Struktury Danych i Złożoność Obliczeniowa

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

Autor:
Kamil Bauer 259102

Prowadzący: dr inż. Dariusz Banasiak
Kod zajęć: K02-10j
Wrocław, 3 czerwca 2022

Politechnika Wrocławska,
Wydział Informatyki i Telekomunikacji

Spis treści

1 Założenia projektowe

Celem projektu była implementacja i pomiar czasu działania algorytmów na grafach w reprezentacji listowej oraz macierzy incydencji.
Zaimplementowanymi algorytmami są:

I algorytmy uzyskania minimalnego drzewa spinającego (MST):

(a) algorytm Jarnika-Prima

(b) algorytm Kruskala

II algorytmy uzyskiwania najkrótszych ścieżek:

(a) algorytm Dijkstry

(b) algorytm Bellmana-Forda

Badania zostały wykonane dla 5 różnych (reprezentatywnych) liczb wierzchołków V oraz następujących gęstości grafu: 25%, 50%, 75% oraz 99%. Koszty wszystkich krawędzi są dodatnie. Pomiar czasu został wykonany wielokrotnie (przynajmniej 50 razy), ponieważ pojedynczy pomiar może być obarczony znacznym błędem, jak również generując za każdym razem nowy zestaw danych - otrzymane wyniki mogą zależeć od rozkładu danych,, a wyniki uśredniono. Pomiar ten został wykonany z wykorzystaniem *std::chrono::high_ resolution_ clock* z rzutowaniem na mikrosekundy.
Losowe generowanie grafu zależało od algorytmu, dla którego był on wykorzystywany - dla wyznaczania MST najpierw jest generowane losowe drzewo spinające, po czym zostały dodawane losowe krawędzie dopóki graf nie osiągnie żądanej gęstości. Aby graf nie stał się multigrafem, nowo dodawane krawędzie są najpierw zapisywane do macierzy sąsiedztwa. Graf do algorytmów MST jest nieskierowany. Natomiast losowo generowany graf do szukania najkrótszej ścieżki generuje najpierw pierścień (łączy kolejne wierzchołki ze sobą), aby istniała ścieżka do każdego wierzchołka. Sposób generowania pozostałych krawędzi jest analogiczny do poprzedniego. Graf używany do wyszukiwania najkrótszych ścieżek jest skierowany.

1.1 Wykorzystane narzędzia

Narzędzia wykorzystane podczas pracy:

- C++11
- *std::chrono::high_ resolution_ clock* do pomiaru czasu
- Visual Studio 2019
- Git

1.2 Link do repozytorium na githubie

Cały kod źródłowy znajduje się na zdalnym repozytorium:
`https://github.com/bauerkamil/SDiZO_2.git`

2 Struktury danych

2.1 Macierz incydencji

Macierz incydencji jest macierzą o wymiarze V x E, gdzie V oznacza liczbę wierzchołków grafu, a E liczbę jego krawędzi. Każdy wiersz tej macierzy odwzorowuje jeden wierzchołek grafu. Każda kolumna odwzorowuje jedną krawędź. Złożoność pamięciowa to O(V * E). Operacje przejrzenia wszystkich krawędzi, sąsiadów danego wierzchołka i sprawdzenie, czy dana krawędź istnieje wykonują się w czasie O(V + E).

2.2 Lista sąsiedztwa

Lista sąsiedztwa jest reprezentacją grafu, w której wykorzystujemy tablicę V-elementową, gdzie V oznacza liczbę wierzchołków. Każdy element tej tablicy jest listą. Lista reprezentuje wierzchołek startowy. Na liście są przechowywane numery wierzchołków końcowych, czyli sąsiadów wierzchołka startowego, z którymi jest on połączony krawędzią. Listy sąsiedztwa są efektywnym pamięciowo sposobem reprezentacji grafu w pamięci komputera, ponieważ zajmują pamięć rzędu O(E), gdzie E oznacza liczbę krawędzi grafu.

2.3 Inne struktury

Ponadto do stworzenia kolejki w trakcie wykonywania algorytmów został wykorzystany kopiec minimalny, jako przerobiony z kopca maksymalnego stworzonego w poprzednim projekcie. Do przekazywania wygenerowanych lub sczytanych z pliku danych o grafie, została wykorzystana lista krawędzi zapisywana jako macierz E x 3, gdzie pierwsze pole to wierzchołek początkowy, drugie - wierzchołek docelowy, a trzecie - waga krawędzi. W ten sposób jest też sczytywane z pliku, poza pierwszą linią, w której zapisana jest liczba krawędzi oraz liczba wierzchołków (rozdzielone spacją). Kolejne wartości są odpowiednio rozdzielone spacją i znakiem nowej linii.

