Algorytmy i złożoność obliczeniowa – projekt 2

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera

Autor: Jakub Smolarczyk

Indeks: 272924

Grupa 18 – termin: czw. parzysty 11:15

Prowadzący: Dariusz Banasiak

Spis treści:

1. Wprowadzenie

Grafem określana jest struktura matematyczna służąca do przedstawiania i badania relacji między obiektami. Relacje te reprezentowane są w postaci krawędzi, przy czym każda krawędź ma swój początek oraz koniec w dowolnym wierzchołku (reprezentującym obiekt) grafu. Krawędzie te mogą być zarówno skierowane jak i nieskierowane, posiadać różne wagi oraz prowadzić do tego samego wierzchołka, z którego wychodzą.

Celem tego projektu było napisanie programu, który zbada wybrane problemy grafowe, w postaci wykonania na grafie wybranych algorytmów oraz zmierzenia czasu ich wykonania. W badaniu badane były grafy reprezentowane macierzą incydencji oraz listą następników. Badane problemy grafowe to:

- Wyznaczanie minimalnego drzewa rozpinającego (MST):
  1. Algorytm Prima

     Polega on na wyborze wierzchołka startowego a następnie zbudowaniu drzewa rozpinającego (grafu spójnego bez cykli) tak aby suma wag jego krawędzi była jak najmniejsza. W tym celu wybiera się krawędź wychodzącą z jednego z tych wierzchołków, które są już częścią budowanego drzewa, preferując krawędzie o najmniejszej wadze, które kończą się w wierzchołku jeszcze nie włączonym do tego drzewa. Algorytm trwa tak długo aż wszystkie wierzchołki staną się częścią drzewa. Złożoność takiego algorytmu wynosi $O(E * \log V)$, przy czym E oznacza liczbę krawędzi grafu, a V oznacza liczbę jego wierzchołków.

  2. Algorytm Kruskala

     Ideą tego algorytmu jest stworzenie V grafów jednowierzchołkowych oraz zbioru krawędzi posortowanych rosnąco względem ich wag. Następnie pobierane są kolejne krawędzie z tego zbioru i sprawdzane jest czy wierzchołki, które łączy, należą do tego samego grafu. Jeśli nie to następuje połączenie grafów, których częścią są te dwa wierzchołki. Algorytm trwa tak długo aż zostaną sprawdzone wszystkie krawędzie ze zbioru. Złożoność tego algorytmu jest podobna do algorytmu Prima i wynosi ona $O(E * \log V)$.

- Wyznaczanie najkrótszej ścieżki w grafie
  1. Algorytm Dijkstry

     W algorytmie tym rozpatrywane są kolejne wierzchołki, rozpoczynając od wybranego wierzchołka startowego. Odległości wszystkich wierzchołków są inicjalizowane wartością nieskończoność, poza wierzchołkiem startowym, którego odległość wynosi 0. W kolejnych krokach podejmuje się próbę relaksacji sąsiednich wierzchołków, aktualnie rozpatrywanego wierzchołka (czyli nadpisania starej odległości, jeśli nowa byłaby krótsza). Kolejne wierzchołki do rozpatrzenia

są wybierane według ich odległości, począwszy od wierzchołków o najmniejszej odległości. Efektem końcowym algorytmu jest wyznaczona najmniejsza możliwa odległość od każdego wierzchołka z osobna, do wierzchołka startowego. Złożoność tego algorytmu wynosi $O(E * \log V)$ w wersji z kolejką lub $O(V^2)$ w wersji „naiwnej" ze zwykła tablicą. Warto zwrócić uwagę, że działanie tego algorytmu opiera się na założeniu, że wagi wszystkich krawędzi grafu są nieujemne.

2. Algorytm Bellmana-Forda

   W algorytmie tym wartości odległości wierzchołków są jednakowo inicjalizowane jak w algorytmie Dijkstry, tak samo również jest wykonywana metoda relaksacji wierzchołków, którą jednak dokonuje się na każdej z krawędzi $V - 1$ razy, bez wybierania konkretnego wierzchołka do rozpatrzenia. Algorytm ten jest pewnym uogólnieniem algorytmu Dijkstry. Mianowicie działa on również na wagach ujemnych z tym wyjątkiem, że nie może istnieć w grafie cykl ujemny osiągalny z wierzchołka startowego. W tym celu na końcu algorytmu podejmowana jest próba dodatkowej relaksacji wierzchołków. Jeżeli na tym etapie algorytmu pozostał jeszcze jakiś wierzchołek do relaksacji to znaczy, że w grafie został wykryty cykl ujemny i działanie algorytmu powinno zostać przerwane. Złożoność tego algorytmu wynosi $O(V * E)$.

2. Założenia projektowe

W napisanym programie przyjęto następujące założenia:

Działanie algorytmów na grafach zostało sprawdzone dla 7 różnych rozmiarów grafów oraz 3 różnych gęstości grafu:

- 10, 20, 40, 75, 150, 250, 500 wierzchołków
- 25%, 50%, 99% gęstości grafu

Ze względu na skrajne czasy wykonania algorytmu dla poszczególnych grafów przyjęto dodatkowe założenia co do liczby powtórzeń wykonania algorytmu:

- Dla 10, 20, 40 wierzchołkowych grafów została wyznaczona średnia z 1000 pomiarów
- Dla 75, 150 wierzchołkowych grafów została wyznaczona średnia z 200 pomiarów
- Dla 250, 500 wierzchołkowych grafów została wyznaczona średnia z 50 pomiarów

Dla każdego pomiaru generowany był nowy graf, nadpisujący poprzednią wersję.

Wagami wszystkich krawędzi w generowanych grafach są liczby całkowite w zakresie od 1 do 2000000.

Graf w postaci macierzy incydencji został zaimplementowany jako tablica dwuwymiarowa o wymiarach V x E, w której wartość 0 odpowiada za brak połączenia, 1 za początek krawędzi, -1 za koniec krawędzi (dla grafu skierowanego a dla grafu nieskierowanego również 1), 2 za krawędź będącą pętlą. Do tego uwzględniona została również tablica wag krawędzi o szerokości E, zawierająca wagi odpowiednich krawędzi.

Graf w postaci listowej został zaimplementowany jako tablica o długości V, zawierająca wszystkie wierzchołki. Każdy wierzchołek w tej reprezentacji posiada własną listę następników, przy czym każde z połączeń posiada informację o swojej wadze, wierzchołku źródłowym oraz wierzchołku docelowym.

