Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Temat *Projekt 2 - grafy*	**Termin zajęć** *Poniedziałek 18:55*
Imię, nazwisko, numer albumu *Paweł Wójcik, 259341*	**Ocena**
Wykładowca *mgr Marta Emirsajłow*	
Kod grupy *Y03-51c*	**Data złożenia sprawozdania** 23.05.2022r.

Sprawozdanie nr. 2

Spis treści

1 Wprowadzenie

Projekt polegał na zaimplementowaniu dwóch typów reprezentacji grafów- macierzy sąsiedztwa i listy sąsiedztwa oraz zaimplementowaniu i sprawdzeniu działania jednego z algorytmów Dijkstry lub Bellmana-Forda. Należało porównać ze sobą działanie różnych typów reprezentacji. Testy były przeprowadzone na skierowanych grafach generowanych losowo dla różnych ilości wierzchołków: 10, 100, 250, 500, 1000 oraz dla różnych gęstości grafów: 25%, 50%, 75%, 100%. Każda z konfiguracji była testowana dla 100 grafów, co daje nam statystyczne uzasadnienie poprawności otrzymanych wyników. Sprawdzenie czasów działania dla różnych konfiguracji daje nam pogląd na szybkość działania algorytmu i typu reprezentacji. Ponadto w programie została zaimplementowana możliwość odczytania grafu z pliku, rozwiązania go i zapisania rozwiązania do pliku wyjściowego.

2 Teoretyczny opis algorytmów

2.1 Algorytm Bellmana-Forda

Służy do znajdowania najkrótszej ścieżki w grafie pomiędzy wierzchołkami dla grafów w których dopuszczane są ujemne wagi krawędzi, jednak należy pamiętać o tym, że w grafie nie może występować ujemny cykl, który spowodowałby złe wyznaczenie ścieżek. W programie zaimplementowane zostało zabezpieczenie przed ujemnymi cyklami, w takim wypadku wyświetlony zostaje komunikat o braku możliwości znalezienia prawidłowych ścieżek i ich wartości. Działanie algorytmu polega na relaksacji krawędzi t.j. przejściu po grafie V-1 razy po każdej krawędzi sprawdzając czy przejście po tej krawędzi dla danego wierzchołka nie skróci ścieżki czyli jej wartości. Złożoność czasowa tego algorytmu w notacji dużego „O" wynosi(gdzie E to krawędzie, a V to wierzchołki):

$$O(E * V)$$

2.2 Algorytm Dijkstry

Służy do znajdowania najkrótszej ścieżki w grafie pomiędzy wierzchołkami dla grafów z krawędziami o nieujemnych wagach. Działanie algorytmu zaimplementowanego w programie polega na znalezieniu ścieżek o minimalnej wartości z wierzchołka początkowego do wszystkich innych wierzchołków. Implementacja algorytmu oparta jest o kolejkę priorytetową t.j. w momencie znalezienia krótszej niż dotychczas ścieżki do wierzchołka dodajemy owy wierzchołek do kolejki priorytetowej z odpowiednią wagą upewniając się czy nie był on już odwiedzany, tym sposobem ograniczamy złożoność czasową algorytmu. Przechodzimy po wszystkich wierzchołkach zdejmując z kolejki priorytetowej wierzchołek o najmniejszej wadze, dzięki czemu mamy pewność przejścia po wszystkich wierzchołkach oraz możemy stwierdzić, że ścieżki wyznaczone przez algorytm są poprawne. Należy zauważyć, że algorytm Dijkstry działa poprawnie tylko dla grafów o krawędziach z wagami nieujemnymi, a jego złożoność czasowa w notacji dużego „O" wynosi(gdzie E to krawędzie, a V to wierzchołki):

$$O(E * logV)$$

2.3 Reprezentacja grafu za pomocą macierzy sąsiedztwa

Ta reprezentacja polega na przechowywaniu grafu w macierzy, gdzie elementy o odpowiednich indeksach to wagi krawędzi między dwoma wierzchołkami. W programie implementacja jest oparta o dwuwymiarową tablicę wskaźników na wagi krawędzi, jeżeli krawędź nieistnieje to wskaźnik na nic nie wskazuje (nullptr). Takie rozwiązanie jest wygodne w implementacji i przeszukiwaniu ale powoduje, że musimy sprawdzić wszystkie elementy macierzy(w programie tablicy) i patrzeć czy krawędź istnieje, a jeżeli tak to dopiero przechodzić do sprawdzenia wagi.