3 Algorytmy uzyskiwania minimalnego drzewa spinającego

3.1 Algorytm Jarnika-Prima

Algorytm zajmujący się wyznaczaniem minimalnego drzewa spinającego w grafie.Na początku dodaje do zbioru A reprezentującego drzewo krawędź o najmniejszej wadze, łączącą wierzchołek początkowy v z dowolnym wierzchołkiem. W każdym kolejnym kroku procedura dodaje do A krawędź o najmniejszej wadze wśród krawędzi łączących wierzchołki już odwiedzone z nieodwiedzonymi. Jeśli struktura A jest kolejką priorytetową opartą na kopcu binarnym, czasowa złożoność obliczeniowa operacji wynosi O(E * logV).

liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	169,5833	492,9565	931,7391	1477,435
30	536,8261	2185,043	4186,87	5644,522
40	1259,348	6090,478	13539,35	16322,7
50	3525,739	14142,96	31589,91	39407,26
60	6490,609	26487,78	53249,43	103495,7
70	10976,26	46530,43	116717,1	252497,7
80	18470,7	75953,22	243032,5	390761,9
90	28908,96	188827,7	317796,2	468855,8

Tabela 1: Uśrednione wyniki eksperymentu dla macierzy incydencji

liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	117,6522	400,3913	872,5217	1189,217
30	389,6957	1649,391	4167,478	5097,652
40	951,7826	5563,826	10357,65	14365,04
50	2418,174	11421,74	27413,17	39476,48
60	5007,87	24535,17	49392,17	101292,7
70	8715,913	39830,91	100883	266440,3
80	16759,35	76653,39	247408,1	418718
90	27919,96	152297,5	307171,4	518987,6

Tabela 2: Uśrednione wyniki eksperymentu dla listy sąsiedztwa

3.2 Algorytm Kruskala

Algorytm operuje na liście posortowanych krawędzi pod względem ich wagi. W każdym kolejnym kroku pobierana jest pierwsza krawędź. Następnie sprawdzany jest warunek, czy w wyniku dodania jej do drzewa nie powstanie cykl. Złożoność obliczeniowa wynosi O(E * logV)

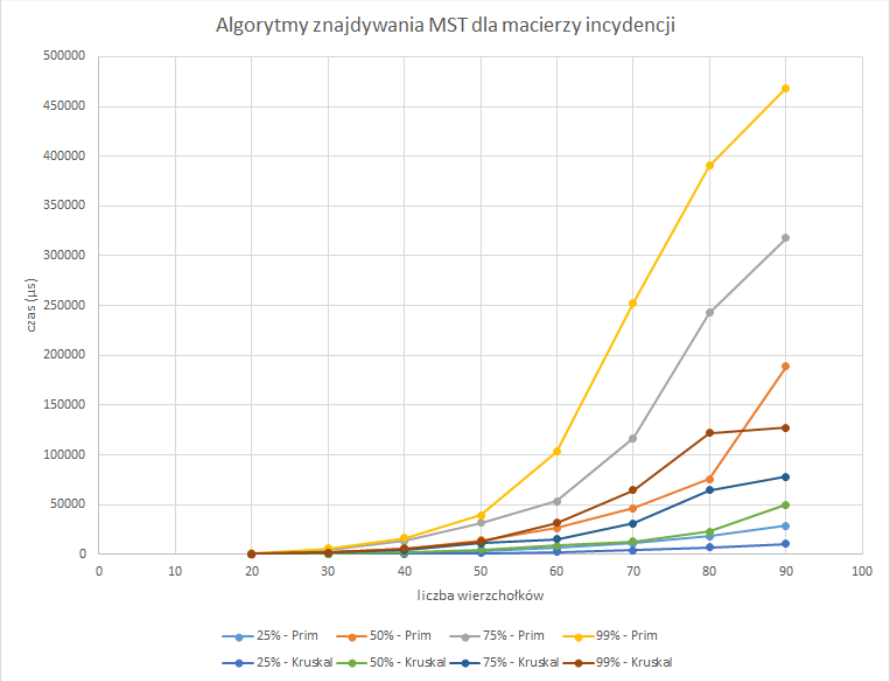
liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	118,0833	321,8696	474,6957	572,2174
30	245,6957	839,6522	1553,696	1975,739
40	550,1739	2000,739	3995,043	5625,87
50	1158,391	4994,174	11258	13134,35
60	2173	8983,043	15239,65	31638,26
70	3872,304	12771,57	31419,48	64510,26
80	7046,174	23108,65	64680,48	122256,2
90	10487,65	49939,65	78057,48	127063,7

