Politechnika Wrocławska

STRUKTURY DANYCH
PROJEKT II

OSKAR REGNER 264215
Kod grupy: K00-35g

Spis treści

1 Wstęp

Sprawozdanie zawiera opis wykonanego projektu, analizę oraz wyciągnięte wnioski. Zadaniem projektowym była implementacja algorytmu Prima do wyznaczania minimalnego drzewa rozpinającego w grafie. Jako część projektu wykonano kilka wariantów implementacji oraz sporządzono ich porównanie. Implementację wykonano w języku Python, testy czasowe przeprowadzono z użyciem biblioteki time, grafy uzupełniono o losowe wagi używając biblioteki random.

2 Reprezentacja grafu

Aby móc przystąpić do implementowania algorytmu prima uprzednio zaimplementowano reprezentacje grafu. Według zaleceń projektu wykonano kilka różnych reprezentacji. Każda reprezentacja była klasą z metodami wypisania grafu jak i dodawania oraz usuwania wierzchołków i krawędzi.

2.1 Graf jako macierz sąsiedztwa

Pierwszym wybranym sposobem reprezentacji grafu była macierz sąsiedztwa. W tej implementacji indeksy macierzy reprezentują wierzchołki, jako element macierzy stworzono osobną klasę przechowującą wagę krawędzi, która różna od zera oznacza, że krawędź istnieje, jak i opcjonalne pole dla danych.
Operacje dodania, usunięcia krawędzi w tej reprezentacji działają w czasie $\mathcal{O}(1)$, a dodanie lub usunięcie wierzchołka w czasie $\mathcal{O}(v^2)$, gdzie v to liczba wierzchołków.

2.2 Graf jako lista sąsiedztwa

Drugim sposobem reprezentacji grafu była lista sąsiedztwa. W tej implementacji każdy indeks listy reprezentuje jeden wierzchołek, na pozycji tego wierzchołka, jeżeli istnieje między nim krawędź, znajduje się pierwszy element listy dwukierunkowej której elementy zawierają zapis wierzchołka pomiędzy którym istnieje krawędź oraz jej wagę.
W tej reprezentacji dodawanie nowych wierzchołków jest prostsze, jednakże operacje dodawania krawędzi, oraz sprawdzania czy dana krawędź istnieje nie działa w czasie pesymistycznym $\mathcal{O}(1)$ a w czasie $\mathcal{O}(d)$, gdzie d to długość listy sąsiedztwa. Zaletą tej metody jest prosty dostęp do sąsiadów wybranego wierzchołka.

2.3 Graf jako lista sąsiedztwa z użyciem słowników

Dla porównania graf jako listę sąsiedztwa zaimplementowano również z użyciem słowników. W tej implementacji wierzchołki reprezentowane są jako klucze, wartościami są słowniki zawierające jako klucz wierzchołek docelowy, a jako wartość wagę krawędzi.

