STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

Zadanie projektowe nr 2: Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera

Autor: Bartosz Rodziewicz, 226105

Prowadzący: dr inż. Dariusz Banasiak

Grupa: Wtorek, TN, 7:30

1. Wstęp teoretyczny

a. Graf

Graf – podstawowy obiekt rozważań teorii grafów, struktura matematyczna służąca do przedstawiania i badania relacji między obiektami. W uproszczeniu graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków.

b. Algorytm Prima

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 tego algorytmu zależy od sposobu zaimplementowania kolejki priorytetowej.

Metoda implementacji kolejki priorytetowej	**Złożoność czasowa**
Zwykła tablica przeszukiwana po wszystkich elementach	$O(V^2)$
kopiec	$O(E \log V)$
kopiec Fibonacciego	$O(E + V \log V)$

c. Algorytm Dijkstry

Algorytm Dijkstry, opracowany przez holenderskiego informatyka Edsgera Dijkstrę, służy do znajdowania najkrótszej ścieżki [SP] z pojedynczego źródła do wszystkich wierzchołków w spójnym grafie o nieujemnych wagach krawędzi.

Złożoność obliczeniowa zależy tak samo od sposobu zaimplementowania kolejki priorytetowej.

Metoda implementacji kolejki priorytetowej	**Złożoność czasowa**
Zwykła tablica przeszukiwana po wszystkich elementach	$O(V^2)$
kopiec	$O(E \log V)$
kopiec Fibonacciego	$O(E + V \log V)$

2. Implementacja zadania

Zadanie zostało wykonane jako program obiektowy.

Grafy przechowywane są w kontenerach z biblioteki STL.

Zaimplementowane zostały dwa algorytmy:

- Algorytm Prima

- Algorytm Dijkstry

3. Metoda testowania i plan eksperymentu

Pomiar czasu dokonywany jest za pomocą obiektu QueryPerformanceCounter.

Testy wykonywane są na dwóch metodach reprezentacji grafu:

- Macierz incydencji
- Lista sąsiedztwa

Dla każdej reprezentacji generowane są losowo grafy o pięciu różnych ilościach wierzchołków:

- 50
- 100
- 150
- 200
- 250

oraz czterech gęstościach:

- 25%
- 50%
- 75%
- 99%.

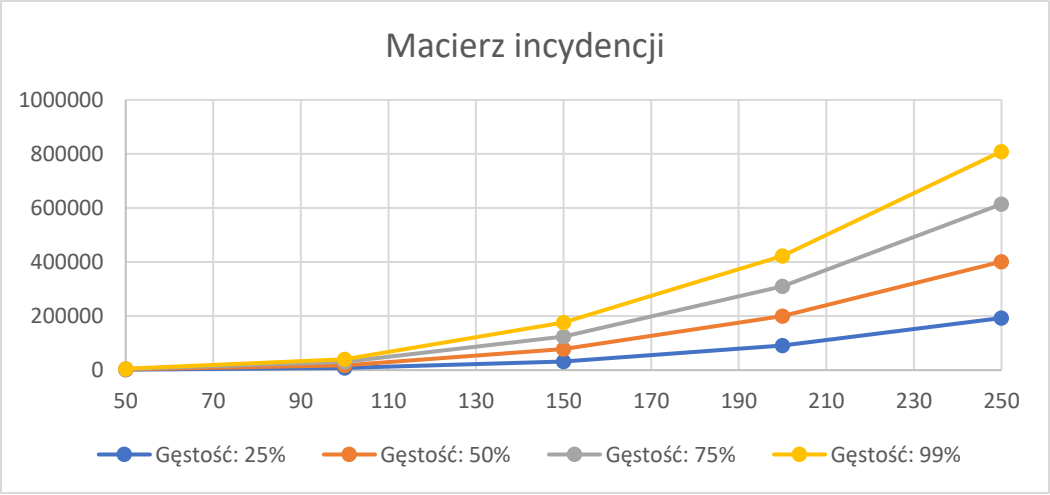
Daje to 40 kombinacji na jeden algorytm.