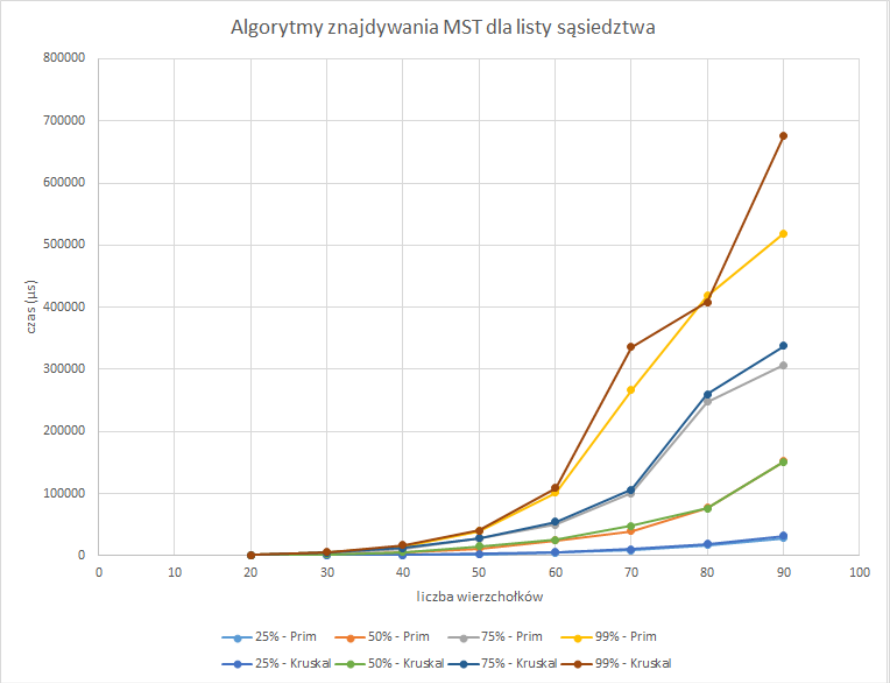
Tabela 3: Uśrednione wyniki eksperymentu dla macierzy incydencji

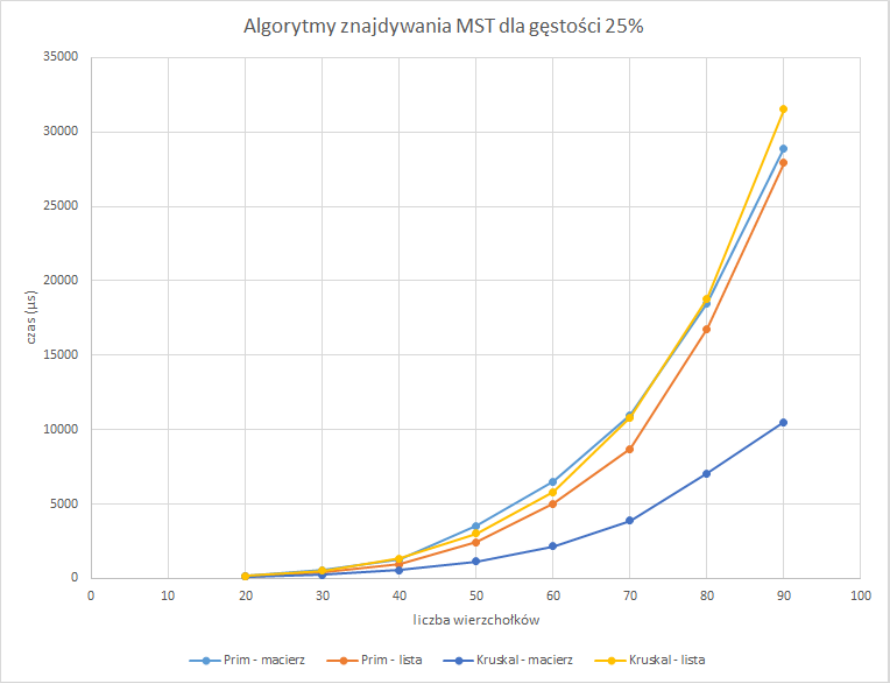
liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	158,8261	499,3913	966,1304	1286,565
30	497,2609	2114,783	4392,957	5999,696
40	1333,522	5613,13	12432,04	17268,22
50	3009,348	14992,17	28640,65	40775,52
60	5808,783	25291,17	54319,35	108761,8
70	10831,91	47814	106053,7	335591,8
80	18811,7	75961,87	259965	408639,7
90	31528	150929,4	337656,9	676590,8

