Politechnika Wrocławska

Badanie efektywności wybranych algorytmów grafowych

Jakub Kowalski, 281033

Projekt realizowany w ramach kursu:
Algorytmy i Złożoność Obliczeniowa

Data oddania projektu:
21.06.2024

Spis treści

1.Cel projektu

Celem projektu było zapoznanie się z wybranymi problemami grafowymi, implementacja algorytmów do ich rozwiązywania oraz przeprowadzenie badań mających na celu analizę wydajności zastosowanych rozwiązań.

2.Implementacja

W celu przeprowadzenia badań zaimplementowano program w języku C++.

2.1.Tryby programu

Program umożliwia uruchomienie dwóch trybów: pojedynczego testu oraz badań.

2.1.1.Tryb pojedynczego testu

Umożliwia wczytanie grafu z pliku tekstowego, a następnie wykonanie wybranego algorytmu. Po jego zakończeniu możliwy jest podgląd wczytanego grafu we wszystkich dostępnych reprezentacjach*(patrz 3.3),* a także wyników działania algorytmu. Rezultaty zapisywane są do pliku wyjściowego.

Pliki wejściowe przyjmują następujący format:

- Pierwsza linijka: liczba wierzchołków i liczba krawędzi
- Kolejne linijki: wierzchołek początkowy, wierzchołek końcowy, waga krawędzi

2.1.2.Tryb badań

Umożliwia wygenerowanie grafu, zgodnie z zadanymi parametrami, a następnie wykonanie wybranego algorytmu. Przeprowadzany jest pomiar czasu wykonania każdego algorytmu, a wyniki są zapisywanie do pliku wyjściowego.

2.2.Pomiar czasu

Każdemu wywołaniu algorytmu będzie towarzyszyć pomiar czasu wykonania w mikrosekundach. Ta jednostka czasu zapewnia możliwość bardziej dokładnych pomiarów, co jest szczególnie przydatne, gdy algorytm potrzebuje niewiele czasu, aby zakończyć swoje działanie i zwrócić wynik.

Skonstruowana jest klasa **Timer**, która notuje czas tuż przed startem algorytmu i od razu po jego zakończeniu, a potem oblicza różnicę między nimi – która stanowi czas wykonania algorytmu.

2.3.Klasy przetrzymujące reprezentacje grafów

Zaprojektowano interfejs **Graph,** który jest abstrakcyjną klasą bazową, definiującą wspólne operacje dla wszystkich reprezentacji grafów. Konkretne reprezentacje są zawarte w klasach implementujących ten interfejs:

- **GraphAdjacency –** macierz sąsiedztwa
- **GraphIncidence –** macierz incydencji
- **GraphList –** lista sąsiedztwa

2.4.Klasy narzędziowe

W programie zaimplementowano klasy narzędziowe, realizujące operacje wspierające:

- **FileHandler –** obsługuje odczyt danych z plików wejściowych oraz zapis wyników do plików wyjściowych
- **RandomHandler –** odpowiada za generowanie losowych grafów na potrzeby badań

2.5.Klasy pomocnicze

Zaimplementowano klasy pomocnicze, które są używane jako wsparcie działania algorytmów

- **Edge –** reprezentuje pojedynczą krawędź grafu
- **EdgeList –** przechowuje zbiór krawędzi, np. wszystkich, albo mających wspólne cechy takie jak wspólny wierzchołek początkowy
- **Disjoint –** reprezentuje strukturę zbiorów rozłącznych, która ma za zadanie efektywnie zarządzać grupami elementów
- **Heap –** reprezentuje strukturę kopca binarnego, w programie jest używany jako kolejka priorytetowa

3.Graf

Grafy to obiekty, które służą do modelowania zbiorów elementów, które parami mają między sobą relacje. Są to zbiory wierzchołków oraz krawędzi łączących dwa wierzchołki [1].

$$G = (V, E)$$

V – zbiór wierzchołków

E – zbiór krawędzi

3.1.Wierzchołki

Wierzchołek to podstawowy element grafu. W dalszej części, w grafie, który zawiera *V* wierzchołków, będziemy je oznaczać kolejnymi liczbami naturalnymi, począwszy od *0,1,2*... aż do *V-1*.

3.2.Krawędzie

Krawędź to połączona para wierzchołków, która reprezentuje relację między nimi. Krawędzie mogą posiadać **wagę –** czyli wartość liczbową, która oznacza koszt jej przejścia. Graf, w którym krawędzie mają wagi jest **grafem ważonym –** takie grafy są tematem rozważań w tym projekcie.

Krawędzie mogą być **skierowane** lub **nieskierowane** – w zależności od tego grafy można podzielić na dwie grupy.

3.2.1.Krawędzie nieskierowane – grafy nieskierowane

Krawędź nieskierowana łączy dwa wierzchołki i prowadzi w obie strony – można ją przejść w obu kierunkach. Graf, w którym krawędzie są nieskierowane, jest grafem nieskierowanym.

3.2.2.Krawędzie skierowane – grafy skierowane

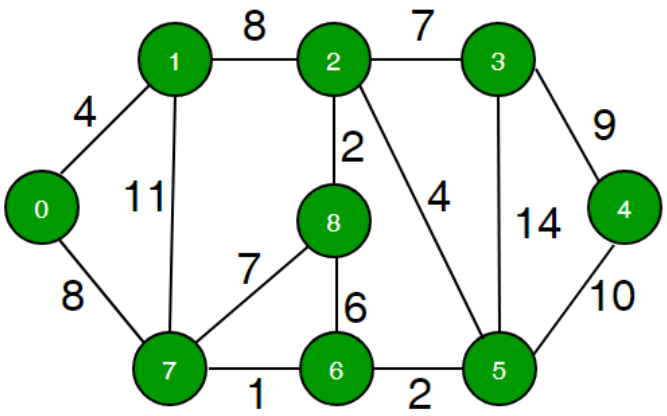
Krawędź skierowana ma przypisany kierunek. W krawędzi (u,v) wierzchołkiem początkowym jest *u,* a końcowym *v.* Można ją przechodzić tylko w takim kierunku – nie w przeciwnym. Graf, w którym krawędzie są skierowane, jest grafem skierowanym.

3.3.Reprezentacja

Istnieją różne formy przedstawiania grafu, czyli wykazywania w skonsolidowanej formie jego wierzchołków oraz krawędzi, wraz ze wszystkimi informacjami takimi jak kierunek krawędzi czy wagi. [3]

3.3.1.Graficzna

Graficzna reprezentacja grafu (rys.1) jest wizualną formą przedstawienia grafu. Wierzchołki są rysowane jako obiekty(punkty, okręgi), a krawędzie to linie, które łączą te obiekty.



Rysunek 1: przykład reprezentacji graficznej grafu (źródło: Geeks for Geeks)[2]

Jest to najbardziej intuicyjna reprezentacja dla człowieka, ale za to nieczytelna dla komputera i niepraktyczna do zastosowania w informatyce. Nie będzie uwzględniana w dalszych rozwiązaniach.

3.3.2.Macierz Sąsiedztwa

Reprezentacja grafu jako kwadratowa macierz sąsiedztwa M o wymiarach *VxV (rys.2)*. Krawędź (u,v) jest przedstawiona w macierzy w miejscu M[u][v], w którym wpisana jest wartość wagi *w* tej krawędzi. Jeżeli krawędź (u,v) nie istnieje, to w miejsce M[u][v] umownie będzie wpisane 0.

$$M[u][v] = w$$

W grafie nieskierowanym waga będzie wpisana również w miejscu M[v][u].

$$M[u][v] = M[v][u]$$

```
Macierz Sasiedztwa
0       3       0       0       3       6
3       0       1       3       0       0
0       1       0       3       0       1
0       3       3       0       0       1
3       0       0       0       0       2
6       0       1       1       2       0
```

Rysunek 2: przykład reprezentacji grafu jako macierz sąsiedztwa

3.3.3.Macierz incydencji

Reprezentacja grafu jako macierz incydencji *M* o wymiarach *VxE (rys.3)*. Krawędź (u,v) jest przedstawiona jako kolumna *e* w macierzy. Wartość wagi będzie wpisana w miejscach M[u][e] oraz M[v][e]. Pozostałe miejsca w kolumnie *e* będą wypełnione 0.

$$M[u][e] = w\,, M[v][e] = w$$

W grafie skierowanym wartość wpisana w miejsce M[v][e] będzie liczbą przeciwną wagi, co będzie oznaczać koniec krawędzi. Wartość dodatnia M[u][e] będzie oznaczać początek tej krawędzi.

$$M[v][e] = -w$$

```
Macierz Incydencji
0       3       3       0       0       0       0       -6      0
0       -3      0       1       0       0       0       0       -3
0       0       0       -1      3       1       0       0       0
-1      0       0       0       -3      0       0       0       3
0       0       -3      0       0       0       2       0       0
1       0       0       0       0       -1      -2      6       0
```

Rysunek 3: przykład reprezentacji grafu jako macierz incydencji

3.3.4.Lista sąsiedztwa

Reprezentacja grafu jako tablica list (rys.4), w której każdemu wierzchołkowi *n* jest przypisana lista jego sąsiadów, czyli wierzchołków, które są końcami krawędzi wychodzących z *n*.

Dla krawędzi (u,v), wierzchołek *v* jest na liście sąsiadów wierzchołka *u*. Jeżeli graf jest nieskierowany, to wówczas wierzchołek *u* będzie uwzględniony na liście sąsiadów *v*. Jeżeli krawędź nie istnieje – to nie ma jej wspomnianej na żadnej z list.

```
0: 5:6  4:3     1:3
1: 3:3  2:1     0:3
2: 5:1  3:3     1:1
3: 1:3  2:3     5:1
4: 5:2  0:3
5: 0:6  4:2     2:1     3:1
```

Rysunek 4: przykład reprezentacji grafu jako lista sąsiedztwa

4.Problemy i algorytmy grafowe

W grafach pojawiają się problemy optymalizacyjne, a istotnym zagadnieniem są sposoby ich rozwiązywania. W tej sekcji przedstawione zostaną algorytmy rozwiązujące wybrane problemy tj. wyznaczanie minimalnego drzewa rozpinającego MST oraz wyznaczanie najkrótszej ścieżki w grafie.

4.1.Wyznaczanie minimalnego drzewa rozpinającego grafu (MST)

Minimalne drzewo rozpinające (MST) – to drzewo rozpinające, które łączy wszystkie wierzchołki grafu za pomocą dokładnie *V-1* krawędzi, nie zawiera cykli, a suma wybranych

krawędzi jest minimalna, czyli nie da się znaleźć takiego zestawu krawędzi, które utworzą MST i będą miały jeszcze mniejszą sumę wag.

Problem wyznaczania minimalnego drzewa rozpinającego jest realizowany dla grafów nieskierowanych.

4.1.1.Algorytm Prima

Algorytm Prima *[4]* wyznacza minimalne drzewo rozpinające zadanego grafu. Działa w sposób zachłanny, dodając stopniowo najlepiej rokujące (najtańsze) krawędzie.

Dla każdego wierzchołka jest przypisywana wartość klucza. Wybrany wierzchołek startowy otrzymuje wartość $key[r] = 0$, a pozostałe elementy $key[v] = INT\_MAX$. Każdy z wierzchołków pamięta też swojego „najtańszego" sąsiada, początkowo żaden element go nie ma, wszyscy mają to pole ustawione na $par[v] = -1$. Końcowo, w polu $par[v]$ będzie numer wierzchołka, z którym dany wierzchołek tworzy krawędź wchodzącą w skład MST, a w $key[r]$ wartość wagi tej krawędzi.

Tworzona jest kolejka priorytetowa, w której umieszczane są wszystkie wierzchołki. Jako kolejka priorytetowa używany jest kopiec binarny [5] – ta reprezentacja zapewnia szybki dostęp do elementu o najmniejszej wartości klucza, a pozyskiwanie takiego elementu jest kluczowe w tym algorytmie.

W kolejnych iteracjach szukamy krawędzi pomiędzy wierzchołkami, spośród których jeden jest już w MST, a drugi jeszcze nie. Po dodaniu krawędzi ten drugi wierzchołek jest dodawany do MST. Proces jest powtarzany tak długo, aż wszystkie wierzchołki nie zostaną usunięte z kolejki priorytetowej.

Za pomocą operacji *Extract-Min* pozyskujemy i usuwamy element *n* o najmniejszej wartości klucza z kolejki priorytetowej. Do MST dodajemy krawędź, której jednym wierzchołkiem jest pozyskany wierzchołek *n*, a drugim wierzchołek $par[n]$. Waga tej krawędzi wynosi $key[n]$. Krawędź nie jest dodawana przy pierwszym przejściu algorytmu – wówczas dodawany jest pierwszy wierzchołek, który nie ma jeszcze z kim stworzyć krawędzi.

Następnie dla rozważanego wierzchołka *n* sprawdzamy wszystkie wychodzące z niego krawędzie. Jeżeli krawędź *e* nie tworzy cyklu, czyli drugi wierzchołek *t* tej krawędzi nie został jeszcze dodany do MST, to jeżeli waga *w* tej krawędzi jest mniejsza niż wartość w kluczu drugiego wierzchołka $key[t]$, to wówczas aktualizowane są pola dla wierzchołka *t:* $key[t] = w \ \ oraz \ \ par[t] = n$. Oznacza to, że najlepszym sposobem, w jaki można aktualnie dołączyć wierzchołek *t* do MST, jest dodanie krawędzi (par[t],t) o wadze *key[t].*

Złożoność obliczeniowa algorytmu Prima, stosującego listę sąsiedztwa oraz kopiec binarny wynosi **O(E*lg(V))**.

4.1.2.Algorytm Kruskala

Algorytm Kruskala *[6]* wykorzystuje strukturę zbiorów rozłącznych [7]. Na początku każdy wierzchołek grafu jest osobnym zbiorem (drzewem). Jest to efektywny sposób wykrywania cykli. Jeżeli podjęto by próbę dodania krawędzi między dwoma wierzchołkami, które są w tym samym zbiorze, otrzymalibyśmy cykl w grafie, które w MST wystąpić nie mogą.

