Politechnika Wrocławska

ALGORYTMY I ZŁOŻONOŚĆ OBLICZENIOWA

Sprawozdanie projektowe

ANALIZA ALGORYTMÓW GRAFOWYCH W JĘZYKU C++

Oleksandr Nedosiek 275978

Projekt, grupa №7

Wtorek TP 9:15

Prowadzący:

mgr inż. Damian Mroziński

Wrocław, 29 czerwca 2025

Spis treści

1. Cel projektu
2. Wstęp teoretyczny
3. Plan badań
4. Przebieg oraz analiza badań
5. Wnioski
6. Literatura

1. Cel projektu

Celem projektu było zaimplementowanie i przeprowadzenie analizy efektywności oraz złożoności niektórych algorytmów grafowych na podstawie dwóch reprezentacji grafu - macierz incydencji oraz lista sąsiedztwa. Badania dotyczyły problemów minimalnego drzewa rozpinającego MST (algorytmy Prima i Kruskala) oraz najkrótszej ścieżki w grafie (algorytmy Dijkstry i Bellmana-Forda).

Dodatkowymi celami było nauczenie się podstawowego skryptowania, dopasowania programu do wymagań/istniejących klas oraz mądrego planowania projektu.

Założenia projektowe zostały umieszczone przez prowadzącego w opisie projektu na e-Portalu.

2. Wstęp teoretyczny

2.1. Reprezentacja grafowa

Grafy mogą być reprezentowane na wiele sposobów, aczkolwiek dwa najczęściej stosowane to:

- Reprezentacja macierzowa (w tym macierz incydencji) – polega na użyciu dwuwymiarowej tablicy, gdzie każdy element M [i] [j] nie równy 0 oznacza istnienie (i ewentualnie wagę) krawędzi między wierzchołkami [i] oraz [j]. Ta reprezentacja jest wygodna dla grafów gęstych, ponieważ dostęp do informacji o istnieniu krawędzi jest przy górnym ograniczeniu O(1), aczkolwiek zużywa dużo O(n^2).
- Reprezentacja listowa (lista sąsiedztwa) – każdy wierzchołek przechowuje listę swoich następników. Takie rozwiązanie bardziej oszczędza pamięć i jest efektywniejsze dla grafów o małej gęstości.

2.2. Wybrane algorytmy

- Algorytm Prima (problem MST),
- Algorytm Kruskala (problem MST),
- Algorytm Dijkstry (problem najkrótszej ścieżki w grafie),
- Algorytm Bellmana-Forda (problem najkrótszej ścieżki w grafie).

Nazwa algorytmu	Lista sąsiedztwa	Macierz incydencji
Algorytm Prima	$O(E \log V)$	$O(V^2)$
Algorytm Kruskala	$O(E \log E)$	$O(V^2 \log V)$
Algorytm Dijkstry	$O(E \log V)$	$O(V^2)$
Algorytm Bellmana-Forda	$O(VE)$	$O(VE)$

Tabela 1. Złożoność obliczeniowa algorytmów dla obu reprezentacji.

2.3. Minimalne drzewo rozpinające (MST)

Celem MST jest znalezienie takiego podzbioru krawędzi w grafie, który łączy wszystkie wierzchołki, nie tworząc przy tym cykli i minimalizując sumę wag krawędzi. W danym projekcie jest ustalone, że algorytmy MST działają na grafach nieskierowanych.

2.3.1. Algorytm Prima

Algorytm polega na dodawaniu do drzewa w każdej iteracji krawędzi o najmniejszej wadze, która prowadzi do nowego, jeszcze nie odwiedzonego wierzchołka, zaczynając od jednego wierzchołka startowego. Działa efektywnie dla reprezentacji listowej, aczkolwiek jest algorytmem zachłannym, a więc daje rozwiązanie optymalne, nie zawsze najlepsze.

2.3.2. Algorytm Kruskala

Algorytm ten również jest zachłanny, lecz działa globalnie. Polega on na sortowaniu wszystkich krawędzi według wag, a następnie dodawaniu je do wyniku o ile nie tworzą cyklu.

2.4. Najkrótsza ścieżka

Celem algorytmów jest znalezienie ścieżek o najmniejszym koszcie z wybranego wierzchołka początkowego do wszystkich innych wierzchołków. Ustalono, że algorytmy dla danego problemu działają na grafach skierowanych.

2.4.1. Algorytm Dijkstry

Dany algorytm działa dla grafów o nieujemnych wagach, ponieważ nie wykrywa takich krawędzi. W każdej iteracji wybiera wierzchołek o najmniejszym aktualnym koszcie i zmienia wartość dojścia do wierzchołka o ile została znaleziona krótsza ścieżka (dokonuje relaksacji).

2.4.2. Algorytm Bellmana-Forda

Algorytm Bellmana-Forda obsługuje również grafy z ujemnymi wagami (ale nie z cyklami o ujemnej sumie wag). Działa podobnie do poprzedniego algorytmu: w każdej iteracji (V-1) wybiera wierzchołek o najmniejszym aktualnym koszcie i dokonuje relaksacji w razie potrzeby.

3. Plan badań

Aby zbadać dane algorytmu grafowe należało zaimplementować je w języku C++, gdzie badane były liczba wierzchołków, gęstość i reprezentacja grafu. Każdy eksperyment był przeprowadzony 50 razy w celu miarodajności badań, a wyniki uśrednione.

