Badanie złożoności obliczeniowej algorytmu Djikstry w zależności od reprezentacji, gęstości i ilości wierzchołków w grafie

Daniel Śliwowski

nr ideksu: 241166, Termin: ŚR 11:15, Data: 07.05.2019,
Prowadzący: Dr inż. Łukasz Jeleń

1 Cel badania

Celem badania jest eksperymentalne sprawdzenie złożoności obliczeniowej zaimplementowanego algorytmu Djikstry w zależności od reprezentacji, gęstości i ilości wierzchołków w grafie.

Reprezentacje dzielimy na :

- Lista sąsiedztwa,
- Macierz sąsiedztwa.

Wybrane ilości wierzchołków:

- 10,
- 50,
- 100,
- 250.

Wybrane gęstości:

- 0%,
- 25%,
- 50%,
- 75%,
- 100%.

2 Przebieg badania

W wyniku nieoptymalnego zaimplementowania struktur danych i algorytmu badania wykonano dla różnej ilości instancji w zależności od ilości wierzchołków i gęstości grafu. Maksymalna ilość instancji była równa 50, dla par gęstość-ilość wierzchołków, przy których algorytm wykonywał się szybko, A minimalna ilość - 5 dla pary 100% - 250. Następnie obliczono średnie czasy działania algorytmu dla każdej pary, wyniki zostały przedstawione w rozdziale "Wyniki badań" w dalszej części sprawozdania.

3 Algorytm Djikstry

Algorytm Djikstry służy do znalezienia najkrótszej ścieżki pomiędzy dwoma wierzchołkami w grafie ważonym o nieujemnych wagach. W szczególności może być to przypadek pomiędzy jednym wierzchołkiem a pozostałymi.

Zaimplementwoany algorytm ma postać:

3.1 Teoretyczna złożoność obliczeniowa

Pesymistyczna złożoność obliczeniowa algorytmu Djikstry została podana i wynosi
$\mathbf{O(E + Vlog(V))}$, gdzie E to ilość krawędzi, a V ilość wierzchołków.

```
Data: Graf g, wierzchołek startowy s, wierzchołek końcowy e
Result: Struktura wynik przechowująca długość drogi i ścieżkę
begin
    wierzchoki ⟵ g.Vertices
    Niech D będzie kopcem przechowującym elementy powiązujące wierzhołek z odległością i poprzednim
      wierzchołkiem
    Niech w będzie stukturą przechowującą wynik.
    {Jeżeli nie ma krawędzi to są nie podłączone, zwróć -1}
    if s.IncidentEdges.IsEmpty or end.IncidentEdges.IsEmpty then
        w.droga = -1
        return w
    end
    {Ustawienie początkowych odległości i poprzednich wierzchołków}
    foreach wierzchołek in wierzchołki do
        i ⟵ 0
        D[i].SetVertex(wierzchołek)
        if wierzchołek = start then
            D[i].SetDis(0)
        else
            D[i].SetDis(∞)
        end
        D[i].SetVia(NULL)
        ++i
    end
    while !D.IsEmpty do
        {Utwórz kopiec }
        D.Heapify
        {Usuś element minimaly }
        elem ⟵ D.RemovMin
        foreach krawędź in elem.GetVertex.IncidentEdges do
            sasiad ⟵ krawed.opposite(elem.GetVertex)
            {Wyszukaj go w kopcu }
            foreach element in D do
                i ⟵ 0
                if D[i].GetVertex() = sasiad then
                    {Dokonaj relaksacji krawedzi }
                    if elem.Distance + krawd.GetElem < D[i].Distance then
                        D[i].Distance ⟵ elem.Distance + krawd.GetElem
                        D[i].SetVia(elem.GetVertex())
                    end
                end
                ++i
            end
        end
        {Jeżeli dojdziemy do elementu końcowego zapisz trase i droge}
        if elem.GetVertex = e then
            tmp ⟵ elem
            {Znajdź poprzedni wierzchołek w kopcu}
            do
                w.AddFront(tmp.GetVertex)
                foreach elem in D do
                    i ⟵ 0
                    if D[i].GetVertex = tmp.GetVia then
                        tmp = D[i]
                    end
                end
            while tmp.GetVertex ≠ NULL
            w.droga = elem.GetDistance()
            return w;
        end
    end
end
```

Algorytm 1: Djikstra(g, start, end)

4 Wyniki badań

Rozdział ten zostanie podzielony na dwie części, pierwsza będzie przedstawiać wyniki działania algorytmu w zależności od gęstości grafu, a druga w zależności od sposobu implementacji.