Tworzona jest tablica wszystkich krawędzi grafu, a następnie jest ona sortowana niemalejąco według wag krawędzi. Do sortowania jest użyty algorytm sortowania szybkiego – Quick Sort. Jest to algorytm o optymalnej złożoności obliczeniowej, więc jest wydajną opcją.

W kolejnych iteracjach są rozważane kolejne krawędzie z posortowanej tablicy. Jeżeli oba wierzchołki połączone tą krawędzią są w innych drzewach, to oznacza, że można dodać tę krawędź do MST, ponieważ jej dodanie nie spowoduje powstania cyklu. Następnie drzewa w których znajdowały się oba wierzchołki są scalane w jedno drzewo.

Do MST musi być dodane dokładnie *V-1* krawędzi, więc po dodaniu takiej liczby krawędzi algorytm może się zakończyć.

Złożoność obliczeniowa algorytmu Kruskala, stosującego strukturę zbiorów rozłącznych wynosi **O(E*lg(E)).**

4.2.Wyznaczanie najkrótszej ścieżki w grafie

Najkrótsza ścieżka w grafie z danego wierzchołka źródłowego *s* do innego wierzchołka docelowego *t* to droga po krawędziach w grafie, która zaczyna się w *s,* a kończy w *t*, a wartość wag krawędzi składających się na nią jest najmniejsza.

Problem wyznaczania najkrótszej ścieżki w grafie realizowany jest dla grafów skierowanych.

4.2.1.Algorytm Dijkstry

Na początku algorytmu Dijkstry *[8]* każdemu wierzchołkowi przypisywana jest wartość klucza, która odpowiada aktualnemu dystansowi do osiągnięcia tego wierzchołka. Przy inicjacji wartość dla wierzchołka startowego wynosi $dist[s] = 0$, a dla pozostałych $dist[n] = INT\_MAX$. Przypisywany jest też poprzednik, czyli wierzchołek poprzedzający dany wierzchołek na najkrótszej ścieżce prowadzący do niego, przy inicjacji wszystkie wierzchołki nie mają takiego poprzednika $prev[n] = -1$.

Tworzona jest kolejka priorytetowa, a w niej umieszczane są wszystkie wierzchołki.

Następnie w kolejnych iteracjach, usuwamy z kolejki priorytetowej element o najmniejszej wartości klucza za pomocą *Extract-Min*. Robimy to tak długo, dopóki wszystkie elementy nie zostaną usunięte z kolejki.

Sprawdzamy wszystkie krawędzie wychodzące z tego usuwanego wierzchołka. Dla każdej z nich dokonujemy **relaksacji –** jeżeli suma wagi *w* tej krawędzi (u,v) oraz wartości klucza wierzchołka *u* $key[u]$ jest mniejsza, niż wartość $dist[v]$ to znaczy, że znaleźliśmy bardziej optymalną drogę prowadzącą do wierzchołka *v*. Wówczas aktualizujemy $dist[v] = dist[u] + w$ oraz $prev[v] = u$. Zmniejszyliśmy dystans potrzebny do osiągnięcia wierzchołka, oraz jego poprzednika na tej ścieżce.

Algorytm Dijsktry działa tylko i wyłącznie wtedy, gdy wartości wag krawędzi są liczbami nieujemnymi. Nie nadaje się do stosowania, jeżeli w grafie występują krawędzie o wagach ujemnych.

Złożoność obliczeniowa algorytmu Dijkstry, używającego kopca binarnego i listę sąsiedztwa wynosi **O(E*lg(V)).**

4.2.2.Algorytm Bellmana-Forda

W algorytmie Bellmana-Forda *[9]* tworzone są tablice, bardzo podobnie jak w algorytmie Dijsktry. Pierwsza to wartości kluczy, które oznaczają wartość aktualnie najkrótszej znanej drogi do danego wierzchołka, początkowo $dist[n] = INT\_MAX$ dla wszystkich wierzchołków poza startowym, dla którego $dist[s] = 0$.

W drugiej są poprzednicy, przy inicjacji $prev[n] = -1$ dla każdego wierzchołka.

Po wstępnej inicjacji tworzona jest lista wszystkich krawędzi grafu.

Następnie *V-1* razy wykonywana jest pętla, w której każda krawędź jest **relaksowana.** Działa to na dokładnie tych samych zasadach, jak w algorytmie Dijkstry. Jeśli dystans dojścia do danego wierzchołka może być zmniejszony, to aktualizujemy jego pola *dist* oraz *prev*.

Po wykonaniu wszystkich iteracji, wykonywana jest jeszcze jedna, dodatkowa, która umożliwia wykrycie cyklu ujemnego. Jeżeli doszłoby do sytuacji, że dystans dojścia do wierzchołka można byłoby zmniejszyć, to wtedy występuje cykl ujemny – niemożliwe jest wtedy znalezienie najkrótszej ścieżki.

Algorytm Bellmana-Forda jest w stanie znaleźć najkrótszą ścieżkę w grafie, w którym wagi przyjmują wartości ujemne. To jego duża zaleta względem algorytmu Dijkstry.

Złożoność obliczeniowa algorytmu Bellmana-Forda wynosi **O(V*E).**

5.Badanie

W celu porównania efektywności działania wybranych algorytmów grafowych przeprowadzono badanie polegające na pomiarze czasu ich wykonania na grafach o określonym rozmiarze i gęstości. W eksperymencie uwzględniono zarówno grafy nieskierowane, jak i skierowane – zależnie od wymagań analizowanych algorytmów.

5.1.Graf w badaniach

Grafy w badaniach są grafami spójnymi. Każdą parę wierzchołków łączy ścieżka, czyli z każdego wierzchołka da się przejść do dowolnego innego. W grafach krawędzie się nie powtarzają. W grafie nieskierowanym może istnieć tylko jedna krawędź (u,v) łącząca dwa wierzchołki. W grafie nieskierowanym mogą istnieć dwie krawędzie, ale muszą być w przeciwnych kierunkach (u,v), (v,u). Wartości wag krawędzi w grafach rozważanych w badaniach są liczbami naturalnymi większymi od 0.

5.2.Rozmiar grafu (liczba wierzchołków)

W badaniach jest sprawdzany wpływ liczby wierzchołków na czas wykonania algorytmów. Wykorzystywane są następujące rozmiary grafów:

- 100 wierzchołków
- 250 wierzchołków
- 500 wierzchołków
- 750 wierzchołków
- 1000 wierzchołków
- 1250 wierzchołków
- 1500 wierzchołków

5.3.Gęstość grafu

Sprawdzana jest gęstość grafu, czyli stosunek liczby krawędzi w grafie do liczby krawędzi w grafie pełnym, czyli grafu, w którym każdy wierzchołek jest bezpośrednio połączony krawędzią z każdym innym wierzchołkiem.

W grafie pełnym nieskierowanym o *n* wierzchołkach liczba krawędzi wynosi $\frac{n*(n-1)}{2}$.

W grafie pełnym skierowanym o *n* wierzchołkach liczba krawędzi wynosi $n * (n - 1)$.

W badaniach wykorzystywane są następujące gęstości grafów:

- 10% / 0,1
- 25% / 0,25
- 50% / 0,5
- 75% / 0,75
- 99% / 0,99

5.4.Generowanie grafu

Grafy do badań o zadanych rozmiarach i gęstościach są generowane losowo. Uwzględnione musi być to, że wygenerowany graf musi być spójny.

Z tego powodu pierwszym krokiem jest wygenerowanie drzewa rozpinającego. Kolejno są generowane krawędzie wychodzące z losowego wierzchołka już podłączonego do drzewa, do losowego wierzchołka jeszcze nie podłączonego.

Jeżeli graf jest nieskierowany, to wygenerowanie drzewa rozpinającego jest wystarczające. W grafie skierowanym nadal są wierzchołki, z których nie wychodzi żadna krawędź (liście drzewa). Należy dodać krawędzie wychodzące z nich, a wierzchołkiem docelowym jest wierzchołek początkowy

Na tym etapie mamy już graf, który jest spójny i kolejnym krokiem jest dodanie tylu krawędzi, żeby osiągnąć zadaną gęstość. Generowane są wszystkie jeszcze nieistniejące krawędzie – są tasowane losowo algorytmem Fishera-Yatesa, i do grafu dodawane są kolejne krawędzie z listy, dopóki nie zostanie osiągnięta wymagana gęstość.

6.Wyniki

W tej sekcji prezentowane są wyniki przeprowadzonych badań wraz z opisami, uwagami, wnioskami.

6.1.Wyznaczanie minimalnego drzewa rozpinającego

Wyniki dotyczące problemu wyznaczania minimalnego drzewa rozpinającego MST, algorytmów Prima oraz Kruskala.

6.1.2.Prim

Reprez.	Lista sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,14	0,24	0,39	0,55	0,73
250	0,54	1,59	3,23	4,95	6,44
500	2,68	6,78	13,70	21,24	28,55
750	6,04	15,25	33,10	55,10	72,71
1000	10,45	28,72	66,03	101,72	137,71
1250	16,34	48,89	105,23	163,70	220,80
1500	24,19	71,91	156,72	241,45	323,39

Tabela 1: średnie czasy wykonania algorytmu Prima, graf jako lista sąsiedztwa

Reprez.	Macierz sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,24	0,34	0,51	0,71	0,77
250	0,99	1,57	2,52	3,06	3,28
500	3,40	5,75	8,48	11,22	12,61
750	7,14	11,04	17,26	24,37	28,01
1000	11,57	18,37	31,49	41,74	51,09
1250	17,40	28,33	45,10	64,01	80,70
1500	24,26	38,83	65,65	91,71	114,68

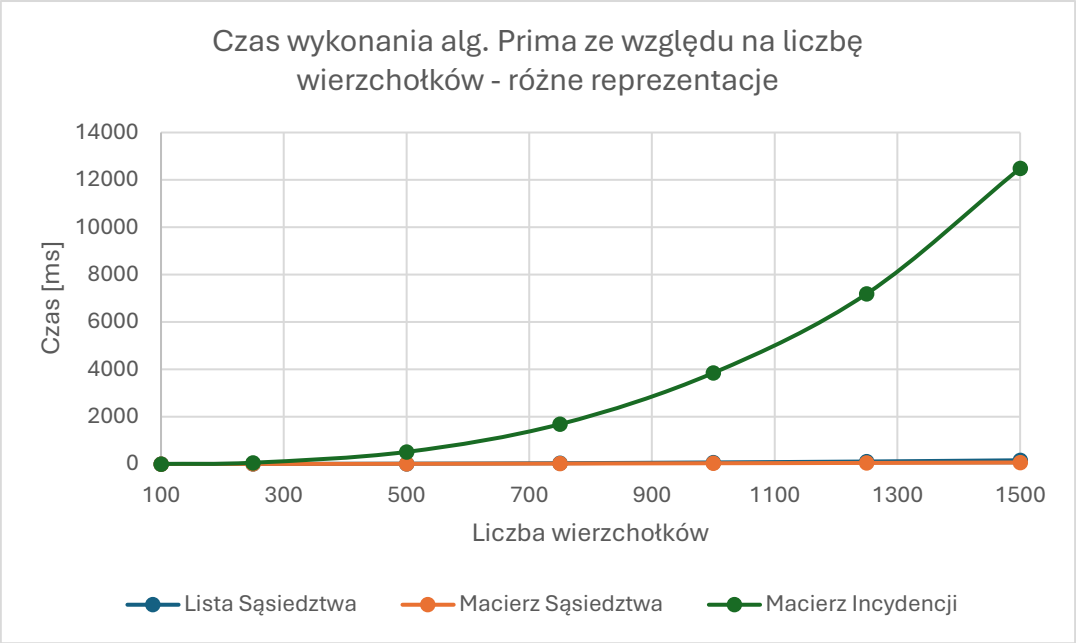
Tabela 2: średnie czasy wykonania algorytmu Prima, graf jako macierz sąsiedztwa

Reprez.	Macierz incydencji				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,58	1,30	2,40	3,91	4,67
250	6,36	19,89	53,81	88,62	119,65
500	85,04	238,78	509,42	798,44	1066,76
750	264,91	711,85	1680,54	2796,72	3813,51
1000	579,51	1655,20	3845,54	6408,96	9228,63
1250	1104,44	3300,73	7190,09	12419,84	18119,67
1500	1992,17	5372,81	12490,99	21434,36	31724,77

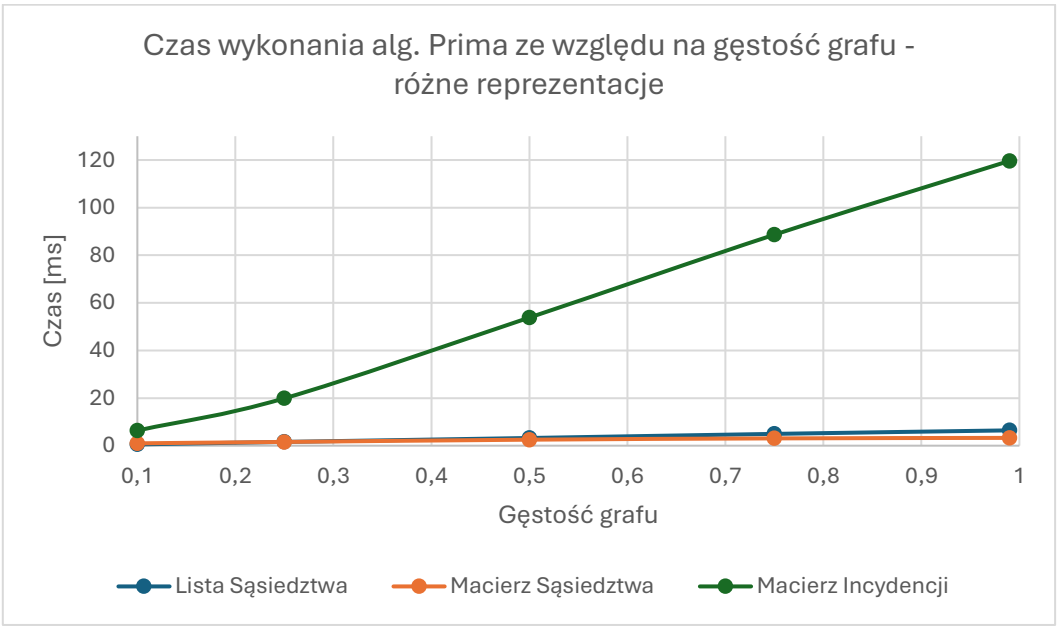
Tabela 3: średnie czasy wykonania algorytmu Prima, graf jako macierz incydencji

Uśrednione wyniki uzyskane przez algorytm Prima są przedstawione w Tabeli 1, Tabeli 2 oraz Tabeli 3.

