SPRAWOZDANIE

STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

ZADANIE PROJEKTOWE NR 2

BADANIE EFEKTYWNOŚCI ALGORYTMÓW GRAFOWYCH W ZALEŻNOŚCI OD ROZMIARU ORAZ SPOSOBU REPREZENTACJI GRAFU W PAMIĘCI KOMPUTERA

Autor:
Patryk Dulęba 259213

Prowadzący:
Dr inż. Dariusz Banasiak

1 Wstęp teoretyczny

1.1 Cel

Zadanie projektowe polegało na zaimplementowaniu oraz dokonaniu pomiaru czasu wykonywania algorytmów grafowych dla grafów przechowywanych w postaci **macierzy incydencji** oraz **listy sąsiedztwa**.

Badane algorytmy:

- Wyznaczanie minimalnego drzewa rozpinającego (MST)
  1. algorytm Prima
  2. algorytm Kruskala
- Wyznaczanie najkrótszej ścieżki w grafie
  1. algorytm Dijkstry
  2. algorytm Bellmana-Forda

1.2 Założenia

Przed implementacją zostały podane założenia co do sposobu wykonywania zadania:

- Program został napisany w języku **C++** przy użyciu środowiska **Visual Studio**
- Wszystkie struktury danych są alokowane dynamiczne
- Wagami krawędzi są liczby całkowite typu **int**

Implementacja Macierzy:

- Macierz jest dwuwymiarową tablicą alokowaną dynamicznie
- Kolumny odpowiadają numerom krawędzi, natomiast wiersze numerom wierzchołków
- Macierz wypełniona jest następującymi wartościami:
  - **0** Brak połączenia wierzchołka z krawędzią
  - **1** Krawędź wychodzi z danego wierzchołka
  - **-1** Krawędź dociera do danego wierzchołka

Implementacja Listy:

- Lista została zaimplementowana jako dynamiczna tablica struktur odpowiadająca konkretnym wierzchołkom grafu
- Struktura pojedynczego wierzchołka zawiera: numer wierzchołka, z którym jest połączony; wagę krawędzi, która je łączy oraz wskaźnik na kolejny sąsiadujący wierzchołek.

1.3 Złożoność obliczeniowa

Badane algorytmy wykonywane na zadanych strukturach posiadają tzw. książkową złożoność obliczeniową, czyli według definicji ilość zasobów komputerowych potrzebnych do jego wykonania. Złożoność możemy podzielić na trzy typy:

- Złożoność optymistyczna – najkrótszy możliwy czas wykonania algorytmu.
- Złożoność średnia – standardowy czas wykonywania algorytmu.
- Złożoność pesymistyczna – najdłuższy możliwy czas wykonania algorytmu.

Legenda oznaczeń:

- **E** liczba krawędzi w grafie
- **V** liczba wierzchołków w grafie

1.3.1 Złożoność algorytmu Prima

O rzędzie złożoności decyduje implementacja kolejki priorytetowej:

- dla implementacji poprzez zwykłą tablicę złożoność wynosi $\mathbf{O(E \cdot V)}$
- dla implementacji kolejki poprzez kopie, złożoność wynosi $\mathbf{O(E \cdot log(V))}$

1.3.2 Złożoność algorytmu Kruskala

Niezależnie od implementacji złożoność obliczeniowa algorytmu Kruskala jest równa:
$\mathbf{O(E \cdot log(V))}$

1.3.3 Złożoność algorytmu Dijkstry

O rzędzie złożoności decyduje implementacja kolejki priorytetowej:

- dla implementacji poprzez zwykłą tablicę złożoność wynosi $\mathbf{O}(V^2)$
- dla implementacji kolejki poprzez kopie, złożoność wynosi $\mathbf{O(E \cdot log(V))}$

1.3.4 Złożoność algorytmu Bellmana-Forda

Niezależnie od implementacji złożoność obliczeniowa algorytmu Bellmana-Forda jest równa:
$\mathbf{O(E \cdot V)}$

2 Plan eksperymentu

W zaimplementowanym programie występuje podział na dwa podprogramy, osobny dla macierzy incydencji i osobny dla listy sąsiedztwa. Dla każdej struktury poza wykonaniem wymaganych algorytmów istnieje możliwość wczytania grafu z pliku tekstowego, oraz automatyczne wygenerowanie grafu z określoną liczbą wierzchołków, oraz gęstością. Dodatkowo każdy graf możemy wyświetlić na ekranie w postaci danej struktury. Pomiar czasu został zaimplementowany za pomocą osobnej funkcji testującej.

Do przeprowadzenia eksperymentu zostały użyte następujące wielkości grafu:

- Liczba wierzchołków: **10, 25, 50, 75, 100**
- Gęstość grafu: **25%, 50%, 75%, 99%**

Pomiar czasu poszczególnych operacji został wykonany za pomocą:
QueryPerformanceCounter.

