Prowadzący : mgr inż. Antoni Sterna
Grupa : 11:15 TN

Struktury danych i złożoność obliczeniowa

Projekt: Etap 2

Daniel Glazer

252743

30.05.2021

Spis treści

1 Wstęp

Celem projektu było napisanie programu za pomocą, którego będziemy mogli zmierzyć czasy odpowiednich algorytmów grafowych zaimplementowanych na macierzy wag i listy połączonych sąsiadów. Badanie algorytmy:

- dla wyznaczania minimalnego drzewa rozpinającego
    1. Algorytm Kruskala
    2. Algorytm Prima
- dla wyznaczania najkrótszych ścieżek w grafie
    1. Algorytm Dijkstry
    2. Bellmana-Forda

2 Złożoności poszczególnych algorytmów

Dla problemów grafowych złożoności algorytmów opierają się liczbie krawędzi - **E** i liczbie wierzchołków - **V**.

Algorytm	Kruskala	Prima	Dijkstry	Bellmana-Forda
Złożoność	$O(E \cdot \log(V))$	$O(V^2)$	$O(V^2)$	$O(E \cdot V)$

Tabela 1: Tabela poszczególnych złożoności algorytmów

3 Plan eksperymentu

Eksperyment polegał na zmierzaniu czasu wykonywania algorytmu dla poszczególnych struktur zaimplementowanych w $C++$. Do pomiaru czasu wykorzystano funkcje *QueryPerformanceCounter()* z biblioteki *<windows.h>*. Do testów przyjęliśmy 5 różnych liczb wierzchołków grafu: 100, 200, 300, 400, 500 oraz następujące gęstości grafu: 25%, 50%, 75%, i 99%. Dla każdej kombinacji wierzchołków i gęstości generowaliśmy 100 losowych grafów dla których był wyliczany czas średni operacji. Dla każdego algorytmu pomiary wyglądały w następujący sposób:

1. Wybieramy ilość wierzchołków i gęstość grafu
2. Generujemy graf
3. Tworzymy na podstawie tego grafu listę i macierz
4. Wykonujemy operacje dla której mierzymy czas
5. Dodajemy wynik
6. Po wykonaniu operacji dla 100 instancji liczymy czas średni operacji i go zapisujemy.

4 Generowanie Grafu

Na początku tworzymy nowy obiekt klasy GraphRepresentation z parametrami wejściowymi takimi jak skierowanie, gęstość i liczba wierzchołków grafu. Następnie obliczamy ilość krawędzi w zależności od parametrów, dla grafu skierowanego ilość dostępnych krawędzi jest dwa razy większa niż dla grafu nieskierowanego stąd wynika rozdzielenie za pomocą klauzuli *if*.

```
//directed - skierowany/nieskierowany graf
//vNumber - ilość wierchołków;
//graphDensity -gęstość grafu
    if (!directed)
        edgesNumber = 0.5 * vNumber * (vNumber - 1) * graphDensity;
    else
        edgesNumber = vNumber * (vNumber - 1) * graphDensity;
```

Następnie tworzymy połączenie zerowego wierzchołka z wszystkimi pozostałymi w grafie, by uniknąć sytuacji w której mamy wierzchołek o stopniu 0. Ilość wygenerowanych krawędzi przez tą operacje odejmujemy od wszystkich krawędzi potrzebnych do uzyskania zadanej gęstości grafu.

```
    for (int i = 1; i < vNumber; i++) {
        createListForNewGraph(directed, 0, i, dist2(gen));
    }
```

Następnie generujemy dwa wierzchołkami i sprawdzamy czy nie są sobie równe i czy istnieje już pomiędzy nimi taka krawędź. Jeśli nie istnieje to tworzymy nowe połączenie pomiędzy tymi wierzchołkami i generujemy wagę krawędzi. W sytuacji w której nie spełniamy warunków, ponownie generujemy wierzchołki w nadziei uzyskania niepokrywającego się połączenia.

```
    for (int i = edgesNumber - verticesNumber - 1; i > 0; i--) {

        int x = dist(gen);
        int y = dist(gen);
        if (x != y) {
            if (isNotConnection(x, y))
                createListForNewGraph(directed, x, y, dist2(gen));
            else
                i++;
        }
    }
}
```

5 Problem minimalnego drzewa rozpinającego

5.1 Algorytm Kruskala

Algorytm Kruskala został zaimplementowany w rekurencyjnej funkcji. W każdym wywołaniu funkcji szukamy najmniejszej krawędzi, której wierzchołki nie zostały wybrane albo nie posiadają tego samego koloru wierzchołka. Funkcje wywołujemy do momentu aż w tablicy, przechowującej MST znajdzie się $V - 1$ krawędzi.

