Struktury Danych i Złożoność Obliczeniowa

Paul Paczyński

Nr albumu: 254307

21.06.2022

Zadanie projektowe nr 2:

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

Prowadzący kurs: Dr inż. Dariusz Banasiak

Termin zajęć: Piątek 11:15 TN

Termin oddania zadania 21.06.2022

Spis treści

1. Wstęp

Zadaniem projektowym było napisanie programu i zmierzenie czasu wykonywania odpowiednich algorytmów grafowych dla grafów reprezentowanych poprzez macierz incydencji i listę sąsiedztwa.

Opracowane algorytmy:

- Wyznaczanie minimalnego drzewa rozpinającego
    1. Algorytm Kruskala
    2. Algorytm Prima
- Wyznaczenie najkrótszej ścieżki w grafie
    1. Algorytm Dijkstry
    2. Algorytm Bellmana-Forda

2. Złożoności obliczeniowe

Przy analizie złożoności obliczeniowej algorytmów grafowych wykorzystuje się dwie zmienne które są zależne od podanego grafu.

- **E** – liczba krawędzi grafu
- **V** – liczba wierzchołków grafu

2.1 Złożoność algorytmu Kruskala

Złożoność obliczeniowa jest zawsze równa:

O(E log(V))

2.2 Złożoność algorytmu Prima

Dla algorytmu Prima złożoność może być różna, w zależności od implementacji. W projekcie użyłem implementacje kolejki stworzonej z kopca z poprzedniego etapu projektu, otrzymując złożoność:

O(E log(V))

2.3 Złożoność algorytmu Dijkstry

Tak samo jak w przypadku algorytmu Prima, o złożoności decyduje implementacja kolejki. W tym przypadku też wynosi:

O(E log(V))

2.4 Złożoność algorytmu Bellmana-Forda

Dla każdego grafu złożoność będzie równa:

O(EV)

3. Plan eksperymentu

W programie zostały zaimplementowane wszystkie obowiązkowe reprezentacje grafu, oraz po dwa algorytmy znajdywania MST, oraz najkrótszej ścieżki dla obu reprezentacji grafu. Program posiada menu konsolowe pozwalające na:

- Wczytanie grafu z pliku
- Wyświetlenie grafu w konsoli
- Losowe zbudowanie grafu na podstawie podanych parametrów
- Przetestowanie wszystkich możliwych algorytmów
- Automatyczne zmierzenie czasu wykonania dla wszystkich algorytmów

Wyniki przedstawione w tym dokumencie zostały wygenerowane następująco:

1. Każdy algorytm był testowany dla następującej liczbie wierzchołków: {50,100,150,200,250}
2. Dla każdej wielkości grafu testowano również jego gęstość: {25,50,75,99}
3. Każdy wynik uśredniano z wyniku 50 przebiegów pętli dla różnych danych

4. Metoda generowania losowego grafu

Podczas implementacji metody generowania losowego grafu musiałem przede wszystkim zapewnić spójność danego grafu. Poniżej opiszę w krokach zamysł działania mojego generatora:

1. Wczytanie parametrów wejściowych – liczba wierzchołków i gęstość
2. Obliczenie ilości krawędzi i rezerwacja wystarczająco szerokiej tablicy
3. Wykonanie pętli która połączy wszystkie wierzchołki niezbędnymi do zachowania spójności krawędziami.
4. Rozlosowanie pozostałych możliwych krawędzi i wpisanie ich w strukturę grafu

5. Wyniki pomiarów

Wszystkie wartości podane są w mikrosekundach.

5.1 Algorytm Kruskala

Implementacja macierzowa

	Gęstość			
Wierzchołki	25	50	75	99
50	1798,3	3597,75	6049,25	8997,6
100	14049,1	41723,05	75490,6	121943,8
150	57547,7	210364,5	384758,2	617485,7
200	159998,5	575343,5	1157300	3693119
250	558322,4	1385516	1750764	3139771

Implementacja listowa

	Gęstość			
Wierzchołki	25	50	75	99
50	451,799	1100,45	1699,2	2650,949
100	3497,299	10145,75	18838,1	30461,8
150	13390,6	43526,95	87929,75	150410,8
200	37830,05	130305,9	284774,1	474829,7
250	89778,65	320731,4	678298,2	1228413

