04.05.2020r.

Paweł Troszczyński 248925

Prowadzący zajęcia: mgr inż. Marcin Ochman

Termin zajęć: wtorek 15:15-16:55

Projektowanie Algorytmów i Metody Sztucznej Inteligencji
Grafy

1. Wstęp

Celem projektu jest zaimplementowanie grafów w postaci listy sąsiedztwa oraz macierzy sąsiedztwa oraz implementacja algorytmu wyszukiwania najkrótszej ścieżki od wybranego wierzchołka do wszystkich pozostałych oraz zbadanie jego efektywności zależnie od liczby wierzchołków, gęstości grafu oraz reprezentacji grafu.

2. Opis wybranego algorytmu:

Algorytm Dijkstry pozwala na znalezienie najkrótszej ścieżki od wybranego wierzchołka do wszystkich pozostałych. Warunkiem działania algorytmu jest to, aby wszystkie wagi krawędzi łączących wierzchołki były nieujemne. W przypadku wystąpienia wag ujemnych należy wykorzystać wolniejszy algorytm Bellmana-Forda, który mimo wolniejszego działania pozwala obliczać drogi zawierające wagi ujemne. Algorytm Dijkstry pozwala określić całkowity koszt najkrótszej drogi oraz ciąg wierzchołków, po których należy się poruszać, aby wyznaczona droga była tą najkrótszą. Algorytm wykorzystuje kolejkę priorytetową do rozważania aktualnie w danym momencie najkrótszej drogi (oznacza to, że priorytetem kolejki jest aktualnie wyznaczona najmniejsza droga). Dodatkowo algorytm poza najkrótszą ścieżką przechowuje poprzedni wierzchołek, co pozwala na określenie wszystkich wierzchołków jakie należy przejść i w jakiej kolejności, aby dotrzeć do konkretnego wierzchołka. Złożoność obliczeniowa tego algorytmu zależy od liczby wierzchołków V oraz liczby krawędzi grafu E, zaś o rzędzie złożoności decyduje implementacja kolejki priorytetowej. Implementacja przez tablicę daje złożoność $O(V^2)$, przez kopiec – O(ElogV), przez kopiec Fibonacciego O(E+VlogV). Pierwsza implementacja jest lepsza dla grafów gęstych(duża liczba krawędzi w stosunku do liczby wierzchołków), zaś druga dla grafów rzadkich(mała liczba krawędzi w stosunku do liczby wierzchołków).

3. Przebieg eksperymentów:

Dla każdej z dwóch reprezentacji grafu(poprzez listę i macierz sąsiedztwa) wygenerowano po 100 instancji składających się z 10, 50, 100 ,500 i 1000 wierzchołków oraz dla 4 różnych gęstości:

- 25%
- 50%
- 75%
- 100% - graf pełny

Dokonano pomiaru czasu wykonywania algorytmu dla każdej z wymienionych kombinacji oraz uśredniono otrzymane wyniki. Liczba krawędzi zależna od przyjętej gęstości została obliczona zgodnie ze wzorem:

$$\mathrm{E} = \mathrm{V}*(\mathrm{V}\text{-}1)*\text{gęstość}$$

oraz wynik został zaokrąglony w górę (np. dla 10 wierzchołków max krawędzi to 90, dla gęstości 25% wychodzi 22,5 więc przyjęto 23 krawędzie).

Wartości wag danych krawędzi były losowane z przedziału od 1 do 100. Przy generowaniu każdego z grafów na początku był generowany graf pełny, następnie obliczana była liczba krawędzi do usunięcia. Program losował krawędzie do usunięcia, jednak aby zapewnić możliwość dostępu z wybranego wierzchołka do każdego innego (uniknąć sytuacji, w której powstały by dwa osobne grafy lub nie dało by się dostać do pewnego wierzchołka) nie usuwano żadnych z krawędzi prowadzących od oraz do ostatniego wierzchołka.

Dodatkowo w programie jest możliwość wczytania grafu z pliku tekstowego oraz zapisanie wyników działania algorytmu do wskazanego pliku. Plik tekstowy musi mieć odpowiedni format danych: w pierwszej linii muszą być kolejno oddzielone przerwami ilość krawędzi, ilość wierzchołków oraz wierzchołek startowy. Każda kolejna linia określa krawędzie grafu, których zdefiniowane musi być tyle ile podane zostało w pierwszej linijce pliku. Format dla krawędzi grafu to:

wierzchołek_początkowy wierzchołek_końcowy waga

W pliku wynikowym znajdują się informację o najmniejszym koszcie drogi od wierzchołka początkowego do wszystkich innych wierzchołków z ciągiem wierzchołków od wierzchołka początkowego, które należy przebyć, aby uzyskać tę wagę.