Pierwsza zauważalna różnica, to rozbieżności pomiędzy reprezentacjami. Zdecydowanie najgorzej wypadła macierz incydencji. Czas potrzebny na wykonanie algorytmu korzystając z tej reprezentacji rósł błyskawicznie w porównaniu z macierzą sąsiedztwa i listą sąsiadów, zarówno wraz ze wzrostem wierzchołków (rys.5), jak i ze wzrostem gęstości grafu (rys.6).



Rysunek 5: czas wykonania algorytmu Prima dla różnych reprezentacji grafu, przy gęstości 50%

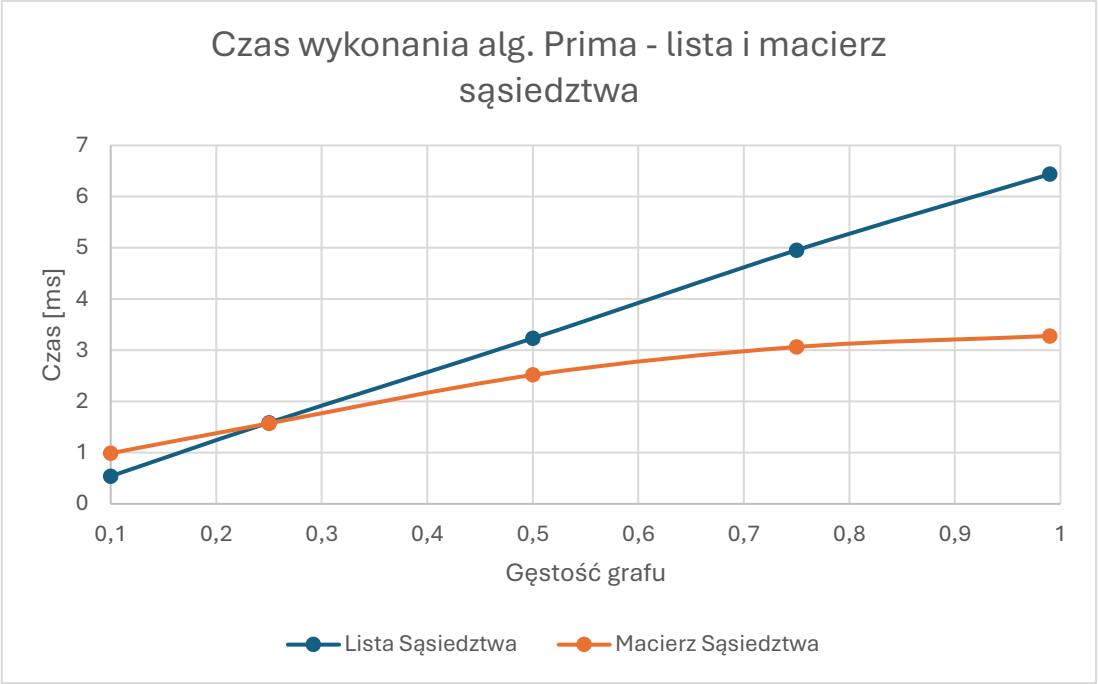


Rysunek 6: czas wykonania algorytmu Prima dla różnych reprezentacji grafu, przy 250 wierzchołkach

Główny czynnik, który sprawia, że macierz incydencji wypada tak słabo, to fragmenty algorytmu, które pobierają krawędzie wychodzące z konkretnego wierzchołka. Jest to bardzo kosztowna operacja w tej reprezentacji – trzeba przejrzeć wszystkie krawędzie w celu sprawdzenia czy nie wychodzą one z zadanego wierzchołka, a jeśli tak, to dla tej krawędzi konieczne jest znalezienie drugiego wierzchołka, który jest połączony tą krawędzią. Im więcej wierzchołków oraz im większa gęstość – tym więcej krawędzi – a więc dłuższy czas wykonania.

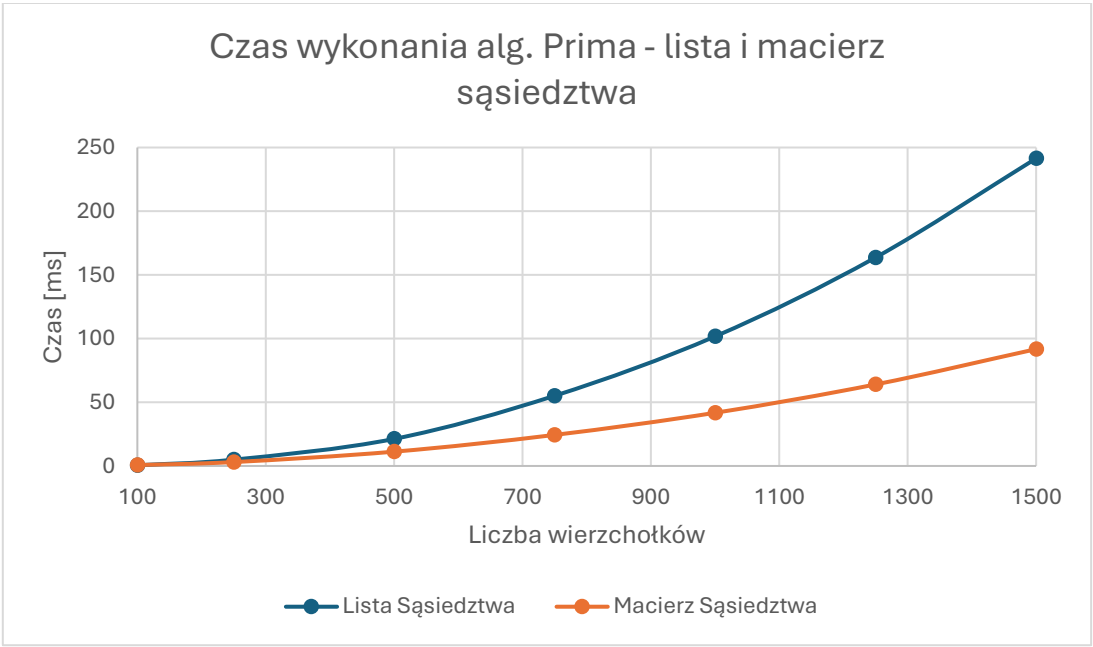
Lista sąsiedztwa i macierz sąsiedztwa nie miały tak drastycznych różnic między sobą – są one względnie niewielkie. Lista sąsiedztwa jest szybsza, gdy grafy są rzadkie. Wraz ze wzrostem gęstości to macierz sąsiedztwa zyskuje przewagę w czasie wykonania. Sytuacja

taka jak na rys.7 (od pewnej gęstości macierz sąsiedztwa jest szybszą reprezentacją) występuje dla każdej liczby wierzchołków.



Rysunek 7: czas wykonania algorytmu Prima dla listy sąsiedztwa i macierzy sąsiedztwa, w zależności od gęstości grafu, przy 250 wierzchołkach

Czas wykonania algorytmu Prima rośnie wraz ze wzrostem liczby wierzchołków. Na rys.5 już można było zauważyć, że najszybciej rośnie czas dla macierzy incydencji. Spośród pozostałych dwóch, to lista sąsiedztwa potrzebuje więcej czasu na wykonanie wraz z rosnącą liczbą wierzchołków w grafie (rys.8).



Rysunek 8: czas wykonania algorytmu Prima dla listy sąsiedztwa i macierzy sąsiedztwa, w zależności od rozmiaru grafu, przy gęstości 0,75.

6.1.2.Kruskal

Reprez.	Lista sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,11	0,25	0,68	0,81	1,04
250	0,57	1,63	3,67	5,76	7,98
500	2,91	8,48	18,56	29,72	40,86
750	7,86	21,72	47,71	78,92	112,81
1000	14,38	42,08	105,14	165,76	229,74
1250	23,34	74,22	187,47	303,14	390,93
1500	34,48	119,47	292,14	474,08	598,96

Tabela 4: średnie czasy wykonania algorytmu Prima, graf jako lista sąsiedztwa

Reprez.	Macierz sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,15	0,28	0,55	0,93	1,33
250	0,76	1,52	2,99	4,39	5,83
500	2,71	6,29	12,63	19,20	26,46
750	6,63	15,85	32,69	51,52	76,13
1000	12,04	30,12	76,05	119,91	160,11
1250	19,41	54,82	138,45	222,43	275,79
1500	28,60	90,76	217,51	354,97	430,44

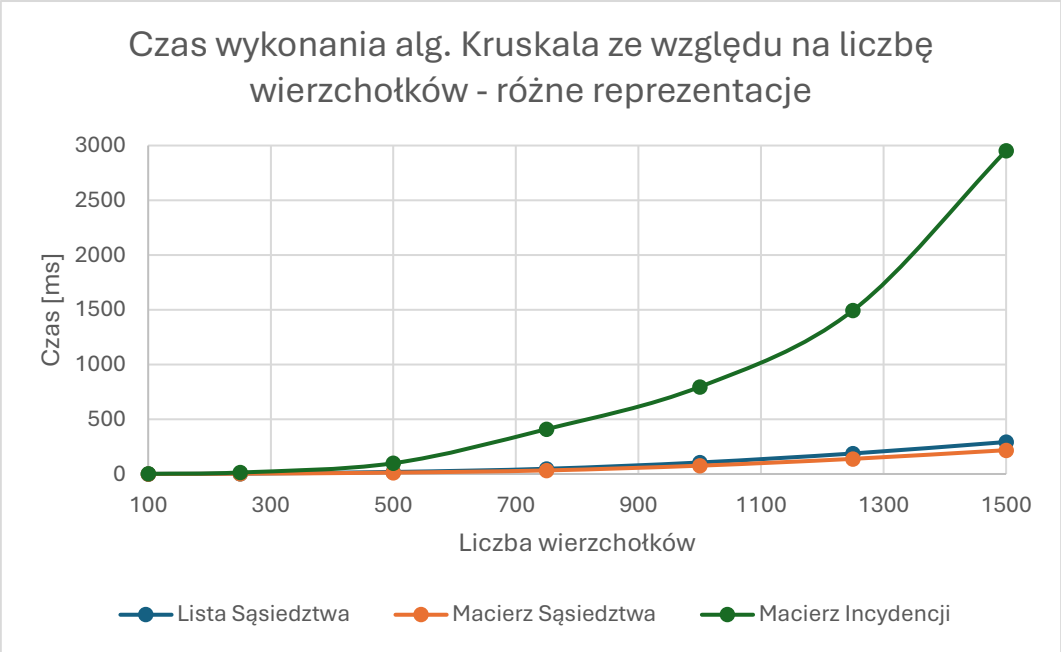
Tabela 5: średnie czasy wykonania algorytmu Prima, graf jako macierz sąsiedztwa

Reprez.	Macierz incydencji				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,25	0,58	1,36	2,17	2,60
250	2,82	6,08	13,84	20,37	21,37
500	14,49	41,42	98,37	148,32	156,49
750	58,67	150,58	409,53	549,00	675,75
1000	100,20	315,59	795,73	1143,38	1594,79
1250	213,07	1004,14	1493,62	2434,69	3745,20
1500	513,06	1373,28	2953,38	4870,39	7147,15

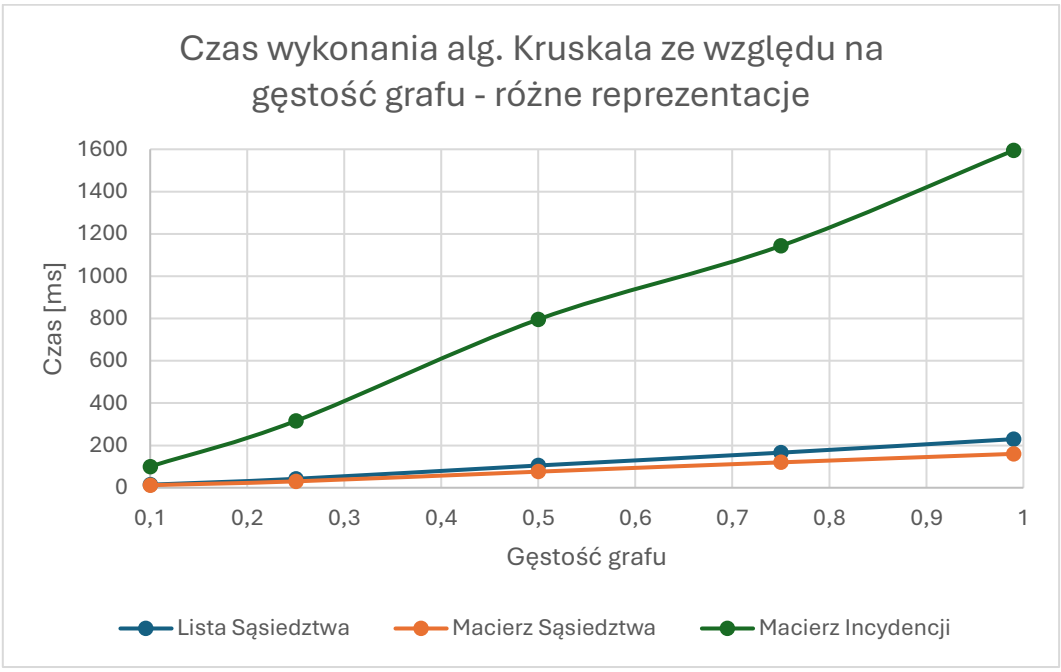
Tabela 6: średnie czasy wykonania algorytmu Kruskala, graf jako macierz incydencji

Uśrednione wyniki uzyskane przez drugi z algorytmów wyznaczania MST – algorytm Kruskala są przedstawione w Tabeli 4, Tabeli 5 oraz Tabeli 6.