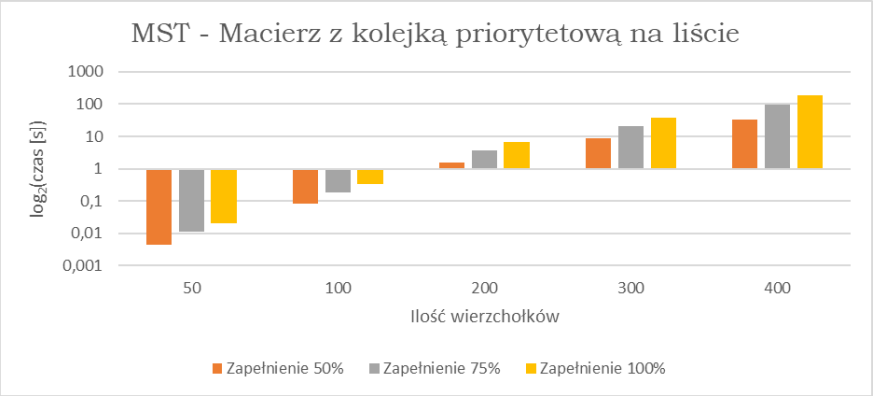
3 Implementacje algorytmu

W każdej implementacji minimalne drzewo rozpinające reprezentowane jest tak samo jak graf. Ogólny algorytm w każdej implementacji jest taki sam, jednakże różnice według zaleceń istnieją w sposobie reprezentacji grafu, wybierania krawędzi o najmniejszej wadze oraz w sposobie przetrzymywania informacji o odwiedzonych wierzchołkach. Wszystkie implementacje działają według kolejnych kroków algorytmu. Początkowo ustawiono wierzchołek startowy jako bieżący wierzchołek i oznaczono go jako odwiedzony. Następnie dla każdej krawędzi wychodzącej z bieżącego wierzchołka, jeśli końcowy wierzchołek nie jest już odwiedzony, dodano krawędź do kolejki priorytetowej wraz z jej wagą. Z kolejki priorytetowej wybrano krawędź o najniższej wadze. Jeśli końcowy wierzchołek tej krawędzi nie został jeszcze odwiedzony, ustawiono go jako bieżący wierzchołek i oznaczono go jako wierzchołek odwiedzony. Ostatecznie dodano krawędź do reprezentacji drzewa rozpinającego i powtórzono operacje wyszukiwania wszystkich krawędzi tego wierzchołka.

3.1 Implementacja I

W pierwszej implementacji zastosowano macierz sąsiedztwa i kolejkę priorytetową napisaną obiektowo jako podwójnie wiązana lista działająca na klasach. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w tablicy dynamicznej.

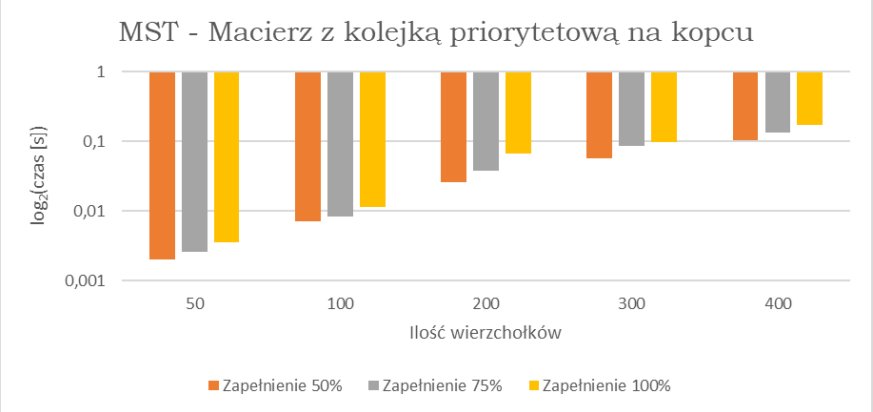
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0045	0,0110	0,0201
100	0,0804	0,1897	0,3454
200	1,5174	3,7514	6,8911
300	8,4829	20,9426	38,8058
400	33,5163	95,4486	187,2659



3.2 Implementacja II

W drugiej implementacji zastosowano macierz sąsiedztwa i kolejkę priorytetową napisaną na tablicy dynamicznej jako kopiec minimalny. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w tablicy dynamicznej.

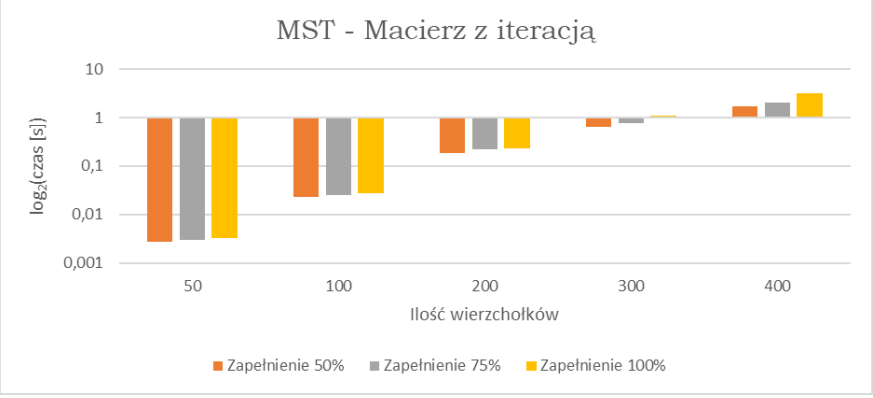
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0020	0,0026	0,0035
100	0,0070	0,0083	0,0115
200	0,0259	0,0376	0,0660
300	0,0577	0,0849	0,0984
400	0,1023	0,1337	0,1710



3.3 Implementacja III

W trzeciej implementacji zastosowano macierz sąsiedztwa, a krawędzi o najmniejszej wadze szukano iteracyjnie przechodząc po macierzy. Podejście to ogranicza tworzenie i operacje na obiektach.