2.4 Reprezentacja grafu za pomocą listy sąsiedztwa

Problem sprawdzania istnienia wszystkich krawędzi rozwiązuje właśnie lista sąsiedztwa, jest to jednowymiarowa tablica wskaźników na listy przechowujące krawędzie danych wierzchołków. Lista składa się ze struktur przechowujących do jakiego wierzchołku prowadzi krawędź, wagę tej krawędzi i wskaźnik na następny element. Takie rozwiązanie powoduje, że nie sprawdzamy dla każdego wierzchołka istnienia krawędzi do wszystkich pozostałych tylko przechodzimy po liście i patrzymy tylko na krawędzie, które

są przypisane do sprawdzanego wierzchołka. Co powoduje, że możemy oczekiwać, zwłaszcza dla grafów o małej gęstości szybszego wykonywania się algorytmów rozwiązujących graf.

3 Wyniki testów

Badania przeprowadzane były dla próbki 100 grafów o odpowiednich ilościach wierzchołków:

- 10
- 100
- 250
- 500
- 1000

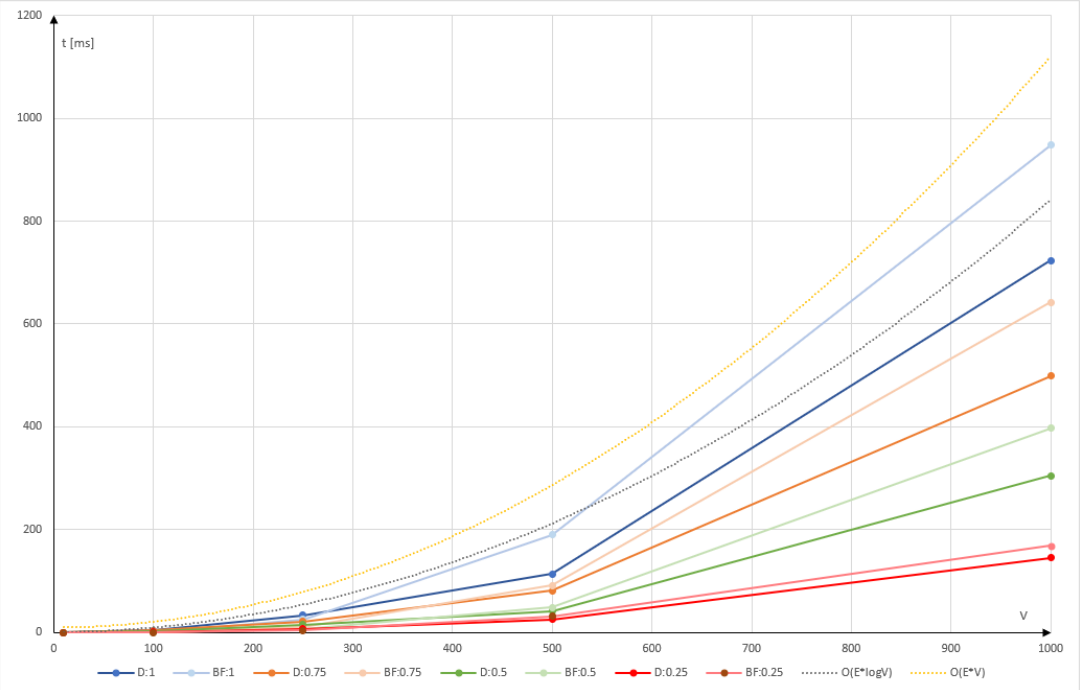
oraz dla różnych gęstości grafu:

- 25%
- 50%
- 75%
- 100%

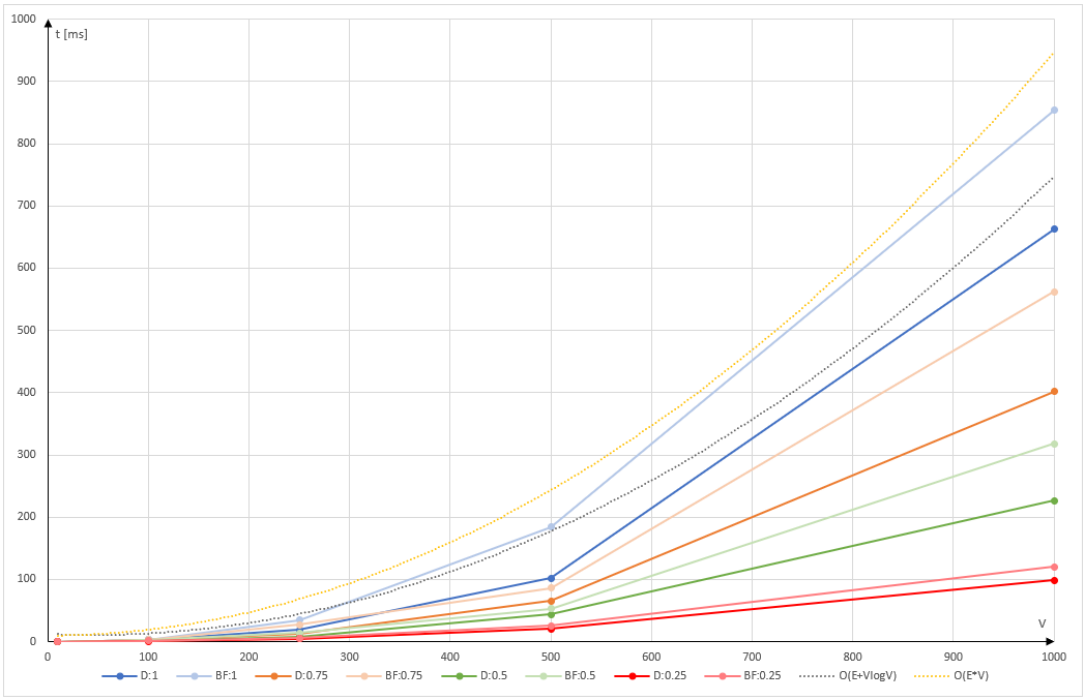
Wyznaczenie czasów działania poszczególnych konfiguracji polegała na wygenerowaniu 100 losowych grafów o określonych parametrach (ilość wierzchołków i gęstość), następnie rozwiązaniu ich wszystkich i wyznaczeniu czasu średniego dla jednego grafu. Poprawność działania algorytmów sprawdzona została za pomocą odczytu grafu z pliku i zapisaniu rozwiązania do drugiego pliku.

Czasy przedstawione w wykresach 1, 2, 3, 4, 5, 6 oraz w tabelach 1, 2, 3, 4 to czasy średnie dla posortowania jednej tablicy z próbki 100 tablic podane w milisekundach. Dzięki takiej konfiguracji możemy poprzeć wyniki eksperymentu statystyczną analizą i sprawdzić działanie algorytmów dla różnych ilości wierzchołków i gęstości grafów co pozwala na ocenę poprawności i jakości działania implementacji.

