Jan Woźniak 234995
11.06.2018 r.

SPRAWOZDANIE

Struktury danych i złożoność obliczeniowa

Zadanie projektowe nr 2:

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

1. Informacje teoretyczne.

Graf może być reprezentowany w pamięci komputera na kilka sposobów. W tych badaniach wykorzystałem dwa z nich: listę następników i macierz incydencji.

Oznaczenia:

V – ilość wierzchołków, E – ilość krawędzi

Lista następników:

Do reprezentacji grafu w postaci listy następników wykorzystuje się tablicę o liczbie elementów równej liczbie wierzchołków grafu. Każdy z tych elementów odpowiada określonemu wierzchołkowi i zawiera wskaźnik na listę sąsiadujących z nim wierzchołków wraz z wagami – w przypadku grafów skierowanych przyjmuje się, że na liście znajdują się albo poprzedniki albo następniki danego wierzchołka. W celu sprawdzenia czy krawędź (u, v) należy do grafu, należy przejść listę, na którą wskazuje element tablicy odpowiadający u.

- Złożoność pamięciowa wynosi O(E)
- Złożoność czasowa dodawania nowej krawędzi wynosi O(1)
- Złożoność przeglądania sąsiadów wierzchołka, sprawdzania istnienia konkretnej krawędzi i usuwania krawędzi wynosi O(E)

Macierz sąsiedztwa:

Reprezentacja grafu w postaci macierzy sąsiedztwa wykorzystuje macierz dwuwymiarową o wymiarach V x V. Element macierzy o indeksie $_{u,v}$ odpowiada krawędzi (u, v) – jeżeli krawędź istnieje, zazwyczaj przyjmuje wartość 1. W przeciwnym razie wartość 0. Sprawdzenie, czy dana krawędź należy do grafu wymaga więc sprawdzenia wartości odpowiadającego jej elementu macierzy.

- Złożoność pamięciowa wynosi $O(V^2)$
- Złożoność przeglądania wszystkich sąsiadów wierzchołka wynosi O(V)
- Złożoność sprawdzenia istnienia konkretnej krawędzi, dodawania i usuwania krawędzi wynosi O(1)

Eksperyment został przeprowadzony dla dwóch problemów grafowych: wyznaczania minimalnego drzewa rozpinającego i poszukiwania najkrótszej ścieżki. Czasową złożoność obliczeniową, oznaczaną za pomocą O(n), określamy jako ilość czasu niezbędnego do rozwiązania problemu w zależności od liczby danych na wejściu.

Wyznaczenie minimalnego drzewa rozpinającego – algorytm DJP (algorytm Prima):

Na początku, algorytm dodaje do zbioru A reprezentującego drzewo krawędź o najmniejszej wadze, łączącą wierzchołek początkowy v z dowolnym wierzchołkiem. W każdym kolejnym kroku procedura dodaje do A najlżejszą krawędź wśród krawędzi łączących wierzchołki już odwiedzone z nieodwiedzonymi. Jeśli struktura A jest kolejką priorytetową opartą na kopcu binarnym to złożoność czasowa wynosi O(E·logV).

Wyznaczanie minimalnego drzewa rozpinającego – algorytm Kruskala:

Algorytm operuje na liście posortowanych krawędzi pod względem ich wagi. W każdym kolejnym kroku pobierana jest pierwsza krawędź. Następnie sprawdzany jest warunek, czy w wyniku dodania jej do drzewa nie powstanie cykl. Złożoność obliczeniowa wynosi O(E·logV).

Poszukiwanie najkrótszej ścieżki w grafie – algorytm Dijkstry:

Na początku działania algorytmu dystans potrzebny do pokonania w celu dostania się do wierzchołka startowego ustawia się na 0. Następnie w pętli pobierany jest wierzchołek nieodwiedzony o najmniejszym koszcie dojścia. Dla każdej krawędzi wychodzącej od wybranego wierzchołka sprawdzane jest, czy koszt dojścia do niego nie jest mniejszy niż dotychczas ustalony. Po pętli o długości V otrzymujemy najmniejsze koszty dojścia od wybranego wierzchołka. Złożoność obliczeniowa algorytmu to O(E·logV).

Poszukiwanie najkrótszej ścieżki w grafie – algorytm Forda-Bellmana:

Algorytm ten działa identycznie w kwestii obliczania kosztów dojścia jak algorytm Dijkstry. Różnica polega na tym, że nie jest wybierany nieodwiedzony wierzchołek o najmniejszym koszcie, lecz wszystkie do których w poprzedniej iteracji dochodziła krawędź. Złożoność obliczeniowa algorytmu to O(E·V).

2. Środowisko pracy i sposób prowadzenia pomiarów.

Program wykorzystywany do eksperymentów napisałem w języku C++. Przy jego projektowaniu przyjąłem następujące założenia:

- struktury wykorzystywane do reprezentacji grafu są alokowane dynamicznie
- do alokacji i zwalniania pamięci wykorzystuję funkcje new i delete
- wierzchołki i krawędzie grafu są reprezentowane jako 32-bitowe liczby całkowite
- wszystkie pomiary są powtarzane 100 razy, a ich wyniki uśredniane

- pomiary dla każdego z algorytmów są wykonywane na 5 ilościach wierzchołków 5, 10, 20, 50, 100, dla każdej ilości dodatkowo powtarzane dla 4 gęstości: 25%, 50%, 75%, 99%
- do pomiarów czasu wykorzystuję napisaną przez siebie klasę Czas, bazującą na bibliotece <windows.h>