W algorytmie Kruskala również najmniej optymalną reprezentacją jest macierz incydencji, której czas wykonania wraz ze wzrostem liczby wierzchołków (rys.9), jak i wzrostem gęstości grafu (rys.10) rósł zdecydowanie szybciej niż dla listy sąsiedztwa i macierzy sąsiedztwa.



Rysunek 9: czas wykonania algorytmu Kruskala dla różnych reprezentacji grafu, przy gęstości 50%



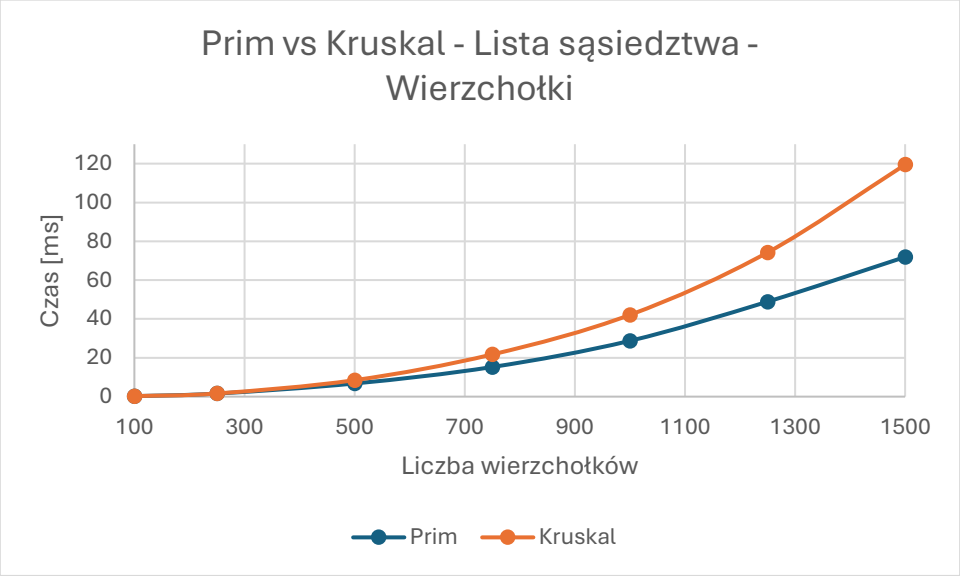
Rysunek 10: czas wykonania algorytmu Prima dla różnych reprezentacji grafu, przy 1000 wierzchołkach

Pomimo, że lista sąsiedztwa i macierz sąsiedztwa to podobne reprezentacje, których czasy są całkiem zbliżone, to na wykresach (rys.6) i (rys.7) widać, że im więcej wierzchołków, im więcej krawędzi, tym różnica między obiema reprezentacjami się powiększa na korzyść macierzy.

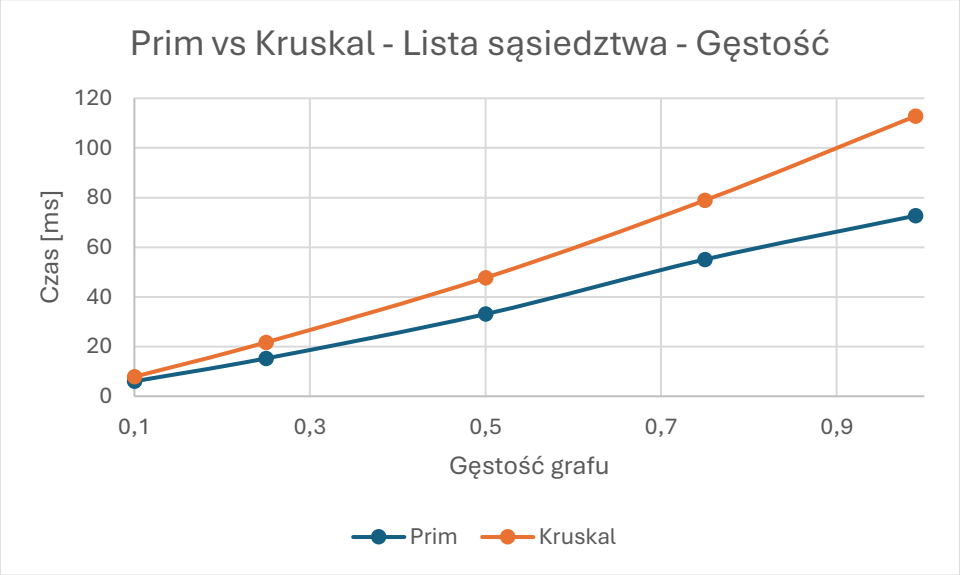
6.1.3.Porównanie algorytmów Prima i Kruskala

Porównując algorytmy wyznaczania minimalnego drzewa rozpinającego trzeba brać pod uwagę używaną reprezentację grafu, ponieważ ma ona kluczowy wpływ na czasy wykonania.

Jeżeli graf był przedstawiony za pomocą listy sąsiedztwa, to lepszym rozwiązaniem jest użycie algorytmu Prima. Przy niewielkiej liczbie wierzchołków i krawędzi (grafy rzadkie) różnice między Primem a Kruskalem są niewielkie, w niektórych przypadkach, to Kruskal jest minimalnie szybszy. Jednak przy przyroście wierzchołków (rys.11) i krawędzi (rys.12) lepszy staje się algorytm Prima.

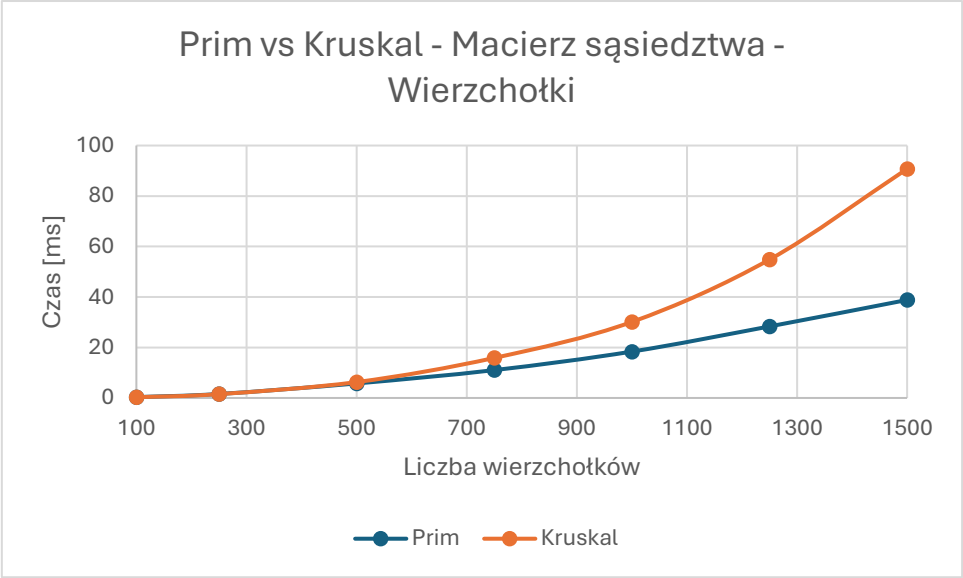


Rysunek 11: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od liczby wierzchołków, graf jako lista sąsiedztwa, gęstość 25%

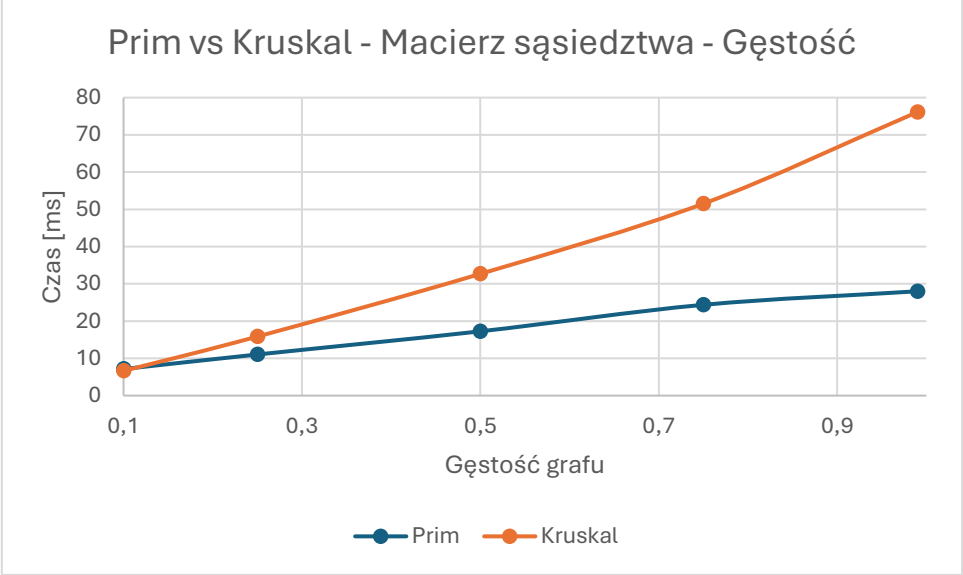


Rysunek 12: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od gęstości grafu, graf jako lista sąsiedztwa, 750 wierzchołków

Dla macierzy sąsiedztwa zachodzi podobna sytuacja. Na wykresach (rys.13) i (rys.14) widzimy podobne trendy, co na wykresach dotyczących listy sąsiedztwa. Gdy liczba wierzchołków jest niewielka, a gęstość grafu mała, to algorytmy wykonują się w niemalże identycznym czasie, czasami nawet algorytm Kruskala minimalnie szybciej. Jednak wraz ze wzrostem tych parametrów uwidocznia się przewaga algorytmu Prima.



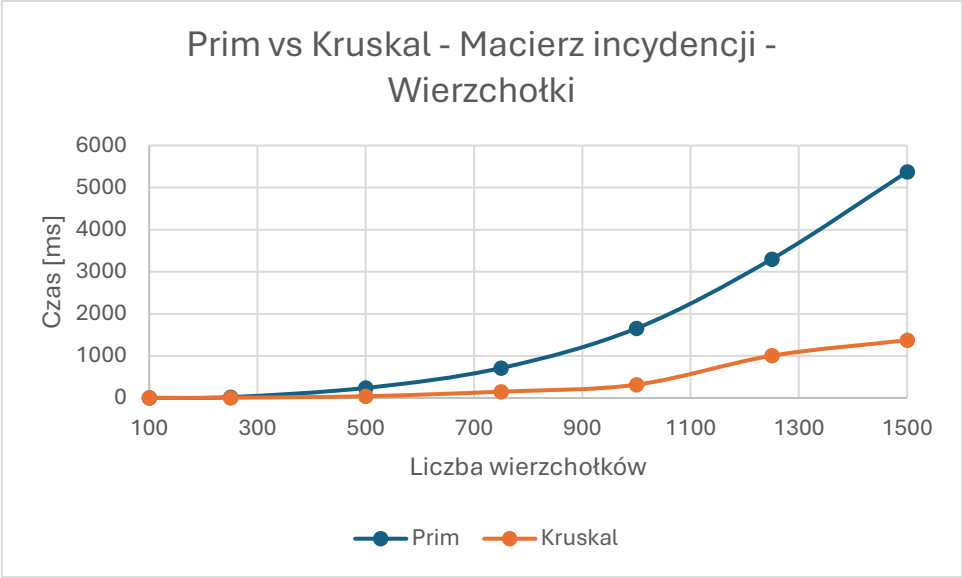
Rysunek 13: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od liczby wierzchołków, graf jako macierz sąsiedztwa, gęstość 25%



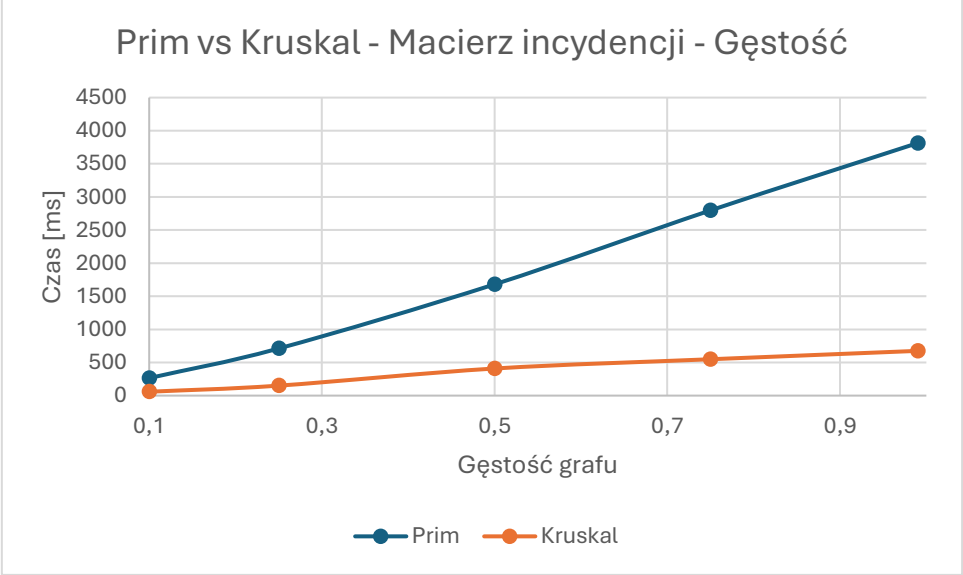
Rysunek 14: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od gęstości grafu, graf jako macierz sąsiedztwa, 750 wierzchołków

Wcześniej ustalono, że macierz sąsiedztwa i lista sąsiedztwa to najlepsze reprezentacje grafu, a porównując algorytmy wyznaczania MST korzystające z tych reprezentacji, to w większości przypadków szybszy jest algorytm Prima – dlatego ogólnie, to on jest lepszym rozwiązaniem w większości przypadków.