W programie zaprezentowane jest przykładowe odczytanie grafu z pliku „example_graph.txt", zastosowanie dla niego algorytmu Dijkstry oraz zapisanie wyników działania tego algorytmu do pliku „result.txt". Przy przeprowadzaniu pozostałych badań nie zapisywano otrzymanych wyników do pliku, ponieważ spowolniłoby to znacząco działanie programu. Przed przeprowadzeniem eksperymentów sprawdzono także poprawność generowania grafów oraz poprawność działania algorytmów na grafie składającym się z 10 wierzchołków.

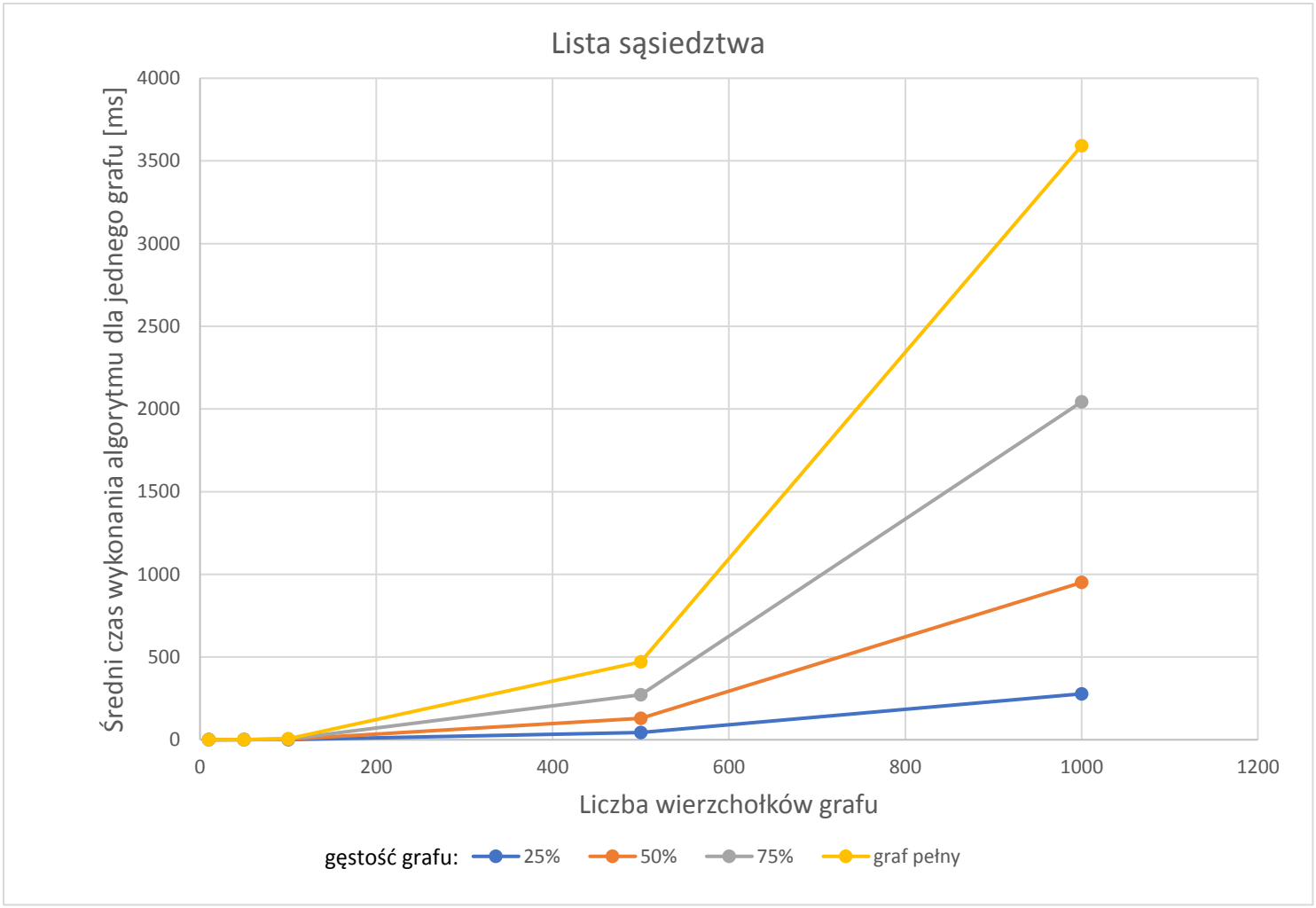
3. Otrzymane wyniki:

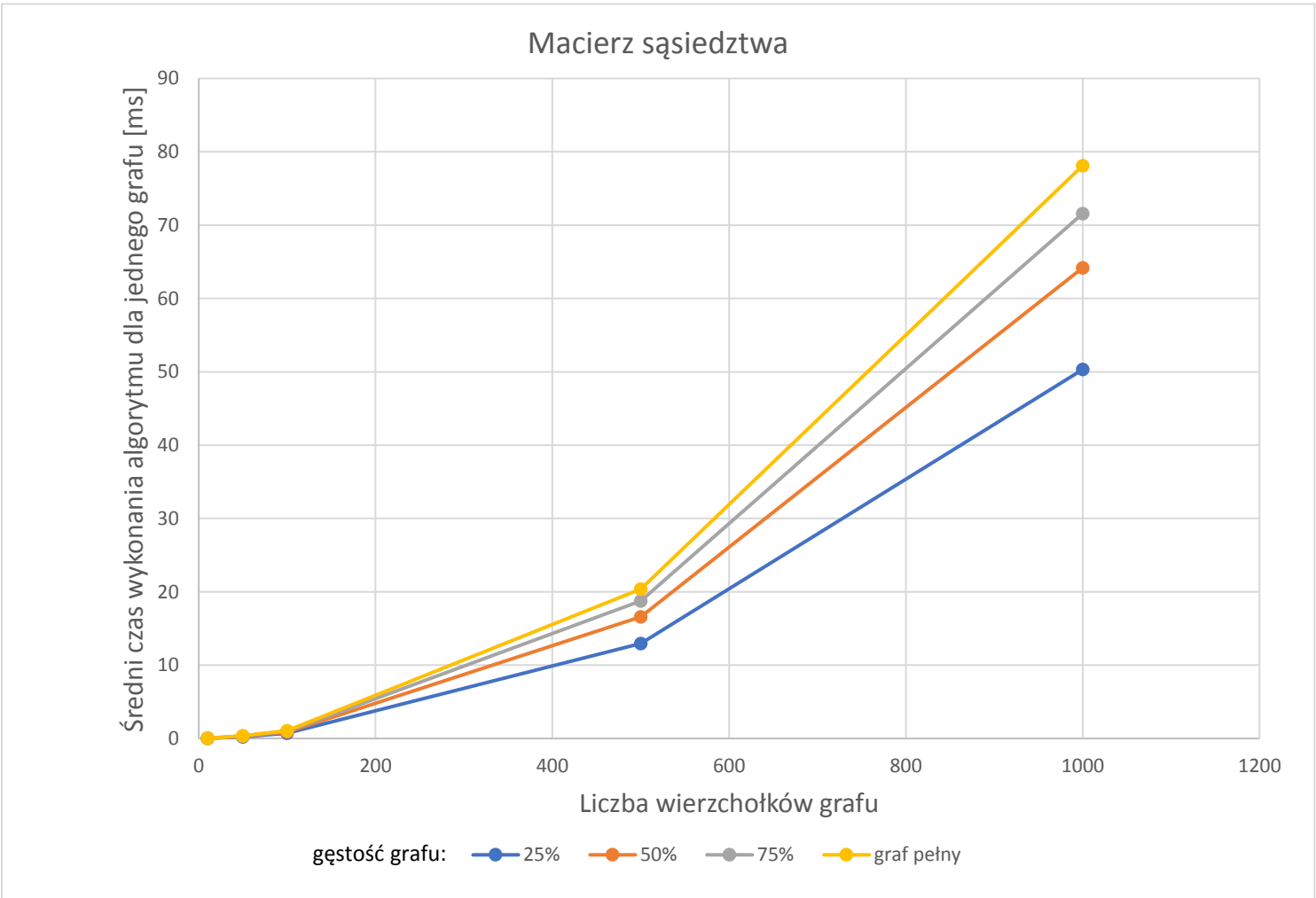
lista sąsiedztwa					
gęstość		25%	50%	75%	100%
liczba elementów	10	0,026	0,034	0,040	0,047
	50	0,284	0,462	0,678	0,985
	100	1,048	2,056	3,440	5,325
	500	43,071	129,092	271,257	471,381
	1000	277,527	950,922	2042,690	3591,190