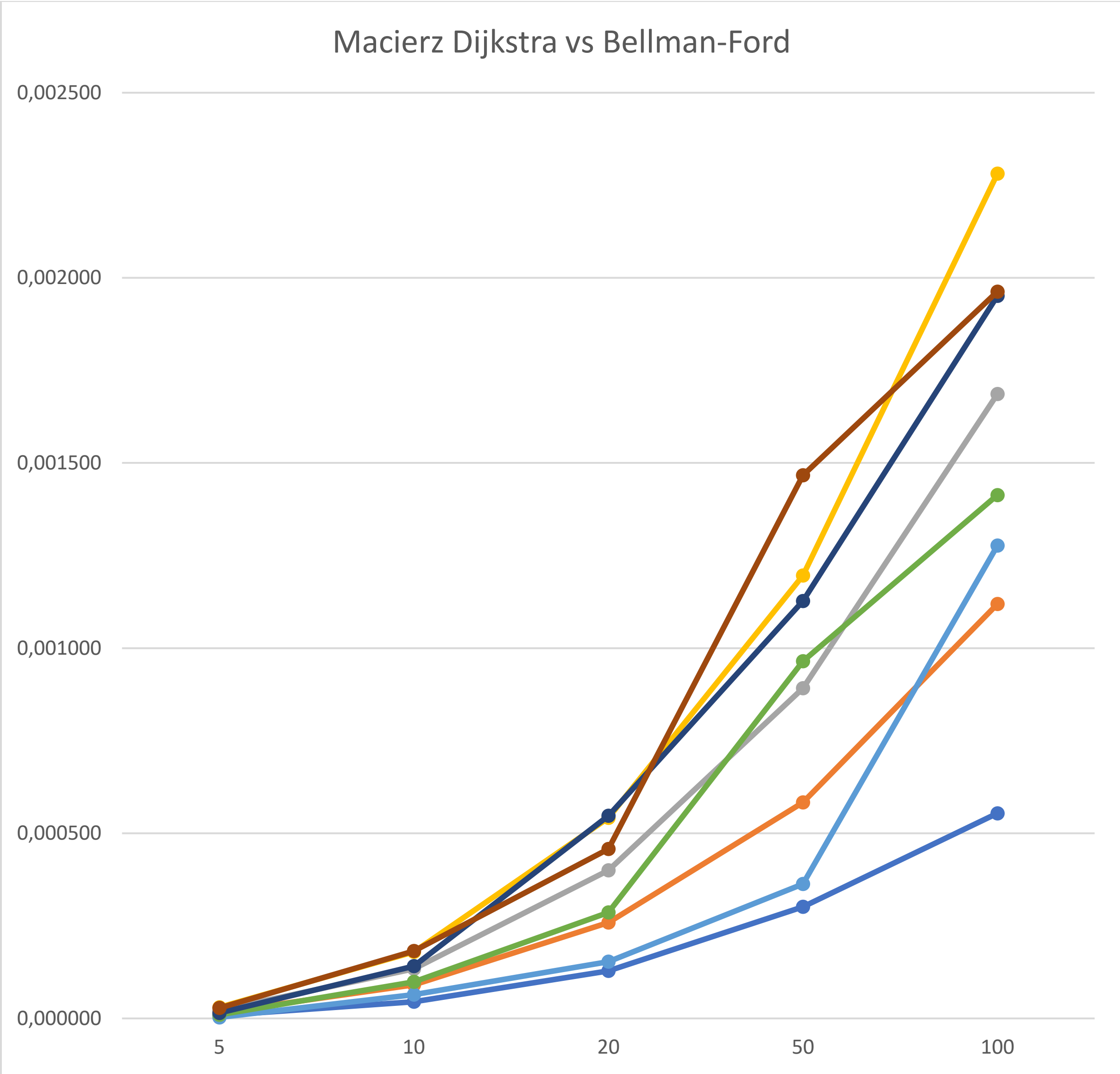
Pomiary przeprowadziłem na laptopie Lenovo Ideapad 700-15ISK, wyposażonym w czterordzeniowy ośmiowątkowy procesor Intel Core i7-6700 HQ pracujący z bazową częstotliwością 2.6 GHz i 16 GB pamięci RAM pod kontrolą systemu operacyjnego Windows 10. Pracowałem w zintegrowanym środowisku programistycznym Code::Blocks 17.12.

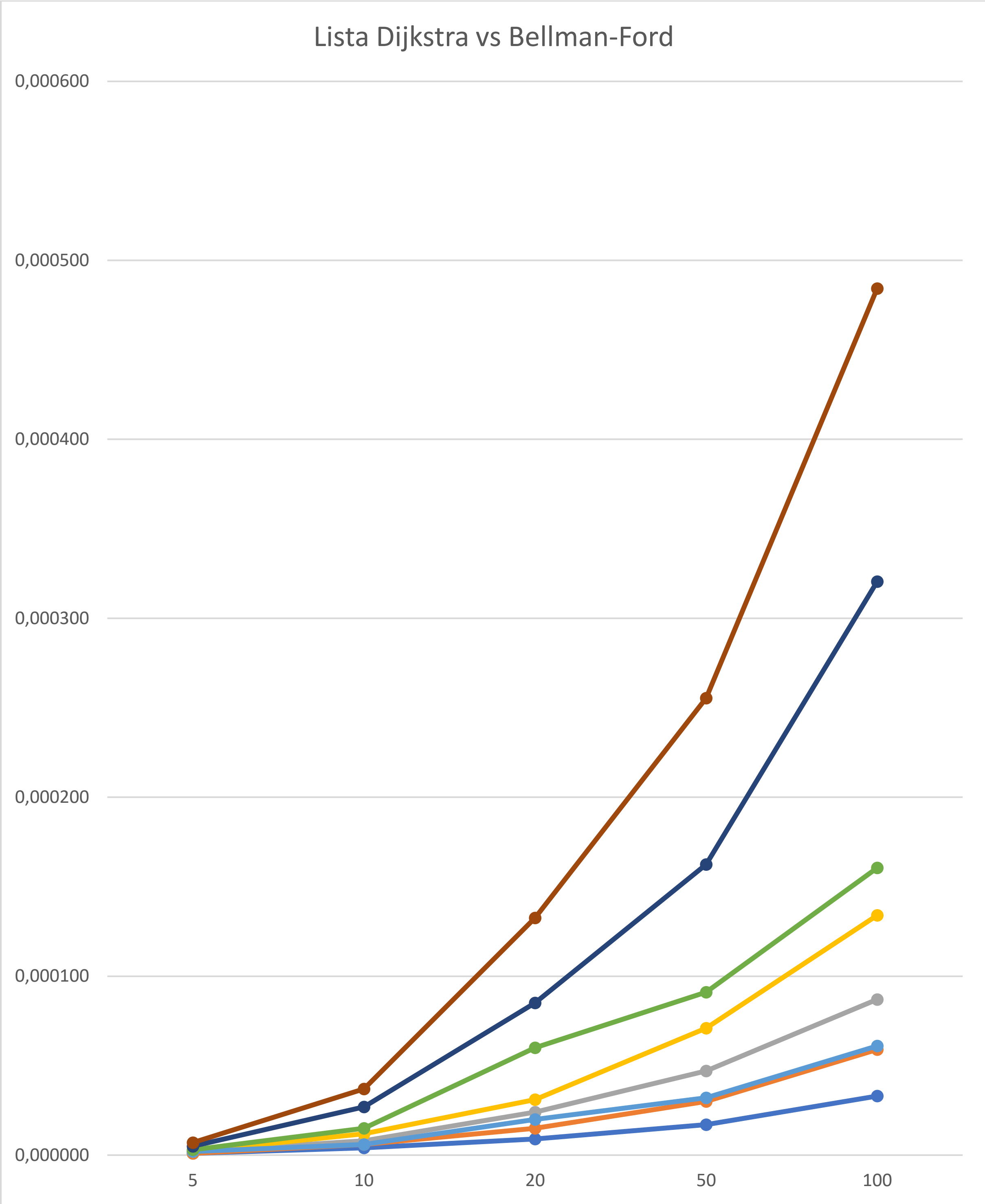
3. Wyniki.