5.2 Wyniki pomiarów dla Algorytmu Kruskala

Ilość wierzchołków	Algorytm Kruskala [ms]							
	Lista				Macierz			
	25%	50%	75%	99%	25%	50%	75%	99%
100	1,21	2,58	4,71	6,18	4,85	5,52	4,62	3,35
200	17,03	42,51	74,44	104,36	37,18	45,66	38,39	27,00
300	81,34	194,72	330,23	433,99	116,23	156,47	134,32	89,54
400	241,81	564,36	1021,30	1481,41	287,93	384,16	314,21	213,45
500	600,81	1134,59	2715,12	5239,86	560,56	722,89	603,02	402,67

Tabela 2: Tabela czasów w milisekundach pomiarów dla Algorytmu Kruskala

6 Algorytm Prima

Algorytm Prima został zaimplementowany w rekurencyjnej funkcji. W każdym wywołaniu funkcji szukamy najmniejszej krawędzi, spośród wierzchołków, do których została już wytyczona krawędź. Dla takich wierzchołków wartość w tablicy *visitedTable* wynosi *true*. Żeby wywołać funkcje dla wierzchołka początkowego ustawiamy wartość w tabeli na *true* i postępujemy zgodnie z algorytmem. Algorytm wykona się $V - 1$ razy.

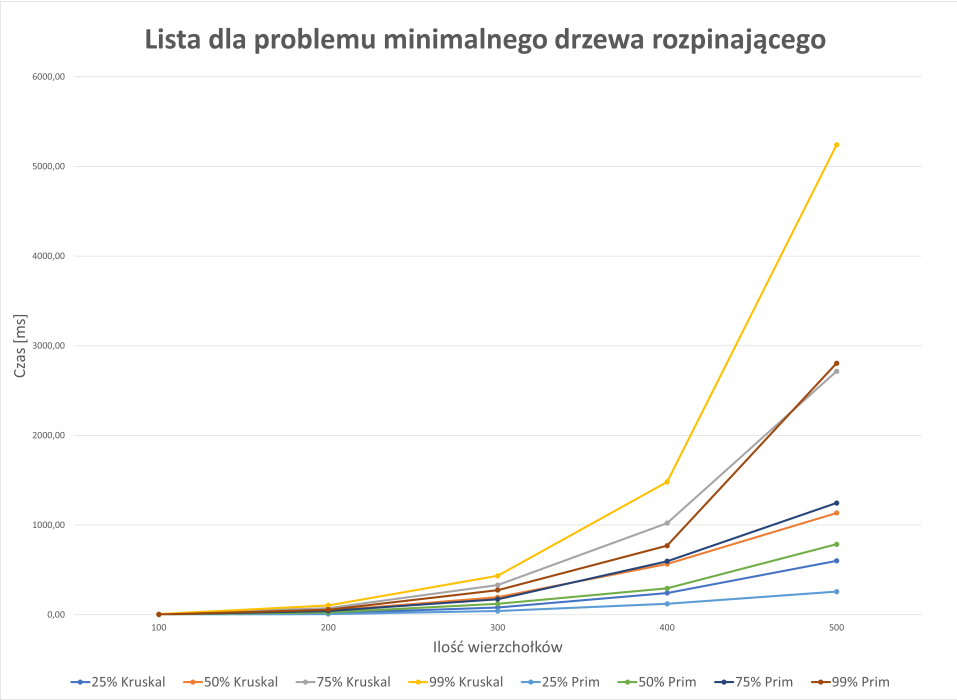
6.1 Wyniki pomiarów dla Algorytmu Prima

Ilość wierzchołków	Algorytm Prima [ms]							
	Lista				Macierz			
	25%	50%	75%	99%	25%	50%	75%	99%
100	0,61	1,50	2,61	3,41	2,99	3,07	3,12	3,14
200	7,73	24,24	42,20	55,85	24,67	24,88	25,67	24,07
300	41,51	120,75	172,98	274,14	88,04	89,71	88,87	87,11
400	120,84	293,73	596,10	770,00	210,09	213,20	213,38	209,91
500	257,19	786,43	1245,30	2804,94	428,47	435,83	409,91	432,60