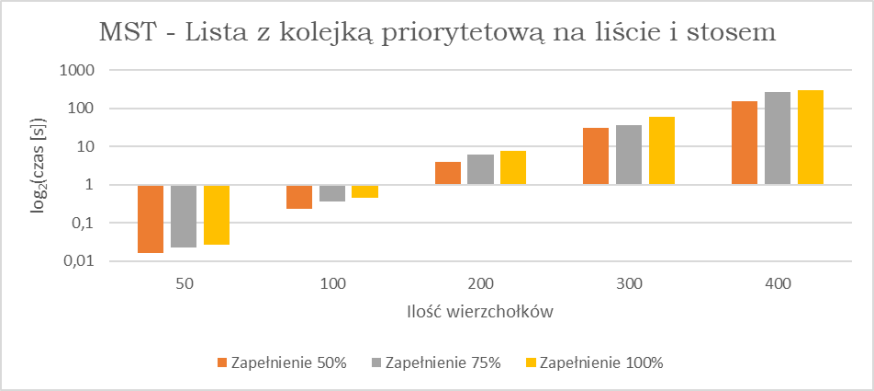
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0027	0,0030	0,0033
100	0,0227	0,0251	0,0275
200	0,1847	0,2248	0,2312
300	0,6336	0,7629	1,0708
400	1,6990	2,0280	3,1295



3.4 Implementacja IV

W czwartej implementacji zastosowano listę sąsiedztwa, kolejkę priorytetową, napisaną obiektowo jako podwójnie wiązana lista działająca na klasach, oraz stos napisany obiektowo jako sekwencja. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w stosie.

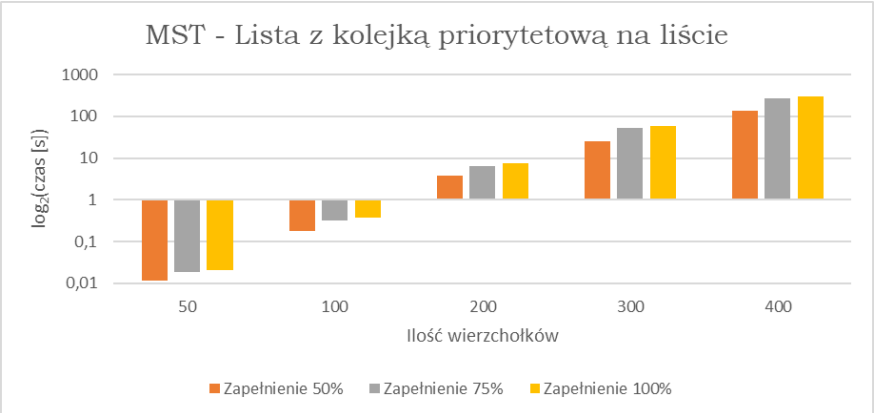
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0162	0,0226	0,0266
100	0,2289	0,3691	0,4549
200	4,0753	6,2537	7,7915
300	30,6555	35,6494	60,1502
400	157,4720	277,2923	306,1916



3.5 Implementacja V

W piątej implementacji zastosowano listę sąsiedztwa oraz kolejkę priorytetową napisaną obiektowo jako podwójnie wiązana lista działająca na klasach. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w tablicy dynamicznej.

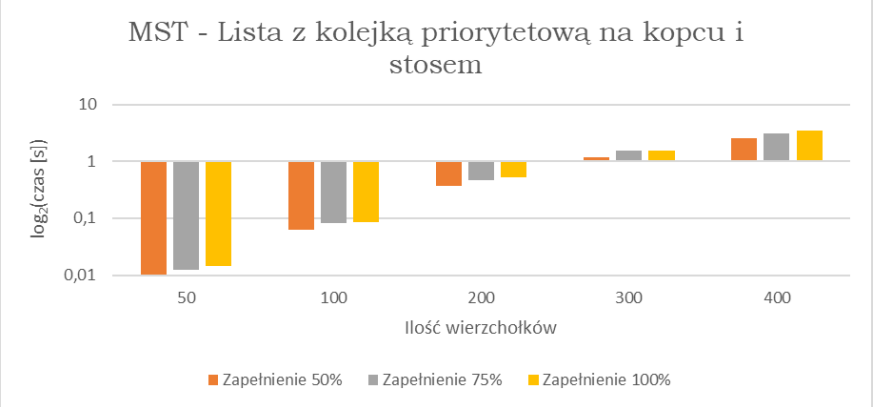
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0116	0,0186	0,0203
100	0,1840	0,3165	0,3837
200	3,8352	6,5301	7,6675
300	25,3586	53,7956	58,2540
400	136,2591	271,2884	301,4350



