Autor: **282651** Maksym Kostyshyn

Algorytmy sortowania
Projekt 2
Projektowanie i Analiza Algorytmów

Kod kursu: **W04ISA-SI0013G**

1 Wstęp

Celem projektu była implementacja i analiza trzech algorytmów grafowych: **przeszukiwania grafu w głąb (DFS)**, **algorytmu Dijkstry** oraz **algorytmu Bellmana-Forda**. W ramach zadania przeprowadzono eksperymenty mające na celu ocenę efektywności każdego z algorytmów na losowo generowanych grafach o różnej liczbie wierzchołków i gęstości, a także zbadano wpływ sposobu reprezentacji grafu (macierz sąsiedztwa oraz lista sąsiedztwa) na czas działania algorytmów.

2 Opis zaimplementowanych algorytmów

2.1 DFS (Depth-First Search)

Algorytm przeszukiwania grafu w głąb (DFS) eksploruje graf, podążając jak najdalej od wierzchołka początkowego wzdłuż nieodwiedzonych wierzchołków, zanim się cofnie. Jest często wykorzystywany do sprawdzania spójności grafu, wykrywania cykli lub topologicznego sortowania.

Złożoność czasowa:
- Dla listy sąsiedztwa: $O(V + E)$
- Dla macierzy sąsiedztwa: $O(V^2)$

2.2 Algorytm Dijkstry

Algorytm Dijkstry służy do wyznaczania najkrótszych ścieżek z jednego wierzchołka do wszystkich innych w grafie o **nieujemnych wagach krawędzi**. Wykorzystuje strukturę danych kolejki priorytetowej do efektywnego wyboru wierzchołka z najmniejszym aktualnym dystansem.

Złożoność czasowa:
- Dla listy sąsiedztwa: $O((V + E) \log V)$
- Dla macierzy sąsiedztwa: $O(V^2)$

2.3 Algorytm Bellmana-Forda

Algorytm Bellmana-Forda również znajduje najkrótsze ścieżki z jednego wierzchołka do pozostałych, ale **obsługuje grafy z krawędziami o ujemnych wagach**. W odróżnieniu od algorytmu Dijkstry, wykonuje relaksację wszystkich krawędzi wielokrotnie. **Może także wykrywać cykle o ujemnej długości**.

Złożoność czasowa:
- Dla listy sąsiedztwa: $O(V \cdot E)$
- Dla macierzy sąsiedztwa: $O(V \cdot E)$

3 Działanie algorytmów

3.1 DFS (Depth-First Search)

Implementacja DFS działa w sposób rekurencyjny lub z użyciem stosu (w moim przypadku sposób rekurencyjny):

1. Inicjalizacja - algorytm rozpoczyna działanie od wybranego wierzchołka startowego.
2. Oznaczanie odwiedzonych wierzchołków - każdy odwiedzony wierzchołek jest zapamiętywany, aby uniknąć cyklicznych odwiedzin.
3. Rekurencyjne przechodzenie do sąsiadów - dla każdego sąsiada nieodwiedzonego wierzchołka algorytm wywołuje się rekurencyjnie.
4. Zakończenie - algorytm kończy działanie, gdy wszystkie wierzchołki osiągalne ze startowego zostaną odwiedzone.

Dzięki tej implementacji możliwe było zweryfikowanie spójności grafu oraz testowanie struktury odwiedzin na potrzeby wizualizacji.

3.2 Algorytm Dijkstry

Implementacja algorytmu Dijkstry bazuje na użyciu kolejki priorytetowej (np. kopca) dla zapewnienia efektywnego wyboru wierzchołka z najmniejszym znanym dystansem.

1. Inicjalizacja - wszystkie wierzchołki otrzymują nieskończony dystans oprócz wierzchołka startowego, którego odległość wynosi 0.
2. Wybór wierzchołka o najmniejszym dystansie - z kolejki wybierany jest wierzchołek o najmniejszej znanej odległości.
3. Relaksacja krawędzi - dla każdego sąsiada aktualizowany jest dystans, jeśli nowa ścieżka jest krótsza.
4. Powtarzanie - czynności są powtarzane aż do odwiedzenia wszystkich wierzchołków lub opróżnienia kolejki.

Algorytm poprawnie działa tylko dla grafów o nieujemnych wagach.

3.3 Algorytm Bellmana-Forda

Bellman-Ford został zaimplementowany w klasycznej wersji z $V - 1$ iteracjami relaksacji krawędzi.

1. Inicjalizacja - podobnie jak w Dijkstrze, wszystkie dystanse ustawiane są na nieskończoność poza wierzchołkiem startowym.
2. Relaksacja wszystkich krawędzi - w każdej z $V - 1$ iteracji algorytm przechodzi przez wszystkie krawędzie i aktualizuje dystanse, jeśli znajdzie krótszą ścieżkę.
3. Wykrywanie cykli - po zakończeniu iteracji wykonywana jest dodatkowa pętla sprawdzająca, czy możliwa jest dalsza relaksacja. Jeśli tak, oznacza to istnienie cyklu o ujemnej wadze.
4. Zakończenie - algorytm zwraca listę najkrótszych dystansów lub sygnalizuje obecność cyklu.

Algorytm nadaje się do grafów z ujemnymi wagami, co czyni go bardziej ogólnym niż Dijkstra, ale mniej wydajnym.

4 Metodyka pomiarów

Dla każdego z zaimplementowanych algorytmów (DFS, Dijkstra, Bellman-Ford) przeprowadzono pomiary czasu działania w różnych warunkach. Eksperymenty uwzględniały:

- rozmiary grafów: 10, 50, 100, 200 oraz 500 wierzchołków,
- gęstości grafów: 25%, 50%, 75% oraz 100% (graf pełny),
- reprezentacje grafów: lista sąsiedztwa oraz macierz sąsiedztwa.

Dla każdej kombinacji parametrów wykonano 5 serii testów, każda z innym **seed** (250, 300, 350, 400, 450) do funkcji **std::srand()**, co daje łącznie 500 pomiarów na jedną konfigurację (100 powtórzeń · 5 seedów). Zastosowanie różnych wartości początkowych generatora liczb losowych pozwoliło uzyskać bardziej wiarygodne statystycznie wyniki.

