Politechnika Wrocławska

Wydział Informatyki
i Telekomunikacji


STRUKTURY DANYCH

# Projekt nr 4

*Wykonali:*
Filip SANOWSKI - 280062
Kacper KRUSZELNICKI - 280150

*Prowadzący:*
mgr. inż Piotr NOWAK

3 czerwca 2025

# Spis treści

link do repozytorium GitHub z pełną implementacją

# 1 Cel projektu

Celem projektu jest porównanie wydajności dwóch klasycznych algorytmów wyznaczania najkrótszych ścieżek w grafie: algorytmu Dijkstry oraz algorytmu Bellmana-Forda. Eksperymenty przeprowadziliśmy dla trzech reprezentacji grafu: macierzy sąsiedztwa, listy sąsiedztwa oraz listy krawędzi. Dla każdej konfiguracji wykonaliśmy pomiary czasu działania, a następnie uśredniliśmy je w celu otrzymania porównywalnych wyników.

# 2 Zastosowane algorytmy

## 2.1 Algorytm Dijkstry

Algorytm Dijkstry służy do wyznaczania najkrótszych ścieżek z pojedynczego źródła w grafie skierowanym lub nieskierowanym, pod warunkiem że wszystkie wagi krawędzi są nieujemne. W każdej iteracji wybierany jest wierzchołek o najmniejszym znanym dystansie, a następnie przeprowadzana jest relaksacja krawędzi wychodzących z tego wierzchołka.

## 2.2 Algorytm Bellmana-Forda

Bellman-Ford wyznacza najkrótsze ścieżki z pojedynczego źródła w grafie skierowanym, również w przypadku krawędzi o ujemnych wagach (ale bez cykli ujemnych). Algorytm wykonuje $V - 1$ iteracji, w każdej przeglądając wszystkie krawędzie i dokonując relaksacji.

# 3 Zastosowane struktury danych

## 3.1 Macierz sąsiedztwa

Macierz sąsiedztwa to reprezentacja grafu w postaci dwuwymiarowej tablicy o rozmiarze $V \times V$, gdzie $V$ to liczba wierzchołków. Każda komórka $(i, j)$ zawiera wagę krawędzi skierowanej z wierzchołka $i$ do $j$. Brak krawędzi oznaczany jest wartością specjalną, np. nieskończonością.

**Zalety:**

- Szybki dostęp do wagi krawędzi $(i, j)$: $O(1)$.
- Prosta i intuicyjna implementacja.

**Wady:**

- Wymaga dużo pamięci: $O(V^2)$ niezależnie od liczby krawędzi.
- Nieefektywna dla rzadkich grafów.

**Złożoność funkcji:**

- `Dijkstra(start)` – $O(V^2)$
- `BellmanFord(start)` – $O(V^3)$

## 3.2 Lista sąsiedztwa

Lista sąsiedztwa to reprezentacja, w której każdy wierzchołek ma przypisaną listę par (sąsiad, waga). Reprezentacja ta jest oszczędna pamięciowo i wydajna w przetwarzaniu grafów rzadkich.

**Zalety:**

- Zajmuje tylko $O(V + E)$ pamięci.
- Pozwala na szybki dostęp do sąsiadów wierzchołka.

**Wady:**

- Sprawdzenie istnienia krawędzi $(i, j)$ może wymagać przeszukania listy.
- Trudniejsza implementacja w porównaniu do macierzy.

**Złożoność funkcji:**

- `Dijkstra(start)` – $O((V + E) \log V)$
- `BellmanFord(start)` – $O(VE)$

### 3.3 Lista krawędzi

Graf reprezentowany jako lista struktur (lub trójek) $(u, v, w)$ opisujących krawędzie. Struktura bardzo prosta i elastyczna – wystarczy jedna tablica. W praktyce szczególnie efektywna przy implementacji Bellmana-Forda.

