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

Maksim Birszel nr indeksu 241353

29 maja 2019

1 Wstęp

Reprezentacja grafu to sposób zapisu grafu umożliwiający jego obróbkę z użyciem programów komputerowych. Do reprezentacji grafów w pamięci komputera wymyślono kilka różnych struktur danych. Każda z nich posiada swoje zalety, lecz również wady. Dlatego należy je rozsądnie dobierać do zadań, w których używamy grafów. Źle dobrana reprezentacja może znacząco wydłużyć czas obliczeń lub rozmiar zajmowanej pamięci. Graf zwykle nie jest strukturą hierarchiczną, jak np. opisane wcześniej drzewa. Jego węzły nie muszą się ze sobą łączyć w określony sposób. Mogą również istnieć grupy węzłów, które nie są w żaden sposób połączone z innymi. Dlatego sposoby reprezentacji drzew niezbyt nadają się do reprezentacji grafów, która powinna zapewniać dostęp do wszystkich wierzchołków bez względu na ich wzajemne połączenia krawędziami. Takie cechy posiadają tablice i macierze.
Trzy najczęściej spotykane sposoby przedstawiania grafów: macierz sąsiedztwa (ang. adjacency matrix), macierz incydencji (ang. incidence matrix) oraz listy sąsiedztwa (ang. adjacency lists). Cechą charakterystyczną tych implementacji jest wykorzystanie tablic do przechowywania danych na temat wierzchołków lub łączących je krawędzi. Wykorzystuje się również struktury mieszane, np. tablicę, której elementami są listy.

1.1 Macierz sąsiedztwa

Graf reprezentujemy za pomocą macierzy kwadratowej A o stopniu n, gdzie n oznacza liczbę wierzchołków w grafie. Macierz tą nazywamy macierzą sąsiedztwa (ang. adjacency matrix). Odwzorowuje ona połączenia wierzchołków krawędziami. Wiersze macierzy sąsiedztwa odwzorowują zawsze wierzchołki startowe krawędzi, a kolumny odwzorowują wierzchołki końcowe krawędzi. Komórka A[i,j], która znajduje się w i-tym wierszu i j-tej kolumnie odwzorowuje krawędź łączącą wierzchołek startowy vi z wierzchołkiem końcowym vj. Jeśli A[i,j] ma wartość 1, to dana krawędź istnieje. Jeśli A[i,j] ma wartość 0, to wierzchołek vi nie łączy się krawędzią z wierzchołkiem vj.

1.2 Macierz incydencji

Macierz incydencji (ang. incidence matrix) jest macierzą A o wymiarze n x m, gdzie n oznacza liczbę wierzchołków grafu, a m liczbę jego krawędzi. Każdy wiersz tej macierzy odwzorowuje jeden wierzchołek grafu. Każda kolumna odwzorowuje jedną krawędź. Zawartość komórki A[i,j] określa powiązanie (incydencję) wierzchołka vi z krawędzią ej.

1.3 Listy sąsiedztwa

Do reprezentacji grafu wykorzystujemy tablicę n elementową A, gdzie n 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. Tablica ta nosi nazwę list sąsiedztwa (ang. adjacency lists).

2 Złożoności obliczeniowe poszczególnych operacji według literatury

- Macierz sąsiedztwa

  Dla macierzy sąsiedztwa sprawdzenie połączenia wierzchołków krawędzią ma klasę złożoności obliczeniowej równą O(1). Wadą jest klasa zajętość pamięci równa O(n^2), gdzie n oznacza liczbę wierzchołków w grafie. Z drugiej strony komórki macierzy mogą być pojedynczymi bitami.
  Z macierzy sąsiedztwa można odczytać wiele pożytecznych informacje. Oto niektóre z nich:

  - Dla grafu nieskierowanego: stopień wierzchołka vi wyznaczamy, zliczając w i-tym wierszu lub i-tej kolumnie liczbę komórek o wartości 1.
  - Dla grafu skierowanego: stopień wychodzący wierzchołka vi wyznaczymy przez zliczenie komórek o wartości 1 w i-tym wierszu, stopień wchodzący wierzchołka vi wyznaczymy przez zliczenie komórek o wartości 1 w i-tej kolumnie.
  - Sąsiedzi wierzchołka vi to te wierzchołki grafu, do których prowadzą krawędzie z vi, a znajdziemy je przeglądając wiersz i-ty.
  - Wierzchołki, dla których vi jest sąsiadem, to te, od których prowadzą krawędzie do vi, a znajdziemy je przeglądając kolumnę i-tą.
  - Wierzchołek vi jest izolowany, jeśli wiersz i-ty i kolumna i-ta zawiera same zera. Dla grafu nieskierowanego wystarczy sprawdzić tylko wiersz lub tylko kolumnę.
  - Wierzchołek vi posiada pętlę, jeśli komórka A[i,i] ma wartość 1.
  - W grafie skierowanym wierzchołki vi i vj są połączone krawędzią dwukierunkową, jeśli jednocześnie komórki A[i,j] oraz A[j,i] mają wartość 1.