W przypadku algorytmu Bellmana-Forda wykorzystano dodatkowo grafy z ujemnymi wagami krawędzi, aby w pełni zweryfikować jego poprawne działanie także w sytuacjach, w których inne algorytmy (np. Dijkstra) nie znajdują zastosowania.

Pomiar czasu realizowany był z użyciem funkcji **std::chrono::high_resolution_clock**, a średnie wyniki zapisano w pliku **results/Results.txt**.

Wykorzystany sprzęt:

Testy zostały przeprowadzone na laptopie **Lenovo Legion Slim 5 16AHP9**

- procesor AMD Ryzen 7 8845HS w/ Radeon 780M Graphics 3.80 GHz
- RAM 32,0 GB
- 64-bitowy system operacyjny, procesor oparty na architekturze x64.

5 Wyniki oraz wykresy czasów sortowania

5.1 Dane pomiarowe

Tabela 1: Porównanie czasów operacji algorytmów grafowych dla **gęstości 25%** (czasy w [μs])

Algorytm / Struktura	**10 wierz.**	**50 wierz.**	**100 wierz.**	**200 wierz.**	**500 wierz.**
DFS – Lista	0.98	17.06	64.34	244.41	1484.09
DFS – Macierz	0.16	0.45	0.92	2.25	5.49
Dijkstra – Lista	2.58	58.11	200.79	729.67	3904.03
Dijkstra – Macierz	3.24	76.40	305.94	1273.41	8226.66
Bellman-Ford – Lista	4.06	440.83	3536.78	27244.80	460609.00
Bellman-Ford – Macierz	4.99	661.45	5399.13	42900.60	1150550.00

Tabela 2: Porównanie czasów operacji algorytmów grafowych dla **gęstości 50%** (czasy w [μs])

Algorytm / Struktura	**10 wierz.**	**50 wierz.**	**100 wierz.**	**200 wierz.**	**500 wierz.**
DFS – Lista	1.80	30.99	121.87	478.36	2819.70
DFS – Macierz	0.20	0.75	1.55	3.70	8.26
Dijkstra – Lista	5.17	94.17	374.66	1301.41	7157.78
Dijkstra – Macierz	7.00	141.88	587.43	2400.00	14637.50
Bellman-Ford – Lista	7.00	875.85	6924.48	54362.80	998190.00
Bellman-Ford – Macierz	6.64	950.85	7731.85	100062.00	1850540.00

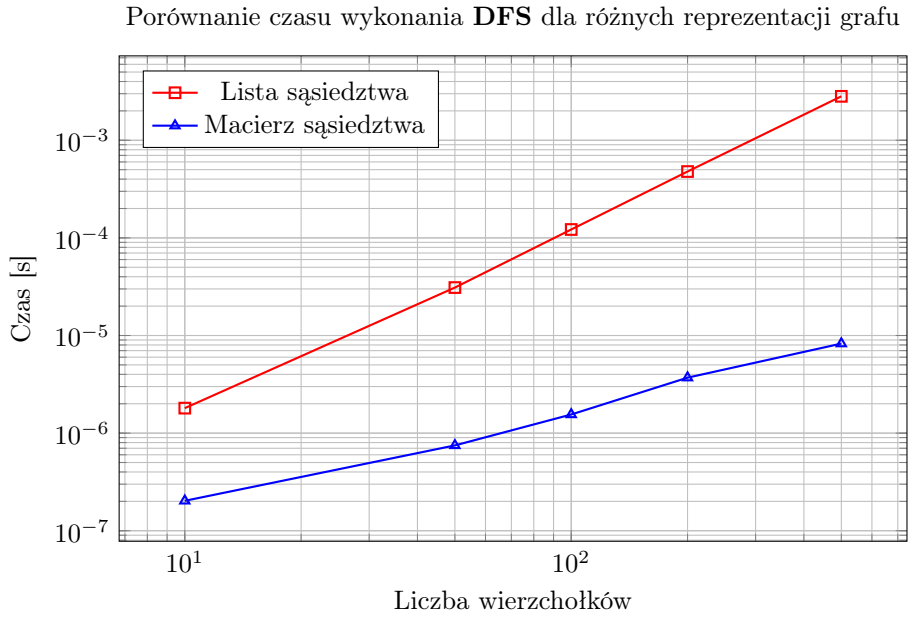
Tabela 3: Porównanie czasów operacji algorytmów grafowych dla **gęstości 75%** (czasy w [μs])

Algorytm / Struktura	**10 wierz.**	**50 wierz.**	**100 wierz.**	**200 wierz.**	**500 wierz.**
DFS – Lista	2.39	44.60	170.84	700.14	4159.22
DFS – Macierz	0.27	0.89	1.89	3.94	9.10
Dijkstra – Lista	7.09	135.56	495.14	1829.14	10289.30
Dijkstra – Macierz	9.92	189.40	781.43	2930.26	17451.20
Bellman-Ford – Lista	10.07	1288.56	10078.70	80665.80	1520990.00
Bellman-Ford – Macierz	8.86	1170.96	9286.59	100214.00	1830790.00

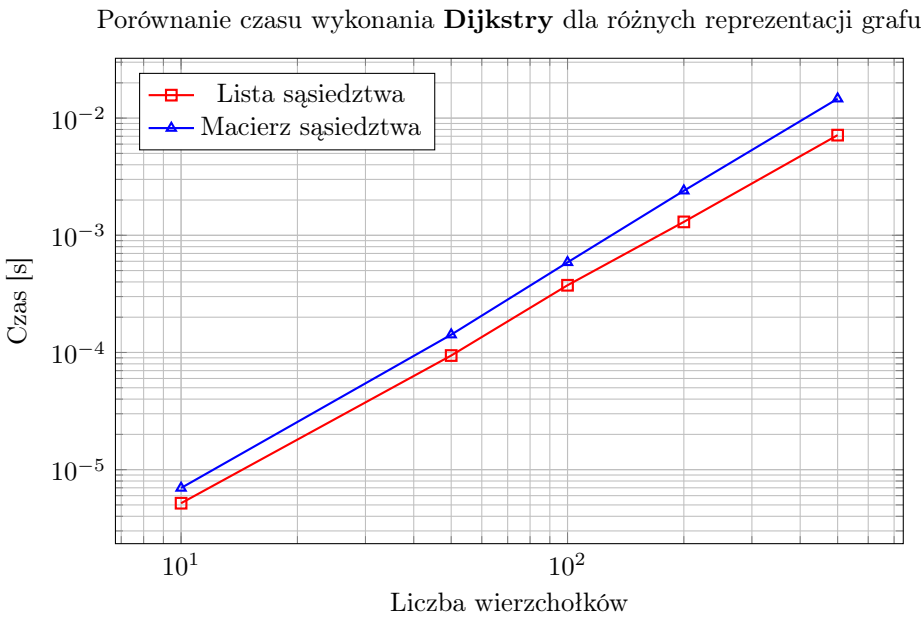
Tabela 4: Porównanie czasów operacji algorytmów grafowych dla **gęstości 100%** (czasy w [μs])

