Struktury Danych i Złożoność Obliczeniowa

Sprawozdanie	Zadanie projektowe nr 2
Imię i nazwisko, nr indeksu:	Piotr Komarnicki, 264486
Termin zajęć:	Wtorek 15:15 TP
Numer grupy ćwiczeniowej:	K03-37p
Prowadzący kurs:	Mgr inż. Antoni Sterna

1. Wprowadzenie

Zadaniem projektowym było zaimplementowanie podstawowych algorytmów grafowych dla dwóch różnych reprezentacji grafu w pamięci komputera:

- macierzy sąsiedztwa
- listy sąsiadów

Algorytmy, które zaimplementowałem w ramach tego projektu można podzielić na dwie grupy ze względu na realizowane zadanie.

Algorytmy wyznaczania minimalnego drzewa rozpinającego:

- algorytm Kruskala
- algorytm Prima

oraz algorytmy wyznaczania najkrótszej ścieżki w grafie ze startowego wierzchołka:

- algorytm Dijkstry
- algorytm Bellmana – Forda

Drugą część projektu polegała na analizie czasowej działania powyższych algorytmów ze względu na ilość wierzchołków i gęstość grafu oraz sprawdzenie jaki wpływ na działanie algorytmu ma sposób przechowywania grafu w pamięci komputera.

2. Złożoność obliczeniowa algorytmów

A. Algorytmy wyznaczania najkrótszej ścieżki

Algorytm Dijkstry w implementacji z kolejką priorytetową w postaci kopca binarnego ma złożoność równą **O(ElogV)**. Z kolei algorytm Bellmana – Forda ma złożoność obliczeniową **O(V*E)**, która jak widać jest większa od złożoności obliczeniowej algorytmu Dijkstry. Dodatkowy koszt związany jest z faktem, że algorytm Bellmana - Forda nadaje się również do szukania dróg w grafach z ujemnymi wagami.

B. Algorytmy wyznaczania minimalnego drzewa rozpinającego

Złożoność obliczeniowa algorytmu Prima w implementacji w postaci kopca binarnego wynosi **O(E log(V))**. Z kolei złożoność algorytmu Kruskala powiązana jest z zastosowaną metodą sortowania krawędzi, dlatego wynosi ona **O(ElogE)**.

3. Plan eksperymentu

A. Implementacja

Algorytmy zostały zaimplementowane w języku C++ z wykorzystaniem biblioteki STL. Numery wierzchołków oraz wagi krawędzi reprezentowane są liczbą 4 bajtowa ze znakiem (int). W reprezentacji macierzy sąsiedztwa oraz w algorytmach zakładamy, że wartość 2147483647 (INT_MAX) reprezentuje nieskończoność lub brak krawędzi w przypadku macierzy sąsiedztwa.

Reprezentacje grafów znajdują się w oddzielnych klasach: Matrix_Graph oraz List_Graph. Każda z tych klas przechowuje graf w pamięci komputera w inny sposób oraz posiada metody pozwalające na uruchomienie algorytmów działających na danej reprezentacji grafu. Każda z tych klas pozwala również na wczytanie grafu z pliku o odpowiednim formacie. Zakładamy, że graf dla problemu minimalnego drzewa rozpinającego jest grafem nieskierowanym, a dla problemu najkrótszej ścieżki jest grafem skierowanym.

Do pomiaru czasu skorzystałem z std::chrono::high_resolution_clock z biblioteki <chrono>. Pomiary przeprowadzono dla grafów o następującej liczbie wierzchołków: 100, 200, 300, 400, 500 oraz dla gęstości 25%, 50%, 75% oraz 99%. Osobno zmierzono działanie algorytmów dla grafu w reprezentacji macierzy sąsiedztwa oraz listy sąsiadów. Każdy pomiar wykonano 100 razy, za każdym razem generowano nowy graf.

B. Generowanie danych do testów

Do generowania grafów testowych stworzono metodę w klasie Tester, która dla zadanej liczby wierzchołków oraz gęstości generuje graf pełny, a następnie losowo usuwa krawędzie tak aby spełnić wymóg dotyczący gęstości grafu. Aby zachować spójność grafu zawsze generowany jest zbiór krawędzi, które łączą wszystkie wierzchołki, następnie krawędzie dodawane są do zbioru dodatkowych krawędzi tak aby gęstość grafu była odpowiednia. Tak utworzony graf zapisywany jest do pliku w odpowiednim formacie i następnie wczytywany na potrzeby testów.

4. Rodzaje testów

A. Testy ręczne

Program pozwala na ręczne testowanie zaimplementowanych algorytmów. Po wybraniu rodzaju problemu pojawia się odpowiednie menu, które pozwala na:

- wczytanie grafu z pliku o odpowiednim formacie
- wyświetleniu grafu w postaci macierzy sąsiedztwa oraz listy sąsiadów
- wywołania oraz wyświetlenia wyników dla poszczególnych algorytmów dla różnych reprezentacji grafu

B. Testy czasowe

Jeśli podamy jako pierwszy argument wywołania test oraz jako drugi argument podamy: Dijkstra, bellmanford, prim lub kruskal to wywołamy automatyczne testy dla odpowiednich algorytmów. Wyniki tych testów zostaną zapisane w pliku z rozszerzeniem csv.

5. Wyniki eksperymentu – algorytm wyznaczania minimalnego drzewa rozpinającego

A. Algorytm Kruskala

Implementacja z wykorzystaniem macierzy sąsiedztwa:

Gęstość grafu	Liczba wierzchołków				
	100	**200**	**300**	**400**	**500**
25%	76	268	588	1000	1507
50%	132	486	1045	1965	3011
75%	160	610	1476	2386	3995
99%	197	768	1937	3241	4804

Tabela 1: Czas wykonania [µs] algorytmu Kruskala dla grafu reprezentowanego przez macierz sąsiedztwa w zależności od gęstości grafu i liczby wierzchołków

Implementacja z wykorzystaniem listy sąsiadów:

Gęstość grafu	Liczba wierzchołków				
	100	**200**	**300**	**400**	**500**
25%	79	251	628	1298	2183
50%	129	533	1557	3117	5015
75%	177	883	2573	4733	8336
99%	239	1326	3697	6994	13285

Tabela 2: Czas wykonania [µs] algorytmu Kruskala dla grafu reprezentowanego przez listę sąsiadów w zależności od gęstości grafu i liczby wierzchołków

B. Algorytm Prima

Implementacja z wykorzystaniem macierzy sąsiedztwa:

Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	58	188	395	645	966
50%	84	279	595	1015	1506
75%	92	238	511	827	1211
99%	53	150	291	453	603

