Sprawozdanie
programu grapher

Szymon Półtorak i Sebastian Sikorski

14.04.2022r

Streszczenie

Niniejszy dokument jest sprawozdaniem z prac projektowych w ramach projektu *grapher* w języku C. W dokumencie został przypomniany cel projektu, opisana struktura folderów, diagram modułów, przedstawione wywołania programu wraz z ich wynikami. Podsumowaliśmy projekt oraz współpracę i wysnuliśmy wnioski na temat tego przedsięwzięcia.

Spis treści

1 Cel projektu - streszczenie

Program *grapher* ma za zadanie generować pliki z grafami, typu *karta w kratkę*, o z góry ustalonym formacie oraz czytanie takich plików w celu znalezienia najkrótszej ścieżki między zadanymi przez użytkownika punktami. Grapher posiada cztery tryby:

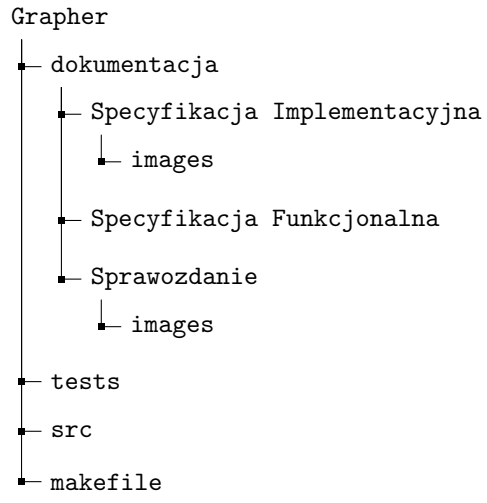
- Wage Mode – program generuje graf spójny o losowych wagach krawędzi,
- Edge Mode – program losuje istnienie krawędzi między wierzchołkami grafu oraz wagi do momentu powstania grafu spójnego,
- Random Mode – program losuje istnienie krawędzi i ich wagi. W tym trybie graf może być niespójny,
- Read Mode – program czyta plik o ustalonym formacie i szuka najkrótszej ścieżki pomiędzy podanymi przez użytkownika punktami.

Po szczegółowe wyjaśnienie funkcjonalności trybów, formatu pliku z grafem oraz znaczenia poszczególnych elementów składni programu odsyłamy do specyfikacji funkcjonalnej projektu *grapher*.

2 Struktura programu

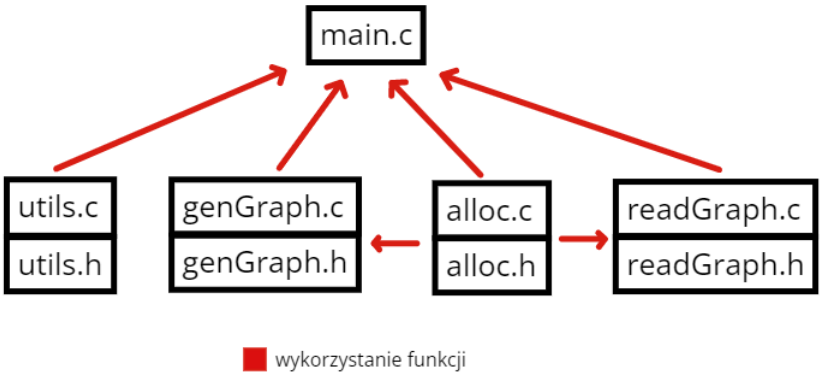
Program *grapher* składa się z 4 folderów nadrzędnych zawierających jego poszczególne elementu. Folder *dokumentacja* zawiera dokumenty opisujące projekt, czyli: specyfikację funkcjonalną i implementacyjną oraz końcowe sprawozdania z projektu. Są w nich pliki **.pdf*, zdjęcia w formacie **.png* i **.jpg* oraz kod źródłowy tych dokumnetów w formacie **.tex*. Folder *tests* zawiera kod odpowiedziany za przeprowadzanie testów programu, natomiast folder *src* zawiera pliki z kodem źródłowym oraz pliki nagłówkowe programu *grapher*.

2.1 Struktura folderów



2.2 Diagram modułów

Projekt *grapher* składa się z modułów: *alloc*, *readGraph*, *genGraph* oraz *utils*. Każdy moduł składa się z pliku nagłówkowego **.h* oraz pliku z kodem źródłowym **.c*. Posiada on również główny moduł *main* sterujący działaniem całego programu i składa się on tylko z pliku źródłowego *main.c*.



