

Sprawozdanie z projektu związanego z algorytmami operującymi na grafach

Adrian Nowosielski
Cezary Skorupski

31.03.2022

Streszczenie

Dokument stanowi sprawozdanie z projektu **Grafy** realizowanego w ramach zajęć projektowych na Politechnice Warszawskiej. Sprawozdanie zawiera wyszczególniony opis aktualnego stanu projektu, zmiany w programie względem specyfikacji funkcjonalnej oraz wnioski z pracy nad projektem.

Spis treści

1	Zarys projektu	1
2	Struktura programu	1
2.1	Struktura katalogów	1
2.2	Struktura modułów	1
2.3	Szczegółowy opis modułów	2
2.4	Diagram modułów	3
3	Struktury danych	4
4	Kompilacja programu	4
5	Przykładowe wyniki działania programu	5
5.1	Generowanie grafu	5
5.2	Czytanie z pliku	6
5.3	Algorytm Bread First Search	6
5.4	Algorytm Dijkstry:	7
6	Zmiany względem specyfikacji funkcjonalnej	8
7	Podsumowanie pracy nad projektem	9
8	Podsumowanie projektu	9
9	Podział pracy w zespole	10
10	Wnioski	10

1 Zarys projektu

Projekt **Grafy** w języku C zakłada stworzenie programu, który będzie generował grafy o postaci kratki. Skrzyżowania kratek to węzły, a linie pionowe i poziome to krawędzie. Dodatkowo badane grafy są ważone, czyli to takie grafy, w których każdej krawędzi przypisana jest pewna wartość liczbową dodatnią. W ramach projektu zespół miał na celu zaimplementowanie:

- Generowanie opisanego grafu w kratkę do pliku .txt z opisanym w specyfikacji funkcjonalnej formacie zapisu grafu.
- Czytanie grafu w kratkę z pliku o określonym w specyfikacji funkcjonalnej formacie.
- Algorytmu Breadth First Search-algorytmu umożliwiającego sprawdzenie, czy podany przez użytkownika graf jest spójny.
- Algorytmu Dijkstra-algorytmu umożliwiającego znalezienie najkrótszej drogi pomiędzy wybranymi przez użytkownika wierzchołkami.

Krokiem, z którym zespół musi się na końcu zmierzyć, to napisanie dokumentu sprawozdania podsumowującego dotychczasowe działania nad projektem.

2 Struktura programu

2.1 Struktura katalogów

Struktura katalogów w programie nie jest skomplikowana. Po wejściu do katalogu głównego programu znajdziemy w nim 3 katalogi:

- **Dokumenty**-katalog, w którym przechowywane są dokumenty projektowe(specyfikacja funkcjonalna,implementacyjna,sprawozdanie).
- **Kod**-katalog, w którym umieszczony jest cały kod potrzebny do uruchomienia programu.
- **Przykłady**-przykładowe pliki wejściowe i parametry uruchomienia programu.

2.2 Struktura modułów

Program składa się z sześciu modułów:

- **bfs_algorithm**- zawiera implementacje algorytmu Breadth First Search
- **dijkstra_algorithm**- zawiera implementacje algorytmu Dijkstry
- **errors**- zawiera potencjalne błędy, które mogą wystąpić podczas działania programu

- **graph**- zawiera generowanie grafu w kratkę
- **priority_queue**- zawiera implementacje kopca, który jest potrzebny do algorytmu Dijkstry.
- **read**- zawiera funkcje, umożliwiające czytanie grafu z pliku .txt o określonym formacie.