Generowanie losowego grafu zostało oparte na stworzeniu najpierw pierścienia pomiędzy wszystkimi wierzchołkami. W pętli iterującej V razy, łączony jest wierzchołek o indeksie [x] z wierzchołkiem o indeksie [(x+1) modulo V]. Taki sposób początkowej generacji grafu zapewnia spójność zarówno dla grafu nieskierowanego jak i skierowanego. W następnej kolejności do grafu dokładane są kolejne krawędzie o losowo dobieranym wierzchołku startowym i wierzchołku końcowym z wyłączeniem krawędzi wielokrotnych oraz pętli.

3. Plan oraz przebieg eksperymentu

Przed wykonaniem pomiaru generowany był nowy graf zarówno w postaci macierzowej jak i listowej w oparciu o podane w punkcie 2 założenia projektowe. Następnie wybierany był odpowiedni algorytm do wykonania oraz liczba jego powtórzeń. Potem w pętli iterowanej odpowiednią ilość razy za każdym razem wywoływane zostało **std::chrono::high_resolution_clock::now()** bezpośrednio przed wykonaniem funkcji z algorytmem po czym zostało ponownie wywołane bezpośrednio po zakończeniu funkcji. Różnica wartości z obu tych wywołań zrzutowana na odpowiednią jednostkę czasu zwracała czas trwania wybranego algorytmu. Z uzyskanych wyników obliczona została średnia dla danej wielkości grafu oraz ich gęstości. Proces ten był powtarzany dla każdej rozpatrywanej wielkości grafu oraz każdego rozpatrywanego algorytmu grafowego.

4. Wyniki eksperymentu

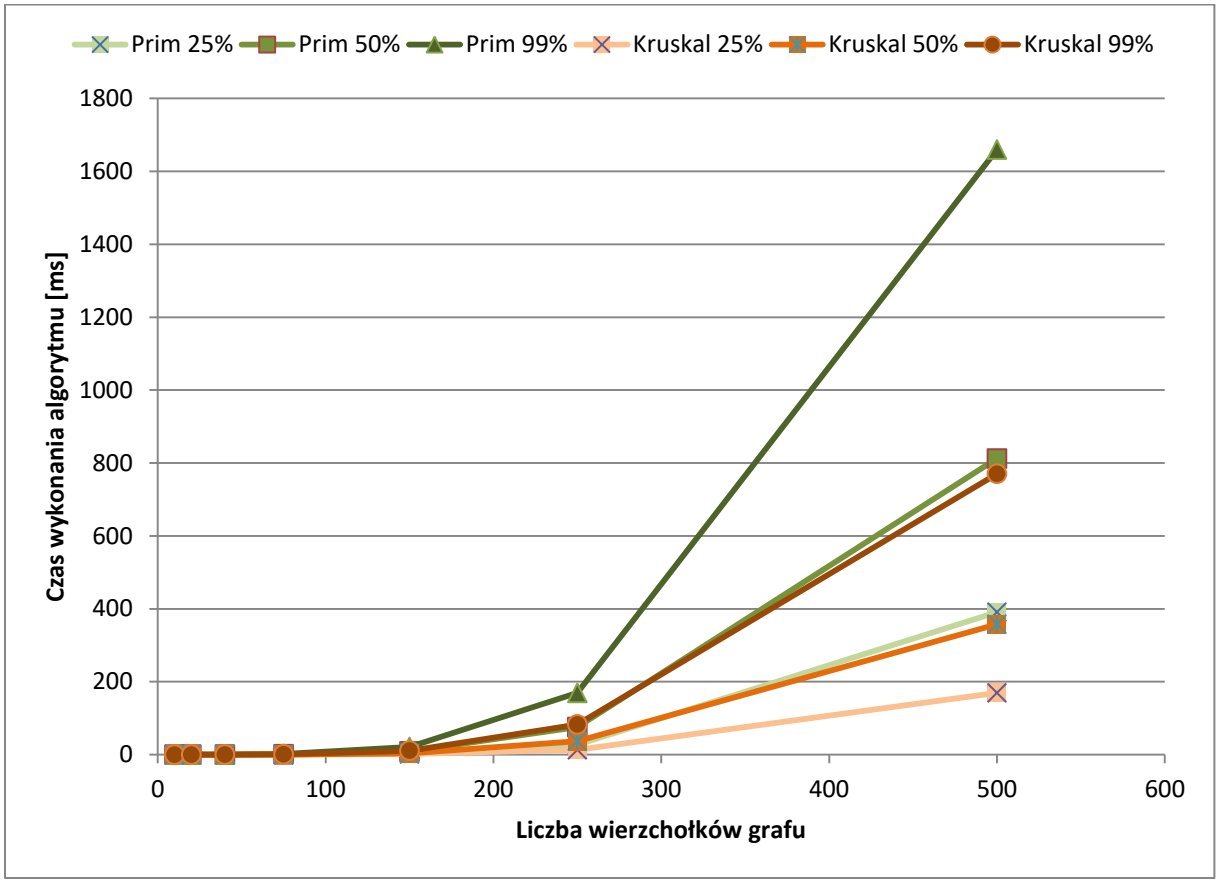
- Wyszukiwanie minimalnego drzewa rozpinającego (macierz incydencji)

	Liczba wierzchołków w grafie						
Gęstość grafu	10	20	40	75	150	250	500
25%	0,002	0,021	0,084	0,46	4,32	27,38	390,92
50%	0,0074	0,029	0,18	1,01	8,82	75,60	811,96
99%	0,0047	0,061	0,32	1,98	21,09	169,24	1659,62

Rys. 1 Średni czas wykonania algorytmu Prima na reprezentacji macierzowej [ms]

Gęstość grafu	Liczba wierzchołków w grafie						
	10	20	40	75	150	250	500
25%	0,002	0,0099	0,038	0,26	1,68	13,01	169,38
50%	0,0053	0,017	0,095	0,54	3,60	37,22	357,64
99%	0,0087	0,045	0,19	1,05	11,06	82,92	770,44

Rys. 2 Średni czas wykonania algorytmu Kruskala na reprezentacji macierzowej [ms]



Rys. 3 Porównanie czasów wykonania algorytmu Prima i Kruskala dla różnych gęstości grafu (reprezentacja macierzowa)