5.2 Algorytm Prima

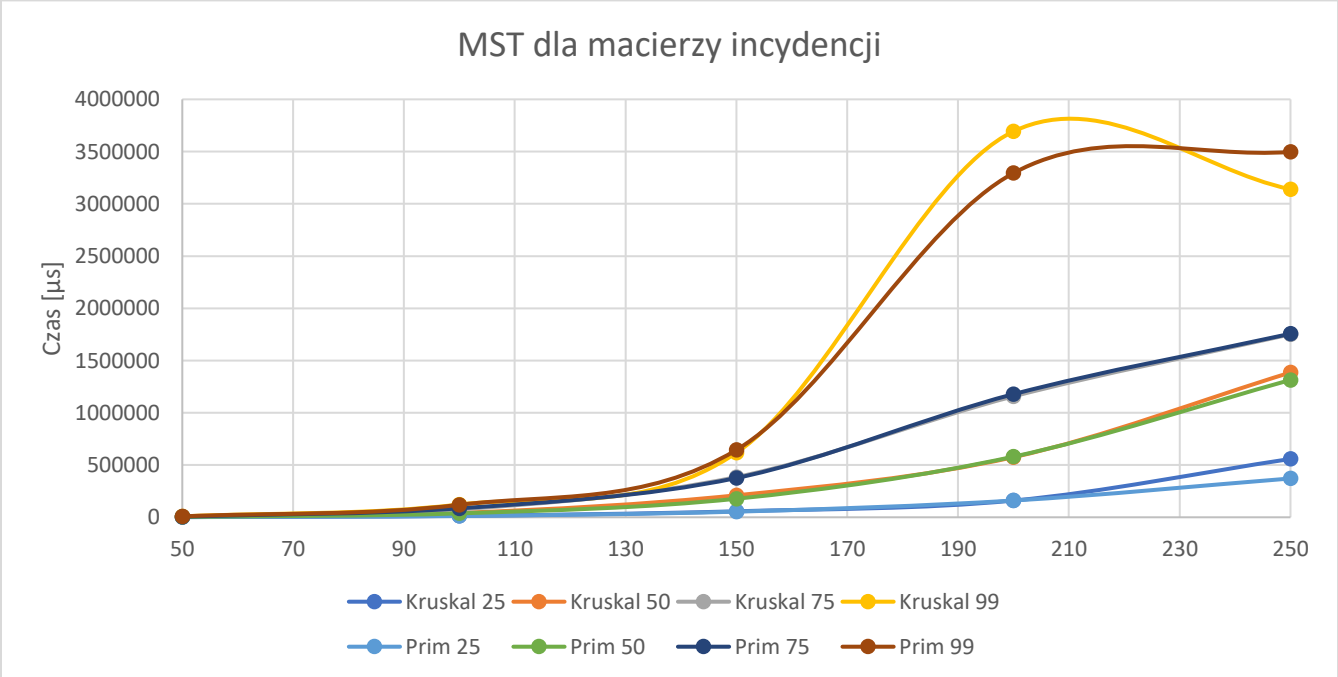
Implementacja macierzowa

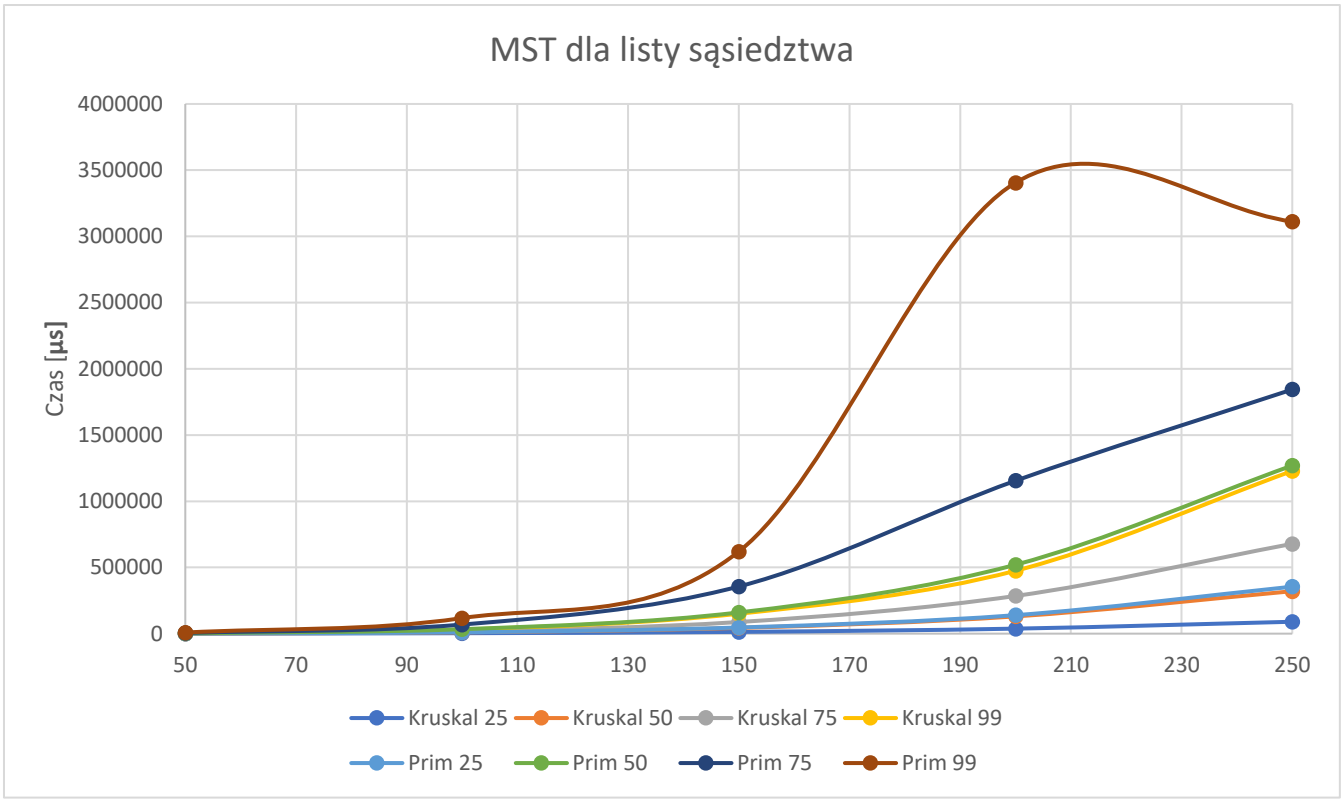
	Gęstość			
Wierzchołki	25	50	75	99
50	1549,15	3350,349	5847,95	8543,5
100	12546,05	37511,65	85690,8	119456,8
150	53875	177397,6	375961,4	644544,3
200	160354,1	580236,1	1178367	3293911
250	371598,1	1314328	1756358	3497398