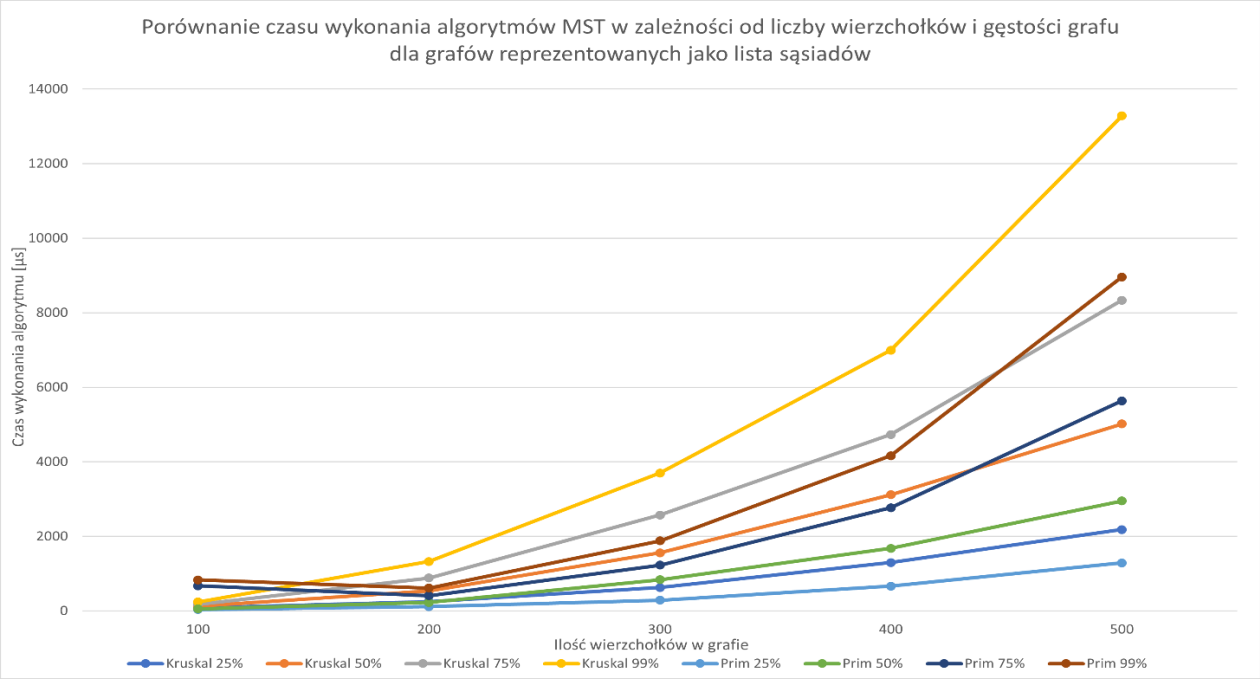
Tabela 3: Czas wykonania [µs] algorytmu Kruskala dla grafu reprezentowanego przez macierz sąsiedztwa w zależności od gęstości grafu i liczby wierzchołków

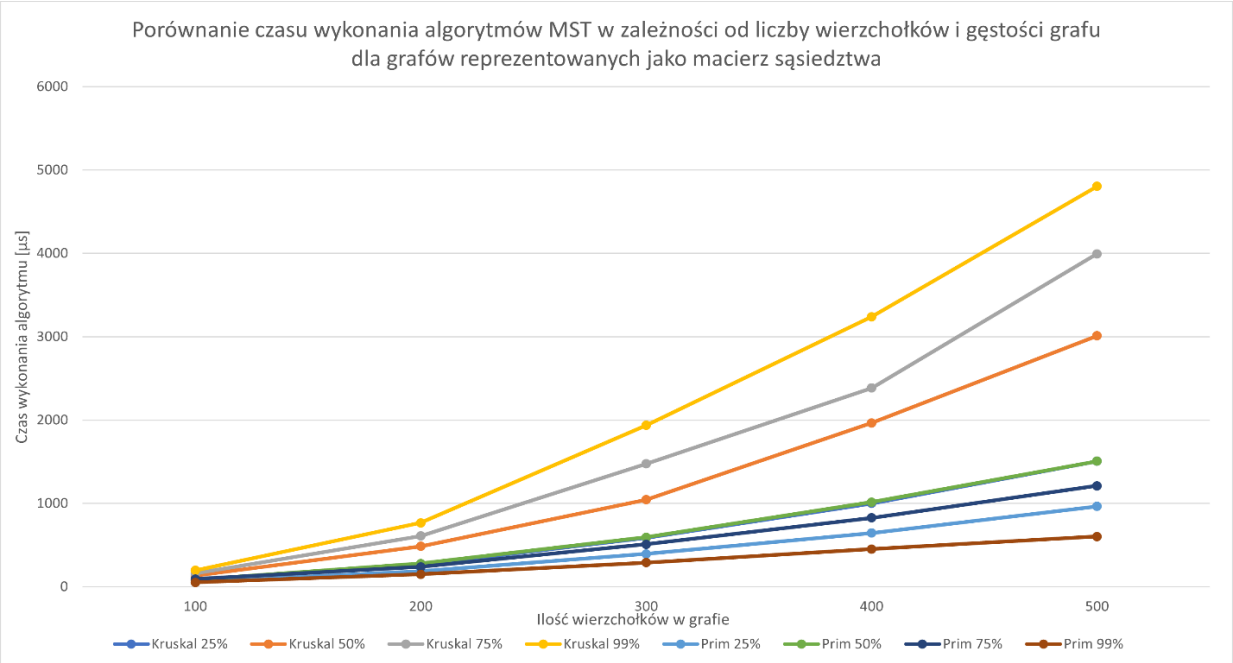
Implementacja z wykorzystaniem listy sąsiadów:

