Struktury Danych i Złożoność Obliczeniowa

Zadanie projektowe 2:

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

Nikita Stepanenko

Nr. albomu: 245816

Spis treści

1. Opis zadania projektowego

W zadaniu projektowym należało zaimplementować dwie metody przechowywania grafów:

1) Macierz incydencji – Tablica dwuwymiarowa o rozmiarze n na m, gdzie n jest ilością wierzchołków grafu, a m ilością jego krawędzi. Pozwala ona w prosty sposób przedstawić graf skierowany. Macierz taka wypełnia się wartościami zgodnie z zasadą:

   1, jeżeli wierzchołek jest początkiem krawędzi

   0, jeżeli wierzchołek nie jest incydentny

   -1, jeżeli wierzchołek jest końcem krawędzi

2) Lista sąsiadów – Jest to jednowymiarowa tablica, zawierająca n innych list. Każdy element tablicy odpowiada wierzchołkowi grafu, natomiast elementami danej listy są wszystkie wierzchołki sąsiadujące z tym wierzchołkiem. Kolejność sąsiadów na liście nie ma znaczenia.

Należało zaimplementować algorytm generujący losowy graf, zarówno w postaci macierzowej jaki i listowej, oraz możliwość wczytywania grafu z pliku. Dla każdej reprezentacji grafu należało zaimplementować:

1) Algorytm Dijkstry – Służy do znajdowania w grafie najkrótszej ścieżki pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi, wyliczając również koszt przejścia każdej z tych ścieżek, czyli sumę wag krawędzi na ścieżce.
2) Algorytm Bellmana-Forda - algorytm służący do wyszukiwania najkrótszych ścieżek w grafie ważonym z wierzchołka źródłowego do wszystkich pozostałych wierzchołków. W odróżnieniu od algorytmu Dijkstry, algorytm Bellmana-Forda działa poprawnie także dla grafów z wagami ujemnymi.
3) Algorytm Prima – Wykorzystywany do określenia minimalnego drzewa rozpinającego. Na początku, algorytm dodaje do zbioru reprezentującego drzewo krawędź o najmniejszej wadze, łączącą wierzchołek początkowy z dowolnym wierzchołkiem. W każdym kolejnym kroku procedura dodaje do zbioru najlżejszą krawędź wśród krawędzi łączących wierzchołki już odwiedzone z nieodwiedzonymi.

2. Złożoności obliczeniowe operacji

1) Algorytm Dijkstry - Złożoność obliczeniowa algorytmu Dijkstry zależy od liczby wierzchołków (V) i krawędzi (E) grafu. O rzędzie złożoności decyduje implementacja kolejki priorytetowej:
    - wykorzystując "naiwną" implementację poprzez zwykłą tablicę, otrzymujemy algorytm o złożoności O(V * V)
    - w implementacji kolejki poprzez kopiec, złożoność wynosi O(E * log(V))
2) Algorytm Bellmana-Forda - Złożoność obliczeniowa algorytmu Dijkstry zależy od liczby wierzchołków (V) i krawędzi (E) grafu. Idea algorytmu opiera się na metodzie relaksacji (dokładniej następuje relaksacja razy każdej z krawędzi)O rzędzie złożoności decyduje implementacja kolejki priorytetowej:
    - Złożoność obliczeniowa pesymistyczna O(E*V)
3) Algorytm Prima - Złożoność obliczeniowa w zależności od implementacji kolejki priorytetowej, gdzie V jest ilością wierzchołków grafu, a E ilością jego krawędzi:
    - Dla wersji opartej na zwykłym kopcu (bądź drzewie czerwono-czarnym) O(E * log(V))
    - Przy zastosowaniu kopca Fibonacciego O(E + V * log(V))

3. Metoda pomiaru czasu

Do pomiaru czasu wykorzystana został biblioteka „Chrono.h”, zawarta w zbiorze standardowych bibliotek języka C++. Biblioteka ta operuje na czasie procesora, co pozwala na uzyskanie dokładności czasowej równej jednemu okresowi przebiegu procesora. Dla pomiarów stosowany czas 1 nanosekundy. Klasa pomiarowa zawarta w programie, została umieszczona w pliku „Time.h”, a jej implementacja prezentuje się w pliku „Time.cpp”.

