STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

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

Michał Salamon 259167

1. Wprowadzenie teoretyczne:

Mój projekt zawiera implementacje grafów w postaci macierzy incydencji oraz listy sąsiedztwa. Na tych strukturach wykonywane są algorytmy grafowe, takie jak wyznaczanie minimalnego drzewa rozpinającego (MST) oraz wyznaczanie najkrótszej ścieżki w grafie.

Minimalne drzewo rozpinające (MST) - drzewo, które zawiera wszystkie wierzchołki grafu *G*, zaś zbiór krawędzi drzewa jest podzbiorem zbioru krawędzi danego grafu o najmniejszej z możliwych wag, tj. takie, że nie istnieje dla tego grafu inne drzewo rozpinające o mniejszej sumie wag krawędzi.

Do wyznaczania MST używam dwóch algorytmów:

- Algorytmu Prima
- Algorytmu Kruskala

Algorytm Prima - algorytm zachłanny wyznaczający tzw. minimalne drzewo rozpinające (MST). Mając do dyspozycji graf nieskierowany i spójny, tzn. taki w którym krawędzie grafu nie mają ustalonego kierunku oraz dla każdych dwóch wierzchołków grafu istnieje droga pomiędzy nimi, algorytm oblicza podzbiór E' zbioru krawędzi E, dla którego graf nadal pozostaje spójny, ale suma kosztów wszystkich krawędzi zbioru E' jest najmniejsza możliwa.

Złożoność obliczeniowa

Złożoność obliczeniowa w zależności od implementacji kolejki priorytetowej:

- Dla wersji opartej na zwykłym kopcu (bądź drzewie czerwono-czarnym) O(|E| * log|V|).
- Przy zastosowaniu kopca Fibonacciego O(|E| + |V|* log|V|).

Schemat działania:

- Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.
- Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MST (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do danego wierzchołka z MDR.
- Powtarzaj, dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
    - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MDR jest najmniejszy.
    - dodaj do obecnego MDR wierzchołek i krawędź realizującą najmniejszy koszt
    - zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Algorytm Kruskala - algorytm grafowy wyznaczający minimalne drzewo rozpinające dla grafu nieskierowanego ważonego, o ile jest on spójny. Innymi słowy, znajduje drzewo zawierające wszystkie wierzchołki grafu, którego waga jest najmniejsza możliwa. Jest to przykład algorytmu zachłannego.

Złożoność obliczeniowa:

Jako zbiór E można wziąć tablicę wszystkich krawędzi posortowaną według wag. Wtedy w każdym kroku najmniejsza krawędź to po prostu następna w kolejności. Sortowanie działa w

$$O(E * \log E) = O(E * \log V)$$

czasie (ponieważ $E < V^2$ zatem $E < 2 * \log V$).

Drugą fazę algorytmu można zrealizować przy pomocy struktury zbiorów rozłącznych – na początku każdy wierzchołek tworzy osobny zbiór, struktura pozwala na sprawdzenie, czy dwa wierzchołki są w jednym zbiorze i ewentualne połączenie dwu zbiorów w jeden. Przy implementacji przez tzw. *las drzew rozłącznych z kompresją ścieżki* operacje te łącznie działają w czasie $O(E * \alpha(E,V))$ gdzie α jest niezwykle wolno rosnącą funkcją (odwrotnością funkcji Ackermanna).

Całkowity czas działania jest zatem $O(E * \log V)$ ze względu na pierwszą fazę – sortowanie. Jeśli wagi krawędzi są już na wejściu posortowane, albo pozwalają na użycie szybszych algorytmów sortowania (na przykład sortowania przez zliczanie), algorytm działa w czasie $O(E * \alpha(E,V))$.

Schemat działania:

- Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
- Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
- Dopóki S nie jest pusty oraz L nie jest jeszcze drzewem rozpinającym:
    - Wybierz i usuń z S jedną z krawędzi o minimalnej wadze.
    - Jeśli krawędź ta łączyła dwa różne drzewa, to dodaj ją do lasu L, tak aby połączyła dwa odpowiadające drzewa w jedno.
    - W przeciwnym wypadku odrzuć ją.

Po zakończeniu algorytmu L jest minimalnym drzewem rozpinającym.

Algorytm Dijkstry - Mając dany graf z wyróżnionym wierzchołkiem (*źródłem*) algorytm znajduje odległości od źródła do wszystkich pozostałych wierzchołków. Łatwo zmodyfikować go tak, aby szukał wyłącznie (najkrótszej) ścieżki do jednego ustalonego wierzchołka, po prostu przerywając działanie w momencie dojścia do wierzchołka docelowego, bądź transponując tablicę incydencji grafu.

Algorytm Dijkstry znajduje w grafie wszystkie najkrótsze ścieżki pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi, przy okazji wyliczając również koszt przejścia każdej z tych ścieżek.

Algorytm Dijkstry jest przykładem algorytmu zachłannego.

Złożoność obliczeniowa:

Złożoność obliczeniowa algorytmu Dijkstry zależy od liczby *V* wierzchołków i *E* krawędzi grafu. O rzędzie złożoności decyduje implementacja kolejki priorytetowej:

- wykorzystując „naiwną" implementację poprzez zwykłą tablicę, otrzymujemy algorytm o złożoności O(V^2)
- w implementacji kolejki poprzez kopiec, złożoność wynosi O(E * logV)
- po zastąpieniu zwykłego kopca kopcem Fibonacciego złożoność zmienia się na O(E + V * logV)

Pierwszy wariant jest optymalny dla grafów gęstych (czyli jeśli E = O(V^2)), drugi jest szybszy dla grafów rzadkich E = O(V) trzeci jest bardzo rzadko używany ze względu na duży stopień skomplikowania i niewielki w porównaniu z nim zysk czasowy.

Schemat działania:

Przez *s* oznaczamy wierzchołek źródłowy, *w(i ,j)* to waga krawędzi *(i ,j)* w grafie.

- Stwórz tablicę *d* odległości od źródła dla wszystkich wierzchołków grafu. Na początku *d[s] = 0* zaś *d[s] =* ∞ dla wszystkich pozostałych wierzchołków.
- Utwórz kolejkę priorytetową *Q* wszystkich wierzchołków grafu. Priorytetem kolejki jest aktualnie wyliczona odległość od wierzchołka źródłowego *s*
- Dopóki kolejka nie jest pusta:
    - Usuń z kolejki wierzchołek *u* o najniższym priorytecie (wierzchołek najbliższy źródła, który nie został jeszcze rozważony)
    - Dla każdego sąsiada *v* wierzchołka *u* dokonaj relaksacji poprzez *u:* jeśli *d[s] + w(u, v) < d[v]* (poprzez *u* da się dojść *v* do szybciej niż dotychczasową ścieżką), to *d[v] := d[u] + w(u, v).*

Na końcu tablica *d* zawiera najkrótsze odległości do wszystkich wierzchołków.

Dodatkowo możemy w tablicy *poprzednik* przechowywać dla każdego wierzchołka numer jego bezpośredniego poprzednika na najkrótszej ścieżce, co pozwoli na odtworzenie pełnej ścieżki od źródła do każdego wierzchołka – przy każdej relaksacji w ostatnim punkcie *u* staje się poprzednikiem *v.*

Algorytm Bellmana-Forda - algorytm służący do wyszukiwania najkrótszych ścieżek w grafie ważonym z wierzchołka źródłowego do wszystkich pozostałych wierzchołków.