Przebieg pomiaru wyglądał w następujący sposób:

1. Podanie liczby wierzchołków oraz gęstości grafu
2. Generowanie grafu o podanych parametrach
3. Wykonanie danego algorytmu wraz z pomiarem czasu
4. Dodanie uzyskanego pomiaru czasu do zmiennej liczącej

Po 50 krotnym wykonaniu powyższych operacji funkcja liczyła i wypisywała średnią

3 Generowanie grafu

Do generowania grafu zostały napisane dwie funkcje, jedna generuje graf nieskierowany dla algorytmów MST, druga skierowany dla algorytmów wyszukiwania najkrótszej drogi. Zasada działania obu jest taka sama. Początkowo z otrzymanych danych wyliczana zostaje liczba potrzebnych krawędzi. Następnie tworzona zostaje tablica przechowująca numery wszystkich wierzchołków, później za pomocą pętli wartości tablicy zostają pomieszane. W kolejnym kroku zostają utworzone krawędzie łączące kolejne wierzchołki zapisane w tablicy. Opisane działania gwarantują nam, iż otrzymany graf z pewnością będzie spójny. Na koniec za pomocą funkcji **rand()** generowane zostają pozostałe krawędzie aż do osiągnięcia wymaganej gęstości.

4 Wyniki pomiarów

4.1 Wyszukiwanie minimalnego drzewa rozpinającego algorytmem Prima oraz Kruskala

	25%	50%	75%	99%
10	14	28	34	39
25	107	203	279	351
50	690	1333	2031	2742
75	2314	6721	8322	12513
100	6892	15710	28580	43010

Tabela 1: Pomiary czasu[μs] algorytmem Prima - graf w postaci macierzy incydencji

	25%	50%	75%	99%
10	15	13	11	14
25	34	43	54	59
50	94	124	160	191
75	169	259	331	411
100	291	416	582	740

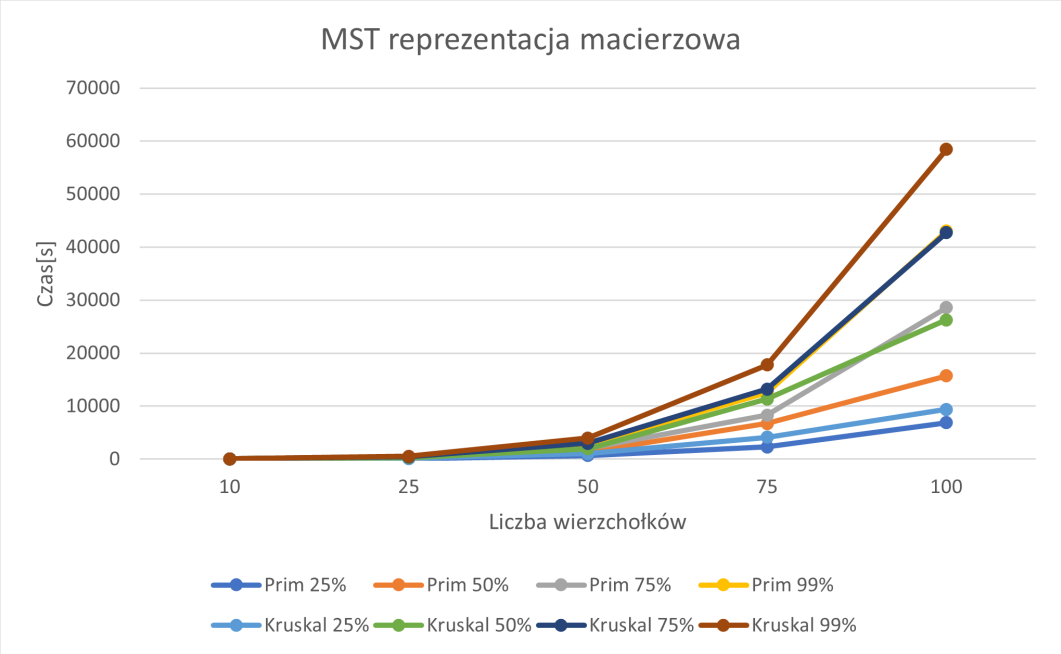
Tabela 2: Pomiary czasu[μs] algorytmem Prima - graf w postaci listy sąsiedztwa

	25%	50%	75%	99%
10	23	36	50	53
25	171	306	421	538
50	1023	1962	3013	3986
75	4117	11355	13204	17784
100	9360	26261	42715	58461

