22.04.2020r.
Emilia Starczyk
249005

PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI

Projekt 2

Dr inż. Łukasz Jeleń
Czwartek 9.15 - 11.00

1. Wstęp

Tematem projektu było zbadanie efektywności działania algorytmu odnajdywania ścieżki o najniższym koszcie od jednego punktu do drugiego. Badania te były do wykonania na dwóch reprezentacjach grafu (na macierzy sąsiedztwa oraz na liście sąsiedztwa) o różnej gęstości i ilości wierzchołków. Dane były dwa algorytmy do wybrania: algorytm Dijkstry oraz algorytm Bellmana-Forda. W swoim programie zaimplementowałam pierwszy z algorytmów. W projekcie założyłam wagi krawędzi od 1 do 100, aby łatwiej było sprawdzić poprawność działania algorytmu.

2. Graf na macierzy sąsiedztwa

Graf na macierzy sąsiedztwa jest grafem opartym o tablicę rozmiaru n *x* n, gdzie n oznacza liczbę wierzchołków w danym grafie. Taką tablicę wykorzystuje się do zapisania informacji o istnieniu krawędzi łączącej dane wierzchołki, np. miejsce w tablica[0][1] będzie informowało nas o istnieniu krawędzi między wierzchołkiem o indeksie 0, a wierzchołkiem o indeksie 1. Istnieją różne implementacje grafu na macierzy. Jedną z implementacji jest zapisanie w takiej macierzy/tablicy wskaźników na obiekty typu Krawędź, w ten sposób bezpośrednio uzyskujemy informacje o wierzchołkach końcowych krawędzi jak i wadze danej drogi. W swoim programie wykorzystuję właśnie taką implementacje.

3. Graf na liście sąsiedztwa

Graf ten oparty jest o tzw. listę sąsiedztwa. Jest to zwykła lista wskaźników na listy wierzchołków sąsiadujących lub też lista wskaźników na listy krawędzi skojarzonych z danym wierzchołkiem. W obu implementacjach uzyskujemy bezpośrednio informację o wierzchołkach sąsiadujących. Przy tej implementacji nie jesteśmy ograniczeni, jeśli chodzi o liczbę węzłów. Przy dodaniu wierzchołka do grafu nie mamy potrzeby rozszerzania i przepisywania tablicy, jak w przypadku implementacji grafu opartej o macierz sąsiedztwa. Wadą tej reprezentacji jest niebezpośredni dostęp do informacji czy dwa wierzchołki są połączone. Musimy przeszukać listę wierzchołków/krawędzi sąsiednich, aby uzyskać taką informację

4. Algorytm Dijkstry

Jest to algorytm służący do wyznaczania najmniejszej ścieżki od wierzchołka startowego do wszystkich pozostałych wierzchołków. Jest to algorytm, którego stosowanie jest ograniczone do grafów o dodatnich wagach. Algorytm ten działa w kilku prostych krokach:

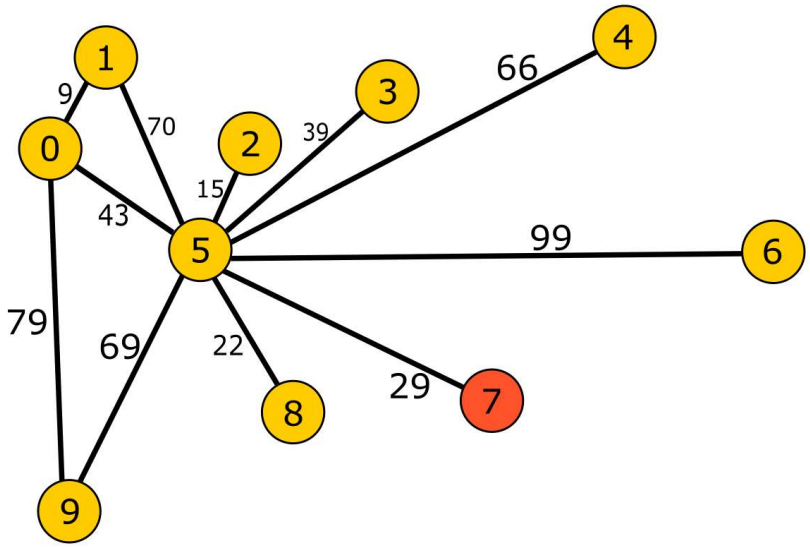
1. Ustawiamy odległości wszystkich wierzchołków od wierzchołka startowego na nieskończoność, dla wierzchołka startowego odległość ustawiana jest na 0.
2. Ściągamy wierzchołek o najmniejszej odległości od wierzchołka startowego z kolejki.
3. Dla wszystkich wierzchołków sąsiadujących z wierzchołkiem ściągniętym sprawdzamy, czy oszacowana wcześniej droga jest większa od drogi, którą byśmy przeszli przez wierzchołek ściągnięty z kolejki, jeśli tak to zmieniamy wartość odległości.
4. Wykonujemy punkty 2 i 3, dopóki kolejka nie będzie pusta.