Algorytm / Struktura	**10 wierz.**	**50 wierz.**	**100 wierz.**	**200 wierz.**	**500 wierz.**
DFS – Lista	2.94	59.37	224.92	915.05	5493.41
DFS – Macierz	0.30	1.08	1.96	4.01	9.38
Dijkstra – Lista	8.59	174.95	625.90	2347.43	13715.50
Dijkstra – Macierz	10.40	216.67	887.56	3203.40	17324.80
Bellman-Ford – Lista	12.88	1736.72	13433.80	107633.00	2076130.00
Bellman-Ford – Macierz	11.06	1406.82	10822.40	107505.00	1892230.00

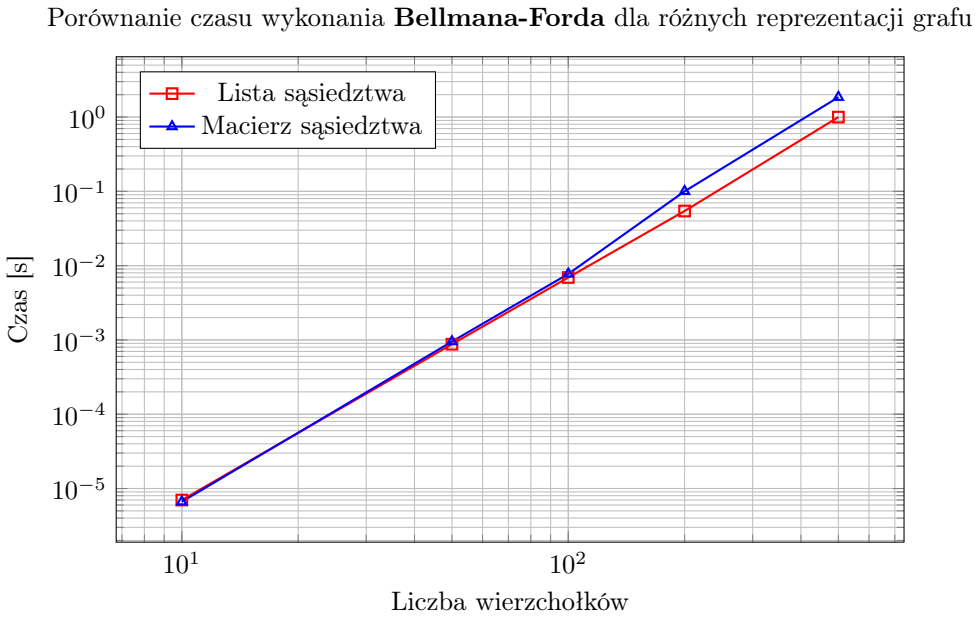
5.2 Porównanie algorytmów dla różnych reprezentacji grafu



Rysunek 1: Porównanie czasu wykonania **DFS** dla listy i macierzy sąsiedztwa (**gęstość grafu 50%**).

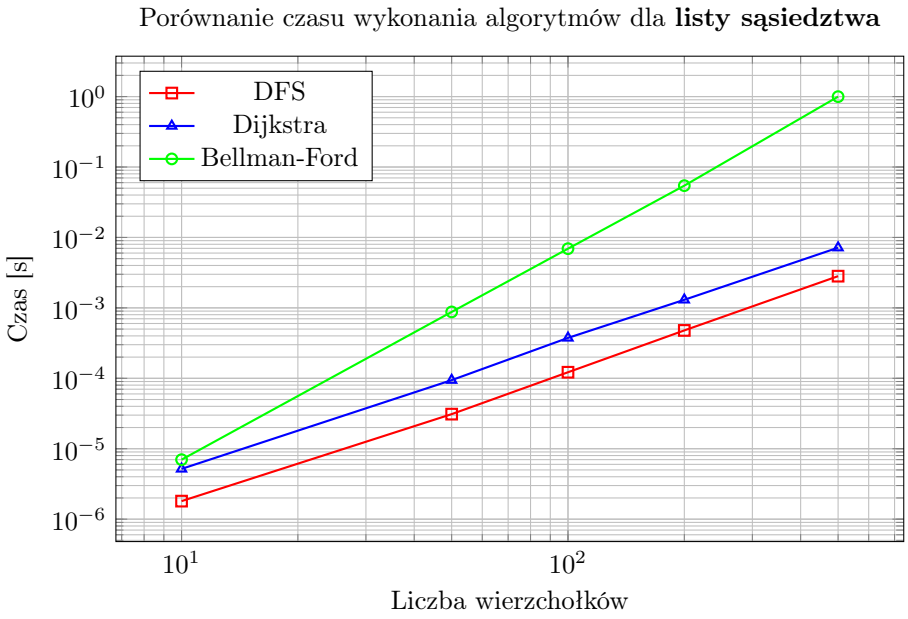


Rysunek 2: Porównanie czasu wykonania **Dijkstry** dla listy i macierzy sąsiedztwa (**gęstość grafu 50%**).