3.6 Implementacja VI

W szóstej implementacji zastosowano listę sąsiedztwa oraz kolejkę priorytetową, napisaną na tablicy dynamicznej jako kopiec minimalny, oraz stos napisany obiektowo jako sekwencja. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w stosie.

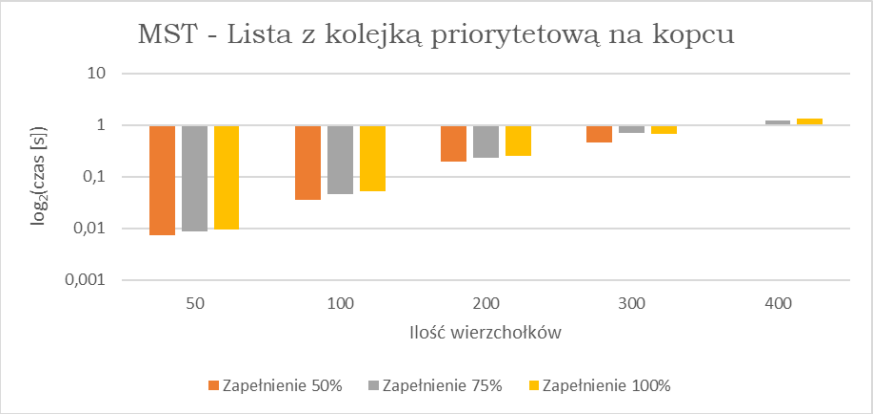
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0104	0,0125	0,0143
100	0,0617	0,0811	0,0843
200	0,3684	0,4662	0,5176
300	1,1680	1,5550	1,5492
400	2,5291	3,1334	3,4478



3.7 Implementacja VII

W siódmej implementacji zastosowano listę sąsiedztwa oraz kolejkę priorytetową, napisaną na tablicy dynamicznej jako kopiec minimalny. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w tablicy dynamicznej.

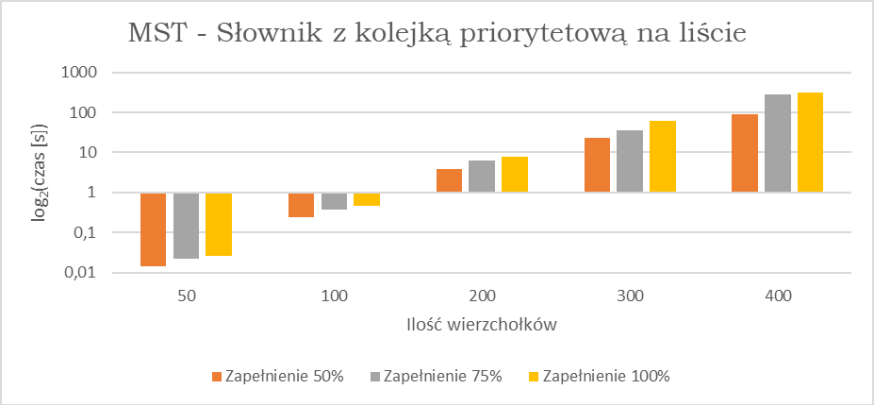
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0073	0,0089	0,0095
100	0,0364	0,0463	0,0532
200	0,2006	0,2340	0,2574
300	0,4657	0,7169	0,6714
400	0,9851	1,2108	1,3511



3.8 Implementacja VIII

W ósmej implementacji zastosowano listę sąsiedztwa zaimplementowaną z użyciem słowników, kolejkę priorytetową, napisana obiektowo jako podwójnie wiązana lista działająca na klasach, oraz stos napisany obiektowo jako sekwencja. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w stosie.