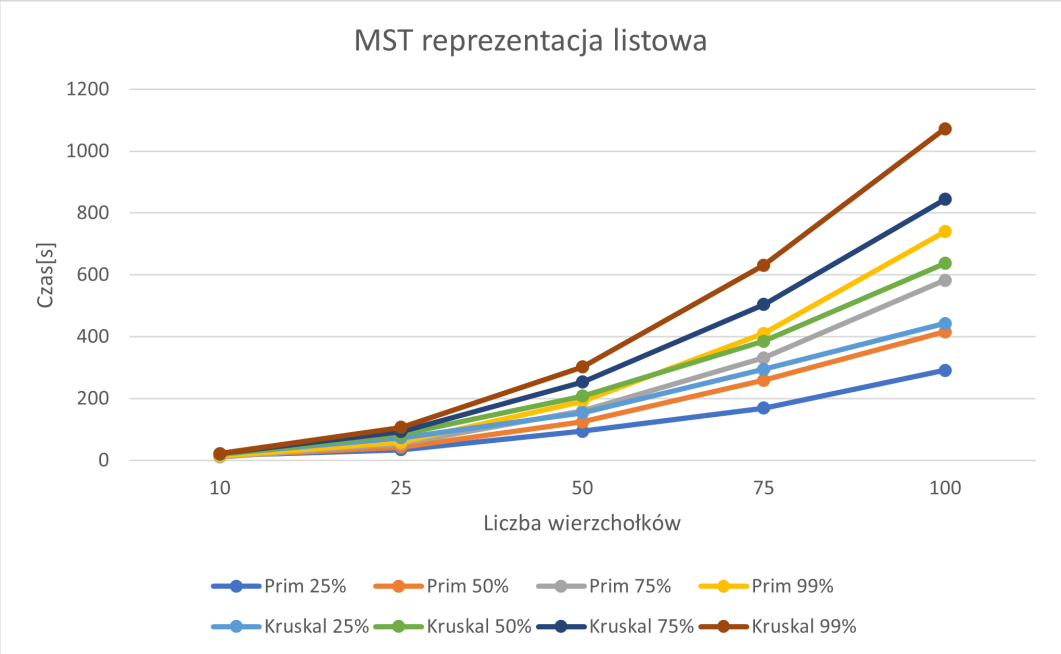
Tabela 3: Pomiary czasu[μs] algorytmem Kruskala - graf w postaci macierzy incydencji

	25%	50%	75%	99%
10	20	19	22	22
25	73	84	93	107
50	154	208	253	302
75	294	385	504	631
100	443	637	844	1072

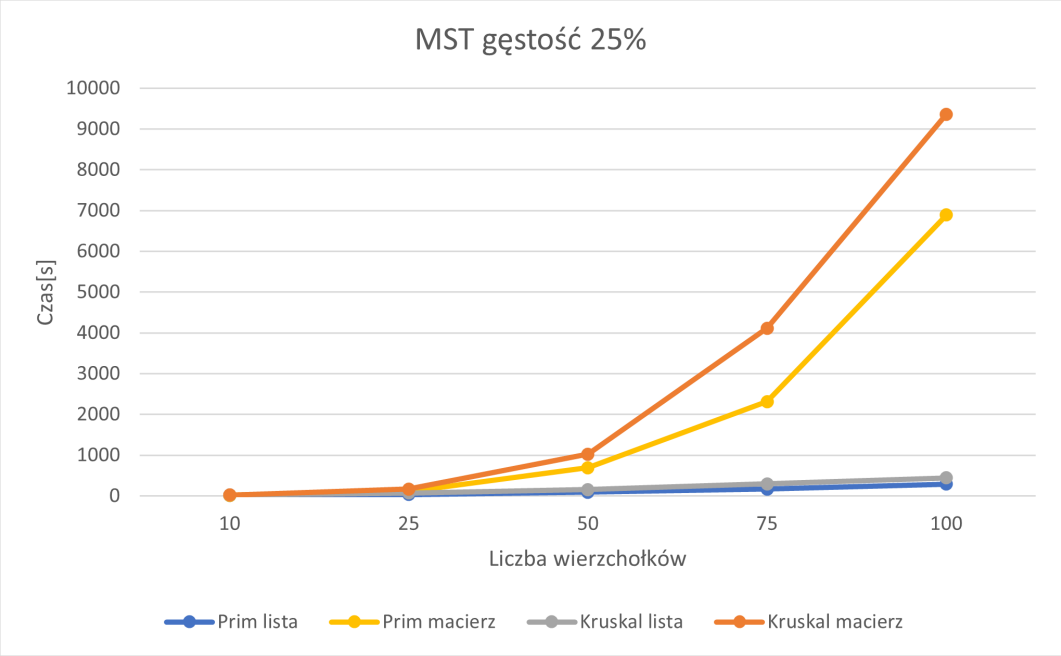
Tabela 4: Pomiary czasu[μs] algorytmem Kruskala - graf w postaci listy sąsiedztwa