Time.h

```
#include <chrono>

using namespace std;
using namespace std::chrono;

class Czas {
public:
    high_resolution_clock::time_point czasPoczatkowy;
    high_resolution_clock::time_point czasKoncowy;

    void czasStart();

    void czasStop();

    long czasWykonania();
};
```

Time.cpp

```
#include "Czas.h"

using namespace std;
using namespace std::chrono;

void Czas::czasStart() {
    czasPoczatkowy = high_resolution_clock::now();
}

void Czas::czasStop() {
    czasKoncowy = high_resolution_clock::now();
}

long Czas::czasWykonania() {

    return duration_cast<nanoseconds>(Czas::czasKoncowy -
Czas::czasPoczatkowy).count();
}
```

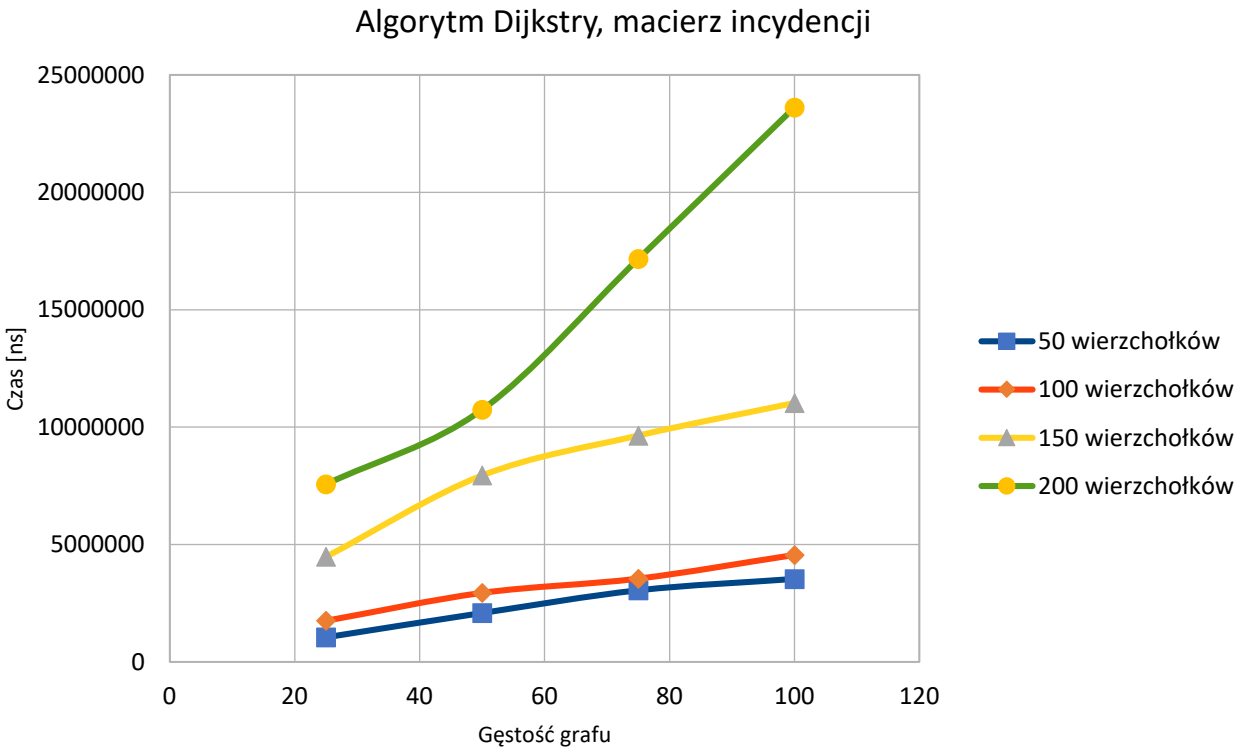
Program pobiera czas przed wykonaniem funkcji i po jej wykonaniu, a następnie oblicza różnicę czasu bazując na ilości okresów przebiegu procesora, które miały miejsce pomiędzy oboma zarejestrowanymi czasami. W wyniku zwraca czas w nanosekundach.