Tabela 3: Tabela czasów w milisekundach pomiarów dla Algorytmu Prima

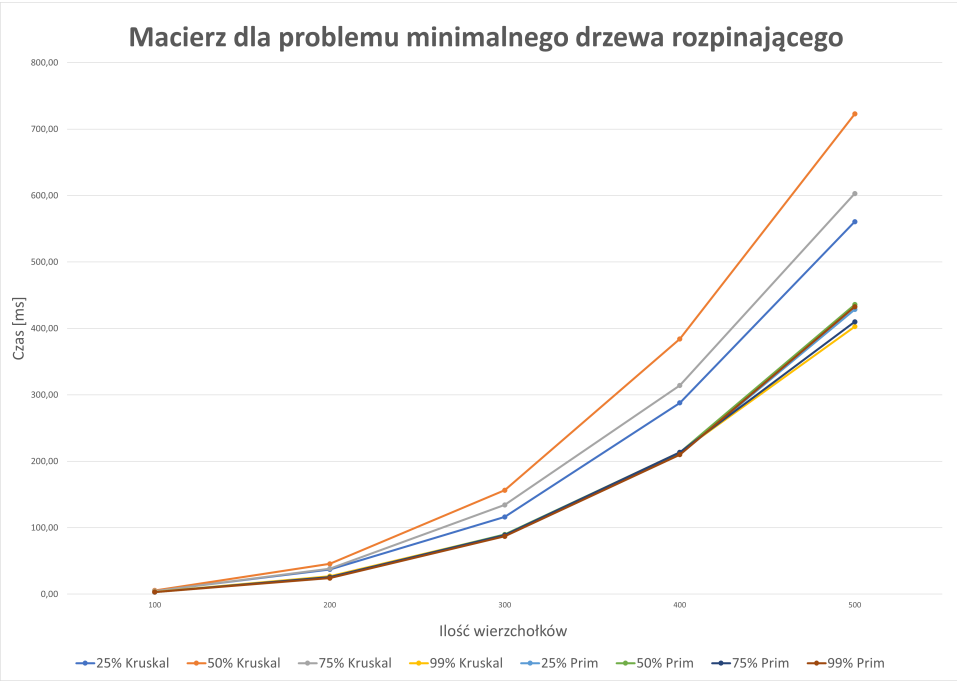
6.2 Wykresy

6.2.1 Dla listy



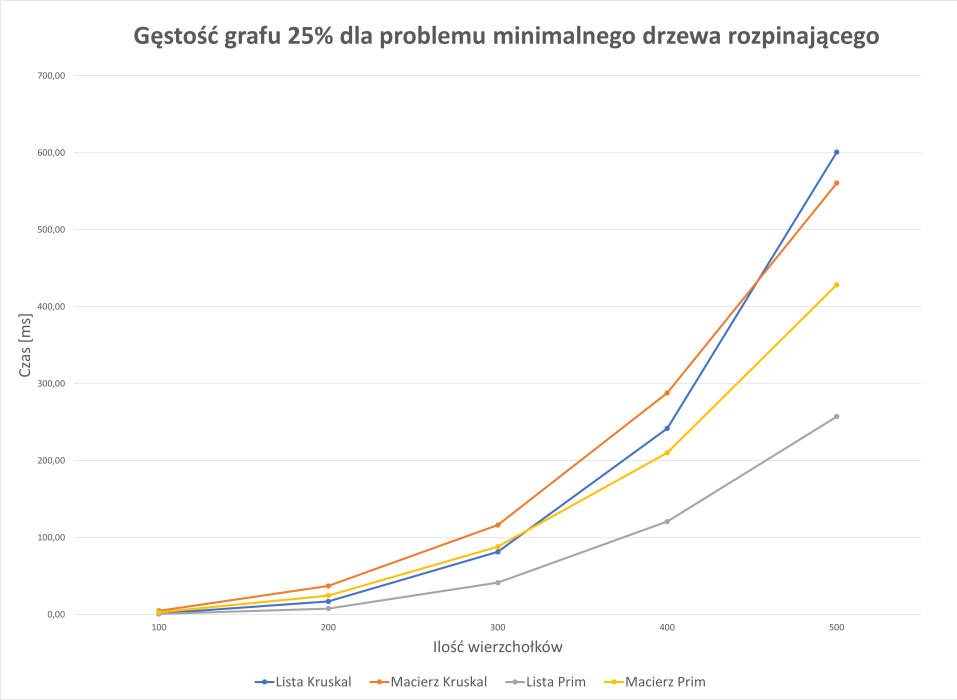
Rysunek 1: Zależność wierzchołków grafu do czasu w zależności od różnej gęstości grafu dla listy

6.2.2 Dla macierzy



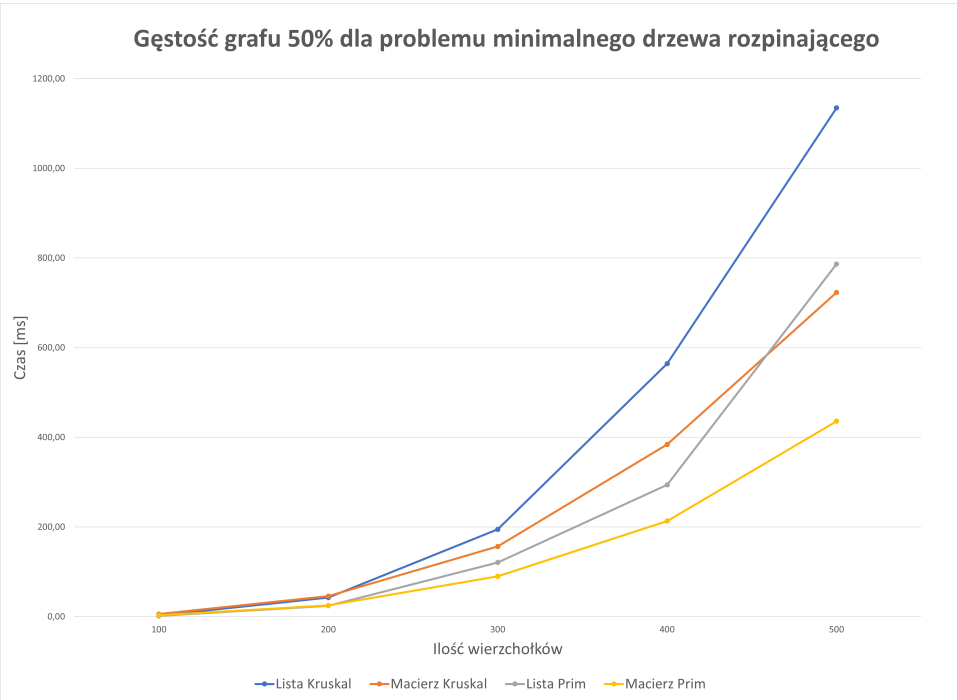
Rysunek 2: Zależność wierzchołków grafu do czasu w zależności od typu struktury