**Zalety:**

- Bardzo kompaktowa – wymaga $O(E)$ pamięci.
- Idealna dla algorytmów przeglądających wszystkie krawędzie (np. Bellman-Ford).

**Wady:**

- Trudny dostęp do sąsiadów konkretnego wierzchołka.

**Złożoność funkcji:**

- `Dijkstra(start)` – $O(VE)$
- `BellmanFord(start)` – $O(VE)$

## 4 Badania

Testy wydajności algorytmów Dijkstry i Bellmana-Forda zostały przeprowadzone na komputerze o poniższej specyfikacji sprzętowej:

- **Procesor:** Intel(R) Core(TM) i5-14600K
- **Liczba rdzeni fizycznych:** 14
- **Liczba wątków logicznych:** 20
- **Pamięć RAM:** 16 GB
- **System operacyjny:** Microsoft Windows 10 Home 64-bit
- **Kompilator:** Microsoft Visual C++ (MSVC)

## 4.1 Opis implementacji

Do przeprowadzenia badań przygotowano własny program testowy w języku C++, który umożliwia generowanie losowych grafów o zadanej liczbie wierzchołków i określonej gęstości. Program obsługuje trzy różne reprezentacje grafów:

- macierz sąsiedztwa,
- lista sąsiedztwa,
- lista krawędzi.

Dla każdej struktury zaimplementowano oba analizowane algorytmy. Wydajność mierzono z wykorzystaniem biblioteki `<chrono>` z dokładnością do nanosekund.

## 4.2 Sposób pomiaru

Testy przeprowadzano według następującej procedury:

1. Dla zadanych wartości liczby wierzchołków $V$ oraz gęstości grafu $G$ generowano losową instancję grafu.
2. Dla każdej struktury grafu uruchamiano oba algorytmy (Dijkstra, Bellman-Ford).
3. Pomiar czasu wykonania każdej operacji powtarzano 100 razy, a następnie obliczano średnią wartość.
4. Wyniki zapisywano w formacie CSV w postaci: `Struktura;Algorytm;V;Gestosc;SredniCzas_ns`.

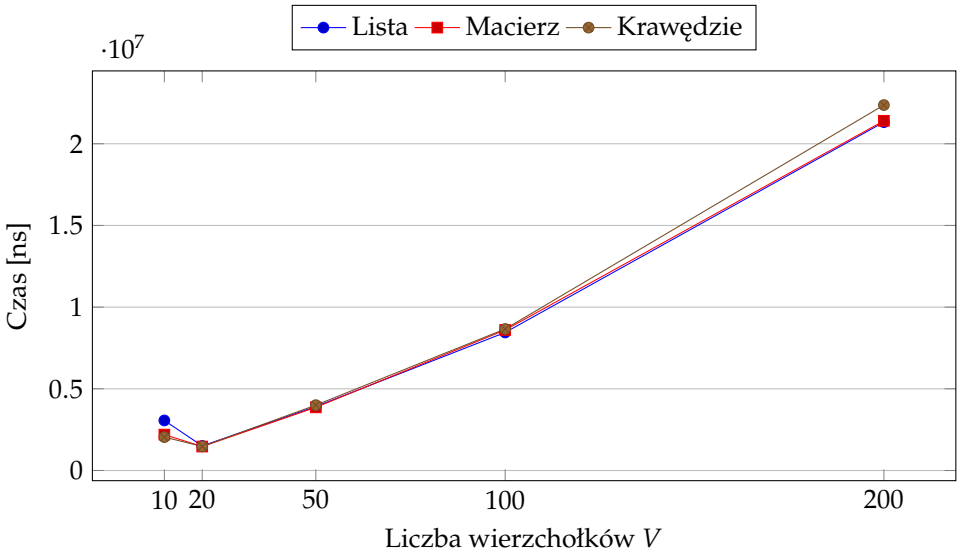
## 4.3 Zakres danych wejściowych

Badania przeprowadzono dla różnych konfiguracji:

- **Rozmiary grafów (liczba wierzchołków):** 10, 20, 50, 100, 200, 400
- **Gęstości grafów:** 25%, 50%, 75%, 100%
- **Liczba powtórzeń dla uśrednienia:** 100

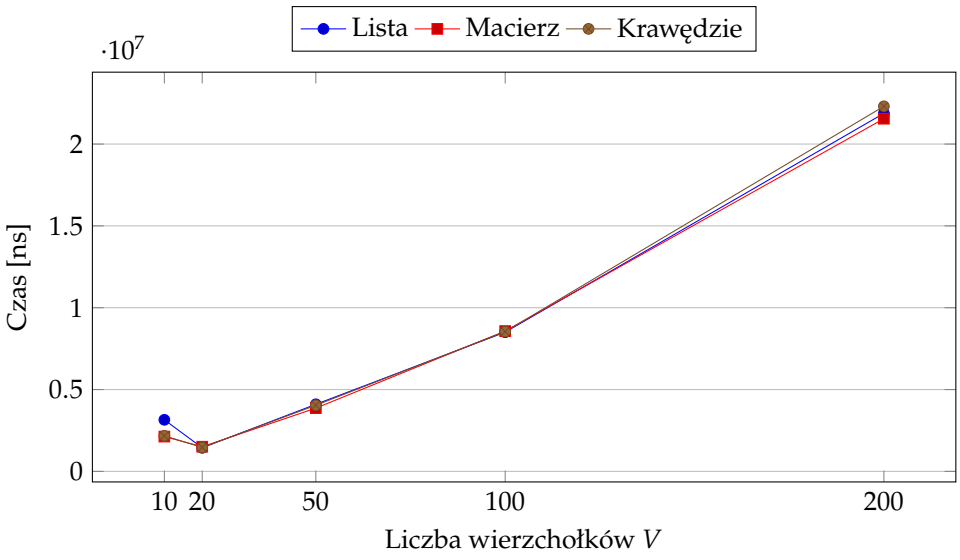
Uzyskane dane zapisano w pliku `wyniki.csv`, a następnie wykorzystano je do wygenerowania wykresów porównujących czasy wykonania w zależności od liczby wierzchołków i gęstości grafu.

# 5 Wyniki eksperymentów



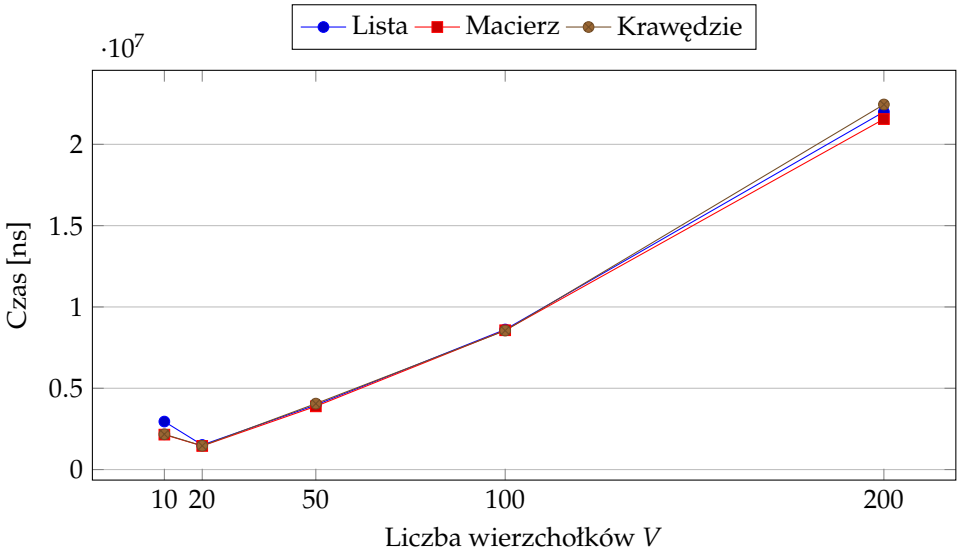

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 3063610 | 2196285 | 2042974 |
| 20 | 1503205 | 1467274 | 1473093 |
| 50 | 3914684 | 3865861 | 4000104 |
| 100 | 8449380 | 8602084 | 8662459 |
| 200 | 21323337 | 21405797 | 22369174 |

