

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

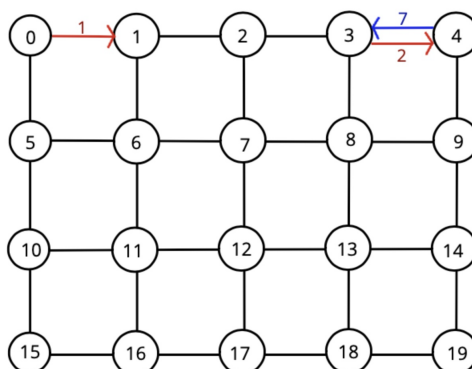
1	Cel projektu	2
2	Struktura programu	2
	2.1 Struktura folderów	3
	2.2 Diagram modułów	3
3	Kompilacja programu	4
4	Przykładowe wywołania i wyniki programu	4
	4.1 Plik wejściowy dla trybów generujących	4
	4.2 Wage Mode	4
	4.3 Edge Mode	5
	4.4 Random Mode	5
	4.5 Read Mode z flagą Standard	6
	4.6 Read Mode z flagą Extended	7
5	Zmiany względem specyfikacji	8
	5.1 Diagram modułów	8
	5.2 Obsługiwane błędy	8
	5.3 Zmiany w strukturach	9
	5.4 Wywoływanie programu	9
	5.5 Struktura folderów	10
	5.6 Makefile	11
	5.7 Zmiany w funkcjach	11
6	Podsumowanie projektu	11
7	Wnioski	11

1 Cel projektu

Celem projektu było stworzenie programu mającego za zadanie generowanie grafów, sprawdzanie ich spójności oraz wyszukiwanie w nich najkrótszej ścieżki między zadanymi przez użytkownika punktami. Grafi są typu *kartka w kratkę*.

- Wage Mode – program generuje graf o losowych wagach dróg między wierzchołkami w taki sposób, że jest on spójny,
- Edge Mode – program losuje istnienie krawędzi między wierzchołkami grafu oraz wagi do momentu powstania grafu spójnego. Do sprawdzania wykorzystuje algorytm BFS,
- Random Mode – program losuje wagi dróg oraz krawędzie między wierzchołkami. W tym trybie graf może być niespójny,
- Read Mode – program odczytuje odpowiednio sformatowany plik i szuka najkrótszej ścieżki między podanymi przez użytkownika punktami za pomocą algorytmu Dijkstry. Format pliku zostanie opisany w osobnej podsekcji, sekcji trzeciej.

Struktura grafu oparta jest na koncepcji "kartka w kratkę" tzn. graf składa się z wierzchołków równo rozmieszczonych na liniach poziomych i pionowych wyznaczanych przez liczbę wierszy i kolumn. Jedyne połączenia zachodzące między wierzchołkami dozwolone są pionowo i poziomo co pokazuje poniższy diagram, na którym zostały zaznaczone jedynie wagi wybranych krawędzi aby zachować czytelność całego diagramu, jednocześnie obrazując schemat połączeń.

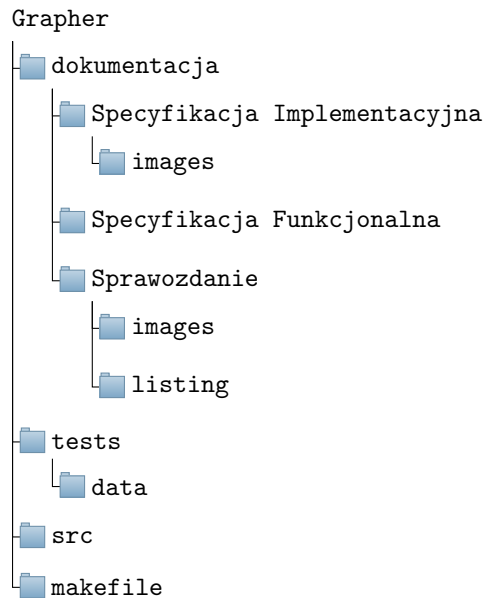


Rysunek 1: Przykład grafu typu "kartka w kratkę"