4. Pomiary

Pomiary dokonane dla grafów zawierających 50, 100, 150 i 200 wierzchołków, o gestościach 25%, 50%, 75%, 100%. Uśrednione wyniki wszystkich pomiarów przedstawiają tabele oraz wykresy umieszczone poniżej.

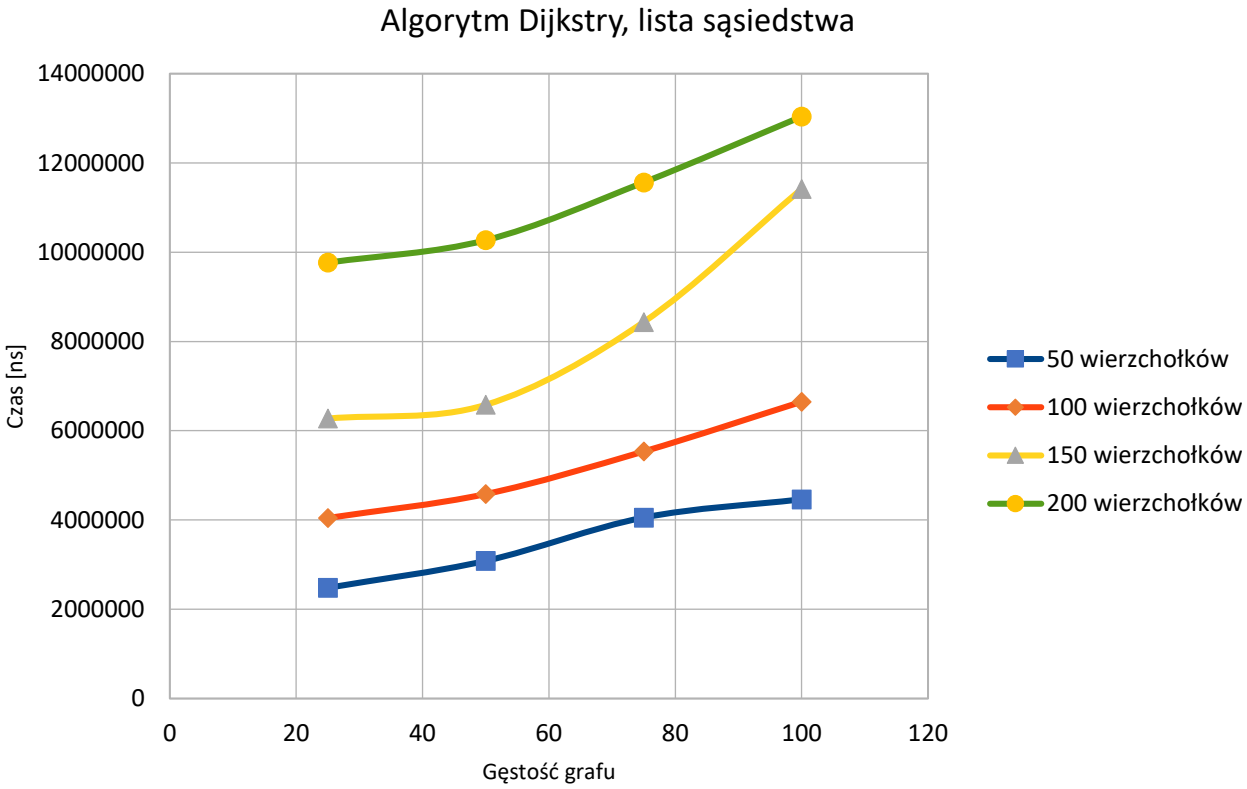
4.1 Algorytm Dijkstry dla implementacji macierzowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	1048000
	50	2083200
	75	3049400
	100	3534400
100	25	1752500
	50	2940600
	75	3547300
	100	4556200
150	25	4476100
	50	7941200
	75	9640400
	100	11027700
200	25	7564400
	50	10742700
	75	17161200
	100	23614700