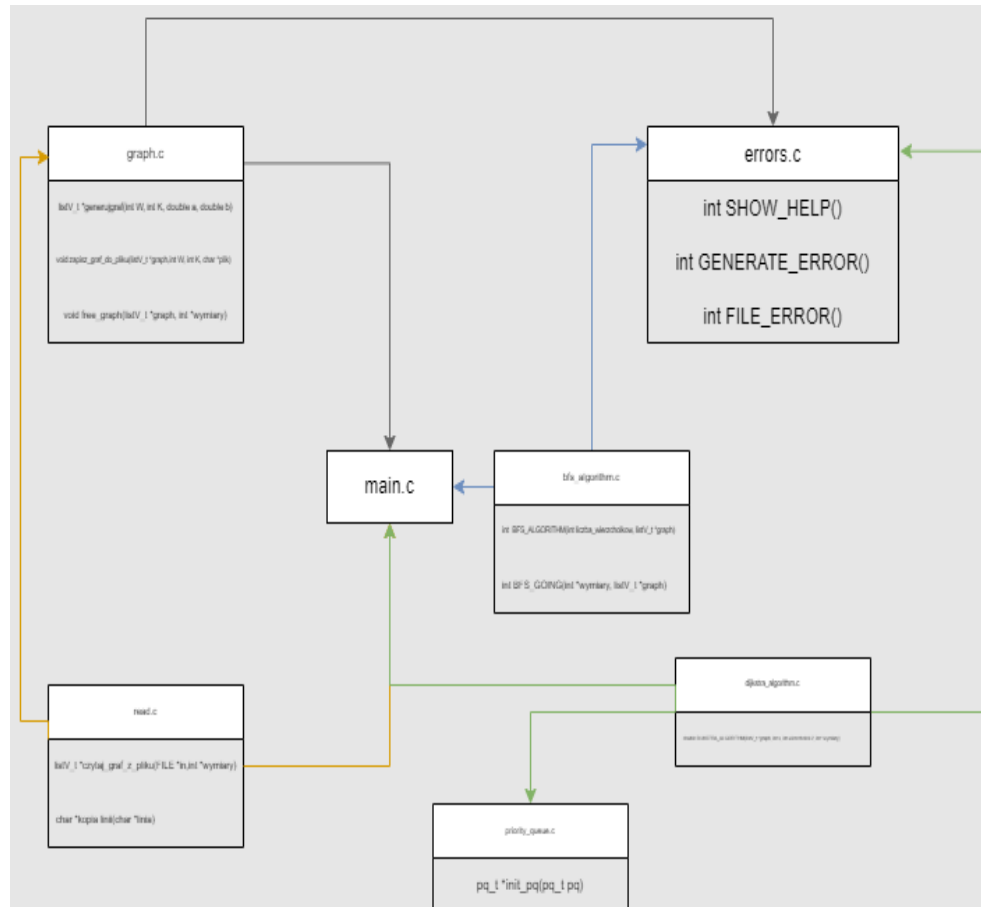
Każdy z modułów składa się z dwóch plików .c i .h, o odpowiedniej nazwie. Programem steruje plik main.c, który zawiera funkcje main całego programu oraz kluczowe makra. W każdym z modułów plik .h zawiera prototypy funkcji oraz ewentualne definicje struktur, stałe oraz makra.

2.3 Szczegółowy opis modułów

- **errors**—moduł w którym umieściliśmy funkcje, które wywołują się w momencie podania nieprawidłowych danych wejściowych przez użytkownika. Zespół zdecydował się na takie rozwiązanie problemu wyświetlania się błędów z powodu wielu wywołań błędów w funkcji main, co na pewno byłoby gorsze dla potencjalnego sprawdzania oraz przeglądania kodu(chodzi zespołowi o wypisywanie wielu funkcji printf w main.c).
- **bfs_algorithm**—w tym module umieściliśmy niezbędne funkcje, które są potrzebne do działania algorytmu BFS, sprawdzającego spójność grafu. W module zawarte są również funkcje realizujące działanie kolejki FIFO, które są potrzebne do poprawnego działania algorytmu. Zespół zdecydował się nie przenosić implementacji kolejki do oddzielnego pliku z niepotrzebnej, wysokiej już liczby plików w programie.
- **graph**—w tym module zespół umieścił niezbędne funkcje do generowania grafu w kratkę, w którym są wszystkie połączenia pomiędzy grafami. Graf w programie jest przechowywany w postaci listy sąsiedztwa.
- **dijkstra_algorithm**—w tym module, zespół umieścił funkcję, która realizuje działanie algorytmu Dijkstra. Zespół zaimplementował dwa podejścia do algorytmu dijkstry-podejście naiwne z gorszą złożonością obliczeniową(kwadratową) oraz podejściem z kopcem ze złożonością superliniową.
- **read**—w przedstawionym module zespół zaimplementował czytanie grafu z pliku .txt o ustalonym formacie. Czytanie z pliku odbywa się poprzez funkcję z biblioteki standardowej-sscanf.
- **priority_queue**—w przedstawionym module zespół napisał funkcję, które są niezbędne do działania kopca, który jest potrzebny do zaimplementowania lepszego podejścia pod względem szybkości działania do algorytmu Dijkstry.