Implementacja listowa

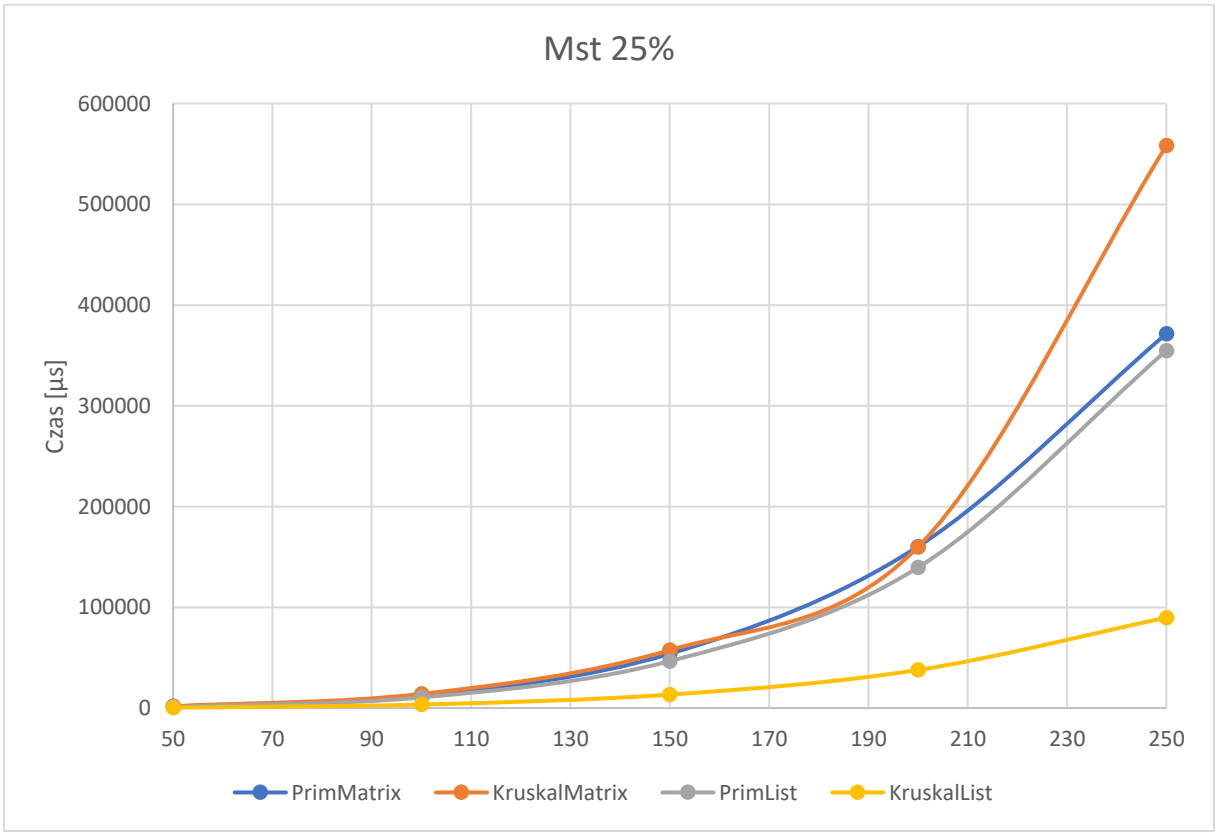
	Gęstość			
Wierzchołki	25	50	75	99
50	1098,45	2450,549	5346,9	8333,55
100	10575,4	33707,6	68722	117571,2
150	46601,75	160236,4	355962,7	619917
200	139557,5	520386,6	1154779	3405234
250	354835,7	1269582	1843703	3111661