Idea algorytmu opiera się na metodzie relaksacji (dokładniej następuje relaksacja |V| - 1 razy każdej z krawędzi).

W odróżnieniu od algorytmu Dijkstry, algorytm Bellmana-Forda działa poprawnie także dla grafów z wagami ujemnymi (nie może jednak wystąpić cykl o łącznej ujemnej wadze osiągalny ze źródła). Za tę ogólność płaci się jednak wyższą złożonością czasową. Działa on w czasie O(|V| * |E|).

Algorytm może być wykorzystywany także do sprawdzania, czy w grafie występują ujemne cykle osiągalne ze źródła.

Lista sąsiedztwa - W teorii grafów i informatyce lista sąsiedztwa to zbiór nieuporządkowanych list służących do reprezentowania grafu skończonego. Każda nieuporządkowana lista na liście sąsiedztwa opisuje zbiór sąsiadów określonego wierzchołka w grafie. Jest to jedna z kilku powszechnie używanych reprezentacji grafów do wykorzystania w programach komputerowych.

Macierz Sąsiedztwa - W teorii grafów i informatyce macierz sąsiedztwa to macierz kwadratowa używana do reprezentowania grafu skończonego. Elementy tej macierzy wskazują, czy pary wierzchołków w grafie są sąsiadujące, czy nie. Jeśli graf jest nieskierowany (tzn. wszystkie jego krawędzie są dwukierunkowe), to macierz sąsiedztwa jest symetryczna. W spektralnej teorii grafów bada się związek między grafem a wartościami własnymi i wektorami własnymi jego macierzy sąsiedztwa. Macierz sąsiedztwa grafu należy odróżnić od macierzy incydencji - innej reprezentacji macierzy, której elementy wskazują, czy pary wierzchołek-krawędź są incydentne, czy nie, oraz od macierzy stopni, która zawiera informacje o stopniu każdego wierzchołka.

Testy wykonywane są na grafach o liczbie wierzchołków: 25, 50, 100, 150, 200. Zmienna jest także gęstość grafów testowanych: 25%, 50%, 75%, 99%. Test wykonuje 100 pomiarów czasów na każdym algorytmie, a następnie wyniki zapisywane są do plików.

Do mierzenia samego czasu używam biblioteki chrono. Na początku tworzymy 2 zmienne, przed wykonaniem jakiegoś algorytmu do 1 zmiennej pobrany zostaje aktualny czas, a po zakończeniu do 2. Następnie oblicza się z ich różnicy czas działania funkcji.

Graf generowany jest przy pomocy parametrów wpisanych przez użytkownika lub wczytany z pliku. Podawane są: liczba krawędzi, gęstość oraz czy graf ma być skierowany czy nie. W zależności od liczby wierzchołków oraz skierowania grafu użytkownik musi podać inną gęstość. Gęstością grafu nazywamy stosunek liczby krawędzi do największej możliwej liczby krawędzi. Dla grafów nieskierowanych wzór ten jest w postaci:

$$D = \frac{2|E|}{|V| \cdot (|V| - 1)}$$

Dla grafów skierowanych:

$$D = \frac{|E|}{|V| \cdot (|V| - 1)}$$

2. Tabele z uśrednionymi wynikami:

Gęstość 25%:

Lista 25 wierzchołków	
metoda	średni czas [ms]
Prim	13,69
Kruskal	25,97
Dijkstra	14,07
Bellman-Ford	15,68

Macierz 25 wierzchołków	
metoda	średni czas [ms]
Prim	15,96
Kruskal	27,3
Dijkstra	17,2
Bellman-Ford	38,43

Lista 50 wierzchołków	
metoda	średni czas [ms]
Prim	56,88
Kruskal	167,4
Dijkstra	58,17
Bellman-Ford	125,27

Macierz 50 wierzchołków	
metoda	średni czas [ms]
Prim	64,44
Kruskal	169,39
Dijkstra	70,9
Bellman-Ford	301,8

Lista 100 wierzchołków	
metoda	średni czas [ms]
Prim	305,21
Kruskal	1664,34
Dijkstra	303,47
Bellman-Ford	1228,96

Macierz 100 wierzchołków	
metoda	średni czas [ms]
Prim	305,16
Kruskal	1668,14
Dijkstra	332,86
Bellman-Ford	2255,08

Lista 150 wierzchołków	
metoda	średni czas [ms]
Prim	910,4
Kruskal	7523,6
Dijkstra	845,41
Bellman-Ford	5124,58

Macierz 150 wierzchołków	
metoda	średni czas [ms]
Prim	827,2
Kruskal	7464,4
Dijkstra	956,17
Bellman-Ford	7249,59

Lista 200 wierzchołków	
metoda	średni czas [ms]
Prim	2231,75
Kruskal	25892,89
Dijkstra	2005,1
Bellman-Ford	20569,93

Macierz 200 wierzchołków	
metoda	średni czas [ms]
Prim	2071,79
Kruskal	25493,67
Dijkstra	2081,6
Bellman-Ford	18173,11

Gęstość 50%:

Lista 25 wierzchołków	
metoda	średni czas [ms]
Prim	16,6
Kruskal	48,06
Dijkstra	17,05
Bellman-Ford	25,89

Macierz 25 wierzchołków	
metoda	średni czas [ms]
Prim	18,63
Kruskal	47,76
Dijkstra	18,4
Bellman-Ford	41,03

Lista 50 wierzchołków	
metoda	średni czas [ms]
Prim	84,3
Kruskal	453,87
Dijkstra	78,36
Bellman-Ford	292,99

Macierz 50 wierzchołków	
metoda	średni czas [ms]
Prim	86,78
Kruskal	451,37
Dijkstra	87,6
Bellman-Ford	350,51

Lista 100 wierzchołków	
metoda	średni czas [ms]
Prim	542,16
Kruskal	5957,08
Dijkstra	460,89
Bellman-Ford	2770,01

Macierz 100 wierzchołków	
metoda	średni czas [ms]
Prim	453,03
Kruskal	5898,28
Dijkstra	493,31
Bellman-Ford	2673,36

Lista 150 wierzchołków	
metoda	średni czas [ms]
Prim	1617,92
Kruskal	27879,16
Dijkstra	1386,11
Bellman-Ford	16254,04

Macierz 150 wierzchołków	
metoda	średni czas [ms]
Prim	1245,09
Kruskal	28576,63
Dijkstra	1517,05
Bellman-Ford	8704,7

Lista 200 wierzchołków	
metoda	średni czas [ms]
Prim	3874,84
Kruskal	96651,12
Dijkstra	4716,55
Bellman-Ford	49347,79

Macierz 200 wierzchołków	
metoda	średni czas [ms]
Prim	2772,87
Kruskal	94262,24
Dijkstra	3665,02
Bellman-Ford	22347,45

Gęstość 75%:

Lista 25 wierzchołków	
metoda	średni czas [ms]
Prim	19,96
Kruskal	77,08
Dijkstra	19,16
Bellman-Ford	38,31

Macierz 25 wierzchołków	
metoda	średni czas [ms]
Prim	20,3
Kruskal	76,41
Dijkstra	21,85
Bellman-Ford	44,53