3.1. Badanie MST

Badanie polegało na porównaniu czasu działania algorytmów Prima i Kruskala na reprezentacjach macierzowej i listowej względem gęstości zbioru sortowanego i liczby wierzchołków w grafie. Liczebność zbioru wierzchołków dla każdego algorytmu była wynosiła 60, 80, 100, 120, 140, 160 oraz 180, a gęstości każdego z tych zbiorów 25%, 50% oraz 99% (prawie graf pełny).

3.2. Badanie najkrótszej ścieżki

Badanie najkrótszej ścieżki było podobne do badania MST i polegało ono na porównaniu czasu działania algorytmów Dijkstry i Bellmana-Forda na reprezentacjach macierzowej i listowej względem gęstości zbioru sortowanego i liczby wierzchołków w grafie. Liczebność zbioru wierzchołków dla każdego algorytmu była taka sama: 60, 80, 100, 120, 140, 160 oraz 180, a gęstości każdego z tych zbiorów również 25%, 50% oraz 99%.

4. Przebieg oraz analiza badań

4.1. Badanie MST

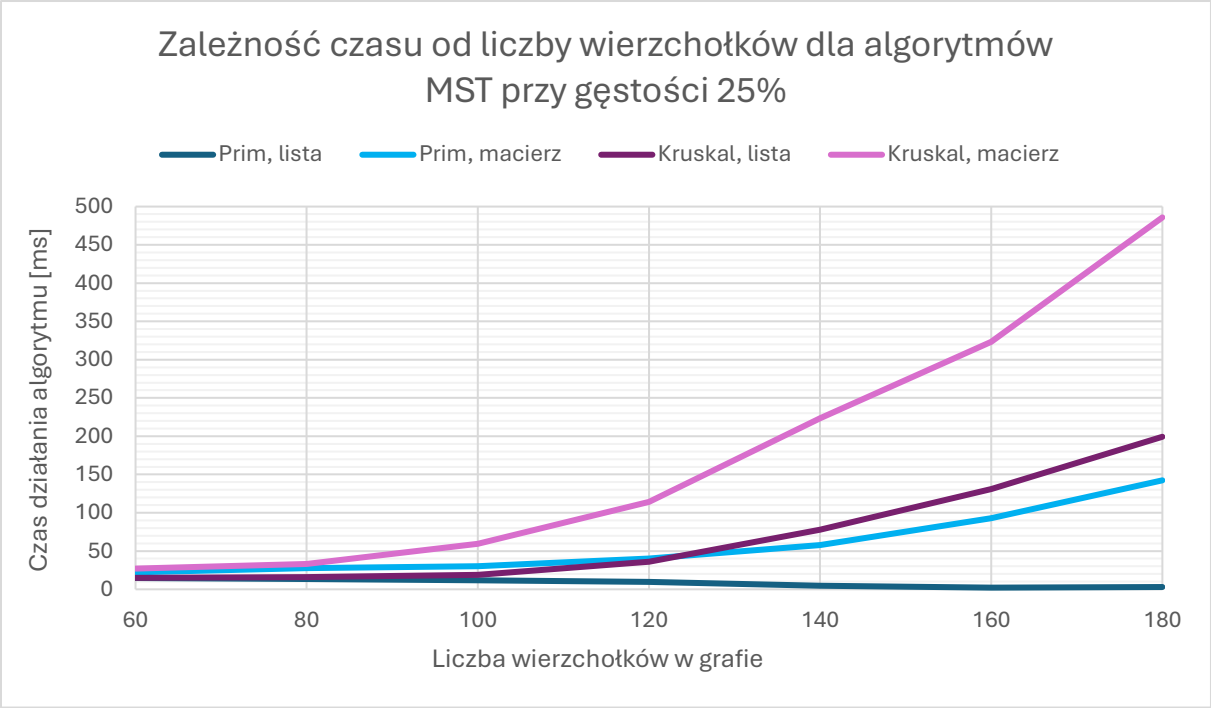
W tym rozdziale zbadamy algorytmy MST Prima i Kruskala, porównamy wpływ gęstości, liczebności zbioru wierzchołków oraz sposobu reprezentacji danego grafu. Poniżej przedstawione są tabele wyników badania, wykres (dla lepszej przejrzystości i analizy), porównujące wszystkie opcje dla danej gęstości oraz wykres, porównujący algorytmy dla wszystkich gęstości.

Czas działania algorytmu Prima, [ms]						
Gęstość, %	25		50		99	
Rozmiar\reprez.	lista	macierz	lista	macierz	lista	macierz
60	14,74	22,14	12,06	26,22	1,44	21,28
80	13,36	27,52	7,92	27,68	2,34	49,36
100	11,68	30,14	2,98	40,66	3,68	105,24
120	9,74	40,34	3,30	98,94	5,48	180,34
140	4,64	57,90	3,42	132,14	8,02	293,34
160	2,18	93,04	5,46	205,58	10,52	435,00
180	3,00	142,36	8,16	312,14	14,42	622,5