6.2.3 Dla 25% gęstości grafu



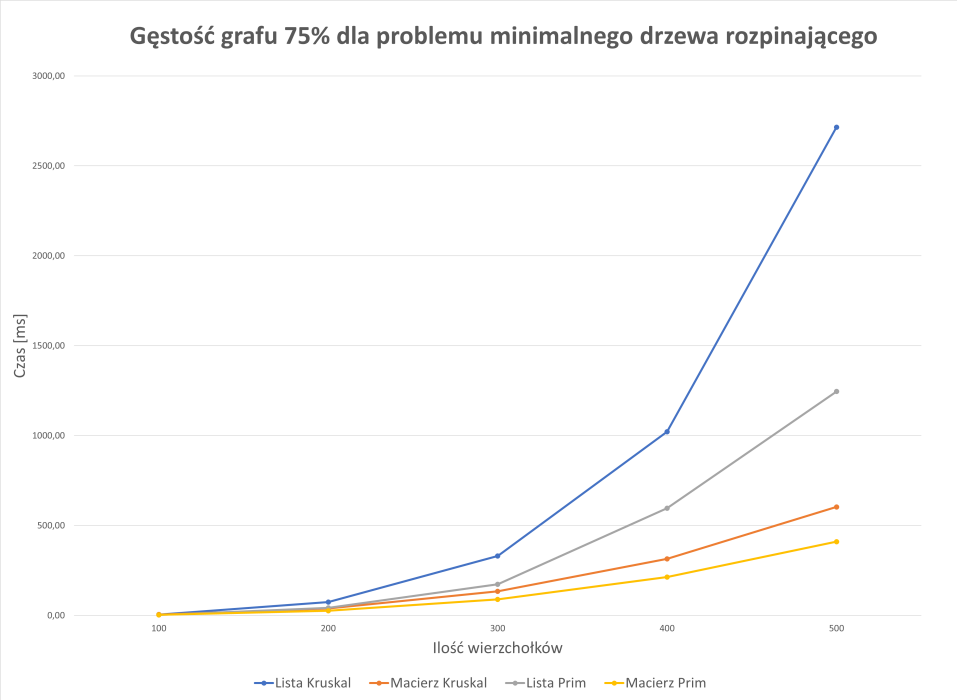
Rysunek 3: Zależność wierzchołków grafu do czasu w zależności od typu struktury

6.2.4 Dla 50% gęstości grafu



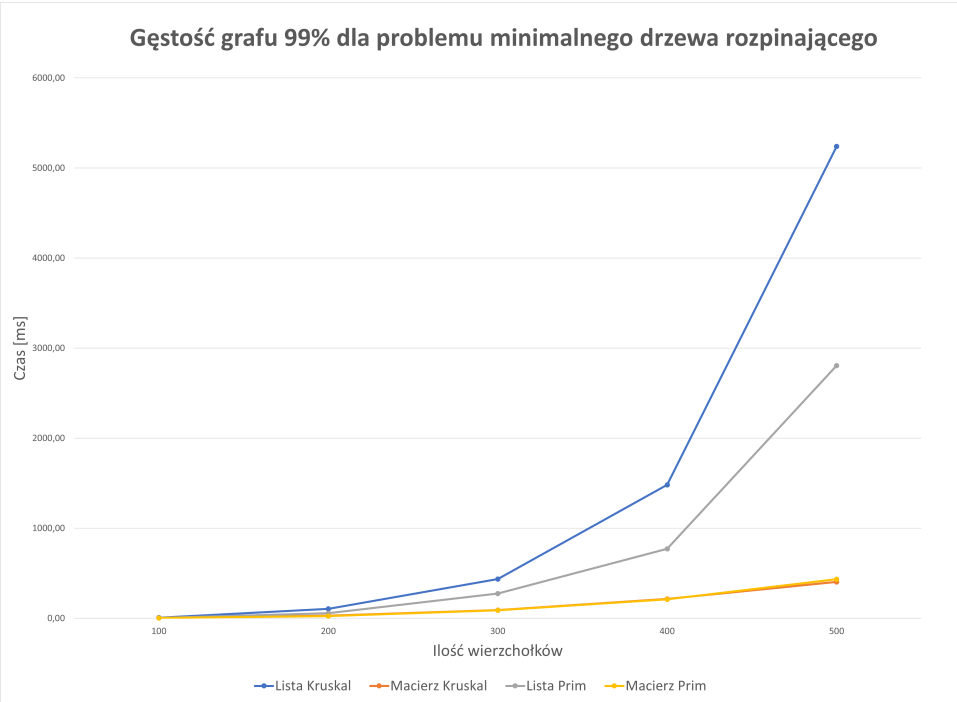
Rysunek 4: Zależność wierzchołków grafu do czasu w zależności od typu struktury