2.4 Diagram modułów

W tej podsekcji, zespół przedstawia diagram modułów, czyli diagram, który służy do ilustracji organizacji i zależności pomiędzy modułami w projekcie:



Rysunek 1: Diagram modułów

3 Struktury danych

Zespół do sprawozdania postanowił dodać podaną sekcję i opisać w jaki sposób dane są przechowywane w pamięci komputera:

- **Przechowywanie grafu**-do reprezentacji grafu wykorzystujemy listę sąsiedztwa. Lista sąsiedztwa zamiast zwykłej tablicy jest realizowana za pomocą tablicy list liniowych. Każdy element listy jest strukturą złożoną z numeru wierzchołka, do którego prowadzi krawędź oraz wagi tego połączenia.
- **Breadth First Search**-w celu zaimplementowania algorytmu, zespół użył kolejki FIFO. Kolejka FIFO jest to liniowa struktura danych, w której nowe dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania.
- **Dijkstra**-w celu uzyskania lepszej złożoności obliczeniowej algorytmu, zespół podjął się zadania napisania algorytmu Dijkstry wykorzystując kopiec. Kopiec to struktura danych oparta na drzewie, w której wartości potomków wężła są w stałej relacji z wartością rodzica (w algorytmie wartość rodzica jest nie większa niż wartości jego potomka). Kopiec pozwala na szybkie znalezienie najmniejszego połączenia pomiędzy węzłami, co znacznie usprawnia złożoność obliczeniową.

4 Kompilacja programu

Skrypt makefile został wyposażony w 4 opcje:

- **Kompilacja domyślna**-kompiluje program standardowo, wszystkie pliki są kompilowane w standardowy sposób.
- **Kompilacja w trybie mem**-Kompiluje program z dodatkowymi informacjami, przydatnymi do sprawdzania np. wycieków pamięci w programie. Analiza wycieków pamięci odbywa się dzięki programowi Valgrind.
- **Clean**-wywołanie make z clean powoduje skasowanie wszystkich plików wykonywalnych oraz obiektowych.

5 Przykładowe wyniki działania programu

Przedstawione poniżej wyniki programu prezentują różne podejścia do uruchomienia programu grafy. Prezentujemy możliwości wszystkich zaimplementowanych flag tak, aby bardziej zarysować możliwości programu:

5.1 Generowanie grafu

Generowanie grafu odbywa się poprzez wpisanie odpowiedniej flagi `-generate` oraz parametrów podczas wywołania programu:

W celu wygenerowania grafu nie potrzebujemy żadnego pliku wejściowego. Musimy jedynie dobrze napisać flagę, która generuje graf: `./grafy -generate [liczba wierszy] [liczba kolumn] [dolny przedział wagi] [gorny przedział wagi]`