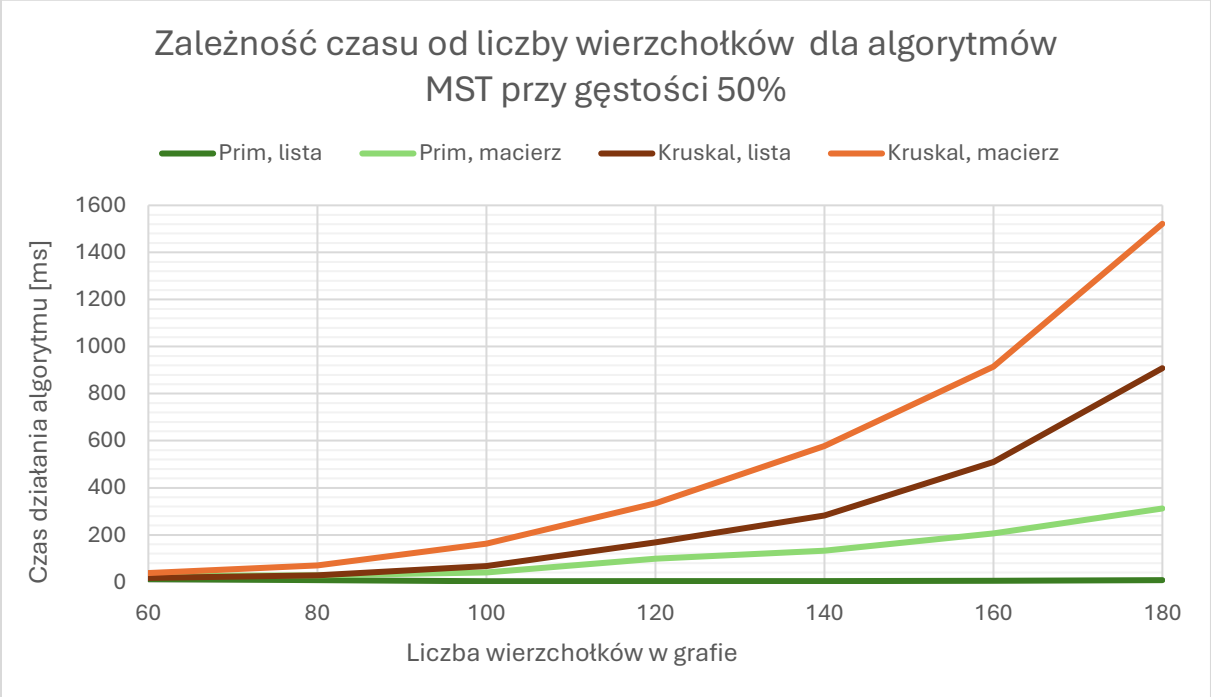
Tabela 2. Czas działania algorytmu Prima dla badanych parametrów.

Czas działania algorytmu Kruskala, [ms]						
Gęstość, %	25		50		99	
Rozmiar\reprez.	lista	macierz	lista	macierz	lista	macierz
60	14,96	27,08	17,06	37,88	33,42	70,86
80	15,86	33,22	28,56	70,32	119,94	223,20
100	18,96	59,40	67,68	163,40	289,78	522,98
120	36,08	114,22	167,98	334,70	652,86	1066,92
140	77,82	223,32	282,98	578,08	1197,68	1960,57
160	131,00	323,44	510,06	914,46	2236,76	3589,06
180	199,26	485,84	908,16	1521,62	4210,32	6326,24

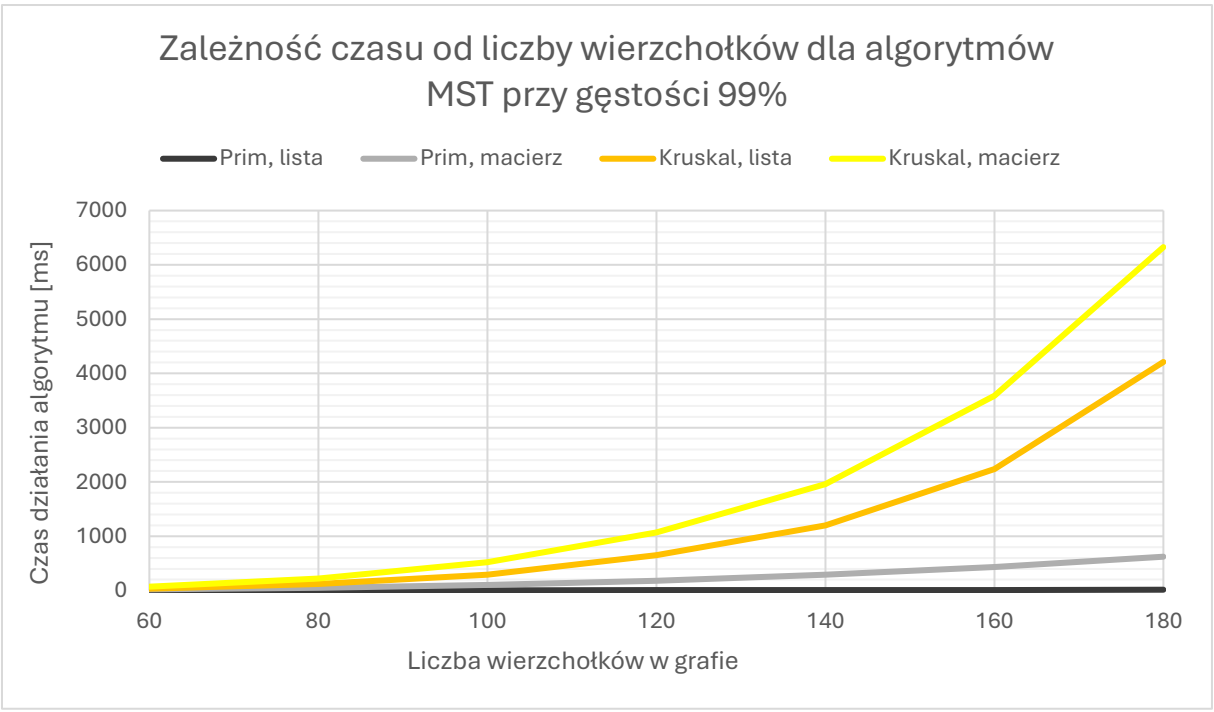
Tabela 3. Czas działania algorytmu Kruskala dla badanych parametrów.