2 Struktura programu

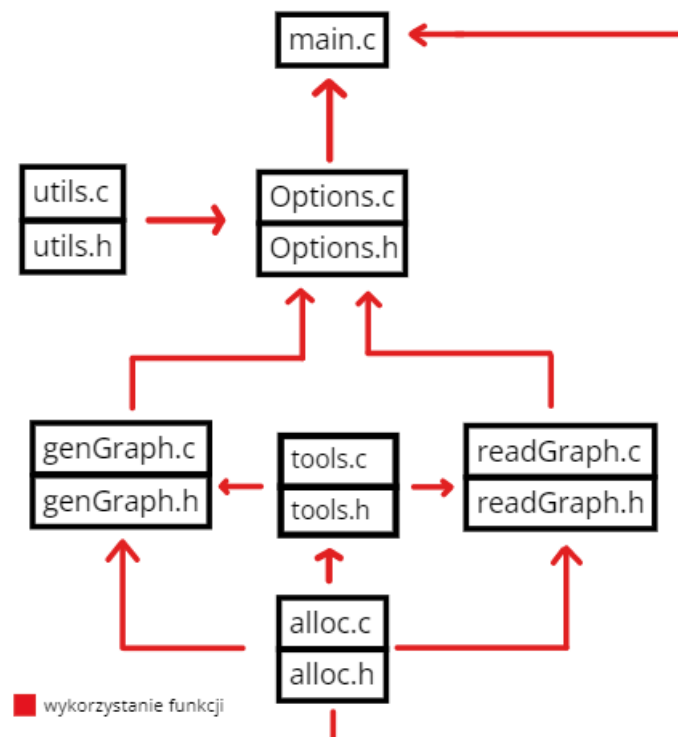
Program *grapher* składa się z 4 folderów nadrzędnych zawierających jego poszczególne elementy. Folder *dokumentacja* zawiera dokumenty opisujące projekt, czyli: specyfikację funkcjonalną i implementacyjną oraz końcowe sprawozdania z projektu. Są w nich pliki **.pdf*, zdjęcia w formacie **.png* i **.jpg* oraz kod źródłowy tych dokumentów w formacie **.tex*. Folder *tests* zawiera kod odpowiedzialny 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 2: 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_s` – również wykonuje to samo zadanie ale program korzysta z trybu *read mode* z flagą *standard*,
- `make rm_e` – działa identycznie jak powyższa z tą różnicą, że z flagą *extended*
- `make test` – komenda służy do wykonania testów funkcji programu.

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 Plik wejściowy dla trybów generujących

W trybach generujących może to być plik pusty ale może to być również plik z danymi z tym, że zostanie on w całości nadpisany.

4.2 Wage Mode

Wywołanie:

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

Wynik:

```
1 4 5
2   1 :2.685866   5 :5.470052
3   2 :3.155808   6 :1.603836   0 :2.151275
4   3 :5.568488   7 :8.678036   1 :6.805621
5   4 :4.893502   8 :6.352131   2 :3.465382
6   9 :8.893915   3 :1.397341
7   0 :2.100994   6 :1.124542   10 :9.470264
8   1 :5.177972   7 :4.706604   11 :3.765574   5 :3.933496
9   2 :9.133201   8 :4.643032   12 :7.491190   6 :5.518209
10  3 :2.872742   9 :4.960544   13 :4.615400   7 :3.383972
11  4 :6.784856   14 :9.889497   8 :7.879598
12  5 :8.470722   11 :5.359549   15 :1.035407
13  6 :9.074558   12 :6.510824   16 :5.603895   10 :7.752595
14  7 :3.316445   13 :9.497397   17 :4.104725   11 :5.781827
15  8 :8.391312   14 :4.502067   18 :6.882821   12 :8.515854
16  9 :3.972331   19 :2.060793   13 :3.222458
```

```

17      10 :6.737906   16 :4.994289
18      11 :2.355659   17 :1.380938   15 :2.485479
19      12 :6.873868   18 :3.253679   16 :6.446023
20      13 :1.489268   19 :5.637651   17 :3.230879
21      14 :1.378765   18 :3.517249

```

4.3 Edge Mode

Wywołanie:

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

Wynik:

```

1 5 7
2      1 :14.079570   7 :18.013451
3      8 :18.027349   0 :8.487977
4      3 :10.579045   1 :18.637221
5      4 :5.258607   10 :7.197692   2 :15.897541
6      5 :19.195051   11 :9.345530   3 :15.480282
7      6 :17.268742   12 :16.839028   4 :7.884089
8      13 :17.236659   5 :14.512482
9      0 :9.893743   8 :12.356867   14 :7.506790
10     1 :7.865755   9 :7.874695
11     2 :8.110882   16 :5.115988   8 :8.157934
12     3 :7.248958   11 :12.474005   17 :18.592836
13     4 :16.970298   12 :16.095942   18 :9.500838   10 :13.332601
14     5 :19.161458   13 :13.960831   19 :18.983178   11 :9.366063
15     6 :10.753299   20 :10.594030   12 :5.453957
16     7 :13.135396   21 :19.473409
17     22 :5.860485   14 :7.825802
18     17 :15.299807   23 :6.726627
19     10 :9.560902   18 :16.138496   24 :12.203447   16 :15.495648
20     11 :6.480293   19 :14.071909   25 :10.351046   17 :5.993468
21     12 :16.341457   20 :19.739661   18 :18.111915
22     13 :8.936962   27 :15.667228   19 :11.762764
23     22 :12.689115   28 :6.990043
24     15 :12.511560   23 :11.550945   29 :18.400750   21 :8.843950
25     24 :7.714718   30 :15.746017   22 :17.931632
26     17 :15.041366   25 :9.103630   31 :15.380775   23 :7.640766
27     18 :9.297662   26 :12.901612   32 :18.555363   24 :16.178947
28     19 :14.363605   27 :6.283068   33 :10.752525   25 :15.325887
29     34 :19.517290   26 :6.844361
30     21 :10.190828   29 :13.772129
31     22 :5.475628   30 :6.971490   28 :6.855536
32     23 :19.712006   31 :15.893046   29 :5.518989
33     30 :6.691543
34     25 :8.755690   33 :5.361426   31 :14.560269
35     34 :8.460465   32 :8.341884
36     27 :5.071621   33 :18.667771

```

4.4 Random Mode

Wywołanie:

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

Wynik:

```

1 6 7
2      1 :3.359505   7 :1.529356
3
4

```

```

5      10 :7.888522  2 :7.963282
6      11 :9.462220
7      6 :8.687498
8      13 :2.485468
9      0 :7.233819  8 :4.918381  14 :5.328805
10     9 :5.858161  15 :9.694184
11     2 :6.325664  10 :9.820746
12     3 :6.913050  17 :6.375269  9 :6.264576
13     4 :3.655974  12 :5.457724  18 :9.830380
14     5 :4.396012  13 :3.488069
15     20 :2.522711
16     21 :1.763478
17     16 :1.095031
18     17 :3.461065  23 :6.904377
19     10 :1.216452  18 :8.587263  24 :8.203541
20     11 :8.343906  19 :3.167662
21     26 :6.071088  18 :5.774019
22
23     22 :3.032561  28 :5.798855
24     15 :5.841460  23 :3.017458  21 :4.999015
25     24 :3.202555
26     17 :1.287662
27     18 :3.593484  26 :1.398753
28     27 :4.213209  33 :7.448281
29     34 :1.739397  26 :4.950368
30     21 :3.343270
31     22 :8.836570  28 :1.504886
32     31 :6.813503
33     30 :8.042800
34     39 :8.945608  31 :7.501781
35     32 :1.988769
36     27 :3.559395
37     28 :6.695392
38     29 :1.941324
39     30 :6.030715  36 :5.437702
40     39 :9.491460  37 :6.246970
41     38 :1.625709
42     33 :2.352715  39 :2.568840
43     40 :5.128235