Jednakże, gdyby użyć trzeciej reprezentacji – macierz incydencji, którą uznano za zdecydowanie gorszą, to sytuacja się odwraca. Na wykresach (rys.15) i (rys.16) widać, że gdy liczba wierzchołków oraz gęstość rosną, to bardziej wydajny jest algorytm Kruskala.



Rysunek 15: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od liczby wierzchołków, graf jako macierz incydencji, gęstość 25%



Rysunek 16: porównanie czasów wykonania algorytmów Prima i Kruskala w zależności od gęstości grafu, graf jako macierz incydencji, 750 wierzchołków

6.1.4.Wartości maksymalne i minimalne

W tabeli 7 przedstawiono maksymalne i minimalne czasy wykonania spośród 50 prób dla każdego przypadku dla algorytmu Prima, a w tabeli 8 dla algorytmu Kruskala.

		0,1			0,25			0,5			0,75			0,99		
		LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC
		Czas [ms]														
100	MAX	0,72	0,73	1,73	0,60	0,83	2,60	0,73	1,20	3,97	0,85	1,92	6,27	0,84	1,81	6,20
	MIN	0,08	0,15	0,38	0,16	0,27	0,94	0,30	0,41	1,86	0,47	0,47	2,89	0,67	0,52	3,91
250	MAX	0,77	1,63	9,70	2,95	1,79	31,50	4,01	2,67	58,40	6,46	3,38	93,70	6,87	3,73	126,25
	MIN	0,49	0,85	5,91	1,48	1,52	18,32	3,12	2,44	52,30	4,72	2,84	86,73	6,17	3,10	117,13
500	MAX	2,90	3,67	91,08	7,31	7,52	244,53	15,26	9,82	519,29	24,02	12,30	807,56	32,03	13,64	1114,57
	MIN	2,61	3,34	82,68	6,59	5,66	235,20	13,23	8,37	500,82	20,31	10,97	785,67	27,25	12,18	1049,55
750	MAX	6,39	7,80	287,77	19,26	17,87	995,78	35,71	19,64	1722,07	100,18	44,37	3042,60	76,97	30,80	3842,52
	MIN	5,86	7,00	260,76	14,65	10,72	689,35	32,06	16,83	1580,71	52,46	23,56	2734,22	71,00	27,41	3764,76
1000	MAX	11,28	11,82	601,04	40,30	19,24	1834,08	68,58	33,42	3952,14	104,60	43,54	6605,24	141,27	53,88	9349,13
	MIN	10,14	11,45	571,93	27,28	18,07	1617,25	63,08	29,70	3737,01	99,70	40,53	6372,67	135,27	50,14	9181,99
1250	MAX	18,21	18,00	1127,51	61,45	30,59	3416,07	109,63	46,82	7254,10	165,99	69,06	12538,43	224,32	85,09	18252,10
	MIN	15,67	17,16	1093,85	46,97	27,87	3134,19	103,11	44,26	7138,70	161,25	62,34	12345,12	217,24	78,16	18047,39
1500	MAX	27,73	27,99	2019,90	75,86	40,32	5438,66	163,78	72,14	13019,02	262,05	94,46	22296,00	368,41	122,48	32408,00
	MIN	23,04	23,59	1967,27	70,18	38,06	5345,24	153,50	62,99	12412,17	236,74	88,90	21298,14	315,14	110,42	31574,22

Tabela 7: czasy maksymalne i minimalne dla algorytmu Prima

		0,1			0,25			0,5			0,75			0,99		
		LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC
		Czas [ms]														
100	MAX	0,28	0,38	1,42	0,50	0,71	1,21	1,21	1,26	2,68	1,19	2,08	3,82	1,59	2,55	3,59
	MIN	0,08	0,10	0,15	0,19	0,21	0,37	0,41	0,41	0,76	0,61	0,61	1,40	0,85	0,80	1,87
250	MAX	0,62	1,64	3,77	1,99	1,60	7,38	4,20	3,17	15,32	6,87	5,83	23,24	11,09	6,43	22,76
	MIN	0,54	0,60	2,60	1,46	1,44	5,82	3,27	2,90	13,36	5,31	4,23	19,18	7,39	5,67	20,65
500	MAX	3,34	3,14	16,03	9,61	7,45	47,70	20,32	18,86	113,29	32,80	20,95	162,05	50,13	33,63	172,89
	MIN	2,69	2,59	13,97	8,01	6,16	39,23	17,97	12,17	95,20	28,64	18,74	141,39	39,31	25,54	147,94
750	MAX	9,73	7,67	62,34	24,26	16,60	165,34	53,05	40,49	524,29	95,43	55,69	577,41	139,76	92,33	721,76
	MIN	7,54	6,45	56,89	20,92	15,29	146,34	45,99	30,68	282,92	75,28	49,74	517,77	109,11	72,75	641,73
1000	MAX	16,08	13,09	109,30	54,20	37,92	343,33	111,13	96,94	849,06	170,75	153,73	1189,45	237,49	165,80	1626,82
	MIN	13,91	11,69	95,69	39,74	28,76	283,05	100,75	69,66	755,90	161,58	112,42	1114,49	224,73	153,85	1564,17
1250	MAX	27,20	24,75	263,03	80,15	64,57	1141,33	193,48	146,33	1536,43	315,45	237,58	2493,74	444,04	293,34	3837,53
	MIN	21,96	18,28	199,96	70,46	51,85	682,82	179,97	131,46	1462,64	291,99	212,38	2391,87	382,35	268,79	3672,09
1500	MAX	36,20	31,02	534,46	124,93	116,64	1404,03	299,67	227,61	3033,28	496,14	384,46	5227,36	633,32	448,10	7317,15
	MIN	33,05	27,95	498,06	112,65	82,95	1339,47	284,27	209,12	2871,00	454,35	334,03	4779,65	586,29	416,67	6961,45

Tabela 8: czasy maksymalne i minimalne dla algorytmu Kruskala

W niektórych przypadkach skrajne wartości mocno odbiegały od wartości średnich (tabele 1-6). Takie wartości uzyskano między innymi w przypadku:

- Prima, 100 wierzchołków, gęstość 0,75, macierz incydencji, MAX
- Prima, 750 wierzchołków, gęstość 0,75, lista sąsiedztwa, MAX
- Kruskala, 1000 wierzchołków, gęstość 0,75, macierz sąsiedztwa, MAX

Te wartości były bliższe wartościom uzyskanym dla innego zestawu parametrów, niż dla tych, które miało ich wywołanie. Mogło to być spowodowane bardzo specyficznym grafem, który sprawiał, że przypadek był pesymistyczny i wykonywał się dłużej, ale powodem mogło być też chwilowe, niezależne obciążenie komputera, które sprawiło, że czas się wydłużył, a przy niewielkich wartościach, odchylenia są zauważalne.

Generalnie jednak wartości oscylowały w granicach wartości średniej. Odchylenie naturalnie rosło wraz ze średnim czasem wykonania, ale względnie pozostawało na podobnym poziomie.

6.2.Wyznaczanie najkrótszej ścieżki w grafie

Wyniki dotyczące problemu wyznaczania najkrótszej ścieżki w grafie, algorytmów Dijkstry oraz Bellmana-Forda.

6.2.1.Algorytm Dijkstry

Reprez.	Lista sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,12	0,29	0,39	0,55	0,76
250	0,53	1,53	3,20	4,83	6,57
500	2,64	6,46	13,39	20,36	27,32
750	5,48	13,73	30,15	49,09	67,39
1000	9,22	24,82	58,11	92,86	126,20
1250	13,69	41,65	95,87	148,89	199,88
1500	20,65	65,32	141,43	217,14	294,72

Tabela 9: średnie czasy wykonania algorytmu Dijkstry, graf jako lista sąsiedztwa

Reprez.	Macierz sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,20	0,33	0,75	0,71	0,78
250	0,88	1,56	2,45	3,08	3,33
500	3,31	5,16	8,09	10,95	13,15
750	6,55	10,48	16,79	23,62	29,07
1000	11,17	18,28	29,54	41,54	51,62
1250	16,75	27,58	46,62	64,57	84,24
1500	23,99	39,97	67,43	96,01	119,28

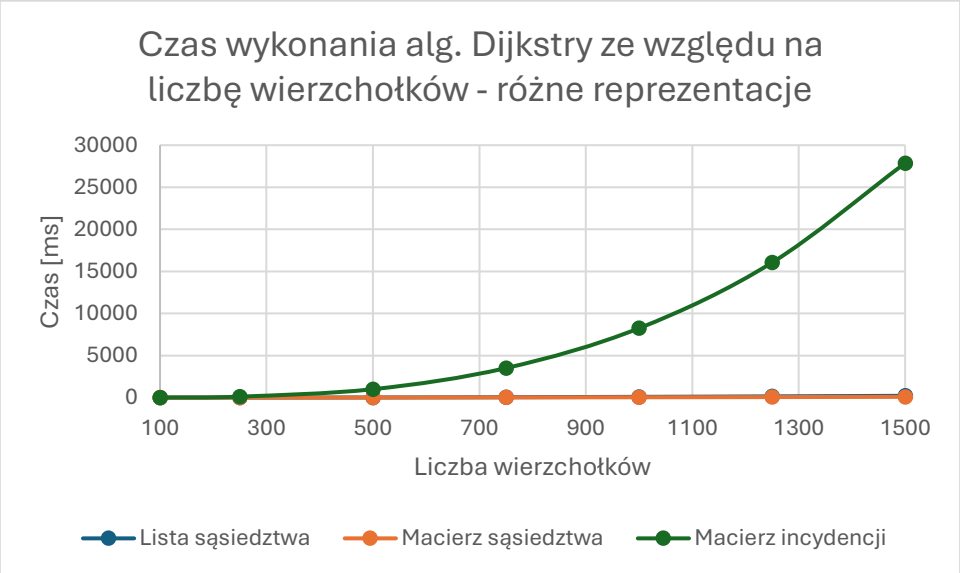
Tabela 10: średnie czasy wykonania algorytmu Dijkstry, graf jako macierz sąsiedztwa

Reprez.	Macierz incydencji				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	0,81	2,69	4,40	5,44	7,08
250	10,83	34,80	77,04	119,58	159,86
500	124,99	323,16	654,32	1001,77	1369,81
750	429,96	1164,16	2327,58	3509,98	4624,34
1000	1032,59	2822,44	5496,36	8245,90	10914,60
1250	2080,13	5312,64	10707,22	16051,05	21178,77
1500	3724,95	9204,61	18536,49	27869,74	40068,88

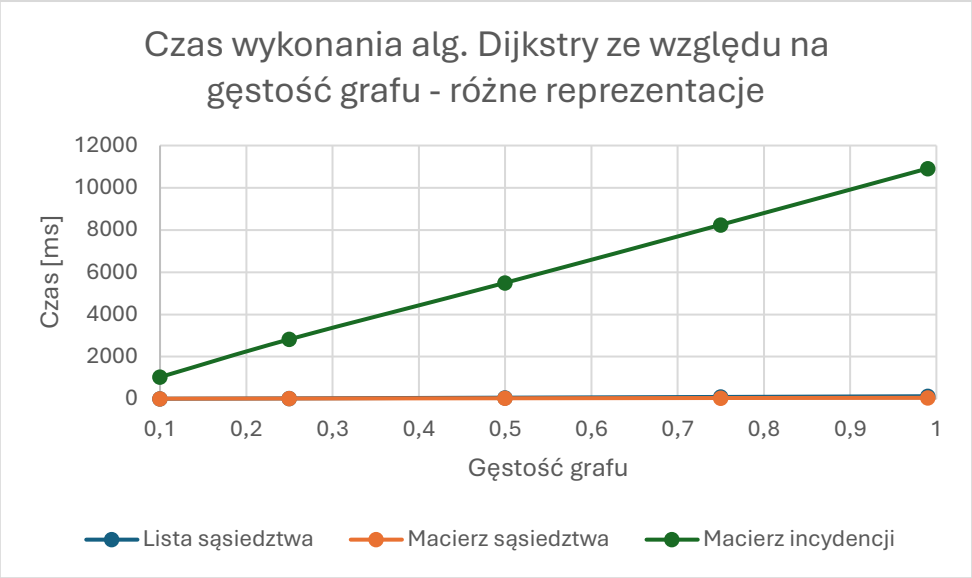
Tabela 11: średnie czasy wykonania algorytmu Dijkstry, graf jako macierz incydencji

Uśrednione wyniki uzyskane przez algorytm Dijkstry są przedstawione w Tabeli 9, Tabeli 10 oraz Tabeli 11.

Algorytm Dijkstry ma cechę wspólną z algorytmem Prima – rozpatruje krawędzie wychodzące z danego wierzchołka. Tak jak w Primie, powoduje to bardzo duży czas wykonania macierzą incydencji, a bardzo szybki wzrost widać na przykładowych wykresach: czasu wykonania w zależności od liczby wierzchołków (rys.17) i gęstości grafu (rys.18).