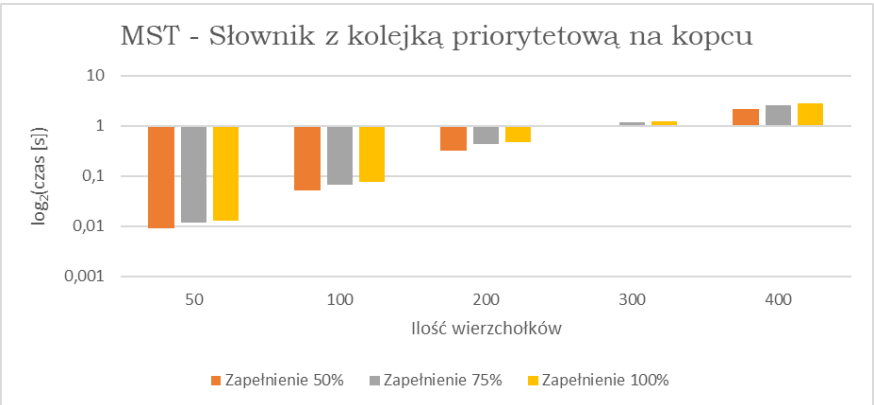
Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0146	0,0219	0,0214
100	0,2370	0,3460	0,3903
200	3,9154	6,3040	7,1767
300	22,5166	34,8152	39,6798
400	89,9847	156,3534	179,3217



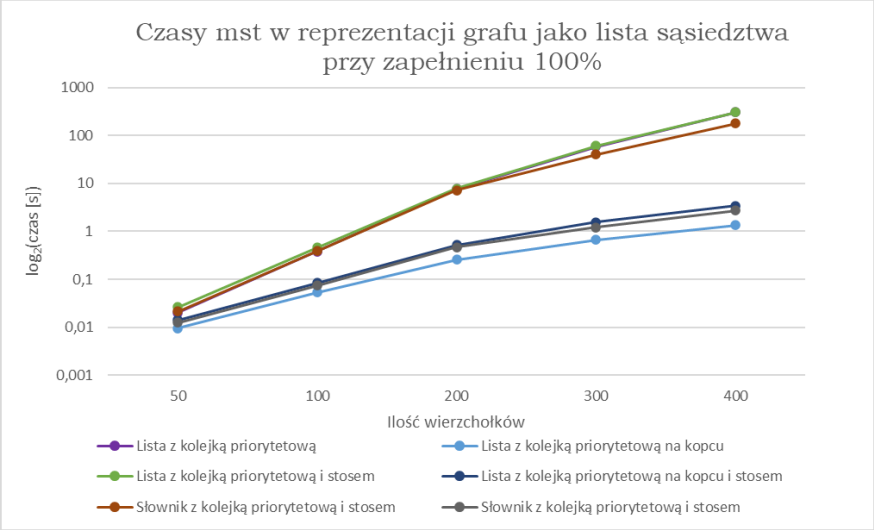
3.9 Implementacja IX

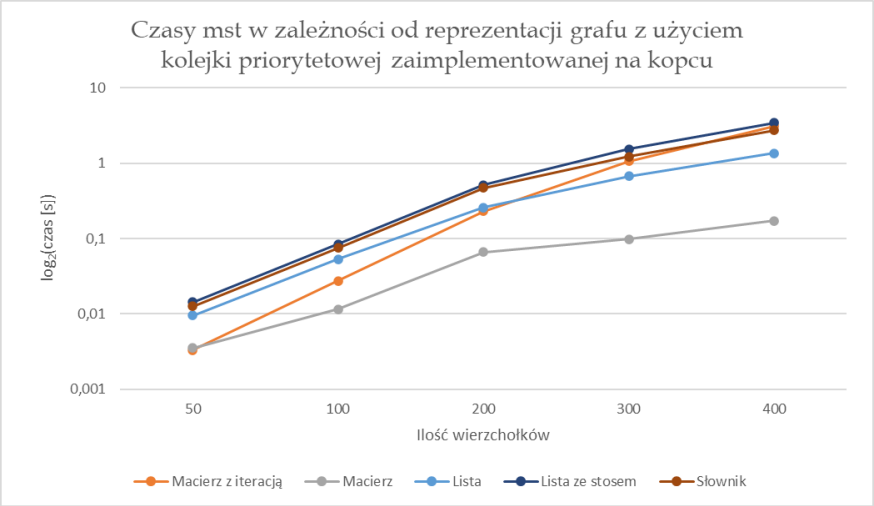
W dziewiątej implementacji zastosowano listę sąsiedztwa zaimplementowaną z użyciem słowników, kolejkę priorytetową, napisaną na tablicy dynamicznej jako kopiec minimalny, oraz stos napisany obiektowo jako sekwencja. Kolejka priorytetowa w tej implementacji służy do selekcji krawędzi której waga jest najmniejsza. Informację o odwiedzonych wierzchołkach przetrzymywano w stosie.