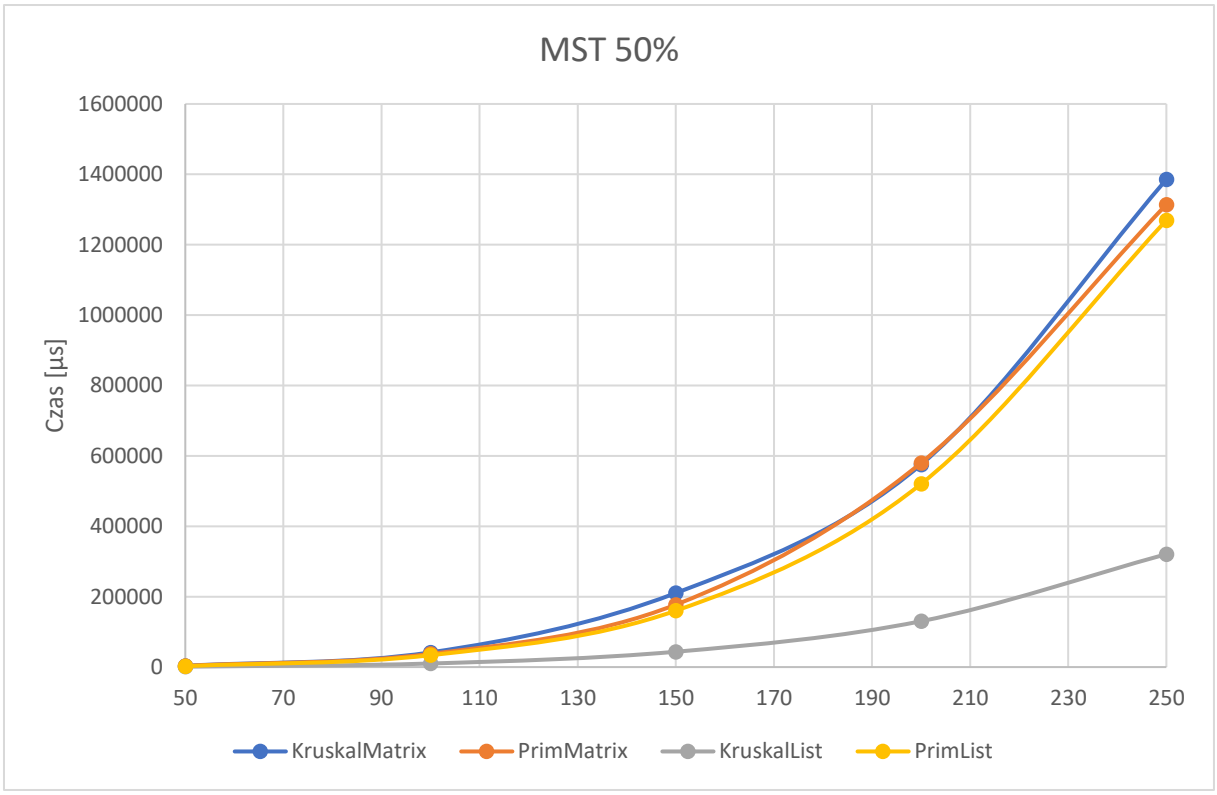
5.3.1 Wykresy typ 1

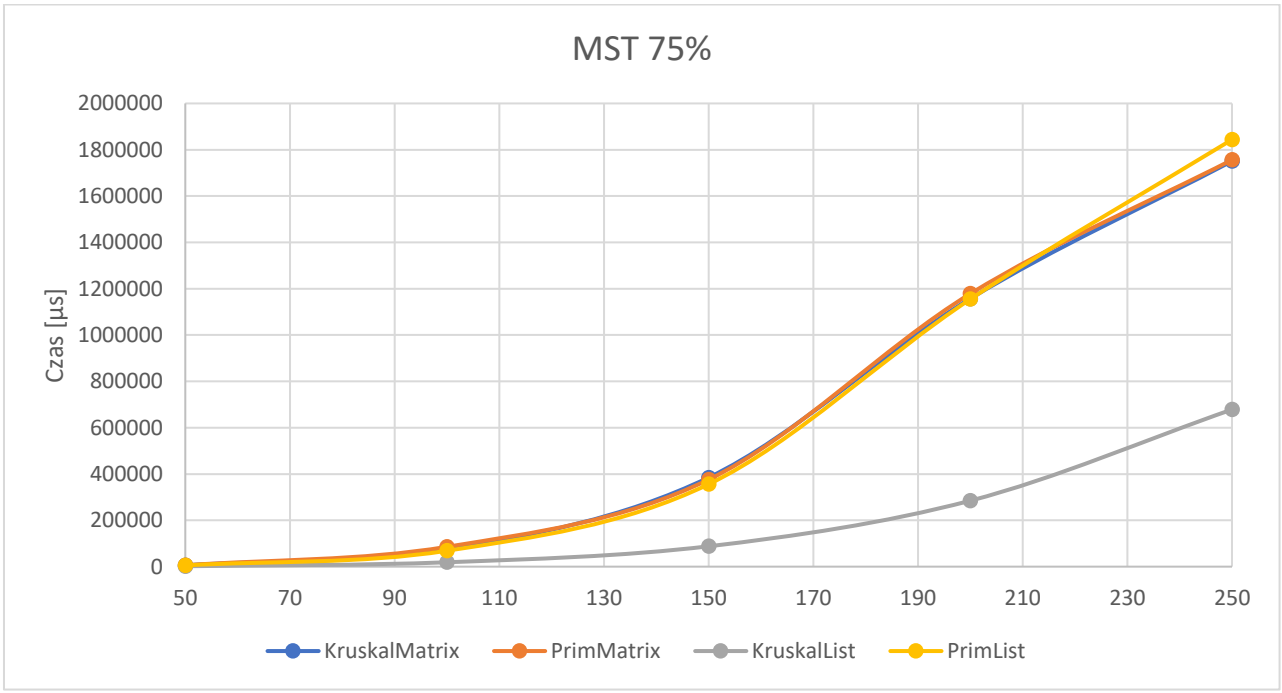


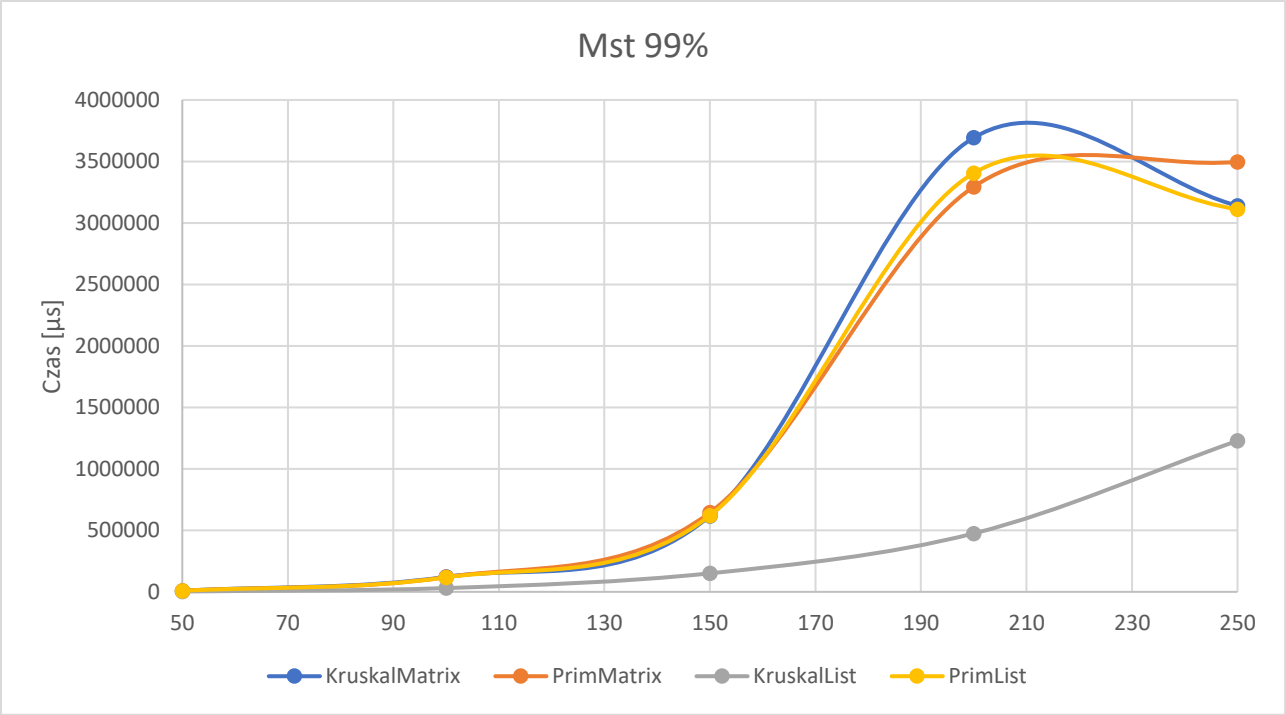


5.3.2 Wykresy typ 2




MST 50%
Czas [µs]
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
50
70
90
110
130
150
170
190
210
230
250
KruskalMatrix
PrimMatrix
KruskalList
PrimList


MST 75%
Czas [µs]
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
50
70
90
110
130
150
170
190
210
230
250
KruskalMatrix
PrimMatrix
KruskalList
PrimList