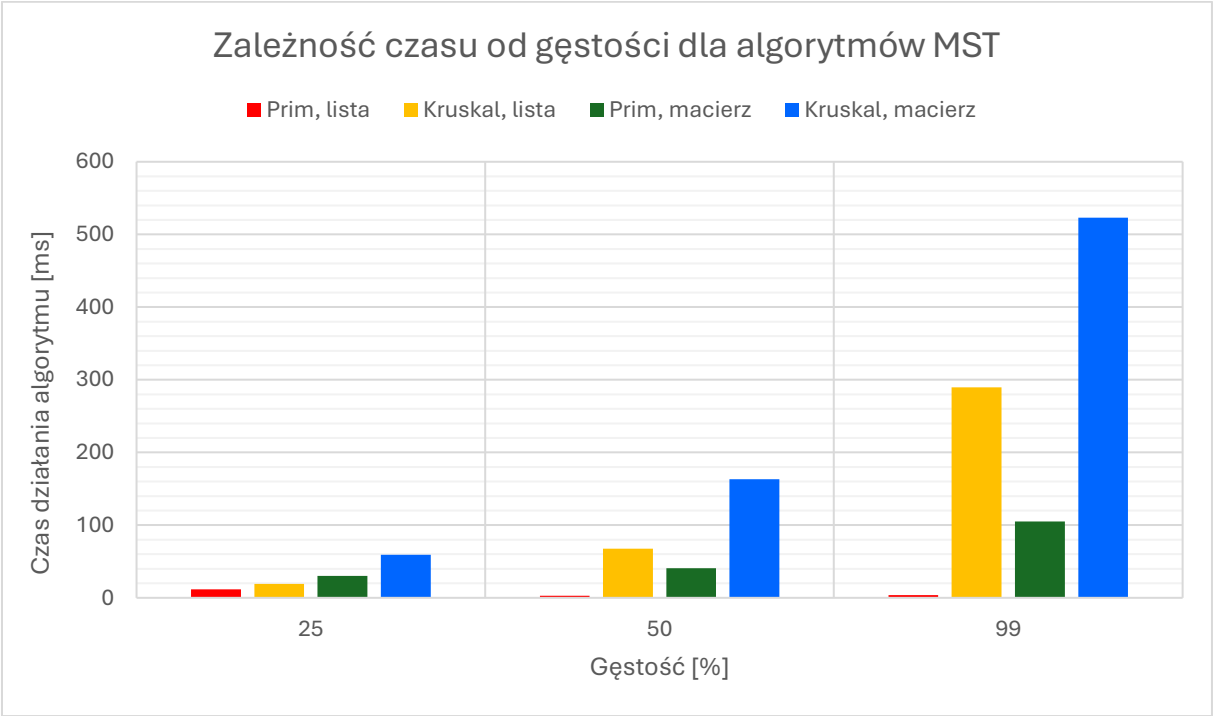
Wykres 1. Zależność czasu działania algorytmów MST Prima i Kruskala od liczby wierzchołków w grafie dla obu reprezentacji przy 25% gęstości.



Wykres 2. Zależność czasu działania algorytmów MST Prima i Kruskala od liczby wierzchołków w grafie dla obu reprezentacji przy 50% gęstości.



Wykres 3. Zależność czasu działania algorytmów MST Prima i Kruskala od liczby wierzchołków w grafie dla obu reprezentacji przy 99% gęstości.



Wykres 4. Zależność czasu działania algorytmów MST Prima i Kruskala od gęstości grafu dla obu reprezentacji przy 100 wierzchołkach.

Po dokładnej analizie powyższych tabel i wykresów możemy wynieść dużo ciekawych spostrzeżeń. Zacznijmy od algorytmu Prima, działającym na liście sąsiedztwa, gdzie dla każdej badanej gęstości wraz z wzrostem ilości wierzchołków w grafie czas działania... malał! Działo się tak dlatego, że algorytm ten ogólnie działa o wiele szybciej na reprezentacji listowej, ponieważ nie musi mieć listy wszystkich krawędzi grafu, tylko listę tych krawędzi, które wychodzą z już odwiedzonych wierzchołków, co znacznie przyśpiesza czas działania, i im więcej jest tych wierzchołków, tym łatwiej jest znaleźć krawędź o najmniejszej wadze (przy tym, że traci więcej czasu na przejście przez te wszystkie krawędzie).