Ilość wierzchołków	Zapełnienie 50%	Zapełnienie 75%	Zapełnienie 100%
50	0,0090	0,0116	0,0125
100	0,0509	0,0654	0,0746
200	0,3119	0,4265	0,4685
300	0,9397	1,1534	1,2268
400	2,1715	2,5326	2,7407



4 Wyniki





5 Wnioski

Porównując ze sobą wszystkie implementacje, warianty wykorzystujące kolejkę priorytetową zaimplementowaną na kopcu są znacząco szybsze od pozostałych działających na zaimplementowanych osobno strukturach. Powodem tego jest inna implementacja kolejki priorytetowej oraz ich różne teoretyczne złożoności, przykładowo dodanie elementu do kolejki zaimplementowanej na liście posiada złożoność $\mathcal{O}(1)$, w porównaniu do złożoności $\mathcal{O}(log_2 n)$ kolejki zaimplementowanej na kopcu. Co więcej specyfika języka programowania oraz brak potrzeby tworzenia dodatkowych obiektów oraz operowania na nich, nawet jeżeli operacje takie jak zwracanie elementu z kolejki priorytetowej stworzonej na liście wiązanej ma teoretyczną złożoność $\mathcal{O}(1)$, działania na obiektach wpływają na praktyczny czas wykonania i wydłużają go bardziej niż operacje na tablicy dynamicznej w implementacji kopca.
Porównując ze sobą implementacje używające różnych reprezentacji grafu a tej samej implementacji kolejki priorytetowej zauważyć można, że implementacje z użyciem macierzy sąsiedztwa działają szybciej od implementacji wykorzystujących listy sąsiedztwa, różnica wynika ze specyfikacji języka programowania i zwiększonej liczby operacji potrzebnych do podejścia obiektowego, dlatego teoretyczne czasy działania na listach, które powinny być szybsze od czasów macierzowych zostają znacząco spowolnione.
Lista sąsiedztwa zaimplementowana z użyciem słowników jest szybsza, jednak ta implementacja nie używa nowych obiektów do przetrzymywania danych o wierzchołkach i krawędziach, jest więc mniej podatna na ewentualne przyszłościowe zmiany w porównaniu do podejścia obiektowego użytego przy liście.
Przy badaniu potwierdzono, że czas wykonania algorytmu jest zależny od wielkości danych wejściowych. Zauważyć można, że pomimo teoretycznej złożoności obliczeniowej wykonywanych kroków każdego algo-

rytmu która ograniczana jest maksymalnie przez $\mathcal{O}(V^2)$, to czasowa zależność według przeprowadzonych badań wykazała złożoność około $\mathcal{O}(V^3)$, czasy działania mogą być zawyżone przez bardziej kosztowne operacje oraz specyfikacje użytych narzędzi. Proporcjonalne różnice pomiędzy zapełnieniami grafów zgadzają się ze złożonością $\mathcal{O}(E\log(V))$ przy implementacjach z użyciem list sąsiedztwa. Porównano czasy przy pełnym zapełnieniu grafu pomiędzy implementacjami używającymi macierzy i list. Implementacje III w porównaniu do implementacji w których użyto kolejki priorytetowej na kopcu uznać można za najmniej odpowiednią do użycia przy większych danych wejściowych, ponieważ tępo wzrostu jej czasu będzie bardziej znacząco rosnąć.
Najwydatniejszą implementacją była implementacja z użyciem reprezentacji grafu jako macierz sąsiedztwa z użyciem kolejki priorytetowej zaimplementowanej na kopcu z użyciem tablicy dynamicznej, wadą tej implementacji może być potrzebna złożoność pamięciowa.