STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

BADANIE EFEKTYWNOŚCI ALGORYTMÓW GRAFOWYCH W ZALEŻNOŚCI OD ROZMIARU INSTANCJI ORAZ SPOSOBU REPREZENTACJI GRAFU W PAMIĘCI KOMPUTERA

PROWADZĄCY:
DR INŻ. DARIUSZ BANASIAK

AUTOR:
BARTOSZ RUDNIK 248893

TERMIN ZAJĘĆ:
WT 15:15 TN

WROCŁAW, 02.06.2020

Spis treści

1. Cel ćwiczenia

W ramach wykonanego ćwiczenia należało zaimplementować algorytmy grafowe rozwiązujące problemy:

a) wyznaczenia minimalnego drzewa rozpinającego – algorytm Prima oraz algorytm Kruskala.
b) wyznaczenia najkrótszej ścieżki w grafie – algorytm Dijkstry oraz algorytm Bellmana-Forda.
c) wyznaczenia maksymalnego przepływu w grafie – algorytm Forda-Fulkersona.

Kolejnym etapem ćwiczenia było dokonanie pomiarów czasu potrzebnego na wykonanie powyższych algorytmów w zależności od sposobu ich zapisu w pamięci komputera, liczby wierzchołków grafu oraz gęstości grafu. Badania miały zostać przeprowadzone dla dwóch sposobów reprezentacji grafu w pamięci komputera:

1) Macierz incydencji – jest to macierz o wymiarach V x E, gdzie V oznacza liczbę wierzchołków w grafie, a E oznacza liczbę krawędzi w grafie. Każdy wiersz macierzy incydencji obrazuje jeden z wierzchołków grafu, a każda kolumna odwzorowuje jedną z krawędzi grafu.

2) Lista sąsiedztwa – jest to lista, w której każdy element listy odpowiada jednemu z wierzchołków grafu oraz zawiera dodatkową tablicę, w której zapamiętywane są wszystkie wierzchołki grafu, które sąsiaduje z wierzchołkiem odpowiadającym temu elementowi listy.

2. Założenia ćwiczenia

Językiem programowania użytym do implementacji algorytmów i interfejsu testowego oraz pomiarowego jest JAVA. Wagi krawędzi w grafach wyrażane są za pomocą liczb całkowitych. W celu uzyskania lepszej dokładności mierzonych wartości, dla każdego algorytmu oraz dla każdego zestawu danych dla tego algorytmu zostało wykonanych 400 pomiarów, uzyskane wyniki zostały następnie uśrednione i są przedstawione w dalszej części sprawozdania. Wszystkie zaimplementowane algorytmy zostały przetestowane dla każdej kombinacji liczb wierzchołków: 10, 50, 250, 500, 1000 i gęstości grafu: 25%, 50%, 75%, 99%. Do przeprowadzenia pomiarów została użyta metoda java.lang.System.nanoTime(). Wszystkie wyniki pomiarów czasu umieszczone w sprawozdaniu zostały podane w mikrosekundach.

3. Wstęp teoretyczny

3.1. Problem wyznaczenia minimalnego drzewa rozpinającego

Minimalne drzewo rozpinające danego grafu jest to takie drzewo, że nie istnieje dla tego grafu inne drzewo rozpinające, którego suma krawędzi jest mniejsza. Drzewo rozpinające grafu zawierza wszystkie wierzchołki grafu oraz część jego krawędzi. W grafie może znajdować się więcej niż jedno minimalne drzewo rozpinające, rozwiązaniem problemu wyznaczenia minimalnego drzewa rozpinającego w grafie jest wskazanie jednego z nich.

3.1.1. Algorytm Prima

Jest to zachłanny algorytm wyznaczający minimalne drzewo rozpinające w grafie. Algorytm Prima działa dla spójnych grafów skierowanych. Algorytm Prima został najpierw wynaleziony w 1930 roku przez Czecha Vojtecha Jarnika, a następnie w 1959 został niezależnie odkryty przez Roberta Prima. W mojej implementacji algorytmu Prima użyłem kolejki priorytetowej zbudowanej na podstawie kopca typu minimum. W takim przypadku Algorytm Prima powinien mieć złożoność obliczeniową **O(|E| * log|V|)**, gdzie |E| jest liczbą krawędzi w grafie, a |V| liczbą wierzchołków w grafie.

3.1.2. Algorytm Kruskala

Jest to zachłanny algorytm wyznaczający minimalne drzewo rozpinające w grafie. Algorytm Kruskala działa dla grafów, które są ważone, spójne i nieskierowane. Algorytm Kruskala został wynaleziony w 1956 roku przez Josepha Kruskala. W mojej implementacji algorytmu Kruskala zastosowałem kolejkę priorytetową, zbudowaną na podstawie kopca typu minimum. W takiej implementacji algorytm Kruskala powinien przyjąć złożoność obliczeniową **O(|E| * log|V|)**, gdzie |E| jest liczbą krawędzi w grafie, a |V| liczbą wierzchołków w grafie.

3.2. Problem wyznaczenia najkrótszej ścieżki w grafie

Jest to problem grafowy polegający na wyznaczeniu w grafie ważonym ścieżki o najmniejszej wadze pomiędzy wybranymi wierzchołkami grafu. W pesymistycznym przypadku będziemy zmuszeni wyznaczyć najkrótsze ścieżki od wierzchołka źródłowego do wszystkich pozostałych wierzchołków w grafie.

3.2.1. Algorytm Dijkstry

Jest to algorytm zachłanny służący do wyznaczanie najkrótszej ścieżki w grafie. Algorytm Dijkstry wyznacza najkrótszą ścieżkę w grafie skierowanym dla pojedynczego wierzchołka źródłowego, dla grafów o nieujemnych wagach krawędzi. Algorytm Dijkstry został opracowany przez holenderskiego informatyka Edsgera Dijkstrę. W swojej implementacji algorytm Dijkstry przypomina algorytm Prima. Do zaimplementowania kolejki priorytetowej dla algorytmu Dijkstry użyłem kopca typu minimum. Dla implementacji tego typu, złożoność obliczeniowa algorytmu Dijkstry powinna wynieść **O(|E| * log|V|)**, gdzie |E| jest liczbą krawędzi w grafie, a |V| liczbą wierzchołków w grafie.

3.2.2. Algorytm Bellmana-Forda

Jest to algorytm służący do rozwiązywania problemu wyznaczenia najkrótszej ścieżki w grafie. Algorytm Bellmana-Forda działa zarówno dla grafów skierowanych o ujemnych jak i nieujemnych wagach krawędzi, dzięki czemu jest bardziej uniwersalny w porównaniu do algorytmu Dijkstry, który też zajmuje się problemem wyznaczenia najkrótszej ścieżki w grafie. Warunkiem poprawnego działania algorytmu Bellmana-Forda jest brak wystąpienia cyklu osiąganego z wierzchołka źródłowego o łącznej ujemnej wadze. Algorytm Bellmana-Forda działa w czasie **O(|V| * |E|)**, gdzie |V| jest liczbą wierzchołków w grafie, a |E| jest liczbą krawędzi w grafie.

3.3. Problem wyznaczenia maksymalnego przepływu w grafie

Jest to problem polegający na wyznaczeniu w grafie maksymalnej wielkości przepływu z wierzchołka startowego do wierzchołka końcowego, jednocześnie biorąc pod uwagę ograniczenia przepustowości nałożone na poszczególne krawędzie w badanym grafie.

3.3.1. Algorytm Forda-Fulkersona

Idea działania algorytmu Forda-Fulkersona opiera się na zasadzie mówiącej, że należy zwiększać przepływ wzdłuż dowolnej ścieżki w grafie, prowadzącej z wierzchołka startowego do wierzchołka końcowego do momentu aż jest to możliwe. Do implementacji algorytmu Forda-Fulkersona potrzebny jest algorytm przeszukujący graf w poszukiwaniu ścieżek rozszerzających. Do tego celu można algorytmu przeszukiwania wszerz oraz algorytmu przeszukiwania w głąb. W celu porównaniu wyników w mojej implementacji użyte zostały oba te algorytmy. Jeśli do implementacji użyliśmy algorytmu przeszukiwania wszerz to złożoność obliczeniowa powinna wynieść $O(|V|^2 * |E|)$, gdzie |V| jest liczbą wierzchołków w grafie, a |E| jest liczba krawędzi. Jeśli do implementacji algorytmu Forda-Fulkersona zastosowaliśmy przeszukiwanie w głąb to złożoność powinna wynosić O(|f| * |E|), gdzie |f| jest maksymalnym przepływem w grafie, a |E| jest liczbą krawędzi w grafie.