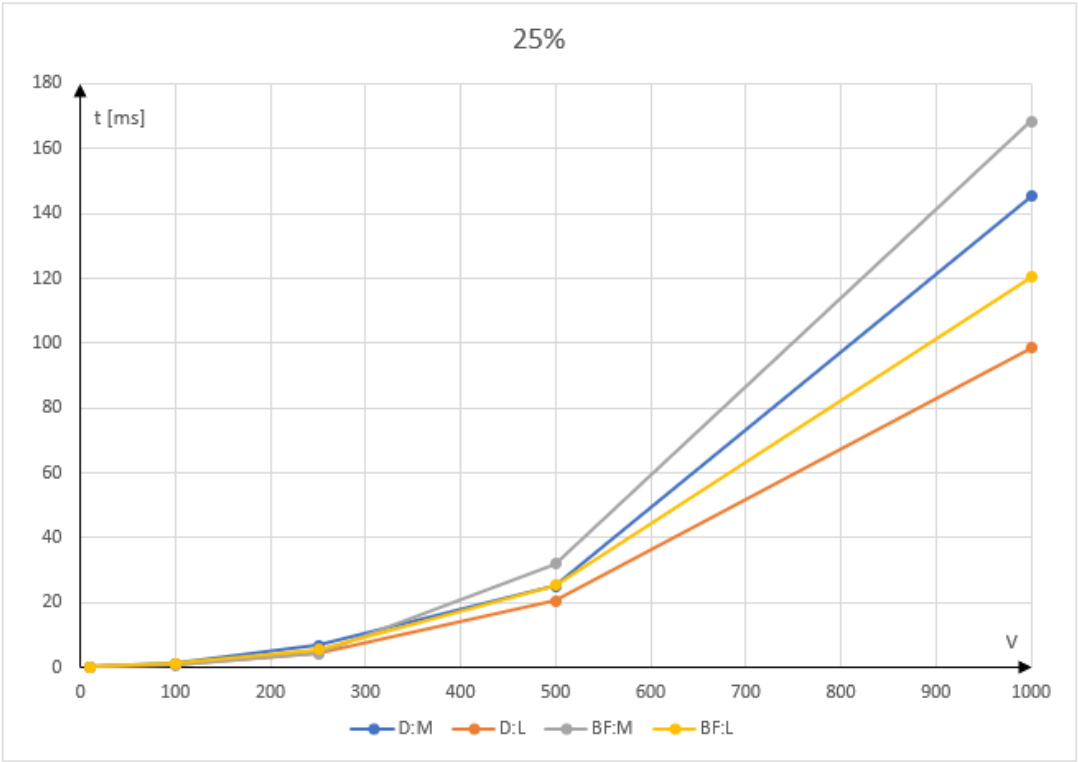
Linie przerywane na wykresach to funkcje ograniczające złożoności pamięciowe z góry z dokładnością do stałych.



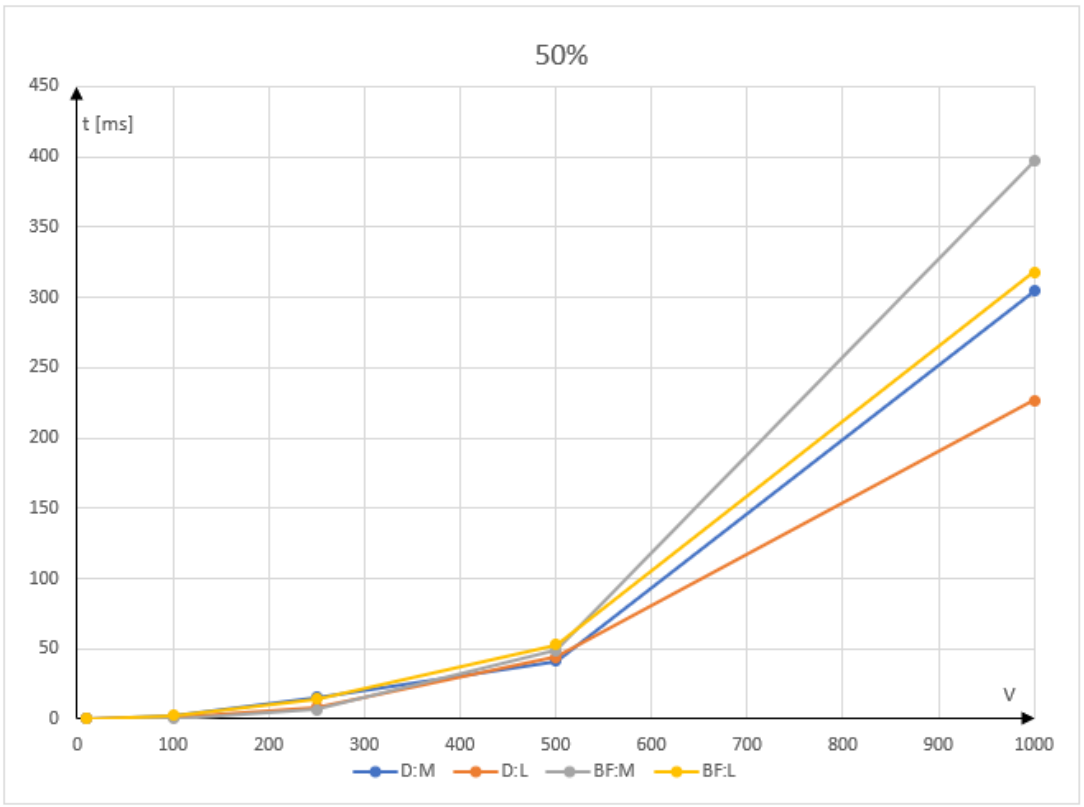
Rysunek 1: Reprezentacja grafu za pomocą macierzy sąsiedztwa



