Specyfikacja implementacyjna projektu w języku C

Kamil Fryszkowski, Oskar Biwejnis

16 marca 2022

1 Wstęp

Celem projektu jest stworzenie programu w języku C, który będzie w stanie znajdować najkrótszą ścieżkę w pomiędzy dwoma wierzchołkami w danym grafie ważonym skierowanym. Użyte zostaną do tego algorytmy BFS oraz Dijkstry. Środowiskiem programistycznym będzie Linux, kod będzie pisany przy użyciu Vim, a wykorzystanym sposobem współpracy i wersjonowania - GIT. Używane biblioteki języka C:

- `stdio.h` - odpowiedzialna za podstawowe komendy wejścia i wyjścia
- `stdlib.h` - odpowiedzialna za komendy takie jak `rand`, `malloc` i inne
- `time.h` - odpowiedzialna za pobranie ziarna do generatora liczb losowych

2 Struktury danych

Do poprawnego funkcjonowania programu niezbędne będzie zaimplementowanie następujących struktur danych:

1. Do przechowywania grafu użyta zostanie lista sąsiedztwa `graph_t`, która będzie dwuwymiarową tablicą par liczb `<wierzcholek>`, `<waga>`. Informacją o tym, z którego wierzchołka wychodzi dane połączenie będzie indeks X tablicy. Wymiar X tablicy będzie zdefiniowany poprzez ilość wierzchołków równą wiersze * kolumny. Wymiar Y tablicy będzie równy 4, ponieważ taka jest maksymalna liczba krawędzi w grafie przypominającym kartkę w kratkę. Niewykorzystane pola będą zawierały wartość `null`.
   Tekstowa reprezentacja przechowywania grafu:

X\Y	**0**	**1**	**2**	**3**
0	1, 0.5..	4, 0.7..	null	null
1	2, 0.5..	5, 0.6...	null	null
2	3, 0.3..	1, 0.1..	6, 0.4..	null

2. Kolejka FIFO użyta zostanie przy działaniu algorytmu BFS. W kolejce tej, pierwszy wprowadzony element będzie pierwszym wyjętym. Do jej implementacji wykorzystana zostanie tablica statyczna, ponieważ znana jest jej długość - równa liczbie wierzchołków. Utworzone będą dwa wskaźniki p i k, wskazujące kolejno początek i koniec kolejki. Przy dokładaniu elementów inkrementowany będzie wskaźnik k, a przy wyjmowaniu wskaźnik p. Możliwe operacje na kolejce:

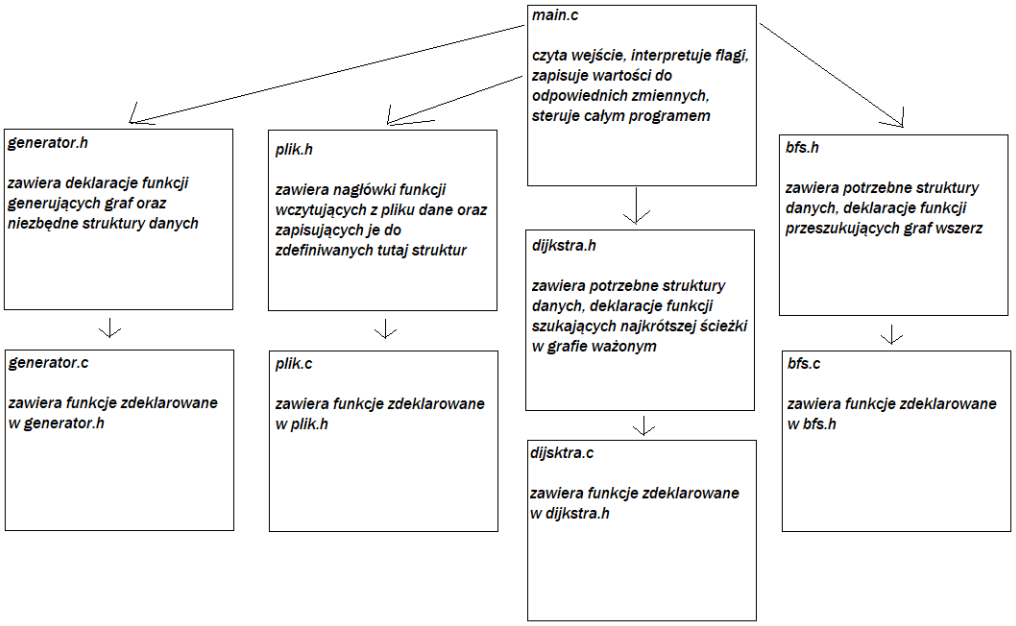
3. Kolejka priorytetowa niezbędna będzie do poprawenego działania algorytmu Dijkstry. Priorytem według które będą wyjmowane elementy z kolejki będzie wartość `d[v]`, mówiąca o długości najkrótszej ścieżki do danego wierzchołka z wierzchołka startowego w danym momencie działania algorytmu. W celu szybszego wyjmowania elementów z kolejki, zostanie ona zaimplementowana na strukturze kopca.