Po wpisaniu np. `./grafy -generate 10 10 0 1`, do pliku tekstowego wczytuje nam się następujący format grafu:

```
1 0 1014826354111938 10 0.618653559208602 10 0.1453807270831339
0 0 7609529404524173 2 0.7582866397767052 11 0.1453807270831339
0 0 1318808062308926 3 0.8874892863219675 12 0.432482674779352
2 0 135964156461602628 4 0.1857863155368465 13 0.6860452277808279
0 0 8921432982069226 5 0.2224280965367056 14 0.9985537483772308
4 0 5631038861177414 6 0.8878139808799280 15 0.44397664488242785
0 0 252567896438818 7 0.2682264562458 16 0.225189815155297
6 0 8595643313565519 8 0.6313311097386083 17 0.7819613296454593
0 0 428482721232355 9 0.297972171871656 18 0.6323814804816532
8 0 4385424651488761 19 0.1428436374820856 20 0.98568031642926624
0 0 4274824926361908 11 0.6378083155808224 28 0.98568031642926624
1 0 3265213390470116 10 0.8441453631241651 12 0.5611180276754862 21 0.6406911418962716
0 0 8948218006181817 11 0.9392436745737383 13 0.1484380776487186 22 0.947518083653887
3 0 8154215611989242 12 0.795178398038543 14 0.283388388427638 23 0.7463822615389520
4 0 2599414887092126 13 0.8364652797368088 16 0.4887635238088081 24 0.6485119314864667
0 0 8716649283881736 14 0.3568328578756413 18 0.289862316218228 26 0.3797751261768892
6 0 8885456375146963 15 0.3385297895808073 17 0.218275768264684 26 0.993853650836748
7 0 2566649983628848 16 0.1531878289749958 18 0.269862493114848 27 0.1380248298519823
8 0 7698497209217911 17 0.2683995325418187 19 0.11494343834668761 28 0.8542338399469878
0 0 1081574076805786 18 0.6452286474865808 29 0.7489283721857889
10 0 1046939637068165 21 0.5914488284654158 30 0.4668799622295797
11 0 851325192565888 28 0.583911121510114 22 0.794864695256697 31 0.8453048355096124
12 0 024924818066749 21 0.9112826124381856 23 0.9268671981649786 32 0.8569995857957898
13 0 521846776894487 22 0.763788972989929 24 0.6488542319721138 33 0.3442310538821929
14 0 2242386639553196 23 0.44476909708472848 25 0.6342148737148999 34 0.2359369263228285
15 0 3012286453261329 24 0.8111895408018466 26 0.7198351219816481 35 0.2680928862664772
16 0 3217848388928281 25 0.2376512895513565 27 0.28522348964483809 36 0.1911985189935874
17 0 319482728788535 26 0.4581582768482248 28 0.681726386829364 37 0.1523836381511686
18 0 1118064666215883 27 0.1382772653895799 29 0.5699948265643878 38 0.979968417777814
19 0 1264847578277761 28 0.3278467414332265 39 0.7883722699184898
20 0 6715156280643455 31 0.1613463791889919 40 0.7316367462193625
21 0 491298484891327 30 0.713663788522678 32 0.1133880316067338 41 0.8473780167831825
22 0 7478720769488462 31 0.4899636608106219 33 0.5912423364777318 42 0.0899491812338091
23 0 2158889915888123 32 0.8462859985813644 34 0.6788437812021858 43 0.9882373681283730
24 0 0956461335648897 33 0.1631668262698923 35 0.3448323110792936 44 0.5966523896881119
25 0 784417862624363 34 0.4875714993574738 36 0.614112763760562 45 0.48837781188286
26 0 6192943626521765 35 0.4881880251295834 37 0.4697173524841278 46 0.6398418861713880
27 0 9997766718782356 36 0.1679242670371776 38 0.3499664680951227 47 0.4631192727472257
28 0 764417862624363 37 0.4875714993574738 39 0.614112763760562 48 0.737152187888246
29 0 3172868798739882 38 0.2989879486486488 40 0.6343183473843781
30 0 9886464891081679 41 0.1226686489766516 48 0.68887974918918151
31 0 4892762122985237 48 0.7852999668313655 42 0.9765425347613834 51 0.7683817345715974
32 0 6858152468377588 41 0.181638887114384 43 0.3888991696448374 52 0.545334221174196
33 0 432263271967463 42 0.8986771951482526 44 0.7602283798239678 53 0.8883662457411951
34 0 2154921722578515 43 0.7769392923364576 45 0.818686562277884 54 0.64889568583395
35 0 4238788512580895 44 0.6358319589183721 46 0.4277334666237013 55 0.9153651245475585
36 0 5416482708145158 45 0.48248776792604536 47 0.1687497767676208 56 0.1774977284882929
37 0 2854238882866295 46 0.5579458733824761 48 0.3962939392758848 57 0.5122374884369453
38 0 1747238997497461 47 0.4953488625186485 49 0.3584821802573981 58 0.8383258292457722
39 0 2727668329698867 48 0.3871196113398165 59 0.3193894689954682
40 0 636245486469535 51 0.7624583664889956 60 0.178744835556761
41 0 4266424840813421 58 0.9112618851176329 52 0.5779989428568822 61 0.4288552890172951
42 0 7287425136792138 51 0.9754274882205677 53 0.886488671169362 62 0.8199418943668211
43 0 1634182870217484 52 0.5713499748617178 54 0.3176278080980587 63 0.3654974479899864
44 0 2159410606663782 53 0.80598343757215 55 0.8082788624969868 64 0.1824963173885852
45 0 2159410606663782 54 0.3218331813448263 56 0.8982788624969868 65 0.0368419788855646
46 0 2838064746833524 55 0.89958894469124 57 0.7469238072827338 66 0.5719574666799949
47 0 8889744286748599 56 0.9938882779846173 58 0.8364888926849488 67 0.6827787386681805
48 0 767678338852161 57 0.238388387688112 59 0.1288628867489296 68 0.3996775672953641
49 0 7808084631847954 58 0.9918197467885073 69 0.5148482744440678
50 0 9372805243493247 61 0.5292127396993818 70 0.4785860433105128
```