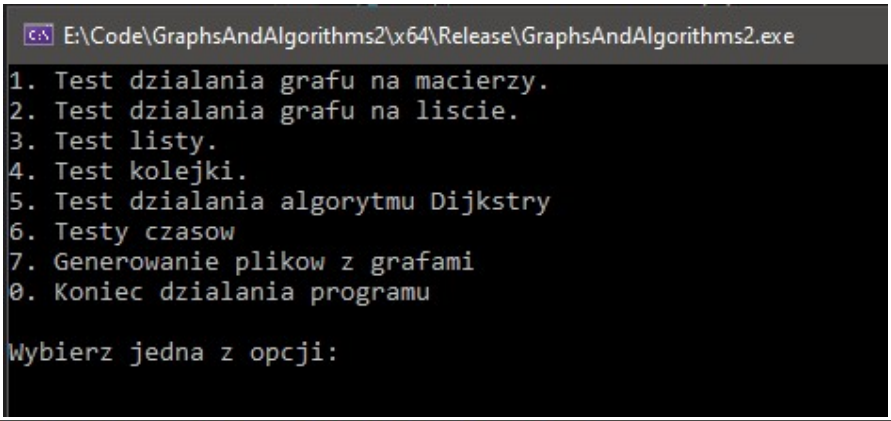
Dany algorytm można uzupełnić o kod wyznaczający kolejne wierzchołki, przez które powinniśmy przejść od wierzchołka startowego do wierzchołka końcowego. Algorytm Dijkstry można stosować w nawigacjach do wyszukiwania najkrótszej drogi między miastami. Wierzchołki mogą symbolizować również lotniska a krawędzie wraz z wagami odległości między nimi.

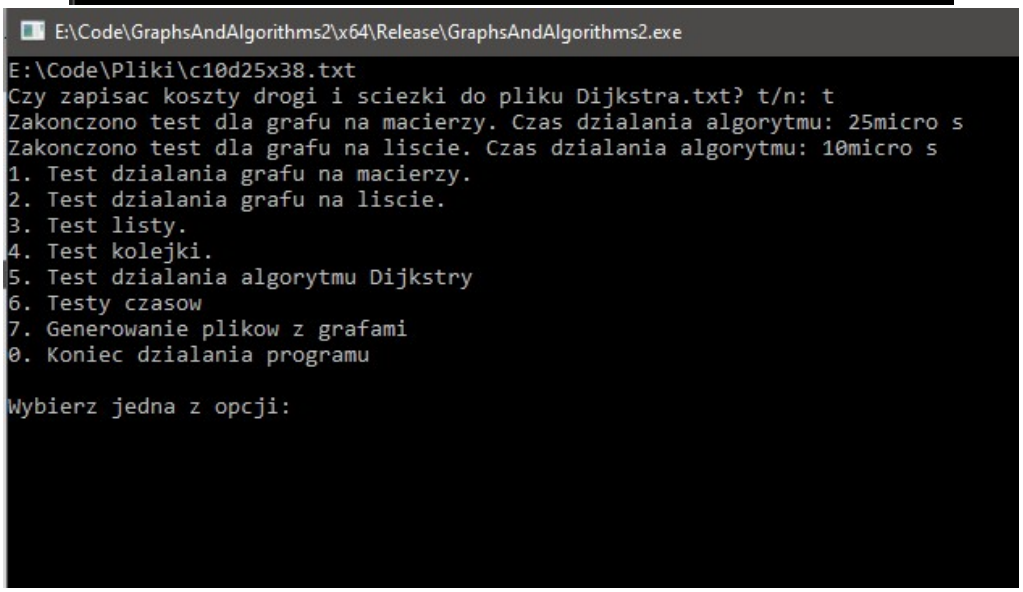
5. Przebieg badań

Przygotowanie programu do testów zaczęłam od napisania szablonu listy dwukierunkowej potrzebnej do stworzenia grafów. Zaimplementowałam również klasę Matrix, potrzebną do stworzenia grafu opartego na macierzy sąsiedztwa. Następnie zaimplementowałam klasy dla obiektów: krawędź oraz wierzchołek. Napisałam również kolejkę priorytetową na kopcu, a kopiec oparłam na wcześniej napisanej liście. Zaimplementowałam również podstawowe metody pozwalające na dostęp do elementów danych klas. Zaimplementowałam grafy wraz z najistotniejszymi metodami. Następnym krokiem było napisanie funkcji generującej grafy potrzebne do zbadania efektywności. Do generowania wykorzystałam funkcję rand() w odpowiednio dobranych zakresach. Zaimplementowałam algorytm Dijkstry oraz dodałam fragment pozwalający na wypisanie ścieżki od jednego punktu do drugiego tak, aby można było przeprowadzić wszystkie testy. Na początku przeprowadziłam małe testy grafie składającym się z 10 wierzchołków i 11 krawędzi: (punkt `5. Test dzialania algorytmu Dijkstry` z menu)
Dane z pliku *c10d25x38.txt*, z którego wczytałam graf do testu:

```
11      10      7
5       0       43
5       1       70
5       2       15
5       3       39
5       4       66
5       6       99
5       7       29
5       8       22
5       9       69
9       0       79
0       1       9
```