4. Wyniki Pomiarów

4.1. Problem wyznaczenia minimalnego drzewa rozpinającego

4.1.1. Algorytm Prima

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	5,71	13,11	70,09	230,62	801,05
50%	12,43	74,30	325,68	1064,00	4027,10
75%	14,24	121,76	779,65	2330,70	9144,73
99%	14,83	120,73	1058,25	2399,69	14695,56

Tabela 1 Czas działania algorytmu Prima [µs] dla listy sąsiedztwa

b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25	7,52	5,55	70,06	185,51	687,83
50	8,88	21,89	247,34	879,55	3597,20
75	9,33	34,73	652,92	2398,91	9619,09
99	10,88	77,77	1140,09	2627,76	17998,99

Tabela 2 Czas działania algorytmu Prima [µs] dla macierzy incydencji

4.1.2. Algorytm Kruskala

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	6,24	46,63	166,60	505,97	1696,36
50%	12,33	119,70	555,12	1804,59	8347,65
75%	16,68	146,90	1128,43	3816,32	14920,25
99%	18,56	170,00	1836,69	6551,14	25512,83

Tabela 3 Czas działania algorytmu Kruskala [µs] dla listy sąsiedztwa

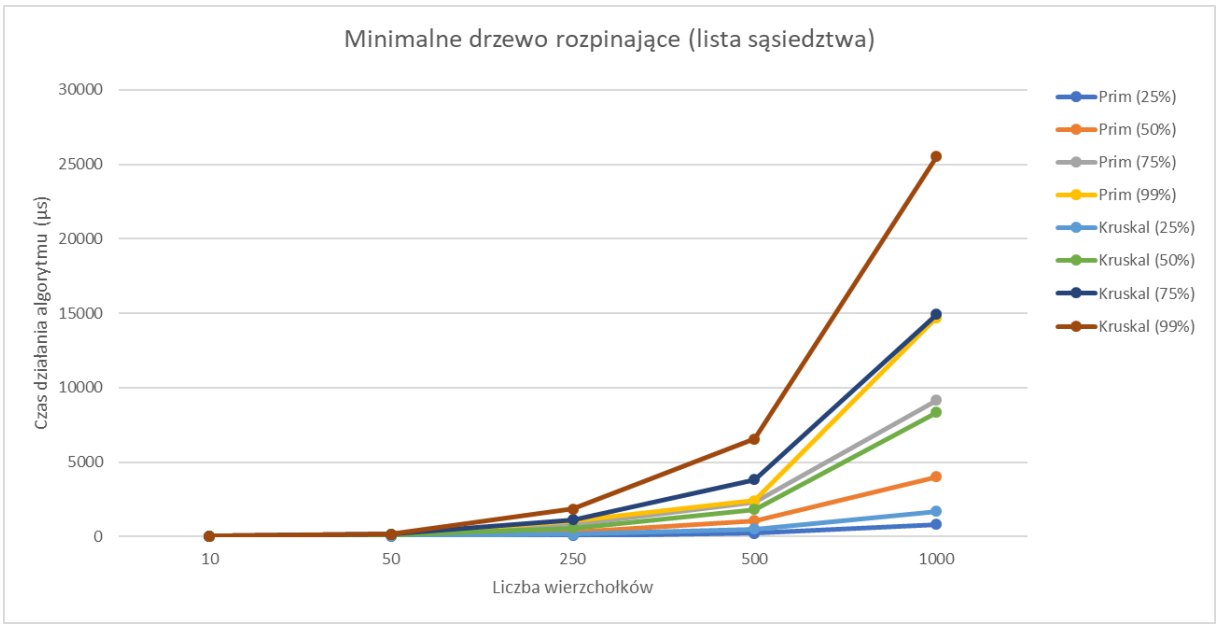
b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	7,83	67,31	192,95	595,16	2523,73
50%	17,75	100,36	590,75	2041,21	8702,46
75%	21,04	145,43	1210,80	4337,12	17786,18
99%	40,88	180,40	2396,16	7557,04	31452,50

Tabela 4 Czas działania algorytmu Kruskala [µs] dla macierzy incydencji

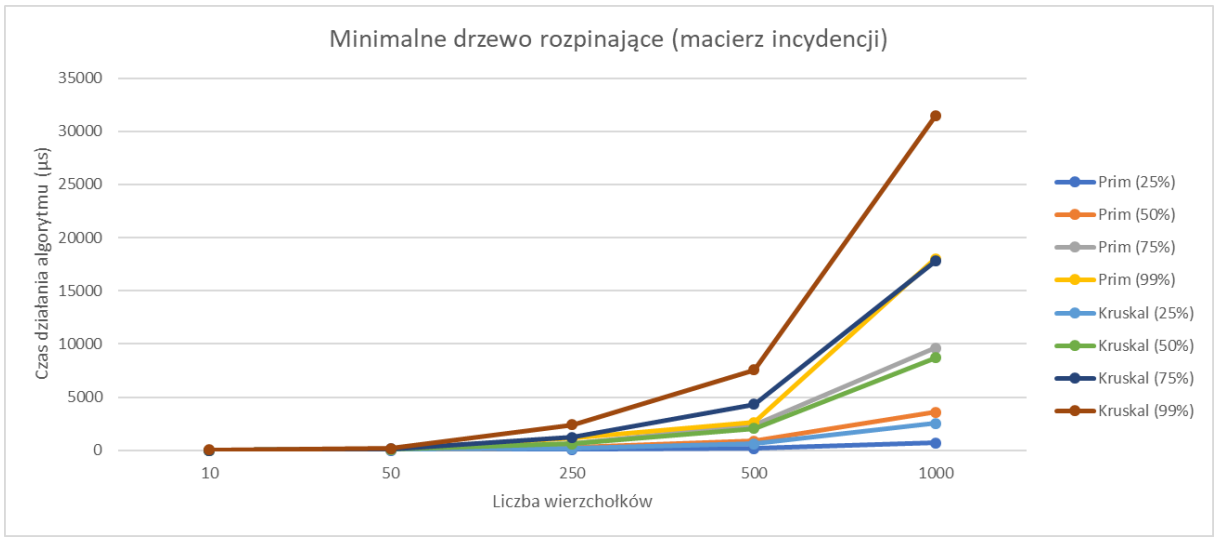
4.1.3. Wykresy typu I

a) Lista sąsiedztwa



Wykres 1 Porównanie czasu działania algorytmu Prima i Kruskala w zależności od liczby wierzchołków dla różnych gęstości

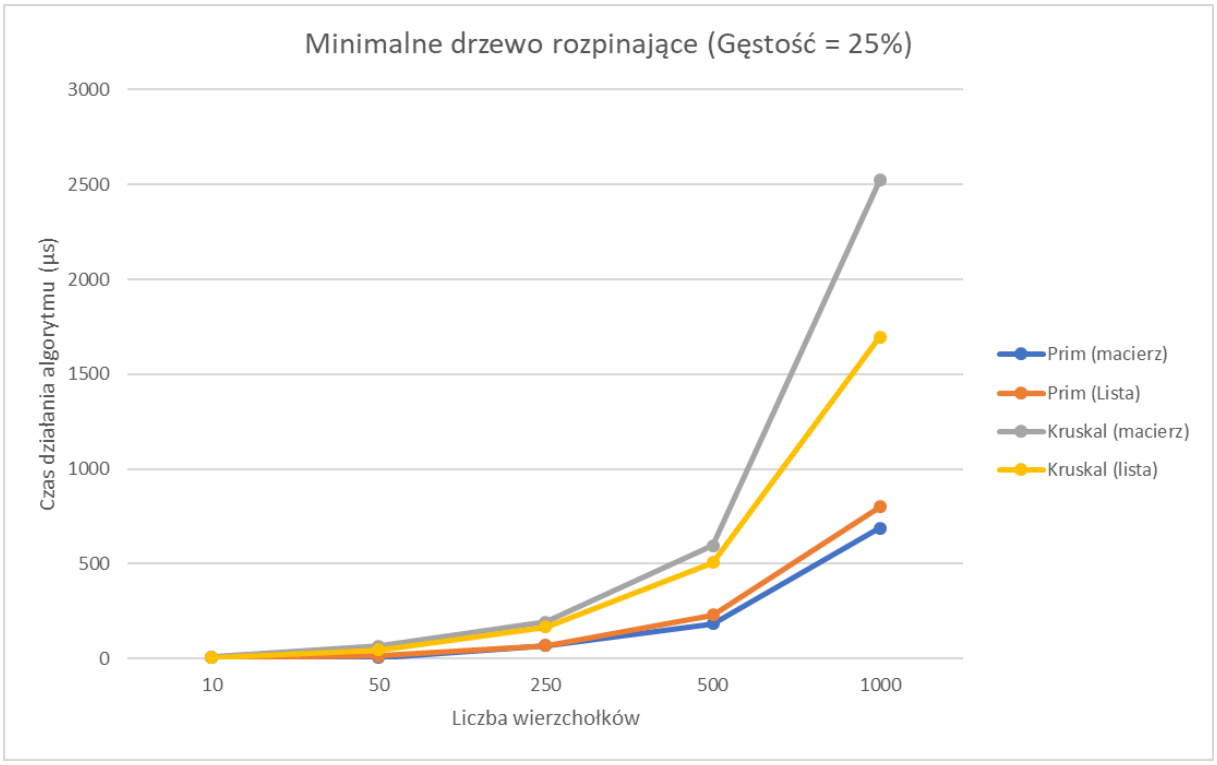
b) Macierz incydencji



Wykres 2 Porównanie czasu działania algorytmu Prima i Kruskala w zależności od liczby wierzchołków dla różnych gęstości