- Macierz incydencji

Macierz incydencji wymaga pamięci o rozmiarze O(m x n). Przydaje się wtedy, gdy algorytm musi posiadać informację o krawędziach (bo przykładowo posiadają one wagi). Pozwala w czasie O(1) sprawdzić, czy wierzchołek vi należy do krawędzi ej. Macierz incydencji nie nadaje się zbyt dobrze do reprezentacji grafów z pętlami, ponieważ wierzchołek startowy i końcowy muszą być różne. Co prawda, można się umówić na specjalną wartość, np. 2, dla wierzchołka, który jest jednocześnie początkiem jak i końcem krawędzie, lecz komplikuje to przetwarzanie macierzy. Z kolei macierz incydencji bez problemu pozwala reprezentować krawędzie wielokrotne. Z macierzy incydencji możemy odczytać te same informacje, co z macierzy sąsiedztwa (chociaż czasami wymaga to więcej działań):

- Dla grafu nieskierowanego: stopień wierzchołka vi wyznaczamy, zliczając w i-tym wierszu liczbę komórek o wartości 1.
- Dla grafu skierowanego: stopień wychodzący wierzchołka vi wyznaczymy przez zliczenie komórek o wartości 1 w i-tym wierszu, a stopień wchodzący wierzchołka vi wyznaczymy przez zliczenie komórek o wartości -1.
- Znalezienie wszystkich sąsiadów wierzchołka vi nie jest bezpośrednie. W tym celu musimy przeglądnąć kolejno cały wiersz i-ty macierzy incydencji. Każdą kolumnę, w której w wierszu i-tym jest wartość 1, przeglądamy aż do napotkanie komórki o wartości -1 (dla grafu nieskierowanego aż do napotkania komórki o wartości 1 leżącej w wierszu różnym od i-tego). Numer wiersza tej komórki określa sąsiada wierzchołka vi. Operacja ta ma złożoność O(n x m).
- Wierzchołek vi jest izolowany, jeśli wiersz i-ty zawiera same zera.
- W grafie skierowanym wierzchołki vi oraz vj są połączone krawędzią dwukierunkową, jeśli istnieją w macierzy incydencji dwie kolumny, które w wierszach i-tym i j-tym mają wartości przeciwne i różne od zera.

- Listy sąsiedztwa

  Listy sąsiedztwa są efektywnym pamięciowo sposobem reprezentacji grafu w pamięci komputera, ponieważ zajmują pamięć rzędu O(m), gdzie m oznacza liczbę krawędzi grafu. Listy sąsiedztwa pozwalają w prosty sposób reprezentować pętle oraz krawędzie wielokrotne, co sprawia, że są bardzo chętnie stosowane w algorytmach grafowych. Oto kilka podstawowych operacji na listach sąsiedztwa:

  - Dla grafu nieskierowanego: stopień wierzchołka vi jest równy liczbie elementów na liście A[i].
  - Dla grafu skierowanego: stopień wychodzący wierzchołka vi jest równy liczbie elementów na liście A[i], stopień wchodzący wierzchołka vi jest równy liczbie elementów o wartości i we wszystkich listach sąsiedztwa.
  - Sąsiedzi wierzchołka vi są określeni przez kolejne elementy listy A[i]
  - Wierzchołki, dla których vi jest sąsiadem, znajdziemy szukając na listach elementu o wartości i.
  - Wierzchołek vi jest izolowany, jeśli lista A[i] jest pusta.

– Wierzchołek vi posiada pętlę, jeśli na liście A[i] znajduje się element o wartości i.

– W grafie skierowanym wierzchołki vi i vj są połączone krawędzią dwukierunkową, jeśli lista A[i] zawiera element o wartości j, a lista A[j] zawiera element o wartości i.

3 Algorytmy grafowe

3.1 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. Algorytm został wynaleziony w 1930 przez czeskiego matematyka Vojtěcha Jarníka, a następnie odkryty na nowo przez informatyka Roberta C. Prima w 1957 oraz niezależnie przez Edsgera Dijkstrę w 1959. Z tego powodu algorytm nazywany jest również czasami algorytmem Dijkstry-Prima, algorytmem DJP, algorytmem Jarníka, albo algorytmem Prima-Jarníka.