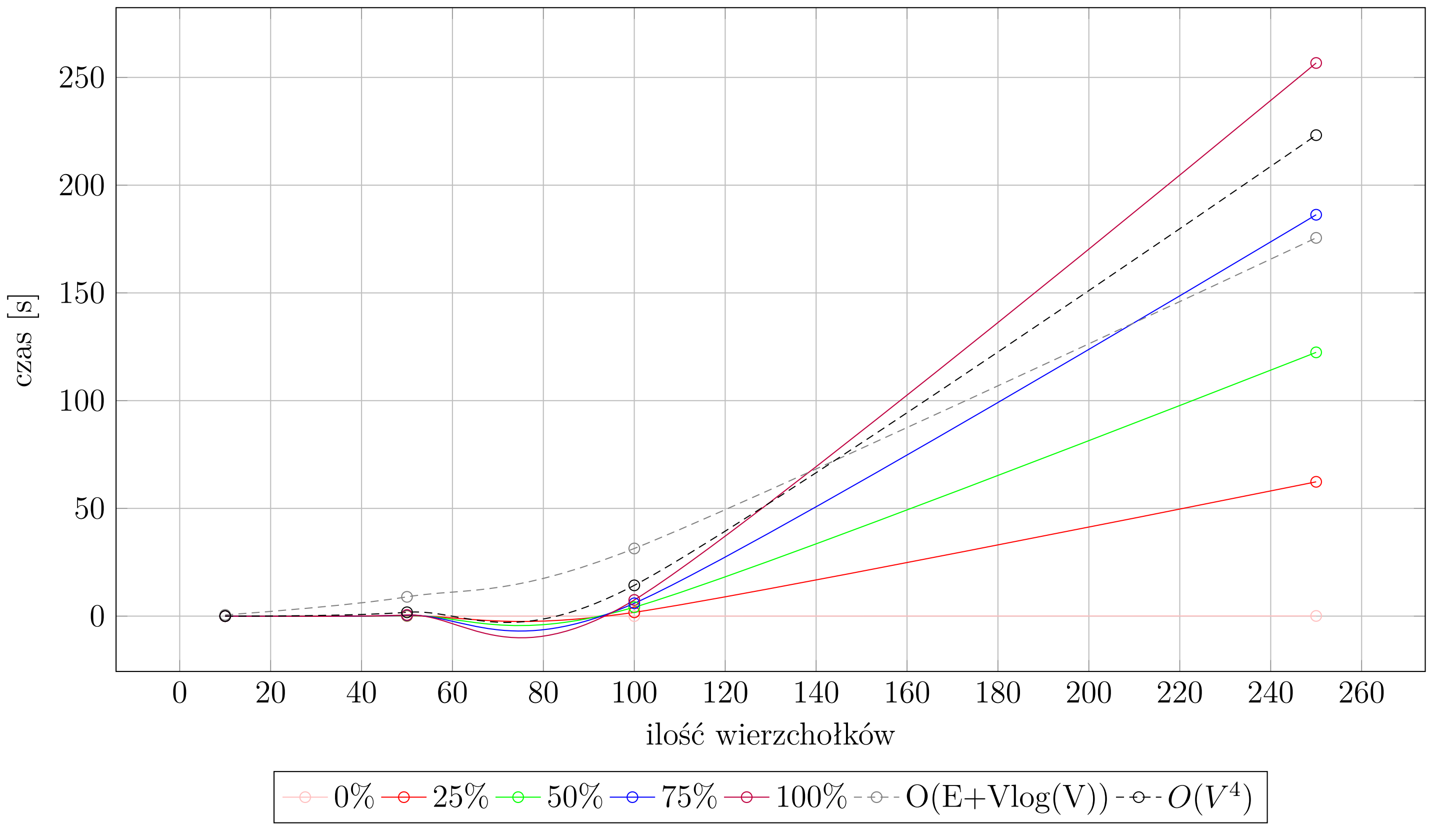
4.1 Wyniki w zależności od gęstości

Poniżej zaprezentowano wyniki badań czasu działania algorytmu w zależności od gęstości grafu. Pierwsza tabela dotyczy implementacji grafu za pomocą listy sąsiedztwa, a druga za pomocą macierzy sąsiedztwa.

Tabela 1: Zależność czasu działania od ilości wierzchołków, dla gęstości będącej parametrem, dla grafu reprezentowanego za pomocą Listy sąsiedztwa

V \ G	0%	25%	50%	75%	100%
10	0.00027s	0.00127s	0.00199s	0.00257s	0.00318s
50	0.00386s	0.15301s	0.27745s	0.41142s	0.53554s
100	0.01447s	1.7634s	4.14885s	5.95736s	7.49931s
250	0.08367s	62.33029s	122.4115s	186.2725s	256.7052s