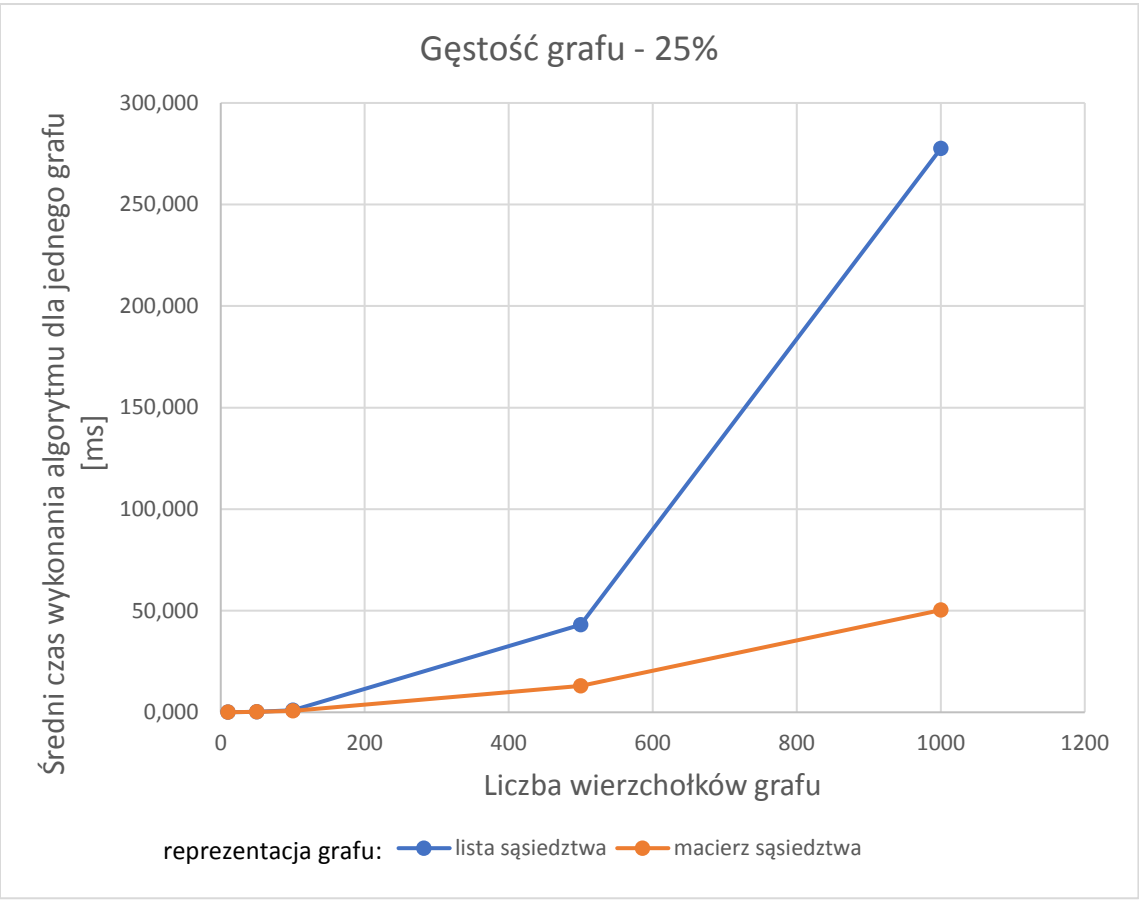
Tabela przedstawiająca średni czas w ms dla listy sąsiedztwa o zadanej gęstości i liczbie elementów dla algorytmu Dijkstry