Lista 50 wierzchołków	
metoda	średni czas [ms]
Prim	113,17
Kruskal	884,59
Dijkstra	99,41
Bellman-Ford	442,24

Macierz 50 wierzchołków	
metoda	średni czas [ms]
Prim	152,59
Kruskal	885,38
Dijkstra	105,21
Bellman-Ford	400,8

Lista 100 wierzchołków	
metoda	średni czas [ms]
Prim	774,74
Kruskal	13003,21
Dijkstra	626,65
Bellman-Ford	5922,09

Macierz 100 wierzchołków	
metoda	średni czas [ms]
Prim	585,08
Kruskal	12656,87
Dijkstra	622,37
Bellman-Ford	3131,37

Lista 150 wierzchołków	
metoda	średni czas [ms]
Prim	2185,2
Kruskal	61964,73
Dijkstra	1948,78
Bellman-Ford	27090,23

Macierz 150 wierzchołków	
metoda	średni czas [ms]
Prim	1658,03
Kruskal	62016,96
Dijkstra	1848,16
Bellman-Ford	10035,57

Lista 200 wierzchołków	
metoda	średni czas [ms]
Prim	6118,62
Kruskal	223302,33
Dijkstra	4556,65
Bellman-Ford	78287,59

Macierz 200 wierzchołków	
metoda	średni czas [ms]
Prim	3817,11
Kruskal	218560,8
Dijkstra	4392,35
Bellman-Ford	26092,14

Gęstość 99%:

Lista 25 wierzchołków	
metoda	średni czas [ms]
Prim	22,67
Kruskal	115,95
Dijkstra	23,16
Bellman-Ford	51,37

Macierz 25 wierzchołków	
metoda	średni czas [ms]
Prim	21,92
Kruskal	111,18
Dijkstra	22,71
Bellman-Ford	51,81

Lista 50 wierzchołków	
metoda	średni czas [ms]
Prim	142,47
Kruskal	1489,35
Dijkstra	133,64
Bellman-Ford	590,98

Macierz 50 wierzchołków	
metoda	średni czas [ms]
Prim	118,02
Kruskal	1471,5
Dijkstra	119,12
Bellman-Ford	458,27

Lista 100 wierzchołków	
metoda	średni czas [ms]
Prim	842,13
Kruskal	21798,63
Dijkstra	859,79
Bellman-Ford	9092,95

Macierz 100 wierzchołków	
metoda	średni czas [ms]
Prim	718,76
Kruskal	21767,71
Dijkstra	724,83
Bellman-Ford	3620,93

Lista 150 wierzchołków	
metoda	średni czas [ms]
Prim	2509,15
Kruskal	106323,41
Dijkstra	2675,91
Bellman-Ford	37030,87

Macierz 150 wierzchołków	
metoda	średni czas [ms]
Prim	2110,17
Kruskal	105637,81
Dijkstra	2116,21
Bellman-Ford	11676,81

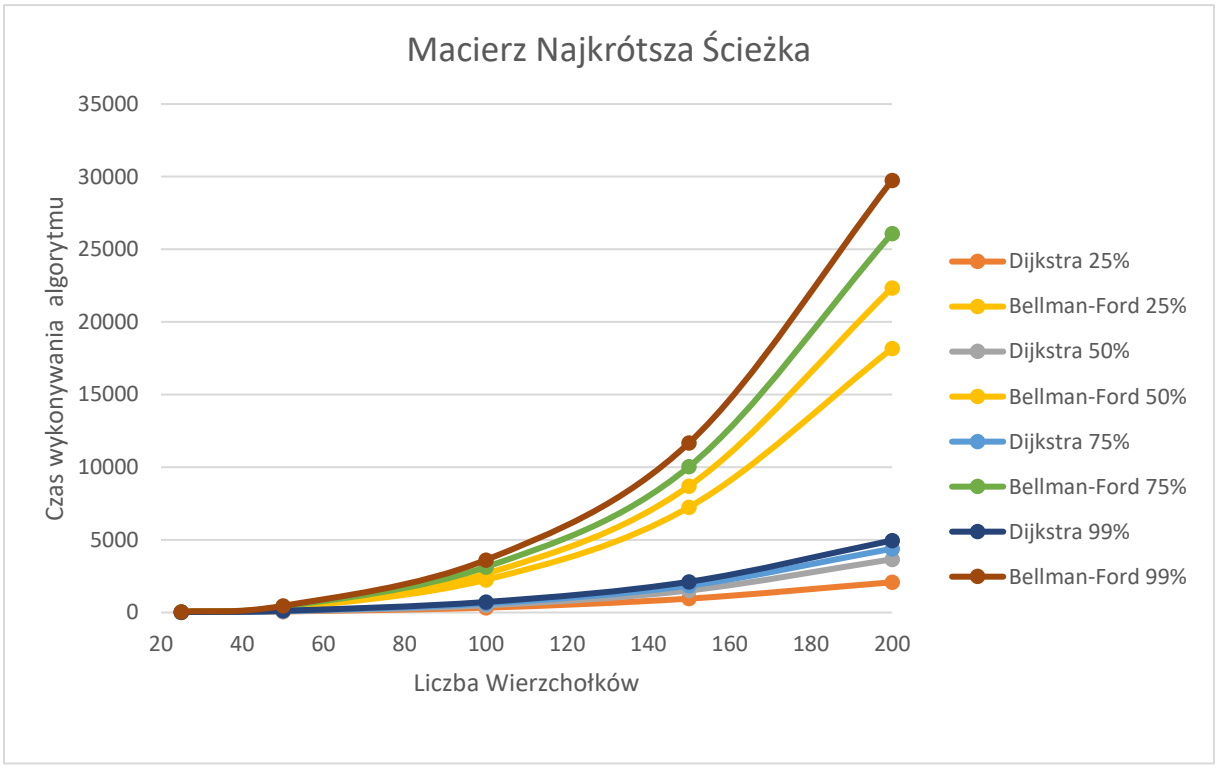
Lista 200 wierzchołków	
metoda	średni czas [ms]
Prim	6162,46
Kruskal	373561,09
Dijkstra	6983,38
Bellman-Ford	108271,25