Rysunek 1: Porównanie czasu działania algorytmu Dijkstry przy gęstości 25%



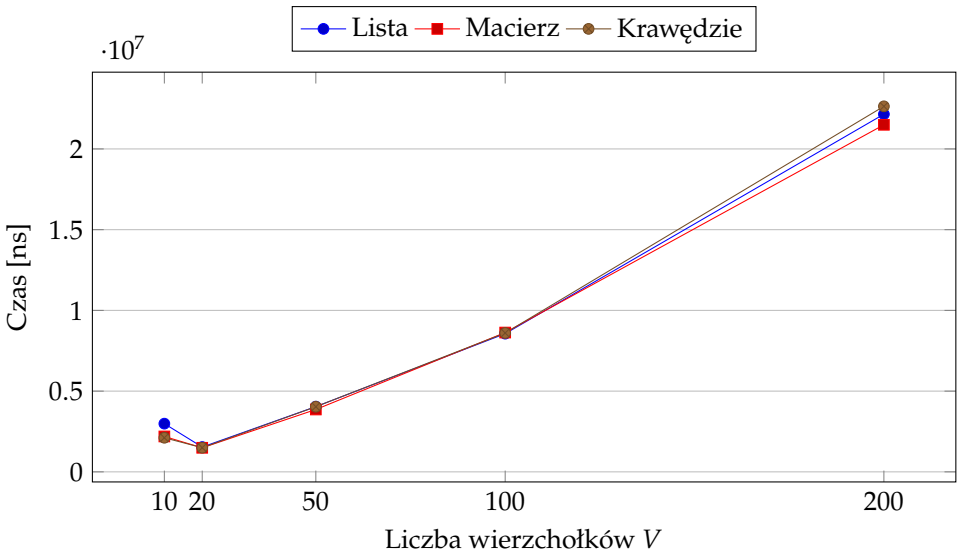

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 3151314 | 2126596 | 2173066 |
| 20 | 1441403 | 1506715 | 1463830 |
| 50 | 4097791 | 3863931 | 4040193 |
| 100 | 8506438 | 8573146 | 8539380 |
| 200 | 21875298 | 21549451 | 22305953 |