Funkcja 5 z menu przechodzi do testu działania algorytmu, pierwsza linijka wyświetla ścieżkę oraz nazwę pliku, dla którego uruchomiłam test. W następnej linijce znajduje się pytanie o to, czy wyniki działania mają zostać zapisane do pliku o nazwie *Dijkstra.txt*.
Wyniki z pliku *Dijkstra.txt*:
Graf na macierzy - wyniki dzialania algorytmu:
W.koncowy: 7 Koszt sciezki: 0 Sciezka: 7
W.koncowy: 5 Koszt sciezki: 29 Sciezka: 7 5
W.koncowy: 2 Koszt sciezki: 44 Sciezka: 7 5 2
W.koncowy: 8 Koszt sciezki: 51 Sciezka: 7 5 8
W.koncowy: 3 Koszt sciezki: 68 Sciezka: 7 5 3
W.koncowy: 0 Koszt sciezki: 72 Sciezka: 7 5 0
W.koncowy: 1 Koszt sciezki: 81 Sciezka: 7 5 0 1
W.koncowy: 4 Koszt sciezki: 95 Sciezka: 7 5 4
W.koncowy: 9 Koszt sciezki: 98 Sciezka: 7 5 9
W.koncowy: 6 Koszt sciezki: 128 Sciezka: 7 5 6
Czas wykonania algorytmu: 25micros
Graf na liscie - wyniki dzialania algorytmu:
W.koncowy: 7 Koszt sciezki: 0 Sciezka: 7
W.koncowy: 5 Koszt sciezki: 29 Sciezka: 7 5

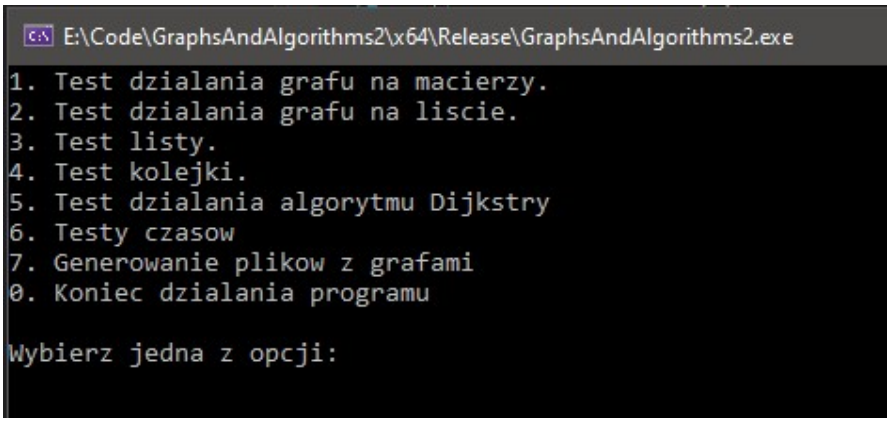
W.koncowy: 2 Koszt sciezki: 44 Sciezka: 7 5 2
W.koncowy: 8 Koszt sciezki: 51 Sciezka: 7 5 8
W.koncowy: 3 Koszt sciezki: 68 Sciezka: 7 5 3
W.koncowy: 0 Koszt sciezki: 72 Sciezka: 7 5 0
W.koncowy: 1 Koszt sciezki: 81 Sciezka: 7 5 0 1
W.koncowy: 4 Koszt sciezki: 95 Sciezka: 7 5 4
W.koncowy: 9 Koszt sciezki: 98 Sciezka: 7 5 9
W.koncowy: 6 Koszt sciezki: 128 Sciezka: 7 5 6
Czas wykonania algorytmu: 10micros

W celu uzyskania pełnych danych do zbadania efektywności uruchomiłam funkcję „Testy czasów" z menu, dla zebrania informacji o czasach wykonywania algorytmu dla 100 instancji każdego z typów grafów. Dane te zapisane zostały w pliku *Czasy.csv*, następnie odpowiednio obrobione i przedstawione w kolejnym punkcie.

Fragment zapisanych danych w pliku *Czasy.csv*:

Liczba wierzcholkow	Gestosc	Macierz[micros]	Lista[micros]
10	0,25	719	258
10	0,5	1638	608
10	0,75	2713	1078
10	1	4308	1520

Opis menu:



Punkt pierwszy umożliwia sprawdzenie działania metod grafu opartego na macierzy.

Punkt drugi umożliwia sprawdzenie działania metod grafu opartego na liście.

Punkt trzeci umożliwia sprawdzenie działania metod listy.

Punkt czwarty umożliwia sprawdzenie działania metod kolejki.

Punkt piąty umożliwia sprawdzenie działania algorytmu Dijkstry.

Punkty szósty umożliwia zebranie danych czasowych z programu.

Punkt siódmy umożliwia wygenerowanie plików z grafami potrzebnymi do testów czasowych.

Punkt zero kończy działanie programu.

6. Wyniki

Gęstość	Liczba wierzchołków	Macierz
	Czas wykonania [s]	
25%	10	0,0000072
	50	0,00015
	100	0,00130
	500	0,02947
	1000	0,46907
50%	10	0,000016
	50	0,000348
	100	0,002059
	500	0,083755
	1000	0,85985
75%	10	0,000027
	50	0,000576
	100	0,0031
	500	0,1702
	1000	1,5285
100%	10	0,000043
	50	0,00084
	100	0,00444
	500	0,29479
	1000	2,50402