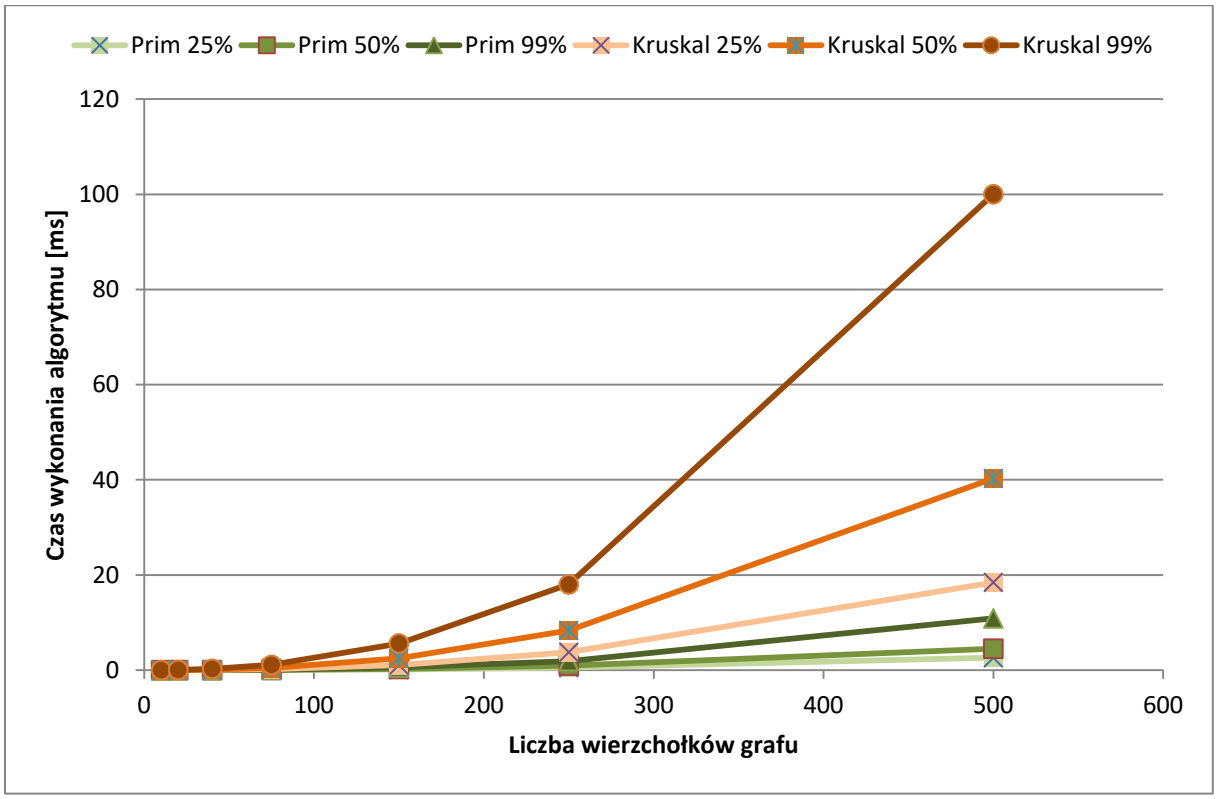
- Wyszukiwanie minimalnego drzewa rozpinającego (reprezentacja listowa)

	Liczba wierzchołków w grafie						
Gęstość grafu	10	20	40	75	150	250	500
25%	0,00066	0,0067	0,015	0,049	0,12	0,54	2,60
50%	0,0013	0,011	0,019	0,045	0,26	0,98	4,52
99%	0,0013	0,008	0,025	0,10	0,59	1,92	10,88

Rys. 4 Średni czas wykonania algorytmu Prima na reprezentacji listowej [ms]

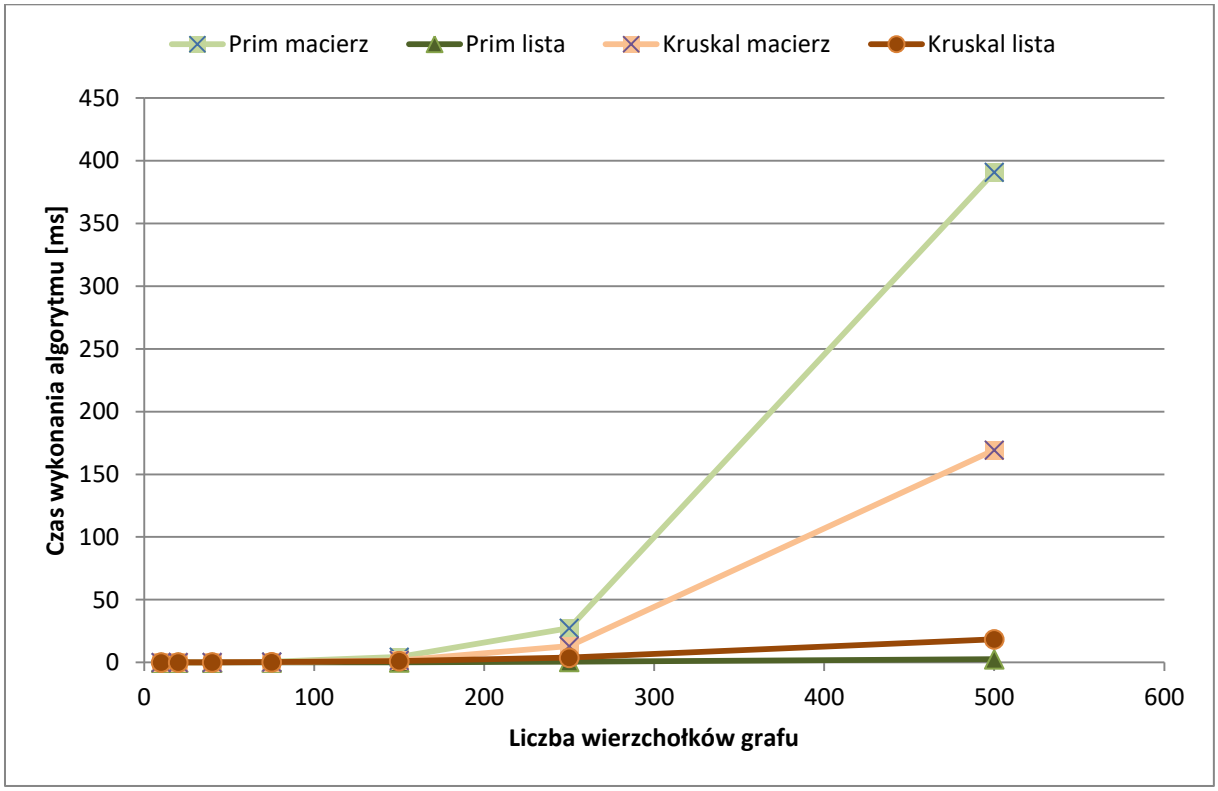
	Liczba wierzchołków w grafie						
Gęstość grafu	10	20	40	75	150	250	500
25%	0,0046	0,012	0,059	0,21	1,08	3,79	18,46
50%	0,0047	0,031	0,13	0,46	2,52	8,36	40,30
99%	0,0099	0,056	0,29	1,12	5,62	18,02	100,01