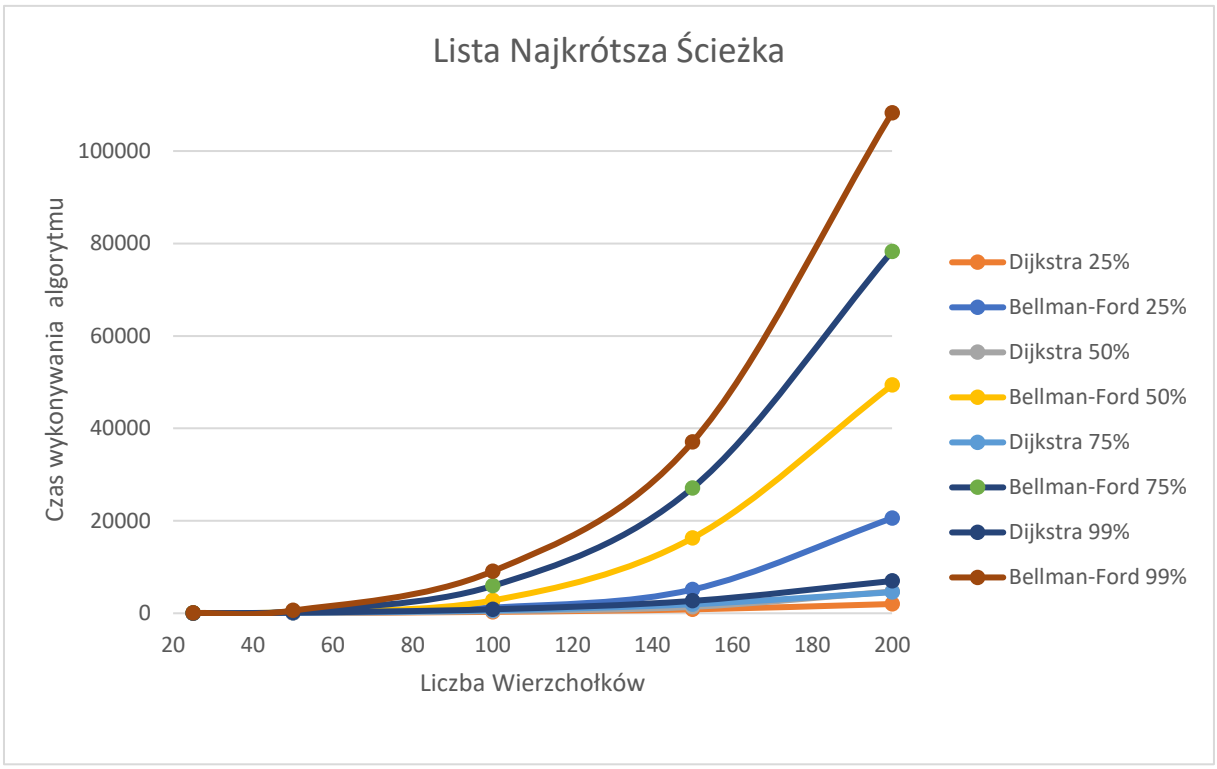
Macierz 200 wierzchołków	
metoda	średni czas [ms]
Prim	5133,99
Kruskal	371750,4
Dijkstra	4958,92
Bellman-Ford	29743,41

3. Wykresy:

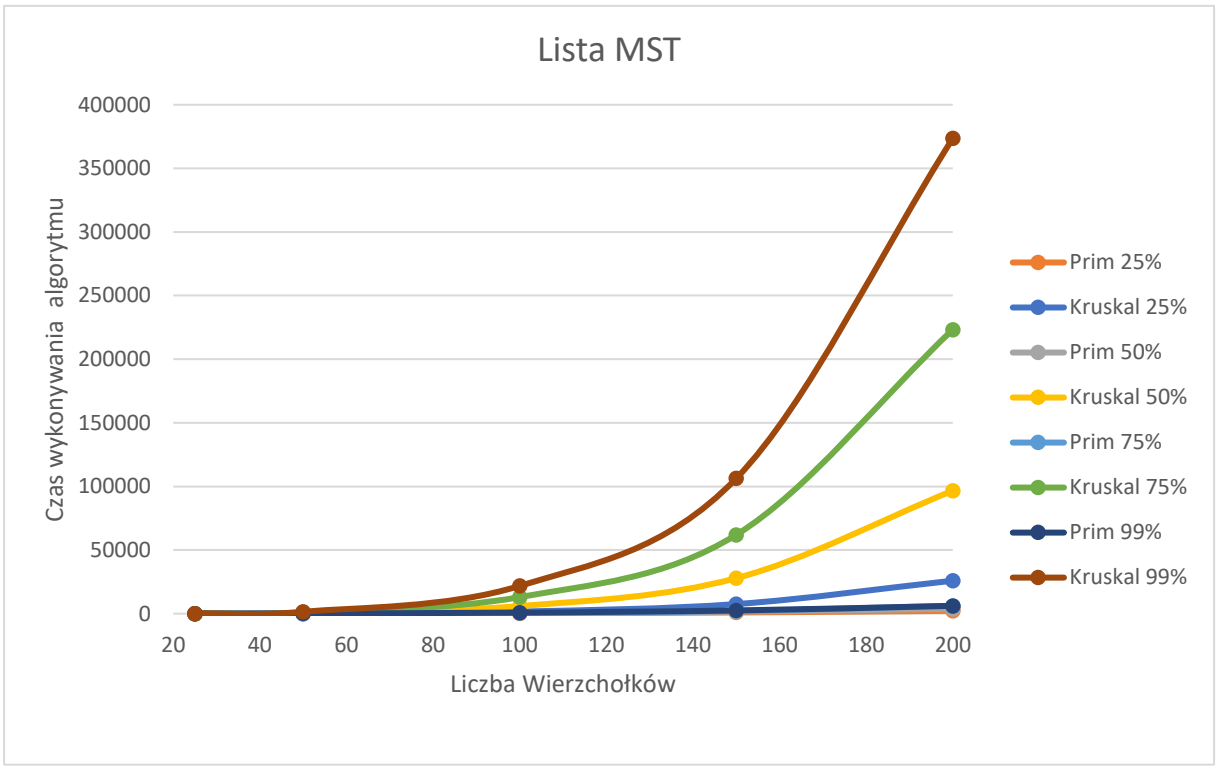
Typ 1 (osobne wykresy dla każdej reprezentacji grafu w formie linii (połączonych punktów), których parametrem jest gęstość grafu i typ algorytmu):

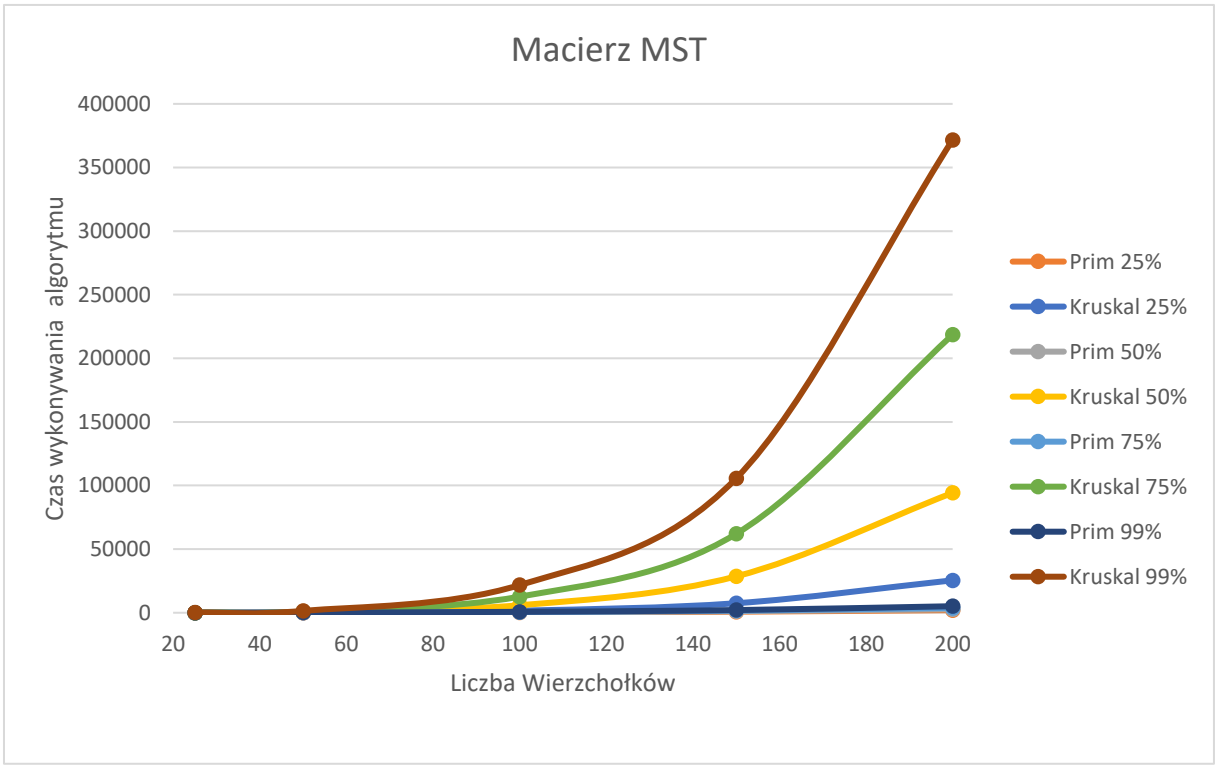
Najkrótsza ścieżka:





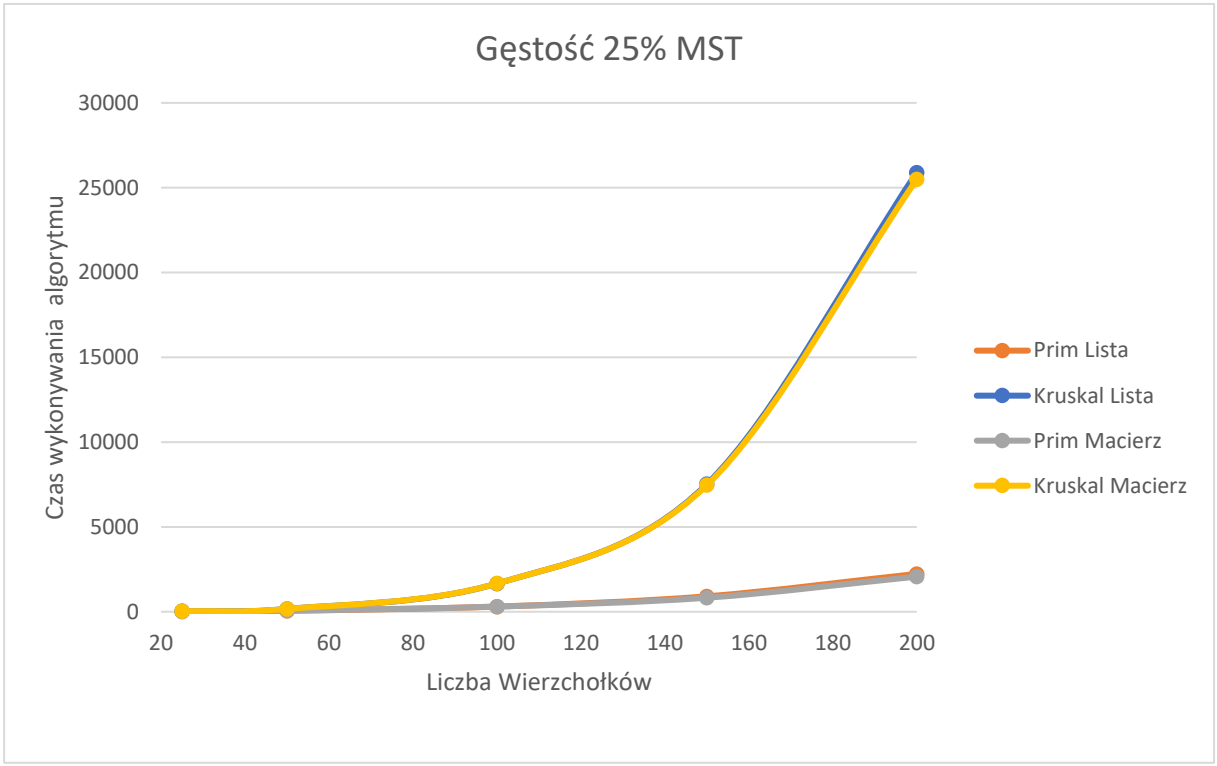
MST:

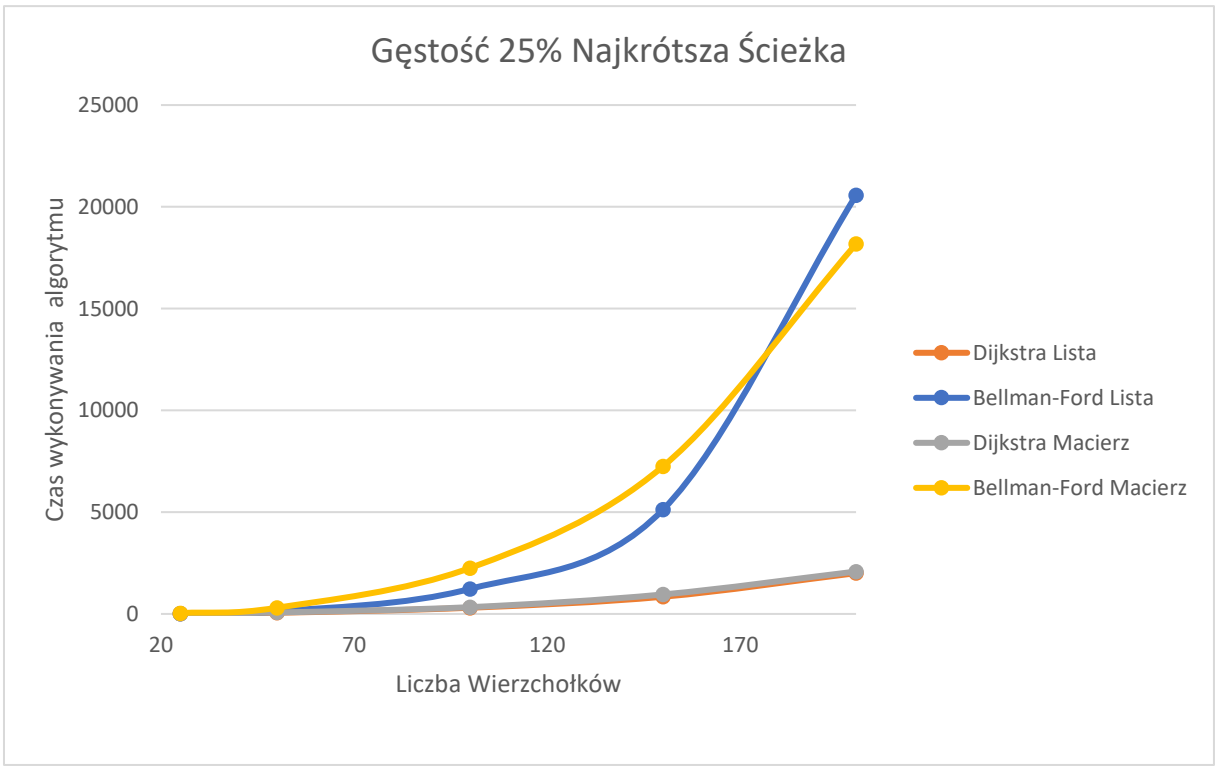

Lista MST
Czas wykonywania algorytmu
400000
350000
300000
250000
200000
150000
100000
50000
0
20
40
60
80
100
120
140
160
180
200
Liczba Wierzchołków
Prim 25%
Kruskal 25%
Prim 50%
Kruskal 50%
Prim 75%
Kruskal 75%
Prim 99%
Kruskal 99%


