Politechnika Wrocławska

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

Łukasz Wdowiak

Prowadzący: dr inż. Dariusz Banasiak
Grupa: K03-37h, Pon 15:15-16:55 TP

Wydział Informatyki i Telekomunikacji
Informatyka Techniczna
IV semestr

1 Wstęp teoretyczny

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera jest ważnym obszarem badań w dziedzinie informatyki i algorytmiki. Algorytmy grafowe są używane do rozwiązywania różnorodnych problemów związanych z grafami, takich jak wyszukiwanie najkrótszej ścieżki czy minimalne drzewo rozpinające.

Efektywność algorytmów grafowych może być mierzona na różne sposoby, ale dwa kluczowe czynniki to rozmiar instancji oraz sposób reprezentacji grafu w pamięci komputera. Rozmiar instancji odnosi się do liczby wierzchołków i gęstości grafu. Gęstością grafu nazywamy stosunek liczby krawędzi do liczby wierzchołków. Im większy graf, tym więcej operacji trzeba wykonać, co może wpływać na czas wykonania algorytmu.

Sposób reprezentacji grafu w pamięci komputera również ma duże znaczenie dla efektywności algorytmów grafowych. Istnieje wiele sposobów reprezentacji grafu, takich jak macierze incydencji czy listy sąsiedztwa. Każda z tych reprezentacji ma swoje zalety i wady pod względem szybkości działania i zajmowanej pamięci.

Algorytmy najkrótszej ścieżki (Dijkstra, Bellman-Ford) oraz algorytmy minimalnego drzewa rozpinającego (Prim, Kruskal) są ważnymi narzędziami w teorii grafów. Algorytm Dijkstry znajduje najkrótszą ścieżkę w grafie skierowanym, podczas gdy Bellman-Ford radzi sobie również z krawędziami o ujemnych wagach. Algorytmy Prima i Kruskala znajdują minimalne drzewo rozpinające w grafie nieskierowanym. Efektywność tych algorytmów może zależeć od rozmiaru grafu, sposobu reprezentacji oraz problemu, który jest rozwiązywany.

2 Reprezentacja grafu

Reprezentacja grafu to sposób przedstawienia grafu w pamięci komputera, który umożliwia skuteczne przetwarzanie i wykonywanie operacji na grafie. Istnieją różne metody reprezentacji grafu, z których dwie popularne to macierz incydencji oraz lista sąsiedztwa.

2.1 Macierz incydencji

Macierz incydencji to dwuwymiarowa tablica, w której wiersze reprezentują wierzchołki, a kolumny reprezentują krawędzie. W komórce macierzy znajduje się liczba reprezentująca wagę krawędzi lub 0 w przypadku braku kra-

wędzi między danym wierzchołkiem a krawędzią. W przypadku grafów skierowanych krawędzie są reprezentowane przez liczby ujemne i dodatnie, które oznaczają odpowiednio kierunek krawędzi. Ta reprezentacja jest skuteczna w przypadku grafów o małej gęstości, gdzie liczba krawędzi jest stosunkowo niewielka. Złożoność pamięciowa tego algorytmu wynosi $O(V * E)$, gdzie V to liczba wierzchołków, a E to liczba krawędzi. Macierz incydencji wymaga pamięci proporcjonalnej do iloczynu liczby wierzchołków i krawędzi.

2.2 Lista sąsiedztwa

Lista sąsiedztwa to struktura danych, w której każdemu wierzchołkowi przypisywana jest lista jego sąsiadów. Ta lista przechowuje informacje o wierzchołkach, do których istnieje krawędź, oraz opcjonalnie informacje o wagach krawędzi. Ta reprezentacja jest szczególnie przydatna dla grafów o dużej gęstości, gdzie liczba krawędzi może być znaczna. Złożoność pamięciowa tego algorytmu wynosi $O(V + E)$, gdzie V to liczba wierzchołków, a E to liczba krawędzi. Lista sąsiedztwa wymaga pamięci proporcjonalnej do sumy liczby wierzchołków i krawędzi.

3 Najkrótsza ścieżka

Algorytmy najkrótszej ścieżki, takie jak Dijkstra i Bellman-Ford, są używane do znajdowania najkrótszych ścieżek w grafach skierowanych lub nieskierowanych. Ich celem jest znalezienie ścieżki o najmniejszej sumie wag krawędzi pomiędzy dwoma wierzchołkami w grafie.

3.1 Algorytm Dijkstry

Algorytm Dijkstry działa w sposób iteracyjny, przechodząc przez wierzchołki grafu, zaczynając od wierzchołka źródłowego. Przypisuje początkową wartość odległości do każdego wierzchołka jako nieskończoność, a następnie aktualizuje te wartości, porównując je z wagami krawędzi. Algorytm wybiera wierzchołek o najmniejszej aktualnej odległości i relaksuje wszystkie krawędzie wychodzące z tego wierzchołka. Proces ten kontynuuje się, aż wszystkie wierzchołki zostaną odwiedzone. Algorytm posiada złożoność czasowa $O(V^2)$, gdzie V to liczba wierzchołków w grafie przy reprezentacji macierzowej. Z kolei przy reprezentacji listowej złożoność wynosi $O(V + E * log(V))$.

3.2 Algorytm Bellmana-Forda

Algorytm Bellmana-Forda również działa w sposób iteracyjny, ale może radzić sobie z krawędziami o ujemnych wagach. Algorytm rozpoczyna się od przypisania początkowych odległości dla wszystkich wierzchołków jako nieskończoność, a odległość do wierzchołka źródłowego jako 0. Następnie relaksuje krawędzie iteracyjnie, sprawdzając, czy można skrócić odległość do danego wierzchołka, poprzez wykorzystanie innych wierzchołków. Algorytm powtarza te kroki V-1 razy, gdzie V to liczba wierzchołków w grafie, aby upewnić się, że wszystkie najkrótsze ścieżki zostały znalezione. Algorytm posiada złożoność czasowa $O(V * E)$, gdzie V to liczba wierzchołków w grafie, a E to liczba krawędzi.