```

4.5 Read Mode z flagą Standard

Plik wejściowy :

```

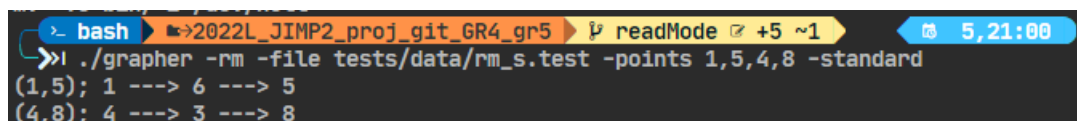
1 4 5
2      1 :2.685866  5 :5.470052
3      2 :3.155808  6 :1.603836  0 :2.151275
4      3 :5.568488  7 :8.678036  1 :6.805621
5      4 :4.893502  8 :6.352131  2 :3.465382
6      9 :8.893915  3 :1.397341
7      0 :2.100994  6 :1.124542  10 :9.470264
8      1 :5.177972  7 :4.706604  11 :3.765574  5 :3.933496
9      2 :9.133201  8 :4.643032  12 :7.491190  6 :5.518209
10     3 :2.872742  9 :4.960544  13 :4.615400  7 :3.383972
11     4 :6.784856  14 :9.889497  8 :7.879598
12     5 :8.470722  11 :5.359549  15 :1.035407
13     6 :9.074558  12 :6.510824  16 :5.603895  10 :7.752595
14     7 :3.316445  13 :9.497397  17 :4.104725  11 :5.781827
15     8 :8.391312  14 :4.502067  18 :6.882821  12 :8.515854
16     9 :3.972331  19 :2.060793  13 :3.222458
17     10 :6.737906  16 :4.994289
18     11 :2.355659  17 :1.380938  15 :2.485479
19     12 :6.873868  18 :3.253679  16 :6.446023
20     13 :1.489268  19 :5.637651  17 :3.230879
21     14 :1.378765  18 :3.517249

```


Wywołanie:

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

Wynik:



```
bash ➤ 2022L_JIMP2_proj_git_GR4_gr5 ➤ readMode +5 ~1 5,21:00
>>| ./grapher -rm -file tests/data/rm_s.test -points 1,5,4,8 -standard
(1,5); 1 ---> 6 ---> 5
(4,8); 4 ---> 3 ---> 8
```

Rysunek 3: Wynik działania dla Read Mode z flagą -standard

4.6 Read Mode z flagą Extended

Plik wejściowy :

```
1 5 7
2 1 :14.079570 7 :18.013451
3 8 :18.027349 0 :8.487977
4 3 :10.579045 1 :18.637221
5 4 :5.258607 10 :7.197692 2 :15.897541
6 5 :19.195051 11 :9.345530 3 :15.480282
7 6 :17.268742 12 :16.839028 4 :7.884089
8 13 :17.236659 5 :14.512482
9 0 :9.893743 8 :12.356867 14 :7.506790
10 1 :7.865755 9 :7.874695
11 2 :8.110882 16 :5.115988 8 :8.157934
12 3 :7.248958 11 :12.474005 17 :18.592836
13 4 :16.970298 12 :16.095942 18 :9.500838 10 :13.332601
14 5 :19.161458 13 :13.960831 19 :18.983178 11 :9.366063
15 6 :10.753299 20 :10.594030 12 :5.453957
16 7 :13.135396 21 :19.473409
17 22 :5.860485 14 :7.825802
18 17 :15.299807 23 :6.726627
19 10 :9.560902 18 :16.138496 24 :12.203447 16 :15.495648
20 11 :6.480293 19 :14.071909 25 :10.351046 17 :5.993468
21 12 :16.341457 20 :19.739661 18 :18.111915
22 13 :8.936962 27 :15.667228 19 :11.762764
23 22 :12.689115 28 :6.990043
24 15 :12.511560 23 :11.550945 29 :18.400750 21 :8.843950
25 24 :7.714718 30 :15.746017 22 :17.931632
26 17 :15.041366 25 :9.103630 31 :15.380775 23 :7.640766
27 18 :9.297662 26 :12.901612 32 :18.555363 24 :16.178947
28 19 :14.363605 27 :6.283068 33 :10.752525 25 :15.325887
29 34 :19.517290 26 :6.844361
30 21 :10.190828 29 :13.772129
31 22 :5.475628 30 :6.971490 28 :6.855536
32 23 :19.712006 31 :15.893046 29 :5.518989
33 30 :6.691543
34 25 :8.755690 33 :5.361426 31 :14.560269
35 34 :8.460465 32 :8.341884
36 27 :5.071621 33 :18.667771
```

Wywołanie:

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

Wynik:

```

bash 2022L_JIMP2_proj_git_GR4_gr5 readMode +5 ~1 5,21:00
./grapher -rm -extended -points 2,7,3,11 -file tests/data/rm_e.test
(2,7); 2(18.637221) ---> 1(8.487977) ---> 0(18.013451) ---> 7
(3,11); 3(5.258607) ---> 4(9.345530) ---> 11