6.2.5 Dla 75% gęstości grafu



Rysunek 5: Zależność wierzchołków grafu do czasu w zależności od typu struktury

6.2.6 Dla 99% gęstości grafu



Rysunek 6: Zależność wierzchołków grafu do czasu w zależności od typu struktury

7 Problem najkrótszych ścieżek w grafie

7.1 Algorytm Dijkstry

Algorytm podobnie jak poprzednie algorytmy został zaimplementowany na rekurencyjnej funkcji. W każdym wywołaniu funkcji wybieramy nieodwiedzony wierzchołek, do którego prowadzi najkrótsza ścieżka z obecnie odkrytych wierzchołków. Następnie sprawdzamy czy możemy dokonać relaksacji, jeśli tak to zmieniamy wartość ścieżki w tablicy i poprzednika dla danego wierzchołka. Kończymy program wraz z odwiedzeniem wszystkich wierzchołków.

7.2 Wyniki pomiarów dla Algorytmu Dijkstry

Ilość wierzchołków	Algorytm Dijkstry [ms]							
	Lista				Macierz			
	25%	50%	75%	99%	25%	50%	75%	99%
100	0,06	0,09	0,09	0,11	0,12	0,14	0,13	0,10
200	0,27	0,44	0,64	0,74	0,50	0,56	0,51	0,42
300	0,81	1,11	1,91	1,69	1,04	1,44	1,23	0,82
400	1,28	2,72	3,96	8,18	1,94	2,48	2,25	1,52
500	2,62	6,32	7,52	15,38	3,11	3,72	3,25	2,63

Tabela 4: Tabela czasów w milisekundach pomiarów dla Algorytmu Dijkstry

8 Algorytm Bellmana Forda

Algorytm działa rekurencyjnie. W każdym wywołaniu sprawdzamy czy można wykonać relaksacje, dla ścieżek połączonych z danym wierzchołkiem. Następnie takie wierzchołki umieszczamy w kolejce, według której są wywoływane kolejne rekurencje. Jeśli została wykonana relaksacja, a wierzchołek był już odwiedzony, to umieszczamy go ponownie na początek kolejki. Jeśli algorytm wykona się więcej razy niż ilość wierzchołków to mamy cykl ujemny.

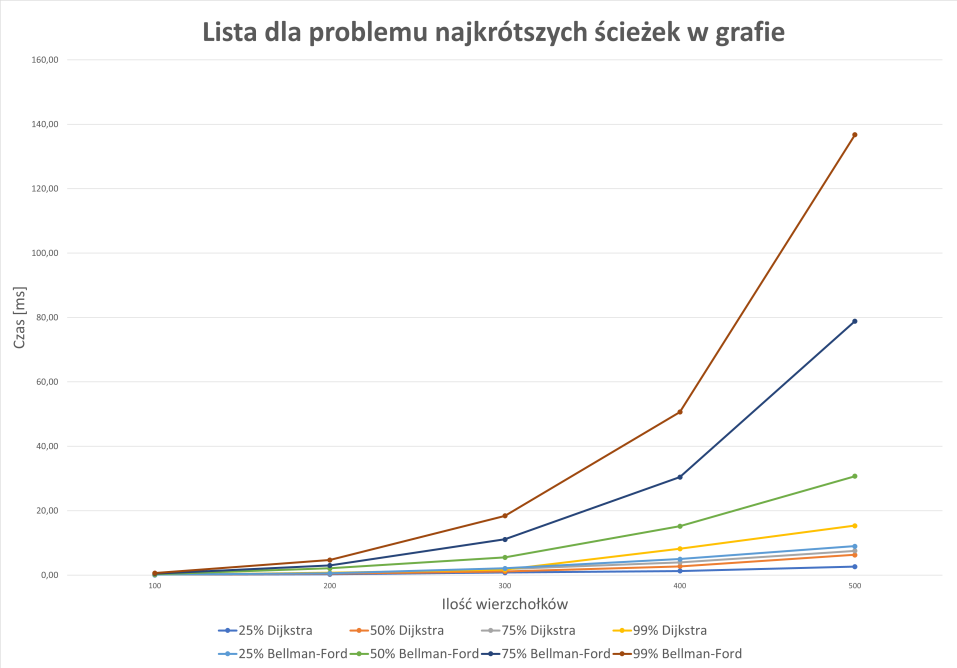
8.1 Wyniki pomiarów dla Algorytmu Bellmana-Forda