4 Minimalne drzewo rozpinające

Algorytmy Prim i Kruskala są wykorzystywane do znajdowania minimalnego drzewa rozpinającego w grafach nieskierowanych, czyli drzewa, które łączą wszystkie wierzchołki grafu, minimalizując sumę wag krawędzi.

4.1 Algorytm Prima

Algorytm Prima rozpoczyna się od wybranego wierzchołka i stopniowo rozbudowuje minimalne drzewo rozpinające poprzez dołączanie kolejnych krawędzi o najmniejszej wadze. Na każdym kroku wybierany jest wierzchołek, który jest najbliżej istniejącego drzewa, i dodawana jest do niego najkrótsza krawędź. Proces ten kontynuuje się, aż wszystkie wierzchołki zostaną odwiedzone i drzewo zostanie ukończone. Algorytm ten posiada złożoność czasową $O(V^2)$, gdzie V to liczba wierzchołków w grafie przy reprezentacji macierzowej. Z kolei przy reprezentacji listowej złożoność wynosi $O(V + E * log(V))$.

4.2 Algorytm Kruskala

Algorytm Kruskala działa na podobnej zasadzie, ale zamiast rozszerzać minimalne drzewo rozpinające od jednego wierzchołka, sortuje wszystkie krawędzie w kolejności rosnącej wag i stopniowo dodaje je do drzewa. Krawędzie są dodawane do drzewa, jeśli nie tworzą cyklu z już dodanymi krawędziami. Proces ten kontynuuje się, aż wszystkie wierzchołki zostaną połączone w jedno drzewo. Algorytm ten posiada złożoność czasową $O(E * log(V))$.

5 Plan projektu

5.1 Zarys projektu

- Program napisansy został w języku C++, bez wykorzystywania biblioteki STL. Projekt implementuje cztery algorytmy grafowe: Dijkstry, Bellmana-Forda, Prima i Kruskala.
- Zaimplementowane algorytmy są testowane na grafach o różnych rozmiarach i gęstościach. Waga krawędzi jest losowana z przedziału od 1 do 10.
- Wszystkie struktury danych są alokowane dynamicznie.
- Dla każdego algorytmu testowane są dwie reprezentacje grafu: macierz incydencji i lista sąsiedztwa. Macierz incydnecji jest dwuwymiarową tablicą liczb całkowitych, z kolei lista sąsiedztwa jest tablicą list.
- Wyniki testów są porównywane pod względem czasu wykonania z pomocą biblioteki chrono. Czas mierzony jest w mikrosekundach.

5.2 Generowanie grafu

Grafy generowane są w taki sposób aby były spójne. Algorytm działa w następujący sposób:

1. Tworzona jest tablica niepołączonych wierzchołków.
2. Następnie tablica jest mieszana losowo.
3. Do grafu dodajemy dwa następne wierzchołki z tablicy i łączymy je krawędzią.
4. Taką operacje wykonujemy dopóki liczba krawędzi w grafie nie będzie równa liczbie wierzchołków minus jeden.
5. W ten sposób otrzymujemy graf spójny.
6. Następnie uzupełniamy graf o pozostałe krawędzie w zależności od zadanej gęstości.

5.3 Eksperymenty

Program testuje cztery algorytmy grafowe: Dijkstry, Bellmana-Forda, Prima i Kruskala dla grafów o różnych rozmiarach i gęstościach. Eskperymenty wykonane zostały dla odpowiednich liczb wierzchołków: 50, 100, 150, 200, 250 oraz odpowiednich gęstości: 25%, 50%, 75%, 100%.

6 Wyniki eksperymentów

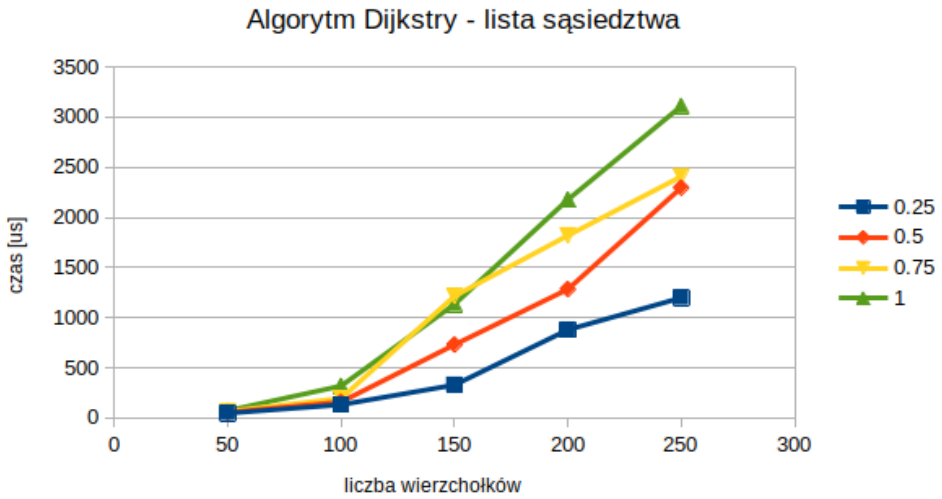
6.1 Wyszukiwanie najkrótszej ścieżki

6.1.1 Algorytm Dijkstry

6.1.1.1 Lista sąsiedztwa

	gęstość [%]			
liczba wierzchołków	25	0.5	0.75	1
	[μs]			
50	46.2	54.1	64.3	71.3
100	130.7	158.1	193.5	318
150	328	729.5	1214.7	1133.6
200	878.3	1282.4	1819	2177.2
250	1197	2295.9	1958	3109.5