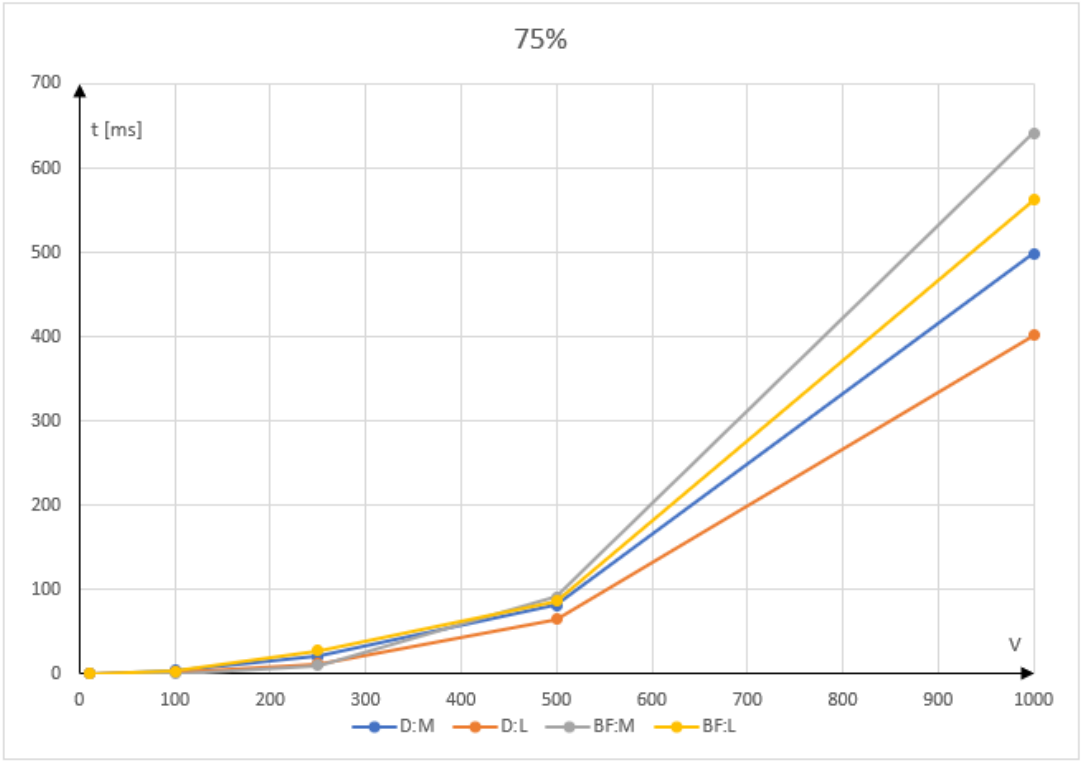
Rysunek 2: Reprezentacja grafu za pomocą listy sąsiedztwa