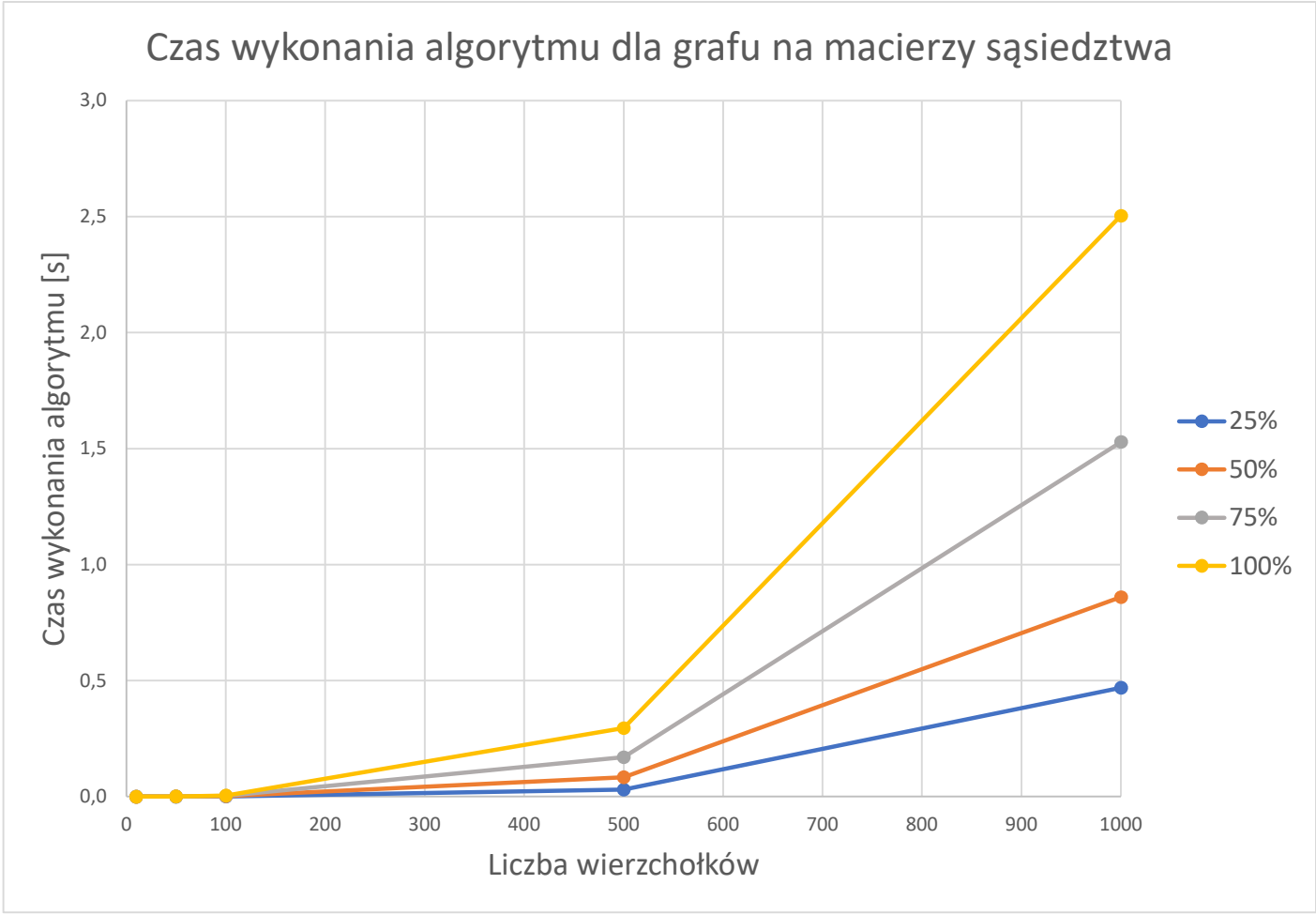


Tabela 1 oraz Wykres 1 Czas wykonania algorytmu dla grafu opartego na macierzy sąsiedztwa

Gęstość	Liczba wierzchołków	Lista
	Czas wykonania [s]	
25%	10	0,0000026
	50	0,000081
	100	0,000817
	500	0,02846
	1000	0,6318
50%	10	0,0000061
	50	0,00020
	100	0,00135
	500	0,09855
	1000	1,22837
75%	10	0,0000108
	50	0,000341
	100	0,0022
	500	0,2307
	1000	2,4200
100%	10	0,000015
	50	0,000506
	100	0,0032
	500	0,4042
	1000	4,0459