Macierz MST
Czas wykonywania algorytmu
400000
350000
300000
250000
200000
150000
100000
50000
0
20
40
60
80
100
120
140
160
180
200
Liczba Wierzchołków
Prim 25%
Kruskal 25%
Prim 50%
Kruskal 50%
Prim 75%
Kruskal 75%
Prim 99%
Kruskal 99%

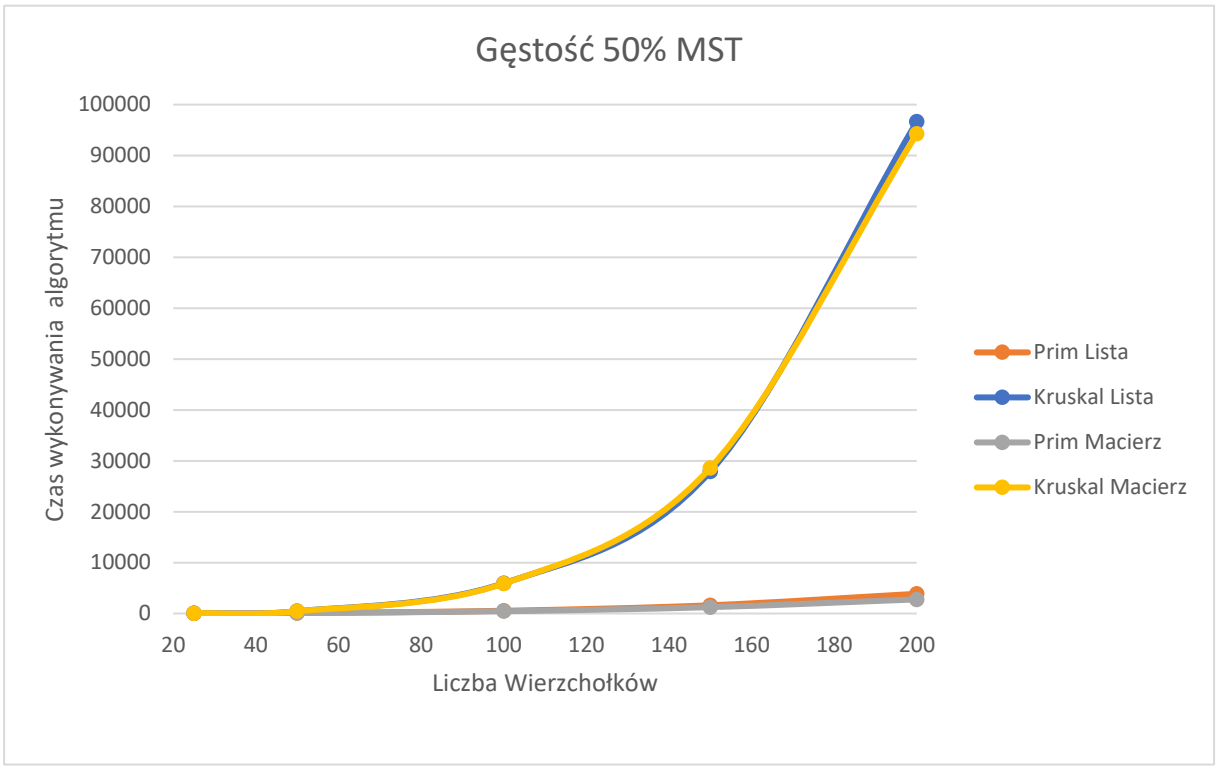
Typ 2 (osobne wykresy dla każdej gęstości grafu w formie linii których parametrem jest typ algorytmu i typ reprezentacji grafu):

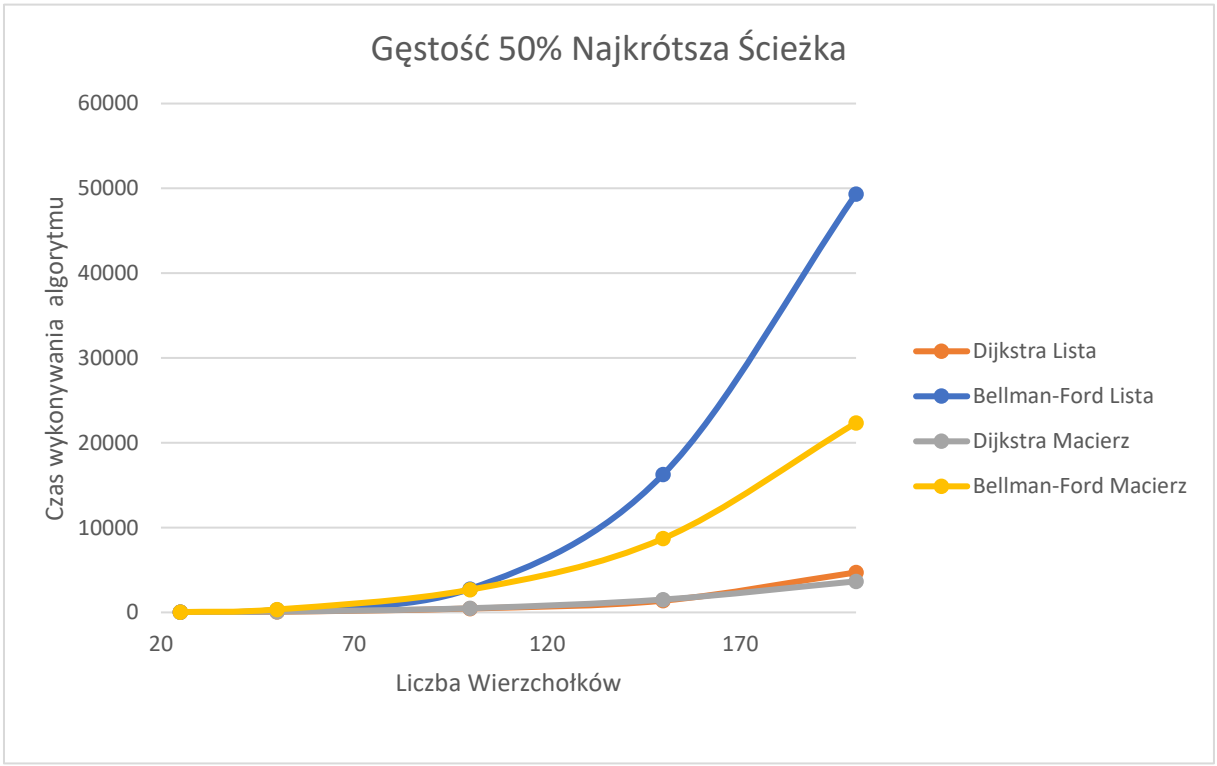
Gęstość 25%:



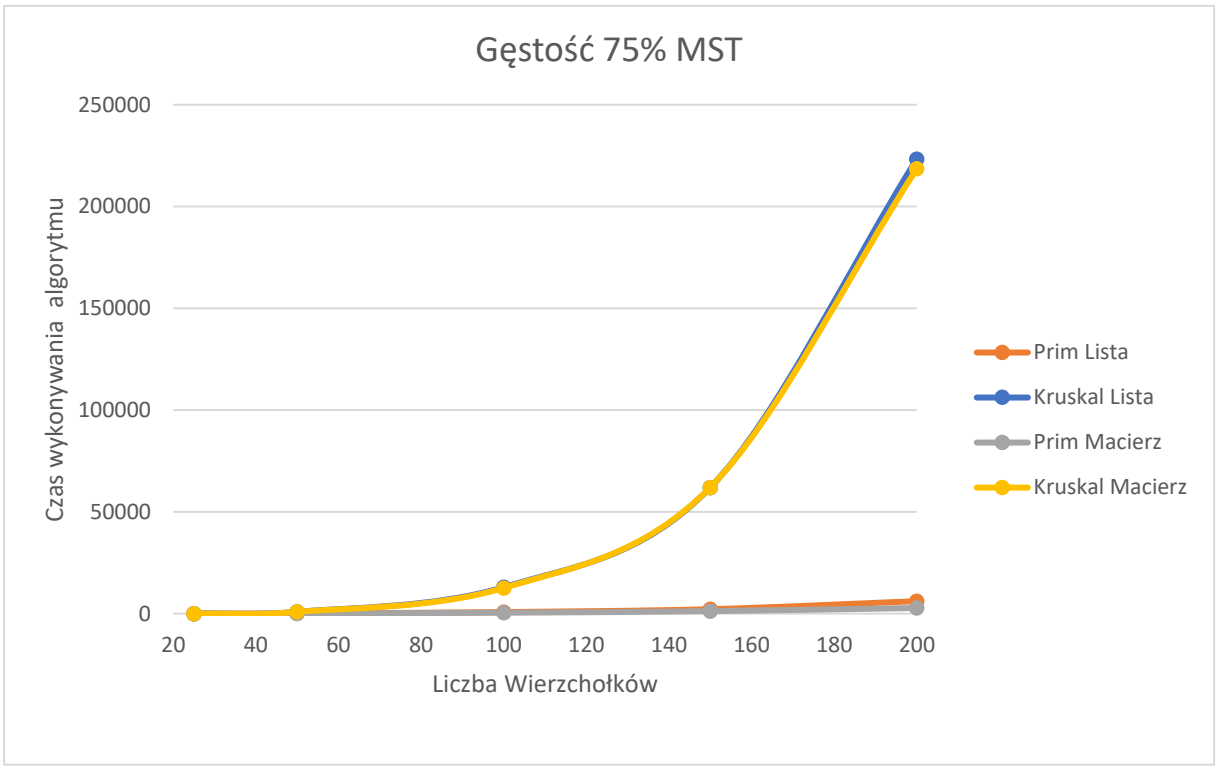


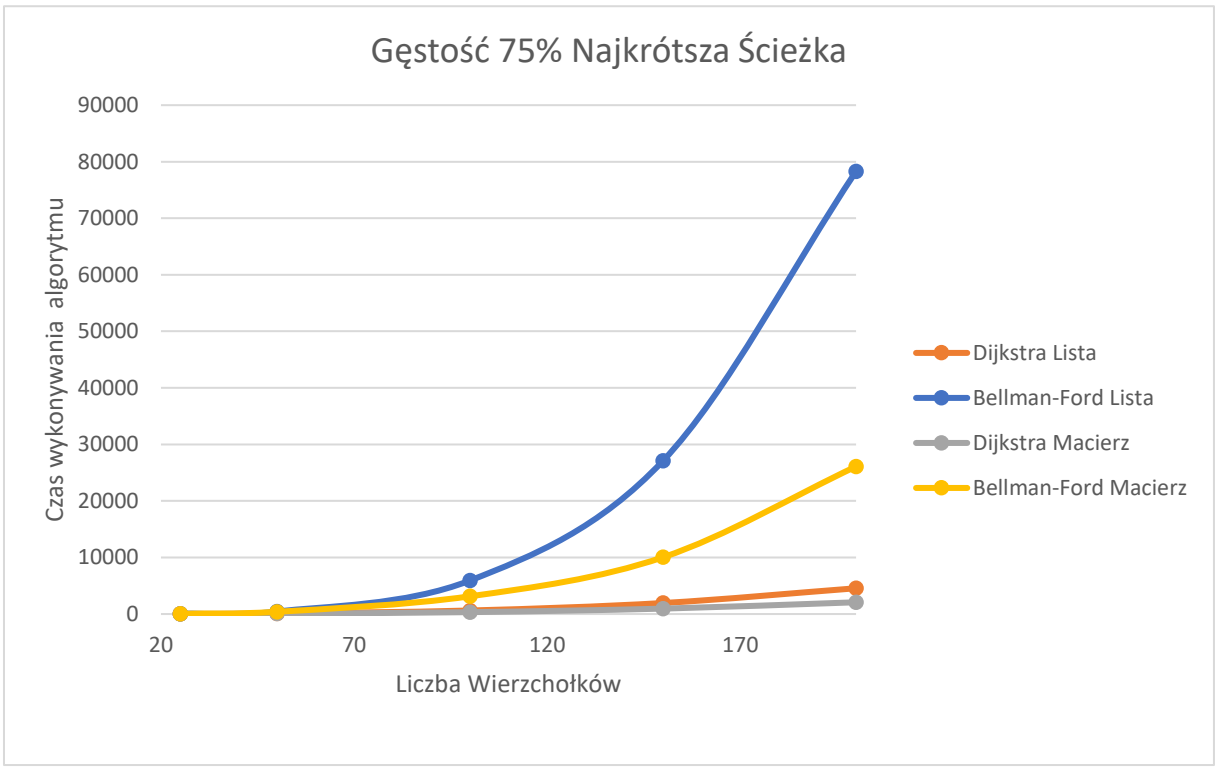
Gęstość 50%:


Gęstość 50% MST
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
Czas wykonywania algorytmu
20
40
60
80
100
120
140
160
180
200
Liczba Wierzchołków
Prim Lista
Kruskal Lista
Prim Macierz
Kruskal Macierz