Rysunek 2: Porównanie czasu działania algorytmu Dijkstry przy gęstości 50%



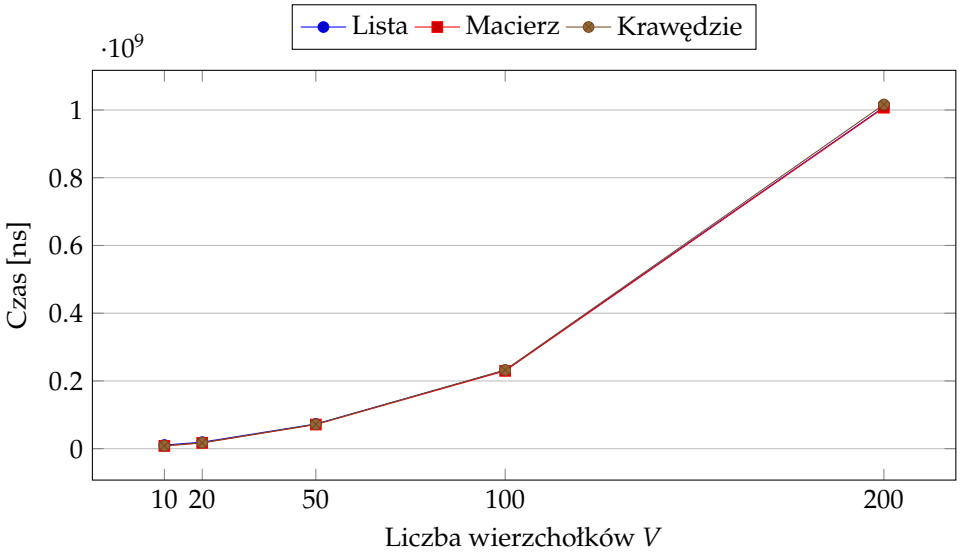

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 2951950 | 2148857 | 2171319 |
| 20 | 1512901 | 1453219 | 1461422 |
| 50 | 3971673 | 3888094 | 4056323 |
| 100 | 8612997 | 8567515 | 8545824 |
| 200 | 21997923 | 21554314 | 22450934 |

Rysunek 3: Porównanie czasu działania algorytmu Dijkstry przy gęstości 75%



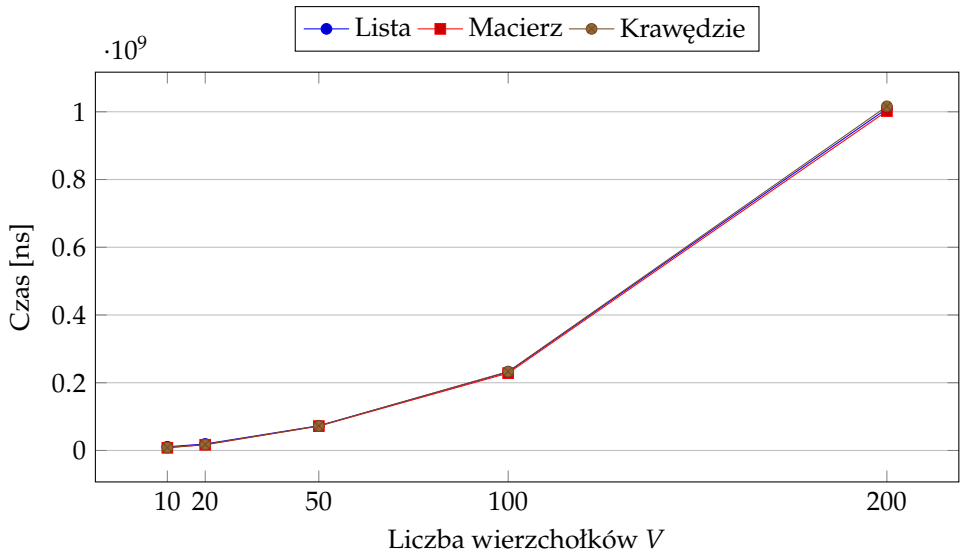

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 2985124 | 2192023 | 2111777 |
| 20 | 1536284 | 1490642 | 1495376 |
| 50 | 4038802 | 3862731 | 4034293 |
| 100 | 8559523 | 8627370 | 8599835 |
| 200 | 22152394 | 21495551 | 22639166 |