Tabela 1 - wyniki pomiarów

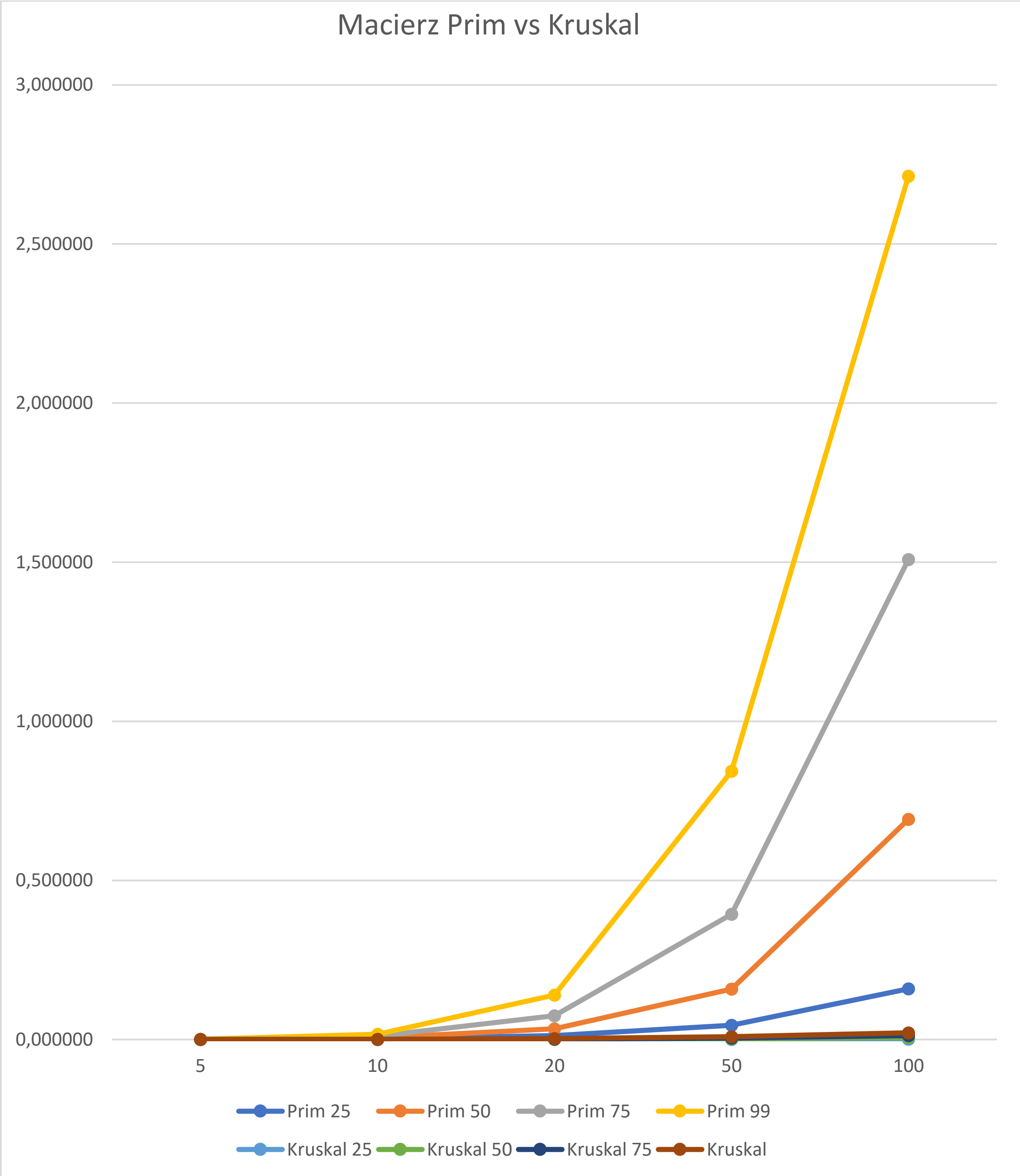
		Macierz Incydencji				Lista sąsiedztw			
l. w.	Wsp.	Dijkstra	Ford	Prim	Kruskal	Dijkstra	Ford	Prim	Kruskal
20	0,25	0,000008	0,000003	0,000178	0,000012	0,000001	0,000002	0,000126	0,000016
	0,5	0,000016	0,000010	0,000385	0,000024	0,000001	0,000003	0,000279	0,000027
	0,75	0,000022	0,000015	0,000608	0,000039	0,000002	0,000005	0,000431	0,000043
	0,99	0,000030	0,000028	0,000866	0,000059	0,000002	0,000007	0,000607	0,000058
40	0,25	0,000045	0,000064	0,002426	0,000068	0,000004	0,000006	0,001334	0,000065
	0,5	0,000091	0,000099	0,005829	0,000186	0,000006	0,000015	0,003659	0,000151
	0,75	0,000134	0,000141	0,010342	0,000366	0,000008	0,000027	0,007521	0,000259
	0,99	0,000180	0,000182	0,016334	0,000565	0,000012	0,000037	0,012423	0,000396
60	0,25	0,000128	0,000153	0,012633	0,000226	0,000009	0,000020	0,006938	0,000195
	0,5	0,000259	0,000286	0,033624	0,000724	0,000015	0,000060	0,024311	0,000496
	0,75	0,000400	0,000547	0,074480	0,001561	0,000024	0,000085	0,060217	0,000977
	0,99	0,000542	0,000457	0,139266	0,002685	0,000031	0,000133	0,123968	0,001629
80	0,25	0,000301	0,000363	0,044403	0,000622	0,000017	0,000032	0,025230	0,000459
	0,5	0,000583	0,000965	0,158261	0,002223	0,000030	0,000091	0,124968	0,001340
	0,75	0,000892	0,001127	0,393695	0,004940	0,000047	0,000163	0,365136	0,003489
	0,99	0,001196	0,001467	0,843599	0,008584	0,000071	0,000255	0,697229	0,007277
100	0,25	0,000554	0,001277	0,159152	0,001386	0,000033	0,000061	0,089088	0,000848
	0,5	0,001119	0,001413	0,691810	0,005391	0,000059	0,000161	0,490418	0,003788
	0,75	0,001686	0,001951	1,508390	0,012145	0,000087	0,000320	1,295790	0,009281
	0,99	0,002281	0,001963	2,712810	0,021066	0,000134	0,000484	2,521490	0,016021