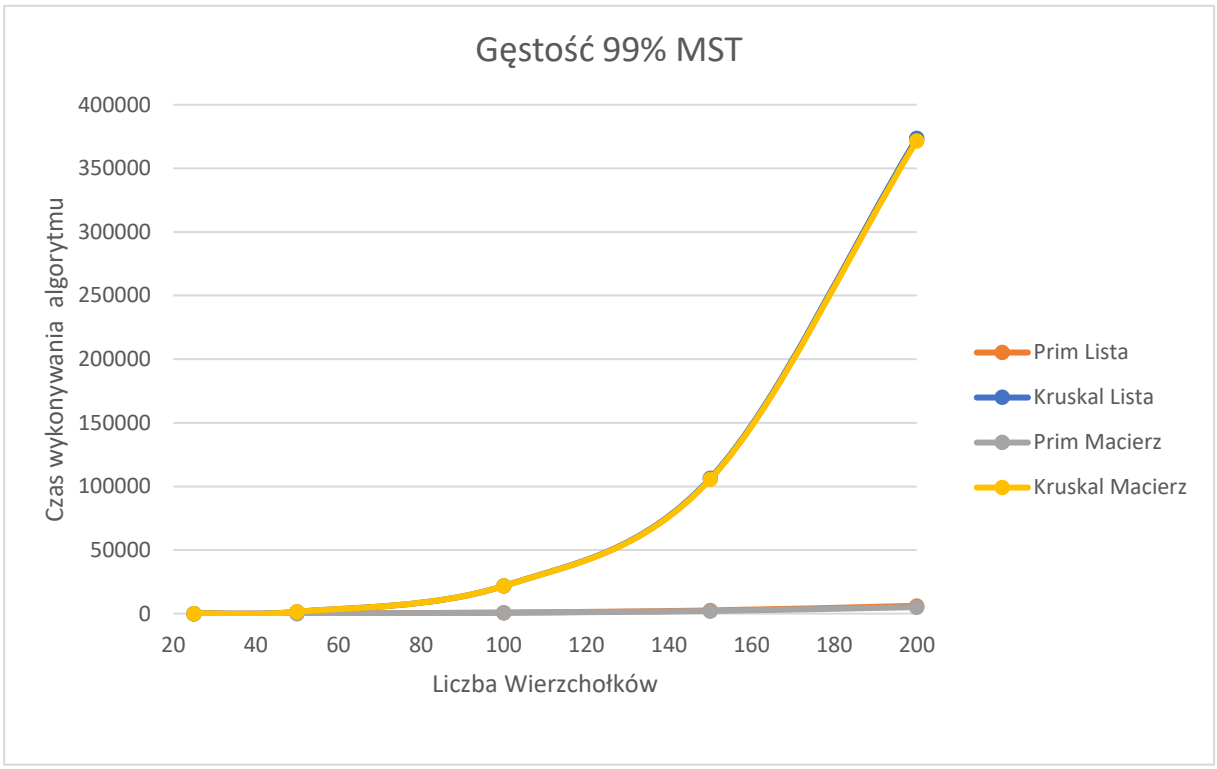

Gęstość 50% Najkrótsza Ścieżka
60000
50000
40000
30000
20000
10000
0
Czas wykonywania algorytmu
20
70
120
170
Liczba Wierzchołków
Dijkstra Lista
Bellman-Ford Lista
Dijkstra Macierz
Bellman-Ford Macierz

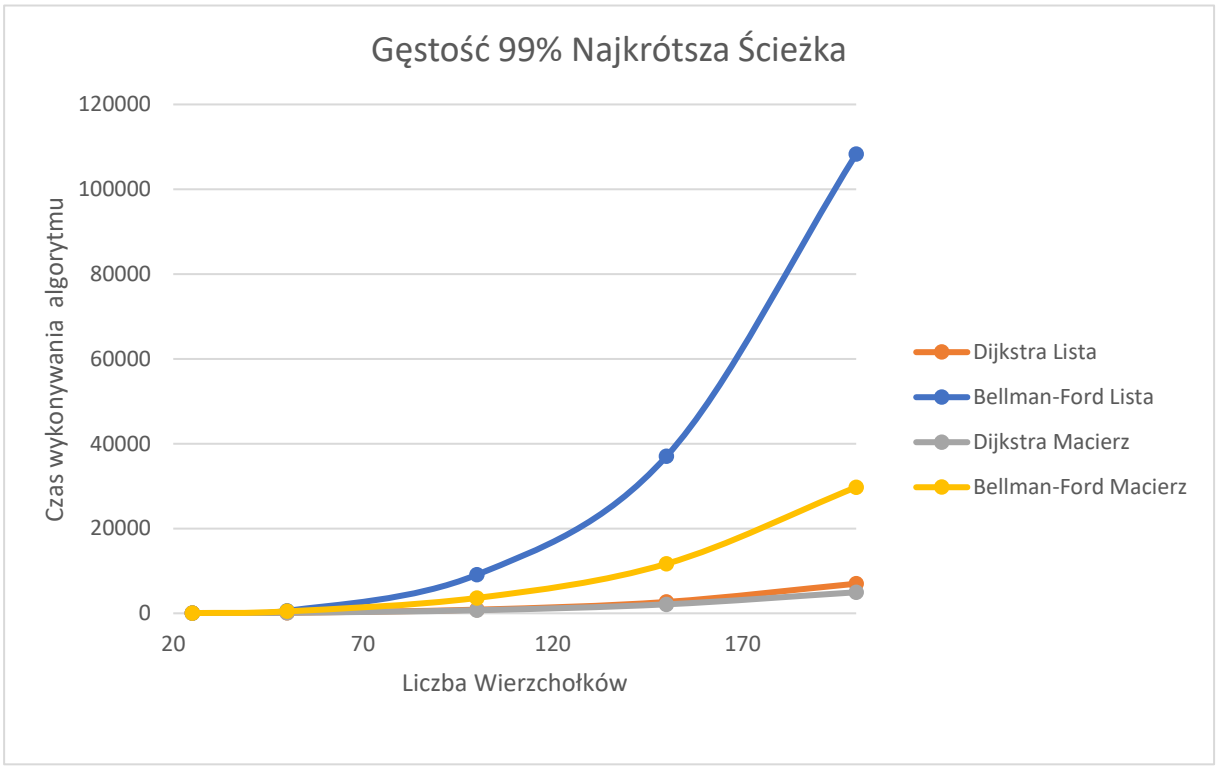
Gęstość 75%:


Gęstość 75% MST
250000
200000
150000
100000
50000
0
Czas wykonywania algorytmu
20
40
60
80
100
120
140
160
180
200
Liczba Wierzchołków
Prim Lista
Kruskal Lista
Prim Macierz
Kruskal Macierz


Gęstość 75% Najkrótsza Ścieżka
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
Czas wykonywania algorytmu
20
70
120
170
Liczba Wierzchołków
Dijkstra Lista
Bellman-Ford Lista
Dijkstra Macierz
Bellman-Ford Macierz

Gęstość 99%:


Gęstość 99% MST
400000
350000
300000
250000
200000
150000
100000
50000
0
Czas wykonywania algorytmu
20
40
60
80
100
120
140
160
180
200
Liczba Wierzchołków
Prim Lista
Kruskal Lista
Prim Macierz
Kruskal Macierz


Gęstość 99% Najkrótsza Ścieżka
120000
100000
80000
60000
40000
20000
0
Czas wykonywania algorytmu
20
70
120
170
Liczba Wierzchołków
Dijkstra Lista
Bellman-Ford Lista
Dijkstra Macierz
Bellman-Ford Macierz

4. Wnioski:

Podczas analizy wniosków z góry widać, że algorytmy poszukiwania MST są bardziej czasochłone, przy uwzględnieniu wielkości struktur, niż algorytmy wyszukiwania najkrótszej ścieżki. W poszczególnych algorytmach MST i najkrótszej ścieżki też widać duże różnice. Bellman-Ford jest dużo mniej wydajny przy większych grafach, w porównaniu do Dijkstry. Tak samo Kruskal w porównaniu do Prima dla grafów z większą ilością wieżchołków jest proporcjonalnie mniej wydajny.

Dla algorytmów najkrótszej ścieżki widać także, że macierz jest bardziej optymalny, gdyż wzraz ze wzrostem liczby wierzchołków czas rośnie wykładniczo dla implementacji listowej, a prawie liniowo dla macierzowej. Za to dla MST dużej różnicy między implementacją nie było.