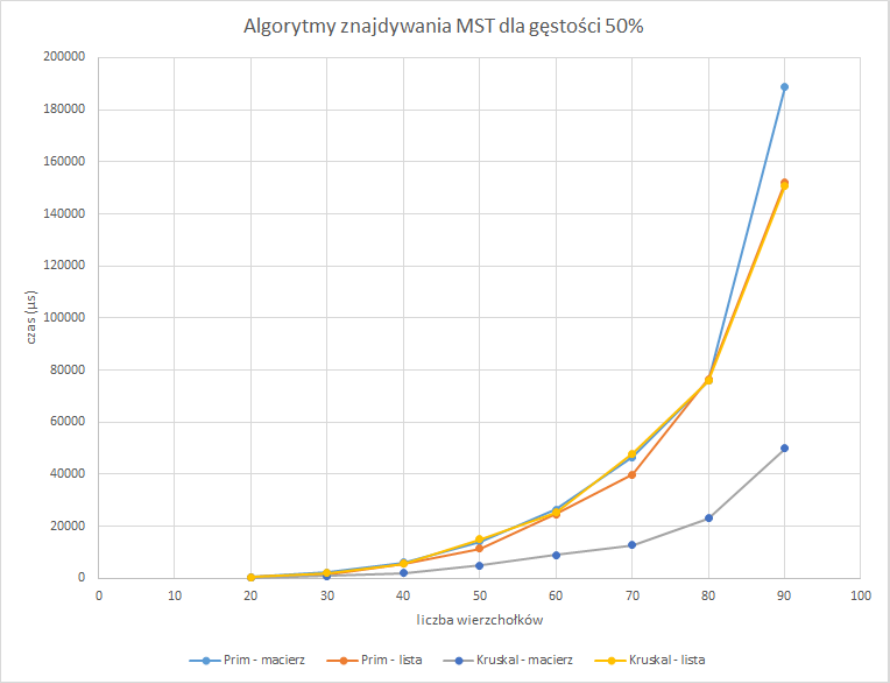
Tabela 4: Uśrednione wyniki eksperymentu dla listy sąsiedztwa