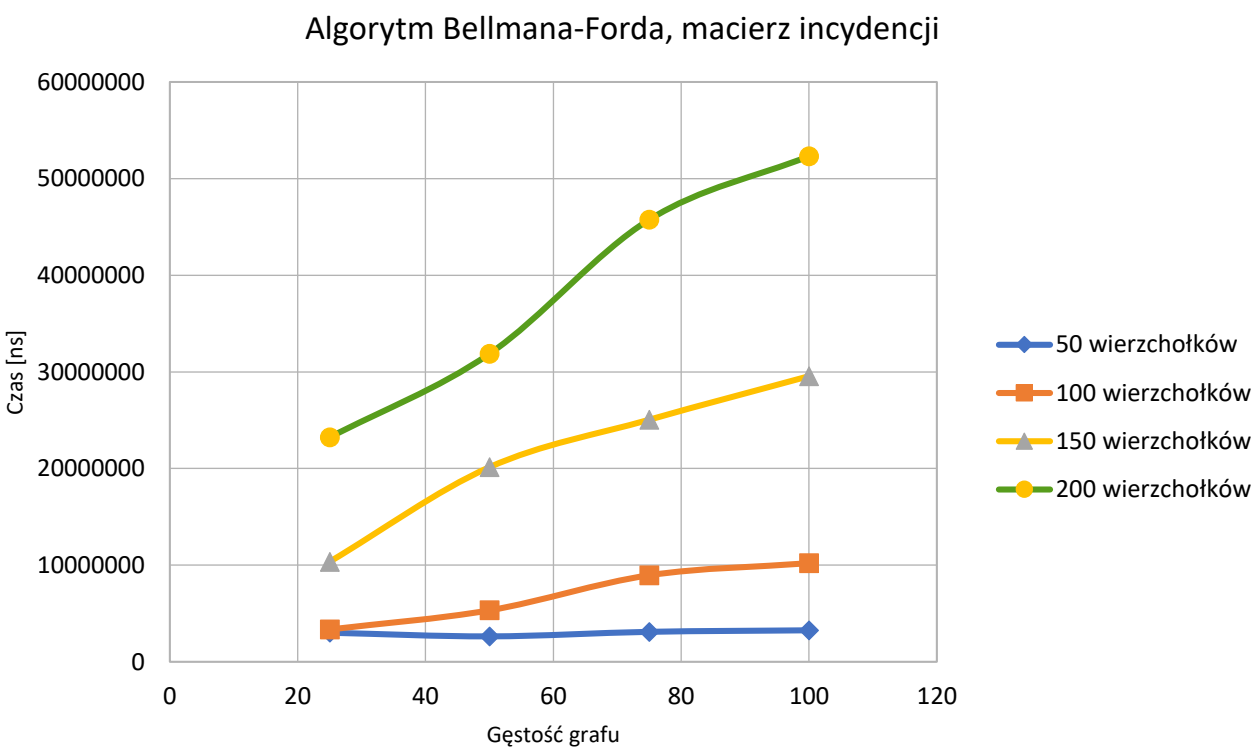
4.2 Algorytm Dijkstry dla implementacji listowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	2480900
	50	3084800
	75	4054200
	100	4463500
100	25	4044600
	50	4581400
	75	5535200
	100	6649600
150	25	6272900
	50	6580800
	75	8435000
	100	11419100
200	25	9765600
	50	10268400
	75	11564300
	100	13038800