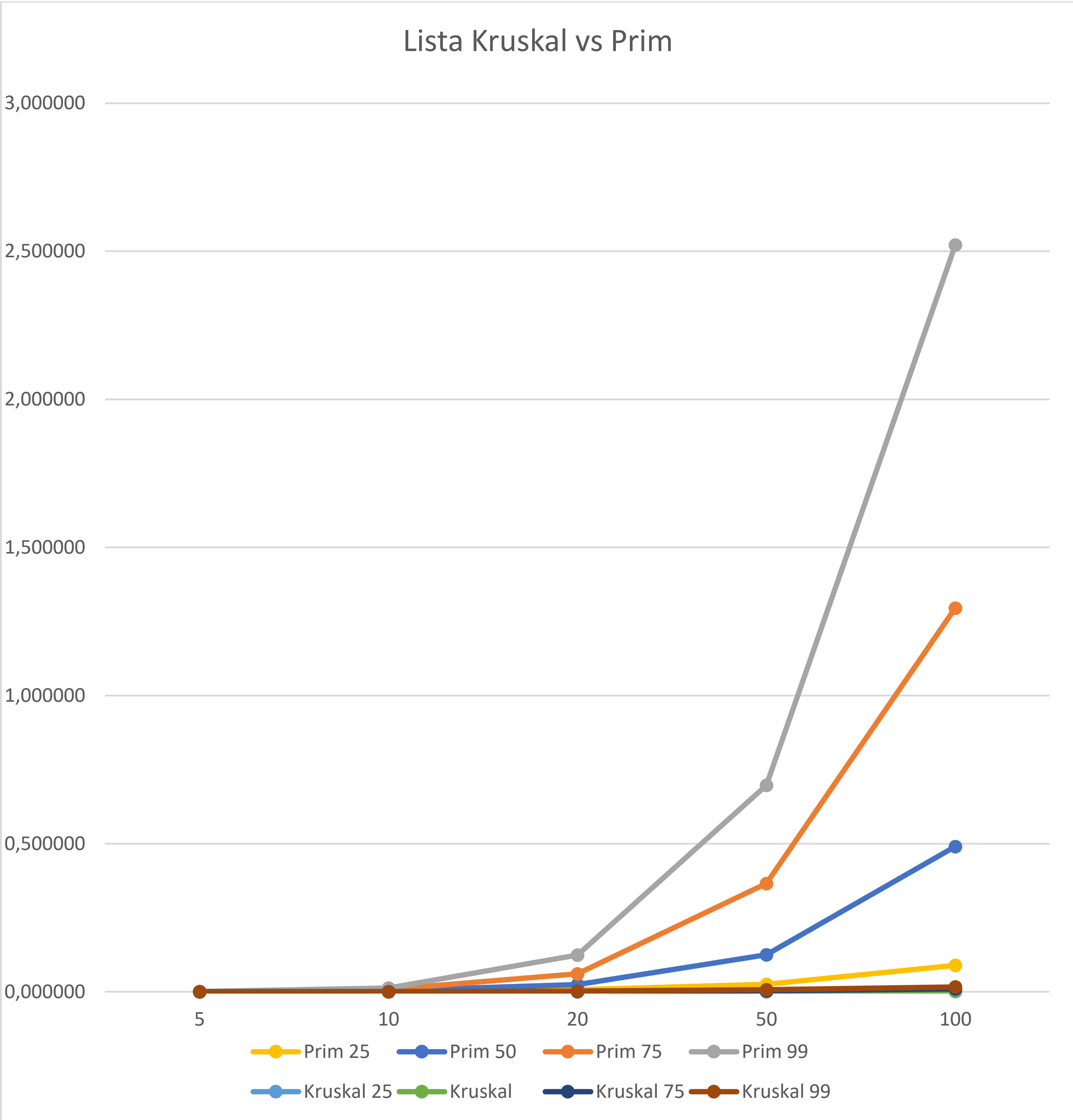
3.1. Wykresy typ 1



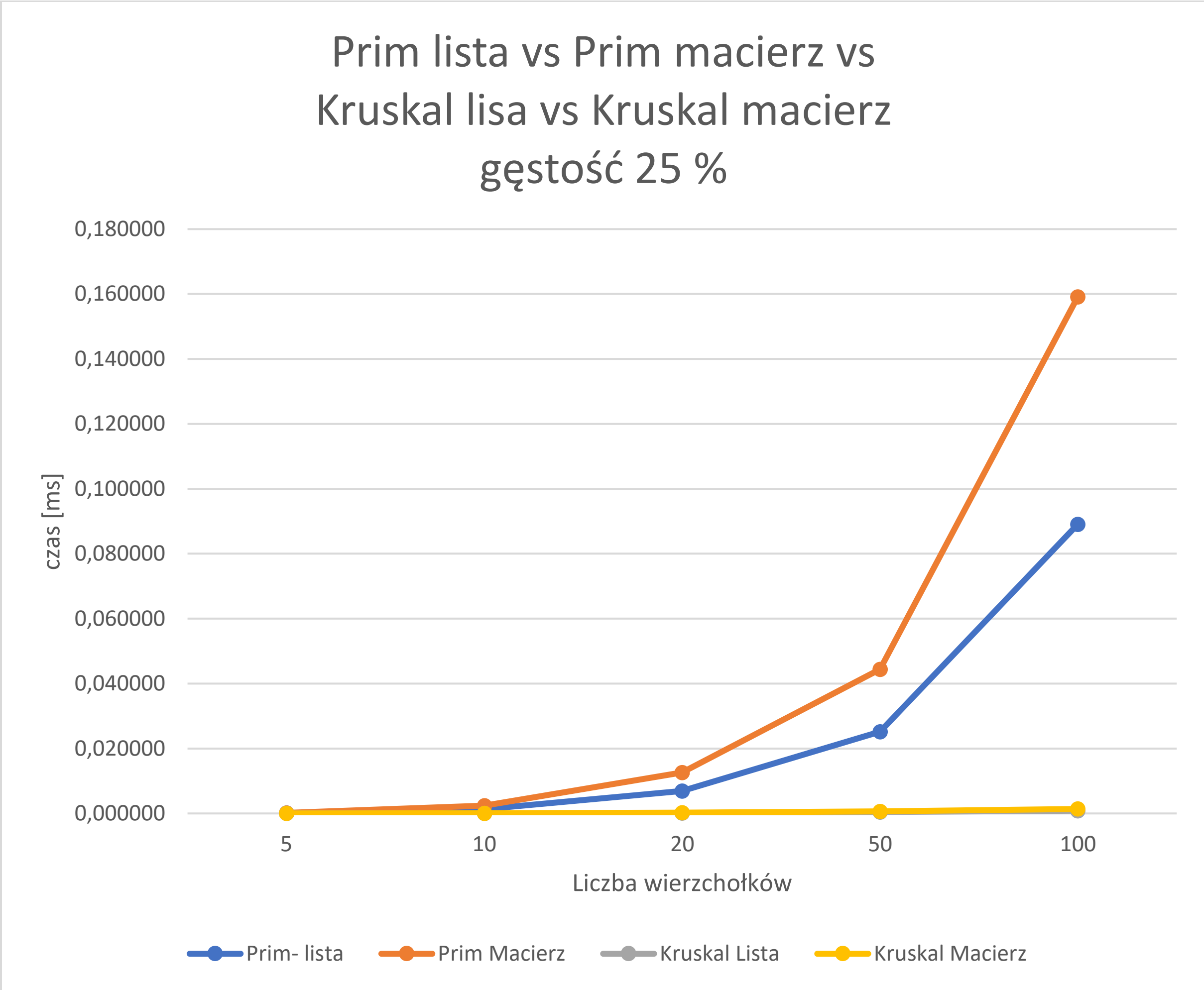


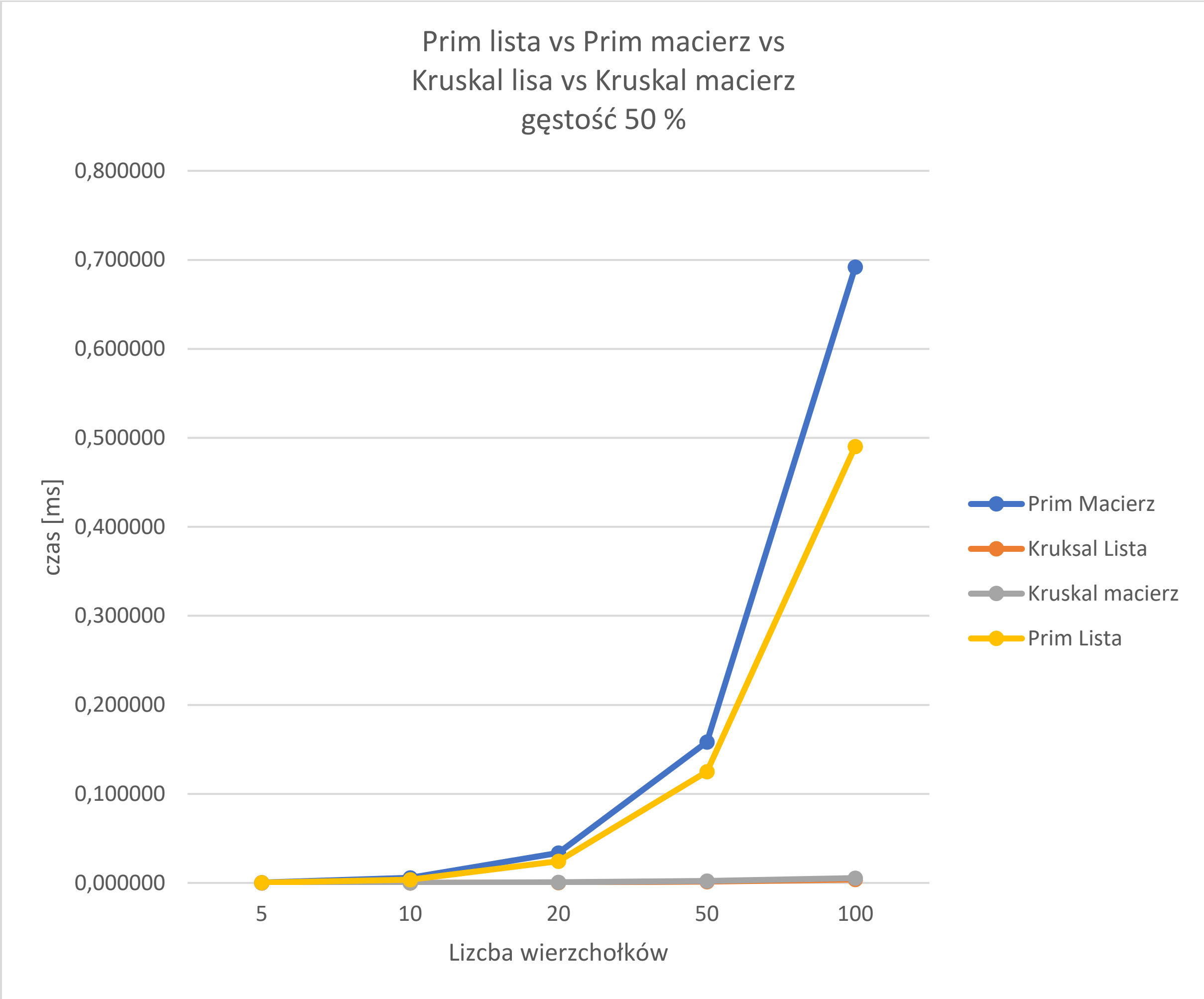
Lista Dijkstra vs Bellman-Ford
0,000600
0,000500
0,000400
0,000300
0,000200
0,000100
0,000000