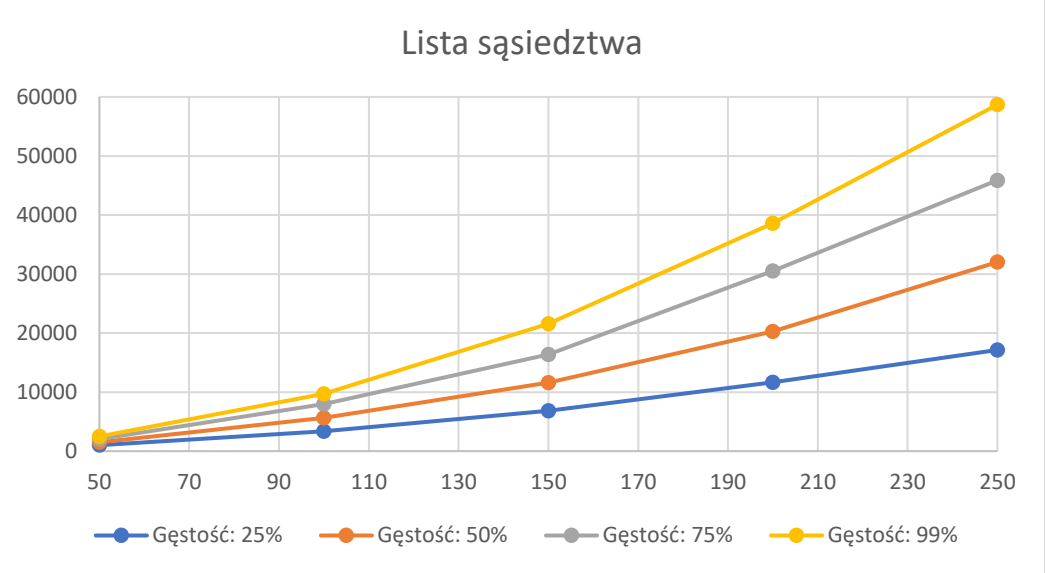
Każda kombinacja jest mierzona 90 razy (dla nowego, wygenerowanego na nowo grafu) i czas jej trwania jest uśredniany.

Wyniki pomiarów podane są w mikrosekundach, chyba że zaznaczone inaczej.