Tablica 1: pomiary czasu wykonania algorytmu Dijkstry dla listy sąsiedztwa

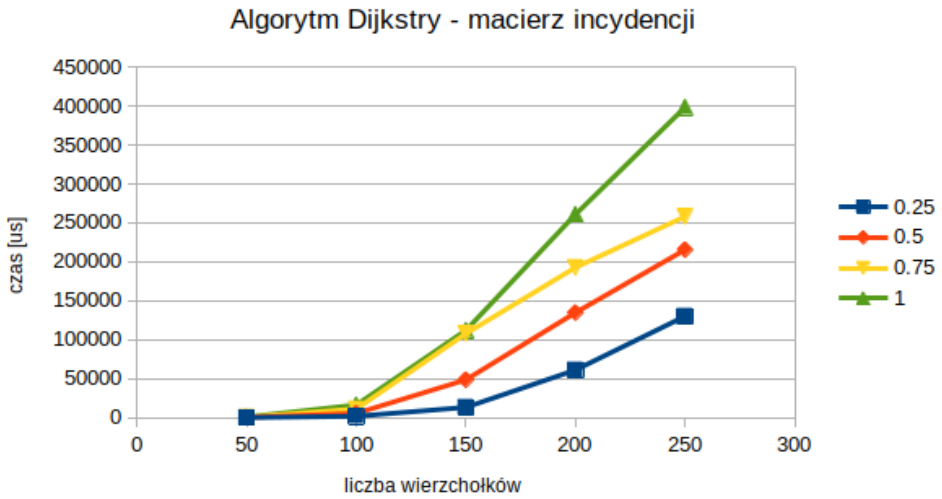


Rysunek 1: Algorytm Dijkstry dla listy sąsiedztwa

6.1.1.2 Macierz incydencji

liczba wierzchołków	gęstość [%]			
	25	0.5	0.75	1
	[μs]			
50	299.5	571	799.8	1042.2
100	2033.5	5667.6	11381.7	16495.8
150	13420	48753	108020	112149
200	61269.1	134872	192799	260884
250	130317	215626	258153	398347

Tablica 2: pomiary czasu wykonania algorytmu Dijkstry dla macierzy incydencji



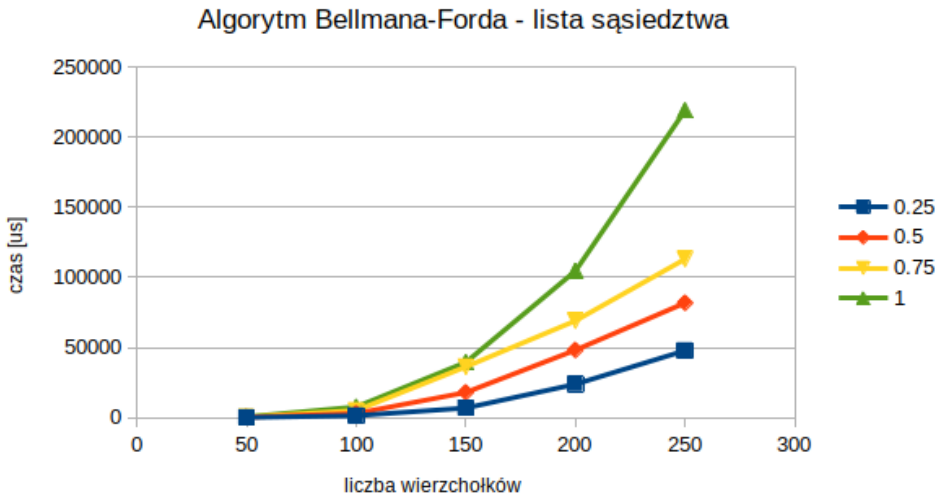
Rysunek 2: Algorytm Dijkstry dla listy sąsiedztwa

6.1.2 Algorytm Bellmana-Forda

6.1.2.1 Lista sąsiedztwa

liczba wierzchołków	gęstość [%]			
	25	0.5	0.75	1
	[μs]			
50	250.6	396.1	568.8	737
100	1563.7	3120.1	5207.9	7742.5
150	6985.3	18057.3	36367.8	39566.2
200	23917.4	48260.1	69151.2	104654
250	47897.9	81855.6	113067	219050