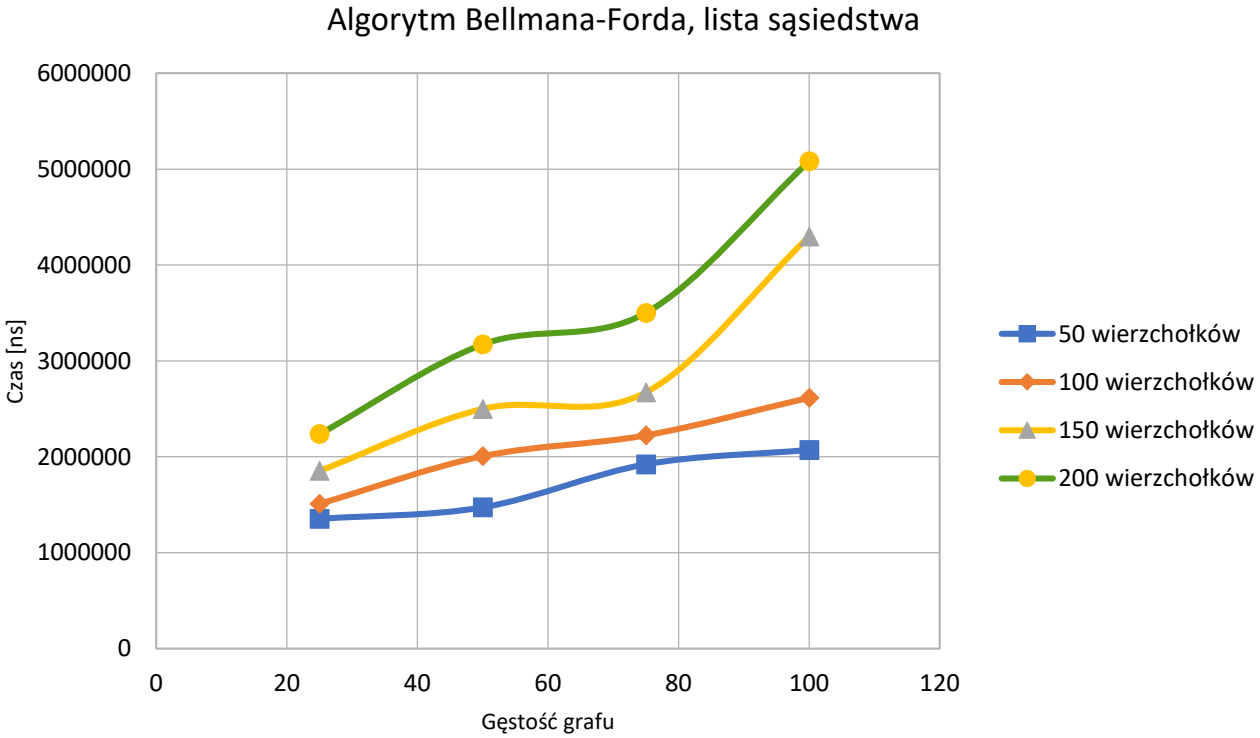
4.3 Algorytm Bellmana-Forda dla implementacji macierzowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	3004800
	50	2628100
	75	3097800
	100	3255000
100	25	3340400
	50	5322000
	75	8952200
	100	10201700
150	25	10349900
	50	20134700
	75	25049900
	100	29554100
200	25	23215300
	50	31868400
	75	45751900
	100	52321200