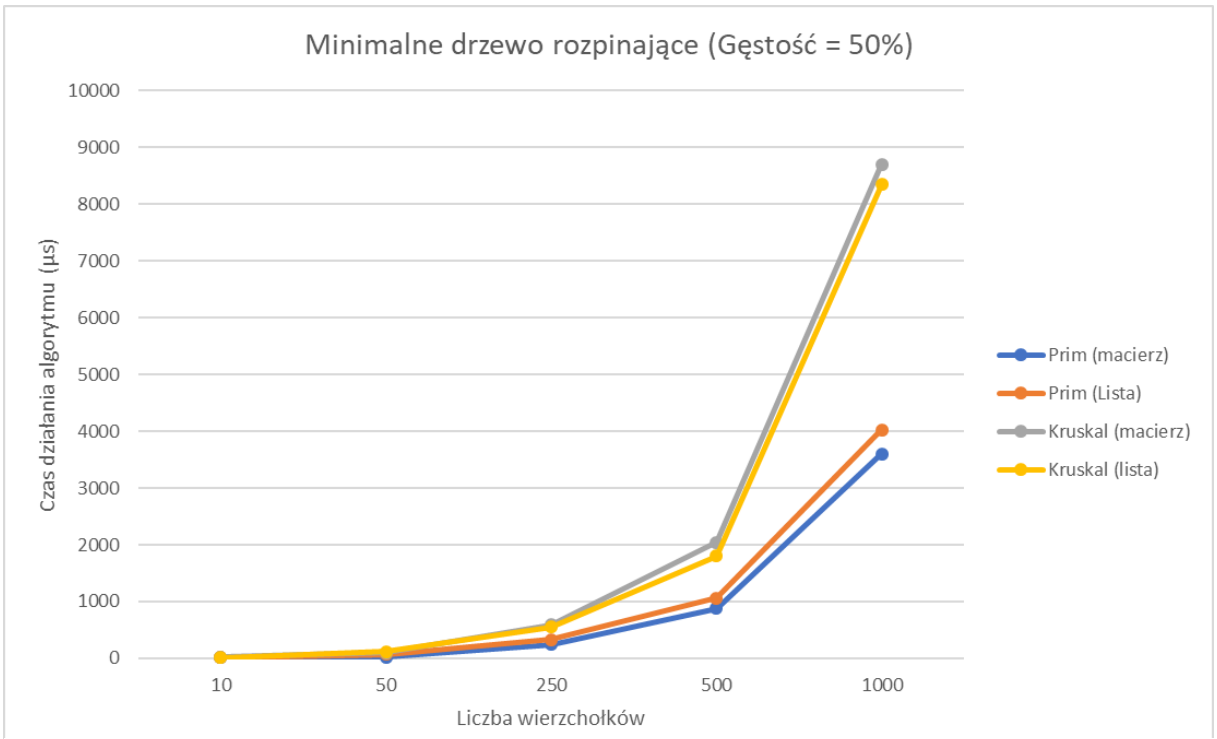
4.1.4. Wykresy typu II

a) Gęstość = 25%



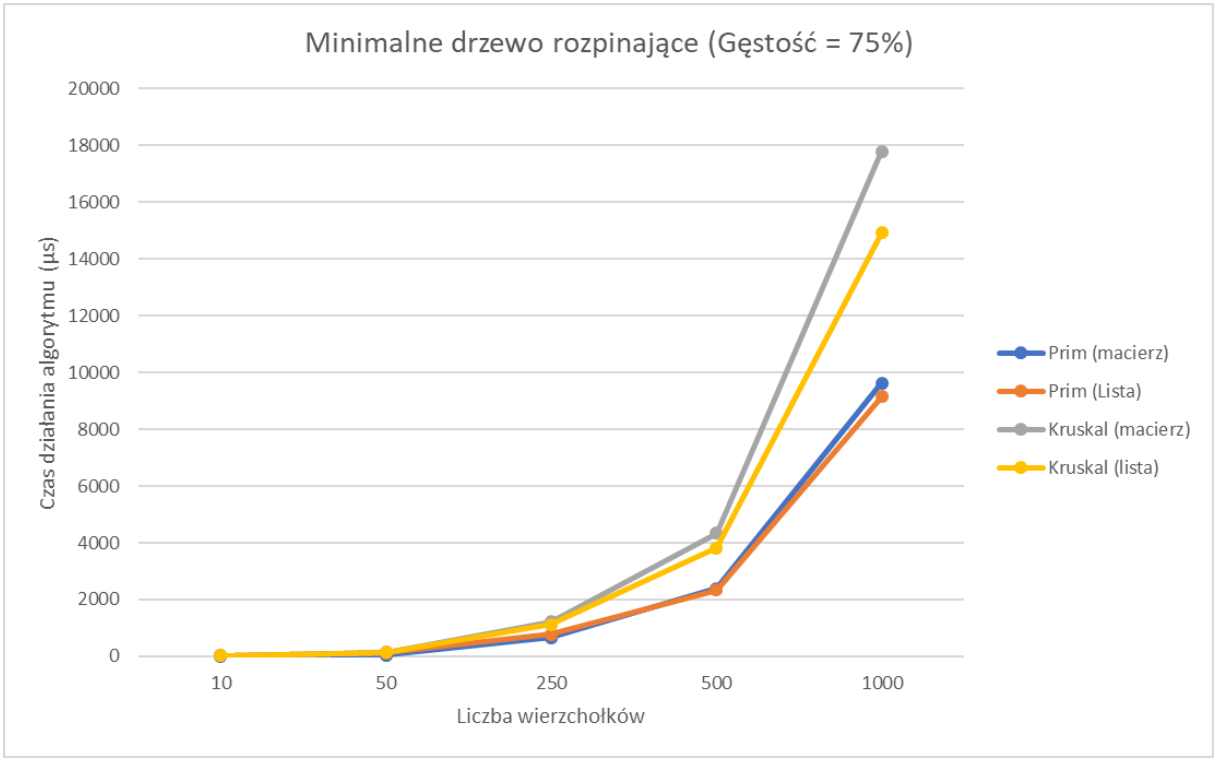
Wykres 3 Porównanie czasu działania algorytmu Prima i Kruskala dla różnych reprezentacji w pamięci komputera