Kolejnym spostrzeżeniem jest to, że dla każdej gęstości oraz dla każdej liczby wierzchołków (i też dla obu reprezentacji) w grafie czas działania algorytmu Kruskala był większy, niż w algorytmie Prima, i rósł znacznie szybciej, niż w przypadku drugiego algorytmu. Dla największej badanej liczby wierzchołków, czyli 180, na liście macierzy (na macierzy incydencji chciałem napisać) algorytm Kruskala działa w najgorszym przypadku 10 razy dłużej (99% gęstości), a na liście sąsiedztwa aż do 60 razy dłużej (25% gęstości). Wynika to z zasad działania samego algorytmu: algorytm Prima używa listy krawędzi wychodzących z odwiedzonych wierzchołków, gdyż algorytm Kruskala przechowuje listę wszystkich krawędzi grafu, i im gęstszy jest graf, tym więcej krawędzi ma do sprawdzenia. Co do tego, że na liście sąsiedztwa Kruskal działa jeszcze dłużej, niż na macierzy, odpowiedź jest prosta: algorytm Prima działa szybciej na liście sąsiedztwa (opisałem wyżej), a reprezentacja macierzowa dla grafów gęstych działa szybciej, ponieważ znalezienie krawędzi jest w czasie stałym.

Jak wspomniano wyżej o sposobie reprezentacji grafu, zmienia ona dość dużo w przypadku czasu działania algorytmów. Dla obu algorytmów macierz incydencji charakteryzuje się znacznie większym czasem działania, zwłaszcza dla większych zbiorów wierzchołków, niż lista sąsiedztwa, dlatego, że posiada ona komórki zerowe, czyli takie, które nie mieszczą żadnej informacji, bo tylko wypełniają braki w macierzy. Lista sąsiedztwa posiada tylko wierzchołki wejściowe, wyjściowe i ewentualnie wagi danych krawędzi, co znacznie przyśpiesza czas działania algorytmów. Aczkolwiek można zauważyć, że im większa jest gęstość grafu, tym mniejsza jest różnica pomiędzy czasami działania obu algorytmów. Zasada prosta: im gęstszy graf – tym mniej jest komórek pustych i tym więcej informacji niesie macierz.

4.2. Badanie najkrótszej ścieżki

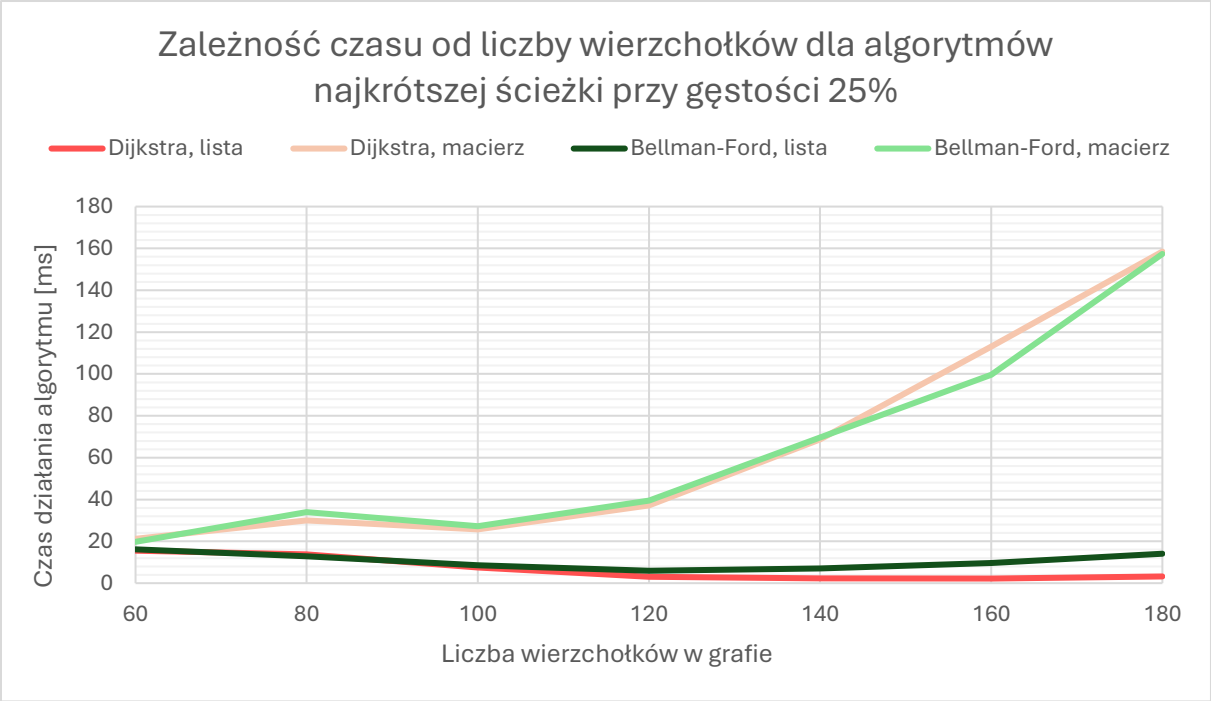
W tym rozdziale zbadamy algorytmy najkrótszej ścieżki Dijkstry i Bellmana-Forda, porównamy wpływ gęstości, liczebności zbioru wierzchołków oraz sposobu reprezentacji danego grafu. Poniżej przedstawione są tabele wyników badania, wykres, porównujący wszystkie opcje dla danej gęstości oraz wykres, porównujący algorytmy dla wszystkich gęstości.