Rysunek 1: Diagram modułów

3 Kompilacja programu

Program trzeba najpierw skompilować w katalogu głównym projektu. Poniżej przedstawiamy wszystkie komendy możliwe do użycia:

- `make` - podstawowa kompilacja programu grapher,
- `make clean` - usuwa z programu wszystkie pliki robocze oraz skompilowany plik do uruchamiania programu *grapher*,
- `make wm` - kompiluje program i uruchamia go w trybie *wage mode* z góry zakładanymi danymi,
- `make rem` - robi to samo co powyższa komenda ale uruchamia program w trybie *random mode*,
- `make em` - wykonuje to samo co powyższe 2 instrukcje ale uruchamia program w trybie *edge mode*,
- `make rm` - również wykonuje to samo zadanie ale program korzysta z trybu *read mode*.

4 Przykładowe wywołania i wyniki programu

W tym rozdziale przedstawimy wywołania programu wraz z ich wynikami dla różnych scenariuszów aby ukazać jak nasz program działa.

4.1 Wage Mode

Plik wejściowy - w trybach generujących jest to plik pusty.
Wywołanie:

```
./grapher -wm -rows 4 -start 1 -file wg.test -end 10 -columns 5
```

Wynik:

4.2 Edge Mode

Plik wejściowy - plik pusty.
Wywołanie:

```
./grapher -em -rows 5 -file em.test -end 20 -columns 7 -start 5
```

Wynik:

4.3 Random Mode

Plik wejściowy - plik pusty.
Wywołanie:

```
./grapher -file rem.test -rem -end 10 -rows 6 -start 1 -columns 7
```

Wynik:

4.4 Read Mode z flagą Standard

Plik wejściowy :
Wywołanie:

```
./grapher -file rm_s.test -rm -points 1,5,4,8 -standard
```

Wynik:

4.5 Read Mode z flagą Extended

Plik wejściowy :
Wywołanie:

```
./grapher -extended -points 2,7,3,11 -file rm_e.test -rm
```

Wynik:

5 Zmiany względem specyfikacji

W nieniejszym rozdziale opisujemy zmiany jakie zaszły między specyfikacją funkcjonalną i implementacyjną, a wersją finalną programu.

5.1 Diagram modułów

Z powodu potrzeby dodania nowego modułu zmienił się również diagram modułów. Poprawną wersję prezentowaliśmy już wyżej. Doszedł moduł *utils* wspomagający pracę *maina* w zakresie obsługi błędów.

5.2 Obsługiwane błędy

W trakcie pisania programu napotkaliśmy na sytuacje, które wymagają zdefiniowania nowych błędów żeby użytkownik wiedział, dlaczego program się wyłączył. Niestety okazało się również, że nasze kody błędów były zbyt duże i program nie mógł zwracać takich wartości dlatego musieliśmy podjąć decyzję o ich zmianie.
Poniższa tabela zawiera wszystkie zadeklarowane błędy w programie:

Nazwa Błędu	Kod	Wyjaśnienie błędu
NO_MODE_FOUND	226	Niepoprawny tryb lub jego brak
NO_FILE_FOUND	231	Nie podano pliku lub plik nie istnieje
WRONG_NUM_OF_ROWS	232	Podano niepoprawną liczbę wierszy
WRONG_NUM_OF_COL	233	Podano niepoprawną liczbę kolumn
WRONG_RANGE_OF_WAGES	234	Zły zakres losowania wartości wag
NO_FLAG_FOUND	235	Nie podany flagi w trybie Read Mode
WRONG_POINTS	228	Podano nieistniejący punkt lub ich złą liczbę
NO_COHERENT	237	Graf jest niespójny
NULL_POINTER_EXCEPTION	228	Alokacja pamięci się nie udała
NOT_READ_MODE	229	Użyto flagi w trybie do generacji, ale działającej tylko w Read Mode
MULTIPILE_MODE_DECLARATION	230	Dokonano próby nadpisania zadeklarowanego wcześniej trybu programu
WRONG_MODE	227	Użyto flagi w trybie Read, ale działającej tylko w trybach generujących
INVALID_DATA	225	Nie podano wymagano argumentu lub podano flagę, która nie istnieje

5.3 Zmiany w strukturach

Struktury również przeszły małe modyfikacje spowodowane nieprzewidzianymi potrzebami. Zaprezentujemy je poniżej.