b) Gęstość = 50%



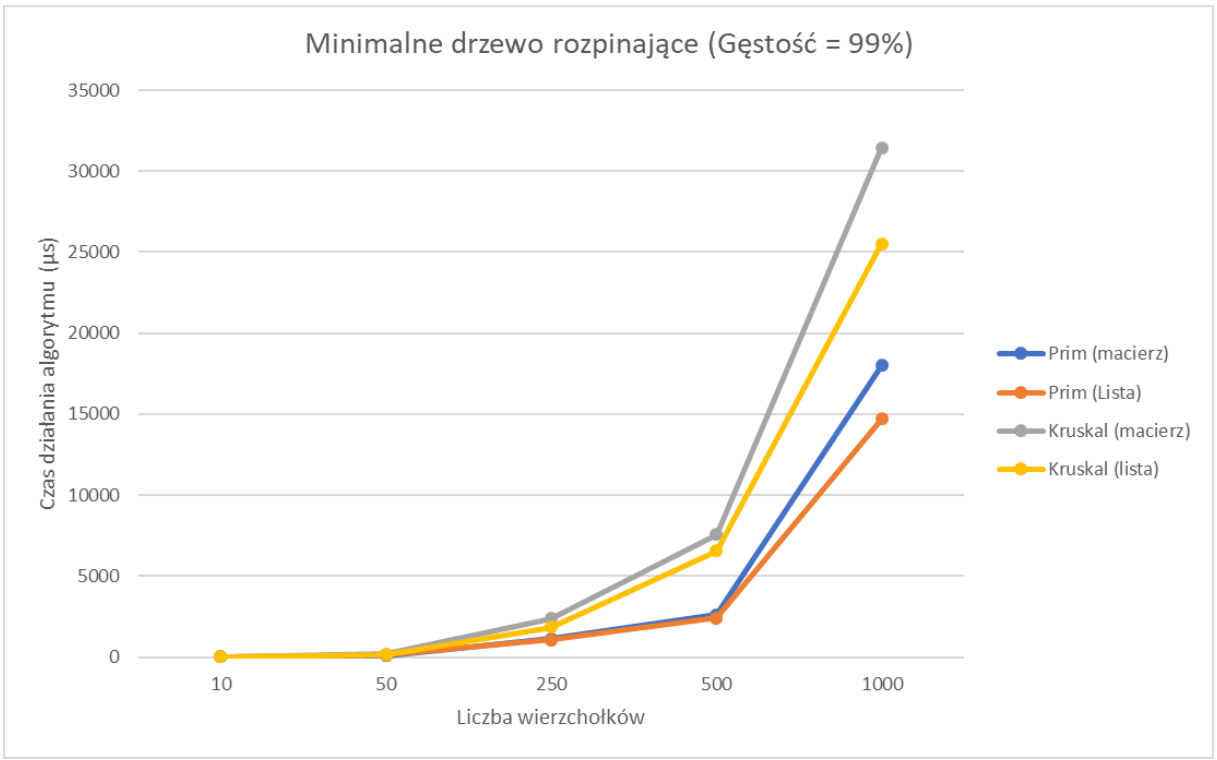
Wykres 4 Porównanie czasu działania algorytmu Prima i Kruskala dla różnych reprezentacji w pamięci komputera

c) Gęstość = 75%



Wykres 5 Porównanie czasu działania algorytmu Prima i Kruskala dla różnych reprezentacji w pamięci komputera

d) Gęstość = 99%



Wykres 6 Porównanie czasu działania algorytmu Prima i Kruskala dla różnych reprezentacji w pamięci komputera

4.2. Problem wyznaczania najkrótszej ścieżki w grafie

4.2.1. Algorytm Dijkstry

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	3,37	11,52	73,88	252,23	778,54
50%	19,54	46,19	499,39	1503,77	5818,77
75%	25,24	226,01	2906,10	7703,50	25865,97
99%	39,30	947,64	16474,14	123473,56	624983,95

Tabela 5 Czas działania algorytmu Dijkstry [µs] dla listy sąsiedztwa

b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	5,19	5,08	58,27	186,25	780,16
50%	6,15	33,75	287,25	1111,99	4838,83
75%	10,65	68,00	1247,50	4191,25	18720,83
99%	15,59	534,56	33881,00	5429,10	446282,11

Tabela 6 Czas działania algorytmu Dijkstry [µs] dla macierzy incydencji

4.2.2. Algorytm Bellmana-Forda

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	2,47	1,91	3,59	5,27	10,42
50%	9,24	12,96	14,01	17,47	22,34
75%	12,42	16,03	18,98	26,73	31,69
99%	20,43	24,22	38,16	45,65	61,37

Tabela 7 Czas działania algorytmu Bellmana-Forda [μs] dla listy sąsiedztwa

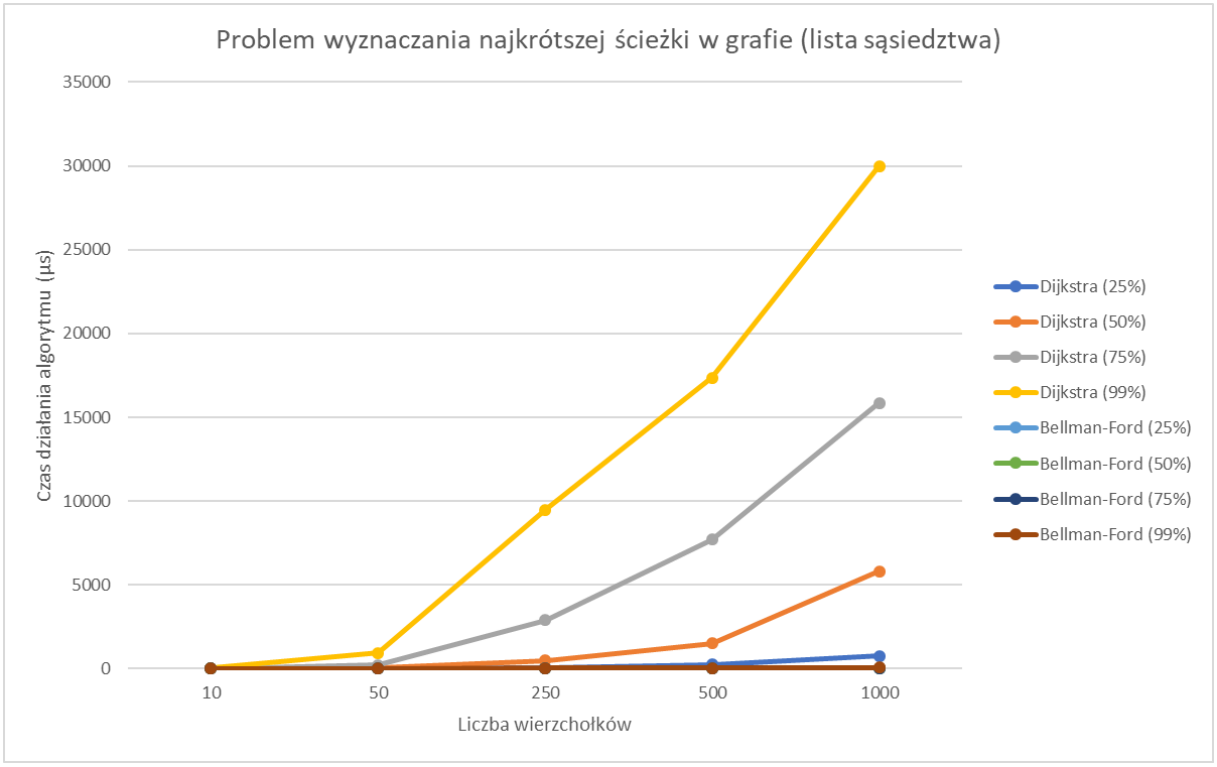
b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	1,33	1,91	13,59	36,65	145,09
50%	1,65	2,59	20,98	73,93	297,30
75%	3,42	3,30	29,23	107,35	458,01
99%	4,04	4,53	47,50	107,35	601,91

Tabela 8 Czas działania algorytmu Bellmana-Forda [μs] dla macierzy incydencji

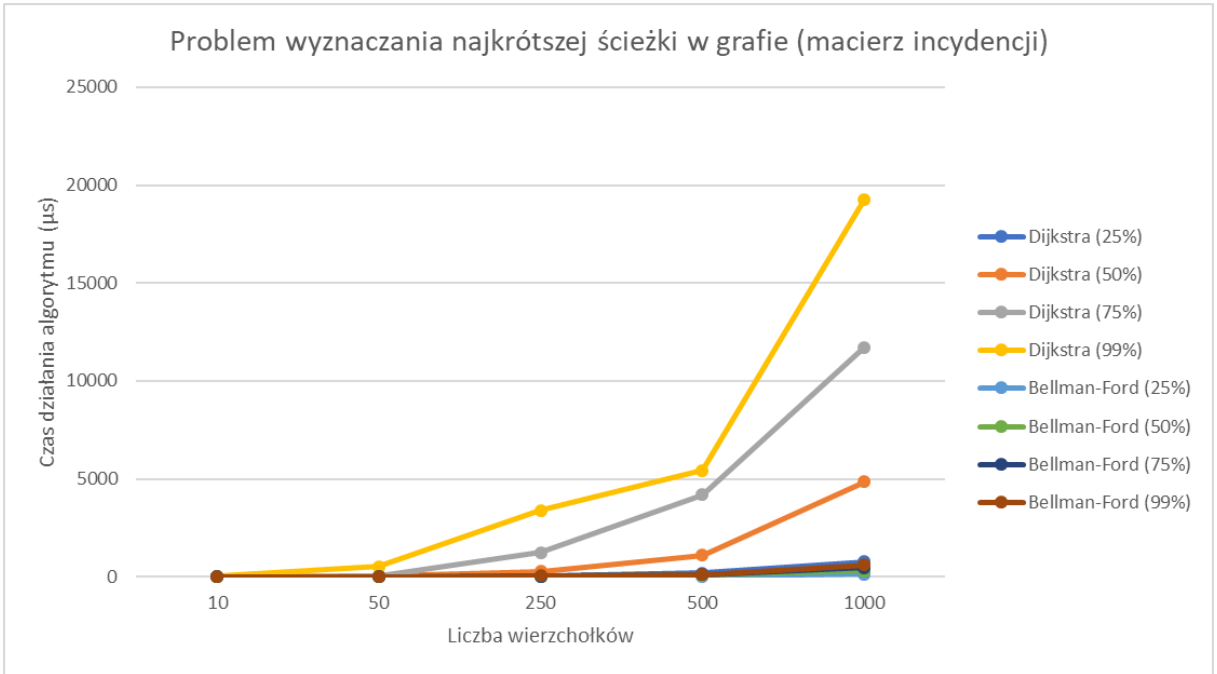
4.2.3. Wykresy typu I

a) Lista sąsiedztwa



Wykres 7 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda w zależności od liczby wierzchołków dla różnych gęstości