3.1.1 Złożoność obliczeniowa algorytmu Prima

Złożoność obliczeniowa zależy 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|) co przy dużej gęstości grafu (takiej, że |E| jest (|V|*log|V|) oznacza duże przyspieszenie.

3.2 Algorytm Dijkstry

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

3.2.1 Złożoność obliczeniowa algorytmu Djikstry

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*log|V|)
- 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.

3.3 Algorytm Kruskala

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. Został on po raz pierwszy opublikowany przez Josepha Kruskala w 1956 roku w czasopiśmie Proceedings of the American Mathematical Society.

3.3.1 Złożoność obliczeniowa algorytmu Kruskala

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 czasie O(E*logV). 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.
Całkowity czas działania jest zatem O(E*logV) ze względu na pierwszą fazę – sortowanie.

3.4 Algorytm Bellmana-Forda

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).

3.4.1 Złożoność obliczeniowa algorytmu Bellmana-Forda

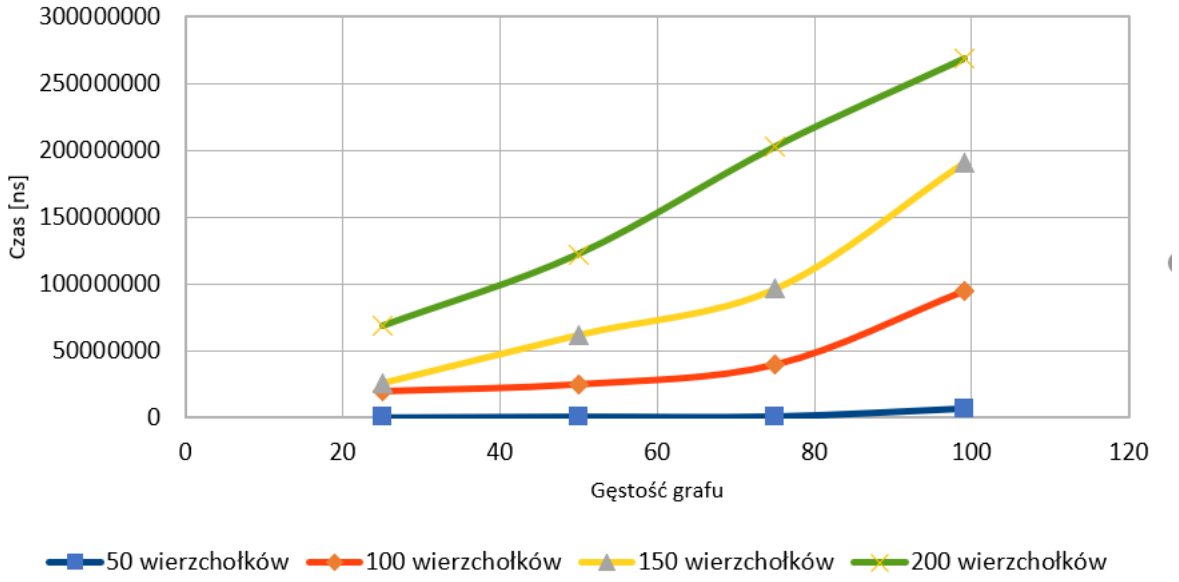
Całkowita złożoność czasowa algorytmu Bellmana-Forda jest rzędu O(n*m).