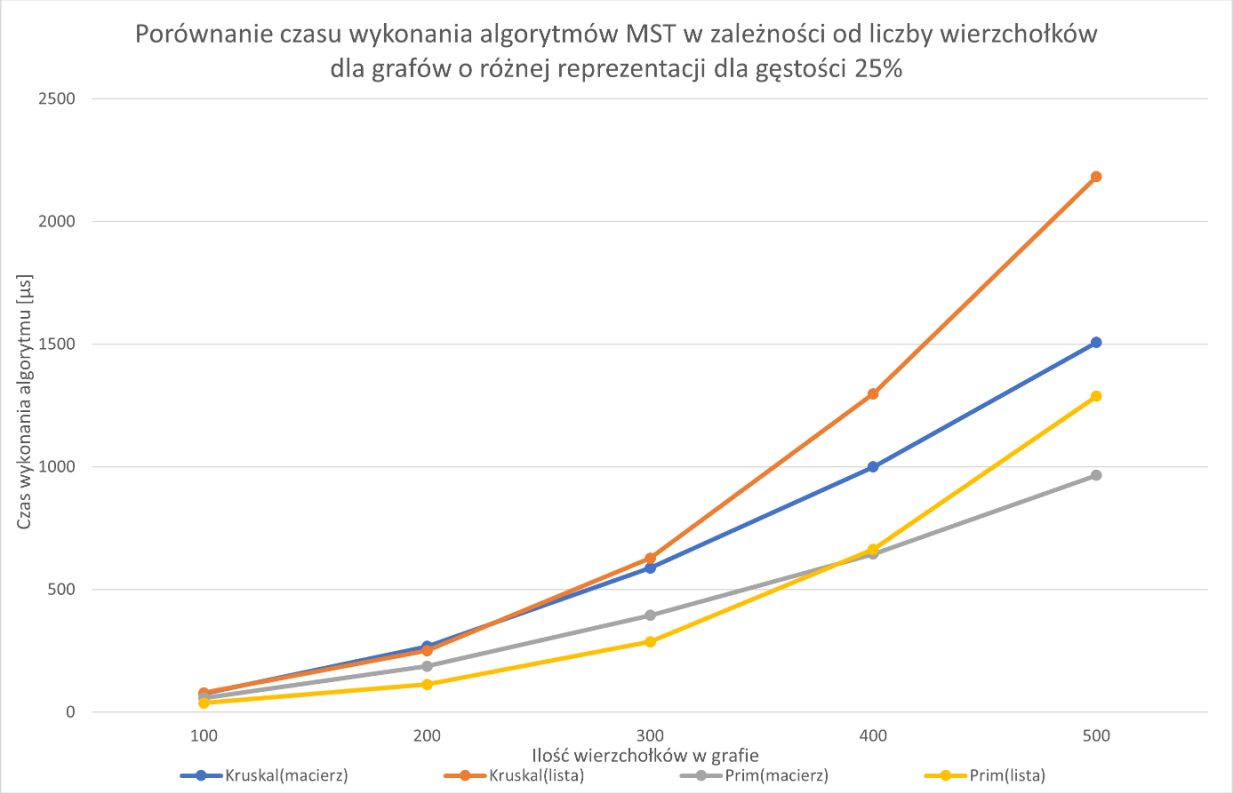
Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	37	113	287	664	1289
50%	53	228	836	1680	2949
75%	673	409	1228	2768	5637
99%	830	613	1880	4167	8960

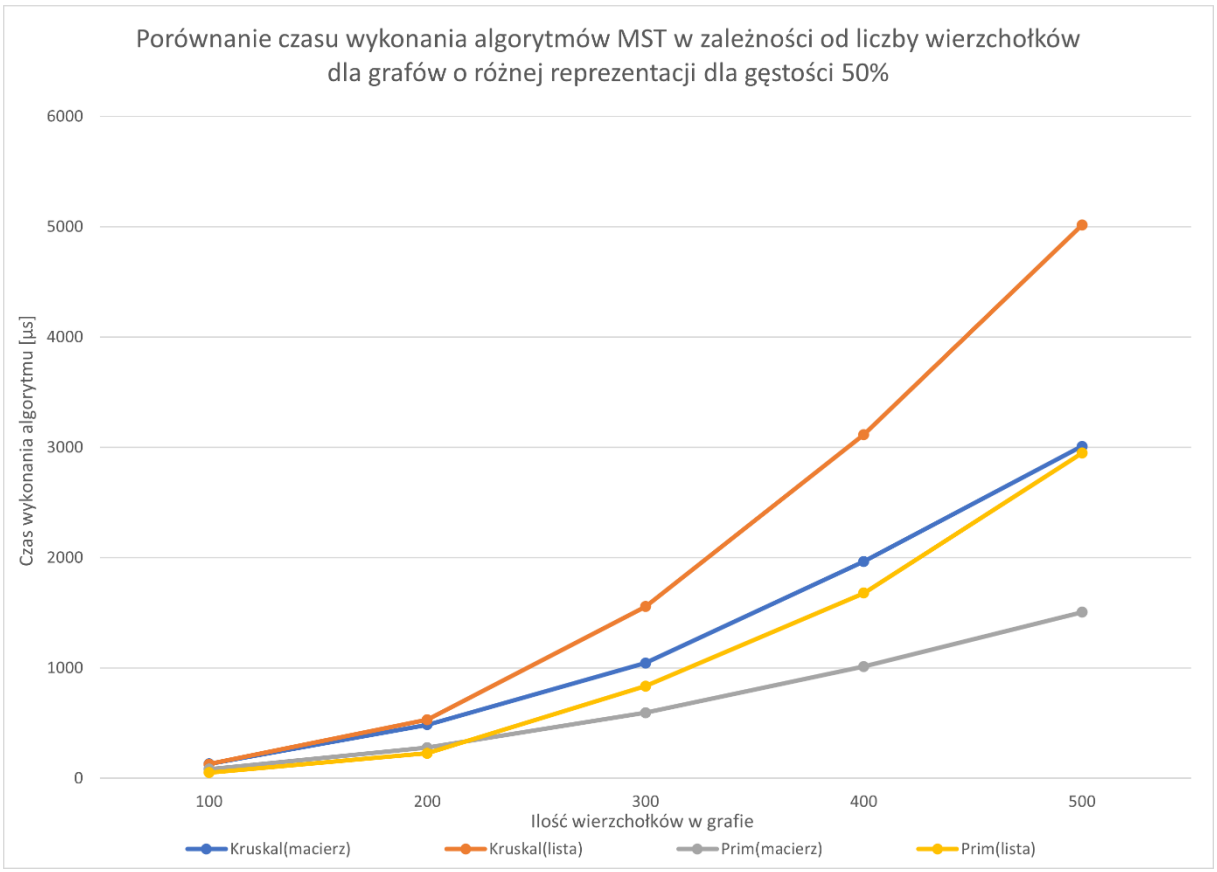
Tabela 4: Czas wykonania [µs] algorytmu Prima dla grafu reprezentowanego przez listę sąsiadów w zależności od gęstości grafu i liczby wierzchołków