Tablica 3: pomiary czasu wykonania algorytmu Bellmana-Forda dla listy sąsiedztwa

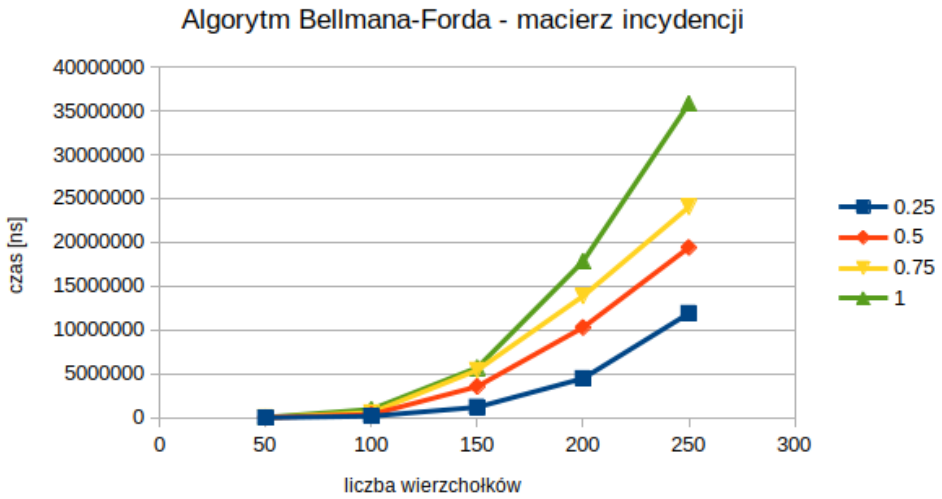


Rysunek 3: Algorytm Bellmana-Forda dla listy sąsiedztwa

6.1.2.2 Macierz incydencji

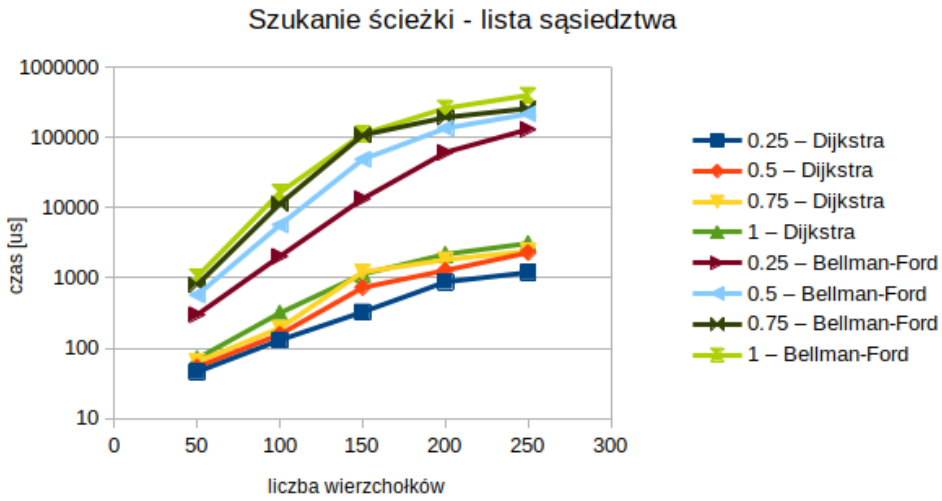
liczba wierzchołków	gęstość [%]			
	25	0.5	0.75	1
	[μs]			
50	13369.4	26014.6	38133.8	52723.9
100	187129	375414	593878	961387
150	1207240	3562990	5432010	5714210
200	4485500	10295800	13878700	17832100
250	11906300	19413200	24039800	35827200

Tablica 4: pomiary czasu wykonania algorytmu Bellmana-Forda dla macierzy incydencji

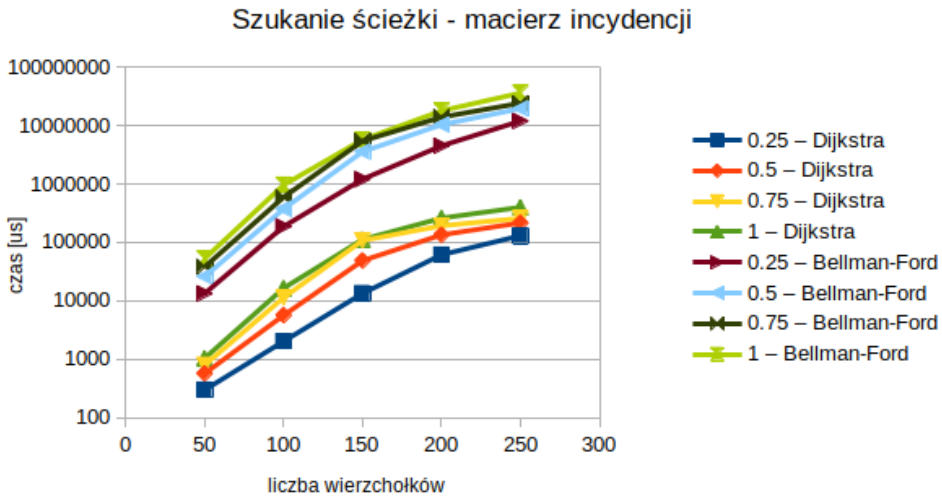


Rysunek 4: Algorytm Bellmana-Forda dla macierzy incydencji

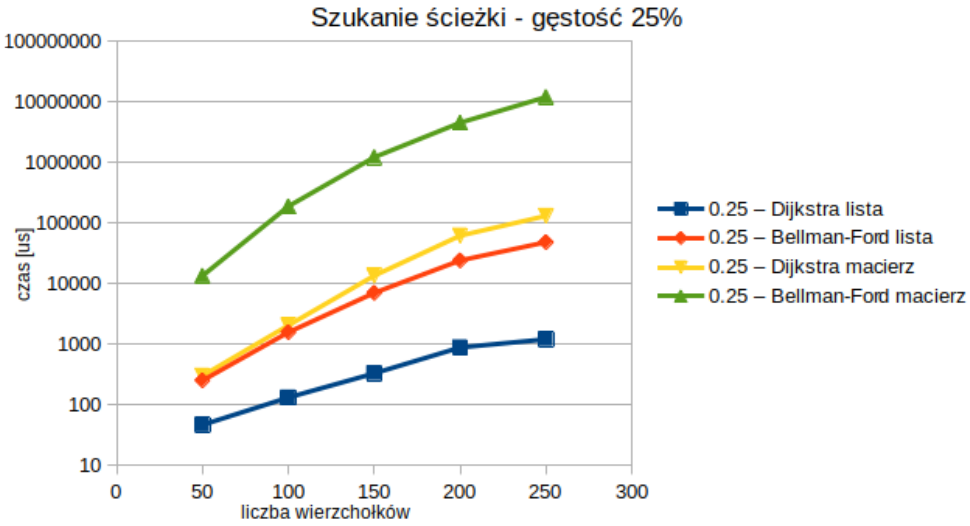
6.1.2.3 Porównanie algorytmów



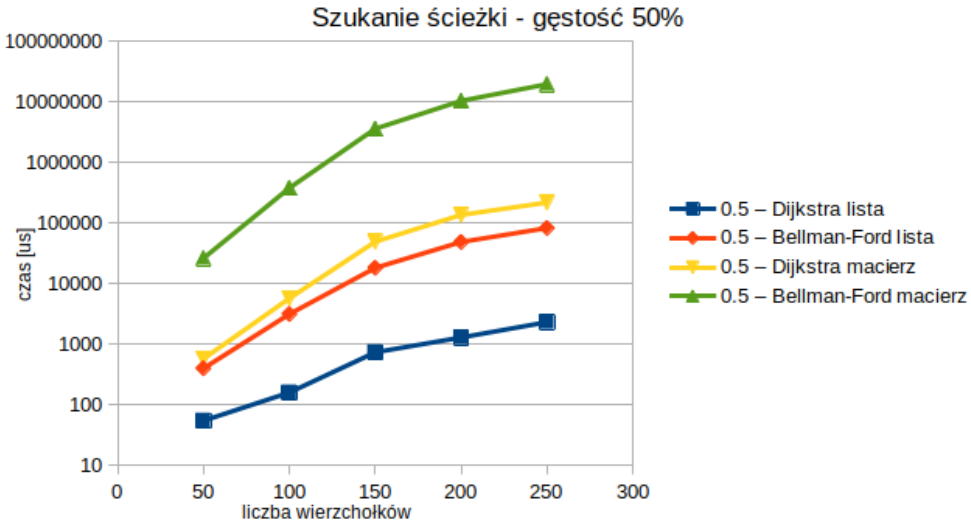
Rysunek 5: Porównanie algorytmów Dijkstry i Bellmana-Forda dla listy sąsiedztwa