Czas działania algorytmu Dijkstry, [ms]						
Gęstość, %	25		50		99	
Rozmiar\reprez.	lista	macierz	lista	macierz	lista	macierz
60	15,52	21,20	11,84	24,56	1,12	21,24
80	13,78	29,94	3,34	21,46	2,30	54,84
100	7,56	25,7	1,60	48,58	4,56	109,88
120	3,06	37,12	2,84	103,48	5,66	191,52
140	2,22	68,68	3,92	159,3	8,28	302,16
160	2,20	113,04	4,66	228,7	10,04	488,62
180	3,28	158,48	7,64	343,02	13,50	669,40

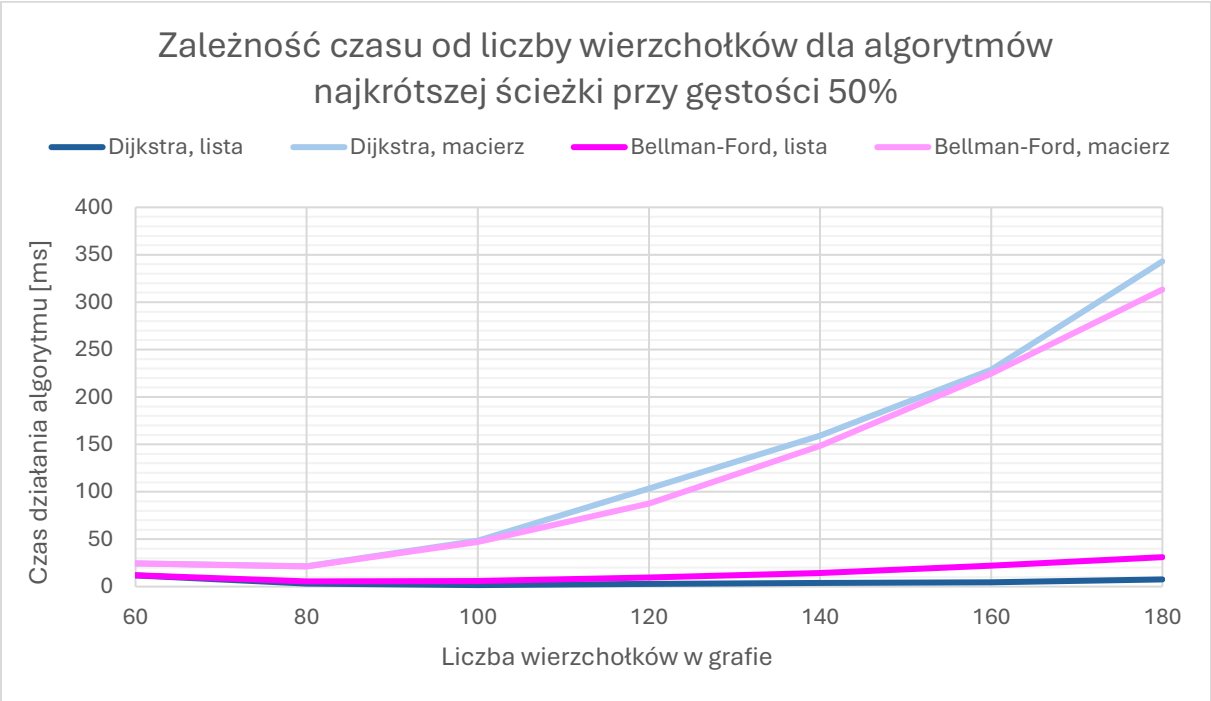
Tabela 4. Czas działania algorytmu Dijkstry dla badanych parametrów.

Czas działania algorytmu Bellmana-Forda, [ms]						
Gęstość, %	25		50		99	
Rozmiar\reprez.	lista	macierz	lista	macierz	lista	macierz
60	16,16	19,76	11,98	24,38	3,74	21,16
80	12,80	34,00	5,50	21,26	6,98	55,44
100	8,60	27,20	5,78	47,22	13,96	114,46
120	5,94	39,50	9,54	87,66	21,76	192,48
140	7,04	69,66	14,18	148,32	31,70	313,66
160	9,68	99,60	22,02	224,58	43,68	454,68
180	14,02	157,36	31,04	313,28	61,18	633,02