5
10
20
50
100

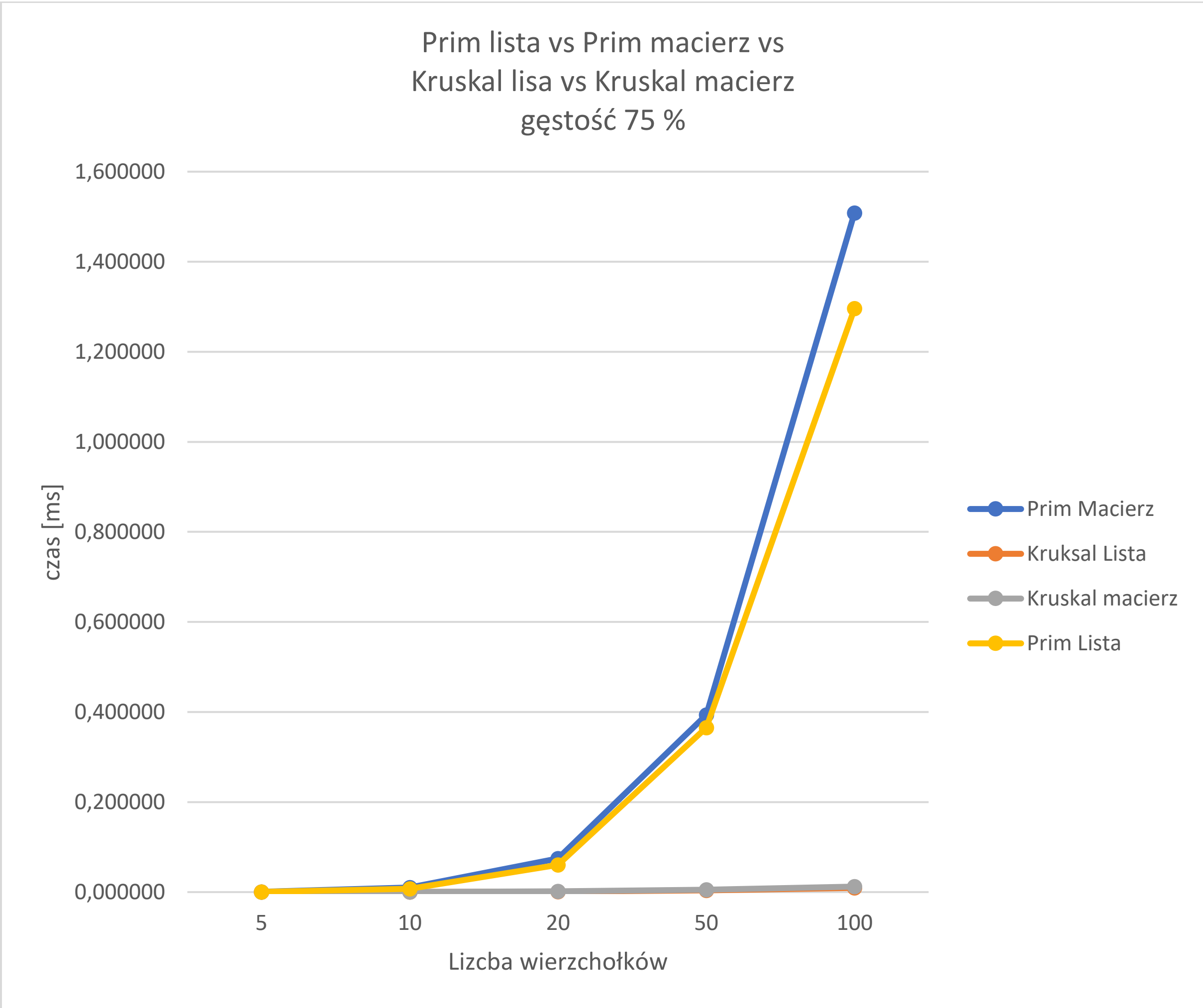

Macierz Prim vs Kruskal
3,000000
2,500000
2,000000
1,500000
1,000000
0,500000
0,000000
5
10
20
50
100
Prim 25
Prim 50
Prim 75
Prim 99
Kruskal 25
Kruskal 50
Kruskal 75
Kruskal