5.4 Algorytm Dijkstry

Implementacja macierzowa

	Gęstość			
Wierzchołki	25	50	75	99
50	149,949	349,799	699,549	1098,7
100	649,549	1199,9	1945,15	3947,7
150	1696,249	2447,85	3947,5	6445,5
200	2584	4238,2	6525,7	8297,35
250	4048,65	6646,1	10094,15	13093,45

Implementacja listowa

	Gęstość			
Wierzchołki	25	50	75	99
50	151,499	349,899	500,9	700,65
100	599,15	1048,05	1600,649	1983,449
150	899,599	1949,95	3400,85	4447,8
200	1847,8	3501,85	5795,7	7643,699
250	2746,85	5147,15	8385,999	11991

5.5 Algorytm Bellmana-Forda

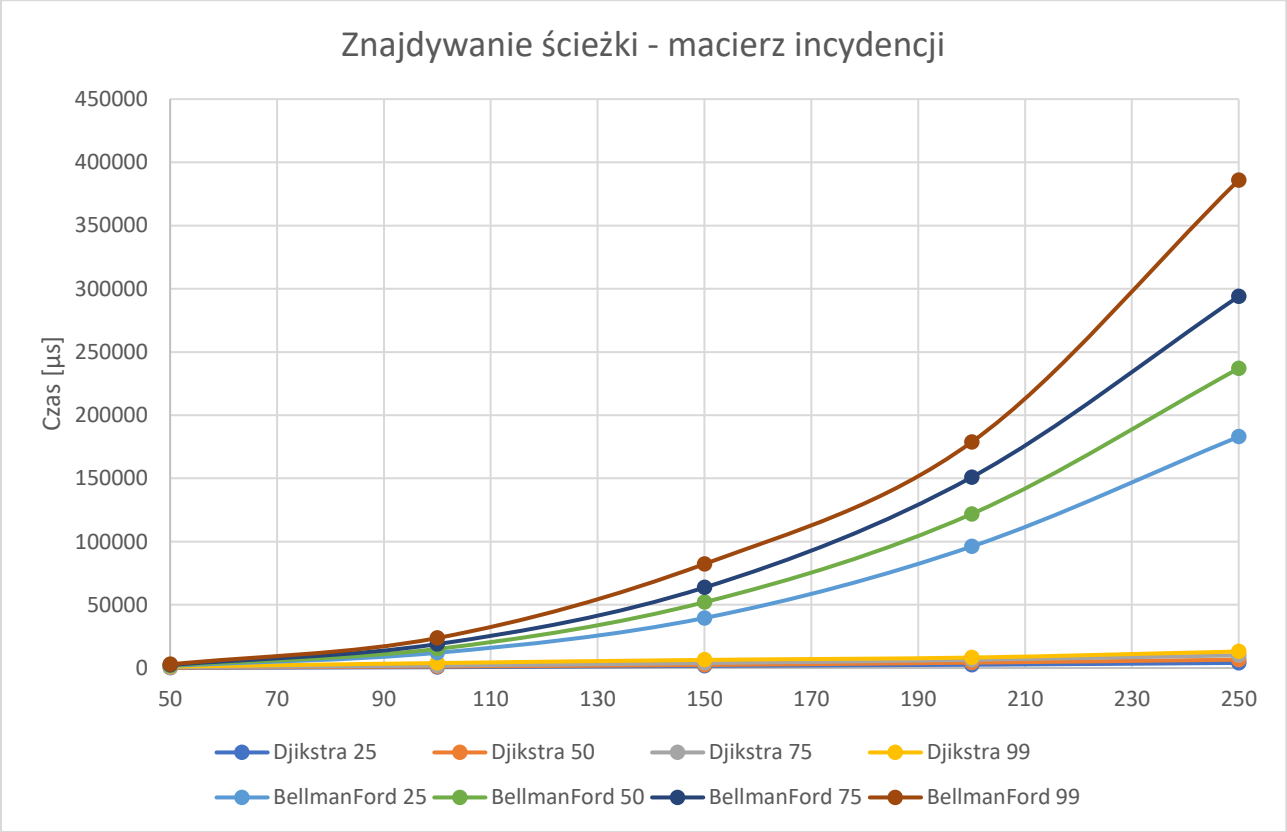
Implementacja macierzowa

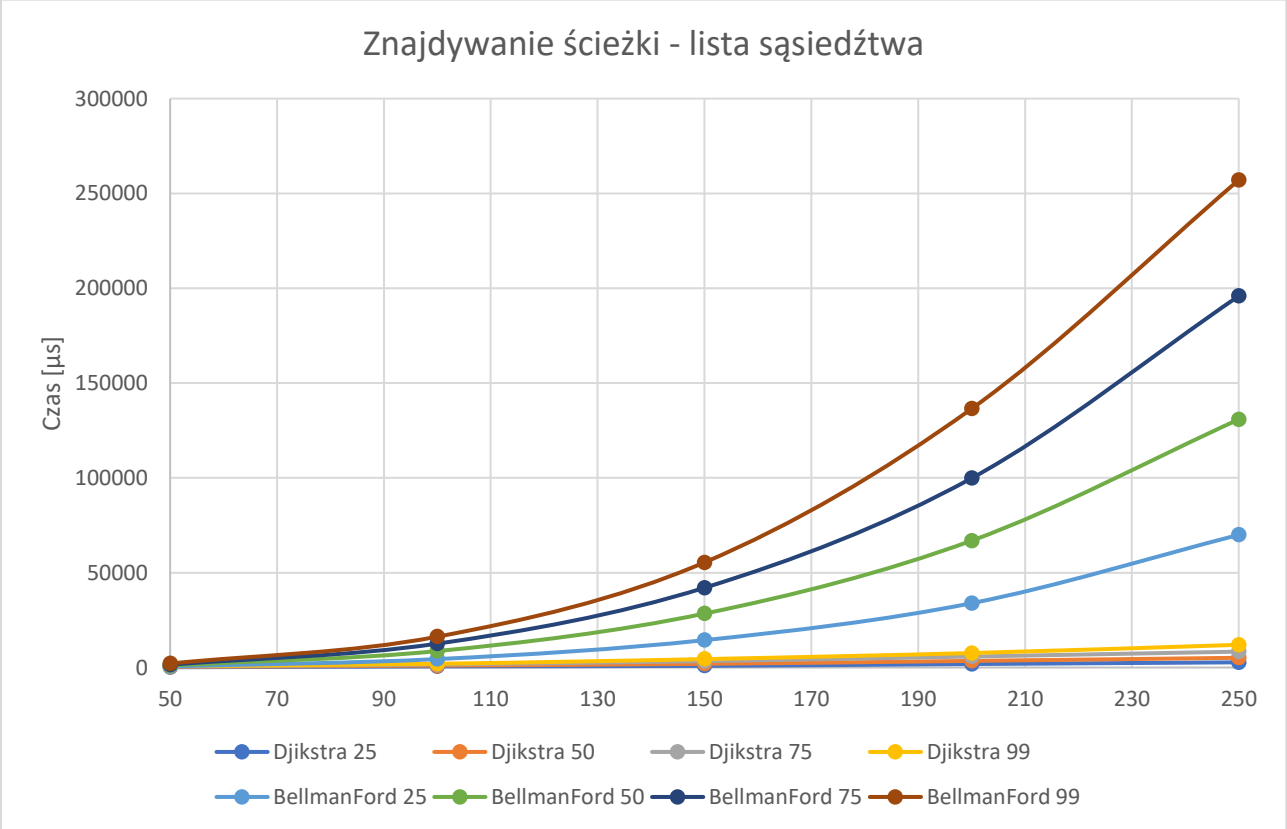
	Gęstość			
Wierzchołki	25	50	75	99
50	1743,75	2049,6	2398,7	2894,3
100	12043,65	15118,65	18938,1	23761,3
150	39564,55	52079,6	63683,8	82302,4
200	96098,25	121820,9	150879,9	178658,6
250	182977,6	236968,6	293985	385930,9