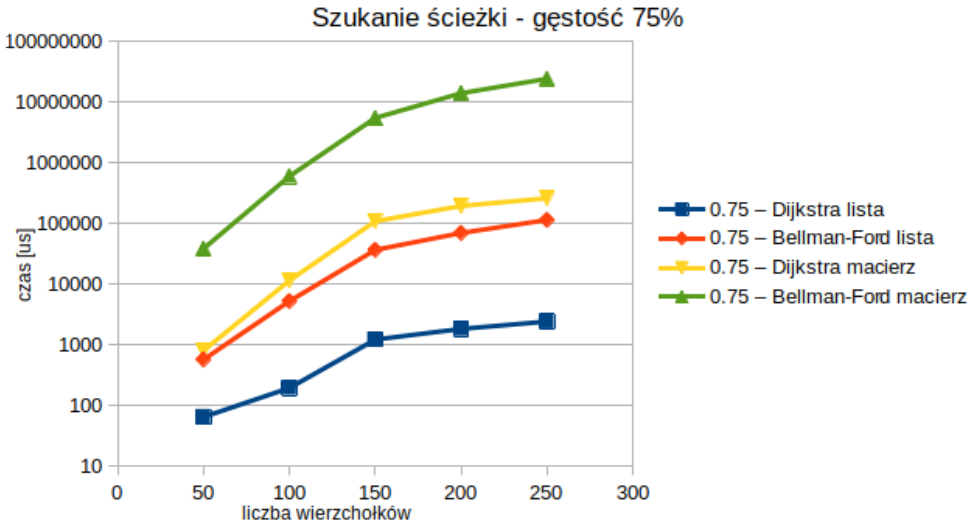
Rysunek 6: Porównanie algorytmów Dijkstry i Bellmana-Forda dla macierzy incydencji



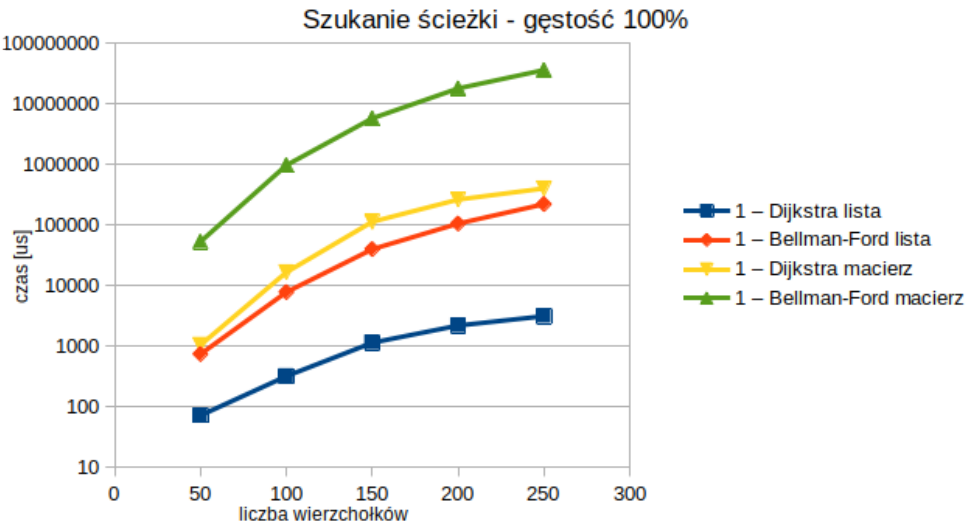
Rysunek 7: Porównanie algorytmów Dijkstry i Bellmana-Forda dla gęstości grafu 25%



Rysunek 8: Porównanie algorytmów Dijkstry i Bellmana-Forda dla gęstości grafu 50%



Rysunek 9: Porównanie algorytmów Dijkstry i Bellmana-Forda dla gęstości grafu 75%



Rysunek 10: Porównanie algorytmów Dijkstry i Bellmana-Forda dla gęstości grafu 100%

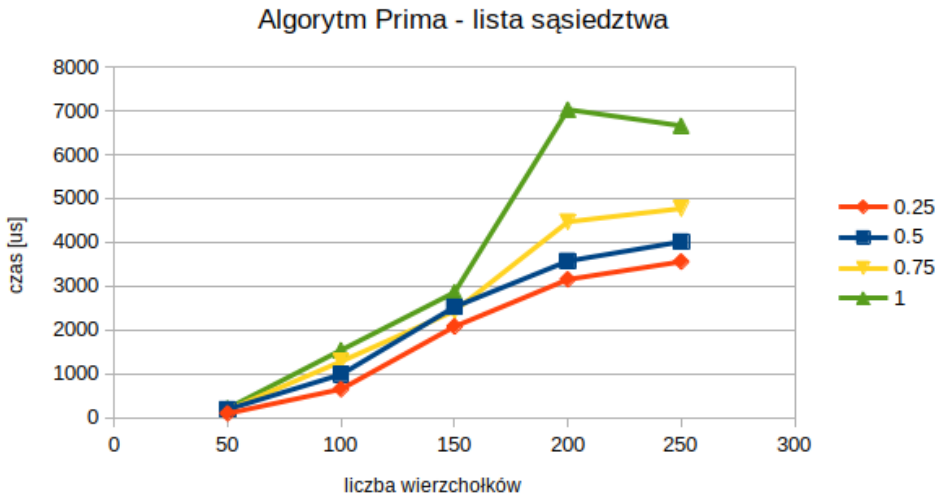
6.2 Wyznaczanie minimalnego drzewa rozpinającego

6.2.1 Algorytm Prima

6.2.1.1 Lista sąsiedztwa

	gęstość [%]			
liczba wierzchołków	25	0.5	0.75	1
	[μs]			
50	98.8	190.5	179.4	214.7
100	653	988.2	1284.3	1542.2
150	2080.3	2528	2435.1	2860.5
200	3154.5	3574.2	4469.9	7028.4
250	3558.9	4014.8	4771.8	6659