Ilość wierzchołków	Algorytm Bellmana-Forda							
	Lista				Macierz			
	25%	50%	75%	99%	25%	50%	75%	99%
100	0,13	0,25	0,47	0,62	0,42	0,71	0,66	0,49
200	0,56	2,11	3,03	4,69	1,92	3,65	2,87	2,14
300	2,11	5,48	11,10	18,40	5,47	7,98	7,80	5,45
400	5,00	15,20	30,45	50,64	10,74	17,82	15,20	9,81
500	8,99	30,70	78,85	136,74	16,87	31,52	25,43	17,33

Tabela 5: Tabela czasów w milisekundach pomiarów dla Algorytmu Bellmana-Forda

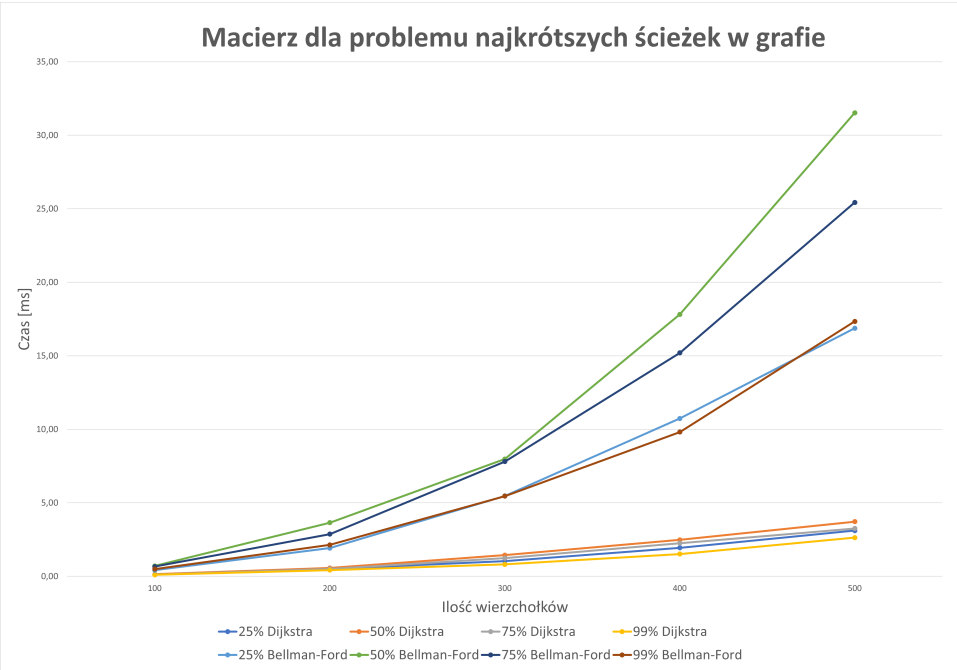
8.2 Wykresy

8.2.1 Dla listy



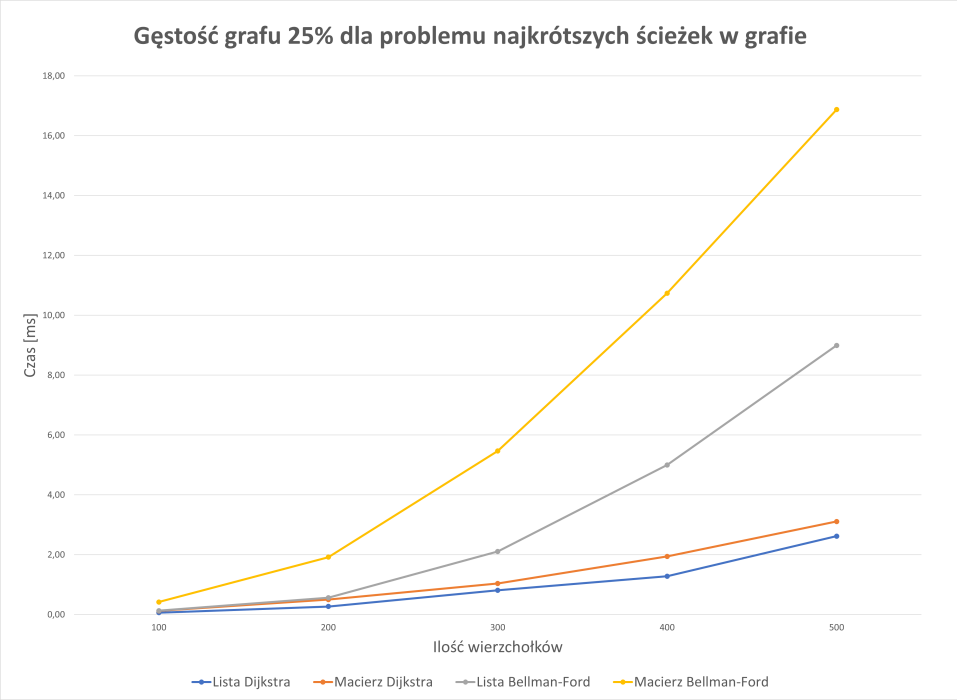
Rysunek 7: Zależność wierzchołków grafu do czasu w zależności od różnej gęstości grafu dla listy