4 Plan eksperymentu

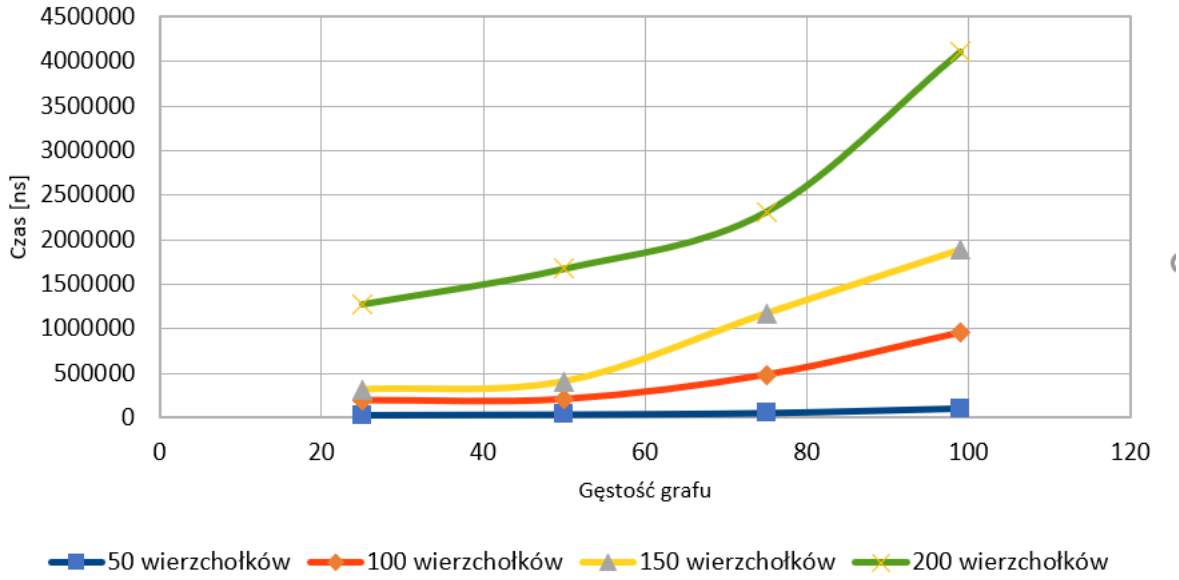
- Wielkość struktur - eksperyment został przeprowadzony dla kilku rozmiarów każdej ze struktur. Kolejno dla 50 elementów, 100 elementów, 150 elementów oraz 200 elementów. Dla każdej z nich pomiar czasu został powtórzony wielokrotnie a wynik tych pomiarów uśredniony.
- Sposób generowania elementów struktur - elementy są generowane losowo przez podanie wielkości poszczególnych wartości zmiennych jak ilość wierzchołków, krawędzi, etc. lub pobierane w całości z pliku tekstowego.

- Do pomiaru czasu wykorzystana został biblioteka „Chrono.h". Na potrzeby uproszczenia pomiaru, przyjęta została jednak dokładność rzędu jednej nanosekundy.
- Wszystkie struktury danych alokowane są dynamicznie, oraz realokowane po dodaniu/usunięciu elementu.
- Dla uproszczenia implementacji została użyta biblioteka STL.