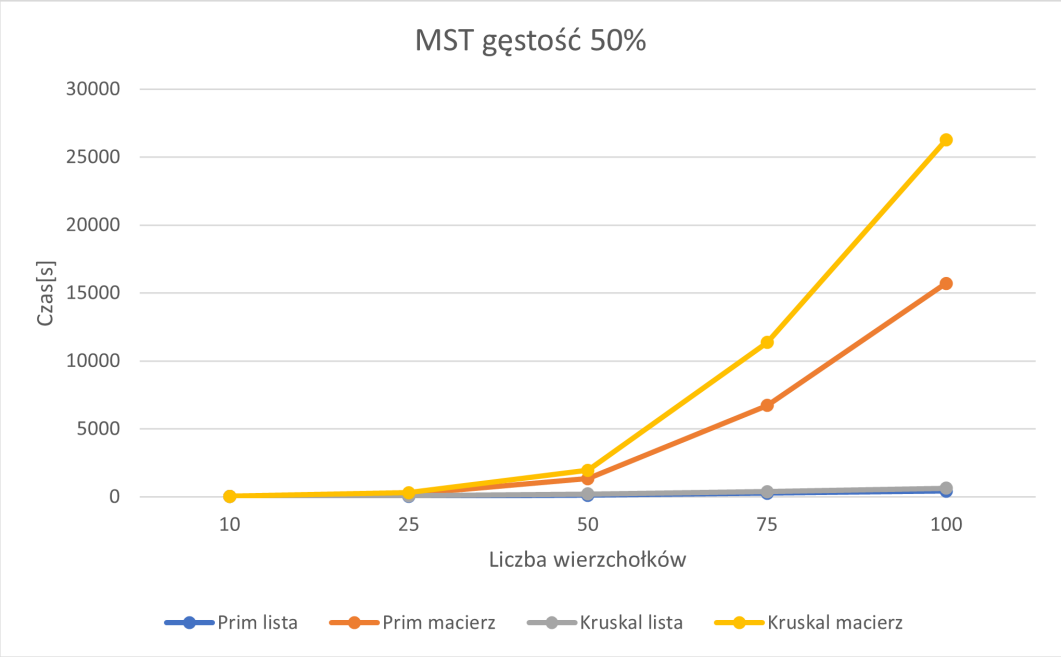
Rysunek 1: MST reprezentacja macierzowa



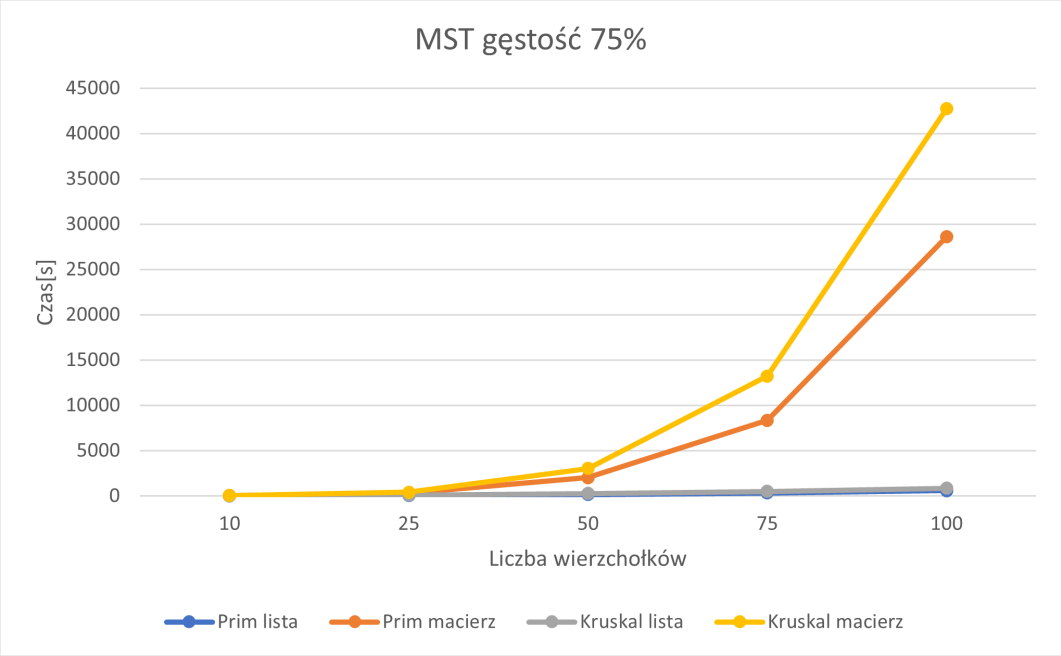
Rysunek 2: MST reprezentacja listowa



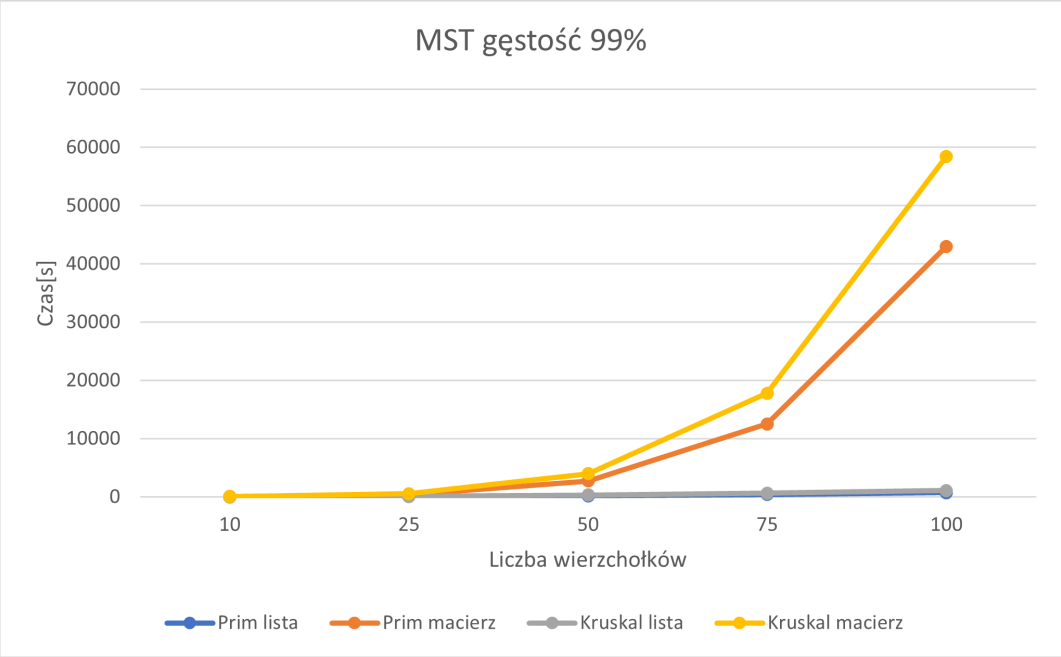
Rysunek 3: MST przy gęstości grafu 25%



Rysunek 4: MST przy gęstości grafu 50%



Rysunek 5: MST przy gęstości grafu 75%



Rysunek 6: MST przy gęstości grafu 99%

4.2 Wyszukiwanie najkrótszej drogi w grafie algorytmem Dijkstry oraz Bellmana-Forda