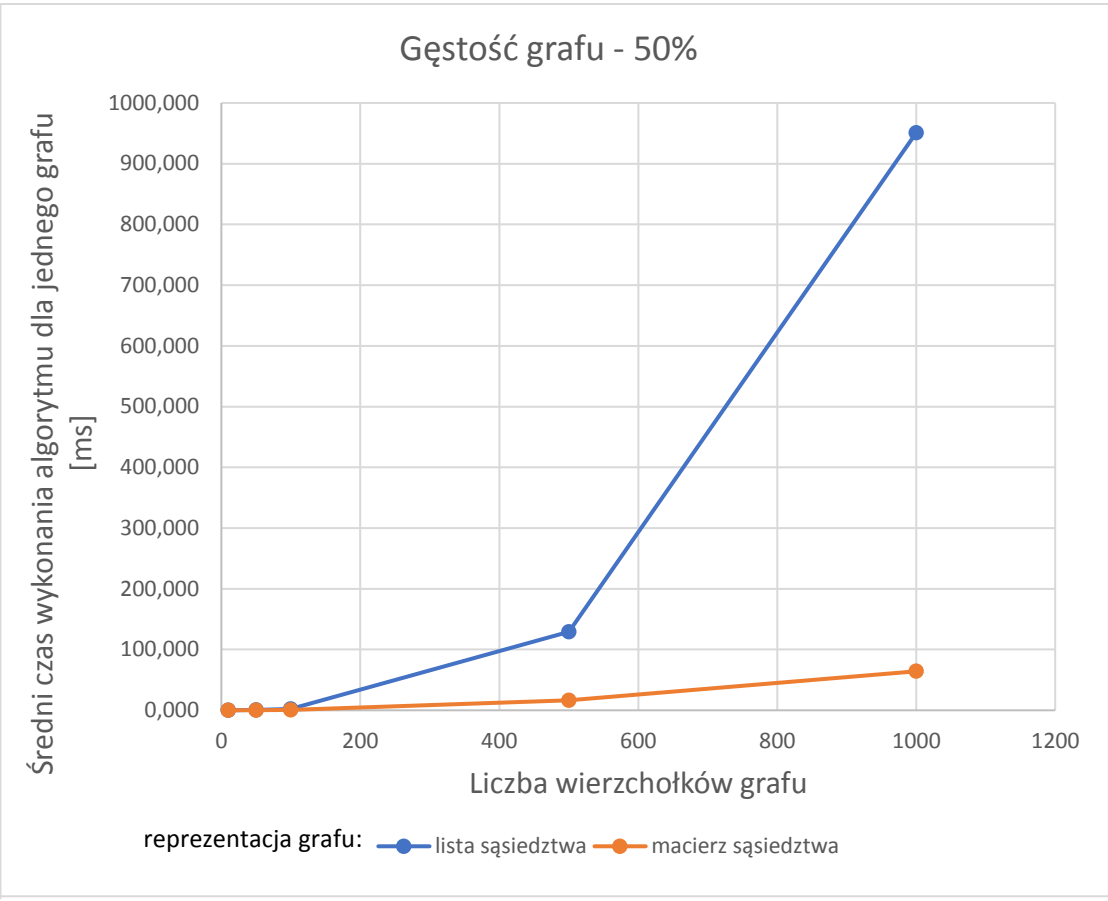
macierz sąsiedztwa					
gęstość		25%	50%	75%	100%
liczba elementów	10	0,025	0,029	0,033	0,037
	50	0,233	0,300	0,348	0,380
	100	0,706	0,869	0,980	1,074
	500	12,940	16,572	18,758	20,374
	1000	50,312	64,186	71,578	78,114

Tabela przedstawiająca średni czas w ms dla macierzy sąsiedztwa o zadanej gęstości i liczbie elementów dla algorytmu Dijkstry


Lista sąsiedztwa
Średni czas wykonania algorytmu dla jednego grafu [ms]
4000
3500
3000
2500
2000
1500
1000
500
0
0
200
400
600
800
1000
1200
Liczba wierzchołków grafu
gęstość grafu:
25%
50%
75%
graf pełny