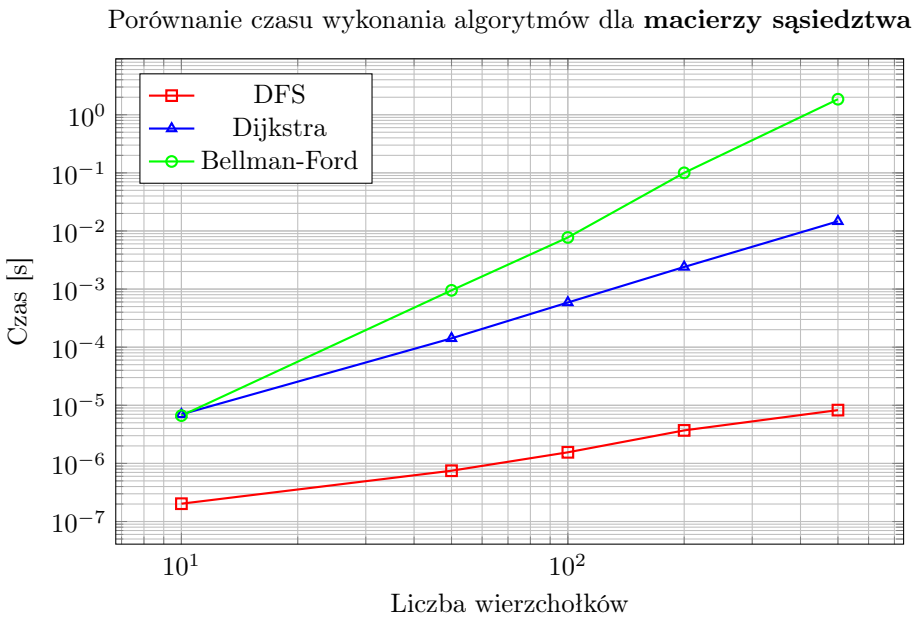


Rysunek 3: Porównanie czasu wykonania **Bellmana-Forda** dla listy i macierzy sąsiedztwa (**gęstość grafu 50%**).

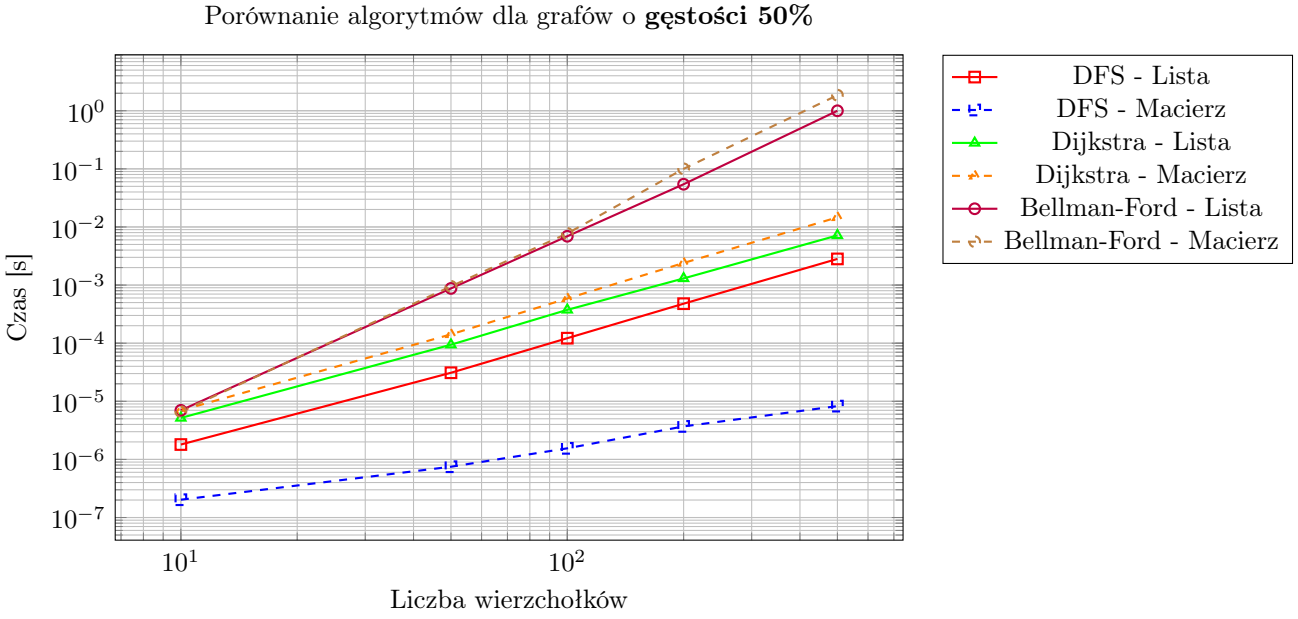
5.3 Porównanie wszystkich algorytmów dla listy i macierzy sąsiedztwa



Rysunek 4: Porównanie czasu wykonania algorytmów dla listy sąsiedztwa (**gęstość grafu 50%**).