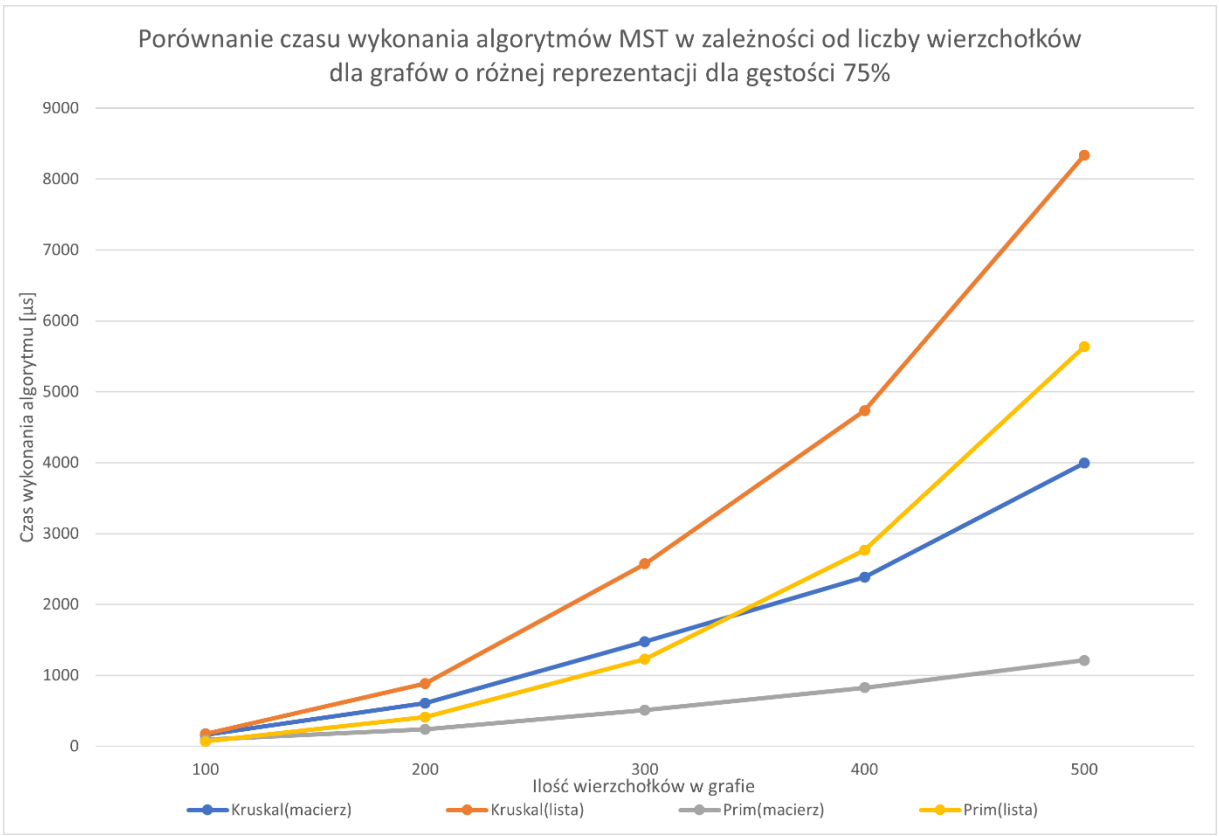
C. Wykresy

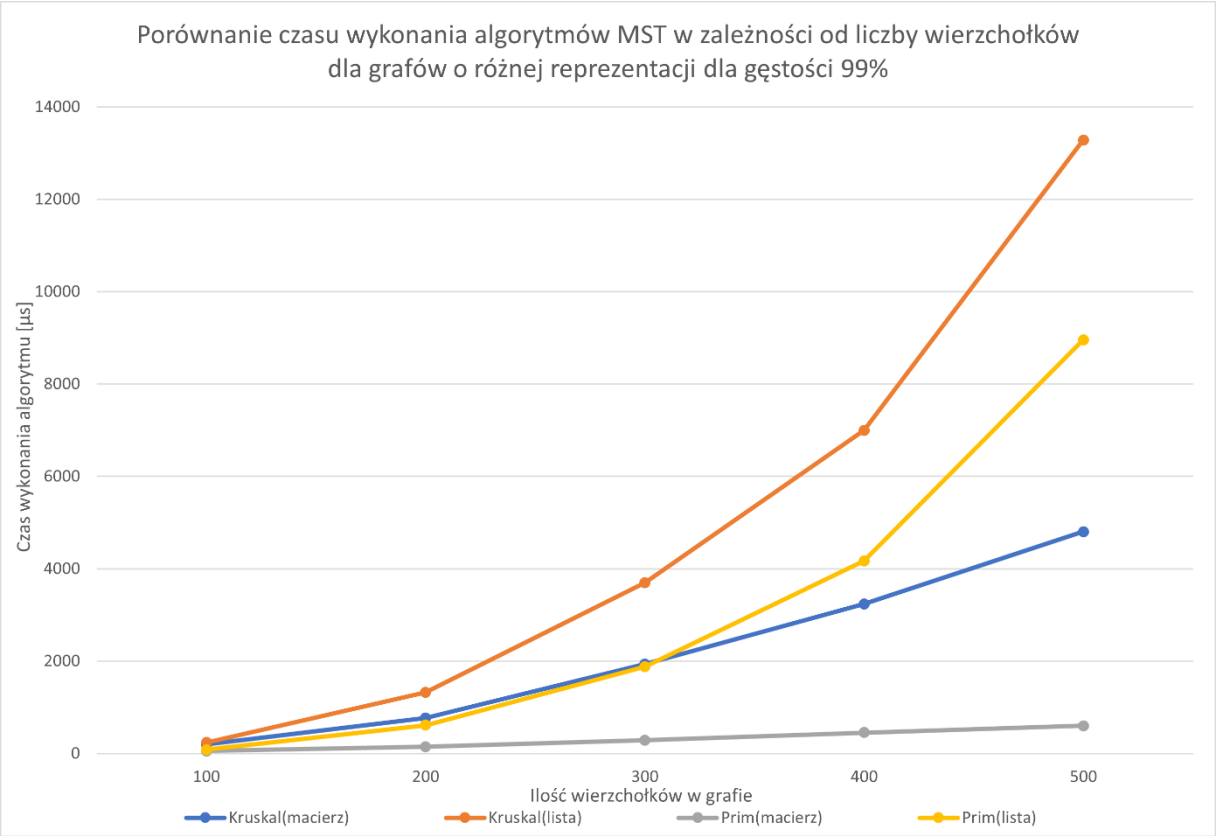



Porównanie czasu wykonania algorytmów MST w zależności od liczby wierzchołków i gęstości grafu dla grafów reprezentowanych jako macierz sąsiedztwa
Czas wykonania algorytmu [µs]
6000
5000
4000
3000
2000
1000
0
100
200
300
400
500
Ilość wierzchołków w grafie
Kruskal 25%
Kruskal 50%
Kruskal 75%
Kruskal 99%
Prim 25%
Prim 50%
Prim 75%
Prim 99%


Porównanie czasu wykonania algorytmów MST w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 25%
Czas wykonania algorytmu [µs]
2500
2000
1500
1000
500
0
100
200
300
400
500
Ilość wierzchołków w grafie
Kruskal(macierz)
Kruskal(lista)
Prim(macierz)
Prim(lista)


Porównanie czasu wykonania algorytmów MST w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 50%
Czas wykonania algorytmu [µs]
6000
5000
4000
3000
2000
1000
0
100
200
300
400
500
Ilość wierzchołków w grafie
Kruskal(macierz)
Kruskal(lista)
Prim(macierz)
Prim(lista)