- `struct entryRead` - ta struktura otrzymała nową zmienną *numberPoints* odpowiedzialną za przetrzymywanie liczby wszystkich punktów podanych przez użytkownika,

```
typedef struct entryRead {
    char* fileName;
    bool printFlag;
    int* points;
    int numberPoints;
} entryR;
```

- `struct graphRead` - teraz *graph* jest typu `node*`,

```
typedef struct graphRead {
    node* graph;
    int rows;
    int columns;
} graphR;
```

- `struct node` - ta struktura otrzymała nową zmienną tablicową *nodeToConnect* oraz wszystkie tablice zostały zmienione ze wskaźników na tablice o określonym rozmiarze.

```
typedef struct node {
    bool edgeExist[4];
    double edgeWeight[4];
    int nodeToConnect[4];
} node;
```

5.4 Wywoływanie programu

Zmianom uległo samo wywołanie programu. Poprzednio zakładaliśmy, że użytkownik będzie musiał przestrzegać kolejności wywołania, ale w czasie pisania programu stwierdziliśmy, że jest to zadanie bezsensowne i teraz użytkownik może wprowadzać przy pomocy odpowiednich flag w dowolnej kolejności. Teraz flagi wymagają od użytkownika podania liczb po niektórych flagach. Poniżej przedstawiamy składnię programu.
Dla trybów, które generują graf:

```
./grapher [tryb] [plik] [wiersze] [kolumny] [początek] [koniec]
```

Dla trybu Read mode:

```
./grapher [tryb] [plik] [flaga] [punkty]
```

Argumenty wymagające podania wartości:

- plik
- wiersze
- kolumny
- początek
- koniec
- punkty

Ważnym odnotowania faktem jest to, że punkty powinny zostawać podawane po przecinku przykładowo:

np.
```
./grapher -points 1,2,3,4
```

Poniżej w tabeli pokazujemy jak wyglądają wszystkie flagi wraz z ich, krótszymi wersjami jednoliterowymi oraz z krótkim opisem ich działania.

Flaga	Literkowy odpowiednik	Funkcja flagi
-points	-p	Służy do określenia punktów w trybie Read Mode.
-file	-f	Służy do załączania pliku, do którego zapisujemy graf lub, z którego czytamy graf.
-rows	-o	Służy do określania liczby wierszy w trybach generujących.
-columns	-c	Służy do wprowadzenia liczby kolumn w trybach generujących.
-start	-t	Pozwala określić początek przedziału losowania wag.
-end	-n	Służy do wprowadzania końca przedziału losowania wag.
-WM	-w	Ustawia tryb działania programu na Wage Mode.
-RM	-r	Ustawia tryb działania programu na Read Mode.
-ReM	-m	Ustawia tryb działania programu na Random Mode.
-EM	-e	Ustawia tryb działania programu na Edge Mode.
-standard	-s	Włącza standardowy sposób wyświetlania ścieżki
-enxtended	-x	Włącza rozszerzony sposób wyświetlania ścieżki

Na koniec dodam, że flagi dotyczące trybów mogą się składać z samych małych liter.

5.5 Struktura folderów

W obecnej strukturze zaprezentowanej w tym sprawozdaniu uwzględniliśmy folderu zawierający dokumentację projektu oraz odpowiedzialny za testy.

5.6 Makefile

Makefile został wzbogacony o nowe komendy, które opisane są w rozdziale *Kompilacja programu*. Poniżej przedstawiamy ich listę:

- `make wm`,
- `make rem`,
- `make em`,
- `make rm`.

6 Podsumowanie projektu

Projekt dotyczący grafów w języku C był realizowany od dnia 24.02.2022r do 14.04.2022r. W ramach niego powstały specyfikacja funkcjonalna i implementacyjna oraz moduły programu *grapher* takie jak: *alloc*, *main*, *genGraph*, *readGraph* i *utilts*. Program można uruchamiać z wieloma flagami, które pozwalają na uruchomienie programu z dostosowanymi przez użytkownika wartościami. *Grapher* można uruchomić w czterech różnych trybach: *Wage*, *Random*, *Edge* oraz *Read*. W trybie *Read* użytkownik ma m.in. możliwość wybrania w jaki sposób wyświetlać najkrótszą ścieżkę między zadanymi przez użytkownika punktami dzieki flagom *-standard* i *-extended*. Program został gruntowanie przetestowany, dlatego nie powinno być żadnych niespodziewanych zdarzeń.

7 Wnioski

Sprawdzanie spójności grafów oraz szukanie w nich najkrótszej ścieżki nie jest zadaniem szybkim i trywialnym, a wrecz przeciwnie jest to zadanie wymagające i skomplikowane. Bardzo pomocne w uproszczeniu tych zadań są algorytmy przeszukiwania wszerz (BFS) oraz Dijkstry. Znacząco

usprawniły i uprościły wykonanie tych właśnie zadań. Przy takich projektach wymagające jest również pilnowanie by program natrafiając na błąd informował dokładnie co i dlaczego się wydarzyło, oraz zapobieganie wyciekom pamięci. Z tym ostatnim wsparło nas narzędzie *valgrind*, które pozwoliło nam na skuteczną walkę z wyciekami.