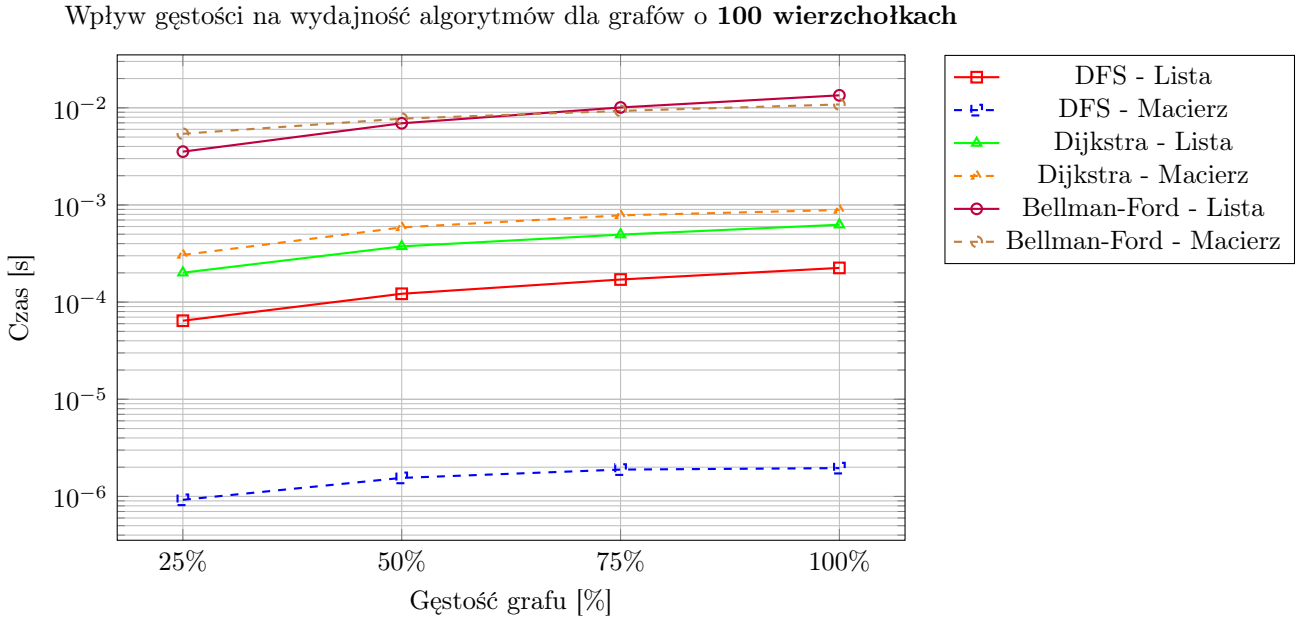


Rysunek 5: Porównanie czasu wykonania algorytmów dla macierzy sąsiedztwa (**gęstość grafu 50%**).

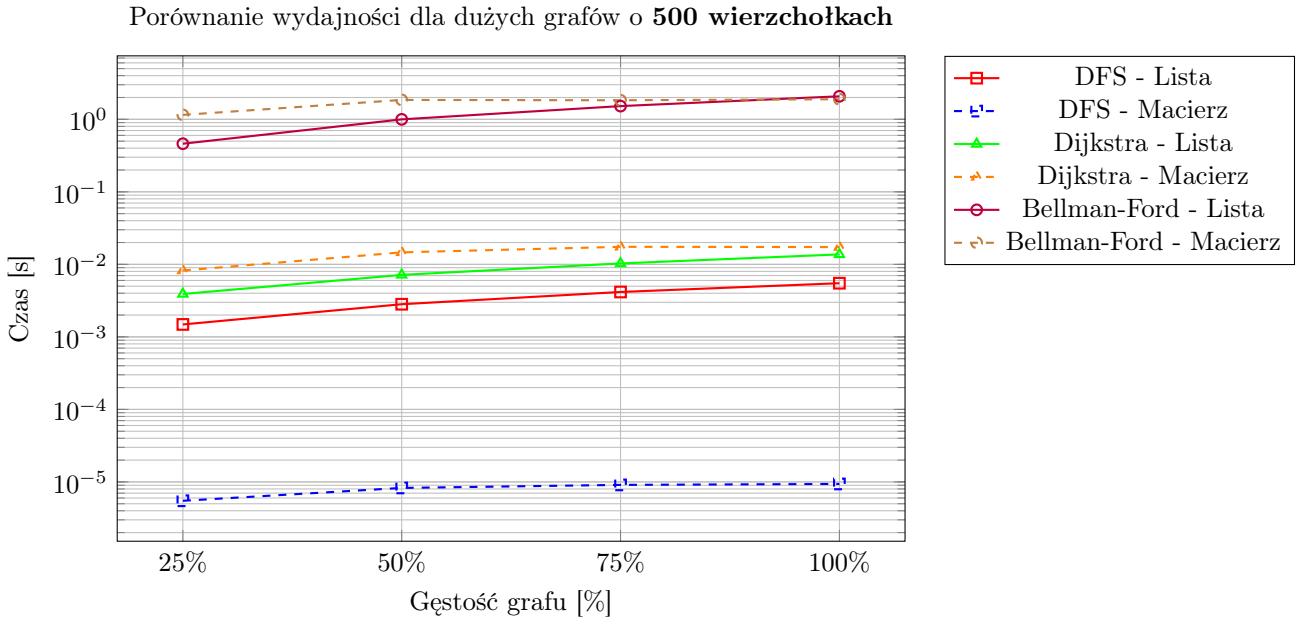


Rysunek 6: Porównanie czasów wykonania algorytmów grafowych dla grafów o **gęstości 50%**.

5.4 Wpływ gęstości na wydajność algorytmów dla grafów o różnych wierzchowkach

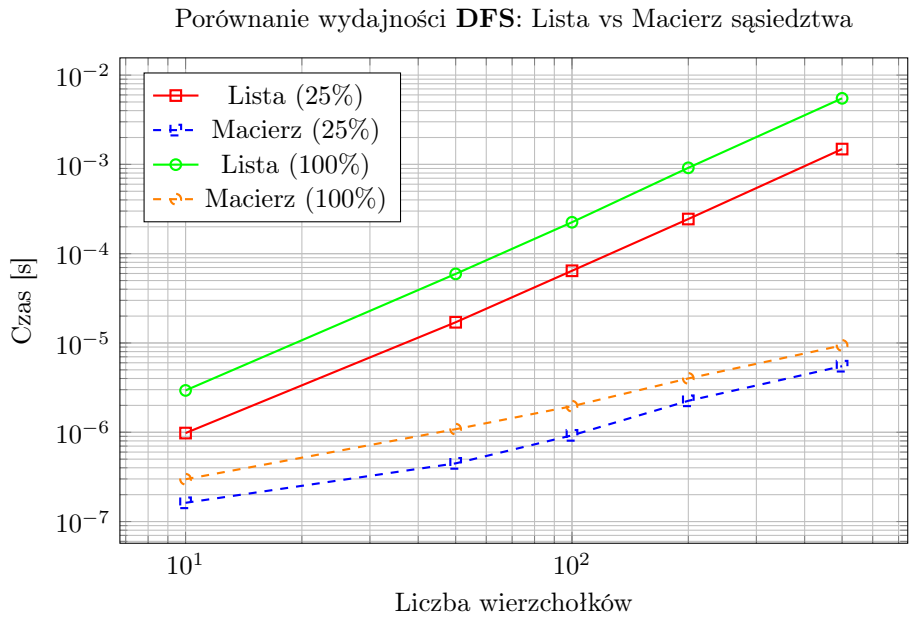


Rysunek 7: Wpływ gęstości grafu na wydajność algorytmów dla grafów o **100 wierzchołkach**.