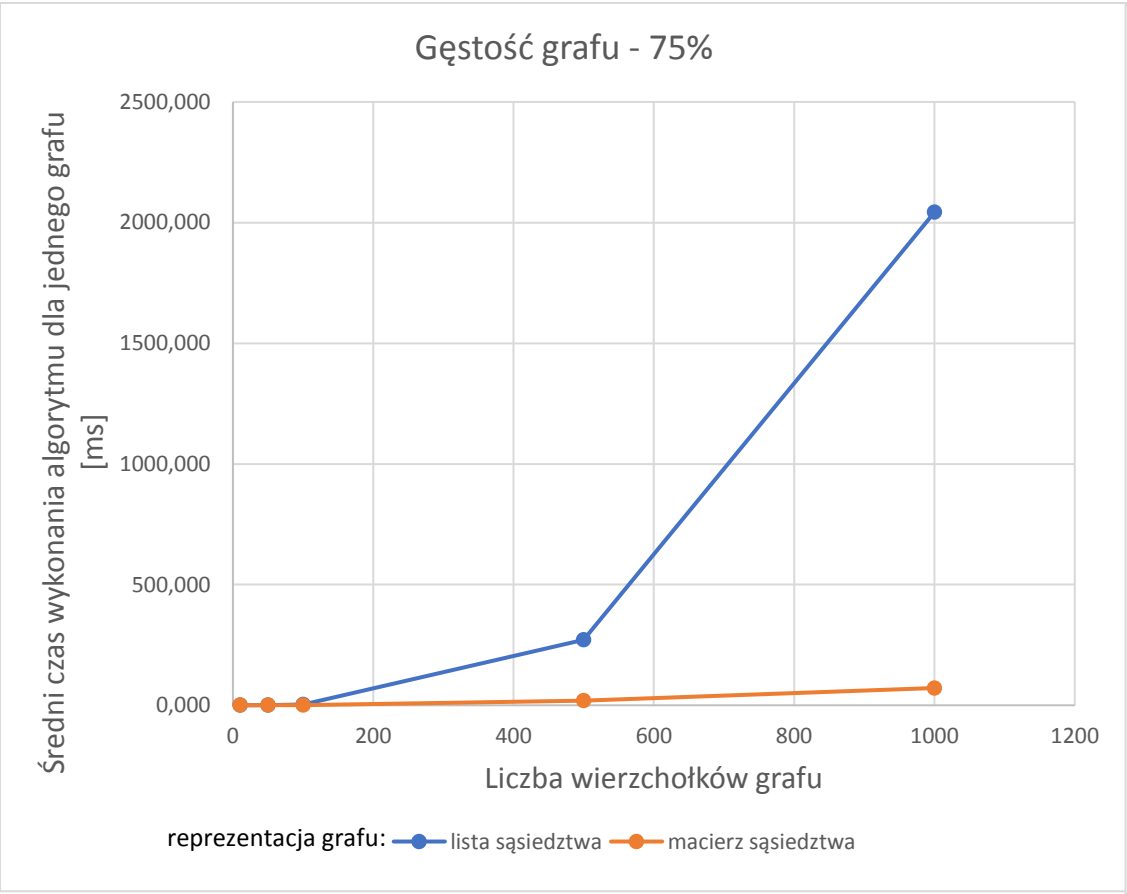

Macierz sąsiedztwa
Średni czas wykonania algorytmu dla jednego grafu [ms]
90
80
70
60
50
40
30
20
10
0
0
200
400
600
800
1000
1200
Liczba wierzchołków grafu
gęstość grafu:
25%
50%
75%
graf pełny