4. Wyniki testów

a. Algorytm Prima

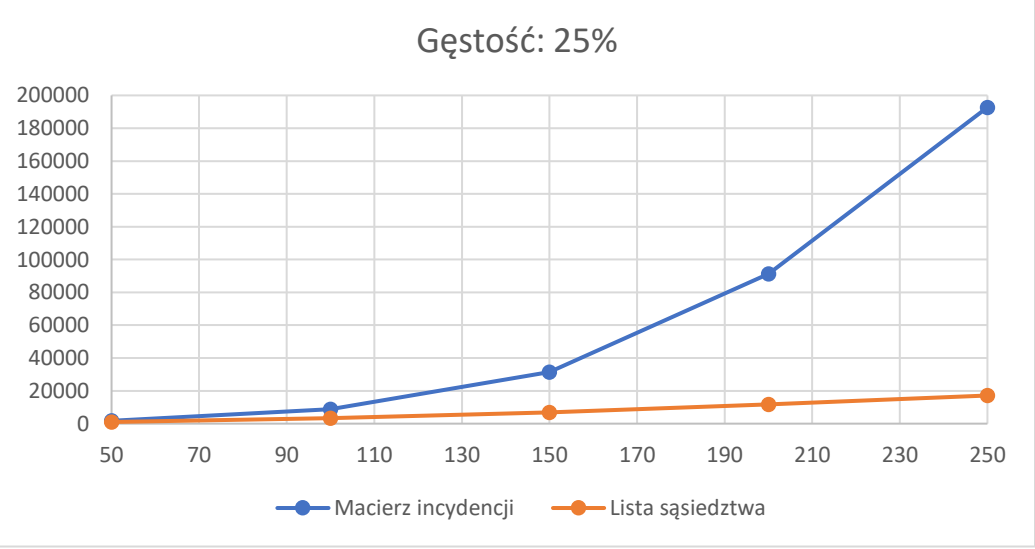


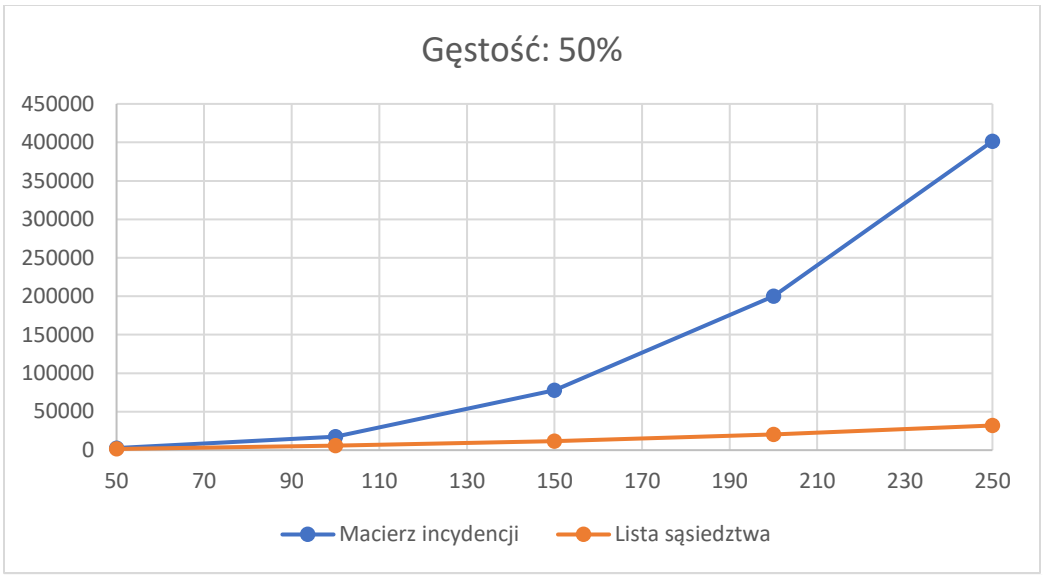


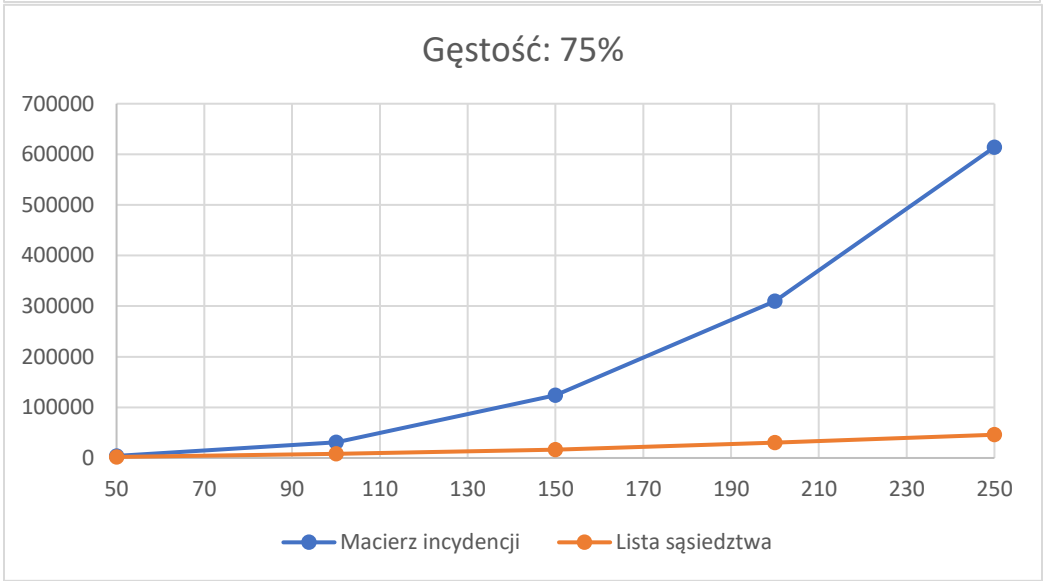
W implementacji algorytmu Prima zastosowałem kolejkę priorytetową w formie kopca, co wydaje mi się, że ładnie pokazane jest na wynikach działania tego algorytmu dla listy sąsiedztwa, gdzie gęstość ma dość spore znaczenie. Wydaje mi się, że potwierdzają one złożoność $O(E \log V)$.