Rys. 5 Średni czas wykonania algorytmu Kruskala na reprezentacji listowej [ms]

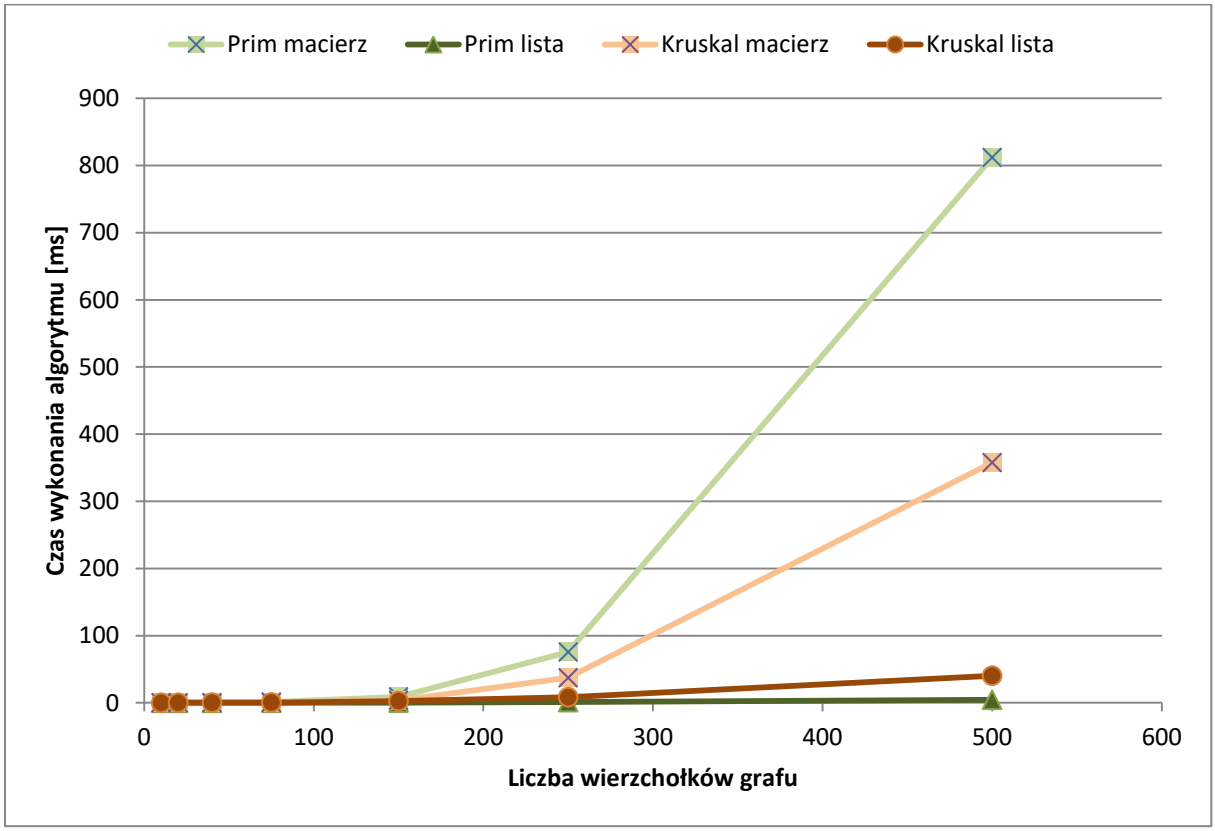


Rys. 6 Porównanie czasów wykonania algorytmu Prima i Kruskala dla różnych gęstości grafu (reprezentacja listowa)

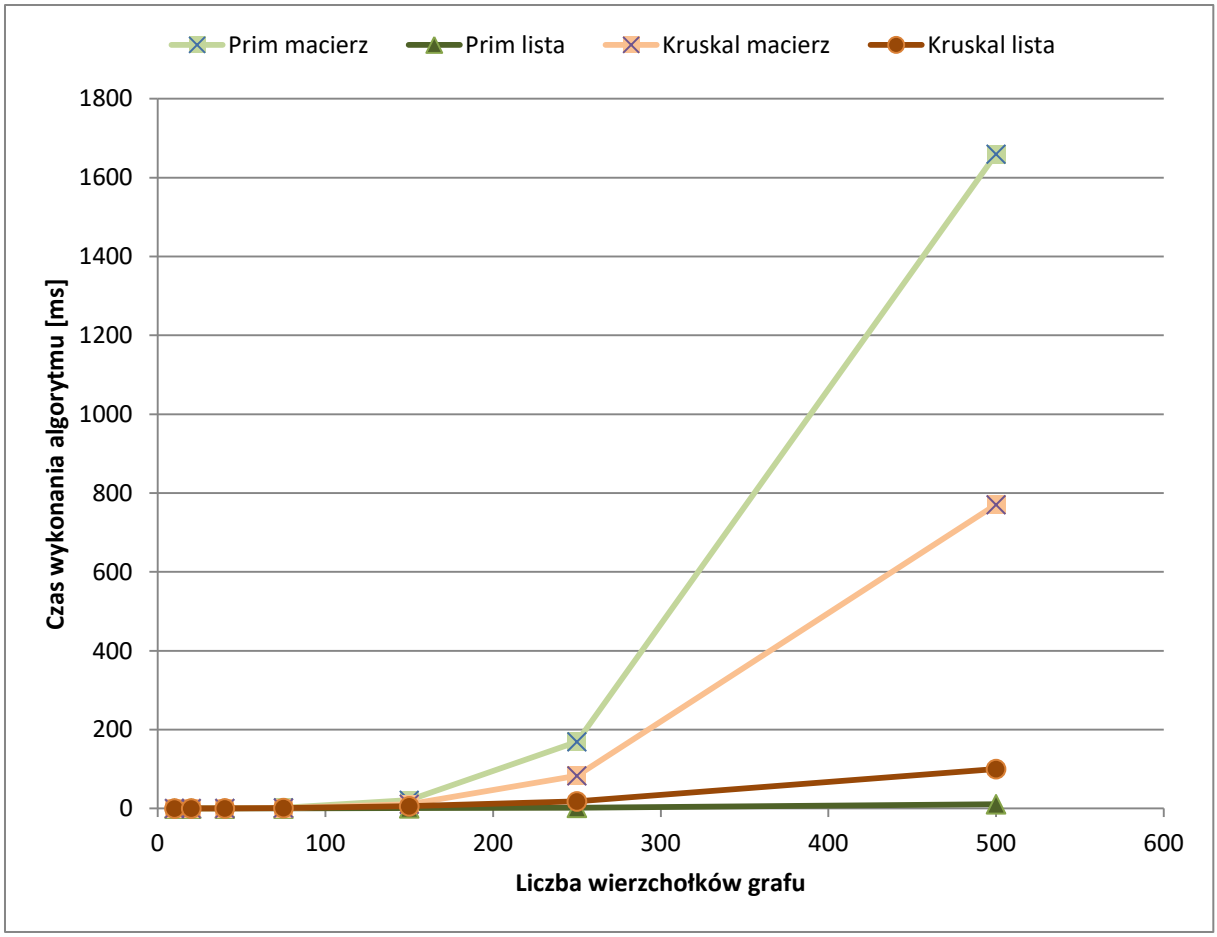
- Wyszukiwanie minimalnego drzewa rozpinającego (porównanie obu reprezentacji)