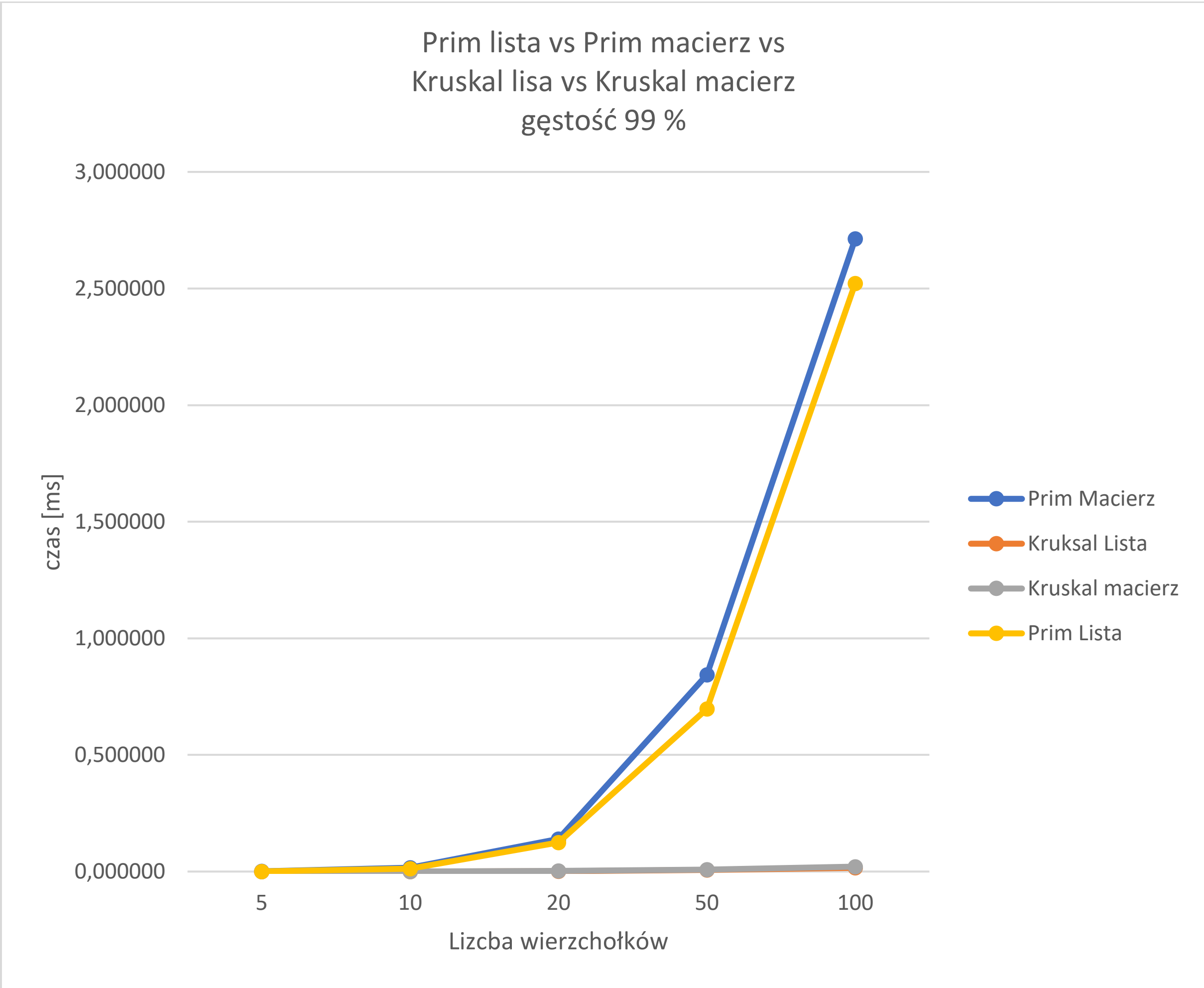


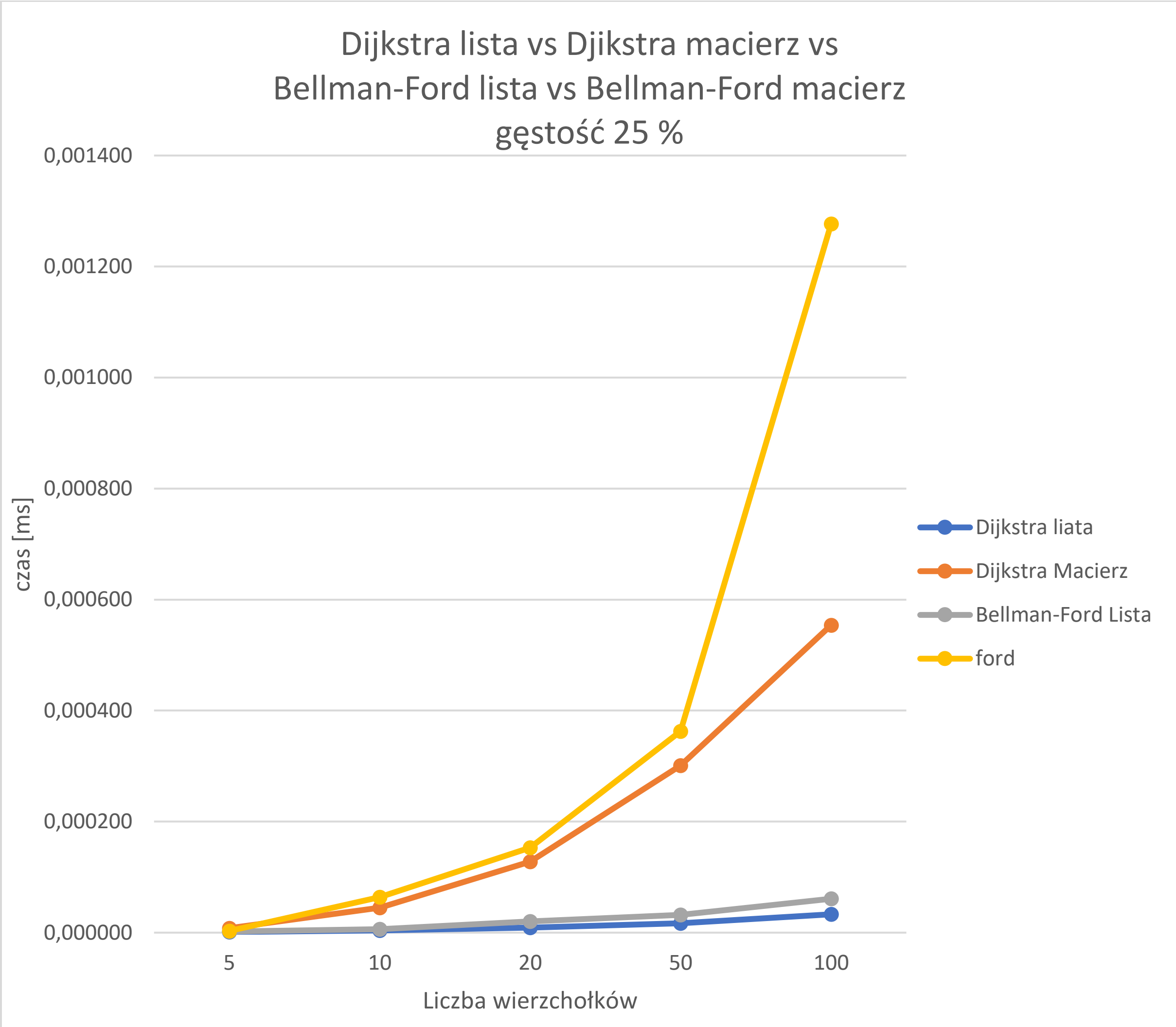
3.2. Wykresy typ 2

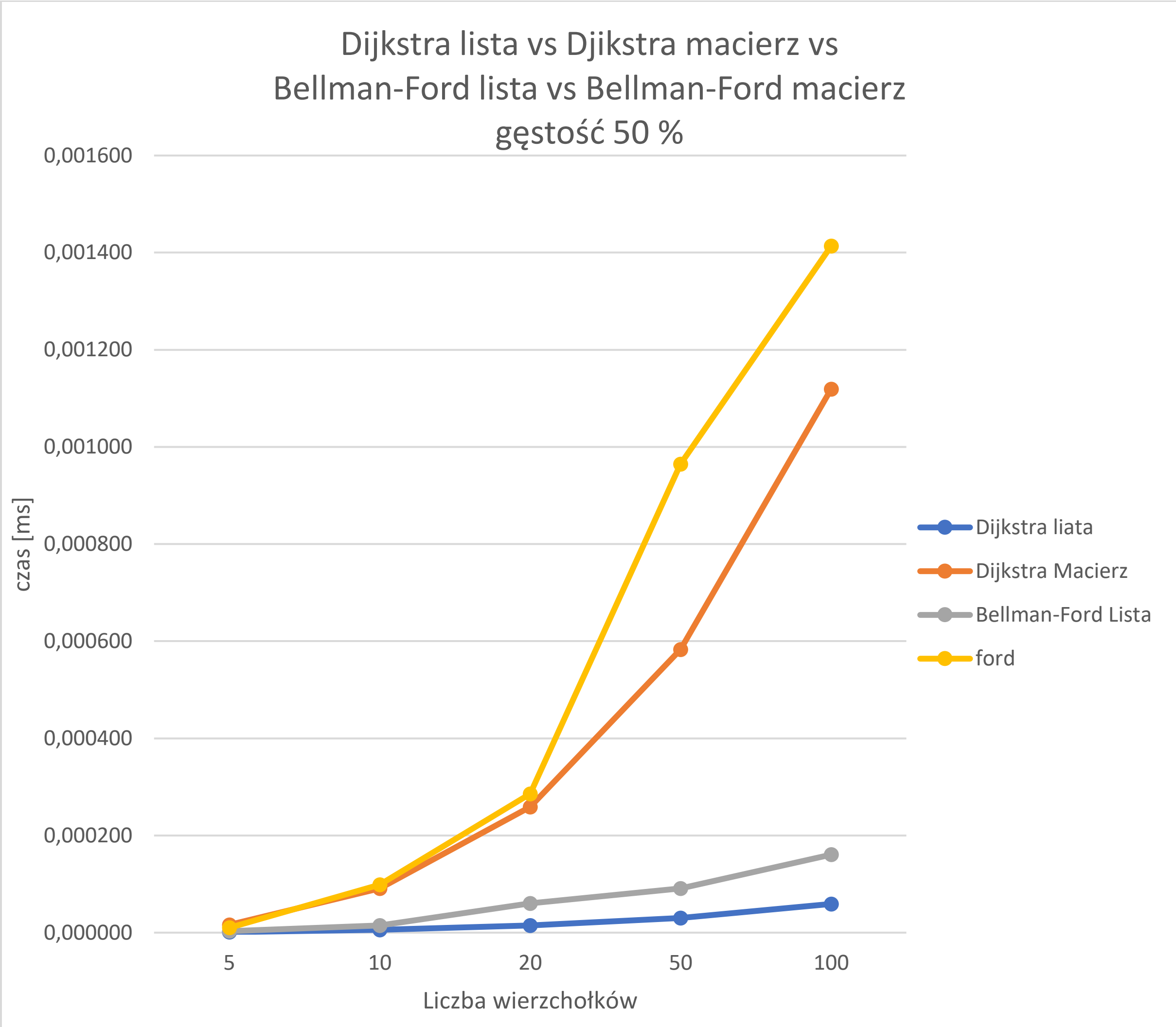

Prim lista vs Prim macierz vs
Kruskal lisa vs Kruskal macierz
gęstość 25 %
czas [ms]
0,180000
0,160000
0,140000
0,120000
0,100000
0,080000
0,060000
0,040000
0,020000
0,000000
5
10
20
50
100
Liczba wierzchołków
Prim- lista
Prim Macierz
Kruskal Lista
Kruskal Macierz