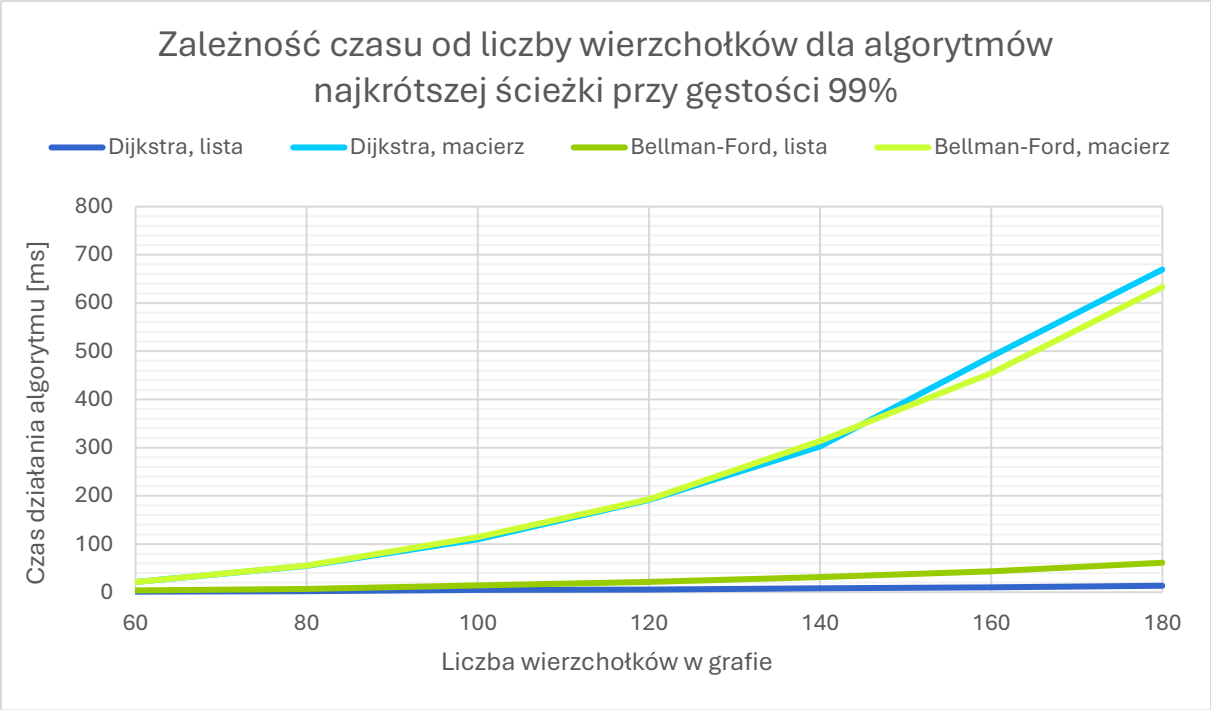
Tabela 5. Czas działania algorytmu Bellmana-Forda dla badanych parametrów.



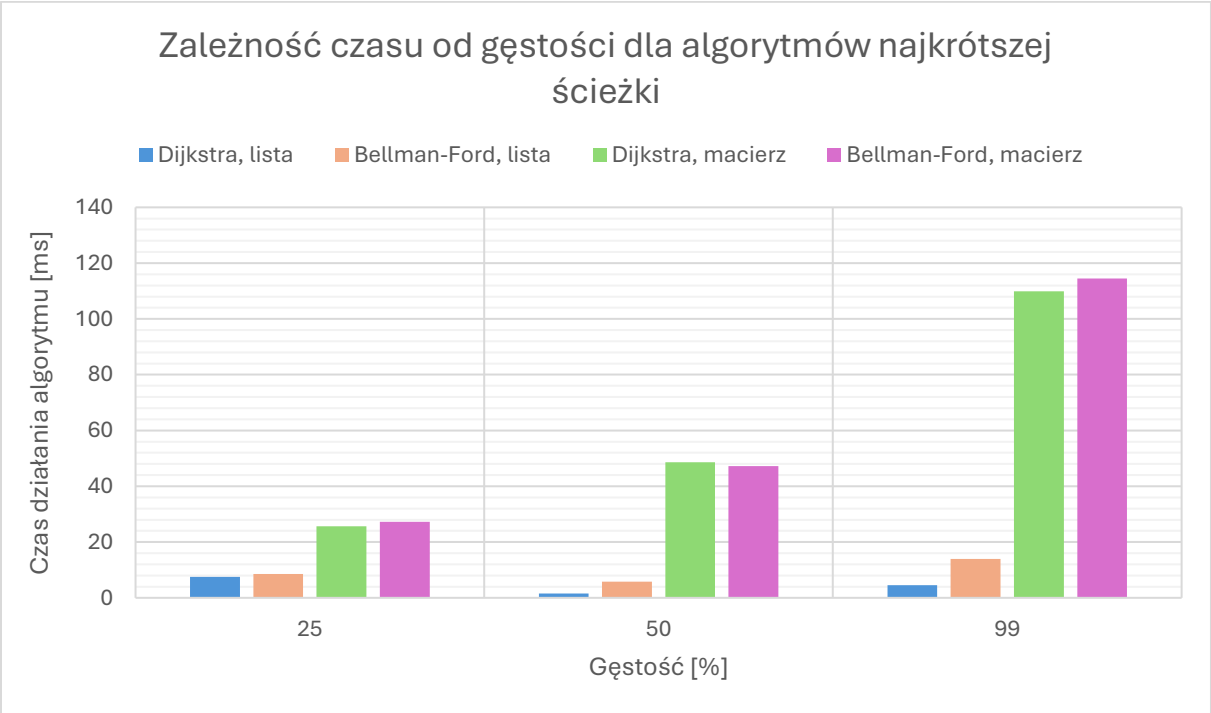
Wykres 5. Zależność czasu działania algorytmów najkrótszej ścieżki Dijkstry i Bellmana-Forda od liczby wierzchołków w grafie dla obu reprezentacji przy 25% gęstości.



Wykres 6. Zależność czasu działania algorytmów najkrótszej ścieżki Dijkstry i Bellmana-Forda od liczby wierzchołków w grafie dla obu reprezentacji przy 50% gęstości.



Wykres 7. Zależność czasu działania algorytmów najkrótszej ścieżki Dijkstry i Bellmana-Forda od liczby wierzchołków w grafie dla obu reprezentacji przy 99% gęstości.



Wykres 8. Zależność czasu działania algorytmów najkrótszej ścieżki Dijkstry i Bellmana-Forda od gęstości grafu dla obu reprezentacji przy 100 wierzchołkach.