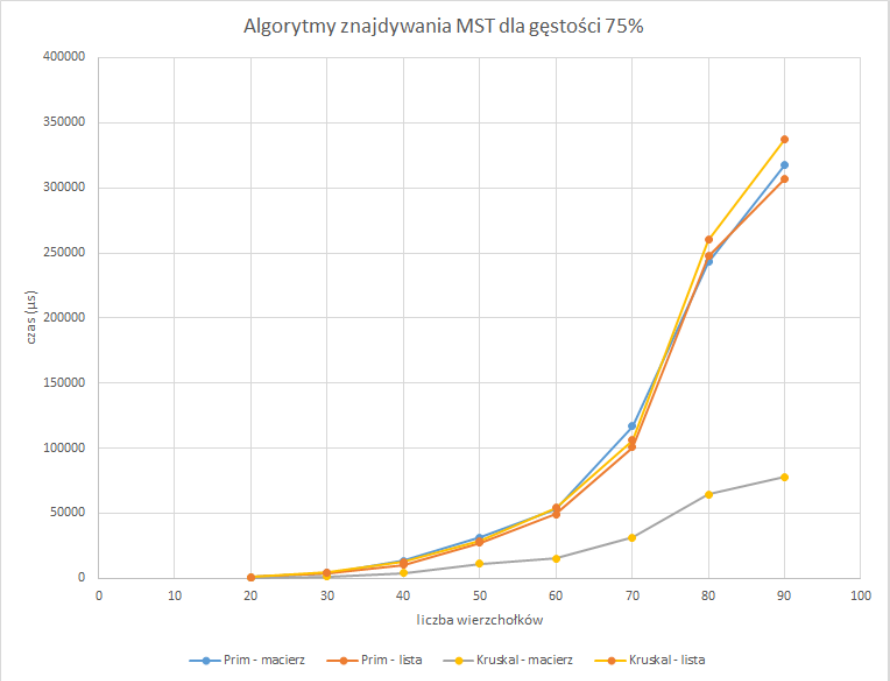
3.3 Porównanie algorytmów

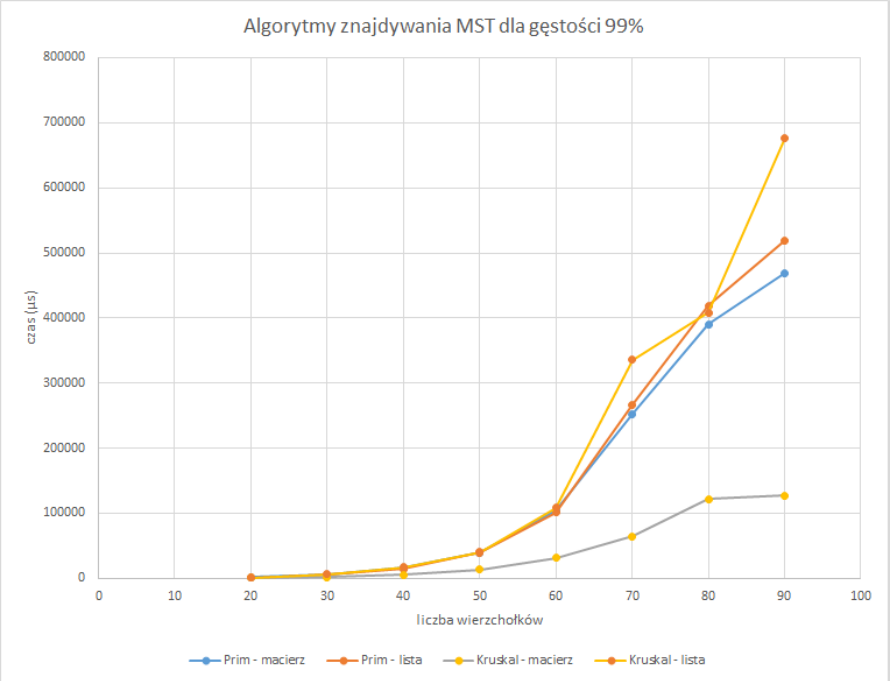



Algorytmy znajdywania MST dla listy sąsiedztwa
800000
700000
600000
500000
400000
300000
200000
100000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
25% - Prim
50% - Prim
75% - Prim
99% - Prim
25% - Kruskal
50% - Kruskal
75% - Kruskal
99% - Kruskal


Algorytmy znajdywania MST dla gęstości 25%
35000
30000
25000
20000
15000
10000
5000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
Prim - macierz
Prim - lista
Kruskal - macierz
Kruskal - lista


Algorytmy znajdywania MST dla gęstości 50%
czas (µs)
liczba wierzchołków
Prim - macierz
Prim - lista
Kruskal - macierz
Kruskal - lista


Algorytmy znajdywania MST dla gęstości 75%
czas (µs)
liczba wierzchołków
Prim - macierz
Prim - lista
Kruskal - macierz
Kruskal - lista



4 Algorytmy uzyskiwania najkrótszych ścieżek

4.1 Algorytm Dijkstry

Algorytm Dijkstry znajduje w grafie najkrótsze ścieżki pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi, wyliczając również koszt przejścia każdej z tych ścieżek, czyli sumę wag krawędzi na ścieżce. Algorytm ma złożoność czasową O(V^2) przy wykorzystaniu wyszukiwania liniowego podczas szukania wierzchołków o najmniejszym koszcie dojścia. Czas ten może być zmniejszony dzięki wykorzystaniu kolejki priorytetowej opartej na kopcu binarnym. Wtedy w korzeniu przechowywany jest wierzchołek o najmniejszej wartości kosztu dojścia. Złożoność czasowa upraszcza się do O(V * logV) – o tyle, ile wynosi czas przywracania własności kopca.

liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	14,43478	26,13043	36,43478	35
30	41,30435	67,04348	107,6957	100,3478
40	72,82609	154,9565	237,2609	232,6957
50	110,5217	284,1304	423,3478	590,5652
60	183,7826	467,913	712,9565	1115,565
70	353,6957	731,4783	1130,783	2383,696
80	521,0435	1157,13	2491,435	2539,478
90	748,3043	1779	2951,13	4642,783

Tabela 5: Uśrednione wyniki eksperymentu dla macierzy incydencji

liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	10,13043	15,34783	13,95652	14,43478
30	16,47826	22,86957	28,78261	38,65217
40	32,6087	50,04348	46,86957	49,78261
50	38,86957	56,82609	65,04348	90,26087
60	60,26087	68,86957	102	132,3043
70	68,34783	95,30435	156,5652	157,0435
80	85,08696	192	190	374,7391
90	134	163,9565	192,6087	249,6364

Tabela 6: Uśrednione wyniki eksperymentu dla listy sąsiedztwa

4.2 Algorytm Bellmana - Forda

Algorytm ten działa identycznie w kwestii obliczania kosztów dojścia jak algorytm Dijkstry. Różnica polega na tym, że nie jest wybierany nieodwiedzony wierzchołek o najmniejszym koszcie, lecz wszystkie do których w poprzedniej iteracji dochodziła krawędź. Złożoność obliczeniowa algorytmu to O(E*V).