Wykres 1: Zależność czasu o ilości wierzchołków dla listy sąsiedztwa

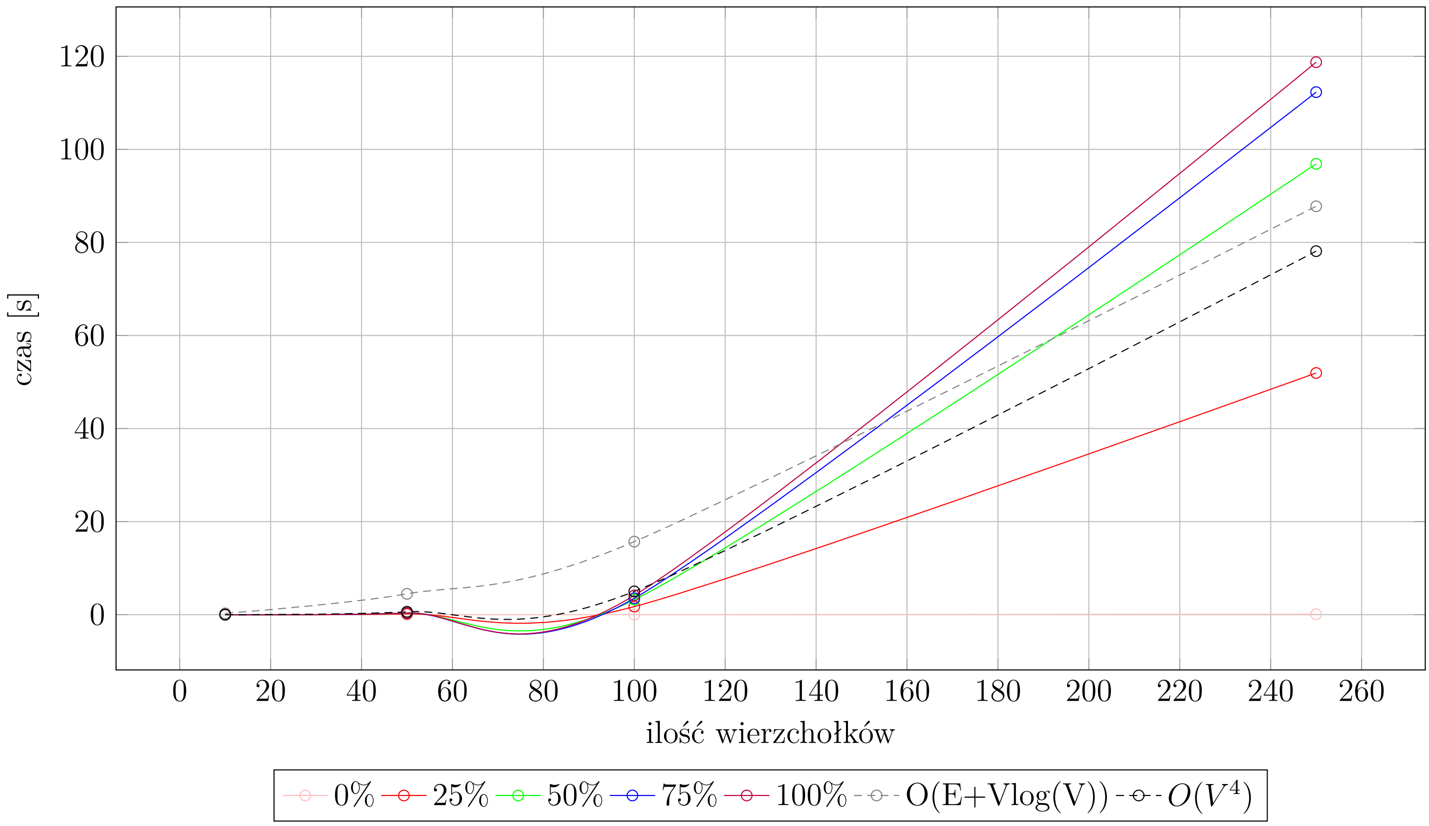


Na podstawie Wykresów 1. i 2. można stwierdzić, że złożoność obliczeniowa zaimplementowanego algorytmu nie zgada się z tą podaną w projekcie. Wynika to najprawdopodobniej przez sposób powiązania dystansu do wierzchołka. W wynik zaproponowanej metody wyszukanie elementu w kopcu zajmuje $\mathbf{O(V)}$ co wraz z czasem $\mathbf{O(V)}$ dla sprawdzenia wszystkich elementów w

Tabela 2: Zależność czasu działania od ilości wierzchołków, dla gęstości będącej parametrem, dla grafu reprezentowanego za pomocą macierzy sąsiedztwa

V \ G	0%	25%	50%	75%	100%
10	0.00035s	0.00131s	0.00169s	0.00197s	0.00205s
50	0.00418s	0.17326s	0.29432s	0.34129s	0.35636s
100	0.01627s	1.73981s	3.20421s	3.51609s	4.13321s
250	0.09288s	51.94416s	96.90067s	112.3039s	118.7434s

Wykres 2: Zależność czasu o ilości wierzchołków dla macierzy sąsiedztwa



kopcu oraz czasem $\mathbf{O(V)}$ dla sprawdzenia wszystkich krawędzi incydentalnych do wierzchołka daje łączną złożoność równą $\mathbf{O}(V^3)$, co zgadza się z linią na wykresach (czarna).