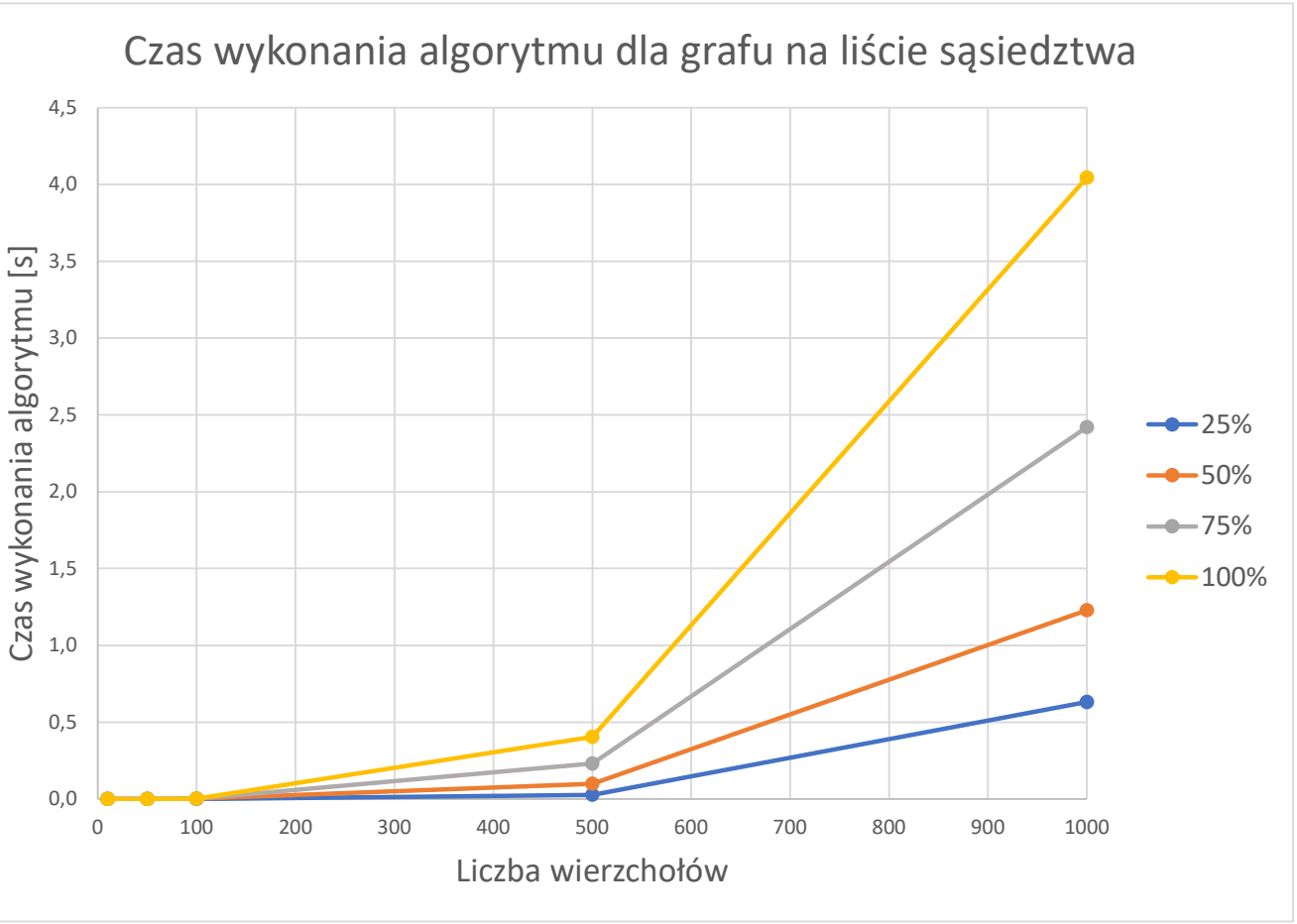
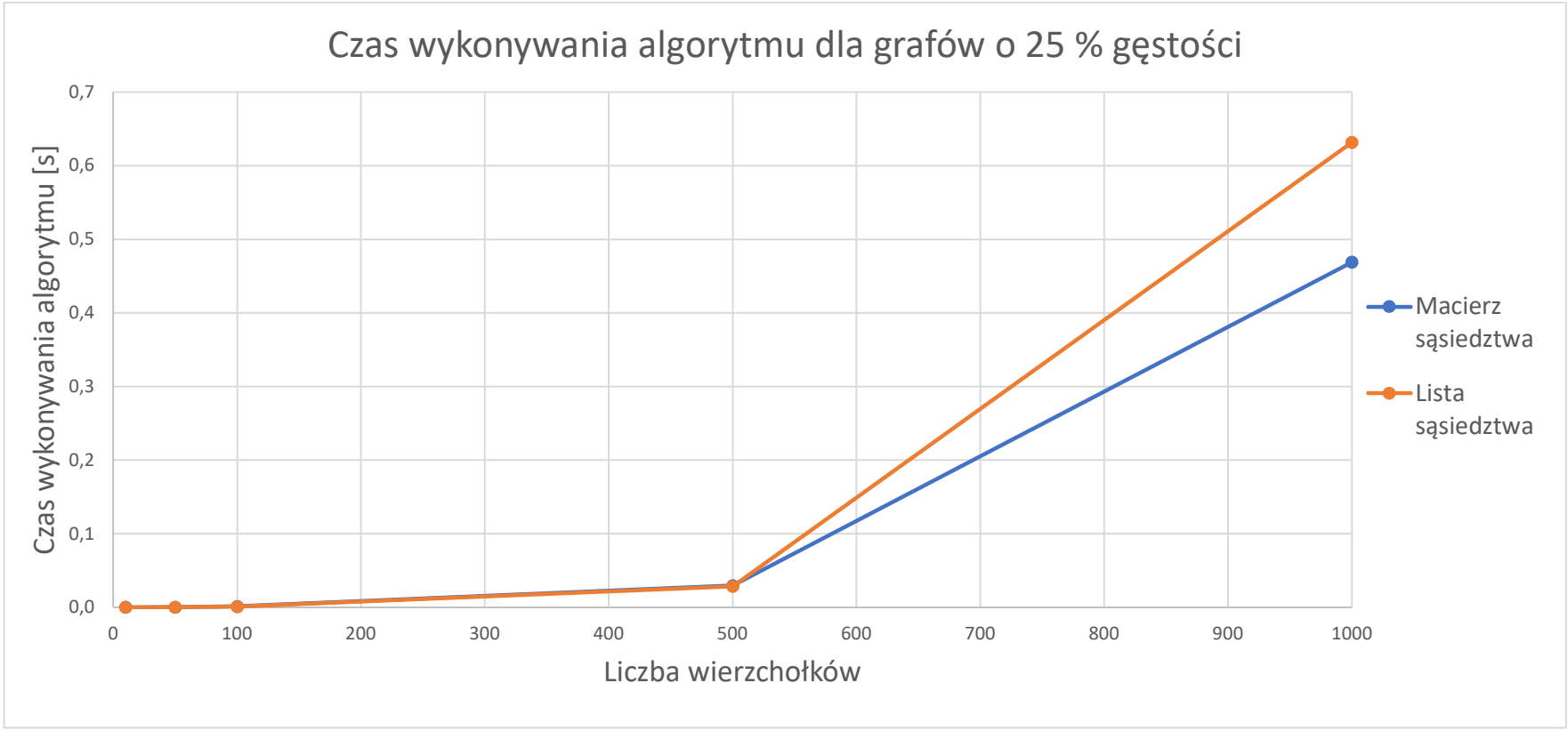