Implementacja listowa

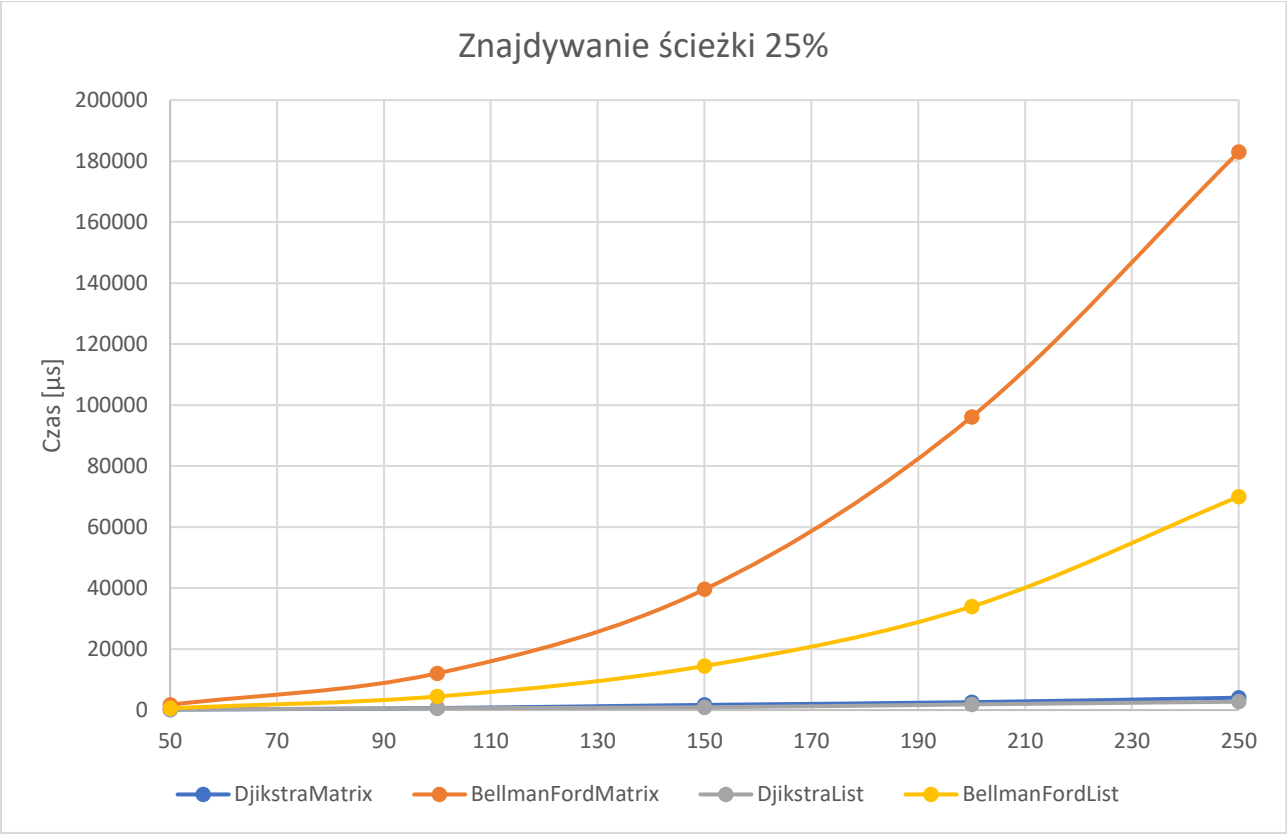
	Gęstość			
Wierzchołki	25	50	75	99
50	600,199	1098,95	1497,799	2099,1
100	4447,6	8691,85	12592,35	16271,1
150	14441,2	28431,15	42052,7	55411,95
200	33924,9	66893,45	99817	136453,6
250	69962,35	130848,2	195899	257073,7