Zauważa się również powiązanie pomiędzy gęstością grafu, a czasem szukania najkrótszej ścieżki. Im pełniejszy jest graf, tym dłużej zajmuje jej znalezienie. Zależność ta zgadza się z tą podaną wraz z projektem.

4.2 Wyniki w zależności od reprezentacji grafu

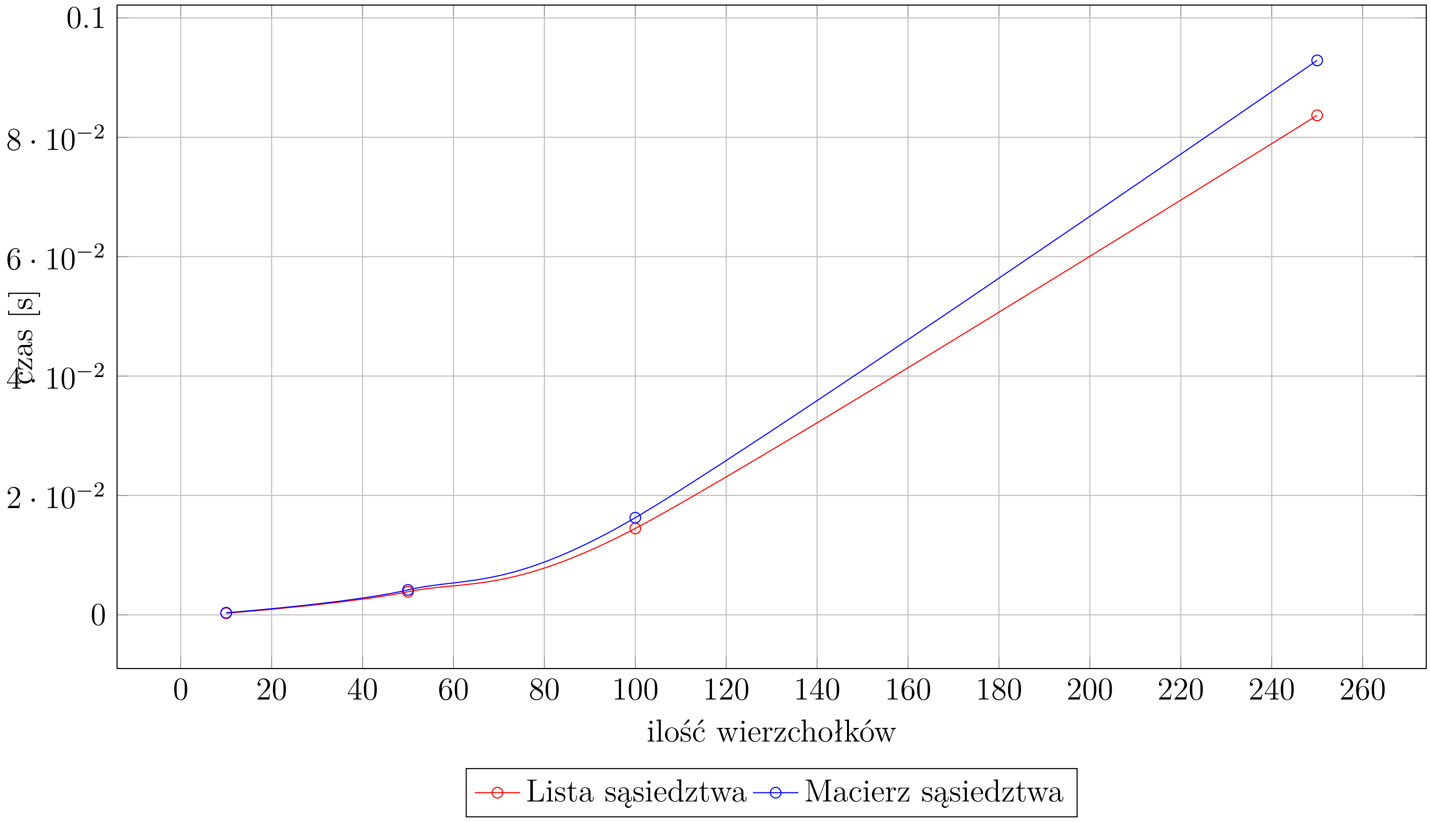
Poniżej znajdują się wyniki badań czasu działania w zależności od typu reprezentacji grafu. Wykresy będą odpowiedio dla gęstości równych 0%, 25%, 50%, 75%, 100%. W tabeli LS oznacza listę sąsiedztwa, a MS macierz sąsiedztwa.