Tablica 5: pomiary czasu wykonania algorytmu Prima dla listy sąsiedztwa

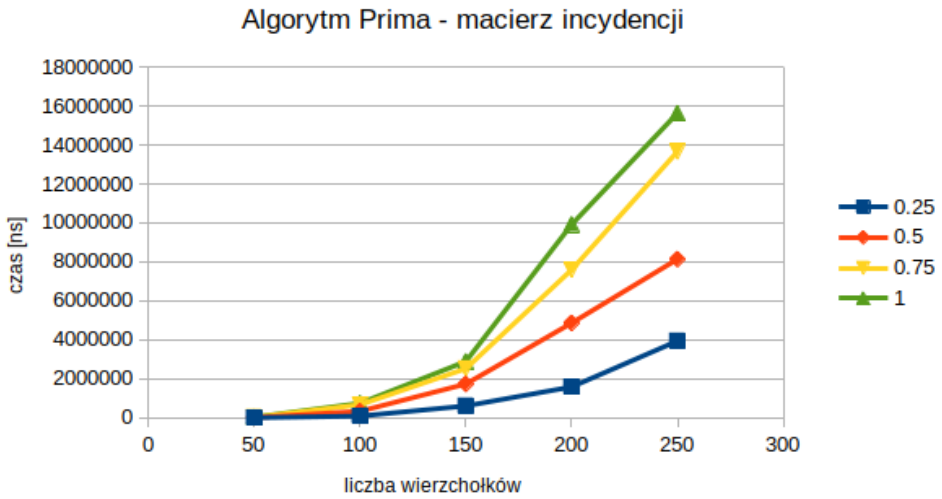


Rysunek 11: Algorytm Prima dla listy sąsiedztwa

6.2.1.2 Macierz incydencji

	gęstość [%]			
liczba wierzchołków	25	0.5	0.75	1
	[μs]			
50	4381.4	15213.3	26397.5	33037.7
100	90108.1	333725	672840	747975
150	607973	1731790	2523640	2891450
200	1585600	4863460	7595420	9919720
250	3943070	8135320	13663200	15632200

Tablica 6: pomiary czasu dla wykonania algorytmu Prima dla macierzy incydencji



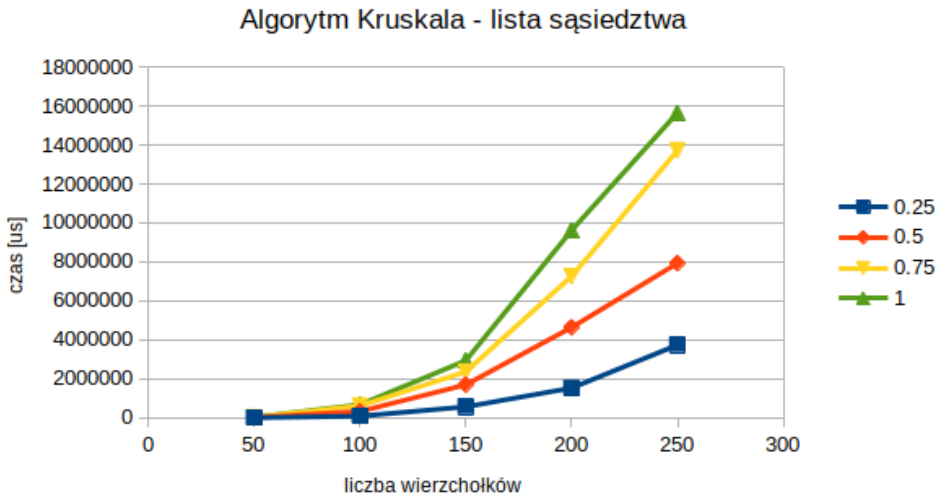
Rysunek 12: Algorytm Prima dla macierzy incydencji

6.2.2 Algorytm Kruskala

6.2.2.1 Lista sąsiedztwa

	gęstość [%]			
liczba wierzchołków	25	0.5	0.75	1
	[μs]			
50	4565.2	14198.3	25529.2	32219.9
100	87859.4	328322	617719	675654
150	565339	1696130	2362380	2939720
200	1528330	4639590	7264690	9622900
250	3733480	7927100	13720300	15649900