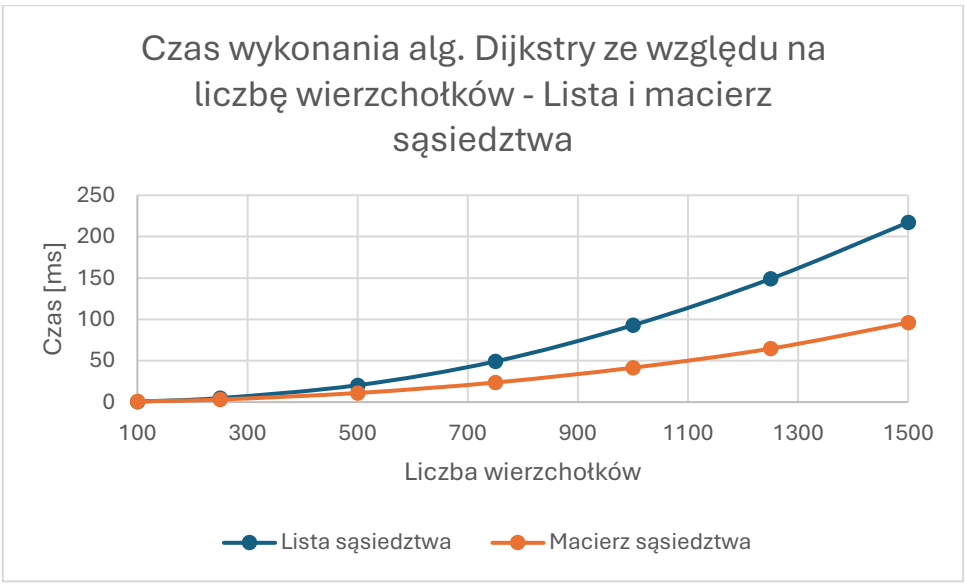


Rysunek 17: czas wykonania algorytmu Dijkstry dla różnych reprezentacji grafu, przy gęstości 75%

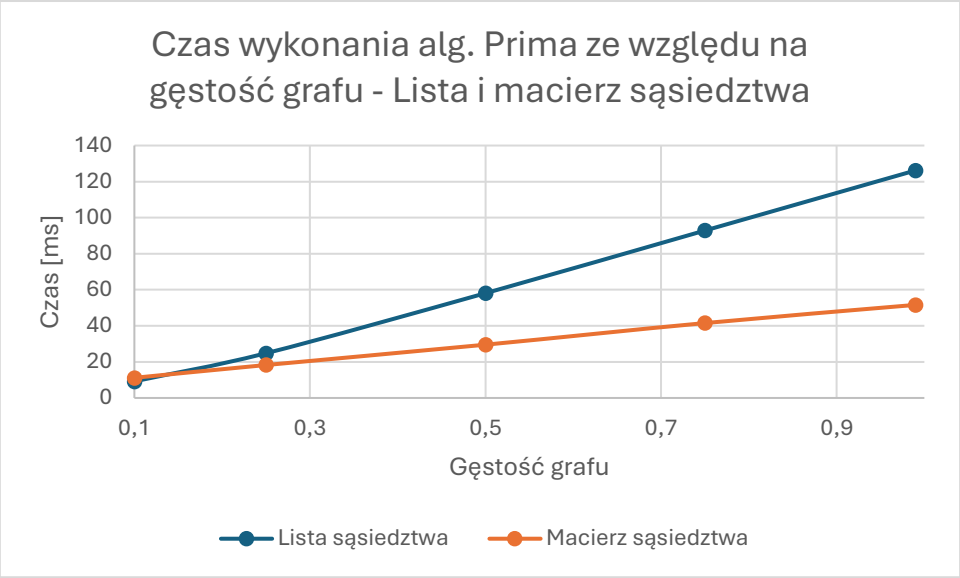


Rysunek 18: czas wykonania algorytmu Dijkstry dla różnych reprezentacji grafu, przy 1000 wierzchołkach

Widoczna jest ogromna przewaga w szybkości wykonania pozostałych dwóch reprezentacji. Żeby lepiej zwizualizować porównanie pomiędzy listą sąsiedztwa i macierzą sąsiedztwa, posłużymy się wykresami (rys.19) i (rys.20), które pokazują zmianę czasu wykonania dla tych dwóch algorytmów.



Rysunek 19: czas wykonania algorytmu Dijkstry dla listy sąsiedztwa i macierzy sąsiedztwa, przy gęstości 75%



Rysunek 20: czas wykonania algorytmu Dijkstry dla listy sąsiedztwa i macierzy sąsiedztwa, przy 1000 wierzchołkach

Sytuacja jest bliźniacza jak w algorytmach wyznaczających MST. Wraz ze wzrostem liczby wierzchołków oraz większą gęstością, wydajniejsza reprezentacja to macierz sąsiedztwa. Natomiast gdy wierzchołków jest niewiele, to różnice między dwiema reprezentacjami są minimalne. Przy małej gęstości grafu, to lista sąsiedztwa działa szybciej.

6.2.2.Algorytm Bellmana-Forda

Reprez.	Lista sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	1,33	1,94	3,68	5,62	7,43
250	11,31	28,72	58,93	92,71	126,79
500	103,97	287,80	554,18	914,95	1325,24
750	313,23	836,57	2060,75	3089,07	4176,58
1000	748,82	2423,58	4944,88	7539,75	10052,01
1250	1615,76	4688,80	9845,98	14892,40	19752,72
1500	3184,58	8362,57	17166,61	26005,32	34231,48

Tabela 12: średnie czasy wykonania algorytmu Bellmana-Forda, graf jako lista sąsiedztwa

Reprez.	Macierz sąsiedztwa				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	1,24	2,93	5,74	7,38	8,70
250	11,77	29,12	58,06	90,78	123,39
500	104,42	284,80	554,29	913,13	1319,29
750	313,71	836,43	2060,89	3095,54	4174,37
1000	749,46	2433,56	4972,77	7559,52	10056,39
1250	1628,13	4713,81	9823,11	14862,48	19726,28
1500	3197,25	8363,11	17132,06	25918,35	34203,51

Tabela 13: średnie czasy wykonania algorytmu Bellmana-Forda, graf jako macierz sąsiedztwa

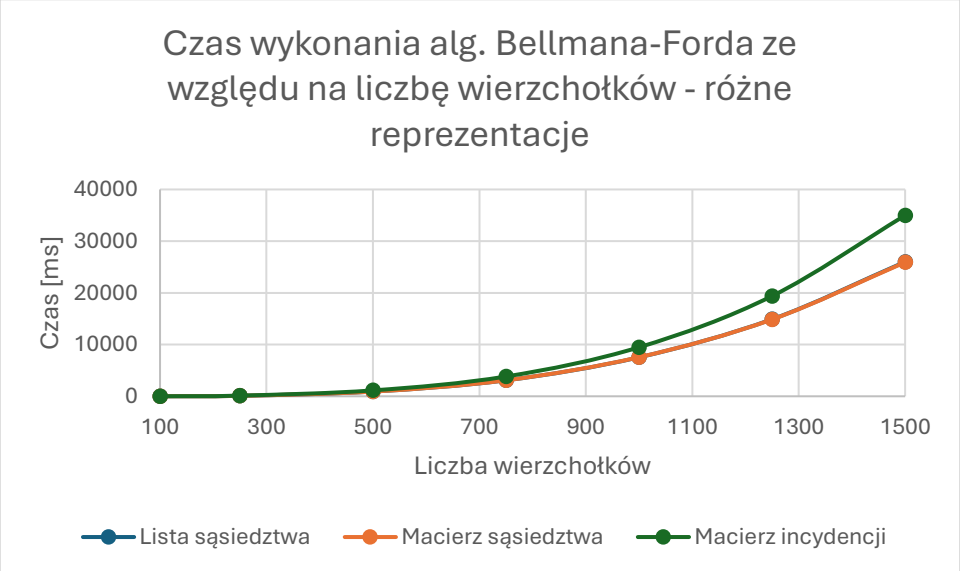
Reprez.	Macierz incydencji				
Gęstość	0,1	0,25	0,5	0,75	0,99
Wierzch	Czas [ms]				
100	1,83	3,56	5,56	8,43	11,04
250	16,81	41,84	80,41	130,19	159,98
500	135,32	377,43	671,93	1138,66	1615,69
750	412,08	1156,34	2509,14	3830,41	5248,02
1000	941,87	3106,47	6122,18	9479,95	13053,65
1250	2041,25	5903,32	12538,92	19388,10	26918,82
1500	4204,43	10879,97	22624,46	34991,22	48583,92

Tabela 14: średnie czasy wykonania algorytmu Bellmana-Forda, graf jako macierz sąsiedztwa

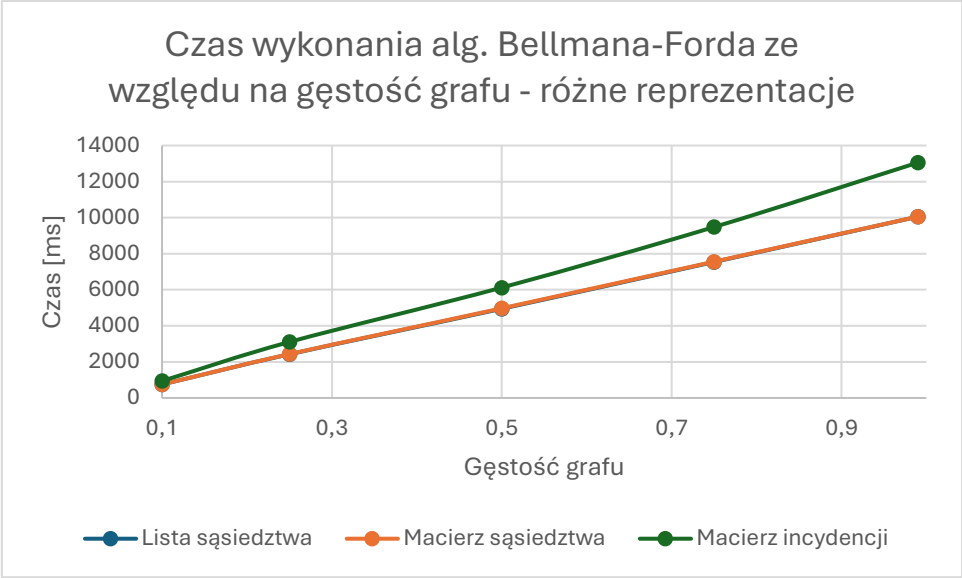
Tabela 12, Tabela 13 oraz Tabela 14 zawierają uśrednione wyniki uzyskane algorytmem Bellmana-Forda.

Algorytm Bellmana-Forda to algorytm, w którym różnice pomiędzy różnymi reprezentacjami są najmniejsze, spośród całej badanej czwórki. Lista sąsiedztwa i macierz sąsiedztwa ponownie okazały się szybsze niż macierz incydencji, ale w przypadku tego algorytmu macierz sąsiedztwa nie staje się szybsza wraz ze wzrostem liczby wierzchołków oraz wzrostem gęstości. Na wykresach (rys.21) i (rys.22) przez całą dziedzinę linie przypisane tym reprezentacjom pokrywają się.

Macierz incydencji, pomimo, że znów okazała się najwolniejszą reprezentacją, to jej ucieczka nie jest tak szybka. Nawet przy dużych grafach, zawierających dużo wierzchołków i krawędzi różnica między nią, a pozostałymi dwiema reprezentacjami nie jest kolosalna (tak jak było w poprzednich algorytmach).



Rysunek 21: czas wykonania algorytmu Bellmana-Forda dla różnych reprezentacji grafu, przy gęstości 75%

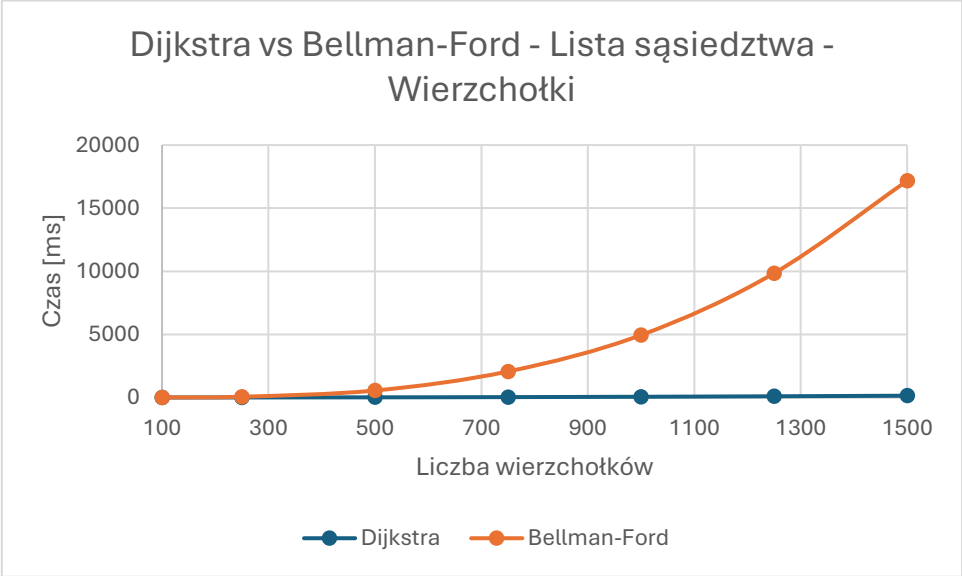


Rysunek 22: czas wykonania algorytmu Dijkstry dla różnych reprezentacji grafu, przy 1000 wierzchołkach

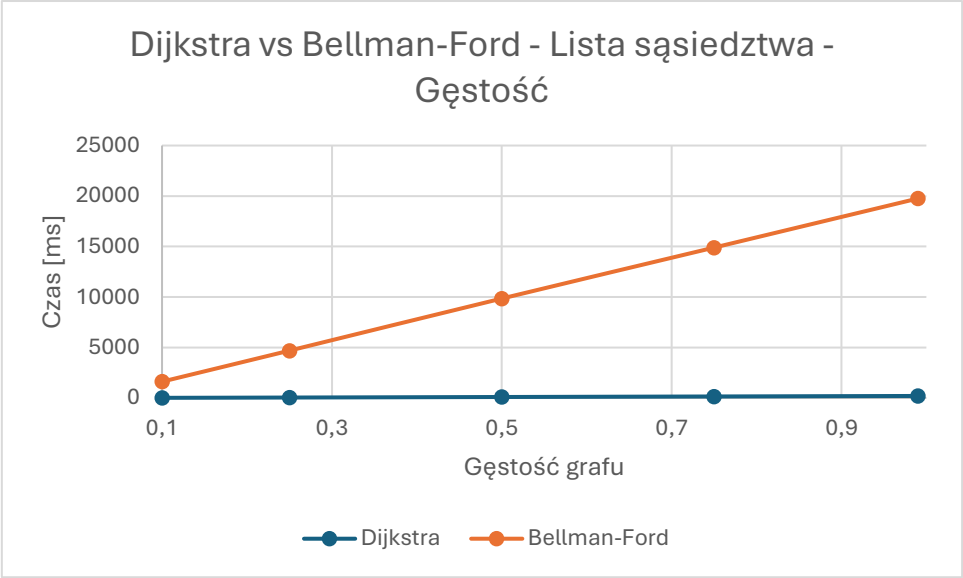
6.2.3.Porównanie algorytmów Dijkstry i Bellmana-Forda

Porównując algorytm Dijkstry z Bellmanem-Fordem wprowadzamy podział na reprezentacje, ze względu na różne wyniki zwracane w zależności od użytej reprezentacji grafu.