```

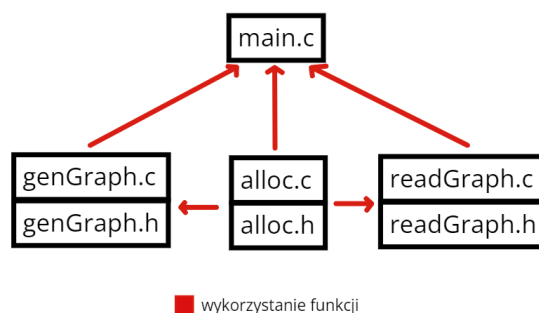
Rysunek 4: Wynik dla Read Mode z flagą -extended

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. Doszedł moduł *utils* wspomagający pracę *maina* w zakresie obsługi błędów. Zostały dodane trzy nowe moduły *tools*, *options* oraz *utils*. Ten pierwszy przechowuje funkcje odpowiedzialne za kolejkę oraz sprawdzanie spójności grafu, drugi wspomaga działanie *maina* w taki sposób, że przejmuje jego odpowiedzialność w zakresie wywoływania funkcji odpowiadających za działanie trybów oraz za przypisywanie wartości z wywołania programu. Moduł *utils* jest modulem współpracującym z *options*. Pomaga on w sprawdzaniu danych wejściowych oraz weryfikuje podanie wszystkich niezbędnych wartości. Poniżej pokażemy starą wersję diagramu modułów, a wersja najnowsza jest przedstawiona wyżej.



Rysunek 5: Diagram modułów – stara wersja

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
MULTIPLE_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
NO_COL_ROWS_FOUND	223	W pliku do czytania nie znaleziono kolumn lub wierszy
NO_NODES_FOUND	220	W trybie nie znaleziono wierzchołków

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 oraz zmieniliśmy typ zmiennej *printFlag* żeby móc sprawdzać poprawnie jego podanie. Dodano również dwie nowe zmienne *rows* i *columns* odpowiedzialne za przechowanie liczby wierzchołków i kolumn,

```

1 typedef struct entryRead {
2     int rows;
3     int columns;
4     char* fileName;
5     short int printFlag;
6     int* points;
7     int numberPoints;
8 } entryR;
```

- **struct graphRead** – została całkowicie usunięta, ponieważ podczas implementacji okazała się bezużyteczna,
- **struct node** – ta struktura otrzymała nową zmienną tablicową *nodeToConnect* oraz wszystkie tablice zostały zmienione ze wskaźników na tablice o określonym rozmiarze.

```

1 typedef struct node {
2     bool edgeExist[4];
3     double edgeWeight[4];
4     int nodeToConnect[4];
5 } 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 poza jednym wyjątek. Owym wyjątkiem jest tryb działania programu, który musi być podawany jako pierwszy argument wywołania programu. 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 odnotowaniem 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
-extended	-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_s`,
- `make rm_e`.

5.7 Zmiany w funkcjach

Postanowiliśmy zrezygnować ze zmiennych będącymi wskaźnikami na wskaźniki, ponieważ stwierdziliśmy, że nie ma sensu ich stosować w przypadku naszego programu. Dzięki dodanym nowym modułom umiejscowienie funkcji się również zmieniło i niektóre funkcje stały się bardziej uniwersalne.

`int checkIfCoherentRead(graphR** graph, entryR* entry)` oraz
`int checkIfCoherentGen(node** graph, entryG* entry)` zostało zastąpione jedną funkcją:

`bool checkIfCoherent(node* graph, int numOfNodes)`, która otrzymuje graf oraz liczbę wierzchołków i zwraca wartość *true* lub *false*.

`void printShortPath(entryR* entry, int* parents) →`
`void printShortPath(entryR* entry, int* predecessors, int startPoint, int endPoint),`

`void printExtendedPath(entryR* entry, int* parents, double* weights) →`
`void printExtendedPath(entryR* entry, int* predecessors, double* weights, int startPoint, int endPoint).`

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 *utils*. 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 dzięki flagom *-standard* i *-extended*. Program został gruntownie 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 wręcz przeciwnie jest to zadanie wymagające i skomplikowane. Bardzo pomocne w uproszczeniu tych zadań są algorytmy przeszukiwania wszcz (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.