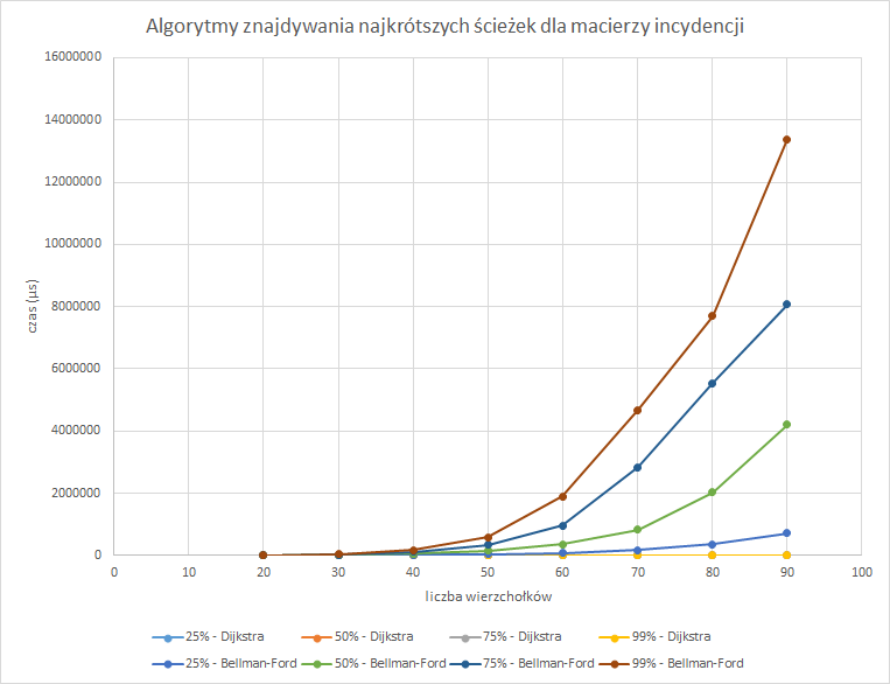
liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	379,7826	1702,826	3430,913	5273,261
30	2466,087	11653,52	26351,78	34748,61
40	11615,83	49609,3	113808,2	161849,2
50	32425	157799,3	340423	583829,2
60	74363,39	366383,9	965680,9	1889504
70	178533,8	833283,7	2832043	4681415
80	363308,5	2034613	5539939	7702463
90	721055,4	4205411	8073770	13378712

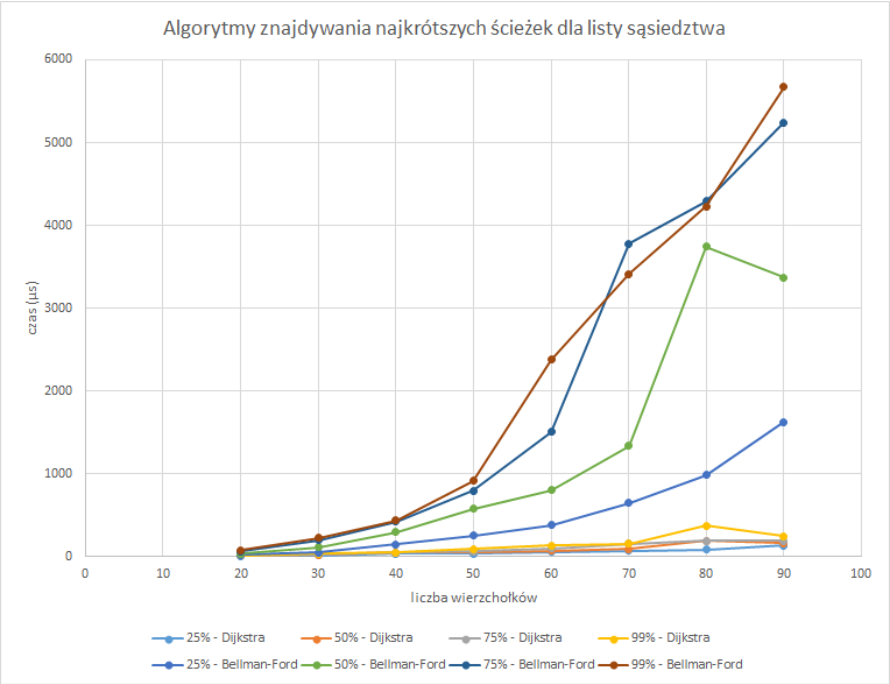
Tabela 7: Uśrednione wyniki eksperymentu dla macierzy incydencji

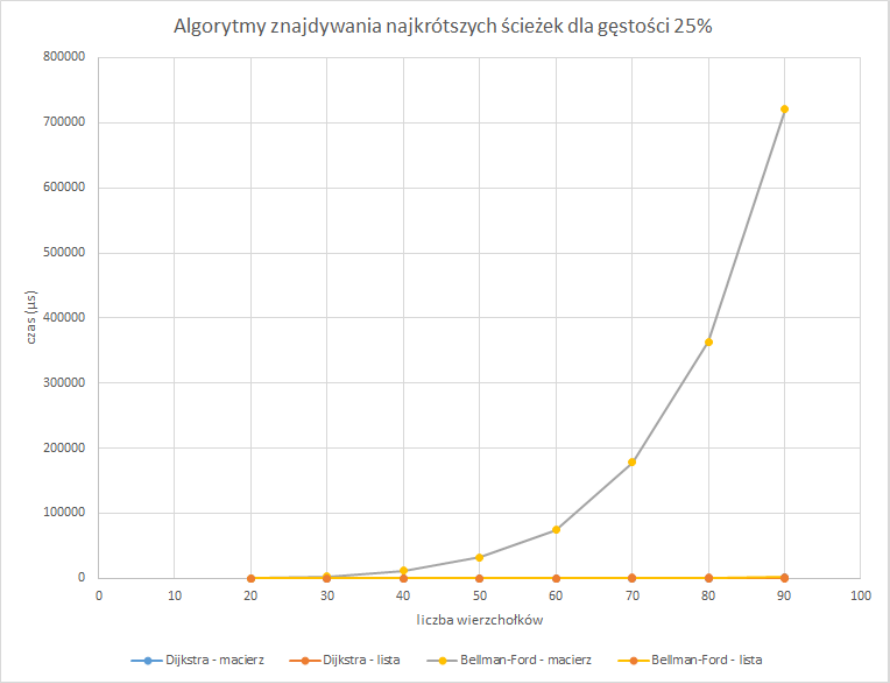
liczba wierzchołków \ gęstość (%)	25	50	75	99
	czas (μs)			
20	19,78261	35,04348	60,47826	79,04348
30	48,69565	111,6957	200,2174	224,6957
40	148,3913	296	420,913	438,9565
50	254,6522	577,3913	796,913	917,6957
60	381,4783	803,1304	1505,652	2378,304
70	650,087	1334,435	3780,217	3416,435
80	983,1304	3746,696	4295,478	4232,87
90	1622,913	3369,696	5241	5669,318