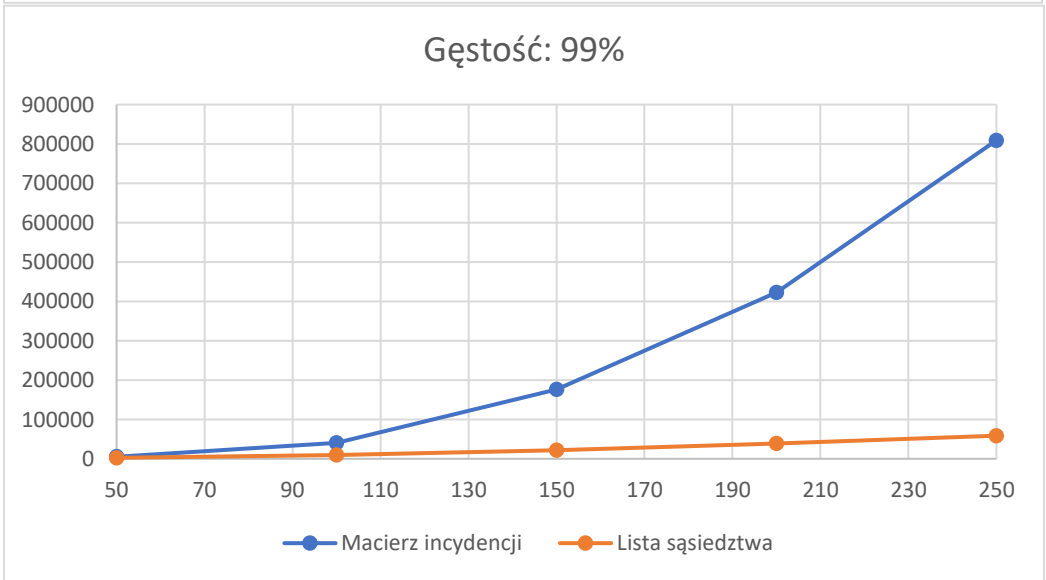
W przypadku macierzy incydencji wartości również są bardzo mocno zależne od gęstości grafu, jednak wydaje mi się, że jest to narzut spowodowany koniecznością przeszukiwania macierzy po wszystkich krawędziach celem znalezienia sąsiadów danego wierzchołka, a później każdy znaleziony wiersz po ilości wierzchołków, by znaleźć koniec krawędzi. Ciężko mi określić jaka może to być złożoność, jednak nie potwierdzają one złożoności książkowej.

Na poniższych wykresach (które generalnie nie różnią się od siebie niczym) widać dokładnie, że macierz incydencji daje znaczący przyrost czasu wykonywania względem listy.