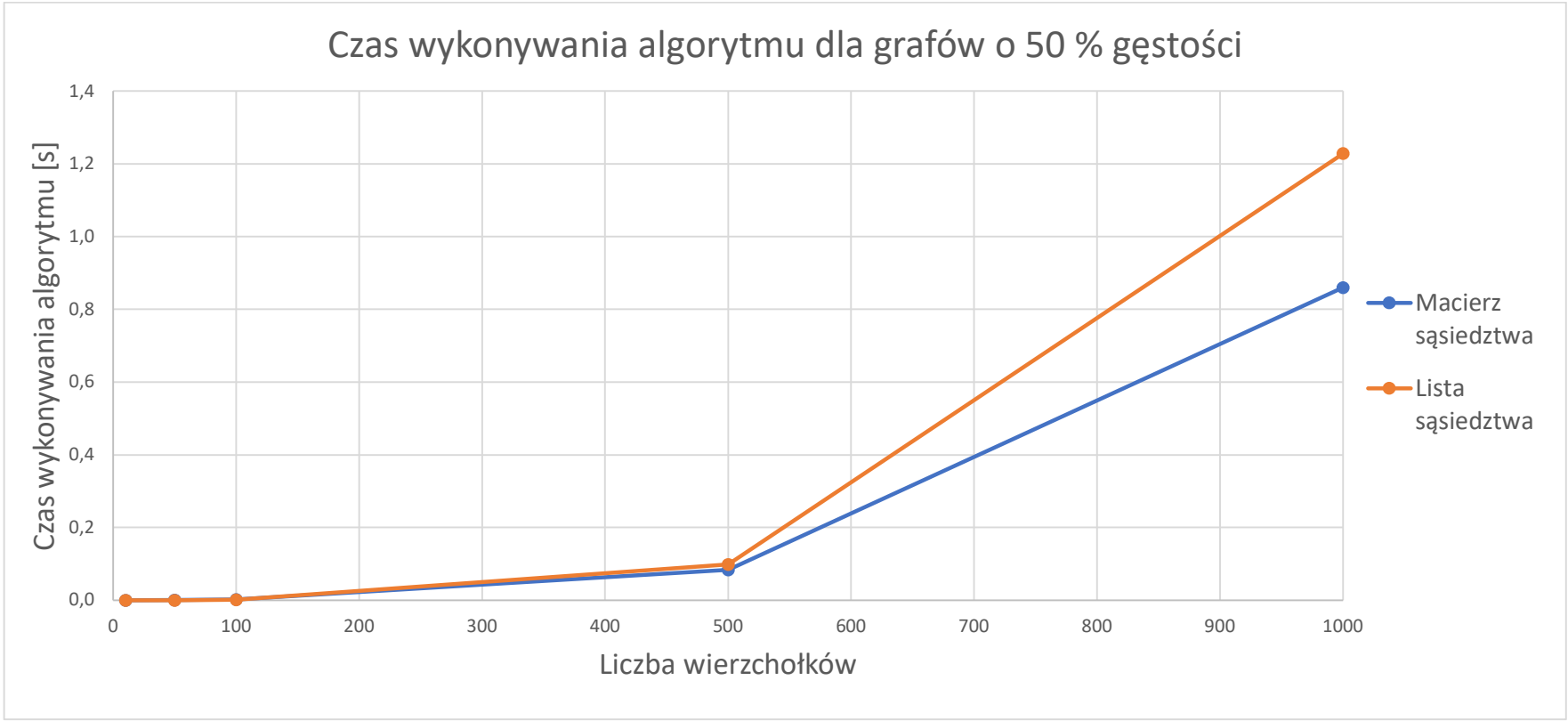