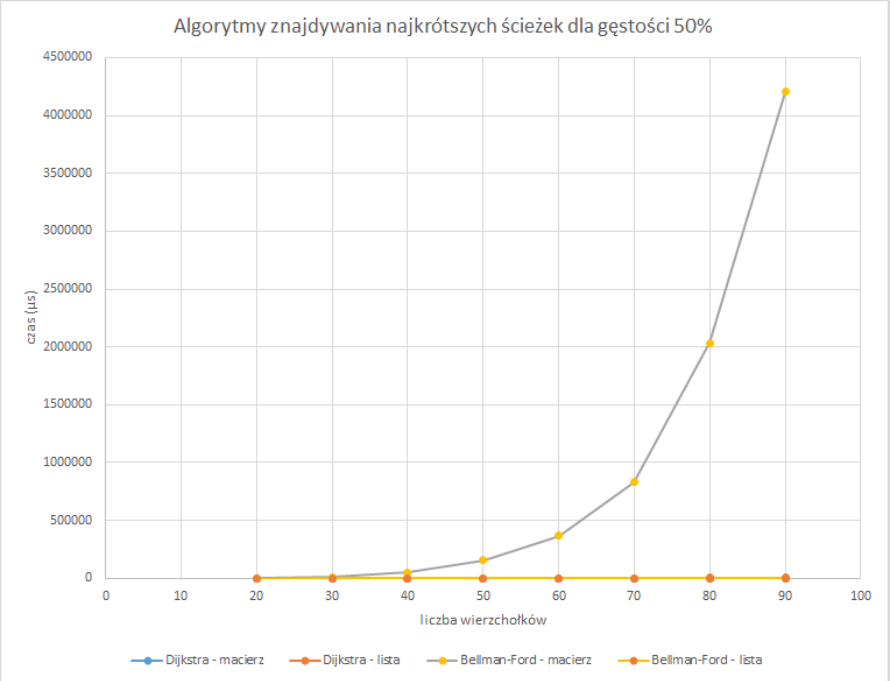
Tabela 8: Uśrednione wyniki eksperymentu dla listy sąsiedztwa