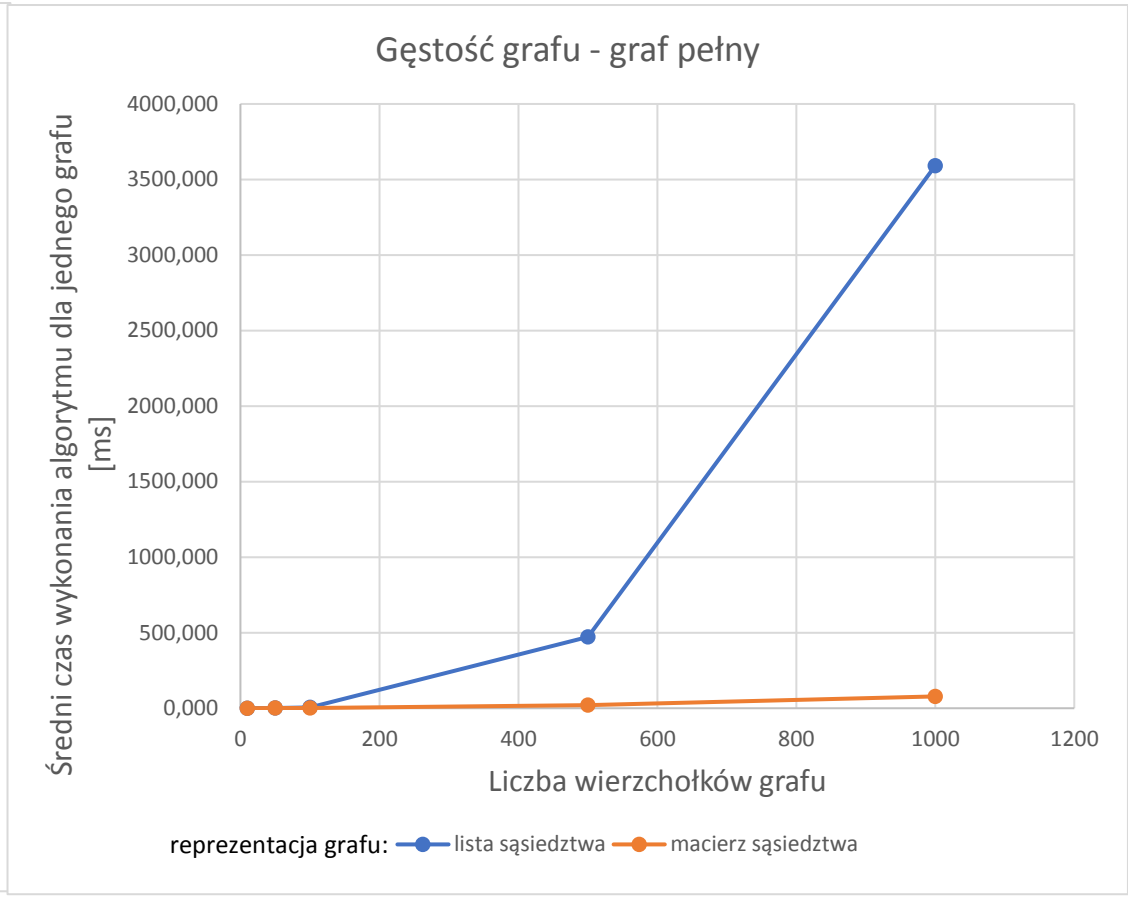




25%			
reprezentacja grafu		lista	macierz
liczba elementów	10	0,026	0,025
	50	0,284	0,233
	100	1,048	0,706
	500	43,071	12,940
	1000	277,527	50,312

50%			
reprezentacja grafu		lista	macierz
liczba elementów	10	0,034	0,029
	50	0,462	0,300
	100	2,056	0,869
	500	129,092	16,572
	1000	950,922	64,186





75%			
reprezentacja grafu		lista	macierz
liczba elementów	10	0,040	0,033
	50	0,678	0,348
	100	3,440	0,980
	500	271,257	18,758
	1000	2042,690	71,578

graf pełny			
reprezentacja grafu		lista	macierz
liczba elementów	10	0,047	0,037
	50	0,985	0,380
	100	5,325	1,074
	500	471,381	20,374
	1000	3591,190	78,114

4. Podsumowanie i wnioski

- W każdym z badanych przypadków algorytm Dijkstry jest wykonywany szybciej, gdy graf reprezentowany jest jako macierz sąsiedztwa niż jako lista sąsiedztwa
- Im graf ma więcej krawędzi (większa gęstość grafu) tym więcej czasu potrzebne jest, aby algorytm wykonał się. Algorytm dla mniejszej liczby krawędzi musi zbadać mniej możliwych dróg co znacząco przyspiesza jego działanie
- Wpływ gęstości różni się zależnie od reprezentacji grafu. W przypadku listy sąsiedztwa zmniejszenie gęstości grafu zmniejsza czas wykonywania algorytmu dużo bardziej niż w przypadku macierzy sąsiedztwa. Np. porównując czas dla 1000 wierzchołków i spadku gęstości ze 100% do 75%. Dla reprezentacji grafu jako lista sąsiedztwa średni czas potrzebny do wykonania algorytmu spada o ok. 43%, podczas, gdy dla tych samych wartości dla macierzy sąsiedztwa spada tylko o ok. 8,5%. Oznacza to, że gęstość grafu ma dużo większy wpływa na graf reprezentowany przez listę sąsiedztwa niż macierz sąsiedztwa
- W przypadku grafów o niewielkiej ilości wierzchołków różnica w średnim czasie między listą a macierzą sąsiedztwa są niewielkie. Jednak im więcej wierzchołków grafu tym różnica w średnim czasie potrzebnym do wykonania algorytmu powiększa się, co powoduje, że lepiej pod względem czasowym wybrać reprezentację grafu jako macierz sąsiedztwa

Literatura:

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
https://en.wikipedia.org/wiki/Complete_graph
https://pl.wikipedia.org/wiki/Graf_(matematyka)
https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
https://pl.wikipedia.org/wiki/Reprezentacja_grafu