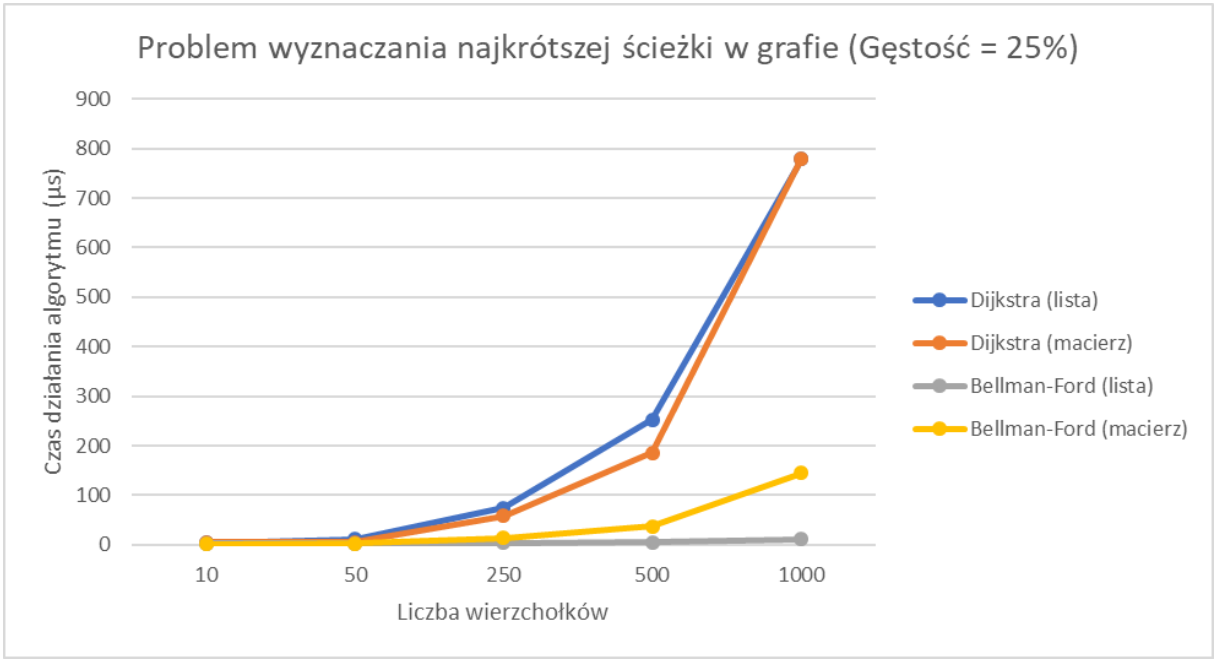
b) Macierz incydencji



Wykres 8 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda w zależności od liczby wierzchołków dla różnych gęstości

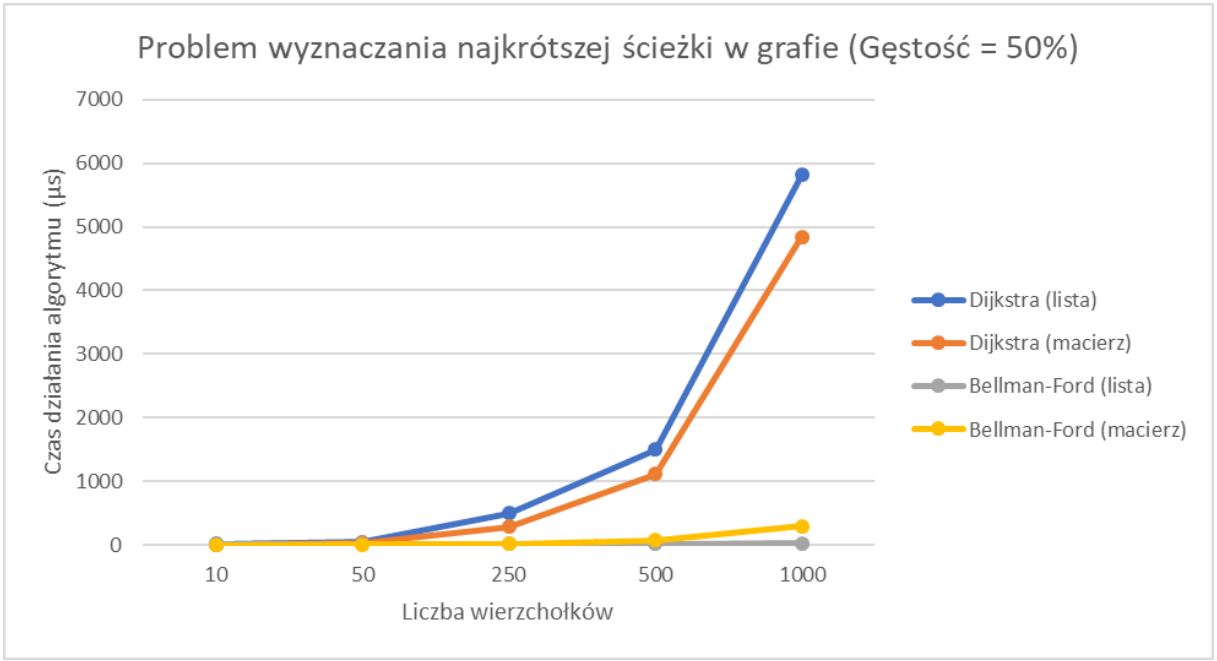
4.2.4 Wykresy typu II

a) Gęstość = 25%



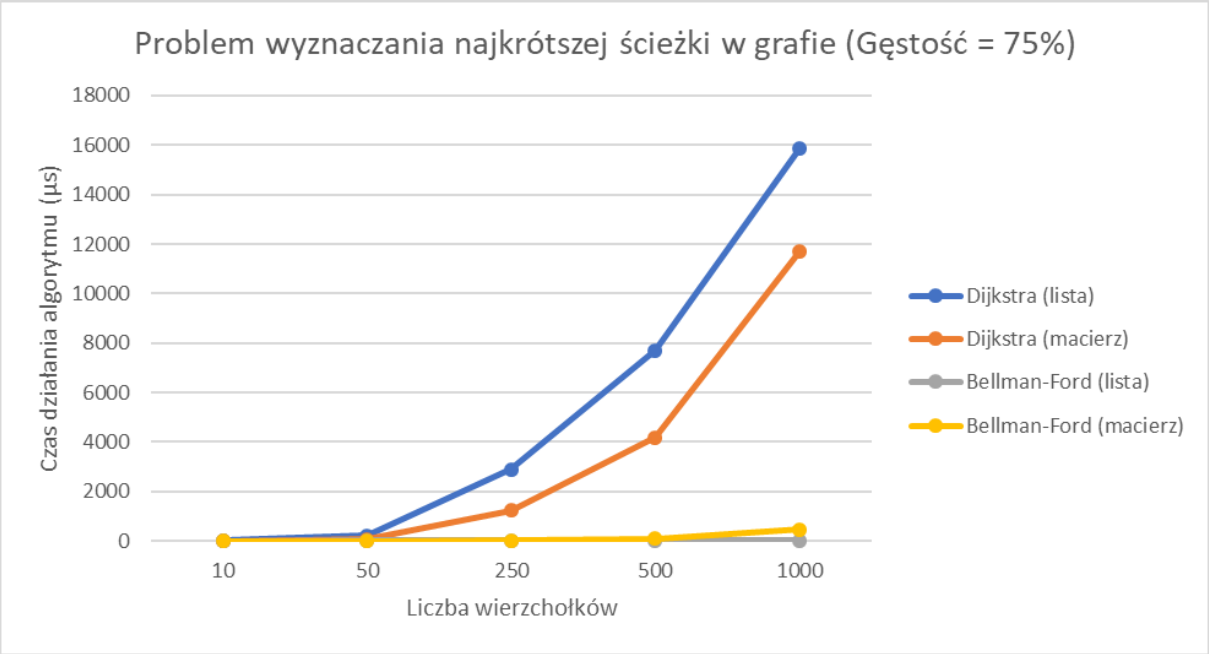
Wykres 9 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda dla różnych reprezentacji w pamięci komputera