Rysunek 2: Plik mygraph po wywołaniu flagi `-generate`

Program wyświetla również błędy przy niepoprawnym wywołaniu generowania grafu:

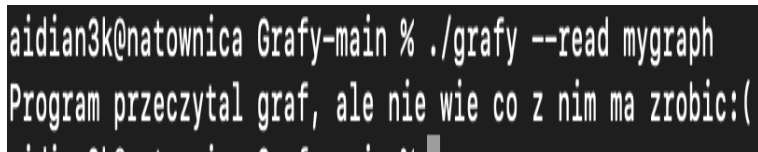
```
aidian3k@natownica Grafy-main % ./grafy --generate 10 10 0
GENERATE_ERROR:
Podano nieprawidłowa liczbe argumentow do wygenerowania grafu
Prawidłowa konwencja generowania grafu:
--generate [liczba wierszy] [liczba kolumn] [waga dolna] [waga gorna]
```

Rysunek 3: Błąd-flaga `-generate`

5.2 Czytanie z pliku

Czytanie z pliku również polega na wpisaniu odpowiedniej flagi podczas uruchomienia programu:

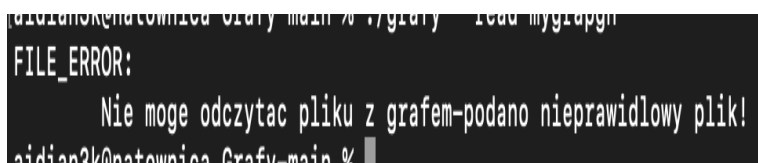
```
./generate --read [nazwa_pliku_z_grafem]
```



```
aidian3k@natownica Grafy-main % ./grafy --read mygraph
Program przeczytał graf, ale nie wie co z nim ma zrobić:(
aidian3k@natownica Grafy-main %
```

Rysunek 4: Odpowiedź programu po wywołaniu flagi `--read` z plikiem

Oczywiście flaga `--read` obsługuje też podstawowe błędy takie, jak np. niepoprawny plik wejściowy:

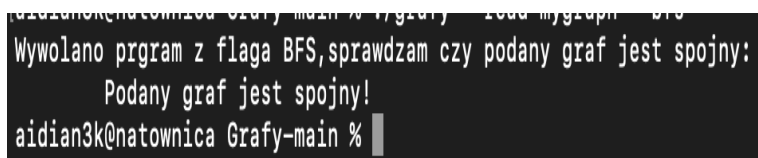


```
aidian3k@natownica Grafy-main % ./grafy --read mygraph
FILE_ERROR:
Nie mogę odczytać pliku z grafem-podano nieprawidłowy plik!
aidian3k@natownica Grafy-main %
```

Rysunek 5: Błąd pliku we flagie `--read`

5.3 Algorytm Bread First Search

Algorytm BFS wywołujemy poprzez dodanie flagi `--bfs` przy uruchomieniu programu. Program daje nam wtedy odpowiedź na pytanie czy podany graf jest spójny, czy nie:



```
aidian3k@natownica Grafy-main % ./grafy --read mygraph --bfs
Wywołano program z flaga BFS, sprawdzam czy podany graf jest spójny:
Podany graf jest spójny!
aidian3k@natownica Grafy-main %
```

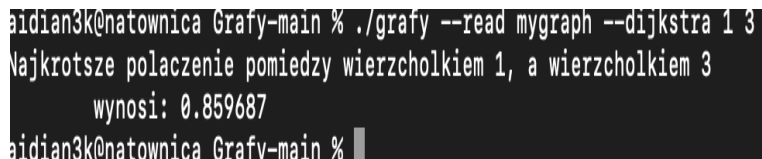
Rysunek 6: Wynik działania flagi `--bfs`

5.4 Algorytm Dijkstry:

Algorytm Dijkstry podobnie do algorytmu Breadth First Search jest uruchamiany poprzez wpisanie odpowiedniej flagi przy uruchomieniu programu.