Rys. 7 Porównanie czasów wykonania algorytmu Prima i Kruskala dla różnych reprezentacji grafu (gęstość grafu 25%)



Rys. 8 Porównanie czasów wykonania algorytmu Prima i Kruskala dla różnych reprezentacji grafu (gęstość grafu 50%)



Rys. 9 Porównanie czasów wykonania algorytmu Prima i Kruskala dla różnych reprezentacji grafu (gęstość grafu 99%)

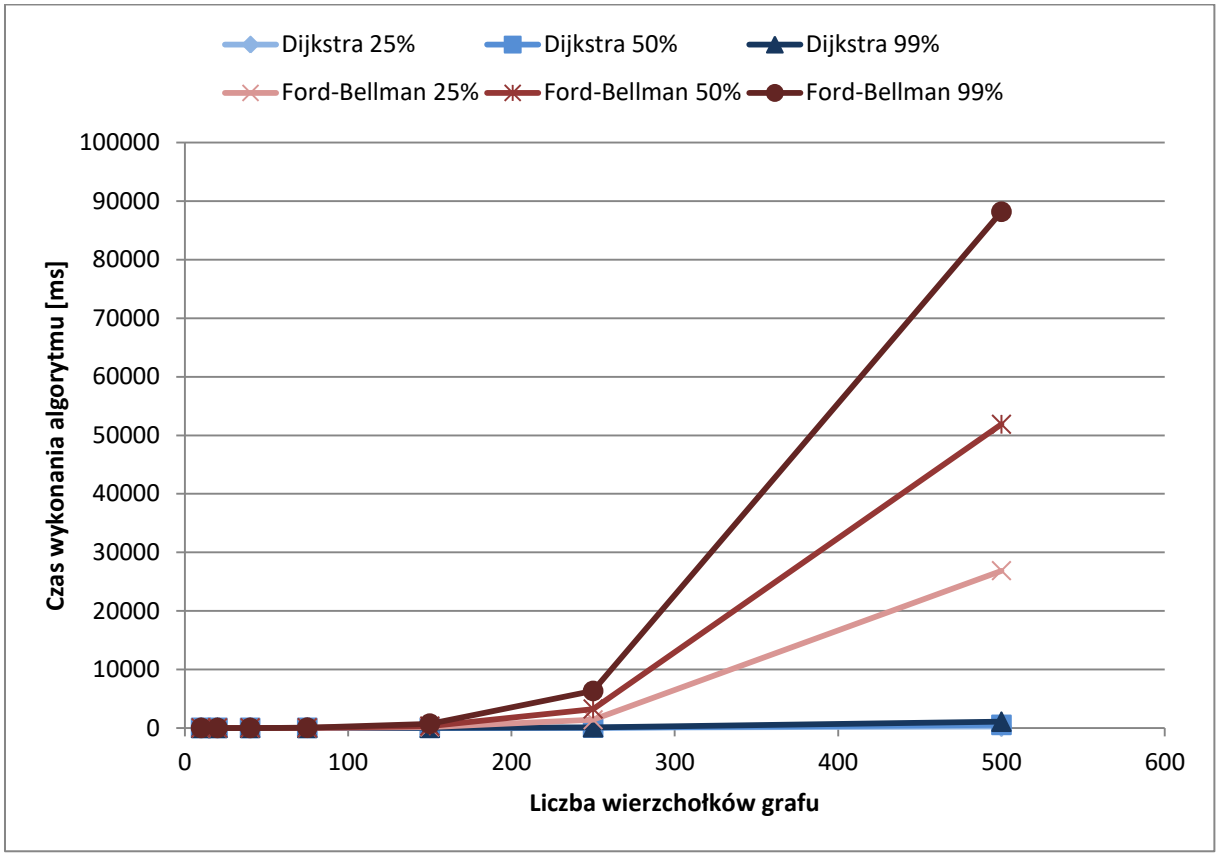
- Wyznaczanie najkrótszej ścieżki w grafie (macierz incydencji)

	Liczba wierzchołków w grafie						
Gęstość grafu	10	20	40	75	150	250	500
25%	0,0013	0,0087	0,055	0,32	2,75	22,97	253,01
50%	0,0053	0,018	0,099	0,66	5,76	57,04	512,94
99%	0,0067	0,029	0,19	1,29	17,79	119,82	1058,76

Rys. 10 Średni czas wykonania algorytmu Dijkstry na reprezentacji macierzowej [ms]