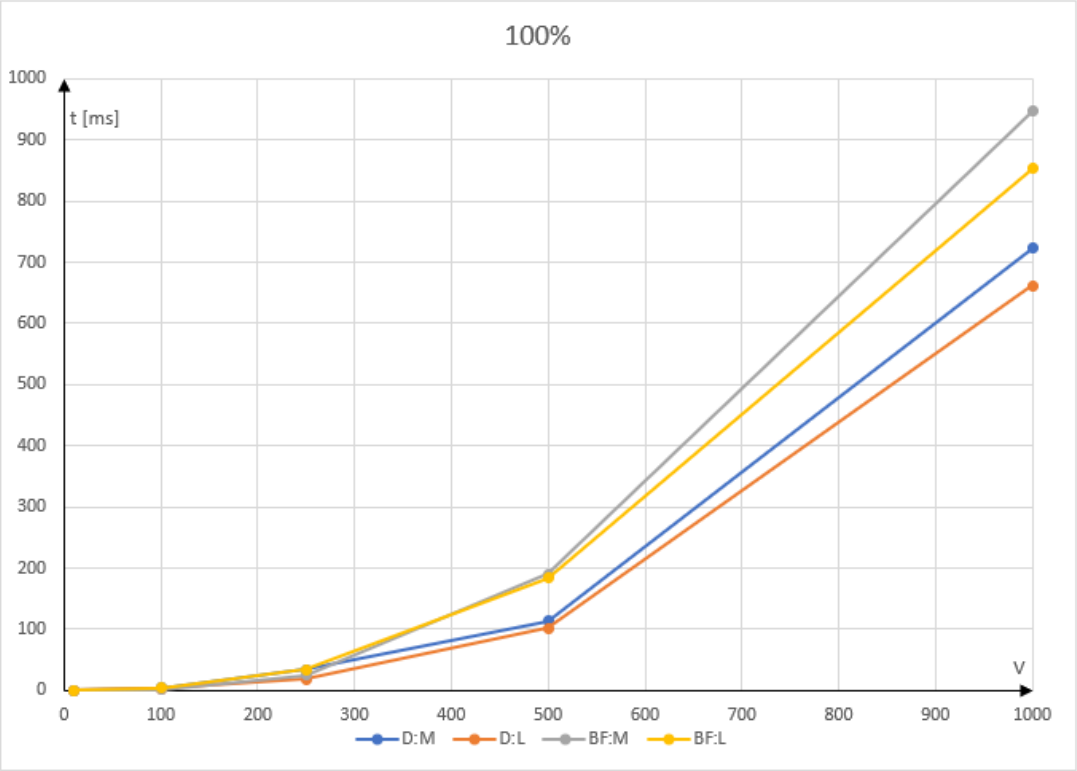
Rysunek 3: Porównanie konfiguracji dla gęstości 25%



Rysunek 4: Porównanie konfiguracji dla gęstości 50%



Rysunek 5: Porównanie konfiguracji dla gęstości 75%



Rysunek 6: Porównanie konfiguracji dla gęstości 100%

Macierz, Dijkstra					
	10	100	250	500	1000
0.25	0,12	1,25	6,73	25,05	145,23
0.5	0,18	2,12	15,41	41,29	304,52
0.75	0,21	3,98	21,28	81,43	498,61
1	0,23	4,56	33,81	113,65	723,49

Rysunek 7: Wyniki algorytmu Dijkstry z macierzą sąsiedztwa

Macierz, Bellman-Ford					
	10	100	250	500	1000
0.25	0,17	0,58	4,21	31,99	168,49
0.5	0,13	0,5	6,97	48,83	396,82
0.75	0,16	0,61	10,04	91,75	642,05
1	0,15	0,93	25,36	190,29	947,74

