującego się w określonym przez użytkownika.
- **`--bfs`** powoduje sprawdzenie spójności grafu oraz wypisanie adekwatnego komunikatu.
- **`--file plikf`** powoduje wczytanie par węzłów z pliku określonego przez użytkownika.
- **`-h|--help`** powoduje wyświetlenie się pomocy dotyczącej uruchamiania programu: tj. objaśnienie użycia poszczególnych flag.
- **`-n liczba_sciezek`** określa ilość tworzonych ścieżek między odpowiednimi węzłami wczytywanymi z pliku.

Użycie

Wzorcowe wywołanie programu zawierające relacje pomiędzy poszczególnymi flagami programu:

```
$: /grafexe [ -h|--help ] | [ [-g nazwa_pliku  -p <x1> -q <x2> -f <k> -t  <l> [--bfs] [-n liczba_sciezek --file plikf]] | [-r nazwa_pliku [ --bfs ] [-n liczba_sciezek --file plikf]  ] ]
```

Przykładowy wygląd pliku z grafem:

2 3

1 :1.1235320000000000 2 :4.1231239999999998
0 :0.9234000000000000
0 :1.9900000000000000 3 :1.1235550000000001 4 :3.1230000000000002
2 :1.9921888000000001
2 :1.1110000000000000 5 :1.1234567899876540
4 :1.1245551100000000

Przykładowy wygląd pliku z parami węzłów, między którymi ma zostać obliczona najkrótsza ścieżka:

0 3
1 9
8 2
5 7
0 12
3 14

Przykładowe wywołania programu

```
./grafexe -g grafjeden -n 2 -w 25 --bfs --file punkty
```

Efektem będzie stworzenie przez program grafu o 25 wierzchołkach oraz zapisanie go do pliku o nazwie “grafjeden”. Następnie zostanie sprawdzona spójność grafu oraz wyznaczona najkrótsza ścieżka pomiędzy pierwszymi dwoma parami węzłów określonych w pliku “punkty”.

```
./grafexe -r plik.txt --bfs --file pary -n 15
```

Efektem będzie odczytanie z pliku o nazwie “plik” grafu, sprawdzenie jego spójności oraz wyznaczenie najkrótszej ścieżki pomiędzy pierwszymi 15 parami węzłów określonych w pliku “pary”.

Teoria

Algorytm Dijkstry - algorytm służący do wyznaczania najkrótszych ścieżek w grafie. Wyznacza on najkrótsze ścieżki z jednego wierzchołka (wierzchołka źródłowego) do pozostałych wierzchołków, przy założeniu, że wagi krawędzi grafu nie są ujemne.

Zasada działania: Tworzymy dwa zbiory wierzchołków Q i S. Zainicjowany zbiór Q zawiera wszystkie wierzchołki grafu, a zbiór S jest pusty. Dla wszystkich wierzchołków u grafu za wyjątkiem startowego v ustawiamy koszt dojścia d(u) na nieskończoność. Koszt dojścia d(v) zerujemy. Dodatkowo ustawiamy poprzednik p(u) każdego wierzchołka u grafu na niezdefiniowany (liczba ujemna). Poprzednie wierzchołki będą wyznaczały w kierunku odwrotnym najkrótsze ścieżki od wierzchołków u do wierzchołka startowego v. Dopóki zbiór Q zawiera wierzchołki, wykonujemy następujące czynności:

- Wybieramy ze zbioru Q wierzchołek u o najmniejszym koszcie dojścia d(u).
- Wybrany wierzchołek u przenosimy ze zbioru Q do zbioru S.
- Dla każdego sąsiada w wierzchołka u, który jest wciąż w zbiorze Q, sprawdzamy, czy
- d(w) > d(u) + waga krawędzi u–w.
- Jeśli tak, to należy wyznaczyć nowy koszt dojścia do wierzchołka w jako:
- d(w) ← d(u) + waga krawędzi u–w.
- Następnie wierzchołek u czynimy poprzednikiem w:
- p(w) ← u.