	25%	50%	75%	99%
10	5	7	10	10
25	30	44	62	80
50	154	289	427	561
75	543	997	1529	1958
100	1191	2355	4163	6403

Tabela 5: Pomiary czasu[μs] algorytmem Dijkstry - graf w postaci macierzy incydencji

	25%	50%	75%	99%
10	3	5	6	4
25	10	12	11	12
50	27	29	33	38
75	53	60	67	82
100	86	98	116	134

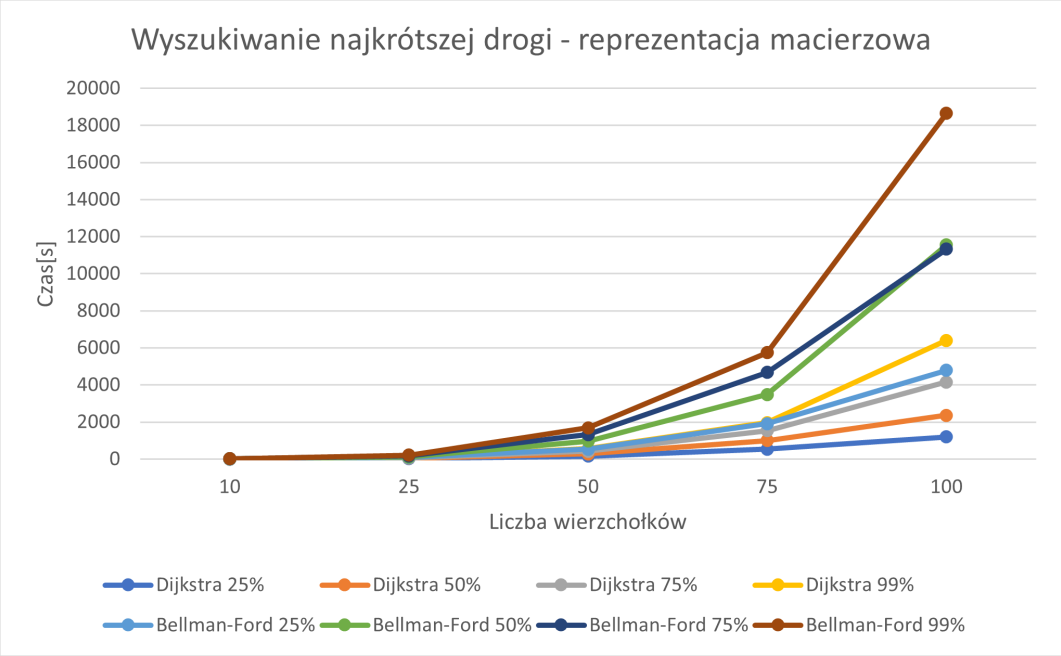
Tabela 6: Pomiary czasu[μs] algorytmem Dijkstry - graf w postaci listy sąsiedztwa