Tabela 3: Zależność czasu działania od rodzaju reprezentacji grafu

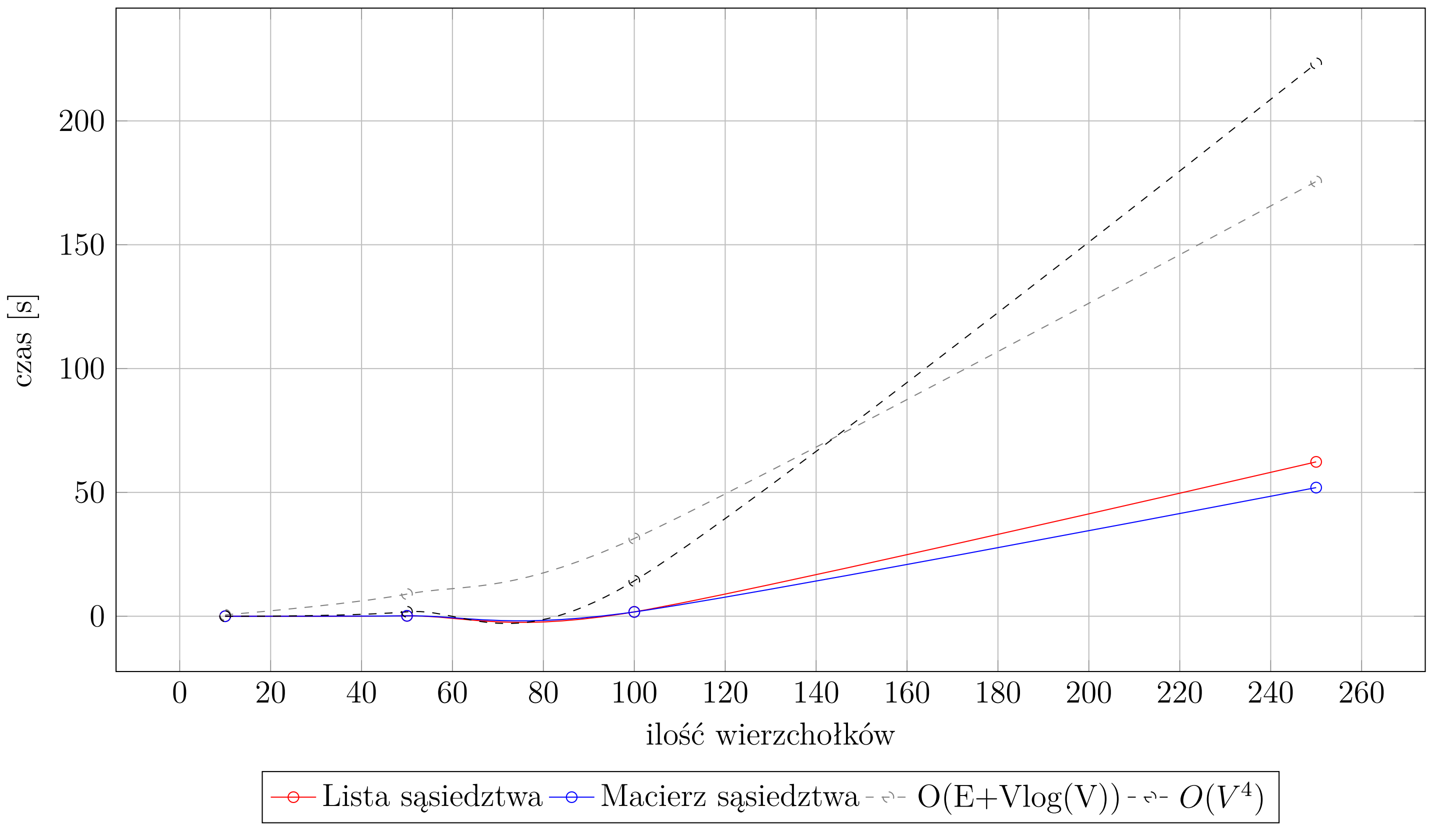
G	rep	10	50	100	250
0	LS	0.000267s	0.003864s	0.01447s	0.083673s
	MS	0.000349s	0.004177s	0.016274s	0.092885s
25	LS	0.001268s	0.15301s	1.7634s	62.33029s
	MS	0.001307s	0.173259s	1.739813s	51.94416s
50	LS	0.001985s	0.277445s	4.148853s	122.4115s
	MS	0.001686s	0.294318s	3.20421s	96.90067s
75	LS	0.002568s	0.411423s	5.957361s	186.2725s
	MS	0.001972s	0.341292s	3.516089s	112.3039s
100	LS	0.003182s	0.535544s	7.499306s	256.7052s
	MS	0.002047s	0.356363s	4.133209s	118.7434s

Na podstawie Wykresów 3-7 stwierdza się, że reprezentacja grafu poprzez macierz sąsiedztwa daje lepsze wyniki działania algorytmu dla prawie każdej wartości gęstości. Wyjątkiem jest przypadek gdy jest ona równa 0% i wiąże się to najprawdopodobniej z szybszym sprawdzaniem, czy wierzchołki mają incydentne krawędzie.