Porównanie czasu wykonania algorytmów MST w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 75%
Czas wykonania algorytmu [µs]
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
100
200
300
400
500
Ilość wierzchołków w grafie
Kruskal(macierz)
Kruskal(lista)
Prim(macierz)
Prim(lista)



6. Wyniki eksperymentu – algorytm znajdowania najkrótszej ścieżki

A. Algorytm Dijkstry

Implementacja z wykorzystaniem macierzy sąsiedztwa:

Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	36	105	216	364	571
50%	45	150	321	543	829
75%	42	119	251	407	598
99%	32	89	168	251	332

Tabela 5: Czas wykonania [µs] algorytmu Dijkstry dla grafu reprezentowanego przez macierz sąsiedztwa w zależności od gęstości grafu i liczby wierzchołków

Implementacja z wykorzystaniem listy sąsiadów:

Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	23	59	123	255	494
50%	29	89	282	569	1418
75%	37	150	455	1029	2423
99%	41	209	661	1676	3236

Tabela 6: Czas wykonania [µs] algorytmu Dijkstry dla grafu reprezentowanego przez listę sąsiadów w zależności od gęstości grafu i liczby wierzchołków

B. Algorytm Bellmana - Forda

Implementacja z wykorzystaniem macierzy sąsiedztwa:

Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	42	157	356	611	943
50%	61	233	509	903	1375
75%	48	173	396	673	1025
99%	23	72	159	266	375

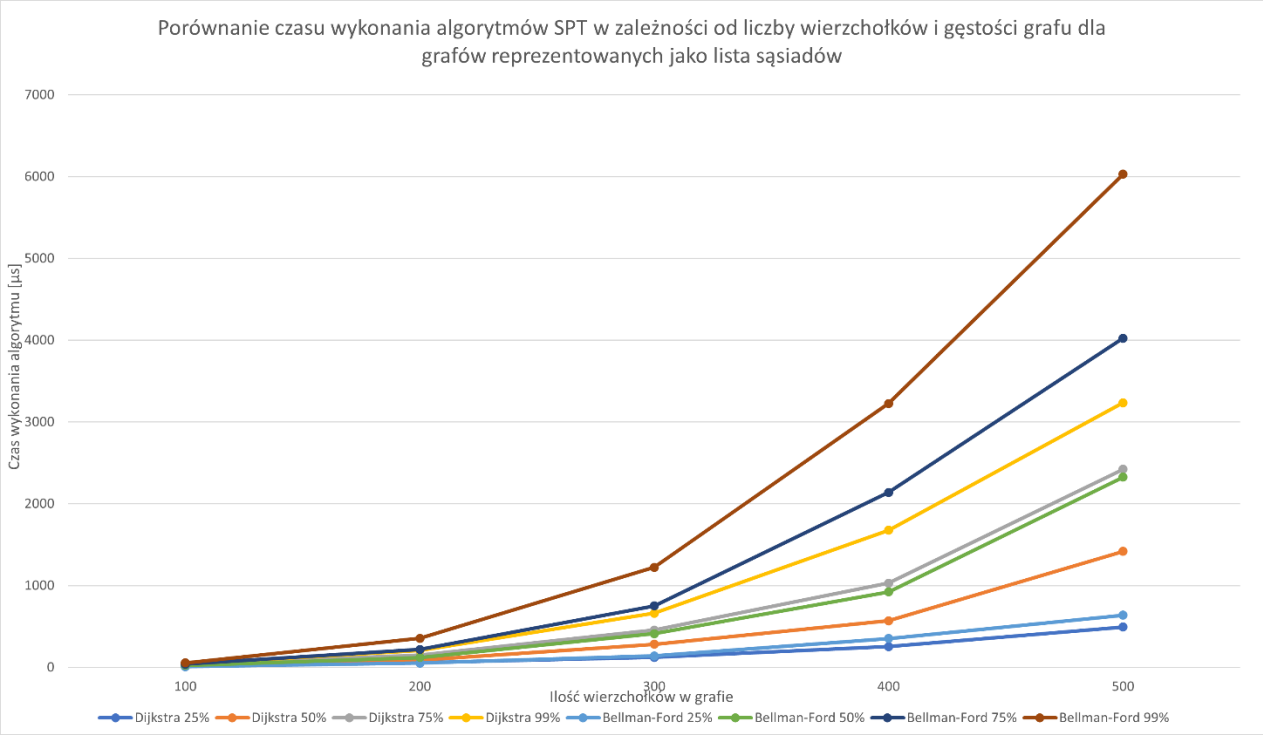
Tabela 7: Czas wykonania [µs] algorytmu Bellmana – Forda dla grafu reprezentowanego przez macierz sąsiedztwa w zależności od gęstości grafu i liczby wierzchołków

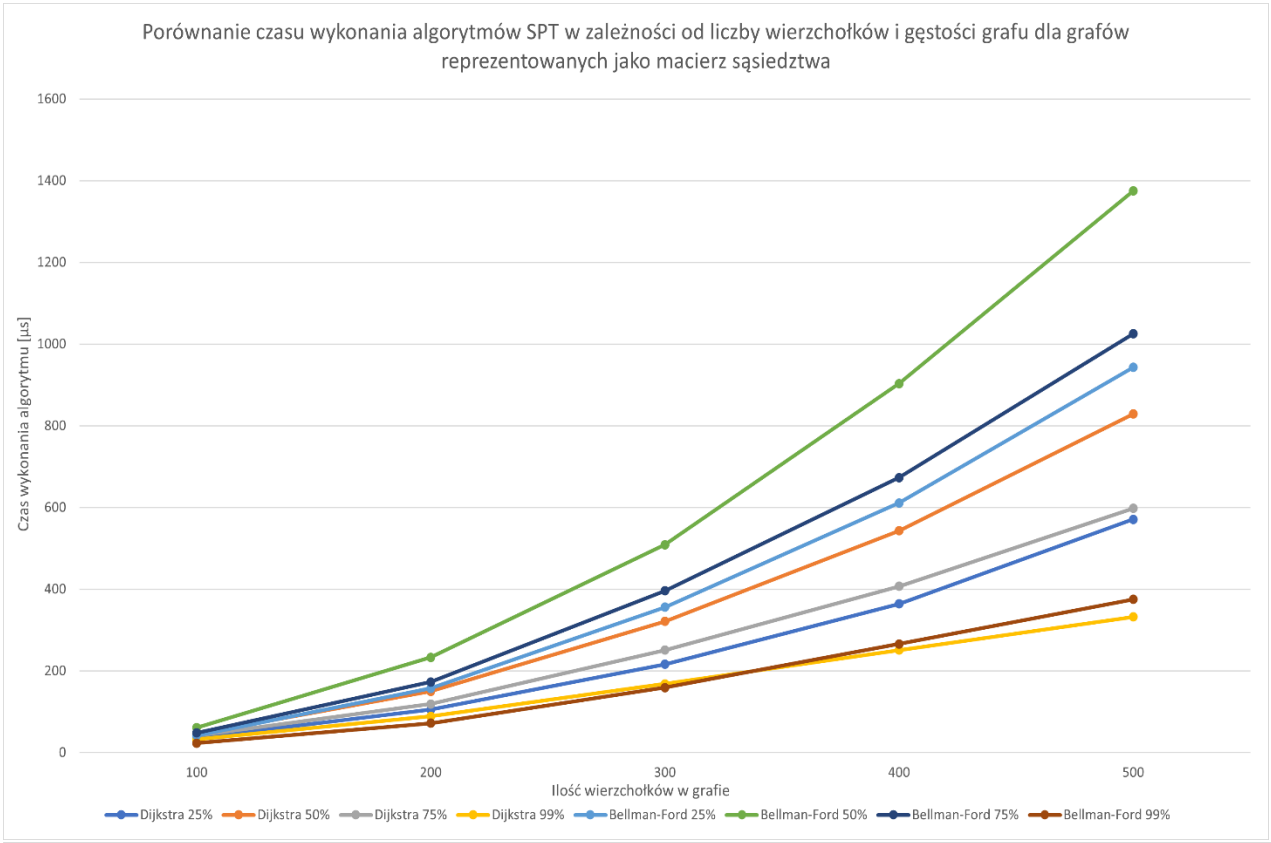
Implementacja z wykorzystaniem listy sąsiadów:

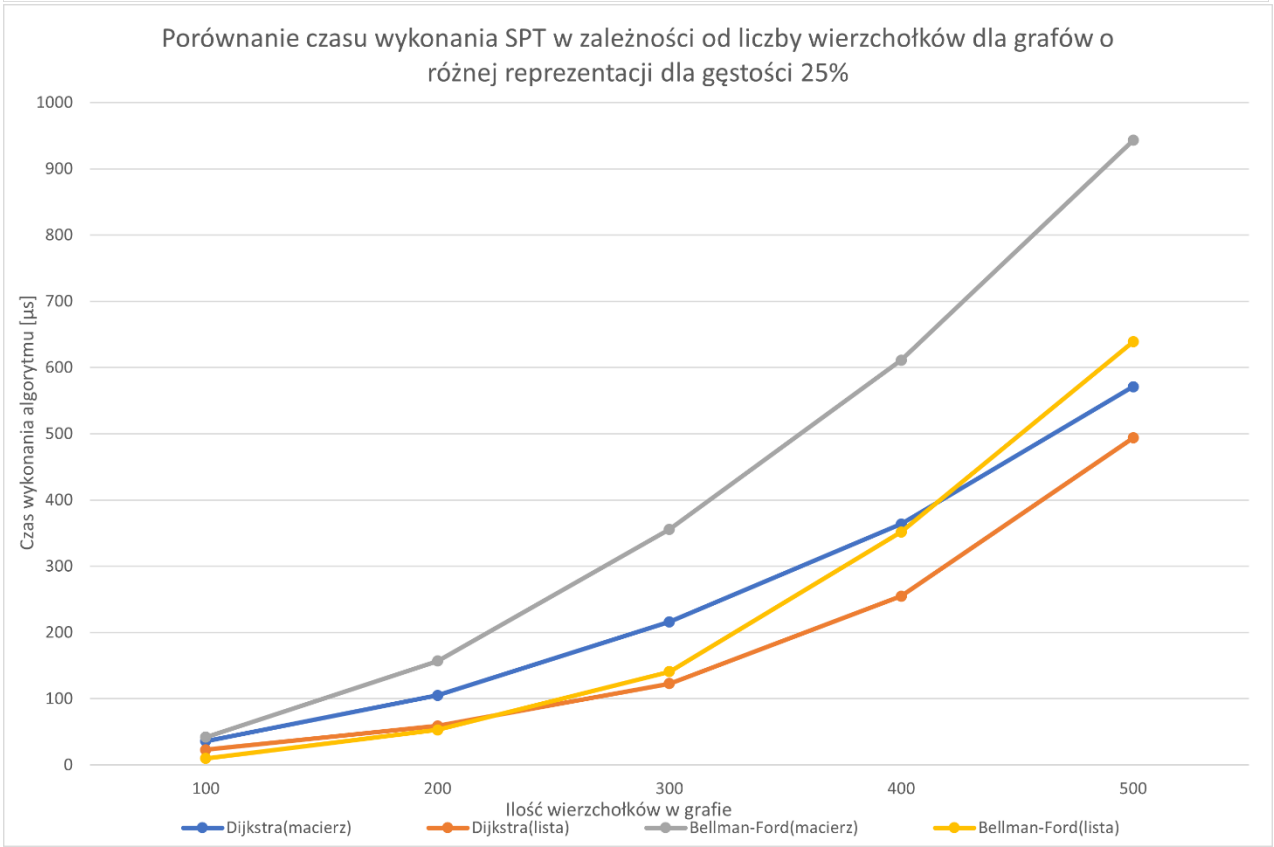
Gęstość grafu	Liczba wierzchołków				
	100	200	300	400	500
25%	10	53	141	352	639
50%	25	121	412	923	2325
75%	40	219	751	2138	4024
99%	54	354	1223	3227	6030