b) Gęstość = 50%



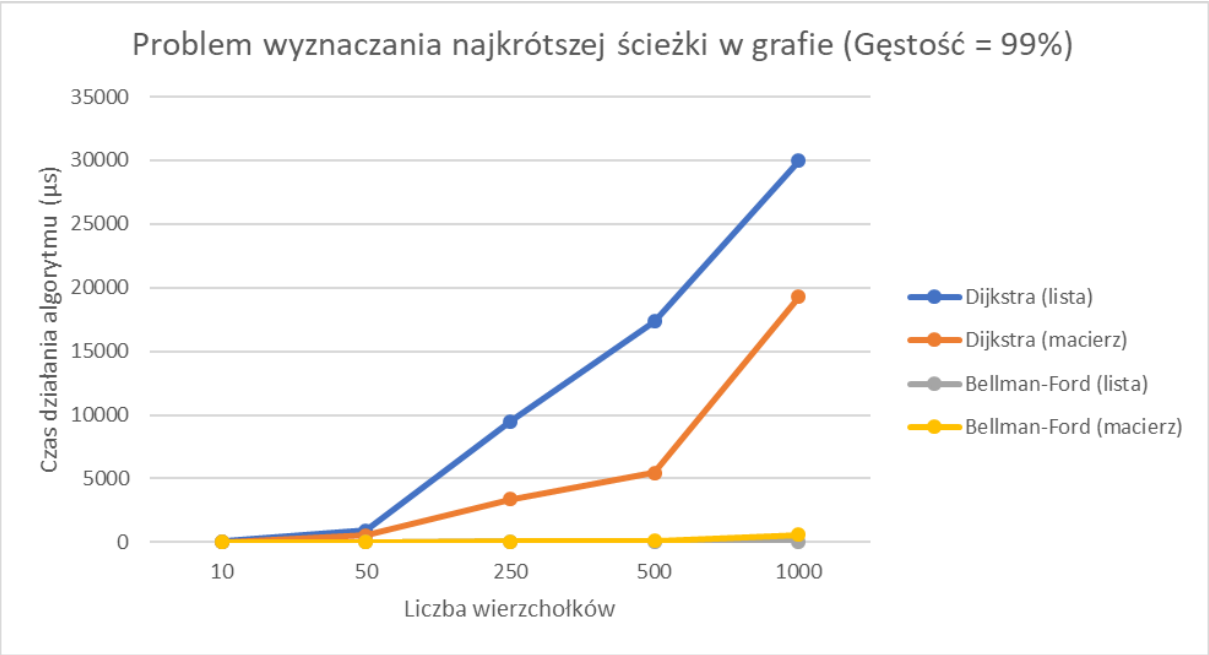
Wykres 10 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda dla różnych reprezentacji w pamięci komputera

c) Gęstość = 75%



Wykres 11 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda dla różnych reprezentacji w pamięci komputera

d) Gęstość = 99%



Wykres 12 Porównanie czasu działania algorytmu Dijkstry i algorytmu Bellmana-Forda dla różnych reprezentacji w pamięci komputera

4.3. Problem maksymalnego przepływu w grafie

4.3.1. Algorytm Forda-Fulkersona (BFS)

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	11,41	15,12	19,56	26,11	35,58
50%	16,64	18,74	21,51	28,52	39,51
75%	18,51	19,51	25,69	31,51	42,51
99%	23,41	25,51	29,61	39,61	59,91

Tabela 9 Czas działania algorytmu Forda-Fulkersona [µs] z (BFS) dla listy sąsiedztwa

b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	17,45	19,94989	24,41	25,67	39,51
50%	30,91	34,12	35,70371	42,12	47,31
75%	37,31	39,51	43,56	49,98	56,57
99%	44,52	49,33	52,76	54,57	62,56

Tabela 10 Czas działania algorytmu Forda-Fulkersona [µs] z (BFS) dla macierzy incydencji

4.3.2. Algorytm Forda-Fulkersona (DFS)

a) Lista sąsiedztwa

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	13,54	16,15	18,26	27,34	29,99
50%	17,58	18,01	27,67	30,041	38,27
75%	19,34	20,59	36,05	40,78	53,49
99%	23,41	29,05	47,43	54,25	75,22

Tabela 11 Czas działania algorytmu Forda-Fulkersona [μs] z (DFS) dla listy sąsiedztwa

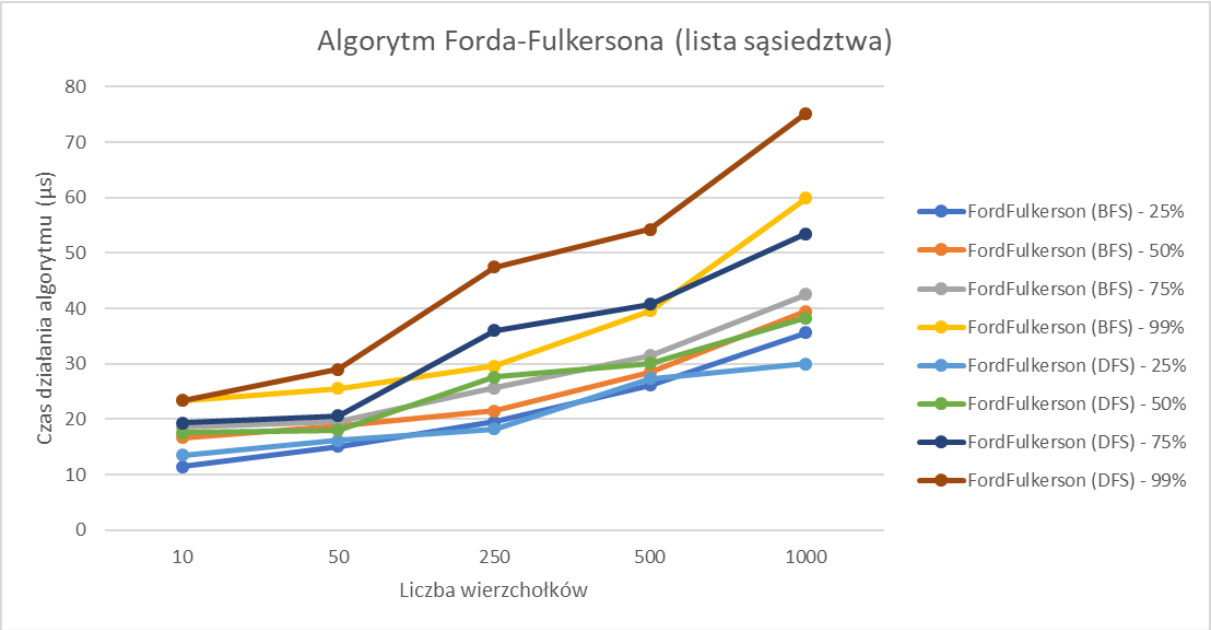
b) Macierz incydencji

	Liczba wierzchołków				
Gęstość	**10**	**50**	**250**	**500**	**1000**
25%	12,55	14,62	19,02	38,73	43,33
50%	14,3	15,51	27,09	50,37	59,36
75%	18,59	19,78	40,59	58,89	70,83
99%	21,82	31,65	58,99	72,43	85,92

Tabela 12 Czas działania algorytmu Forda-Fulkersona [μs] z (DFS) dla macierzy incydencji