Przykład uruchomienia algorytmu:

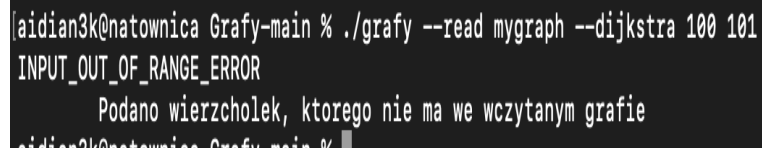
`./grafy -read [nazwa pliku z grafem] -dijkstra [wierzcholek a] [wierzcholek b]` Po wywołaniu flagi program daje nam odpowiedź, jaka jest najmniejsza odległość pomiędzy wierzchołkami a i b:



```
aidian3k@natownica Grafy-main % ./grafy --read mygraph --dijkstra 1 3
Najkrotsze polaczenie pomiedzy wierzchołkiem 1, a wierzchołkiem 3
wynosi: 0.859687
aidian3k@natownica Grafy-main %
```

Rysunek 7: Wynik działania flagi `-dijkstra` dla wierzchołków 1 i 3

Oczywiście tak jak w przypadku innych wyników działania programu, algorytm dijkstry jest odporny na podanie niepoprawnych argumentów przez użytkownika. Na przykład błędem może być podanie wierzchołka, którego nie ma w podanym przez użytkownika grafie:



```
aidian3k@natownica Grafy-main % ./grafy --read mygraph --dijkstra 100 101
INPUT_OUT_OF_RANGE_ERROR
Podano wierzcholek, ktorego nie ma we wczytanym grafie
aidian3k@natownica Grafy-main %
```

Rysunek 8: Wynik działania algorytmu dijkstra dla niepoprawnie podanych wierzchołków

Oczywiście w algorytmie dijkstry jest więcej możliwych błędów. Zostały one opisane w specyfikacji funkcjonalnej oraz sprawozdaniu (opisane w rozdziale 6).

6 Zmiany względem specyfikacji funkcjonalnej

Założeniem projektu od momentu zakończenia pisania specyfikacji funkcjonalnej było ściśle trzymanie się planu programu, który był zawarty w specyfikacji funkcjonalnej. W procesie powstawania programu praca nad projektem zweryfikowała założone w specyfikacji funkcjonalnej idee i skłoniła zespół projektowy do zmiany następujących rzeczy:

- **Usunięcie flagi** `–save`-początkowo zespół zakładał, że dodanie dodatkowej flagi urozmaici program pod względem funkcjonalności. Niestety w ciągu pisania programu okazało się, że flaga `–save` jest zbędna i lepiej zastąpić flagę `–save` automatyzacją zapisu grafu po wywołaniu flagi `–generate`(zakładamy, że jeśli użytkownik ma już zapisany graf, to nie ma sensu go jeszcze raz zapisywać). Plusami rozwiązania problemu jest większa czytelność programu oraz brak zbędnych flag.
- **Usunięcie błędów:**
 - **INCORRECT_NUMBER_OF_ARGS**-błąd zastąpiono błędem `SHOW_HELP()`.
 - **UNKNOWN_FLAG**-kompletne usunięcie flagi. Zespół uznał, że jeśli użytkownik wpisuje flagi, które nie są częścią programu, to program powinien wyświetlać podręcznego HELPA w celu pokazania możliwości programu.
 - **NO_OUT**-usunięto z powodu usunięcia flagi `–save`.
 - **FILENAME_TAKEN**-usunięto z powodu usunięcia flagi `–save`.
 - **COHERENT_ERROR**-w programie graf zawsze jest spójny.
 - **INADEQUATE_NUMBERS_ERR**-zastąpiono innym błędem.
 - **INPUT_ERR**zastąpiono innym błędem.
- **Dodanie nowych błędów**

Podczas pisania programu zespół dodał błędy, których nie uwzględnił w specyfikacji funkcjonalnej:

Komunikat	Przyczyna
SHOW_HELP	Niepoprawny sposób uruchomienia programu.
GENERATE_ERROR	Niepoprawny sposób generowania grafu.
DIJKSTRA_ERROR	Niepoprawny sposób wywoływania algorytmu Dijkstra.
FILE_ERROR	Podano nieprawidłowy plik wejściowy.
INPUT_OUT_OF_RANGE	Podano wierzcholek, którego nie ma we wczytanym grafie.
WEIGHT_ERR	Podano niepoprawny przedział wag krawędzi.
MALLOC_ERR	Brakuje pamięci
INPUT_INT_ERR	Podano niedodatnia liczbę kolumn lub wierszy do generowania grafu!