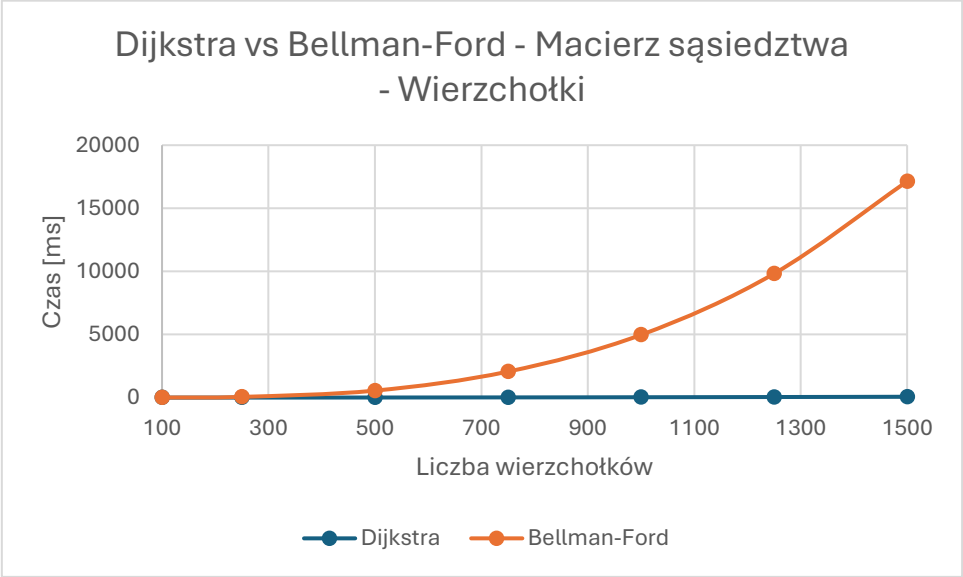
Sprawa jest klarowna, gdy używana jest lista sąsiedztwa lub macierz sąsiedztwa – algorytm Dijkstry jest zdecydowanie szybszy. Wykresy porównujące oba algorytmy w zależności od liczby wierzchołków (rys.23) i (rys.25) oraz w zależności od gęstości grafu (rys.22) i (rys.24) nie pozostawiają złudzeń i świetnie obrazują jak o wiele więcej czasu zajmuje wykonanie algorytmu Bellmana-Forda.



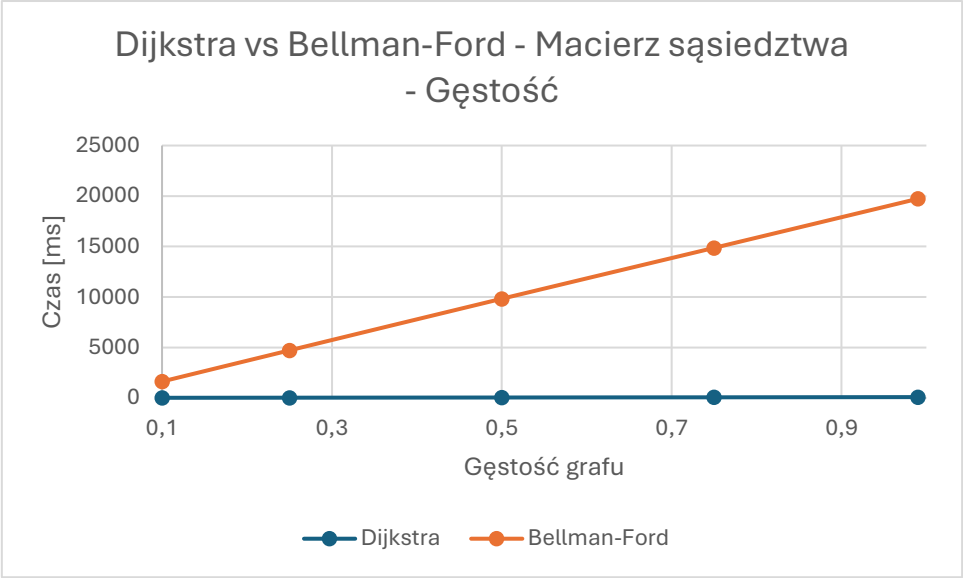
Rysunek 23: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od liczby wierzchołków, graf jako lista sąsiedztwa, gęstość 50%



Rysunek 24: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od gęstości grafu, graf jako lista sąsiedztwa, 1250 wierzchołków



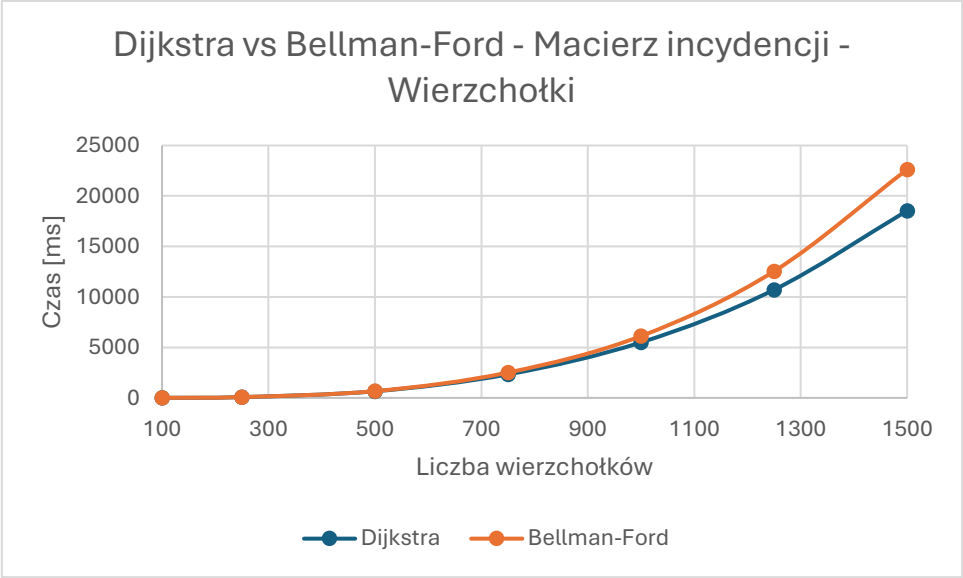
Rysunek 25: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od liczby wierzchołków, graf jako macierz sąsiedztwa, gęstość 50%



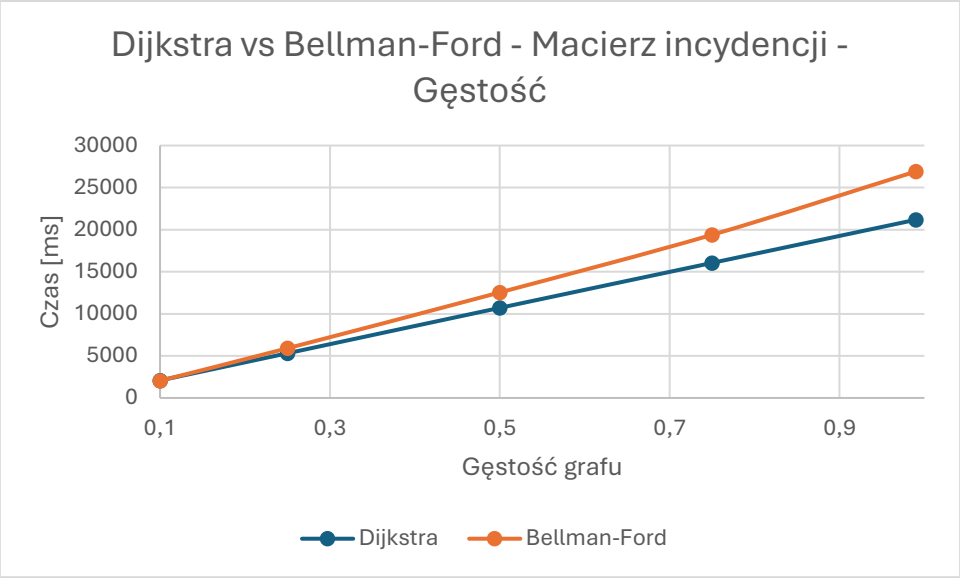
Rysunek 26: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od gęstości grafu, graf jako macierz sąsiedztwa, 1250 wierzchołków

Dla najlepszych reprezentacji grafu zdecydowanie wydajniejszy jest algorytm Dijkstry, ale trzeba pamiętać o jego ograniczeniach. Nie radzi on sobie z grafami, w których krawędzie mają wagi ujemne. Dlatego Bellman-Ford, pomimo, że zajmuje dużo więcej czasu, nadal ma swoje zastosowania.

Gdy graf jest przedstawiony w programie jako macierz incydencji, to również szybszy jest algorytm Dijsktry, ale w tym wypadku różnica nie jest tak duża. (rys.27) i (rys.28)



Rysunek 27: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od liczby wierzchołków, graf jako macierz incydencji, gęstość 50%



Rysunek 28: porównanie czasów wykonania algorytmów Dijsktry i Bellmana-Forda w zależności od gęstości grafu, graf jako macierz incydencji, 1250 wierzchołków

6.2.4.Wartości maksymalne i minimalne

W tabelach przedstawiono wartości maksymalne i minimalne uzyskane przez algorytmy Dijsktry (tabela 15) oraz Bellmana-Forda (tabela 16) w różnych konfiguracjach.

		0,1			0,25			0,5			0,75			0,99		
		LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC
		Czas [ms]														
100	MAX	0,29	0,55	2,54	0,46	0,92	4,38	0,57	1,44	7,71	0,61	1,56	7,09	0,83	1,39	9,59
	MIN	0,08	0,15	0,57	0,16	0,26	1,45	0,33	0,41	3,04	0,52	0,47	4,71	0,72	0,53	6,35
250	MAX	0,67	0,96	13,36	1,60	1,76	37,25	3,81	2,58	88,58	5,20	8,77	127,73	12,26	3,76	168,22
	MIN	0,51	0,85	10,08	1,49	1,52	33,92	3,09	2,40	75,34	4,60	2,84	117,03	6,13	3,16	156,15
500	MAX	6,54	3,48	141,99	6,97	5,42	335,95	17,65	8,37	715,32	26,20	11,40	1075,61	30,96	14,13	1476,05
	MIN	2,49	3,24	122,88	6,28	5,07	317,35	12,51	7,95	643,31	19,66	10,77	987,42	26,60	12,89	1348,42
750	MAX	6,07	6,83	450,46	16,28	11,13	1209,07	35,06	17,59	2353,69	53,85	24,94	3536,50	69,90	35,81	4695,38
	MIN	5,35	6,46	422,93	13,29	10,28	1099,22	29,04	16,48	2295,09	47,51	22,98	3485,08	65,96	28,33	4593,24
1000	MAX	11,80	14,05	1112,54	28,11	26,16	2863,10	61,50	31,85	5856,45	98,34	43,21	8282,35	129,63	53,75	11524,94
	MIN	8,91	10,94	1015,01	23,77	17,72	2718,84	56,57	29,02	5462,09	91,28	41,09	8212,07	124,24	50,98	10847,85
1250	MAX	18,46	18,75	2105,15	53,25	29,80	5368,38	98,60	54,88	11292,69	157,78	67,83	16244,91	203,33	94,98	21813,13
	MIN	13,14	16,48	2050,09	40,27	27,05	5281,19	94,61	45,68	10651,20	146,06	62,59	15947,51	194,37	77,24	21072,20
1500	MAX	24,41	26,84	3959,52	68,67	41,83	9265,48	152,07	71,99	19571,72	231,71	107,93	30022,01	345,54	129,46	53262,71
	MIN	19,53	23,46	3676,86	63,61	39,51	9150,33	138,72	65,56	18422,79	210,42	89,73	27635,06	285,37	116,20	36570,56

Tabela 15: czasy maksymalne i minimalne dla algorytmu Dijkstry

		0,1			0,25			0,5			0,75			0,99		
		LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC	LISTA	M SĄS	M INC
		Czas [ms]														
100	MAX	2,47	3,47	3,95	2,57	5,81	7,66	3,83	9,81	6,11	7,47	10,93	9,54	7,85	13,44	13,95
	MIN	0,62	0,69	0,78	1,73	1,81	2,17	3,61	3,69	5,10	5,48	5,57	8,06	7,26	7,30	10,47
250	MAX	11,78	16,62	18,66	35,34	34,03	43,37	60,74	61,62	89,22	94,32	96,69	138,40	133,77	135,45	169,75
	MIN	11,16	11,39	16,08	28,34	28,31	40,53	58,16	57,32	77,30	91,33	89,31	125,96	124,20	120,31	156,63
500	MAX	113,19	109,59	151,33	298,57	300,92	389,83	756,07	755,33	886,99	926,61	932,68	1161,40	1344,85	1372,15	1645,90
	MIN	103,09	103,47	131,64	279,76	277,46	366,91	535,45	530,80	650,88	900,36	898,77	1118,12	1301,76	1300,82	1592,16
750	MAX	326,59	317,83	421,16	873,15	930,13	1278,55	2092,19	2105,90	2542,03	3134,91	3135,64	3894,59	4331,39	4241,56	5327,83
	MIN	310,90	311,91	407,95	822,80	824,34	1055,90	2027,43	2031,03	2481,70	3046,89	3042,66	3787,63	4114,25	4111,06	5167,98
1000	MAX	759,71	761,42	952,48	2465,42	2489,04	3190,11	5053,68	5090,86	6294,20	7638,01	7798,91	9610,82	11082,02	10999,45	14350,70
	MIN	741,28	742,30	928,26	2394,30	2400,51	2993,29	4883,55	4896,20	6047,76	7441,92	7479,22	9341,83	9893,80	9890,21	12745,07
1250	MAX	1763,03	1789,06	2196,55	4828,14	5131,79	6025,79	9969,56	9964,30	12928,55	15323,10	15040,22	19869,64	20185,13	19884,78	27798,52
	MIN	1559,94	1555,99	1973,36	4611,77	4608,66	5810,84	9721,66	9733,52	12316,08	14769,35	14726,31	19109,76	19523,81	19592,20	26603,90
1500	MAX	3257,59	3289,98	4263,00	8587,26	8498,89	11322,05	17512,24	17574,87	23085,66	28075,25	27280,12	38944,13	35028,80	36510,04	50040,47
	MIN	3130,39	3140,55	4142,34	8257,83	8239,15	10696,87	16943,97	16896,15	22325,87	25666,74	25628,70	34320,08	33962,15	33832,82	47788,29