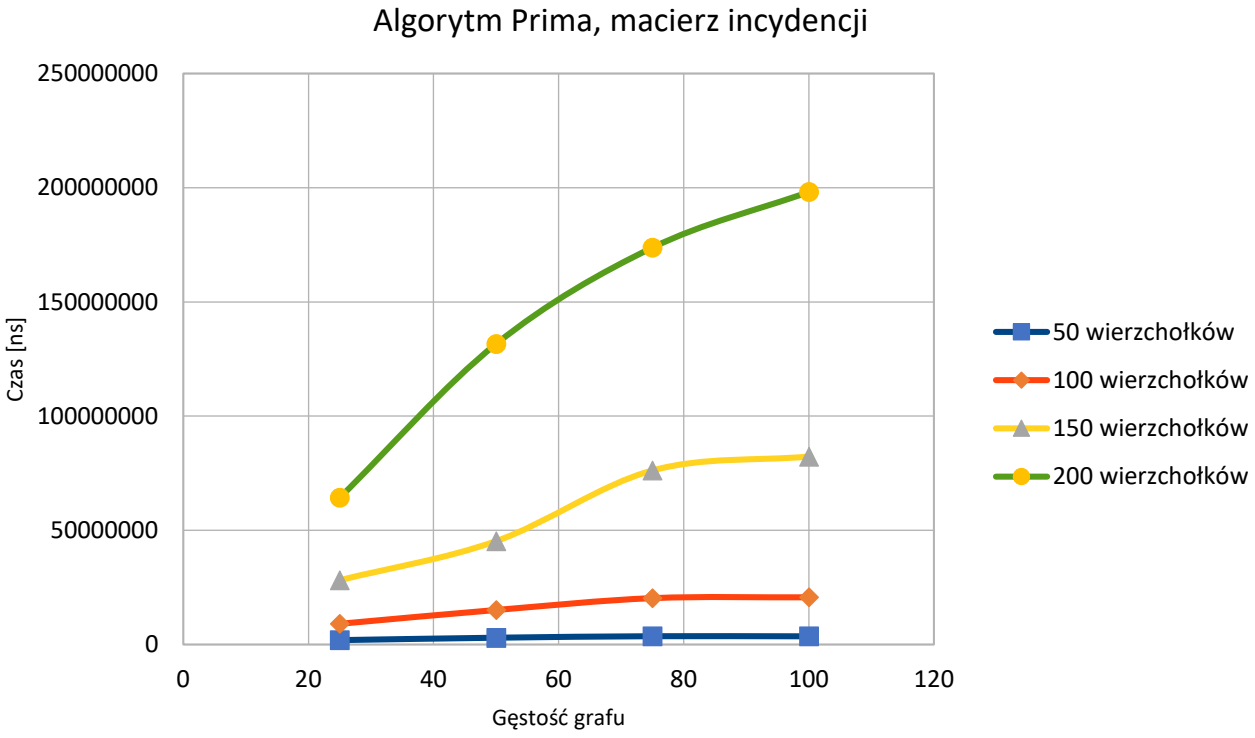
4.4 Algorytm Bellmana-Forda dla implementacji listowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	1353000
	50	1472500
	75	1921600
	100	2071700
100	25	1509200
	50	2006100
	75	2223600
	100	2615500
150	25	1852700
	50	2499700
	75	2671500
	100	4296700
200	25	2236200
	50	3171500
	75	3500500
	100	5080400