8.2.2 Dla macierzy



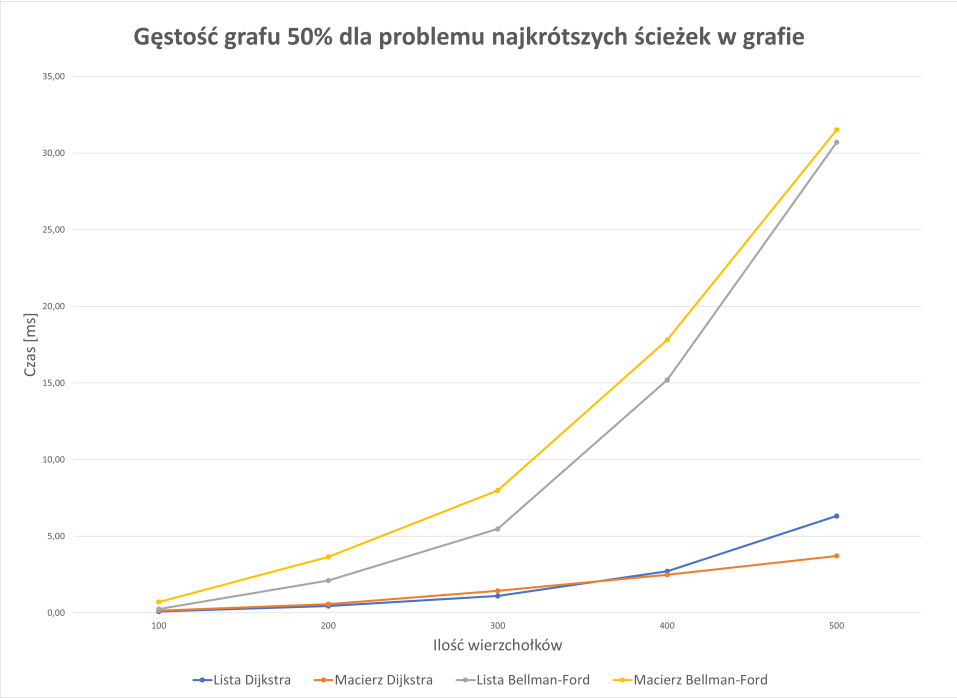
Rysunek 8: Zależność wierzchołków grafu do czasu w zależności od różnej gęstości grafu dla macierzy

8.2.3 Dla 25% gęstości grafu



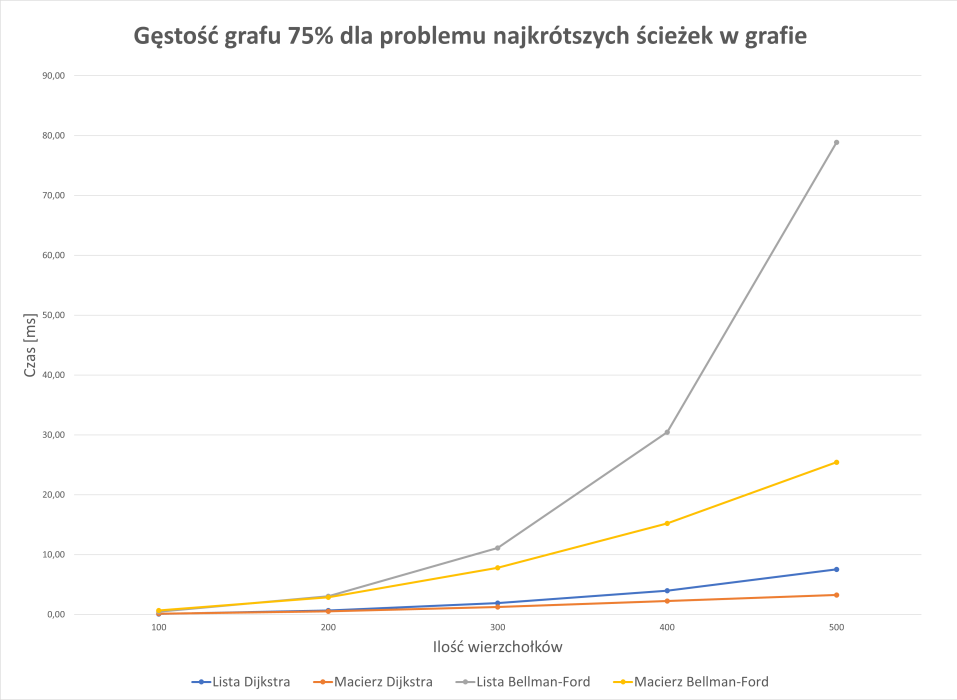
Rysunek 9: Zależność wierzchołków grafu do czasu w zależności od typu struktury