	25%	50%	75%	99%
10	6	14	16	15
25	79	122	174	218
50	534	973	1320	1682
75	1905	3481	4674	5742
100	4781	11557	11324	18645

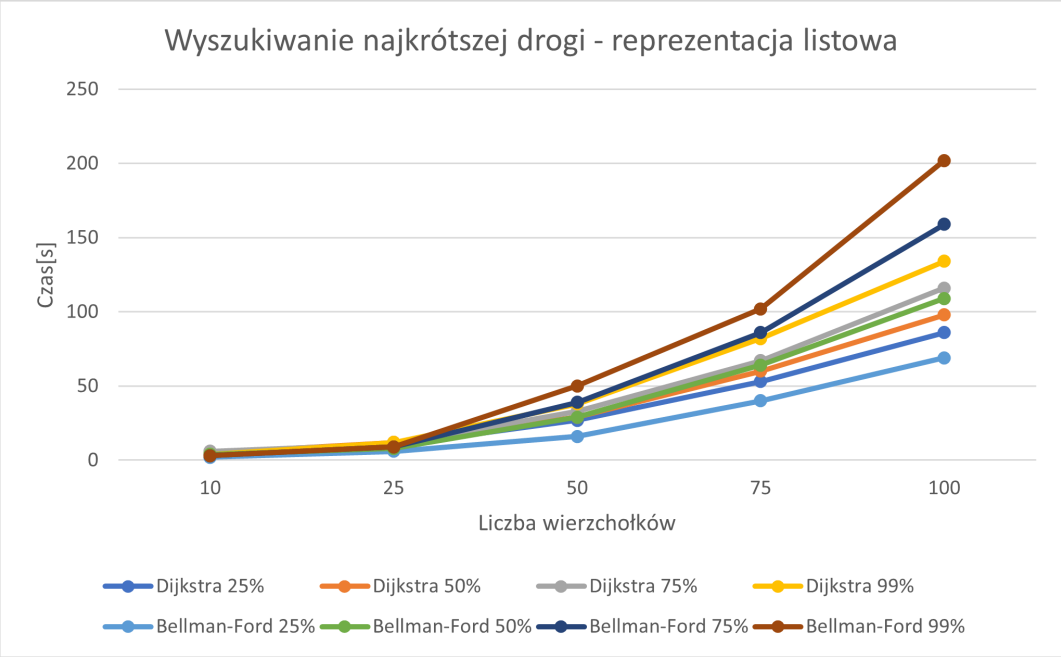
Tabela 7: Pomiary czasu[μs] algorytmem Bellmana-Forda - graf w postaci macierzy incydencji

	25%	50%	75%	99%
10	2	4	3	3
25	6	8	9	9
50	16	29	39	50
75	40	64	86	102
100	69	109	159	202

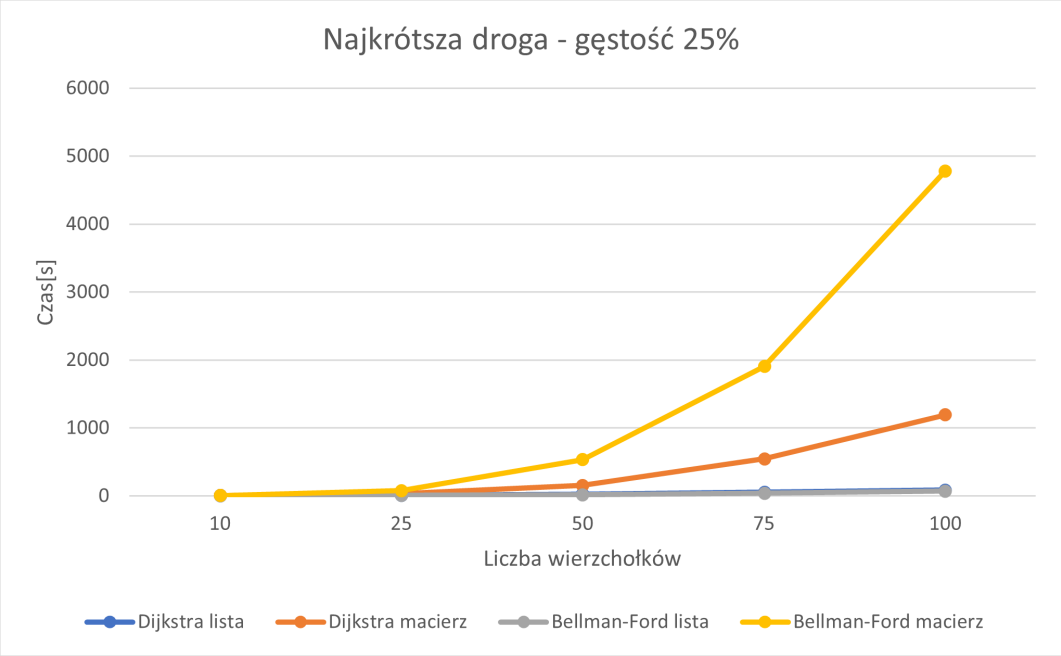
Tabela 8: Pomiary czasu[μs] algorytmem Bellmana-Forda - graf w postaci listy sąsiedztwa