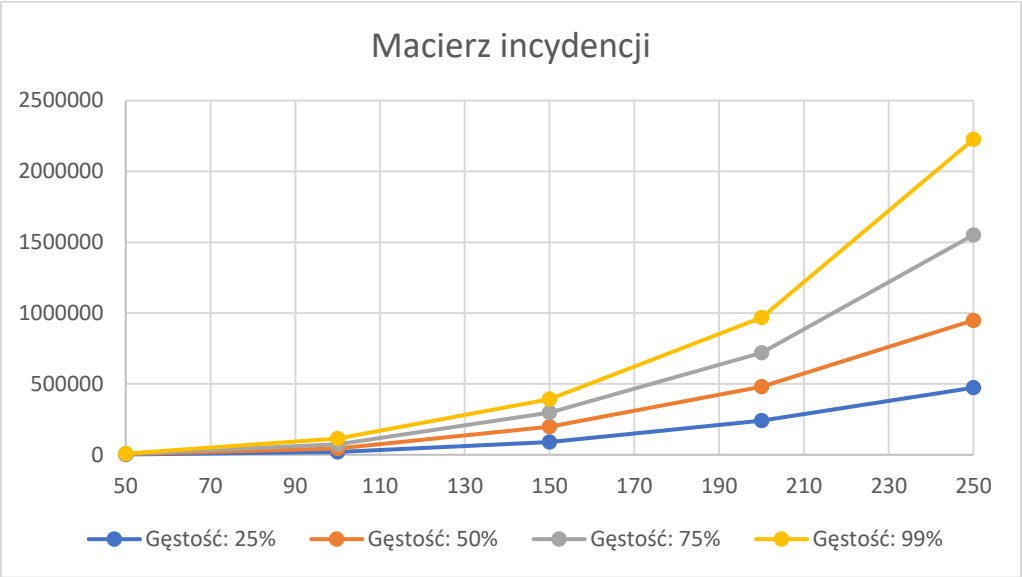


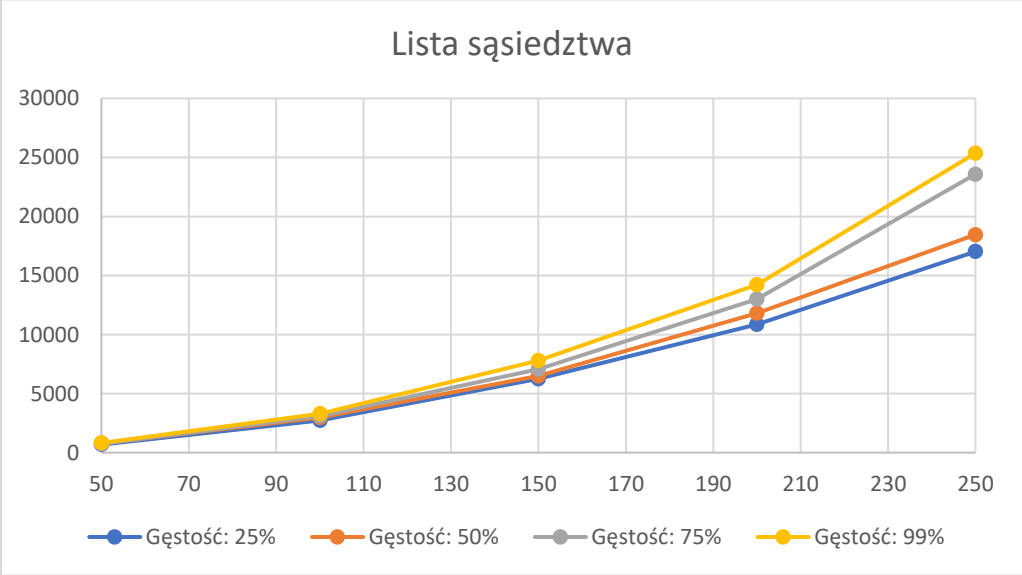

Gęstość: 50%
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa


Gęstość: 75%
700000
600000
500000
400000
300000
200000
100000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa


Gęstość: 99%
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa

b. Algorytm Dijkstry



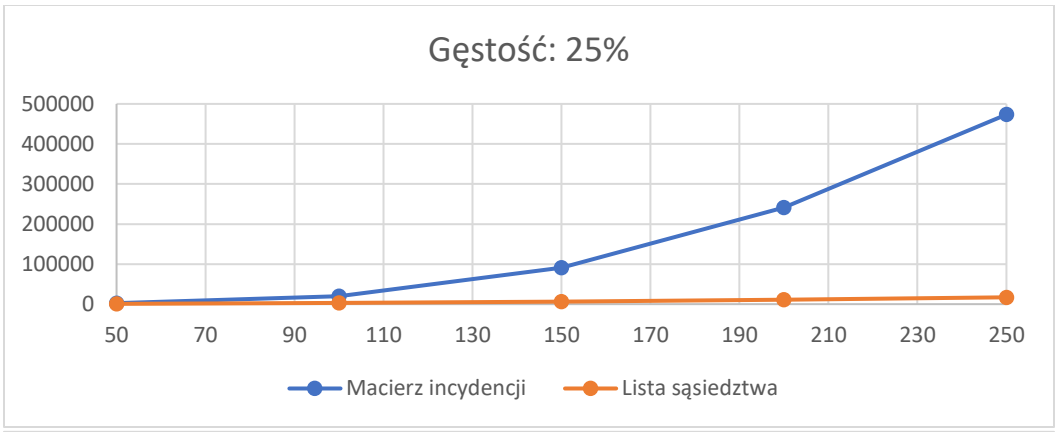


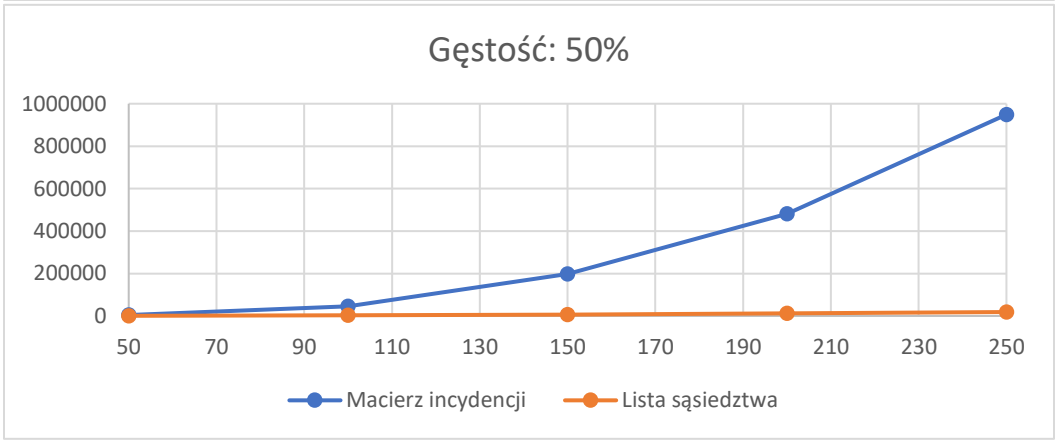
W implementacji tego algorytmu zastosowałem najzwyklejszą kolejkę priorytetową, czyli tablicę.