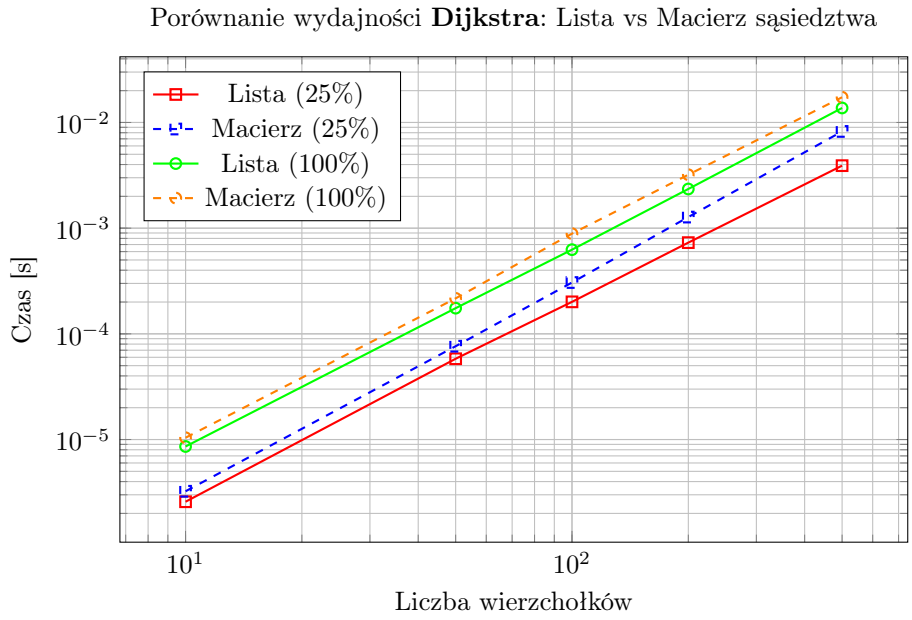


Rysunek 8: Porównanie wydajności algorytmów dla grafów o **500 wierzchołkach** w zależności od gęstości.

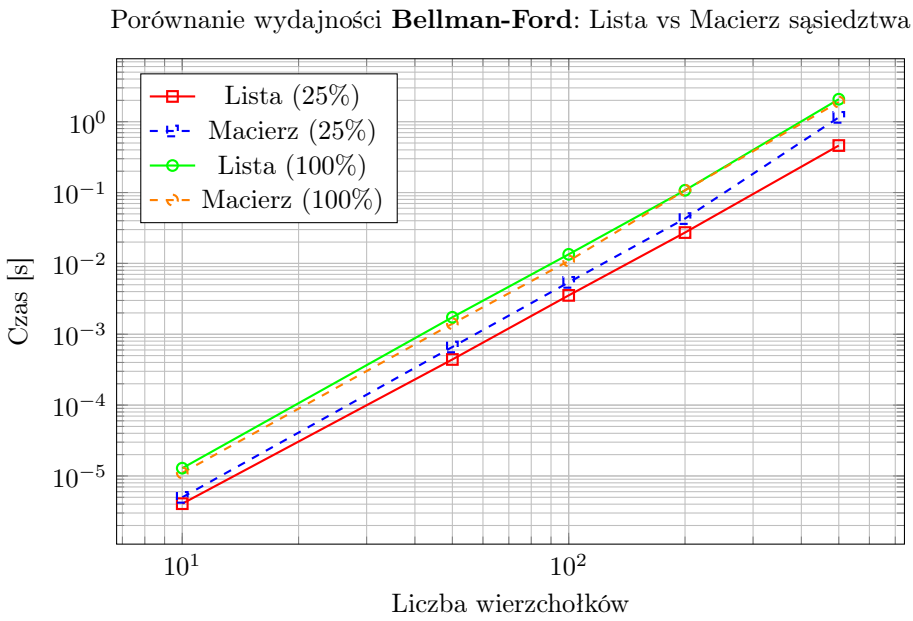
5.5 Porównanie algorytmów dla różnych implementacji



Rysunek 9: Porównanie wydajności DFS dla **listy i macierzy sąsiedztwa** przy różnych gęstościach.

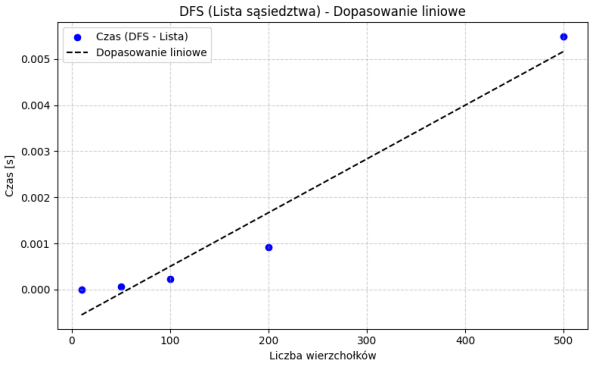


Rysunek 10: Porównanie wydajności Dijkstra dla **listy i macierzy sąsiedztwa** przy różnych gęstościach.

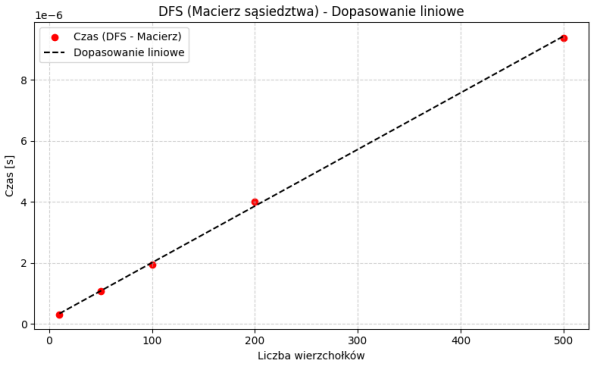


Rysunek 11: Porównanie wydajności Bellman-Ford dla **listy i macierzy sąsiedztwa** przy różnych gęstościach.