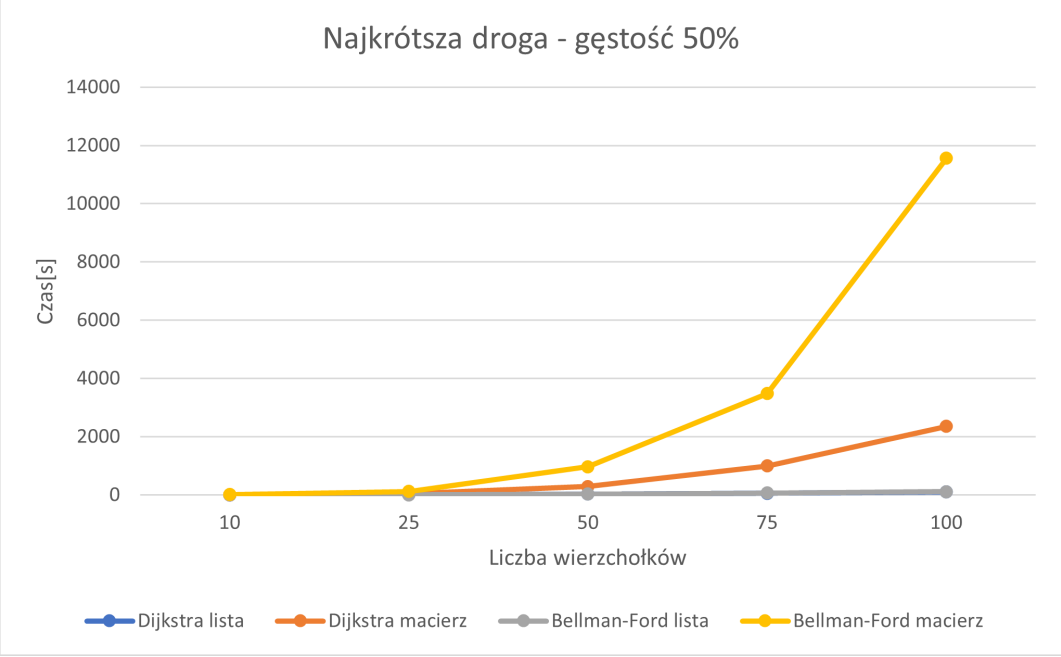
Rysunek 7: Wyszukiwanie najkrótszej drogi w grafie - reprezentacja macierzowa