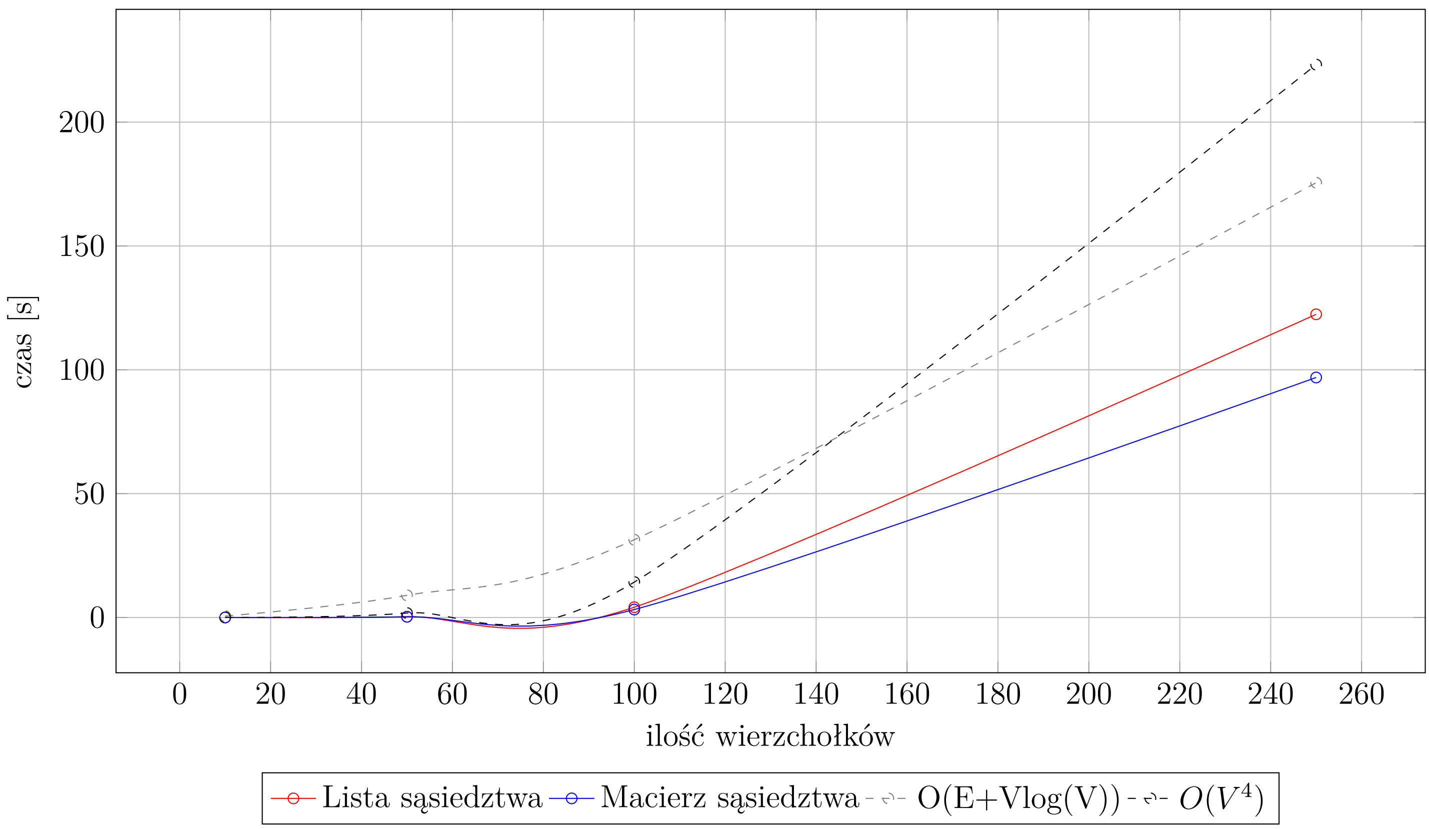
Wykres 3: Zależność czasu o ilości wierzchołków dla gęstości równej 0%