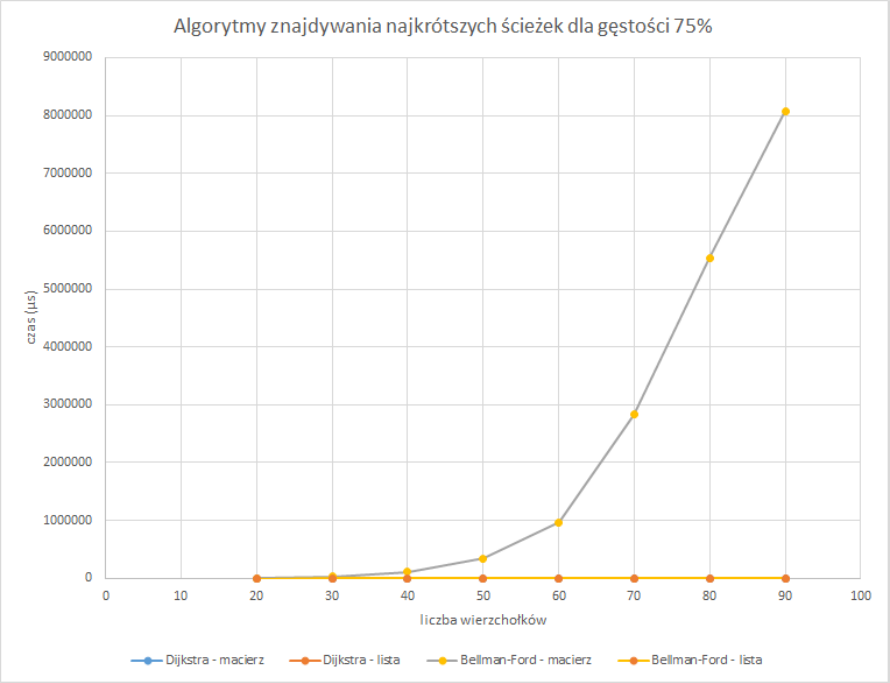
4.3 Porównanie algorytmów

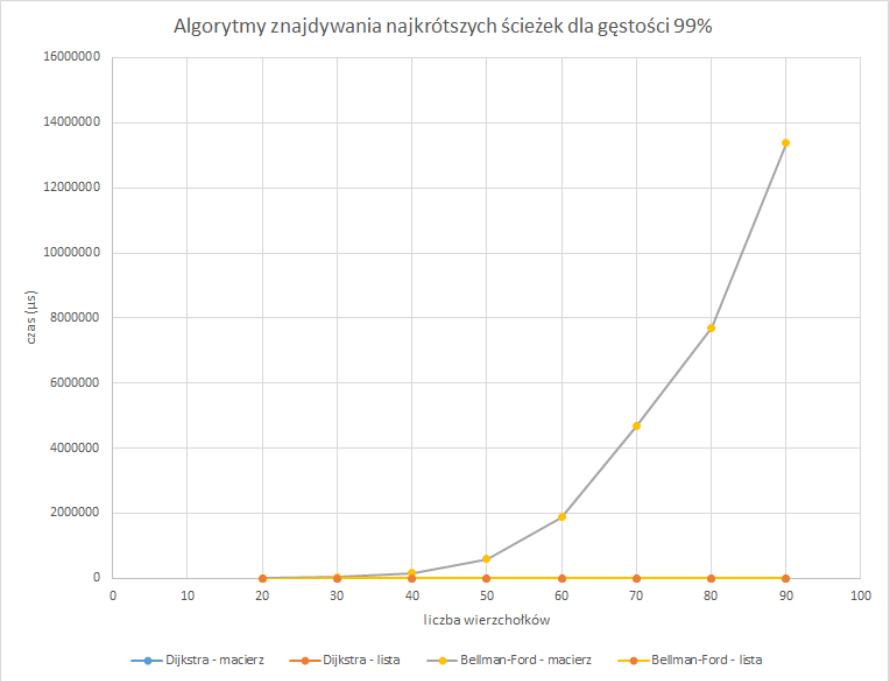





Algorytmy znajdywania najkrótszych ścieżek dla gęstości 25%
800000
700000
600000
500000
400000
300000
200000
100000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
Dijkstra - macierz
Dijkstra - lista
Bellman-Ford - macierz
Bellman-Ford - lista


Algorytmy znajdywania najkrótszych ścieżek dla gęstości 50%
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
Dijkstra - macierz
Dijkstra - lista
Bellman-Ford - macierz
Bellman-Ford - lista


Algorytmy znajdywania najkrótszych ścieżek dla gęstości 75%
9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
Dijkstra - macierz
Dijkstra - lista
Bellman-Ford - macierz
Bellman-Ford - lista


Algorytmy znajdywania najkrótszych ścieżek dla gęstości 99%
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
czas (µs)
0
10
20
30
40
50
60
70
80
90
100
liczba wierzchołków
Dijkstra - macierz
Dijkstra - lista
Bellman-Ford - macierz
Bellman-Ford - lista

5 Wnioski

Obie reprezentacje dały zbliżone wyniki dla algorytmu Prima.

Dla algorytmu Kruskala widać znaczną przewagę stosowania macierzy incydencji, jako, że dana krawędź nie powtarza się w niej dwukrotnie w grafie nieskierowanym (w przeciwieństwie do listy sąsiedztwa, gdzie krawędź ta pojawi się w obu listach odpowiednich wierzchołków).

W przypadku algorytmu Bellmana-Forda pomiary dla macierzy incydencji dawały gorsze rezultaty niż w przypadku listy sąsiedztwa. Prawdopodobnie jest to spowodowane koniecznością sprawdzania wielu pustych miejsc w macierzy przechowującej dany graf. Aby dostać wierzchołki: docelowy i początkowy należy przeiterować po całej kolumnie danej krawędzi, co sprowadza się do sprawdzenia V wartości (gdzie V to liczba wierzchołków) dla każdej krawędzi. Jest to zaś wykonywane E * (V-1) razy, jako że trzeba przejrzeć wszystkie krawędzie V-1 razy.

Również algorytm Dijkstry wypadł lepiej dla reprezentacji w postaci listy sąsiedztwa, z tego samego powodu. Jednak algorytm Dijkstry bazuje na oznaczaniu odwiedzanego danego wierzchołka, algorytm nie wykonuje się aż tyle razy.

6 Bibliografia

1. "Wprowadzenie do algorytmów" Clifford Stein, Ron Rivest i Thomas H. Cormen
2. `https://eduinf.waw.pl/inf/alg/001_search/0125.php`