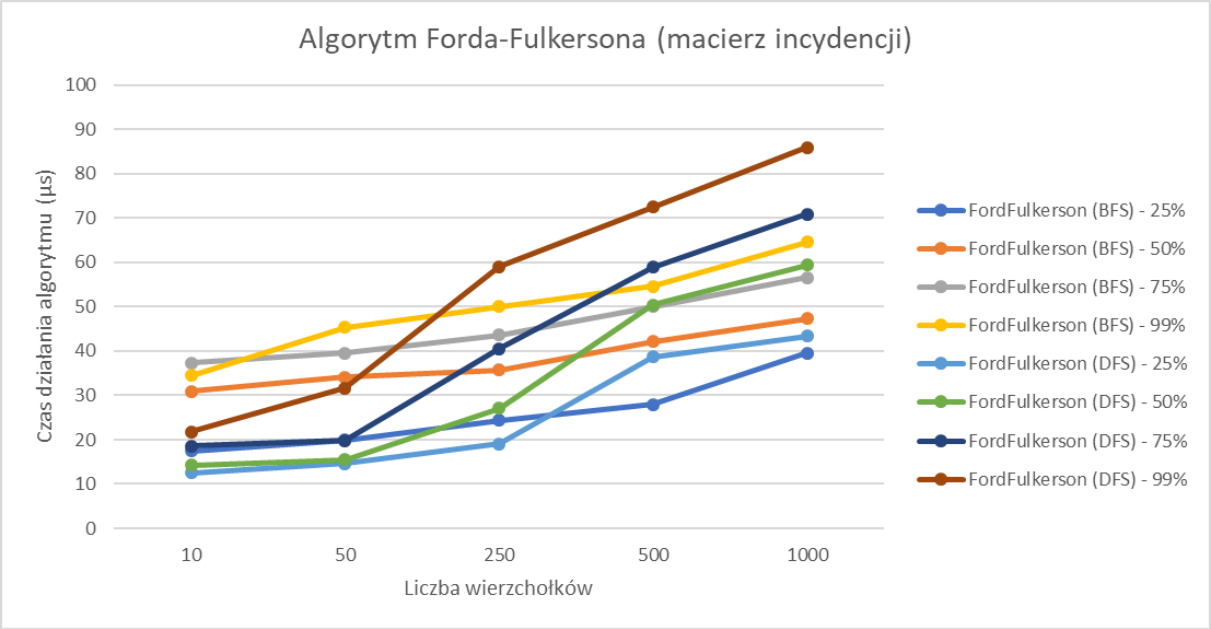
4.3.3. Wykresy typu I

a) Lista sąsiedztwa



Wykres 13 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS w zależności od liczby wierzchołków dla różnej gęstości

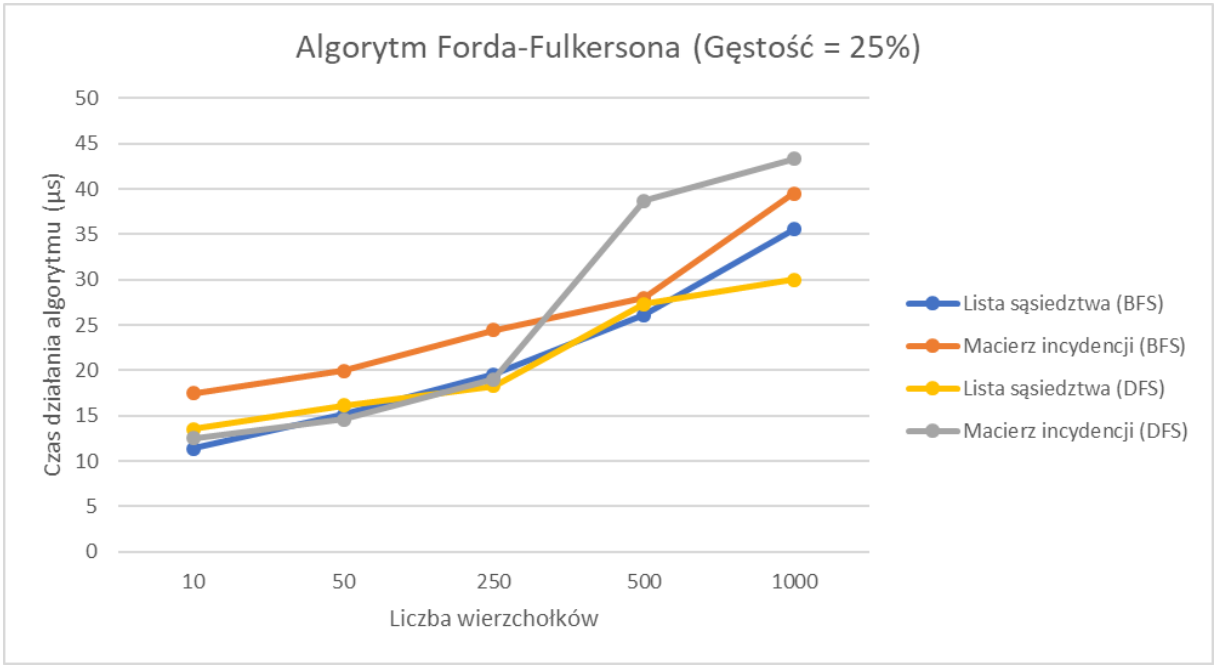
b) Macierz incydencji



Wykres 14 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS w zależności od liczby wierzchołków dla różnej gęstości