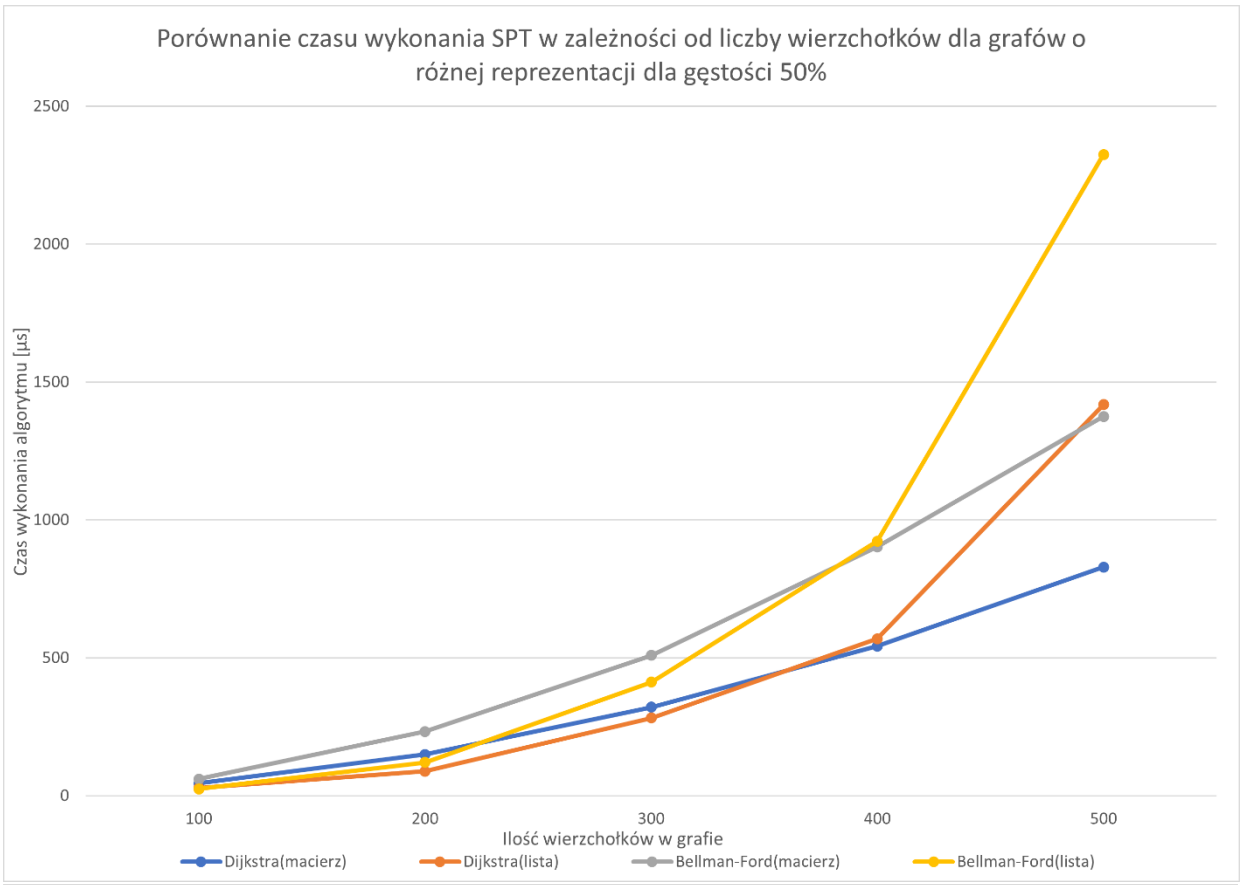
Tabela 8: Czas wykonania [µs] algorytmu Bellmana-Forda dla grafu reprezentowanego przez listę sąsiadów w zależności od gęstości grafu i liczby wierzchołków