Tablica 7: pomiary czasu dla wykonania algorytmu Kruskala dla listy sąsiedztwa

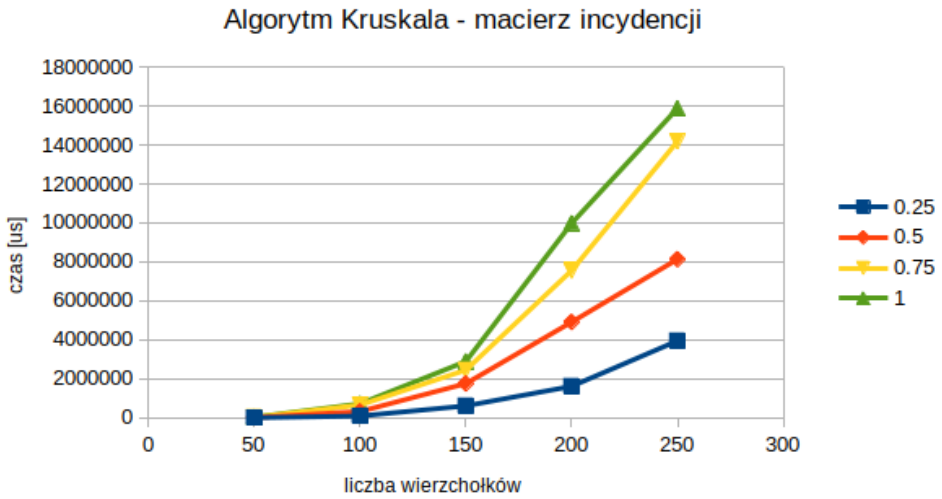


Rysunek 13: Algorytm Prima dla macierzy incydencji

6.2.2.2 Macierz incydencji

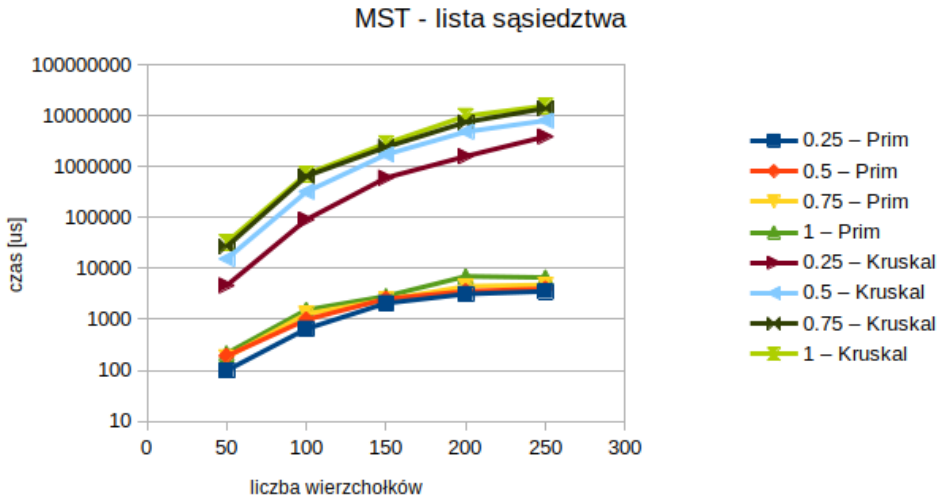
	gęstość [%]			
liczba wierzchołków	25	0.5	0.75	1
	[μs]			
50	4644	15416.8	27019.4	32967
100	91866	323097	658338	723943
150	611020	1753970	2451020	2892360
200	1617660	4913760	7556850	9982490
250	3949320	8132030	14187800	15877500

Tablica 8: pomiary czasu dla wykonania algorytmu kruskala dla macierzy incydencji

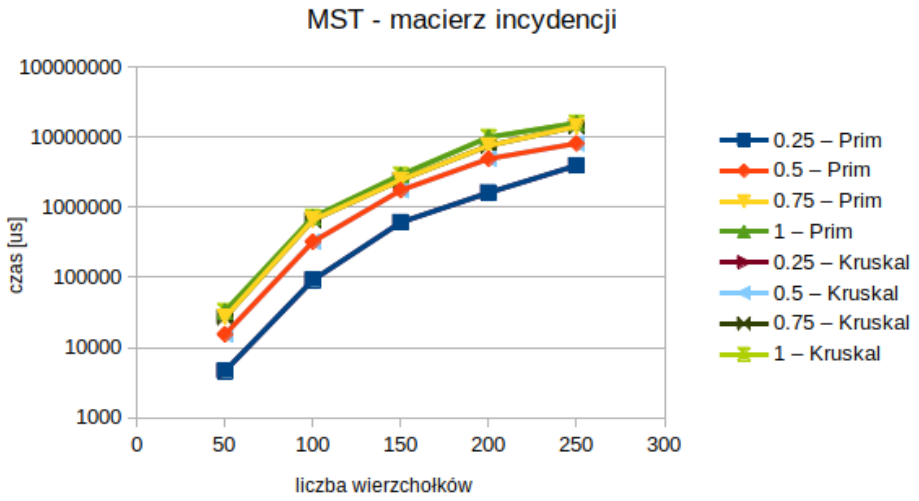


Rysunek 14: Algorytm Prima dla macierzy incydencji

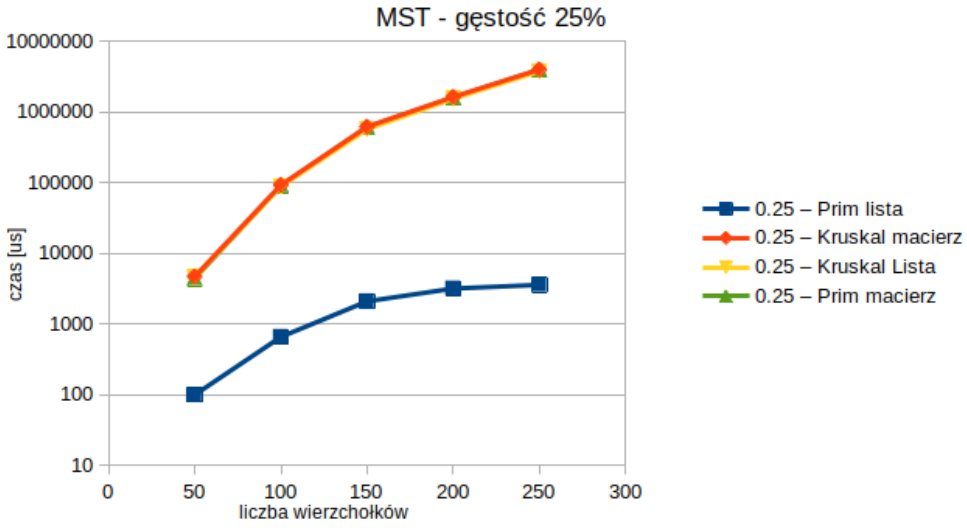
6.2.2.3 Porównanie algorytmów



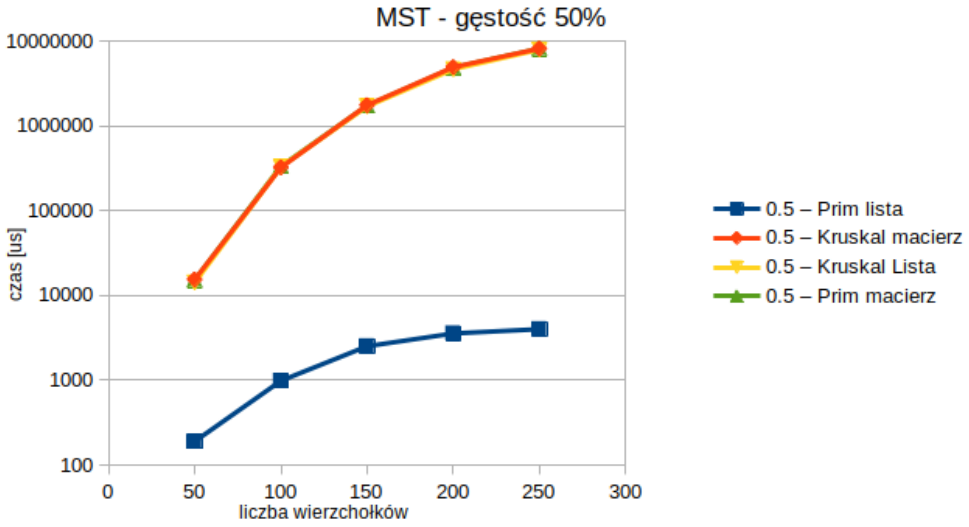
Rysunek 15: Porównanie algorytmów Prima i Kruskala dla listy sąsiedztwa