Analizując powyższe tabele i wykresy z pewnością można stwierdzić, że przypominają one wyniki z badania MST. Podobnie, jak i w badaniu MST z algorytmem Prima na liście sąsiedztwa, tutaj czas działania algorytmu Dijkstry na liście sąsiedztwa maleje wraz z wzrostem gęstości grafu. Pewnie dlatego, że im gęstszy graf, tym łatwiej jest znaleźć lokalnie najlepszą ścieżkę, gdyż w grafach rzadkich trzeba przejść przez cały (albo większość) graf, żeby tę krawędź w ogóle znaleźć.

Kolejnym spostrzeżeniem jest to, że w porównaniu do badania MST, gdzie algorytm Kruskala był wolniejszy w każdym przypadku, w tym badaniu żaden algorytm nie jest „lepszy" od innego dla wszystkich badanych zmiennych, lecz oba algorytmy działają gorzej na macierzy incydencji i to znacznie. Im większa jest ilość wierzchołków, tym więcej czasu działa algorytm i tym szybciej ten czas rośnie. Dzieje się tak dlatego, że każdy z tych algorytmów musi przeszukać całą macierz w celu znalezienia odpowiedniej krawędzi, gdyż w przypadku listy ma ją podaną na tacy – listę wierzchołków w odpowiedniej kolejności, potrzebnej do dalszego działania algorytmu. Niestety, nie pomogło i nie uratowało macierz incydencji to, że dla gęstszych grafów znajdywanie informacji ma czas stały.

Warto jednak zaznaczyć, iż dla macierzy incydencji dla każdej gęstości grafu algorytm Dijkstry działa wolniej, niż algorytm Bellmana-Forda, co jest przeciwstawieniem dla reprezentacji listowej – Dijkstra działa szybciej, niż Bellman-Ford, i im większa jest ilość wierzchołków i im gęstszy jest graf – tym większa jest ta różnica. To też wynika z definicji algorytmów: Bellman-Ford wykonuje V-1 iteracji, przeszukując każdą krawędź grafu, gdyż Dijkstra może przeszukać też każdą krawędź grafu V-1 razy, ale w większości przypadków algorytm kończy swoje działania szybciej – kiedy kolejna iteracja nie zmieni wyniku.

5. Wnioski

Przeprowadzone badania wykazały znaczącą różnicę pomiędzy algorytmami grafowymi, gdyż potwierdziła się wiedza teoretyczna – jak na temat algorytmów, tak i na temat ich sposobu reprezentacji. Wyniki eksperymentów jasno pokazują, że wybór odpowiedniego algorytmu oraz sposobu reprezentacji ma ogromny wpływ na efektywność czasową programu, szczególnie dla większej liczby wierzchołków i gęstości grafu.

Wpośród algorytmów minimalnego drzewa rozpinającego łatwo jest stwierdzić, iż algorytm Prima jest o wiele szybszy, niż algorytm Kruskala, dla każdej liczebności zbioru wierzchołków, dla każdej gęstości i dla obu reprezentacji. Różnice w czasie działania są szczególnie zauważalne dla większych i gęstszych grafów. Warto też zaznaczyć, iż algorytm Prima działa znacznie szybciej na liście sąsiedztwa dzięki lokalności działania i mniejszemu nakładowi pracy na obsługę struktury danych.

W przeciwieństwie do problemu MST, dla problemu najkrótszej ścieżki nie ma jednoznacznego zwycięzcy. Oba algorytmy mają swoje zalety i wady, które zależą od reprezentacji i struktury grafu. Dijkstra na liście sąsiedztwa działa bardzo szybko i wydajnie – zdecydowanie lepiej, niż Bellman-Ford, szczególnie w gęstych grafach. Z kolei na macierzy incydencji różnica się zmniejsza, a w niektórych przypadkach Bellman-Ford okazuje się szybszy od Dijkstry. Dzieje się tak głównie z powodu stałego czasu dostępu do elementów w macierzy oraz zasady działania algorytmu Dijkstry. Należy również pamiętać, iż Bellman-Ford obsługuje grafy z ujemnymi wagami, co czyni go algorytmem bardziej uniwersalnym, choć wolniejszym w ogólności.

Warto wspomnieć o wpływie reprezentacji grafu. W zdecydowanej większości przypadków lista sąsiedztwa była efektywniejsza, niż macierz incydencji, zwłaszcza dla grafów rzadkich. Macierz incydencji z kolei zyskuje sens przy bardzo dużej gęstości grafu, jednakże nadal wymaga znacznie większej pamięci i czasu na przetwarzanie danych – szczególnie dla dużych grafów.

Podsumowując wszystko: dla problemu MST najbardziej opłacalnym wyborem jest algorytm Prima na liście sąsiedztwa. Dla problemu najkrótszej ścieżki najbardziej efektywnym będzie algorytm Dijkstry również na liście sąsiedztwa (warto zauważyć, iż wagi krawędzi nie mogą być ujemne). Wybór odpowiedniej reprezentacji grafu powinien być uzależniony od struktury danych (gęstość) i charakterystyki zadania, ponieważ nie istnieje jedno uniwersalne rozwiązanie. Ostatecznie, nawet takie znane algorytmy mogą znacząco się różnić się wydajnością w zależności od kontekstu ich zastosowania.

6. Literatura

- Thomas H. Cormen, Wprowadzenie do algorytmów, PWN, 2022
- https://pl.wikipedia.org/wiki/Algorytm_Bellmana-Forda
- https://www.w3schools.com/dsa/dsa_algo_graphs_bellmanford.php
- https://visualgo.net/en