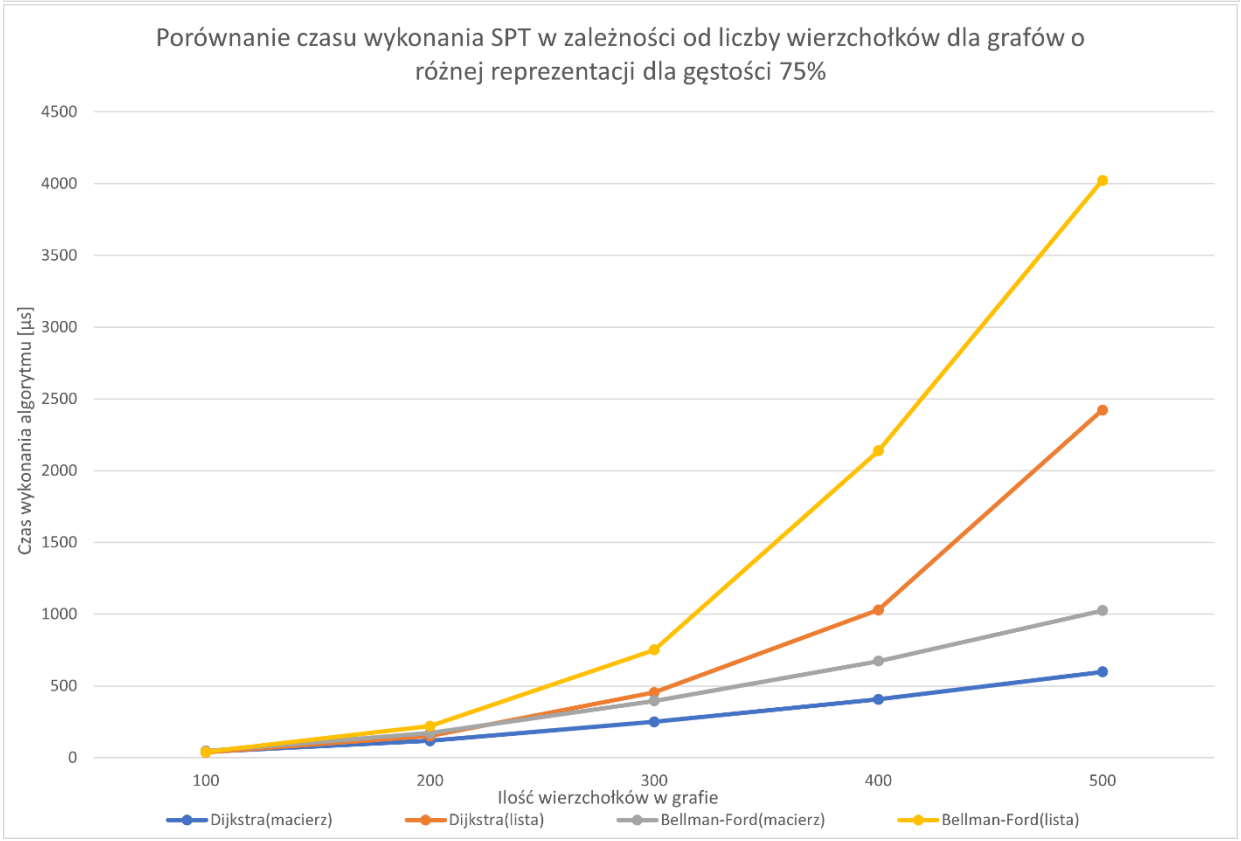
C. Wykresy

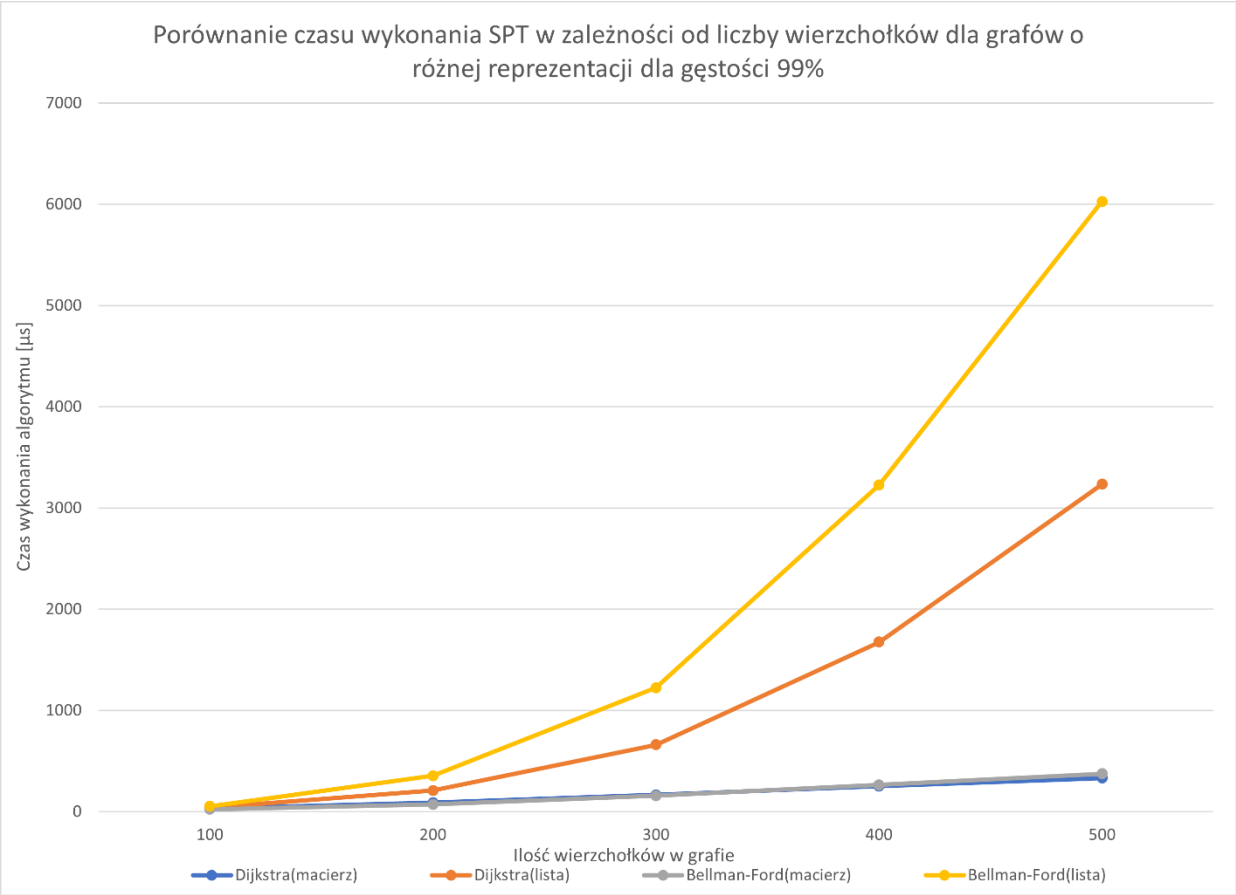



Porównanie czasu wykonania algorytmów SPT w zależności od liczby wierzchołków i gęstości grafu dla grafów reprezentowanych jako macierz sąsiedztwa
Czas wykonania algorytmu [µs]
1600
1400
1200
1000
800
600
400
200
0
100
200
300
400
500
Ilość wierzchołków w grafie
Dijkstra 25%
Dijkstra 50%
Dijkstra 75%
Dijkstra 99%
Bellman-Ford 25%
Bellman-Ford 50%
Bellman-Ford 75%
Bellman-Ford 99%


Porównanie czasu wykonania SPT w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 25%
Czas wykonania algorytmu [µs]
1000
900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
Ilość wierzchołków w grafie
Dijkstra(macierz)
Dijkstra(lista)
Bellman-Ford(macierz)
Bellman-Ford(lista)


Porównanie czasu wykonania SPT w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 50%
Czas wykonania algorytmu [µs]
2500
2000
1500
1000
500
0
100
200
300
400
500
Ilość wierzchołków w grafie
Dijkstra(macierz)
Dijkstra(lista)
Bellman-Ford(macierz)
Bellman-Ford(lista)


Porównanie czasu wykonania SPT w zależności od liczby wierzchołków dla grafów o różnej reprezentacji dla gęstości 75%
Czas wykonania algorytmu [µs]
4500
4000
3500
3000
2500
2000
1500
1000
500
0
100
200
300
400
500
Ilość wierzchołków w grafie
Dijkstra(macierz)
Dijkstra(lista)
Bellman-Ford(macierz)
Bellman-Ford(lista)



7. Wnioski

Otrzymane wyniki po części pokrywają się z założeniami teoretycznymi.

Rozbieżność w uzyskanych wynikach widać przy algorytmach działających na macierzy sąsiedztwa. Wraz ze wzrostem gęstości grafu czas wykonania zaczyna spadać. Możliwe, że jest to związane z kompilatorem, który jest w stanie lepiej zoptymalizować działanie programu w takich przypadkach. Problem ten nie pojawia się przy algorytmie Kruskala gdzie odczytujemy wartości z macierzy tylko raz w celu stworzenia tablicy z krawędziami.

Wyniki uzyskane dla reprezentacji grafu w postaci listy sąsiadów zachowują się zgodnie z założeniami. Widać jak duży wpływ na czas wykonania algorytmów ma ilość krawędzi przy tej reprezentacji grafu.

Dla algorytmów minimalnego drzewa rozpinającego jesteśmy w stanie zaobserwować, że wykonanie algorytmu Kruskala trwa znacząco dłużej w porównaniu do Prima. Różnica ta zwiększa się wraz ze wzrostem ilości wierzchołków i gęstości grafu a zatem dla większej ilości krawędzi. Algorytm Kruskala radzi sobie lepiej z rzadkimi grafami jednak ciężko to zauważyć, ponieważ najmniejsza badana gęstość to 25%.

Dla algorytmów najkrótszej drogi w grafie jesteśmy w stanie zauważyć, że algorytm Dijkstry jest szybszy od algorytmu Bellmana – Forda dla grafów o tych samych parametrach. Jest to związane z faktem, że algorytm Bellmana – Forda jest sobie w stanie poradzić z grafem zawierającym krawędzie o ujemnych wagach co jest nie możliwe do wykonania korzystając z algorytmu Dijkstry.

Patrząc na zestawienie czasów wykonania algorytmów ze względu na reprezentację grafu w pamięci komputera można zauważyć, że macierz sąsiedztwa radzi sobie lepiej z grafami, które są gęstsze. Lista sąsiadów jest lepsza dla grafów o mniejszej gęstości, które można częściej spotkać w prawdziwych problemach. Ponieważ lista sąsiadów przechowuje jedynie informację o istniejących krawędziach to przy rzadkich i dodatkowo skierowanych grafach pozwala nam zaoszczędzić dużą ilość pamięci w porównaniu do macierzy sąsiedztwa.