Tabela 2 oraz Wykres 2 Czas wykonania algorytmu dla grafu opartego na liście sąsiedztwa



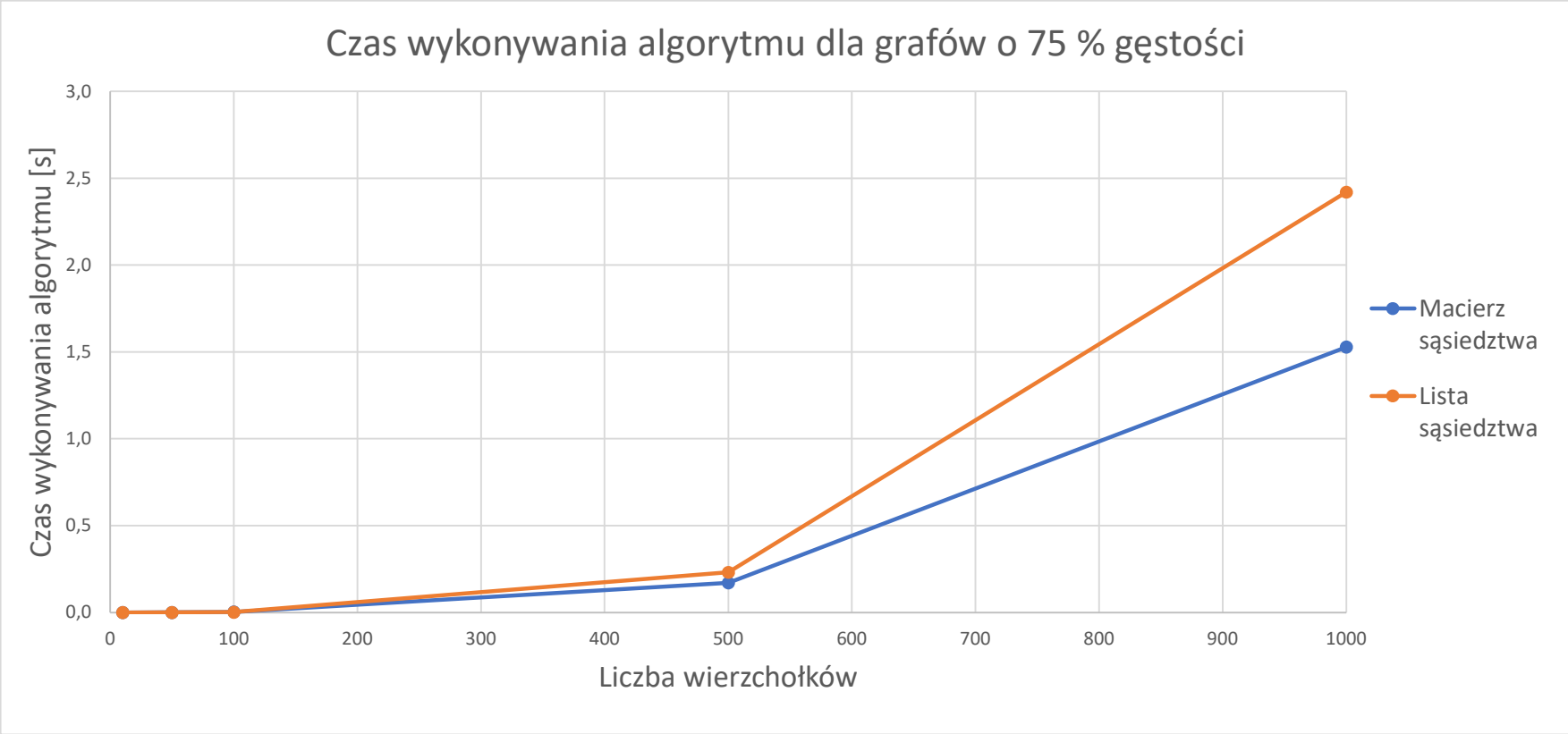
Gęstość	Liczba wierzchołków	Macierz [s]	Lista [s]
25%	10	0,0000072	0,0000026
	50	0,00015	0,000081
	100	0,00130	0,000817
	500	0,02947	0,02846
	1000	0,46907	0,6318