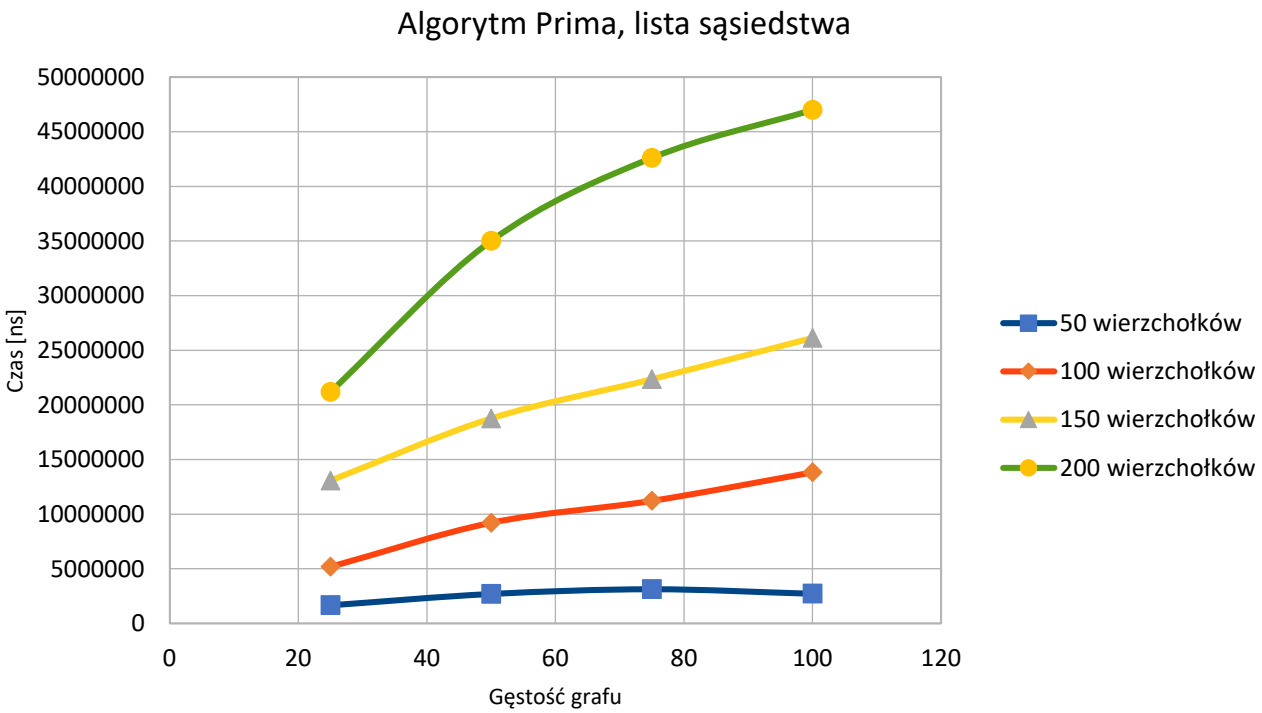
4.4 Algorytm Prima dla implementacji listowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	1920500
	50	2969900
	75	3624500
	100	3578500
100	25	9063500
	50	15111600
	75	20299700
	100	20653000
150	25	28147800
	50	45256700
	75	76209000
	100	82291800
200	25	64268100
	50	131579700
	75	173772400
	100	198128600



4.4 Algorytm Prima dla implementacji listowej

Ilość wierzchołków grafu	Gęstość [%]	Czas [ns]
50	25	1660200
	50	2694500
	75	3125800
	100	2714600
100	25	5179100
	50	9211600
	75	11214900
	100	13823900
150	25	13076500
	50	18759100
	75	22349200
	100	26129000
200	25	21194700
	50	35011600
	75	42605400
	100	46994200



5. Wnioski

Czasem złożoność algorytmów się nie zgadzała z oczekiwaniem, ale z powodu uśredniania wyników po kilkokrotnemu zmierzeniu wyniki mam zgodnie z oczekiwaniem.

Różnice pomiędzy otrzymanymi wynikami, a wynikami zakładanymi wynikają z niedokładności pomiarów lub potencjalnych błędów/braku optymalizacji algorytmów. Też wpływają procesy działające w tle.

Czas wykonywania wszystkich algorytmów grafowych jest ściśle związany z ilością wierzchołków i krawędzi badanego grafu.

6. Bibliografia

- Wikipedia – Algorytm Dijkstry

  http://pl.wikipedia.org/wiki/Algorytm_Dijkstry

- Wikipedia – Bellmana-Forda

  https://pl.wikipedia.org/wiki/Algorytm_Bellmana-Forda

- Wikipedia – Algorytm Prima

  http://pl.wikipedia.org/wiki/Algorytm_Prima