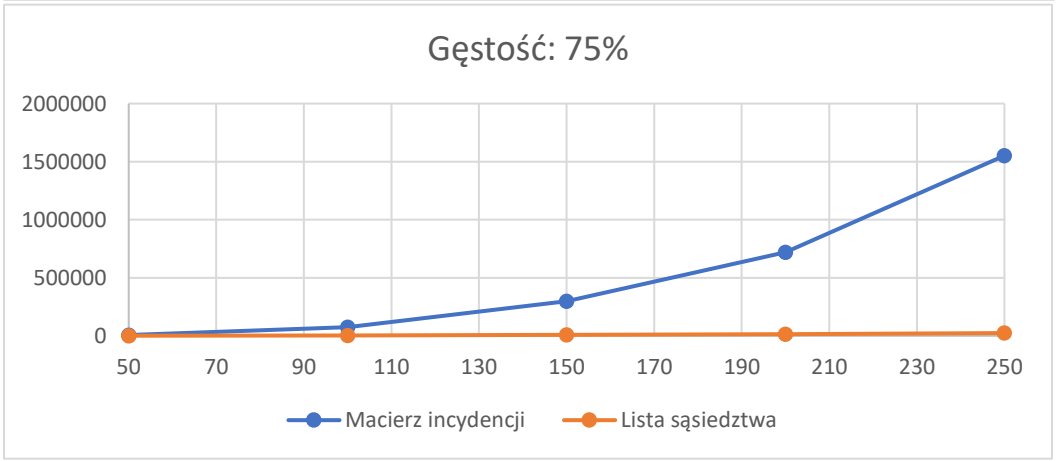
Wydaje mi się, że widać to na wynikach dla listy, gdzie są one podobne do złożoności $O(V^2)$.

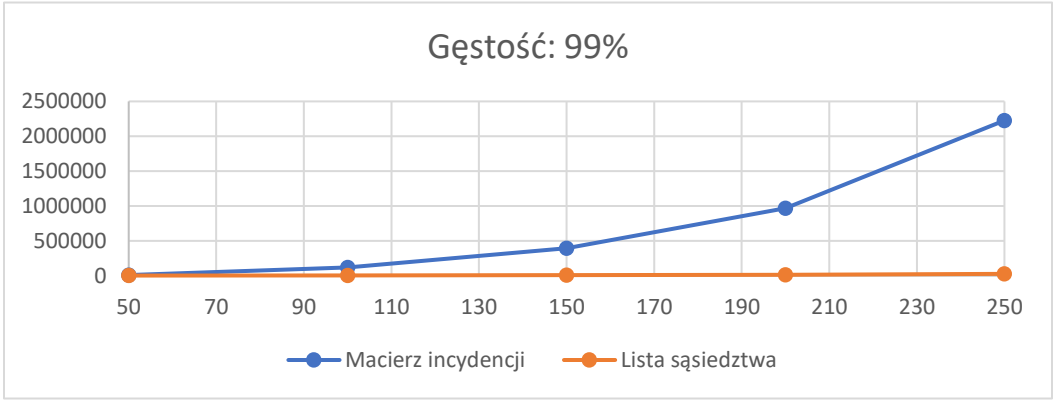
Reprezentacja również w tym miejscu posiada narzut spowodowany szukaniem sąsiadów i z tego powodu wyniki nie potwierdzają reprezentacji książkowej.

Tak jak wyżej wykresy porównujące wydajność macierzy do listy są identyczne.


Gęstość: 25%
500000
400000
300000
200000
100000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa


Gęstość: 50%
1000000
800000
600000
400000
200000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa


Gęstość: 75%
2000000
1500000
1000000
500000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa


Gęstość: 99%
2500000
2000000
1500000
1000000
500000
0
50
70
90
110
130
150
170
190
210
230
250
Macierz incydencji
Lista sąsiedztwa

5. Wnioski

- Macierz incydencji jest bardzo nie wygodną i nie optymalną w użyciu reprezentacją grafu
- Przy pisaniu takiego programu należy poświęcić więcej czasu na organizacje i komentowanie kodu
- Powinienem poświęcić więcej czasu, by zachować kompatybilność multipratformową Windows/Linux, ponieważ niezachowanie jej spowodowało mi duże problemy