3 Tryby pracy programu

1. Tryb `loadMode` służy do pracy z gotowym plikiem, na podstawie którego zostanie utworzony graf oraz ruchomiony zostanie algorytm Dijkstry w celu znalezienia najkrótszej ścieżki.

2. Tryb `allRandMode` wygeneruje graf z losowymi krawędziami pomiędzy wierzchołkami oraz losowymi wagami, następnie znajdzie najkrótszą ścieżkę algorytmem Dijkstry.
3. Tryb `conMode` wygeneruje graf podobnie do `allRandMode`, lecz sprawdzi za pomocą algorytmu BFS spójność grafu i w przypadku braku spójności będzie go generował dopóki taki nie będzie. W następnym kroku znajdzie najkrótszą ścieżkę algorytmem Dijkstry.
4. Tryb `randWeightMode` wygeneruje graf ze wszystkimi krawędziami i wylosuje ich wagi, następnie znajdzie najkrótszą ścieżkę algorytmem Dijkstry.

4 Modularna struktura programu



Rysunek 1: Graficzna reprezentacja struktury programu (opracowanie własne)

- main.c zawiera:
  - `int readArguments(int argc, char** argv)` - funkcję odpowiedzialną za rozpoznanie podanych argumentów, zwracającą 0 przy poprawnym działaniu i 1 przy błędnym działaniu.
  - `int main(int argc, char**argv)` - funkcję zapewniającą obsługę wszystkich pozostałych funkcji.
  - `typedef struct graph_t` - strukturę, przechowującą graf
- plik.h zawiera funkcje:
  - `int readFile(char* in)` - odpowiedzialną za czytanie danych wejściowych, zwracającą 0 przy poprawnym działaniu i 1 przy błędnym działaniu.
  - `int writeFile(char* out)` - odpowiedzialną za wypisanie danych wejściowych, zwracającą 0 przy poprawnym działaniu i 1 przy błędnym działaniu
  - `int writeResults(char* out)` - odpowiedzialną za wypisanie wyników, zwracającą 0 przy poprawnym działaniu i 1 przy błędnym działaniu
- generator.h zawiera funkcje:
  - `int generateWeigh(double low, double high)` - odpowiedzialną za generowanie wag dla krawędzi w przedziale od low do high
  - `int generateConnetions(int a)` - odpowiedzialną za generowanie krawędzi w zależności od wartości `a`, gdzie `a` oznacza tryb pracy generatora

- bfs.h zawiera funkcje:
  - `int bfs(graph_t* graph)` - odpowiedzialną za przeszukiwanie grafu wszerz
- dijkstra.h zawiera funkcje:
  - `int dijkstra(graph_t* graph)` - odpowiedzialną za znajdowanie najkrótszej ścieżki w grafie

5 Algorytm BFS

Działanie algorytmu BFS, czyli algorytmu przeszukiwania grafu wszerz, można streścić następująco:

Wybieramy wierzchołek startowy. Następnie odwiedzamy po kolei wszystkie wierzchołki sąsiadujące z wierzchołkiem startowym. Następnie odwiedzamy po kolei wszystkie nieodwiedzone wierzchołki sąsiadujące z wierzchołkami sąsiadującymi z wierzchołkiem startowym i tak dalej...
Niezbędna do funkcjonowania BFSa jest kolejka FIFO (odnośnik do pkt 2.) oraz tablica która dla każdego wierzchołka będzie posiadała informację o jego stanie:

0 – nieodwiedzony (kolor biały)
1 – dodany do kolejki lecz nieodwiedzony (kolor szary)
2 - odwiedzony (kolor czarny)

Złożoność pamięciowa
Dla wykorzystanej w tym programie listy sąsiedztwa złożoność pamięciowa opisana jest wzorem O(V+E) gdzie: V - liczba wierzchołków, E - liczba krawędzi.

Złożoność czasowa
Pesymistyczny przypadek to taki, gdy algorytm musi odwiedzić wszystkie wierzchołki oraz krawędzie, wtedy złożoność czasowa wynosi O(V+E), gdzie V - liczba wierzchołków, E - liczba krawędzi.

6 Algorytm Dijkstry

Algorytm dijkstry służy do znajdowania **najkrótszej ścieżki** w grafie ważonym. Przez **ścieżkę** pomiędzy dwoma wierzchołkami rozumiemy kolejno uporządkowane wierzchołki, które należy odwiedzić aby dostać się z wierzchołka startowego do końcowego. Z kolei **najkrótszą** ścieżką jest ta, której suma wag krawędzi jest najniższa. Algorytm Dijkstry, podczas działania, znajduje najkrótsze ścieżki do każdego wierzchołka do którego można dojść z wierzchołka startowego

Niezbędne do poprawnego działania będzie utworzenie:

- Tablicy d[V], gdzie V oznacza liczbę wszystkich wierchołków w grafie. Przechowywać będzie ona informację o długości aktualnej najkrótszej ścieżki z wierzchołka startowego do danego wierzchołka. Dla wierzchołka startowego s, d[s] przyjmuje początkowo wartość 0, a dla każdego innego nieskończoność.
- Tablicy p[V], gdzie V oznacza liczbę wszystkich wierchołków w grafie. Przechowywać będzie ona informację o poprzednim wierzchołku w najkrótszej ścieżce do danego wierzchołka. Początkowo jest pusta.
- Kolejki priorytetowej opartej na strukturze kopca. W kolejce tej początkowo będą wszystkie wierzchołki, a priorytetem wyciągania elementów z niej będzie najniższa wartość d[v] dla danego wierzchołka v.

Algorytm Dijkstry można przedstawić następująco: (źródło: pl.wikipedia.org)

```
Dijkstra(G,w,s):
   dla każdego wierzchołka v w V[G] wykonaj
      d[v] := nieskończoność
      poprzednik[v] := niezdefiniowane
   d[s] := 0
   Q := V
   dopóki Q niepuste wykonaj
      u := Zdejmij_Min(Q)
      dla każdego wierzchołka v – sąsiada u wykonaj
         jeżeli d[v] > d[u] + w(u, v) to
            d[v] := d[u] + w(u, v)
            poprzednik[v] := u

   Wyświetl("Droga wynosi: " + d[v])
```

Rysunek 2: Algorytm Dijkstry (źródło: pl.wikipedia.org)

7 Testowanie

Poprawność działania programu należy sprawdzić kolejno testami, oraz wyniki porównać z wcześniej przygotowanymi danymi:

- `void testAllRandModeGenerator(int rows, int cols, double minValue, double maxValue, int dec);` - funkcja sprawdzająca poprawność działania generator w trybie AllRandMode
- `void testRandWeightModeGenerator(int rows, int cols, double minValue, double maxValue, int dec);` - funkcja sprawdzająca poprawność działania generator w trybie RandWeightMode
- `void testConModeGenerator(int rows, int cols, double minValue, double maxValue, int dec);` - funkcja sprawdzająca poprawność działania generator w trybie ConMode
- `int testBFS(int start, int end);` - funkcja sprawdzająca poprawność działania algorytmu BFS
- `double testDijkstra(int start, int end);` - funkcja sprawdzająca poprawność działania algorytmu Dijkstry

8 Źródła

1. https://pl.wikipedia.org