8.2.4 Dla 50% gęstości grafu



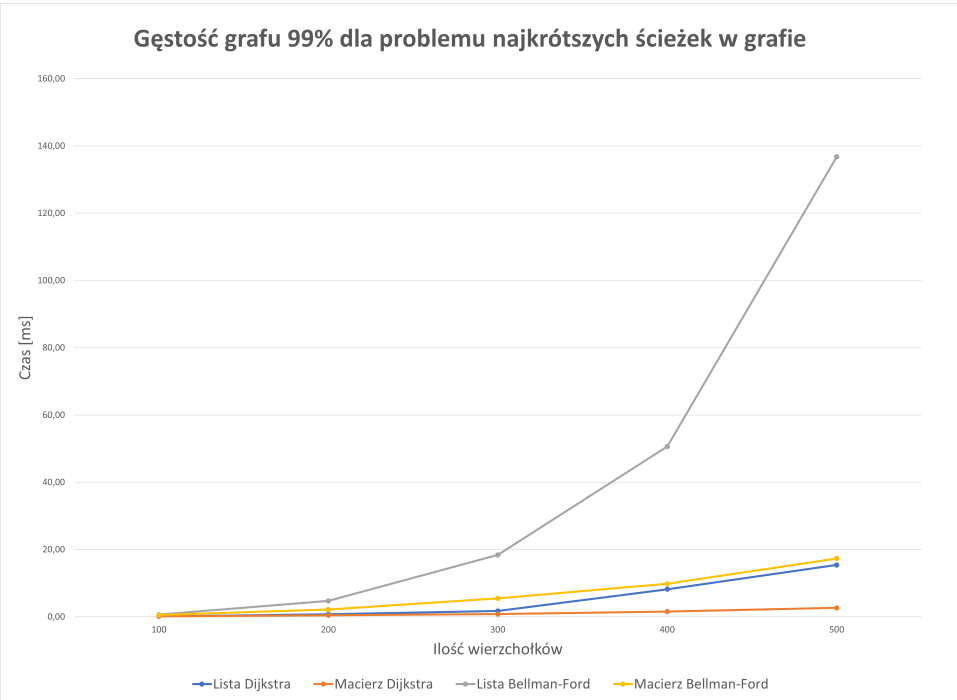
Rysunek 10: Zależność wierzchołków grafu do czasu w zależności od typu struktury

8.2.5 Dla 75% gęstości grafu



Rysunek 11: Zależność wierzchołków grafu do czasu w zależności od typu struktury

8.2.6 Dla 99% gęstości grafu



Rysunek 12: Zależność wierzchołków grafu do czasu w zależności od typu struktury

9 Wnioski

Dla większości pomiarów możemy zauważyć, że lista sprawdza się lepiej od macierzy w momencie, w którym gęstość grafu jest mniejsza niż 50% dla bardziej gęstych grafów algorytmy wykonujące się na macierzy wag osiągają lepsze czasy niż lista połączonych sąsiadów.

Algorytm Kruskala ma złożoność czasową $O(E \cdot \log V)$, więc w głównej mierze zależy od ilości krawędzi czyli gęstości grafu. Jak możemy zauważyć z wykresów gęstość grafu ma wpływ na czas działania algorytmu, ale większe znaczenie ma struktura na której jest zaimplementowany algorytm. Jeśli jest to lista to dla gęstości powyżej 50% wyniki będą odstawały od pozostałych.

Dla Algorytmu Prima złożoność czasowa wynosi $O(V^2)$, więc jest kwadratowo zależna od wierzchołków. Dla macierzy wag możemy zauważyć, że gęstość grafu nie wpływa na wyniki pomiaru co by potwierdzało złożoność liczbową z literatury, ale dla listy sytuacja wygląda trochę inaczej. Dla listy przy większej gęstości, czasy pomiarów również rosną co zaprzecza złożoności na podstawie literatury. Ta różnica wynika najprawdopodobniej z samej specyfikacji listy, która dla większych gęstości grafu osiąga większe czasy.

Porównując oby dwa algorytmy dla tego samego problemu możemy stwierdzić, że algorytm Prima jest szybszym algorytmem niż Kruskal, bo jest zależny od wierzchołków, a nie na gęstości grafu, która ma także wpływ na struktury w, których jest przechowywany graf.

Algorytm Dijkstry podobnie jak Algorytm Prima ma złożoność $O(V^2)$. Z wykresów wynika, że złożoność czasowa nie jest zależna od gęstości grafu co potwierdza złożoność zawartą w literaturze.

Algorytm Bellmana-Forda w literaturze ma złożoność czasową $0(E \cdot V)$. Więc wraz z wzrastającą gęstością grafu i liczby wierzchołków czas wykonywania się algorytmu rośnie. Dla pomiarów wykonanych w tym projekcie dla macierzy występują pewne rozbieżności. Dla gęstości 50% i 75% uzyskujemy gorsze czasy niż dla gęstości 99%. Rozbieżność ta wynika najprawdopodobniej z tego, że macierz uzyskuje lepsze czasy dla większych gęstości grafu.

Porównując oba algorytmy dla problemu najkrótszych ścieżek w grafie, możemy stwierdzić, że Algorytm Dijkstry jest algorytmem szybszym niż Algorytm Bellmana-Forda. Algorytm Bellmana-Forda najwięcej traci na szybkości obliczeń z powodu zabezpieczenia przed cyklami ujemnymi, których drugi algorytm nie wykrywa.