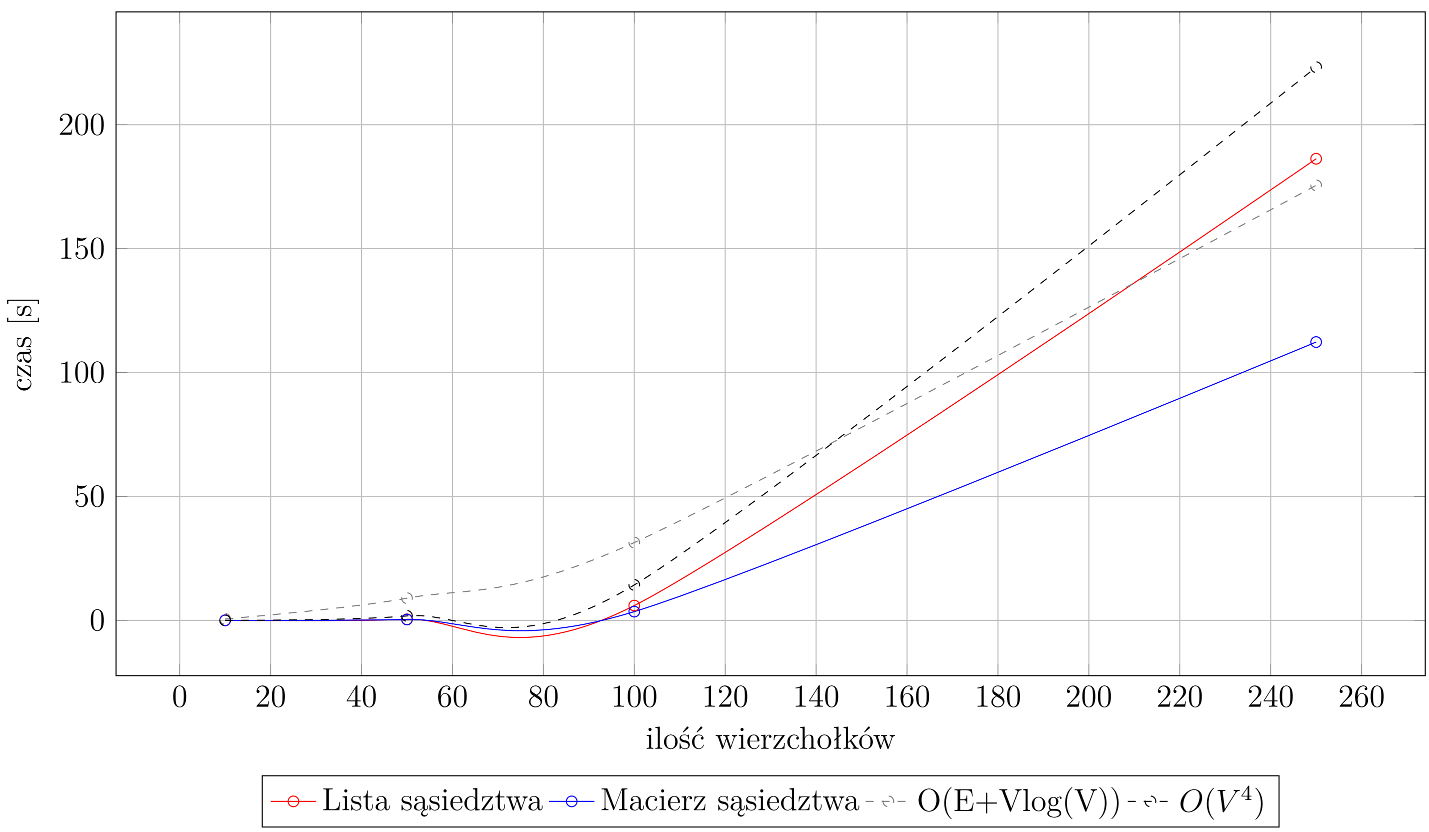
Wykres 4: Zależność czasu o ilości wierzchołków dla gęstości równej 25%