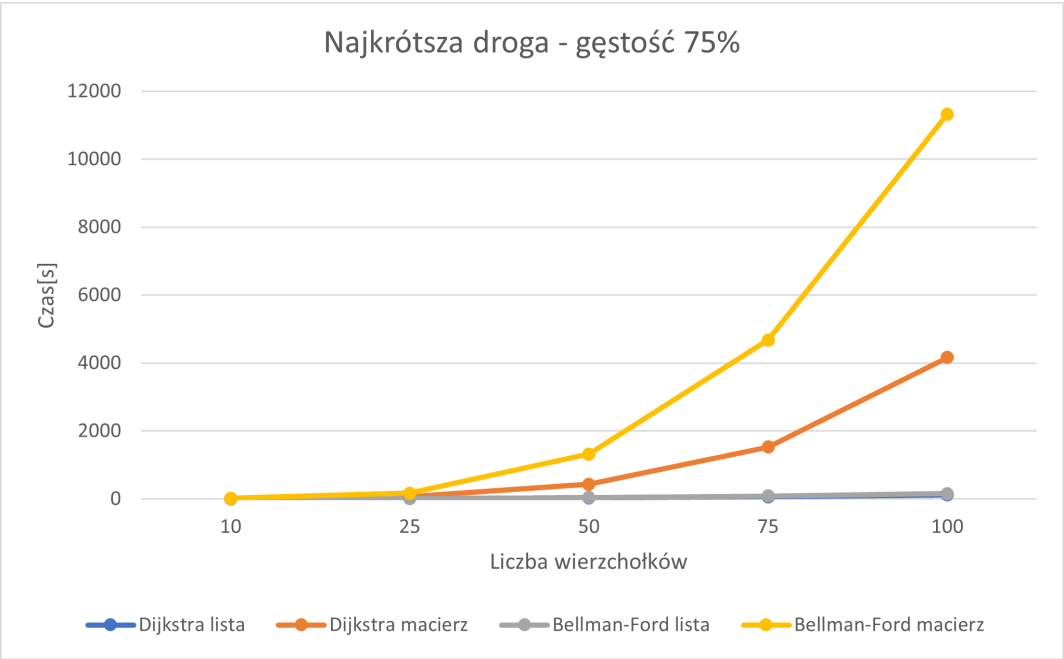
Rysunek 8: Wyszukiwanie najkrótszej drogi w grafie - reprezentacja listowa



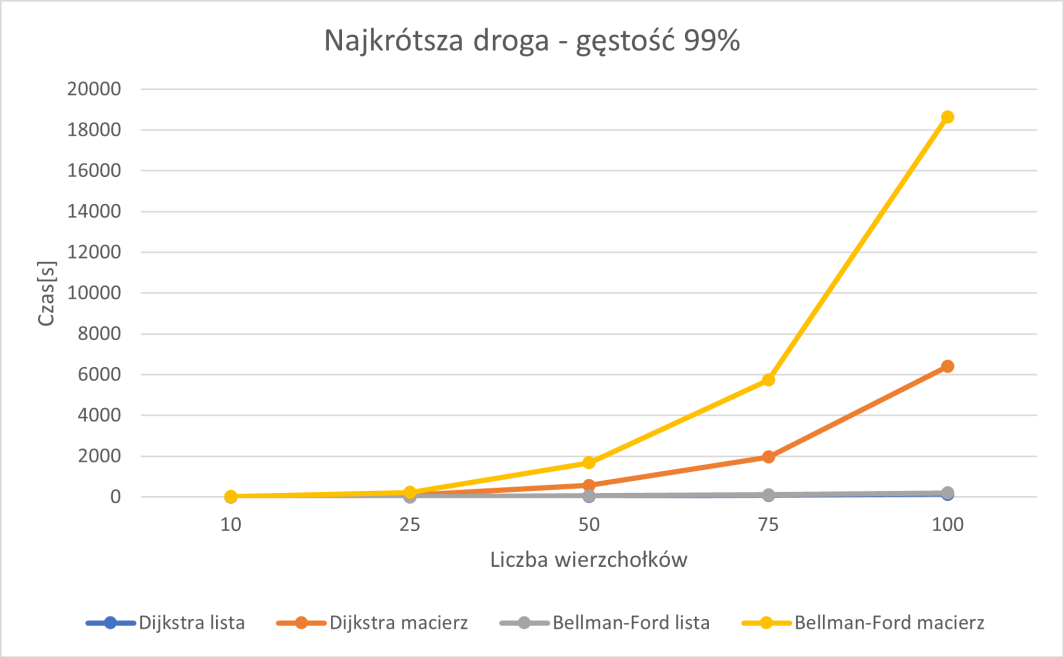
Rysunek 9: Wyszukiwanie najkrótszej drogi w grafie przy gęstości grafu 25%



Rysunek 10: Wyszukiwanie najkrótszej drogi w grafie przy gęstości grafu 50%



Rysunek 11: Wyszukiwanie najkrótszej drogi w grafie przy gęstości grafu 75%



Rysunek 12: Wyszukiwanie najkrótszej drogi w grafie przy gęstości grafu 99%

5 Wnioski

Na podstawie otrzymanych pomiarów można stwierdzić, iż wykonywanie poszczególnych algorytmów na liście sąsiedztwa jest znacznie szybsze niż na macierzy incydencji. Spowodowane jest to specyfiką budowy każdej ze struktur. Lista umożliwia nam szybkie poruszanie się między sąsiednimi wierzchołkami poprzez wskaźniki, natomiast przy implementacji za pomocą macierzy incydencji, aby znaleźć sąsiadujący wierzchołek, musimy przeszukiwać całą macierz, czyli tablicę dwuwymiarową, co zajmuje znacznie więcej czasu. Otrzymane za pomocą wykresów złożoności obliczeniowe w niektórych przypadkach różnią się od tych książkowych, co wynika z błędów podczas implementacji poszczególnych algorytmów.