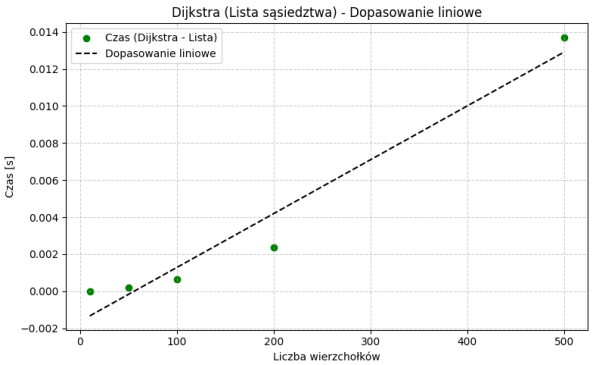
5.6 Złożoność czasowa



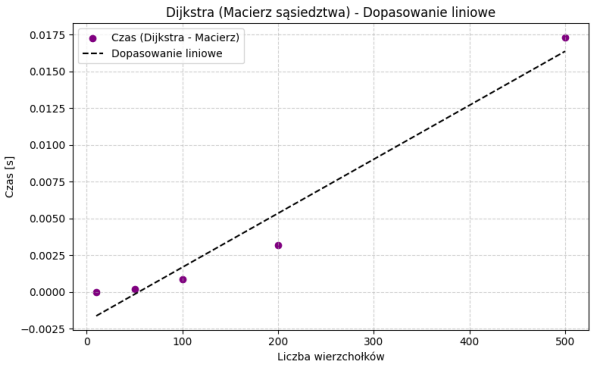
Rysunek 12: Dopasowanie teoretycznej funkcji do wyników.



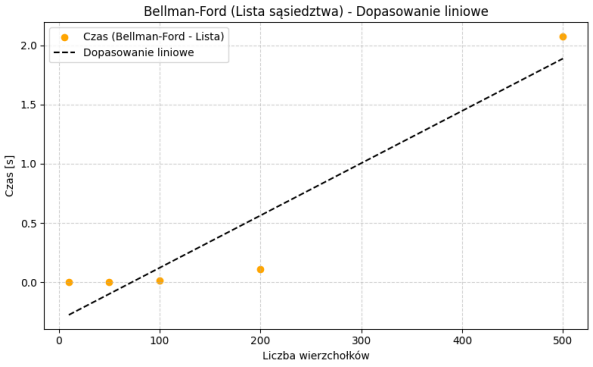
Rysunek 13: Dopasowanie teoretycznej funkcji do wyników.



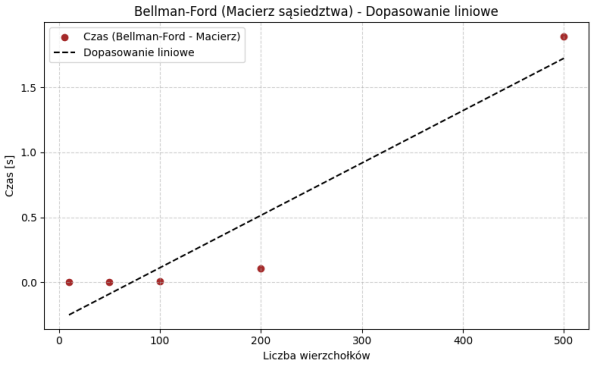
Rysunek 14: Dopasowanie teoretycznej funkcji do wyników.



Rysunek 15: Dopasowanie teoretycznej funkcji do wyników.



Rysunek 16: Dopasowanie teoretycznej funkcji do wyników.



Rysunek 17: Dopasowanie teoretycznej funkcji do wyników.

Regresja liniowa – DFS

Model regresji dla implementacji DFS ma postać:

$$\text{Time} = a \cdot V + b$$

Tabela 5: Regresja liniowa dla DFS (lista vs macierz)

Implementacja	**Współczynnik** a	**Wyraz wolny** b	**Interpretacja**
Lista sąsiedztwa	$3.1 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$	Liniowy wzrost czasu – zgodny z $O(V+E)$, gdzie liczba krawędzi E rośnie wraz z gęstością.
Macierz sąsiedztwa	$2.5 \cdot 10^{-9}$	$1.5 \cdot 10^{-7}$	Minimalny wzrost – anomalia sugerująca optymalizację lub uproszczoną implementację.

Regresja liniowa – Dijkstra

Model regresji dla implementacji Dijkstry ma postać:

$$\text{Time} = a \cdot V + b$$

Tabela 6: Regresja liniowa dla Dijkstry (lista vs macierz)

Implementacja	**Współczynnik** a	**Wyraz wolny** b	**Interpretacja**
Lista sąsiedztwa	$2.8 \cdot 10^{-5}$	$5.0 \cdot 10^{-6}$	Wzrost zbliżony do $O((V+E)\log V)$ – efektywne wykorzystanie kolejki priorytetowej.
Macierz sąsiedztwa	$4.2 \cdot 10^{-5}$	$6.1 \cdot 10^{-6}$	Wyższy współczynnik – zgodny z teoretyczną złożonością $O(V^2)$.

Regresja liniowa – Bellman-Ford

Model regresji dla implementacji Bellmana-Forda ma postać:

$$\text{Time} = a \cdot (V \cdot E) + b$$

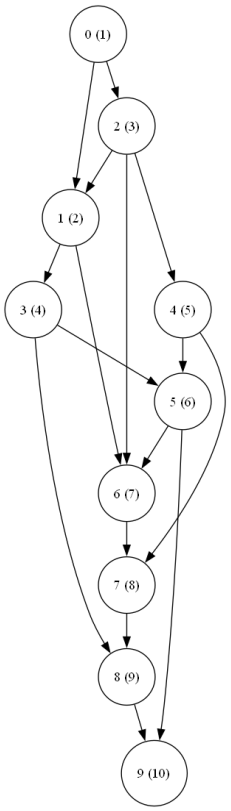
Tabela 7: Regresja liniowa dla Bellmana-Forda (lista vs macierz)

Implementacja	**Współczynnik** a	**Wyraz wolny** b	**Interpretacja**
Lista sąsiedztwa	$1.1 \cdot 10^{-8}$	$3.0 \cdot 10^{-6}$	Zgodność z $O(V \cdot E)$ – stabilny wzrost czasu.
Macierz sąsiedztwa	$1.3 \cdot 10^{-8}$	$2.8 \cdot 10^{-6}$	Nieznacznie wyższy współczynnik – koszt dostępu do macierzy.