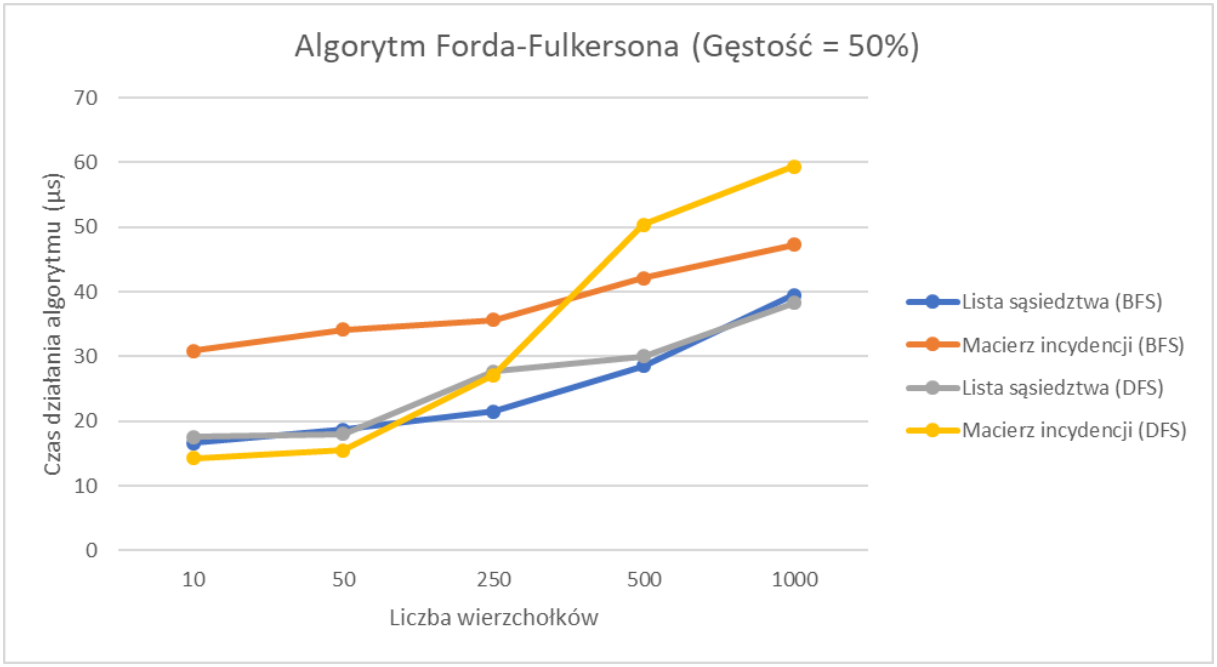
4.3.4. Wykresy typu II

a) Gęstość = 25%



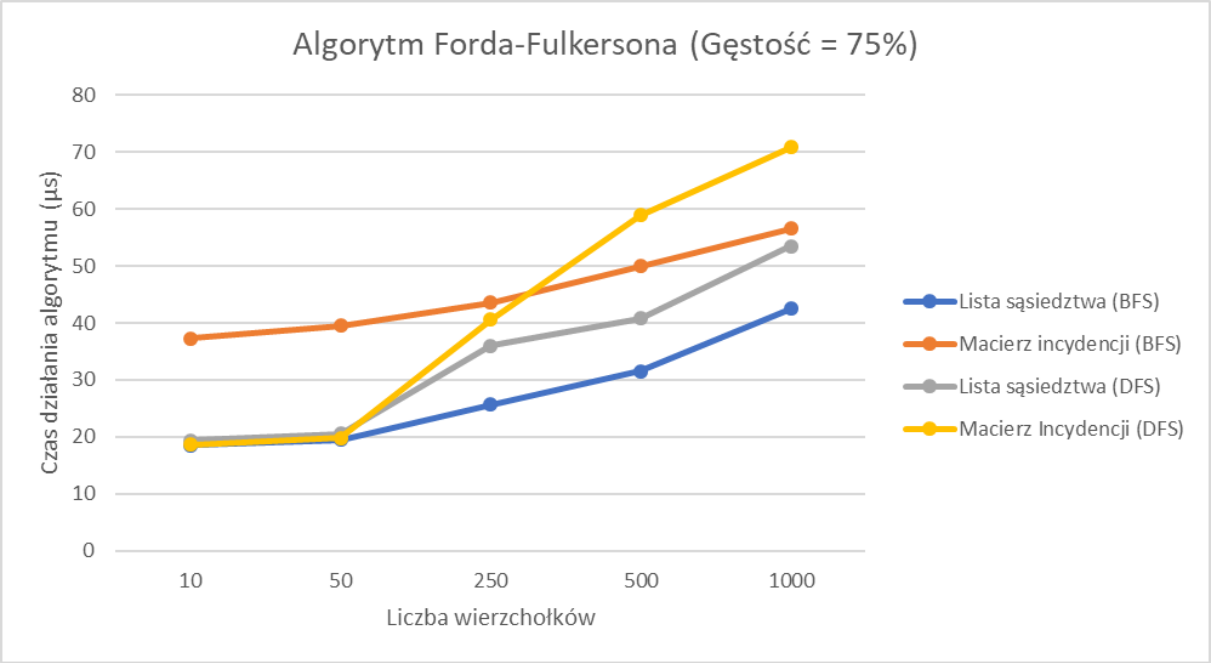
Wykres 15 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS dla różnych reprezentacji w pamięci komputera

b) Gęstość = 50%



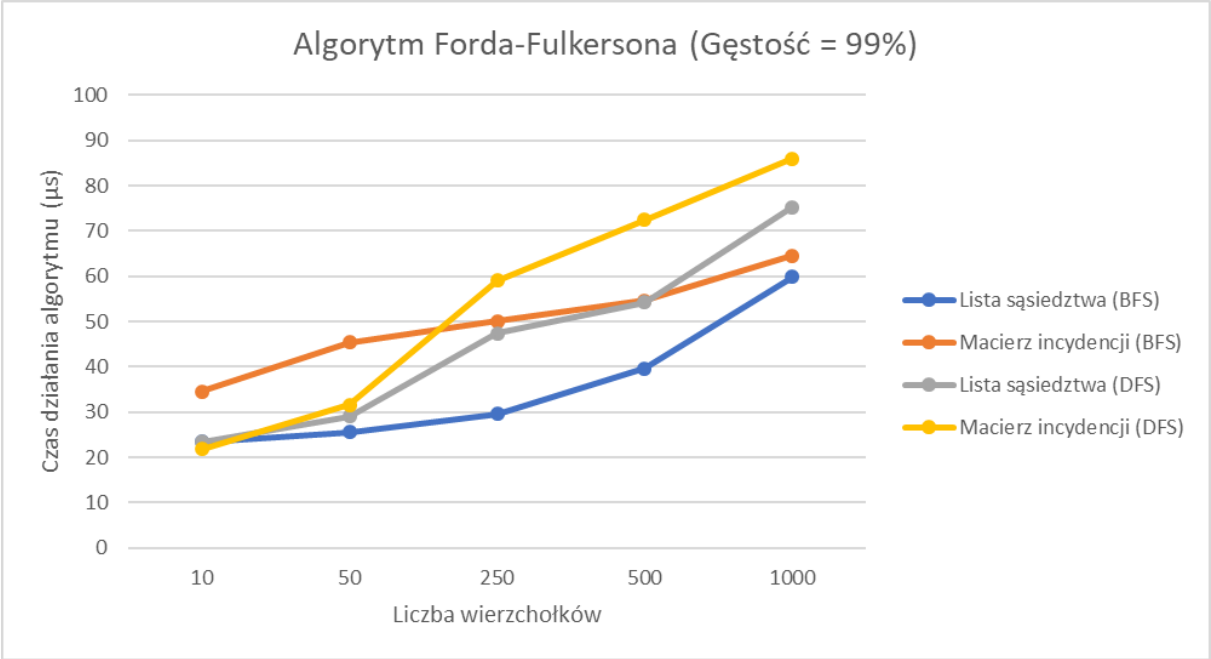
Wykres 16 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS dla różnych reprezentacji w pamięci komputera

c) Gęstość = 75%



Wykres 17 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS dla różnych reprezentacji w pamięci komputera

d) Gęstość = 99%



Wykres 18 Porównanie czasu działania algorytmu Forda-Fulkersona z BFS i z DFS dla różnych reprezentacji w pamięci komputera

4. Wnioski

4.1. Problem wyznaczania minimalnego drzewo rozpinającego

Dla liczby wierzchołków w grafie nie większej niż 250 czas działania algorytmu Prima oraz algorytmu Kruskala zarówno dla macierzy incydencji jak i dla listy sąsiedztwa był bardzo zbliżony. Zauważalna różnica w wydajności obu algorytmów widoczna jest dla badań przeprowadzonych dla 500 i 1000 wierzchołków w grafie. W tych przypadkach lepszy okazał się algorytm Prima. Różnice w czasie działania pomiędzy listą sąsiedztwa oraz macierzą incydencji są niewielki, jednak w każdym z przypadków dla algorytmu Kruskala lepsza okazała się reprezentacja w postaci listy sąsiedztwa, dla algorytmu Prima w przypadku mniejszej gęstości krawędzi w grafie lepsza była macierz incydencji, a kiedy gęstość wzrosła lepsze wyniki osiągnęła lista sąsiedztwa. Różnice te są jednak bardzo małe i mogą w pewnym stopniu wynikać z niedostatecznej ilości przeprowadzonych testów lub niedoskonałości narzędzi, którymi były przeprowadzana pomiary czasu. Lepsze wyniki pomiarów dla obu algorytmów moglibyśmy otrzymać w przypadku zaimplementowania kolejki priorytetowej przy pomocy kopca Fibonacciego.

4.2. Problem najkrótszej ścieżki w grafie

W przypadku algorytmów wyznaczania najkrótszej ścieżki w grafie możemy wyraźnie zaobserwować lepszy czas działania jednego algorytmu od drugiego. W przypadku teoretycznym Algorytm Dijkstry powinien osiągać lepsze wyniki pomiarów niż Algorytm Bellmana-Forda ze względu na specyfikę swojego działania oraz fakt, że algorytm Bellmana-Forda w przeciwieństwie do algorytmu Dijkstry działa poprawnie również dla ujemnych liczb. Widoczne na wykresach różnice mogą wynikać z mało wydajnej implementacji algorytmu Dijkstry. Różnice w czasie działania pomiędzy algorytmami są widoczne zarówno dla reprezentacji w formie listy sąsiedztwa jaki i w formie macierzy incydencji. W przypadku algorytmu Bellmana-Forda, dla gęstości 25% wynik pomiaru dla macierzy incydencji jest gorszy niż dla listy sąsiedztwa, jednak wraz ze wzrostem gęstości różnica ta się znacząco zmniejsza. Algorytm Dijkstry dla wszystkich gęstości działał mniej wydajnie dla reprezentacji w postaci listy sąsiedztwa.

4.3. Problem maksymalnego przepływu w grafie

W przypadku Algorytmu Forda-Fulkersona badaniu zostały podane dwie jego wersje, pierwsza wykorzystująca algorytm przeszukiwania wszerz oraz druga korzystająca z algorytmu przeszukiwania w głąb. Zarówno w przypadku reprezentacji grafu w postaci macierzy incydencji jak i listy sąsiedztwa gorsze wyniki pomiarów uzyskała wersja z algorytmem przeszukiwania w głąb. Dla macierzy incydencji różnica ta jest lepiej widoczna ponieważ dla 1000 wierzchołków algorytm Forda-Fulkersona z przeszukiwaniem wszerz dla gęstości 99% uzyskał lepszy wynik niż wersja z przeszukiwaniem w głąb dla gęstości 77%. Dla obu wersji algorytmu Forda-Fulkersona oraz dla wszystkich gęstości, reprezentacja grafu w postaci macierzy incydencji uzyskała gorsze wyniki niż reprezentacja w postaci listy sąsiedztwa. Różnice w pomiarach pomiędzy wersją z algorytmem przeszukiwania w głąb oraz wersją z algorytmem przeszukiwania wszerz mogą wynikać z faktu, że algorytm przeszukiwania w głąb generuje zazwyczaj dłuższe ścieżki powiększające co spowolnia działania całego algorytmu.

5. Bibliografia

a) „Wprowadzenie do algorytmów” Thomas H. Cormen, Charles Leiserson, Ronald L. Rivest