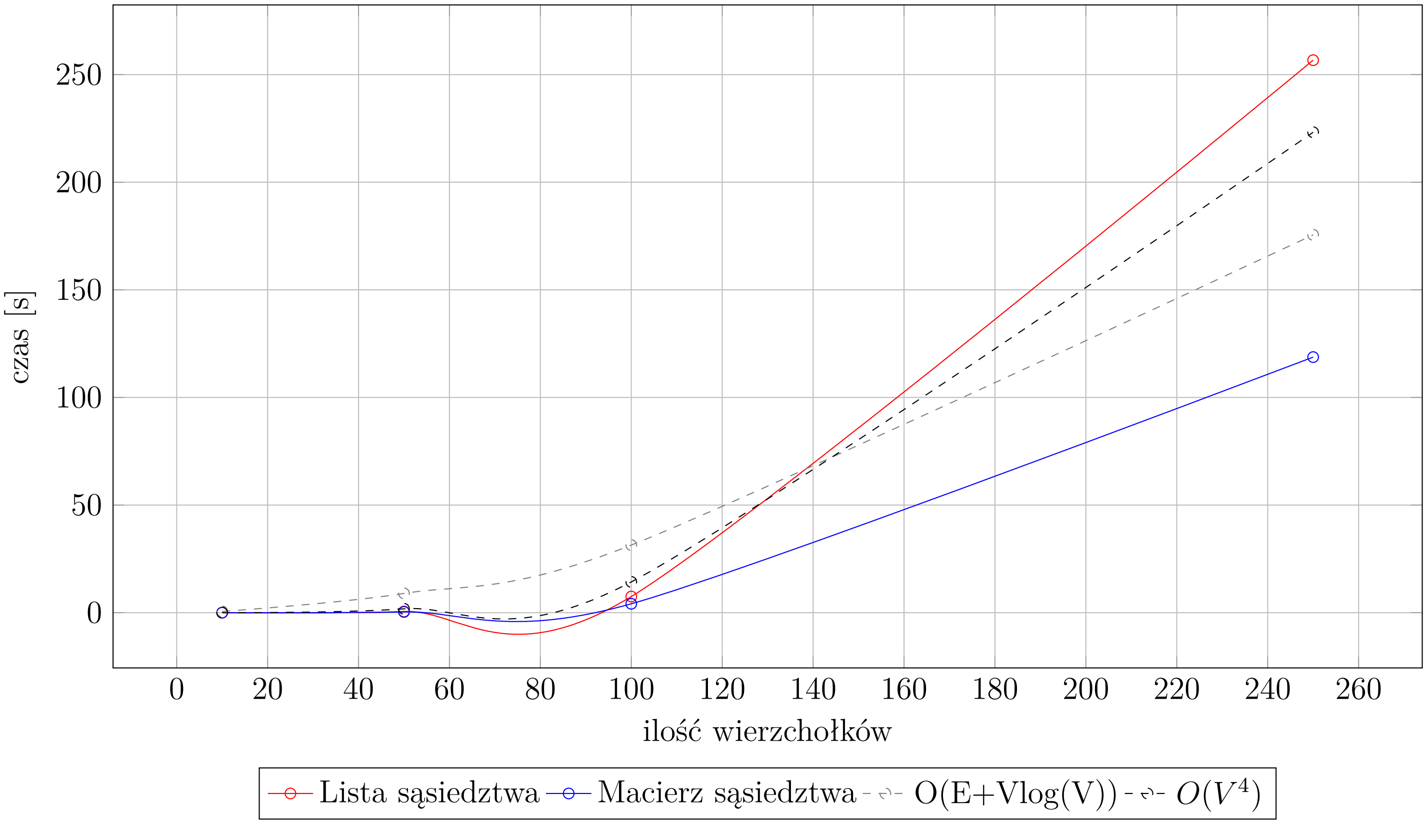
Wykres 5: Zależność czasu o ilości wierzchołków dla gęstości równej 50%



Wykres 6: Zależność czasu o ilości wierzchołków dla gęstości równej 75%



Wykres 7: Zależność czasu o ilości wierzchołków dla gęstości równej 100%



5 Wnioski

Złożoność obliczeniowa zaimplementowanego algorytmu Djikstry do znajdywania najkrótszej ścieżki w grafie ważonym nie zgadza się z tą podaną w zadaniu projektowym. Na podstawie badań stwierdza się, że wynosi ona $\mathbf{O}(V^3)$ i, jak zostało wcześniej wspomniane, wiąże się to najprawdopodobniej z zaproponowaną implementacją powiązania długości ścieżki z wierzchołkiem. Złożoność tą obrazują wykresy 1 i 2.

Na podstawie wykresów 1 i 2 zauważa się również, że na czas działania algorytmu ma wpływ gęstość grafu. Im pełniejszy był graf, tym dłużej zajmowało znalezienie ścieżki, wynika to z tego, że algorytm dla każdego wierzchołka musi sprawdzić więcej krawędzi.

Wykresy 3-7 pokazują zależność czasu działania od sposobu reprezentacji grafu. Na ich podstawie można stwierdzić, że algorytm wykonuje się szybciej dla grafów reprezentowanych za pomocą macierzy sąsiedztwa. Jedynym wyjątkiem była gęstość grafu równa 0% i wiąże się to najprawdopodobniej z szybszym czasem sprawdzania istnienia indycentnych krawędzi dla reprezentacji przez listę sąsiedztwa.

6 Sugerowane ulepszenia

Na podstawie wyciągniętych wniosków proponuje się następujące usprawnienia algorytmu:

1. Ulepszenie metody powiązania wierzchołka z odległością: W momencie, gdy przyjmie się, że wierzchołki przechowują elementy typu całkowitego i że każdy z nich przechowuje inną liczbę z zakresu 0-V, to tablicę odległości można przedstawić za pomocą zwykłej tablicy, w której wartość elementu wierzchołka stanowi indeks odpowiedniej odległości.

2. Problem z przeciekami pamięci: Zastosowanie inteligentnych wskaźników i uważniejsze zwalnianie pamięci.

Literatura

[1] Michael T. Goodrich, Roberto Tamassia, David Mount. Data Structures and Algorithms in C++ Second Edition.

[2] Generowanie losowego grafu [07.05.2019] - https://www.geeksforgeeks.org/erdos-renyl-model-generating-random-graphs/