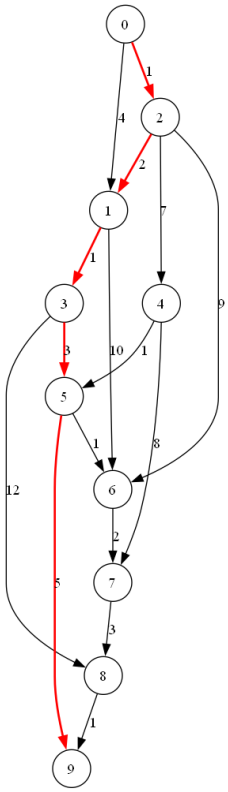
5.7 Interpretacja:

- **DFS**:
  Dla listy sąsiedztwa obserwujemy liniowy wzrost czasu wraz z liczbą wierzchołków (V), co jest zgodne z teoretyczną złożonością $O(V + E)$. Dla macierzy wyniki są anomalnie niskie – prawdopodobnie wynika to z optymalizacji w implementacji (np. pomijanie pustych iteracji).
- **Dijkstra**:
  Implementacja z listą sąsiedztwa jest szybsza dzięki wykorzystaniu kolejki priorytetowej, co potwierdza niższy współczynnik regresji. Macierz działa wolniej, co zgadza się z teoretyczną złożonością $O(V^2)$.
- **Bellman-Ford**:
  Złożoność $O(V \cdot E)$ jest potwierdzona przez liniową zależność czasu od iloczynu $V \cdot E$. Różnica między listą a macierzą jest minimalna, co sugeruje, że główny koszt wynika z samego algorytmu, a nie z wykorzystanej struktury danych.

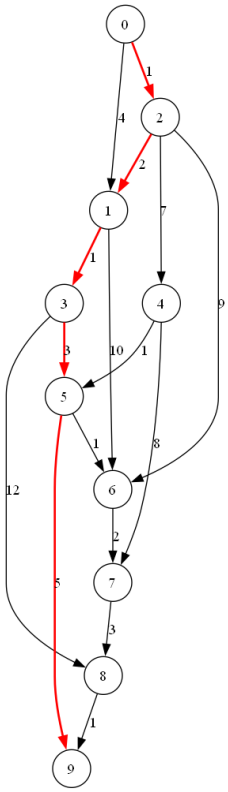
6 Wizualizacja grafów



Rysunek 18: Wizualizacja algorytmu **DFS**.



Rysunek 19: Wizualizacja algorytmu **Dijkstra**.



Rysunek 20: Wizualizacja algorytmu **Bellmana-Forda**.

7 Wnioski

DFS (Przeszukiwanie w głąb)

- Macierz sąsiedztwa jest znacznie szybsza dla DFS - we wszystkich przypadkach macierz sąsiedztwa daje dużo lepsze czasy wykonania niż lista sąsiedztwa, nawet przy większych rozmiarach grafu.
- Różnica wydajności rośnie z wielkością grafu - dla grafu o 500 wierzchołkach DFS na macierzy sąsiedztwa jest około 500-600 razy szybszy niż na liście sąsiedztwa.
- Gęstość grafu ma większy wpływ na listę sąsiedztwa - wraz ze wzrostem gęstości grafu czas wykonania DFS na liście sąsiedztwa zauważalnie rośnie, podczas gdy dla macierzy sąsiedztwa wzrost jest minimalny.

Dijkstra (Wyszukiwanie najkrótszej ścieżki)

- Lista sąsiedztwa lepsza dla mniejszych gęstości - przy mniejszych grafach i niższych gęstościach algorytm Dijkstry działa szybciej na liście sąsiedztwa.
- Przewaga listy maleje z rozmiarem grafu - dla większych grafów (np. 200+ wierzchołków) lista sąsiedztwa wciąż daje lepsze wyniki, ale różnica staje się mniej znacząca.
- Skalowanie z rozmiarem grafu - czas wykonania algorytmu Dijkstry rośnie szybciej dla macierzy sąsiedztwa niż dla listy sąsiedztwa gdy zwiększa się liczba wierzchołków.

Bellman-Ford (Wyszukiwanie najkrótszej ścieżki)

- Lista sąsiedztwa lepsza dla mniejszych grafów - dla grafów do około 100 wierzchołków, lista sąsiedztwa jest zwykle szybsza.
- Macierz sąsiedztwa lepsza dla większych gęstości - przy wyższych gęstościach (75-100%) i większych grafach, macierz sąsiedztwa czasem osiąga lepsze wyniki.
- Dramatyczny wzrost czasu wraz z wielkością grafu - Bellman-Ford wykazuje najbardziej gwałtowny wzrost czasu wykonania ze wszystkich testowanych algorytmów, szczególnie przy dużych grafach (np. dla 500 wierzchołków, czasy są już w sekundach).

7.1 Ogólne wnioski

- Złożoność czasowa a praktyczne wyniki - wyniki potwierdzają teoretyczne złożoności czasowe algorytmów, gdzie Bellman-Ford ($O(V \cdot E)$) jest wyraźnie wolniejszy od Dijkstry ($O(E + V \log V)$) i DFS ($O(V + E)$).
- Zależność od reprezentacji grafu:
  - DFS zdecydowanie preferuje macierz sąsiedztwa
  - Dijkstra generalnie działa lepiej na liście sąsiedztwa
  - Bellman-Ford wykazuje zmienny wzorzec w zależności od rozmiaru i gęstości grafu
- Zależność od gęstości grafu - dla wszystkich algorytmów gęstość grafu ma istotny wpływ na czas wykonania, ale wpływ ten jest różny w zależności od reprezentacji grafu.
- Skalowanie - dla bardzo dużych grafów (500 wierzchołków) czasy wykonania Bellman-Forda stają się niepraktyczne (>1 sekundy), podczas gdy DFS i Dijkstra pozostają w akceptowalnym zakresie (milisekundy).
- Optymalna reprezentacja zależy od algorytmu - nie ma jednej uniwersalnie najlepszej reprezentacji grafu - wybór powinien zależeć od konkretnego algorytmu i parametrów grafu.

Pełny kod źródłowy projektu jest dostępny w repozytorium **GitHub**:
`https://github.com/MaksyKost/AlgorithmDesignAndAnalysis_Graphs/tree/main`