Tabela 3 oraz Wykres 3 Czas działania algorytmu dla grafów o gęstości równej 25%



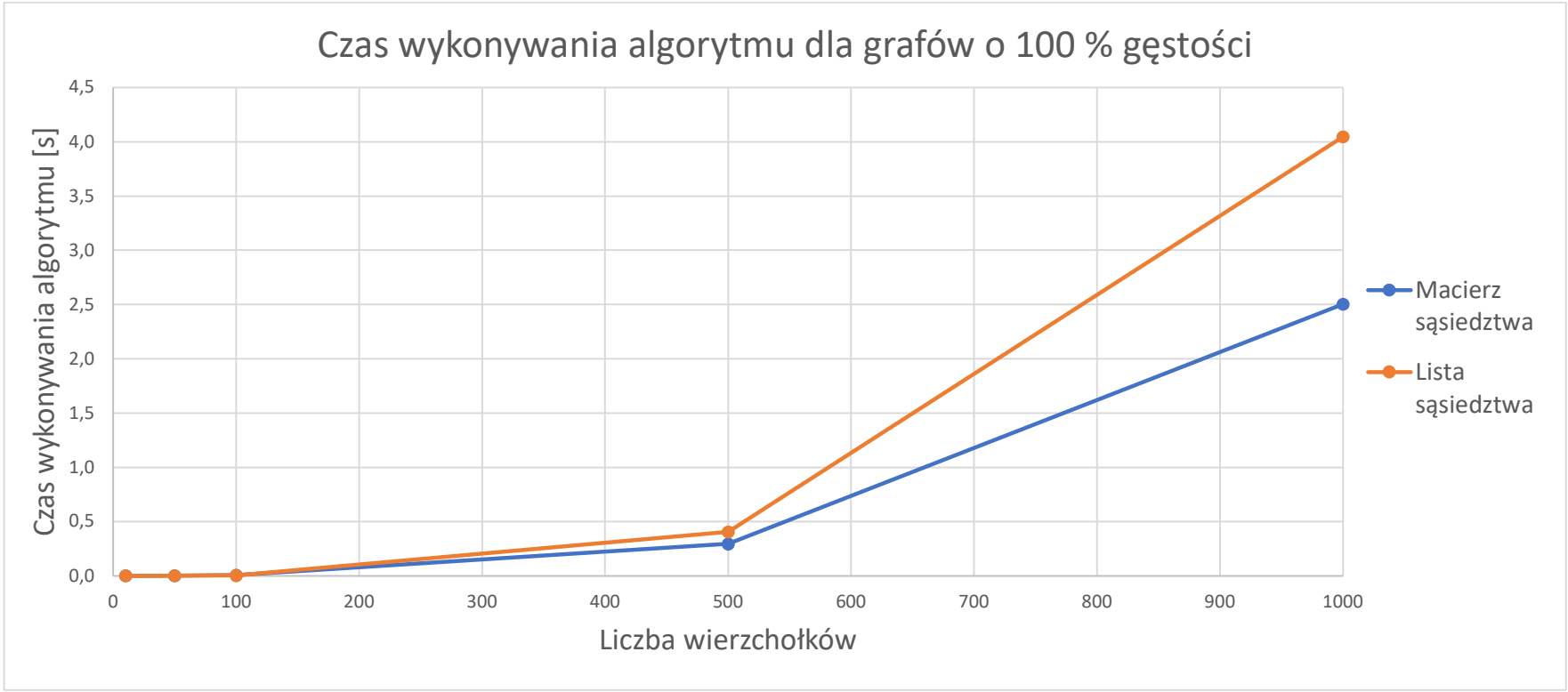
Gęstość	Liczba wierzchołków	Macierz [s]	Lista [s]
50%	10	0,000016	0,0000061
	50	0,000348	0,00020
	100	0,002059	0,00135
	500	0,083755	0,09855
	1000	0,859845	1,22837

Tabela 4 oraz Wykres 4 Czas działania algorytmu dla grafów o gęstości równej 50%



Gęstość	Liczba wierzchołków	Macierz [s]	Lista [s]
75%	10	0,000027	0,0000108
	50	0,000576	0,000341
	100	0,0031	0,0022
	500	0,1702	0,2307
	1000	1,5285	2,4200

Tabela 5 oraz Wykres 5 Czas działania algorytmu dla grafów o gęstości równej 75%



Gęstość	Liczba wierzchołków	Macierz [s]	Lista [s]
100%	10	0,0000431	0,000015
	50	0,00084	0,000506
	100	0,00444	0,0032
	500	0,29479	0,4042
	1000	2,50402	4,0459

Tabela 6 oraz Wykres 6 Czas działania algorytmu dla grafów o gęstości równej 100%

7. Podsumowanie

Z otrzymanych danych wynika, że algorytm dla obu implementacji grafu działa w dosyć podobnych czasach dla małych ilości wierzchołków, niezależnie od gęstości grafu. Jednakże, algorytm ten działa zdecydowanie lepiej dla implementacji grafu na macierzy przy większych ilościach wierzchołków. Czasy działania algorytmu są praktycznie mniejsze dla każdego z przypadków (dla małych ilości wierzchołków algorytm jest szybszy dla implementacji grafu na liście). Gęstość grafu w przypadku implementacji na macierzy nie miała aż tak dużego znaczenia, jak w przypadku implementacji opartej na liście sąsiedztwa. Wraz ze wzrostem ilości wierzchołków czas wykonywania algorytmu dla obu grafów rośnie. Na czas działania algorytmu ma wpływ nie tylko sam sposób implementacji algorytmu, ale także sposób uzyskania poszczególnych elementów grafu, jak i kolejki. Implementacja grafu na podstawie listy sąsiedztwa jest mniej optymalna pod względem złożoności obliczeniowej. Powodem jest potrzeba iteracji po elementach, aby dostać się do tego konkretnego. Implementacja ta jest jednak lepsza pod względem ilości potrzebnej pamięci, nie musimy przechowywać pustych pól, jak w przypadku macierzy.

Algorytm Dijkstry dla implementacji grafu na macierzy ma złożoność $O(V^2)$, jednak w moim wypadku może być ona trochę większa, ponieważ zaimplementowałam macierz jako klasę i potrzebuję czasu, aby dostać się do danych elementów. Algorytm Dijkstry dla implementacji grafu opartej na liście ma złożoność O(E*log V), ponieważ na drugim wykresie możemy zauważyć, że od pewnego momentu wykres rośnie szybko do góry.

Bibliografia

- https://en.wikipedia.org/wiki/Graph_(abstract_data_type) - data dostępu do strony, ostatni raz: 21.04.20 r.
- https://eduinf.waw.pl/inf/utils/002_roz/ol011.php - data dostępu do strony, ostatni raz: 21.04.20 r.
- https://en.wikipedia.org/wiki/Adjacency_list - data dostępu do strony, ostatni raz: 21.04.20 r.
- Piotr Wróblewski, *Algorytmy, struktury danych i techniki programowania. Wydanie V,* Helion, 2015.