BFS - algorytm przeszukiwania grafów wszerz, wykorzystujący kolejkę FIFO (First in first out). Przechodzenie grafu rozpoczyna się od wybranego wierzchołka *s* i polega na sprawdzeniu wszystkich osiągalnych z niego wierzchołków. Wynikiem działania algorytmu jest drzewo przeszukiwania wszerz o korzeniu w *s*, zawierające wszystkie wierzchołki osiągalne z *s*. Do każdego z tych wierzchołków prowadzi dokładnie jedna ścieżka z *s*, która jest jednocześnie najkrótszą ścieżką w grafie wejściowym.

Źródła:

- https://pl.wikipedia.org/wiki/Przeszukiwanie_wszerz
- https://eduinf.waw.pl/inf/alg/001_search/0126.php
- https://eduinf.waw.pl/inf/alg/001_search/0138.php

Komunikaty oraz błędy

Wystąpienie większości błędów wiąże się z przerwaniem działania programu. Przedstawione komunikaty to schematy komunikatów, które będą wyświetlane, a w miejsce pogrubionych wyrażeń wstawiane będą odpowiednie zmienne.

- Podanie argumentu bez wcześniejszego podania flagi:
  "*Nie określono rodzaju argumentu **dana***."

- Podanie liczby węzłów w postaci liczby większej od jeden i niecałkowitej lub w postaci liczby nie większej niż jeden:
  "*Wprowadzono niepoprawną liczbę **węzłów***." err2

- Podanie liczby określającej początek przedziału, z którego losowane są wagi przejść w grafie większej niż liczby określającej koniec tego przedziału:
  "*Podany przedział <**k;l**> nie jest poprawny.* " err3

- Podanie złej nazwy flagi:
  " *'**-flaga**' - Nieznana flaga.*" err4

  *Wyświetlona zostanie pomoc dotycząca działania programu, taka sama jak w przypadku podania flagi "-h" lub "--help".

- Podanie nazwy pliku nieistniejącego lub podanie niepoprawnej nazwy przy flagach odczytujących dane z plików:
  "*Nie znaleziono pliku o podanej nazwie.*"

- Nie podanie argumentu po fladze:
  "*Nie wprowadzono argumentu dla flagi '**-flaga**'.*"

- Brak jakichkolwiek danych wejściowych:

  *Wyświetlona zostanie pomoc dotycząca działania programu, taka sama jak w przypadku podania flagi "-h" lub "--help".

- Wprowadzenie ilości kolumn lub wierszy bez podania flagi "-g":
  "*Nie wprowadzono flagi '-g', dlatego program nie utworzy grafu o określonych parametrach." err5*

- Zainicjowanie algorytmu BFS używając flagi "--bfs", bez wcześniejszego podania wczytywanego pliku wraz z flagą "-r" lub bez generowania grafu:
  "*Nie podano grafu, którego spójność miałaby zostać określona".*

- Nieodpowiednio zapisany graf w odczytywanym pliku:
  "*Graf zapisany w pliku '**plik.txt**' przyjmuje nieodpowiedni format.*" err6

- Niemożliwe jest połączenie jakąkolwiek ścieżką określonej przez użytkownika pary węzłów:
  "*Niemożliwe jest znalezienie ścieżki pomiędzy danymi wierzchołkami: **węzeł1** i **węzeł2**.*"

- Nieodpowiednio zapisane pary wierzchołków w podanym pliku, pomiędzy którymi mają zostać wyznaczone najkrótsze ścieżki:
  "*Pary wierzchołków zapisane w pliku '**plik.txt**' przyjmują nieodpowiedni format.*"

- Wprowadzenie nieodpowiedniej liczby (mniejszej niż zero) określającej jeden z końców przedziału, z którego losowane zostaną wagi:
  "*Wprowadzono niewłaściwą liczbę wyznaczającą przedział, z którego losowane będą wagi.*" err8

- Nieodpowiednia (nienaturalna) liczba określająca ilość wyszukanych ścieżek:
  "*Wprowadzono nieodpowiednią liczbę określającą ilość par, między którymi wyznaczone będą ścieżki.*" err7

- Liczba określająca ilość tworzonych ścieżek jest naturalna i większa niż ilość par określających początek i koniec danej drogi.
  *"Wprowadzono za mało danych (par węzłów) do pliku '**nazwa_pliku**'."*

- Nie podano przedziału wagowego dla generowanego grafu.
  "Brak przedzialu wagowego dla generowanego grafu". err8
- Inne komunikaty związane z problemami w poszczególnych plikach np. braku możliwości zaalokowania pamięci dla grafu lub nieodpowiedniego formatu pliku.