Gęstość grafu	Liczba wierzchołków w grafie						
	10	20	40	75	150	250	500
25%	0,0047	0,045	0,74	8,42	153,46	1382,16	26866,7
50%	0,0054	0,094	1,51	18,74	317,86	3228,1	51871,8
99%	0,0099	0,21	3,078	36,21	723,93	6310,68	88186,6

Rys. 11 Średni czas wykonania algorytmu Forda-Bellmana na reprezentacji macierzowej [ms]



Rys. 12 Porównanie czasów wykonania algorytmu Dijkstry i Ford'a-Bellman'a dla różnych gęstości grafu (reprezentacja macierzowa)

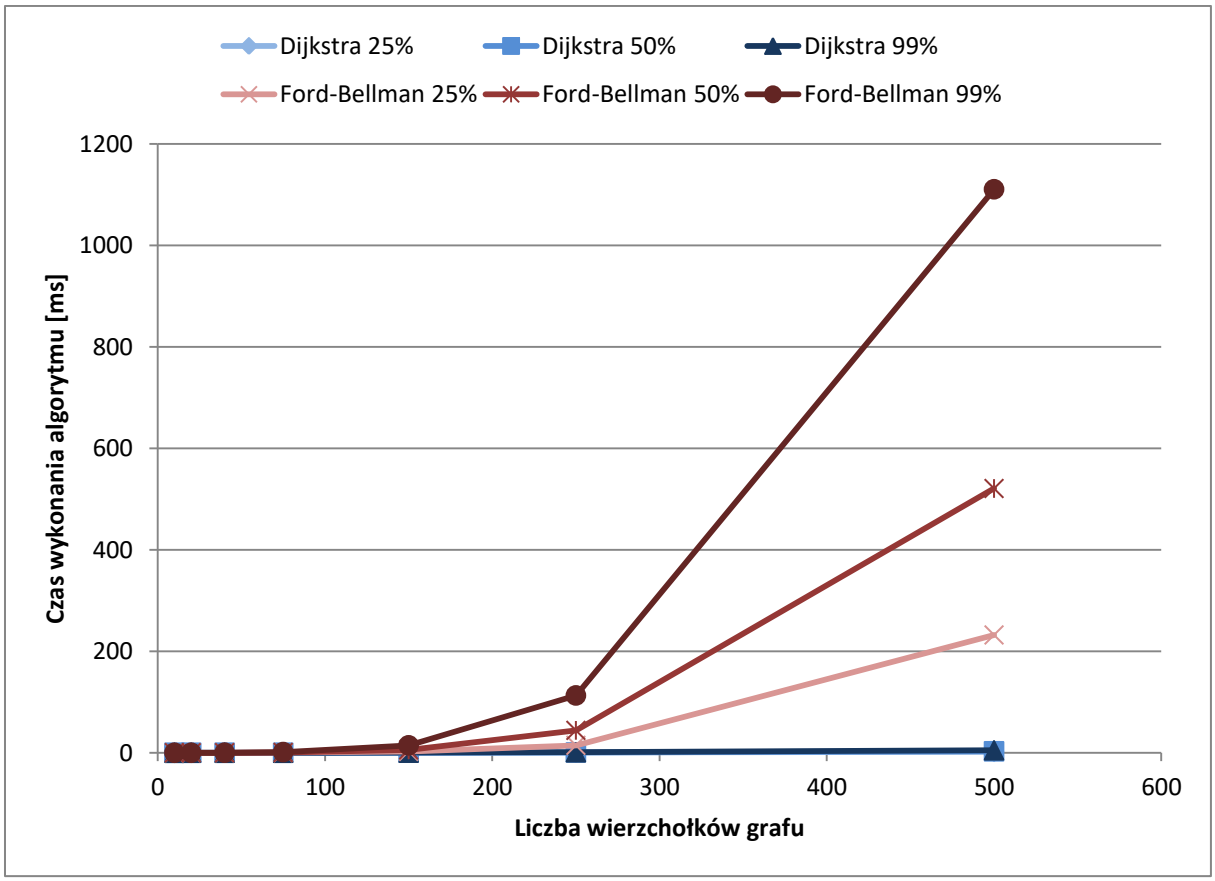
- Wyznaczanie najkrótszej ścieżki w grafie (reprezentacja listowa)

Gęstość grafu	Liczba wierzchołków w grafie						
	10	20	40	75	150	250	500
25%	0,00067	0,0081	0,013	0,025	0,14	0,44	1,94
50%	0,0013	0,0054	0,015	0,031	0,13	0,52	2,84
99%	0,0014	0,0041	0,017	0,035	0,26	0,98	4,76

Rys. 13 Średni czas wykonania algorytmu Dijkstry na reprezentacji listowej [ms]

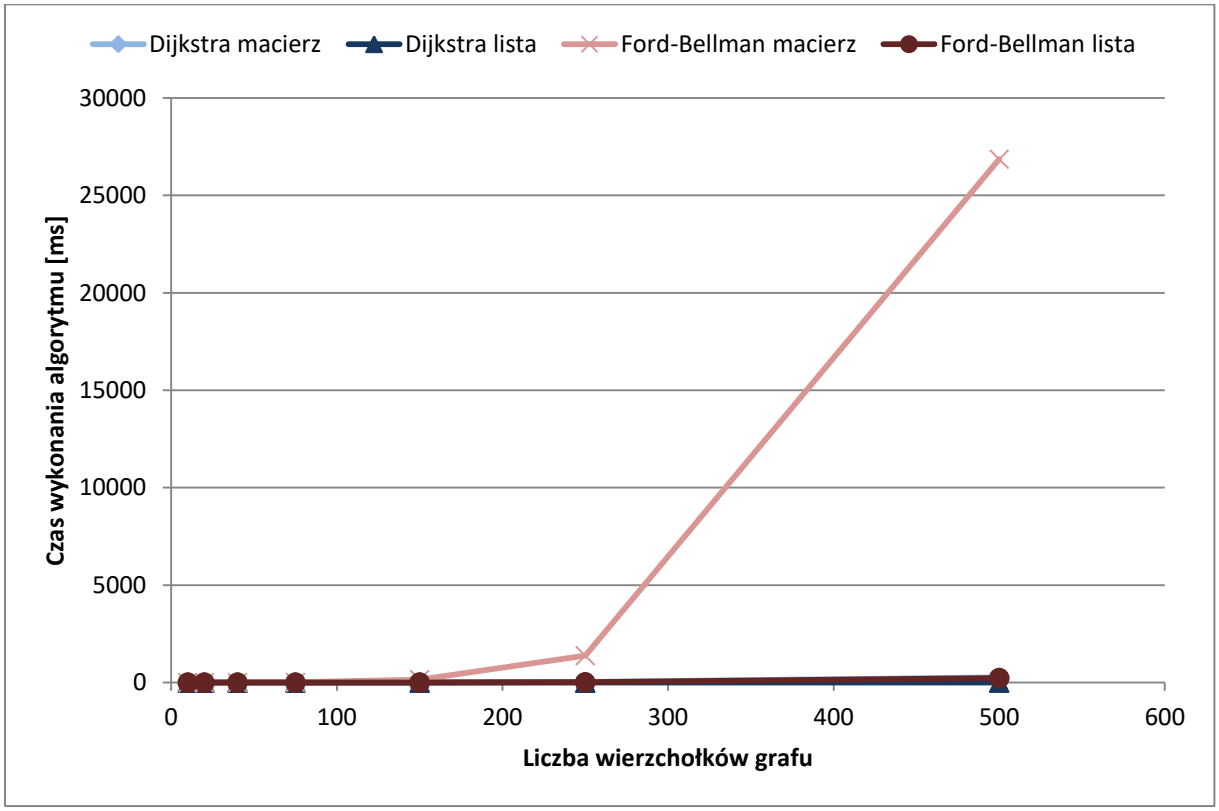
Gęstość grafu	Liczba wierzchołków w grafie						
	10	20	40	75	150	250	500
25%	0,0013	0,0093	0,037	0,23	2,62	14,62	232,24
50%	0,0019	0,0086	0,086	0,49	5,69	44,16	520,94
99%	0,0033	0,018	0,16	1,28	14,61	112,98	1110,88