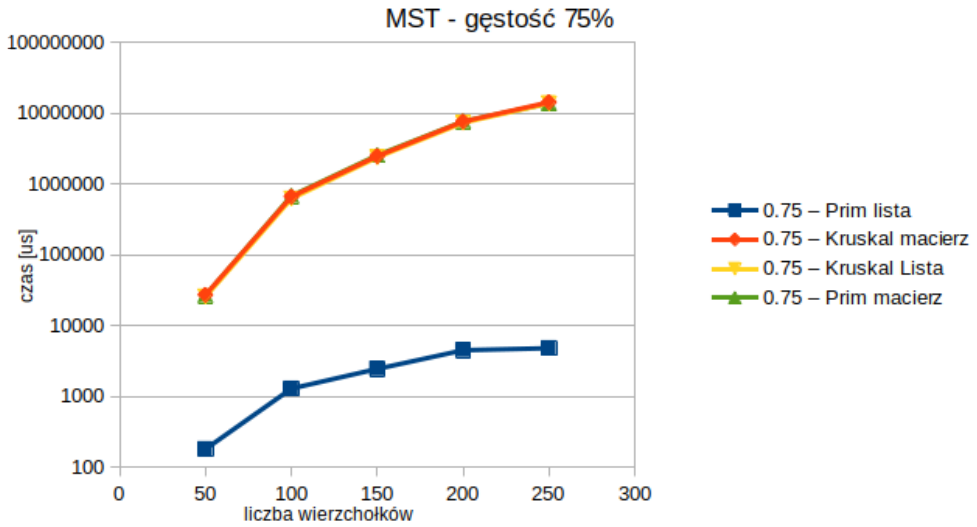
Rysunek 16: Porównanie algorytmów Prima i Kruskala dla macierzy incydencji



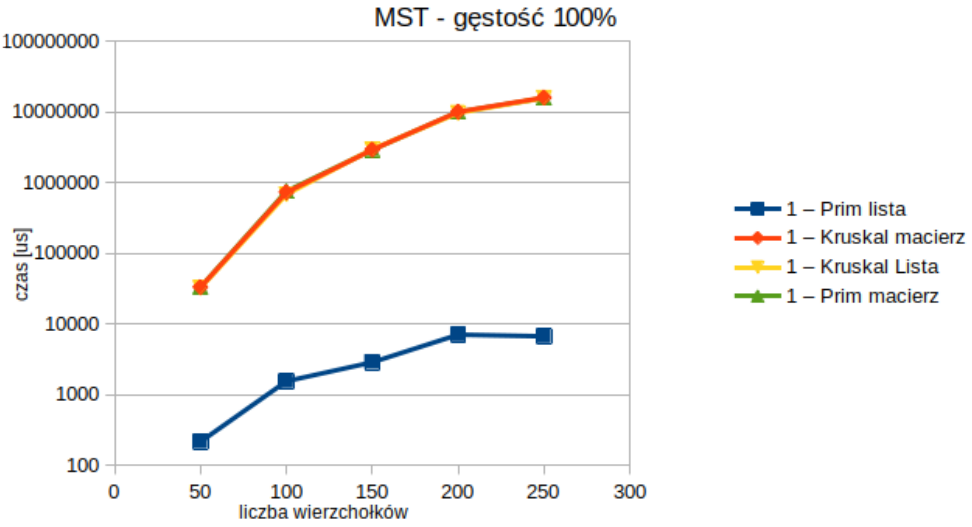
Rysunek 17: Porównanie algorytmów Prima i Kruskala dla gęstości grafu 25%



Rysunek 18: Porównanie algorytmów Prima i Kruskala dla gęstości grafu 50%



Rysunek 19: Porównanie algorytmów Prima i Kruskala dla gęstości grafu 75%



Rysunek 20: Porównanie algorytmów Prima i Kruskala dla gęstości grafu 100%

7 Wnioski

Dla algorytmów wyszukiwania najkrótszej ścieżki, algorytm Dijkstry jest znacznie szybszy od algorytmu Bellmana-Forda. Z kolei dla algorytmów wyszukiwania minimalnego drzewa rozpinającego, algorytm Prima jest znacznie szybszy od algorytmu Kruskala. Rozróżniając algorytmy po reprezentacji grafu, algorytmy dla listy sąsiedztwa są znacznie szybsze od algorytmów dla macierzy incydencji. Dzieję się tak, ponieważ dla macierzy incydencji algorytm musi najpierw znaleźć odpowiednie krawędzie, przeszukując praktycznie całą macierz. Z kolei dla listy sąsiedztwa, algorytm musi jedynie przejść przez listę sąsiedztwa danego wierzchołka.

Złoności obliczeniowe algorytmów różnią się od tych teoretycznych, ponieważ w programie wykorzystywane są struktury danych, które nie są optymalne (przykładem może być tutaj struktura implementująca tablicę dynamiczną, gdzie zajmuje ona możliwe najmniej miejsca). Oraz przy implementacji algorytmów nie zawsze wykorzystywane są optymalne rozwiązania. Takie jak korzystanie z kolejki piorytetowej, czy kopca minimanlnego, które są optymalne dla poszczególnych algorytmów.

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ron Rivest, Clifford Stein (2022) *Introduction to algorithms 4th edition*, MIT Press and McGraw-Hill.

[2] Prezentacje dr inż. Jarosława Mierzwy