Specyfikacja implementacyjna programu ***grafexe***

Podział i opis plików

- main.c
  Odpowiada za uporządkowanie działania podprogramów znajdujących się w pozostałych plikach, sprawdzenie wystąpienia ewentualnych błędów oraz odpowiednie odczytanie argumentów i flag.

- komunikaty.c (komunikaty.h)
  Przechowuje i pozwala wyświetlić odpowiednie komunikaty błędów.

- bfs.c (bfs.h)
  Pozwala sprawdzić spójność określonego grafu wykorzystując kolejkę

- dij.c (dij.h)
  Pozwala odnaleźć najkrótszą ścieżkę między określonymi parami węzłów.

- generator.c (generator.h)
  Pozwala wygenerować graf o określonej wielkości i wygenerować wagi ścieżek na podstawie podanego przez użytkownika przedziału.

  Jeżeli użytkownik zdecyduje się na wygenerowanie grafu, to zostanie on wygenerowany w macierzy sąsiedztwa, a następnie zapisany w postaci listy znajdującej się w podanym przez użytkownika pliku.

- czytacz.c (czytacz.h)/czytacztemp.c
  Pozwala na wczytanie grafu oraz pliku zawierającego pary węzłów pomiędzy, którymi obliczone zostaną najkrótsze ścieżki i zapisanie ich do odpowiednich tablic.
  Odpowiada za zaalokowanie pamięci dla określonej wielkości grafu i wypełnienie go domyślnie liczbami “-1”.

  W pliku czytacz.h znajduje się struktura odpowiedzialna za przechowywanie grafu:

  ```
  typedef struct e{
        double* graph;
        int columns;
        int rows;
  } *graph_t;
  ```

  Plik czytacztemp.c wykorzystywany przy obsłudze (testowaniu) większych grafów.

- pqmin.c (pqmin.h)

Odpowiada za obsługę kolejki priorytetowej.
W pliku pqmin.h znajdują się struktury odpowiedzialne za kolejkę priorytetową wykorzystywaną w algorytmie dijkstry:

```
typedef struct {
	int vert; // numer indeksu
	double dist; // dystans
} * v_t;


typedef struct {
	v_t *q;  // the queue of v_t
	int n;	// actual # of doubles in it
	int s;	// it's size
} * pq_t;
```

- queue.c (queue.h)

Odpowiada za interakcje z kolejką.
W pliku queue.h znajduje się struktura odpowiedzialna za kolejkę, która następnie wykorzystywana jest w algorytmie BFS, czy Dijkstry:

```
typedef struct q{
	int* que;
	int front;
	int rear;
	int HowMany;
	int SIZE;
} *ptr_t;
```

- bfs_test.c; test_dij.c; test_czytacz.c; test_que.c; test pqmin.c

W tych plikach znajdują się programy testujące poszczególne fragmenty programu. Wykonanie testów znajdujących się w tych plikach opisane zostało w rozdziale “Testy”.

- makefile

Struktury danych

Graf przechowywany w postaci macierzy sąsiedztwa (tablicy jednowymiarowej) zawierającej informacje o możliwych przejściach między parami węzłów oraz wagach tych przejść.
Generator generuje graf w postaci macierzy sąsiedztwa, a następnie zapisuje ten graf w postaci listy w docelowym pliku.
Wykorzystane są struktury przedstawione już przy okazji opisu poszczególnych plików.
Istotnym elementem całego programu jest wykorzystanie tablic, które są podstawowym elementem przechowywania danych w ***grafexe***.

Testy

Testy przeprowadzane są po osiągnięciu kolejnych istotnych elementów implementacyjnych, jak testy kolejki. Wyniki testów zapisane są w pliku dane oraz, w utworzonym po przeprowadzeniu testu, pliku "log". Testy algorytmu BFS polegały na sprawdzeniu spójności grafów specjalnie tworzonych oraz generowanych przez generator. Część wyników testów samego generatora (w postaci macierzy sąsiedztwa) oraz generatora w połączeniu z plikiem main znajduje się w folderze dane/Testy generatora . Można przeprowadzić przykładowe testy poszczególnych podprogramów poprzez użycie komend "make test"; "make testgen"; "make test2" lub "make testgen2".