Tabela 16: czasy maksymalne i minimalne dla algorytmu Bellmana-Forda

W przypadku algorytmów wyznaczających najkrótsze ścieżki w grafie, również znajdują się zestawy parametrów dla których wartość maksymalna bądź minimalna jest odchylona dość mocno od wartości średniej. Przykłady to:

- Algorytm Dijkstry, 1000 wierzchołków, gęstość 0,25, Macierz sąsiedztwa, MAX
- Algorytm Dijkstry, 1250 wierzchołków, gęstość 0,5, Macierz sąsiedztwa, MAX
- Algorytm Bellmana-Forda, 100 wierzchołków, gęstość 0,5, Macierz sąsiedztwa, MAX
- Algorytm Bellmana-Forda, 100 wierzchołków, gęstość 0,25, Macierz Incydencji, MAX
- Algorytm Bellmana-Forda, 500 wierzchołków, gęstość 0,5, wszystkie trzy reprezentacje, MAX

Bellman-Ford to algorytm o większej złożoności, więc bezwzględne różnice muszą być większe niż w algorytmie Dijkstry. Nie zważywszy na to, i tak częściej odchylenia występowały właśnie w algorytmie Bellmana-Forda, szczególnie przy mniejszej liczbie wierzchołków. Powody ich występowania mogą być takie same, jak w przypadku różnic w algorytmach wyznaczania MST.

7.Porównanie z inną implementacją

W tej sekcji porównana zostanie implementacja algorytmu Kruskala oraz Prima z innymi. Wykorzystany zostanie znaleziony i dostosowany do celu tego zadania dataset.

7.1.Dataset

W porównaniu zostanie wykorzystany dataset, który pochodzi ze strony networkrepository.com [10]. Jego nazwa to USAIR97. Zawiera on 332 wierzchołki oraz 2126 krawędzi. Jest grafem spójnym, nieskierowanym oraz ważonym.

Wagi w tym grafie były liczbami z częścią dziesiętną, dlatego wcześniej w programie Excel przygotowano ten zbiór, tak, żeby spełniał on założenia naszego programu, czyli wagi liczbami naturalnymi – w tym celu pomnożono oryginalne wagi przez 10000. Ponadto na początku pliku tekstowego zamieszczono liczbę wierzchołków oraz krawędzi, gdyż tego wymaga program.

7.2.Inne implementacje

Moje implementacje algorytmów Prima i Kruskala zostaną porównane z implementacjami tych algorytmów w Pythonie. Będą one pochodzić ze strony Serwisu Edukacyjnego w Tarnowie [11].

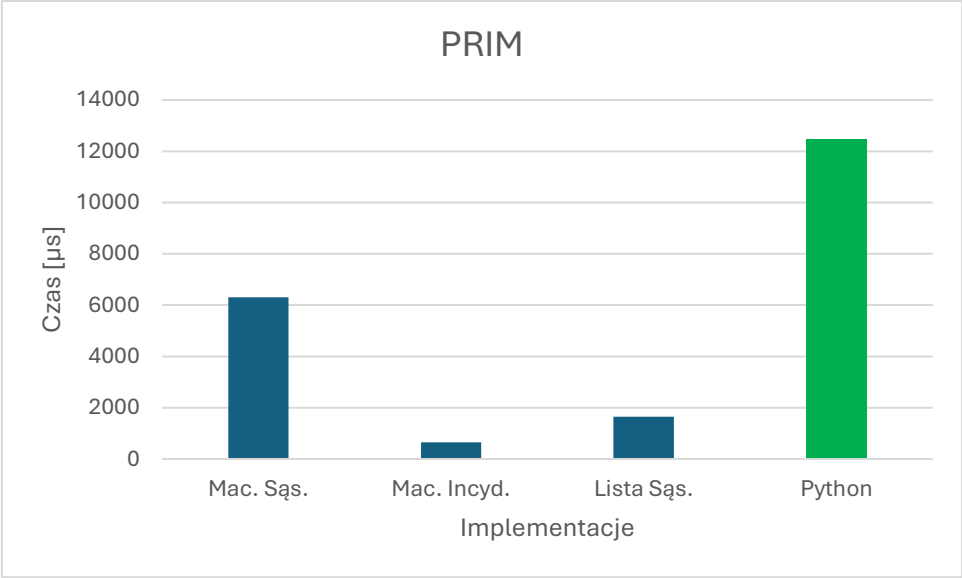
Użyty kod musi być delikatnie zmodyfikowany. Dodany zostanie pomiar czasu, żeby można było dokonać porównania. Oryginalny kod wymaga wprowadzanie krawędzi używając strumienia wejściowego. W celu użycia gotowego datasetu z pliku wejściowego zmienimy fragmenty odpowiedzialne za wczytywanie danych wejściowych, aby pobrały dane z pliku tekstowego, zawierającego dataset.

7.3.Porównanie

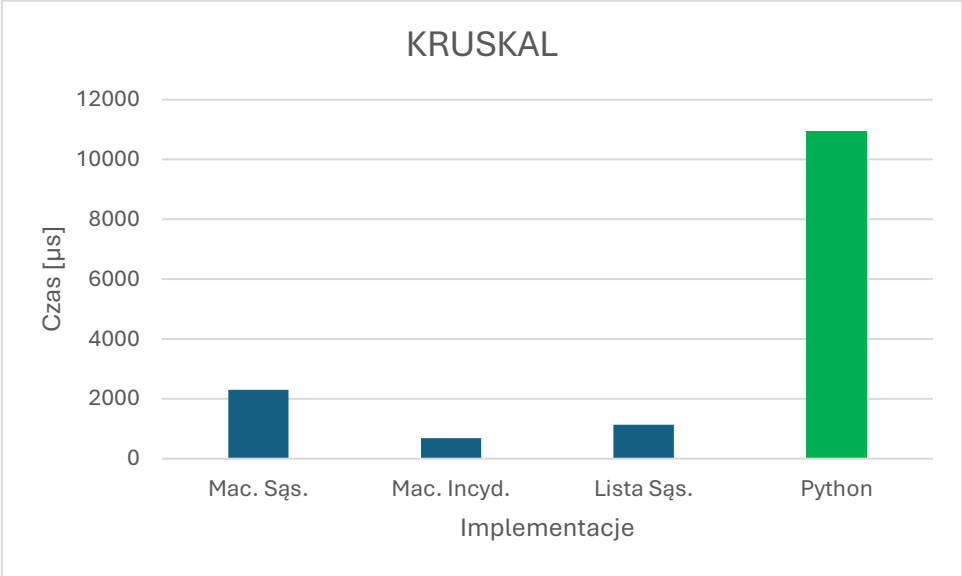
Zewnętrzna implementacja w Pythonie zarówno w przypadku algorytmu Prima (rys.29) jak i algorytmu Kruskala (rys.30) okazała się wolniejsza dla wybranego datasetu. Wszystkie trzy reprezentacje na których przetestowaliśmy algorytmy na znalezionym datasecie umożliwiły osiągnąć lepsze wyniki.

Na pewno wpływ ma na to C++, który jest językiem szybszym niż Python. Różnica nie jest duża, ale testowany zbiór nie miał ogromnych rozmiarów – można podejrzewać, że wraz ze wzrostem liczby krawędzi i wierzchołków różnica między implementacjami powiększała by się.

Co ciekawe oba algorytmy wyznaczające MST grafu z datasetu najszybciej wykonały się dla macierzy incydencji – co jest odmianą dla badań. Wpływ może mieć na to gęstość tego grafu, która była mniejsza niż testowane grafy w sekcji badań.



Rysunek 29: porównanie implementacji w C++ z implementacją w Pythonie – algorytm Prima



Rysunek 30: porównanie implementacji w C++ z implementacją w Pythonie – algorytm Kruskala

8.Podsumowanie

8.1.Wnioski z badań

Badania przeprowadzano na grafach w trzech reprezentacjach, Najlepszą w większości przypadków była **macierz sąsiedztwa.** Ta reprezentacja zapewniała najlepsze wyniki nawet przy bardzo gęstych grafach. W grafach, w których gęstość była mała, algorytmy wykonywały się szybciej używając **listy sąsiedztwa,** choć różnice między macierzą a listą nie były duże.

Obie reprezentacje zachowywały się tak dla wszystkich czterech badanych algorytmów.

Najgorsza reprezentacja to **macierz incydencji.** Wszystkie algorytmy wykonywały się najdłużej, gdy korzystały z macierzy incydencji. Szczególnie mocno spowalniały algorytmy Prima i Dijkstry – algorytmy bazujące na przeglądaniu krawędzi wychodzących z danego wierzchołka.

W badaniach dotyczących algorytmów wyznaczania minimalnego drzewa rozpinającego mierzono czasy wykonania algorytmów Prima i Kruskala.

Bardziej wydajnym algorytmem okazał się **algorytm Prima,** co zgadzałoby się z książkowymi złożonościami. Jego przewaga nad Kruskalem rosła wraz ze wzrostem liczby wierzchołków i krawędzi. Przy niewielkich gęstościach i rozmiarach grafu, oba algorytmy wykonywały się w podobnych czasach.

Aczkolwiek używając nieefektywnej macierzy incydencji to **algorytm Kruskala** jest znacznie szybszy, przez to, że ta reprezentacja ma szczególnie zły wpływ na algorytm Prima.

W badaniach dotyczących wyznaczania najkrótszej ścieżki zbadano algorytmy Dijkstry i Bellmana-Forda.

Wydajniejszym okazał się **algorytm Dijkstry,** co również pokrywa się ze złożonościami obliczeniowymi. W szczególności, gdy wykorzystano listę sąsiedztwa i macierz sąsiedztwa, algorytm Dijkstry wykonywał się nieporównywalnie szybciej niż algorytm Bellmana-Forda. W przypadku użycia macierzy incydencji, nadal Dijkstra był szybszy, ale różnica nie była już tak ogromna.

Pomimo przewagi czasowej Dijkstry, trzeba pamiętać, że **algorytm Bellmana-Forda** ma znaczącą zaletę – potrafi znaleźć najkrótszą ścieżkę w grafie, w którym wagi krawędzi są ujemne. W takich grafach algorytm Dijkstry nie zadziała, więc wówczas należy stosować Bellmana-Forda.

8.2.Problemy, obserwacje, przemyślenia dotyczące pracy nad projektem

Podczas tego projektu zmierzyłem się kilkukrotnie z problemem zbyt długiego czasu wykonania. Moje pierwotne implementacje, choć były poprawne, a dla małych grafów, na których początkowo testowałem ich jakość wykonywało się szybko, to przy większych ilościach wierzchołków i krawędzi trwały bardzo długo. Szukałem wszystkich miejsc, przez które mój program drastycznie spowalniał, a następnie znajdowałem sposoby, by zmodyfikować kod, tak, by mieć możliwość przeprowadzenia badań w realnym czasie. Takie problemy spotkałem m.in. przy generowaniu grafu, tworzeniu listy wszystkich krawędzi, tworzeniu listy krawędzi z danego wierzchołka.

Po wprowadzonych poprawkach badania mogły wykonać się w optymalnym czasie (kilkunastu godzin). Użyłem skryptu, który uruchamiał po kolei badania

Problemem było też wygenerowanie grafu skierowanego. Musiał on być spójny, z każdego wierzchołka powinno się dojść do dowolnego innego. Było kilka pomysłów, takich jak stworzenie cyklu Hamiltona, ale ostatecznie zastosowałem sposób opisany wcześniej w tym dokumencie.

Problematyczna była sekcja porównania z inną implementacją na datasecie z Internetu. Bardzo trudno było znaleźć taki, który zawierałby graf ważony, spójny, nie powtarzałyby się krawędzie, a w dodatku miał odpowiednią ilość wierzchołków i gęstość.

9.Przypisy i bibliografia

9.1.Przypisy

[1] R. Sedgewick, "Algorytmy", str.530

[2] https://media.geeksforgeeks.org/wp-content/uploads/graphhh.png

[3] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 23.1, str. 526-529

[4] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 24.2, str.570-574

[5] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 7, str.173-185

[6] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 24.2, str.568-570

[7] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 22.3, str.505-508

[8] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 25.2, str.593-597

[9] T. Cormen et.al., "Wprowadzenie do algorytmów", Rozdział 25.3, str.598-600

[10] Dataset USAir97, Networkrepository: https://networkrepository.com/USAir97.php

[11] Implementacje algorytmów Prima oraz Kruskala, Serwis edukacyjny I LO w Tarnowie: https://eduinf.waw.pl/inf/alg/001_search/0141.php

9.2.Bibliografia

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Wprowadzenie do algorytmów (3. wyd.). Wydawnictwo Naukowe PWN.
- Sedgewick, R. (2011). Algorytmy (4. wyd.). Wydawnictwo Naukowe PWN.