Rysunek 4: Porównanie czasu działania algorytmu Dijkstry przy gęstości 100%



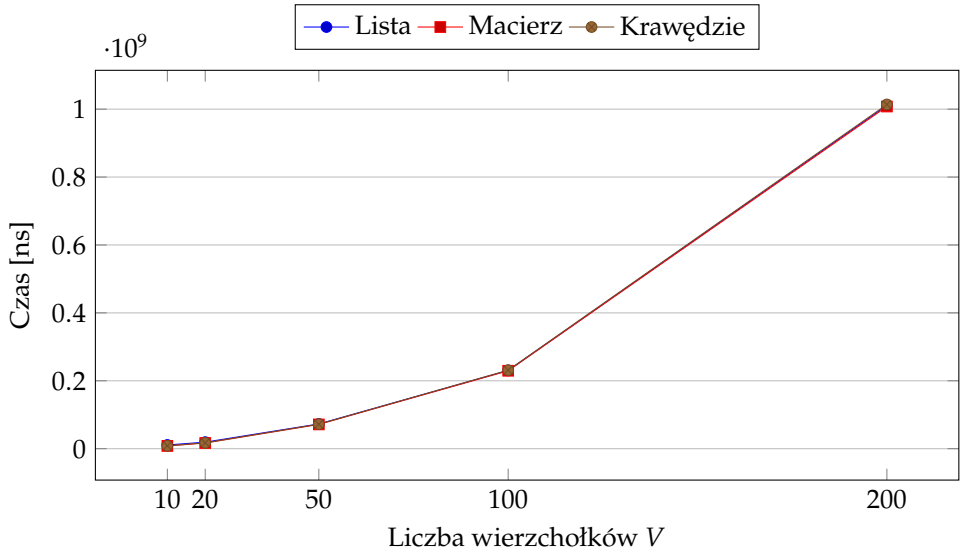

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 10722526 | 8109619 | 8490170 |
| 20 | 19292394 | 17013353 | 17451136 |
| 50 | 72936148 | 71268360 | 72092339 |
| 100 | 231463982 | 229769796 | 232378512 |
| 200 | 1008393006 | 1006742773 | 1016263430 |

Rysunek 5: Porównanie czasu działania algorytmu Bellman-Ford przy gęstości 25%



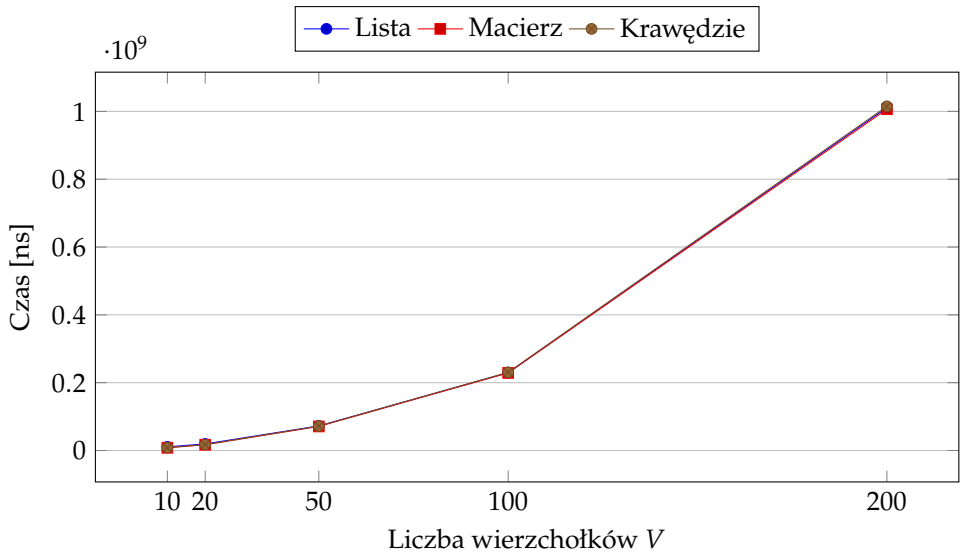

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 10872578 | 7942300 | 8479339 |
| 20 | 19278500 | 16831339 | 17601791 |
| 50 | 73100754 | 71928687 | 72598544 |
| 100 | 231844582 | 228392690 | 233021268 |
| 200 | 1009129206 | 1002109840 | 1016107810 |