Rys. 14 Średni czas wykonania algorytmu Ford'a-Bellman'a na reprezentacji listowej [ms]

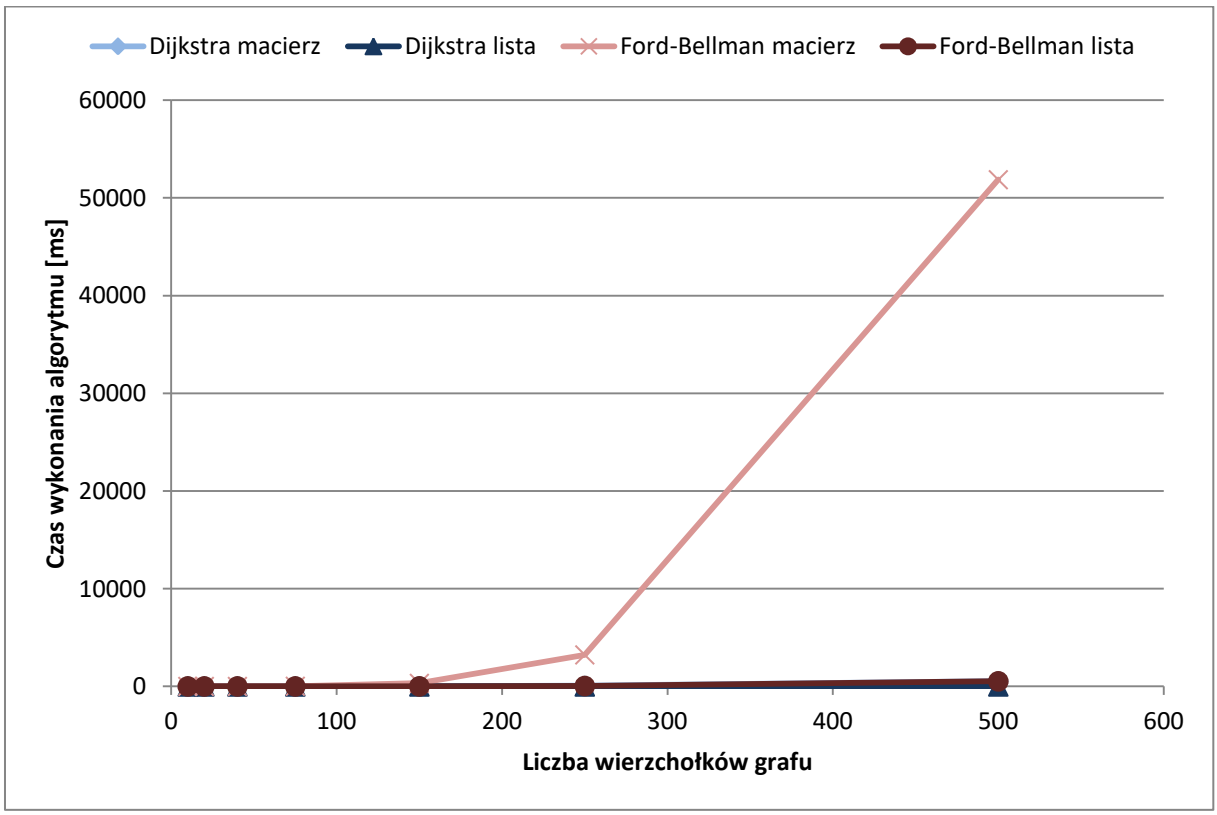


Rys. 15 Porównanie czasów wykonania algorytmu Dijkstry i Ford'a-Bellman'a dla różnych gęstości grafu (reprezentacja listowa)