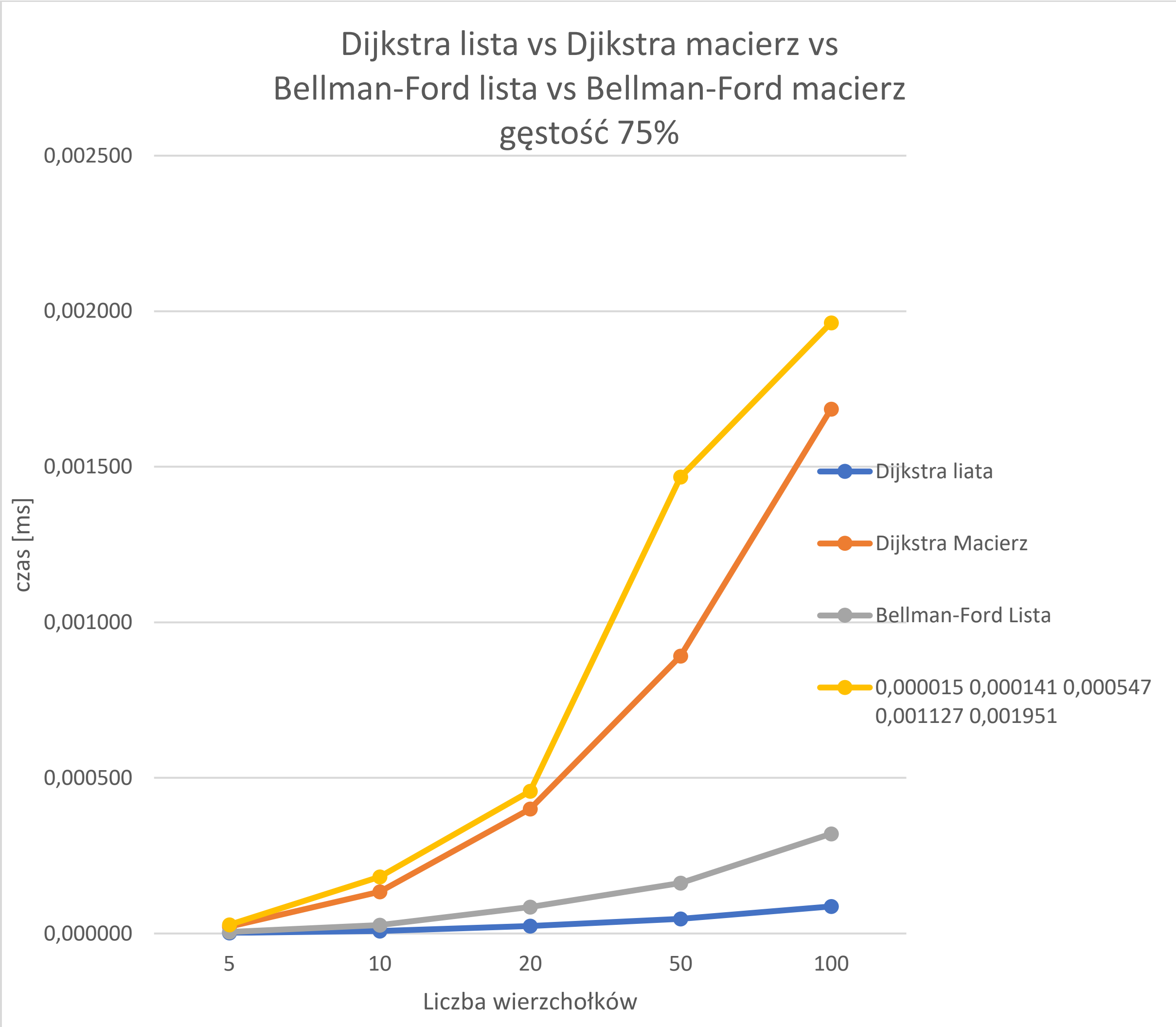

Prim lista vs Prim macierz vs
Kruskal lisa vs Kruskal macierz
gęstość 50 %
0,800000
0,700000
0,600000
0,500000
0,400000
0,300000
0,200000
0,100000
0,000000
czas [ms]
5
10
20
50
100
Lizcba wierzchołków
Prim Macierz
Kruksal Lista
Kruskal macierz
Prim Lista


Prim lista vs Prim macierz vs
Kruskal lisa vs Kruskal macierz
gęstość 75 %
1,600000
1,400000
1,200000
1,000000
0,800000
0,600000
0,400000
0,200000
0,000000
czas [ms]
5
10
20
50
100
Lizcba wierzchołków
Prim Macierz
Kruksal Lista
Kruskal macierz
Prim Lista

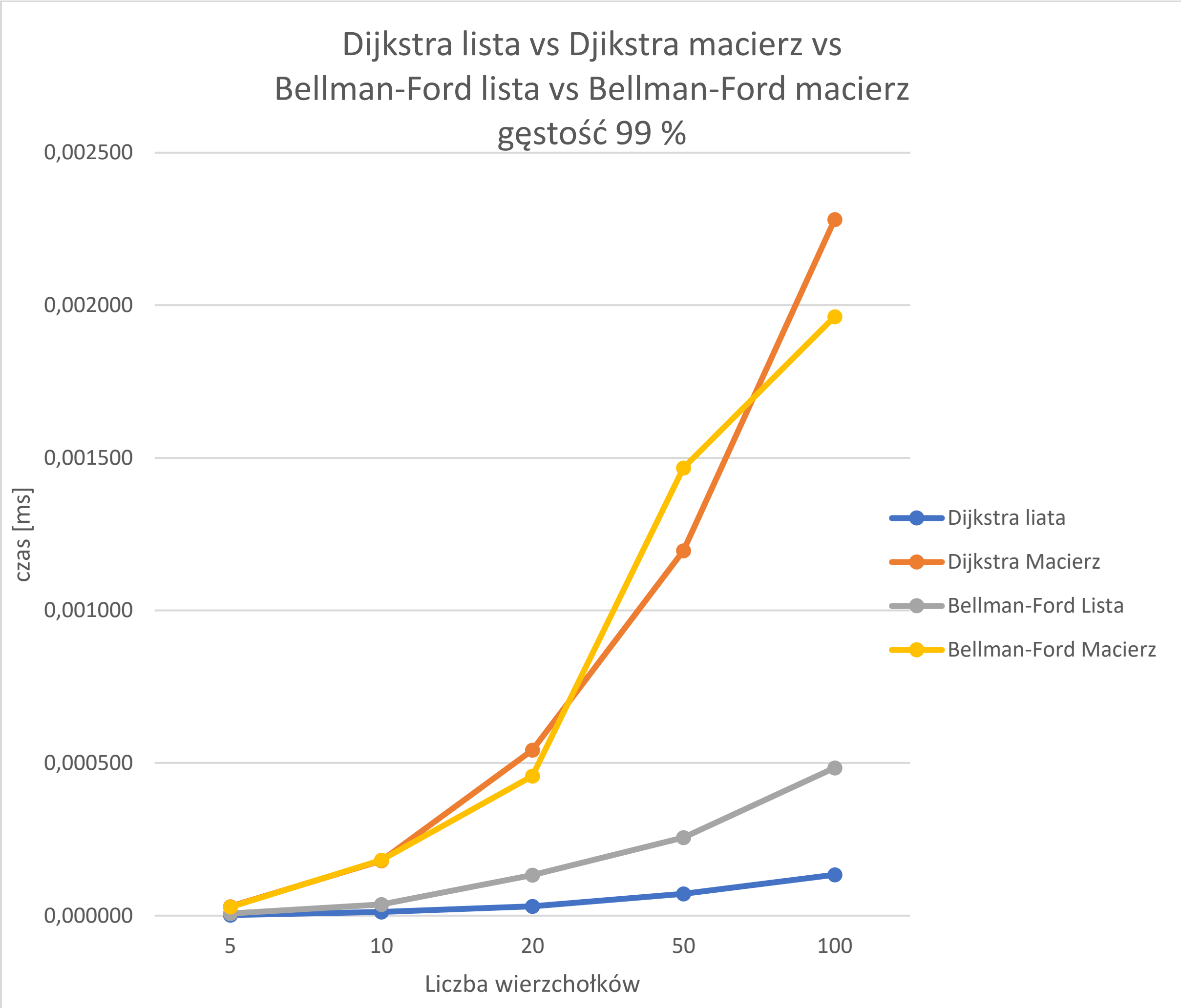

Prim lista vs Prim macierz vs
Kruskal lisa vs Kruskal macierz
gęstość 99 %
3,000000
2,500000
2,000000
1,500000
1,000000
0,500000
0,000000
czas [ms]
5
10
20
50
100
Lizcba wierzchołków
Prim Macierz
Kruksal Lista
Kruskal macierz
Prim Lista


Dijkstra lista vs Djikstra macierz vs
Bellman-Ford lista vs Bellman-Ford macierz
gęstość 25 %
0,001400
0,001200
0,001000
0,000800
0,000600
0,000400
0,000200
0,000000
czas [ms]
5
10
20
50
100
Liczba wierzchołków
Dijkstra liata
Dijkstra Macierz
Bellman-Ford Lista
ford


Dijkstra lista vs Djikstra macierz vs
Bellman-Ford lista vs Bellman-Ford macierz
gęstość 50 %
0,001600
0,001400
0,001200
0,001000
0,000800
0,000600
0,000400
0,000200
0,000000
czas [ms]
5
10
20
50
100
Liczba wierzchołków
Dijkstra liata
Dijkstra Macierz
Bellman-Ford Lista
ford



Dijkstra lista vs Djikstra macierz vs
Bellman-Ford lista vs Bellman-Ford macierz
gęstość 75%
0,002500
0,002000
0,001500
0,001000
0,000500
0,000000
czas [ms]
5
10
20
50
100
Liczba wierzchołków
Dijkstra liata
Dijkstra Macierz
Bellman-Ford Lista
0,000015 0,000141 0,000547
0,001127 0,001951



4. Wnioski

Implementacja macierzowa pozwala w przeciwieństwie do listy wykonywać większość operacje w czasie stałym. Przez to wzrost czasu wykonania dla niej jest mniejszy niż dla listy.