Rysunek 6: Porównanie czasu działania algorytmu Bellman-Ford przy gęstości 50%



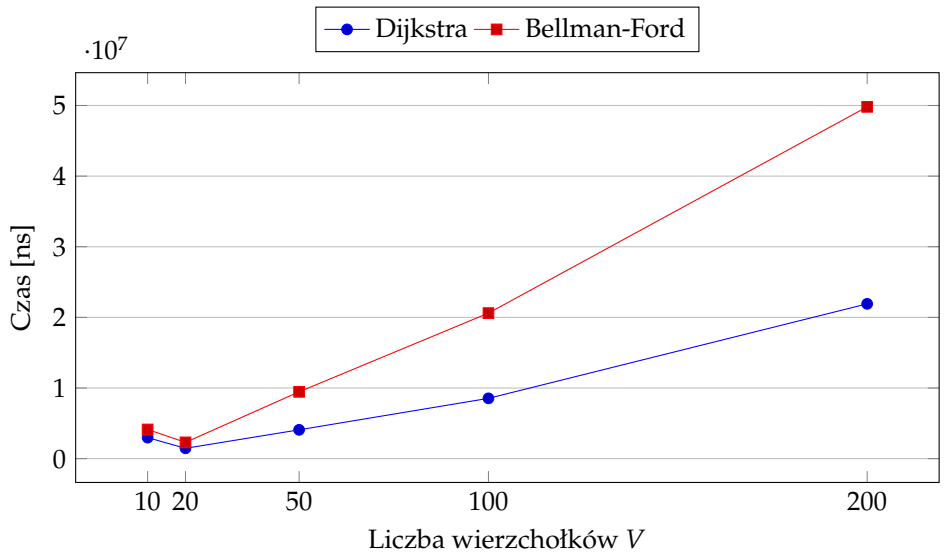

| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 10948629 | 8130019 | 8380505 |
| 20 | 19324356 | 16721990 | 17295841 |
| 50 | 72891385 | 71498191 | 72184587 |
| 100 | 230352374 | 229556409 | 231294918 |
| 200 | 1010824534 | 1007740240 | 1013771943 |

Rysunek 7: Porównanie czasu działania algorytmu Bellman-Ford przy gęstości 75%




| Liczba wierzchołków | Lista [ns] | Macierz [ns] | Krawędzie [ns] |
|---|---|---|---|
| 10 | 10730708 | 7978204 | 8341337 |
| 20 | 19424413 | 16944601 | 17291882 |
| 50 | 72478752 | 70916606 | 71960357 |
| 100 | 229370275 | 228768087 | 230578875 |
| 200 | 1011011580 | 1006499822 | 1014953536 |

Rysunek 8: Porównanie czasu działania algorytmu Bellman-Ford przy gęstości 100%




Rysunek 9: Porównanie czasów działania algorytmów dla listy sąsiedztwa przy gęstości 100%

# 6 Wnioski

- Algorytm Dijkstry okazał się znacznie szybszy od algorytmu Bellmana-Forda, szczególnie dla grafów o dużej liczbie wierzchołków i niskiej gęstości. Wykresy dla gęstości 100% wyraźnie pokazują przewagę Dijkstry w każdej strukturze.
- Najlepsze rezultaty czasowe dla algorytmu Dijkstry uzyskano przy wykorzystaniu listy sąsiedztwa, co wynika z jej efektywności pamięciowej oraz możliwości szybkiego dostępu do sąsiadów w połączeniu z kolejką priorytetową.

- W przypadku algorytmu Bellmana-Forda najefektywniejszą strukturą danych okazała się lista krawędzi. Dzięki temu, że algorytm i tak przechodzi liniowo po wszystkich krawędziach, ta struktura nie wprowadza dodatkowego narzutu.
- Macierz sąsiedztwa, mimo prostej implementacji, nie jest efektywna przy dużych i rzadkich grafach. Jej czas działania szybko rośnie wraz z liczbą wierzchołków.
- Dla grafów o dużej gęstości (75%–100%) różnice między strukturami stają się mniej istotne – wszystkie struktury uzyskują zbliżone wyniki czasowe.

# 7 Źródła


- GeeksforGeeks. *Graph Algorithms*. Pozyskano z: `https://www.geeksforgeeks.org/graph-data-structure-and-a`
- Wikipedia. (2024). *Dijkstra's algorithm*. Pozyskano z: `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`
- Wykłady profesora Jarosława Rudego