7 Podsumowanie pracy nad projektem

Współpraca zespołowa podczas czasu trwania projektu przebiegała sprawnie i bez większych problemów. Problemem, który w pewnym momencie projektu zatrzymał zespół w działaniu była niewiedza o tym, jak właściwie ma wyglądać program pod kątem generowania oraz odczytywania grafu. Po konsultacjach projektowych, zespół wiedział, co ma robić i dalsza współpraca przebiegała właściwie. Wartym odnotowania współczynnikiem, który wpłynął na wydajność pracy zespołu było korzystanie z systemu kontroli wersji git. Dzięki systemowi kontroli wersji, zespół mógł sprawnie wymieniać się aktualnymi wersjami programu oraz analizować dokonane przez współpracownika zmiany w programie. Pomocne były także konsultacje projektowe, na których członkowie zespołu wyjaśniali zastosowane w programie rozwiązania oraz przydzielali sobie kolejne zadania. Pomagało to znacznie zorganizować pracę i zrozumieć program pod kątem koncepcyjnym. Podsumowując, współpraca projektowa była na zadowalającym poziomie mimo drobnych błędów i pozwalała na wspólne rozwiązanie problemu i uczenie się efektywnej pracy w zespole programistycznym.

8 Podsumowanie projektu

Projekt **Grafy** został całkowicie zrealizowany w dniach 24.02.2022 do 31.03.2022 przez Adriana Nowosielskiego oraz Cezarego Skorupskiego. W trakcie tego okresu zespołowi udało się stworzyć dokumentację składającą się ze specyfikacji funkcjonalnej oraz sprawozdania końcowego oraz działający program GRAFY z makefile'm. W celu zwiększenia intuicyjności oraz efektywności użytkowania programu przez użytkownika w programie jest wiele rodzajów błędów oraz pomocnych wskazówek, które ułatwiają korzystanie z programu. Zespół daje również możliwość własnego testowania programu pod kątem potencjalnych błędów oraz wycieków pamięci (w katalogu Przykład są zawarte przykładowe niepoprawne dla programu grafy). W konsekwencji pracy nad obsługą błędów oraz wyciekami pamięci, dzięki programowi Valgrind, działanie programu w większości przypadków powinno być zgodne z oczekiwaniami.

9 Podział pracy w zespole

W tej sekcji przedstawimy tylko najważniejsze podziały pracy w zespole. Podczas projektu, staraliśmy się pracować razem nad problemami, dlatego w niektórych zadaniach jest wpisane, że zadanie zostało wykonane wspólnie, ponieważ nie robiła tego tylko jedna osoba. Podczas trwania projektu, zespół starał się pracować razem, by pracować efektywniej nad kodem i wiedzieć co się dzieje w danym momencie trwania projektu:

Specyfikacja funkcjonalna-Zadanie wspólne

Generowanie grafu-Cezary Skorupski

Czytanie grafu z pliku-Adrian Nowosielski

Algorytm BFS-Adrian Nowosielski

Implementacja kopca-Cezary Skorupski

Algorytm Dijkstry-Zadanie wspólne

Testowanie programu-Cezary Skorupski

Wprowadzenie kontroli błędów do programu-Adrian Nowosielski

Zwalnianie pamięci-Zadanie wspólne

Sprawozdanie-Zadanie wspólne

10 Wnioski

Projekt **Grafy** jest zadaniem, które pozwala na różnorodne podejście do problemu grafu i zaimplementowanie różnych rozwiązań, które znacznie wpłyną na szybkość oraz działanie programu. W przypadku algorytmu Dijkstry, czyli algorytmu, który może przeszukiwać grafy o bardzo dużej liczbie wierzchołków, pomocne jest zastosowanie narzędzia, które maksymalnie zmniejszy czas wyszukiwania najkrótszego połączenia pomiędzy wybranymi wierzchołkami. Użyte w programie podejście do algorytmu Dijkstry poprzez kopiec pozwoliło ograniczyć problem szybkości działania algorytmu, ale zmniejszyła czytelność kodu poprzez pojawienie się nowych plików. Dodatkowo implementacja kopca do algorytmu Dijkstry zajęła zespołowi dużo czasu. Istotnym aspektem godnym poruszenia jest ograniczenie do zera wycieków pamięci w programie Grafy, które w przypadku języka C są często ciężkie do wykrycia. Do zwalniania pamięci okazało się potrzebne używanie dodatkowego oprogramowania, które pomagało zespołowi szukać wycieki pamięci, co znacznie ułatwiło pracę przy jej zwalnianiu.