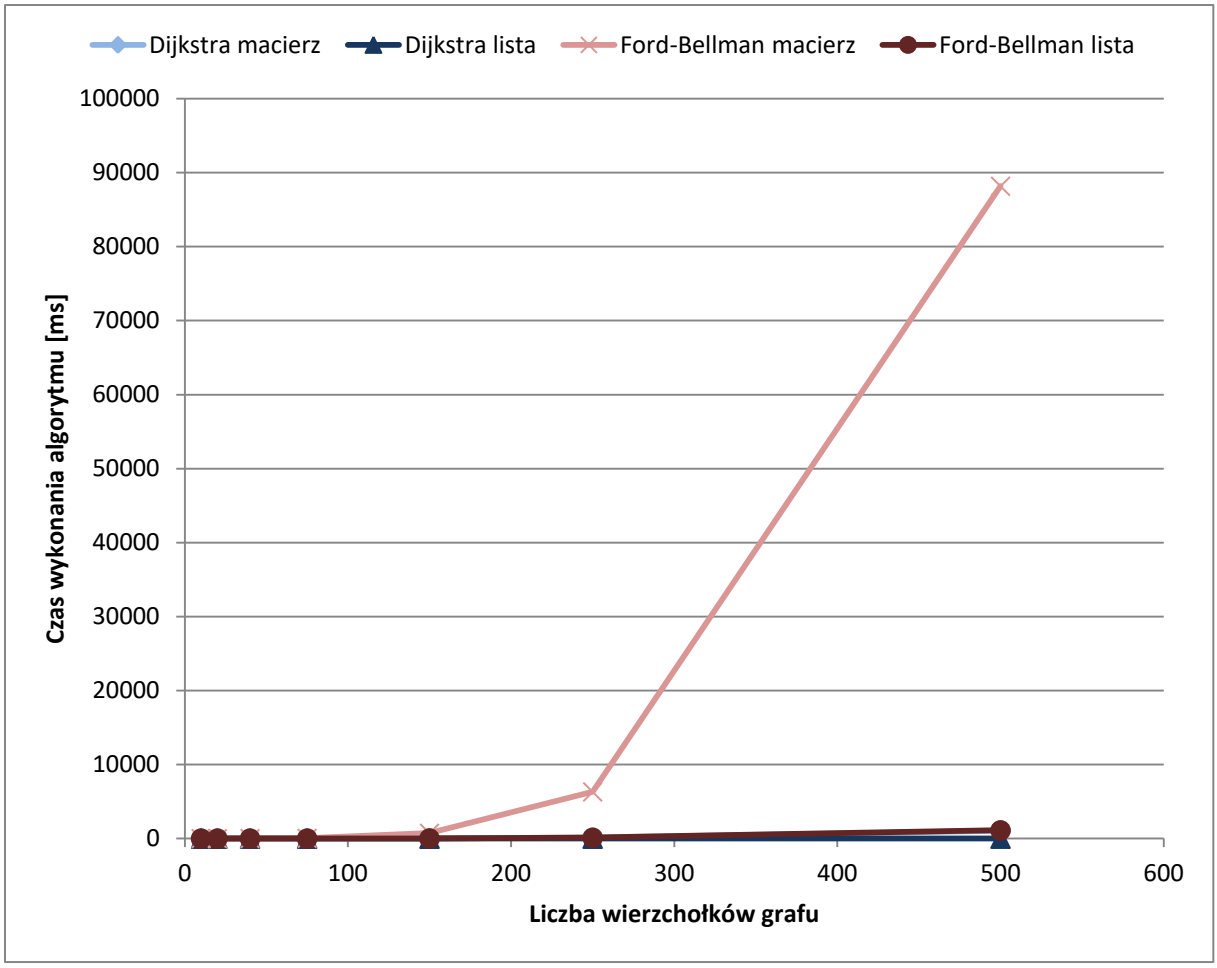
- Wyznaczanie najkrótszej ścieżki w grafie (porównanie obu reprezentacji)



Rys. 16 Porównanie czasów wykonania algorytmu Dijkstry i Ford'a-Bellman'a dla różnych reprezentacji grafu (gęstość grafu 25%)



Rys. 17 Porównanie czasów wykonania algorytmu Dijkstry i Ford'a-Bellman'a dla różnych reprezentacji grafu (gęstość grafu 50%)



Rys. 18 Porównanie czasów wykonania algorytmu Dijkstry i Ford'a-Bellman'a dla różnych reprezentacji grafu (gęstość grafu 99%)

5. Podsumowanie i wnioski

Na podstawie uzyskanych tabel oraz wykresów przedstawionych w powyższych wynikach, można dokonać następujących obserwacji oraz uzyskać wnioski:

- Teoretyczne złożoności obliczeniowe algorytmów Prima i Kruskala są podobne na poziomie $O(E * \log V)$, co jednak nie ma tak oczywistego pokrycia w uzyskanych wynikach. Dla prawie każdej serii pomiarowej uzyskana została następująca kolejność algorytmów pod względem czasu ich trwania (zaczynając od najgorszego przypadku):
  - Algorytm Prima (macierz)
  - Algorytm Kruskala (macierz)
  - Algorytm Kruskala (lista)
  - Algorytm Prima (lista)

  Wskazuje to na zależność czasu trwania algorytmu nie tylko od sposobu reprezentacji grafu, ale również od sposobu implementacji konkretnych algorytmów dla danej reprezentacji.

- W przypadku algorytmów wyznaczania najkrótszej ścieżki w grafie wyniki w przybliżeniu pokrywają się z teoretycznymi złożonościami obliczeniowymi tych algorytmów. Zdecydowanie szybszym okazał się algorytm Dijkstry o złożoności obliczeniowej $O(V^2)$ [nie zastosowano kolejki]. Dla porównania algorytm Ford'a-Bellman'a charakteryzuje się złożonością obliczeniową $O(V * E)$. Dla grafu pełnego $E = V(V-1)$, więc algorytm Ford'a-Bellman'a powinien trwać dłużej aż $V-1$ razy. Zauważona dysproporcja pomiędzy sprawdzonymi algorytmami nie okazała się aż tak duża, jednak wciąż algorytm Ford'a-Bellman'a był wielokrotnie wolniejszy od algorytmu Dijkstry. Warto również zauważyć, że tak samo jak w przypadku problemu wyznaczania minimalnego drzewa rozpinającego zauważona została znaczna przewaga reprezentacji listowej nad reprezentacją w postaci macierzy incydencji pod względem czasowym. Wynika to z faktu, że w reprezentacji listowej uwzględnione są prawie wyłącznie istniejące połączenia i tylko one są przeszukiwane w algorytmach na tej reprezentacji. W macierzy incydencji natomiast przeważnie większość pól przyjmuje wartość 0 oznaczającą brak połączenia, co spowalnia przeszukiwanie macierzy w poszukiwaniu odpowiedniej krawędzi, zwłaszcza dla dużych grafów.

6. Literatura

Materiały do ćwiczeń dr inż. Jarosława Mierzwy
https://pl.wikipedia.org/wiki/Algorytm_Prima
https://pl.wikipedia.org/wiki/Algorytm_Kruskala
https://pl.wikipedia.org/wiki/Algorytm_Dijkstry
https://pl.wikipedia.org/wiki/Algorytm_Bellmana-Forda
c++ - How to convert std::chrono::high_resolution_clock::now() to milliseconds, microseconds, ...? - Stack Overflow