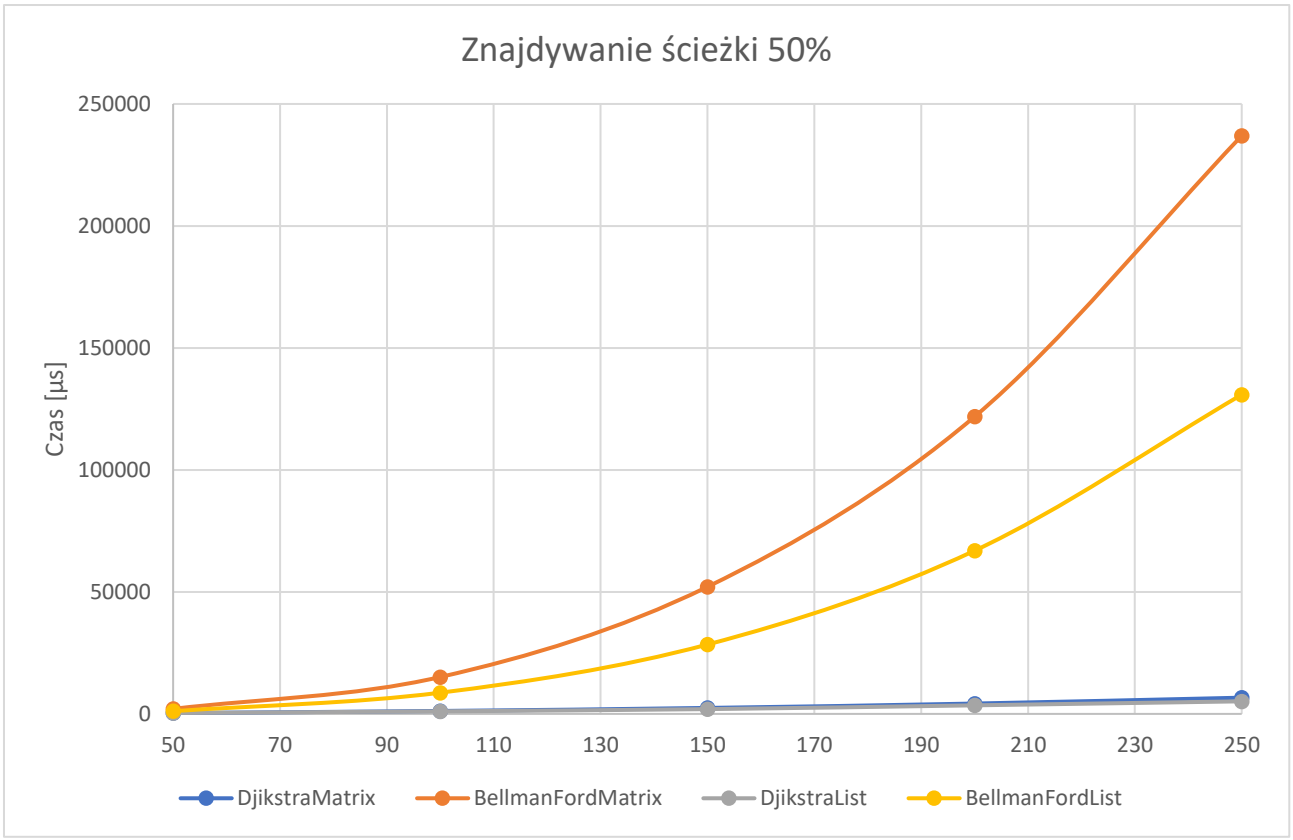
5.6.1 Wykresy typu 1

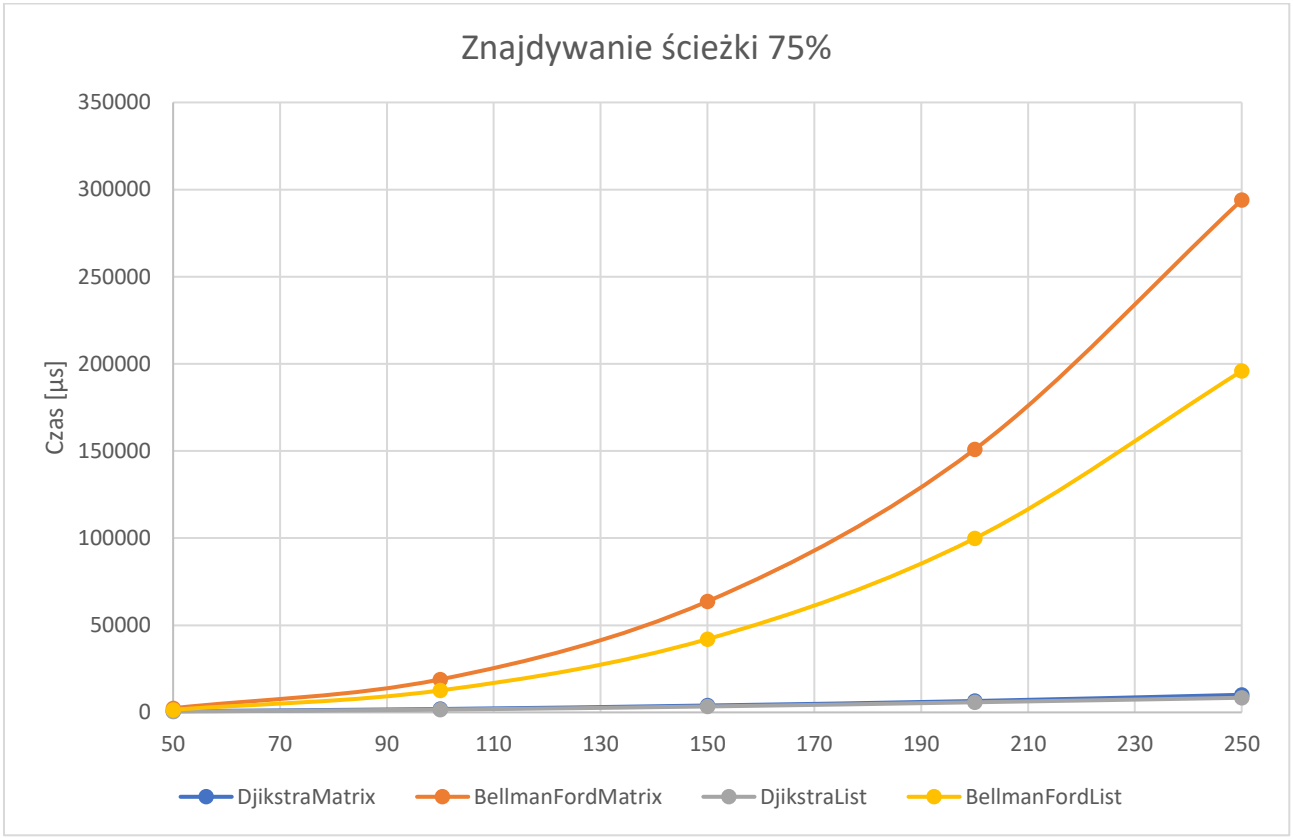


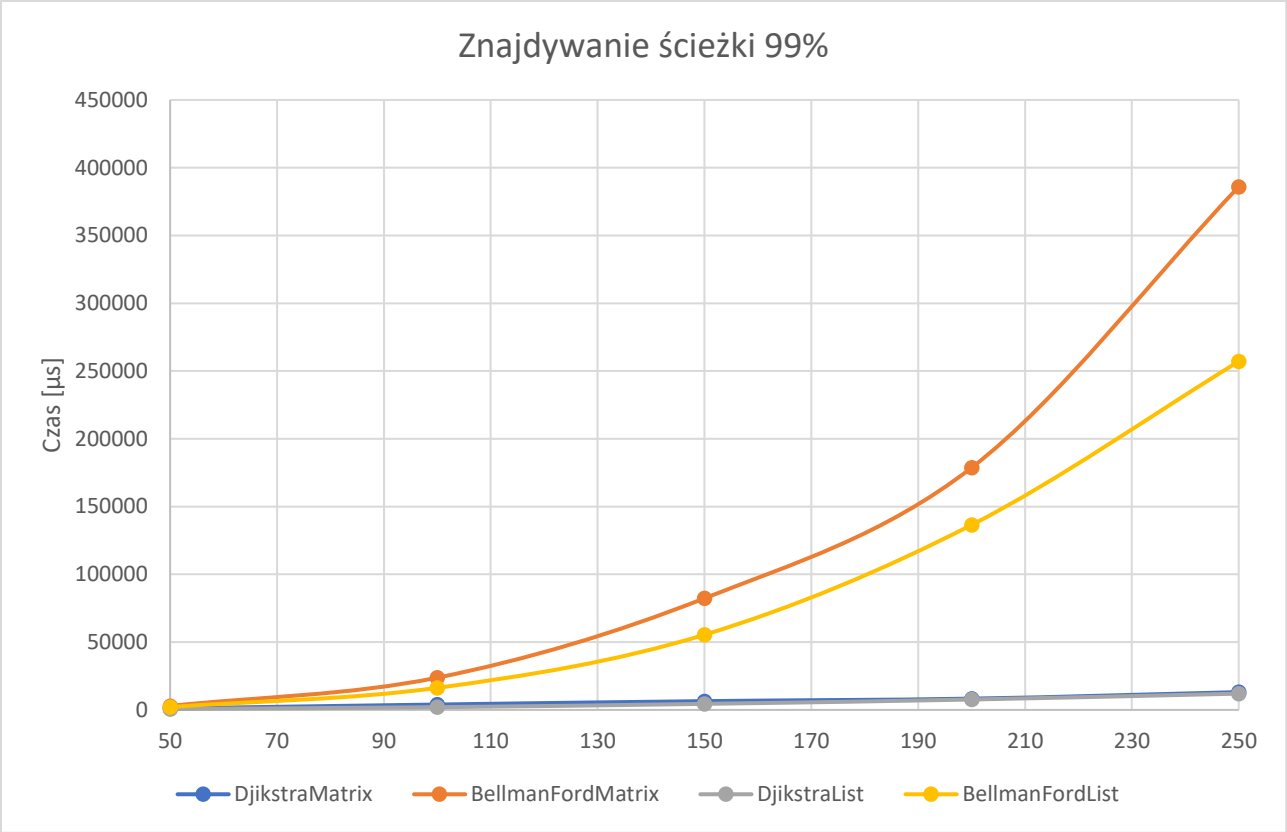


5.6.2 Wykresy typu 2




Znajdywanie ścieżki 50%
Czas [µs]
250000
200000
150000
100000
50000
0
50
70
90
110
130
150
170
190
210
230
250
DjikstraMatrix
BellmanFordMatrix
DjikstraList
BellmanFordList


Znajdywanie ścieżki 75%
Czas [µs]
350000
300000
250000
200000
150000
100000
50000
0
50
70
90
110
130
150
170
190
210
230
250
DjikstraMatrix
BellmanFordMatrix
DjikstraList
BellmanFordList



6. Wnioski

Problem MST

W większości przypadków struktura grafu miała wpływ na działanie algorytmu. W przedstawionych wynikach algorytm Kruskala często był lepszy od algorytmu Prima. Listowa reprezentacja grafu otrzymywała znacznie lepsze efekty, głównie przez sam fakt długości iteracji jaką trzeba wykonać, aby sprawdzić wszystkie zera znajdujące się w grafie opartym na macierzy, których w liście sąsiedztwa nie ma. Algorytm Kruskala powinien zostać zaimplementowany znacznie lepiej, traciłem bardzo dużo czasu na niepotrzebne iteracje fakt, że obie struktury przetrzymywały poszczególne krawędzie podwójnie.

Problem znajdywania ścieżki

Tak jak w przypadku algorytmów MST wybór listy sąsiedztwa był lepszym i szybszym rozwiązaniem. Dodatkowo jak można łatwo zauważyć algorytmy Dijkstry sprawdzały się znacznie lepiej, mimo implementacji algorytmu Bellmana-Forda wyłącznie dla liczb dodatnich. Dzieje się tak dlatego, że algorytm Bellmana-Forda potrafi radzić sobie z wagami ujemnymi, gdzie Dijkstra nie daje rady. Taką cechę otrzymujemy poprzez utratę wydajności, gdyż w algorytmie Bellmana-Forda musimy zrobić tyle iteracji pętli co mamy wierzchołków, oraz dodatkowy jeden aby sprawdzić istnienie ujemnego cyklu. W mojej implementacji nie zastosowałem ostatniego obiegu tej pętli ze względu na możliwość posiadania tylko wartości dodatnich przez krawędzie.