Rysunek 8: Wyniki algorytmu Bellmana-Forda z macierzą sąsiedztwa

Lista, Dijkstra					
	10	100	250	500	1000
0.25	0,15	0,87	4,44	20,47	98,5
0.5	0,17	1,56	8,4	44,16	226,68
0.75	0,18	2,19	12,51	65,12	401,97
1	0,2	2,71	18,5	102,36	662,32

Rysunek 9: Wyniki algorytmu Dijkstry z listą sąsiedztwa

Lista, Bellman-Ford					
	10	100	250	500	1000
0.25	0,15	1,25	5,47	25,26	120,38
0.5	0,16	2,47	14,28	53,09	318,12
0.75	0,13	2,93	27,58	86,38	562,15
1	0,17	3,47	34,58	183,85	853,58

Rysunek 10: Wyniki algorytmu Bellmana-Forda z listą sąsiedztwa

4 Wnioski

Obie implementacje reprezentacji grafu t.j. macierz sąsiedztwa i lista sąsiedztwa oraz oba algorytmy t.j. algorytm Dijkstry i algorytm Bellmana-Forda działają poprawnie, odczytują graf z pliku w formacie określonym w założeniach projektu(należy jednak pamiętać, że plik wejściowy nie może zawierać pustych linii bo może to powodować nieprawidłowe działanie algorytmu), a następnie zapisuje rozwiązany graf do pliku wyjściowego w formacie określonym w założeniach projektu.

Przechodząc do interpretacji otrzymanych wyników czasowych możemy określić, że oba algorytmy działały prawidłowo w swoich złożonościach czasowych co widać na wykresach 1 i 2. Dla implementacji w moim projekcie reprezentacja grafu za pomocą macierzy jest ewidentnie wolniejsza niż reprezentacja za pomocą lista sąsiedztwa dla wykonywania algorytmów rozwiązywania grafów, lecz warto zauważyć, że im większa gęstość grafu tym ta różnica stosunkowo zmniejsza się co widać na wykresach od 3 do 6. Wynika to ze sposobu przeszukiwania grafów podczas ich przechodzenia, w liście sprawdzamy tylko wierzchołki które dla danego grafu istnieją i mają określoną wagę ponieważ do każdego wierzchołka dopisana jest lista przechowująca jego krawędzie, a dla macierzy przeszukujemy potencjalne połączenie z każdym innym wierzchołkiem i dopiero wtedy sprawdzamy czy istnieje.

Złożoności czasowe również zdają się być zgodne z założeniami teoretycznymi ponieważ niezależnie od sposobu reprezentacji grafu dla algorytmu Dijkstry otrzymaliśmy złożoność czasową w notacji dużego „O” wynoszącą:

$$O(E * logV)$$

A dla algorytmu Bellmana-Forda:

$$O(E * V)$$

Podsumowując, algorytmy działają poprawnie znajdując odpowiednie najkrótsze ścieżki i ich wagi, program poprawnie odczytuje graf z pliku i poprawnie również go zapisuje, a złożoności czasowe dla algorytmów są zgodne z założeniami teoretycznymi. Ponadto reprezentacje grafów wydają się działać odpowiednio z założeniami teoretycznymi więc możemy stwierdzić, że implementacje reprezentacji grafów i algorytmów rozwiązujących grafy są poprawne i działają zgodnie z oczekiwaniami.

5 Bibliografia

- https://pl.wikipedia.org/wiki/Reprezentacja_grafu
- https://eduinf.waw.pl/inf/alg/001_search/0123.php#wagi
- https://eduinf.waw.pl/inf/utils/002_roz/ol011.php
- https://eduinf.waw.pl/inf/alg/001_search/0138a.php
- https://pl.wikipedia.org/wiki/Algorytm_Bellmana-Forda
- https://eduinf.waw.pl/inf/alg/001_search/0124.php
- https://pl.wikipedia.org/wiki/Algorytm_Dijkstry