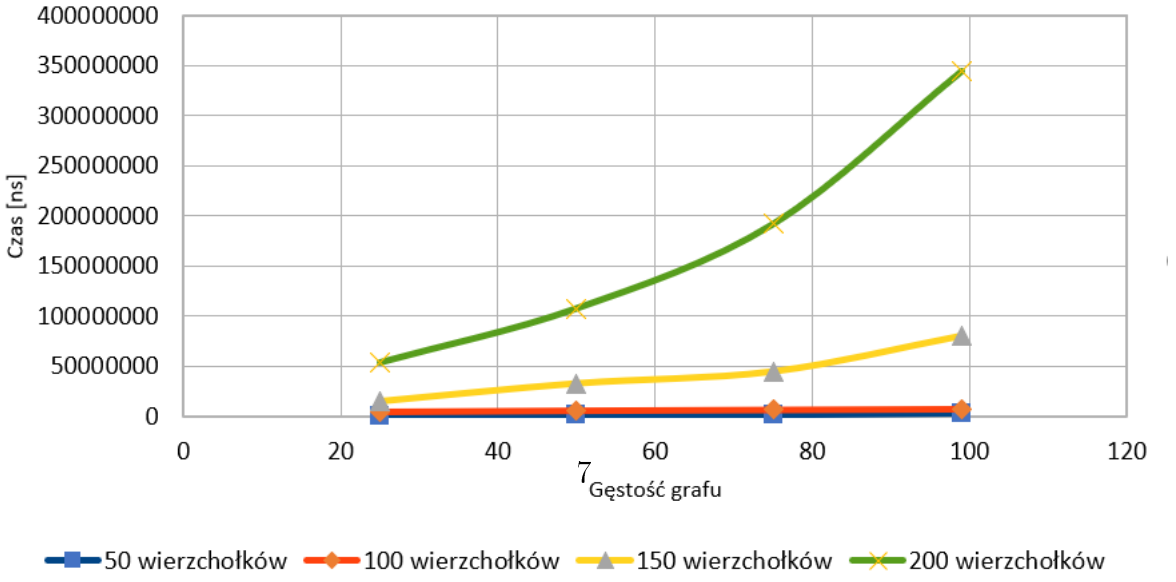
5 Zestawienie wyników eksperymentu



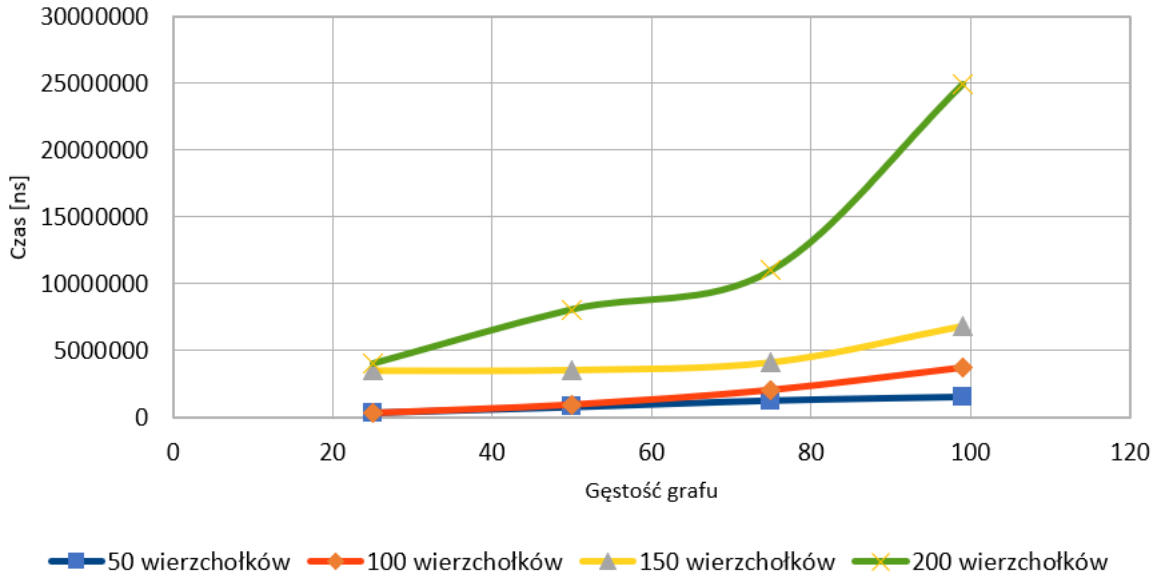
Rysunek 1: Algorytm Djikstry w implementacji macierzowej



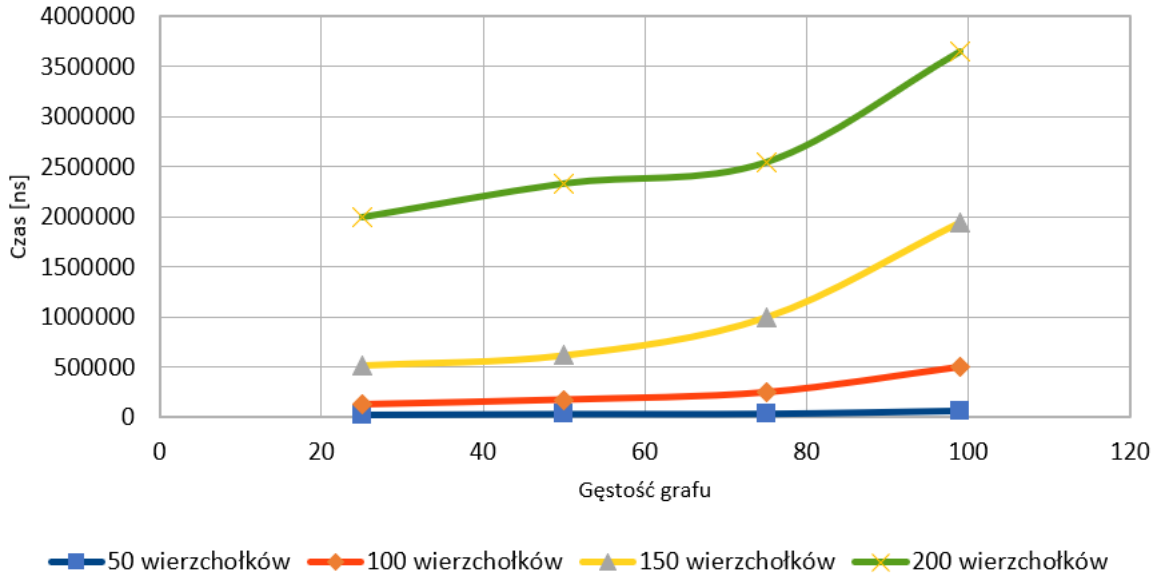
Rysunek 2: Algorytm Djikstry w implementacji listowej



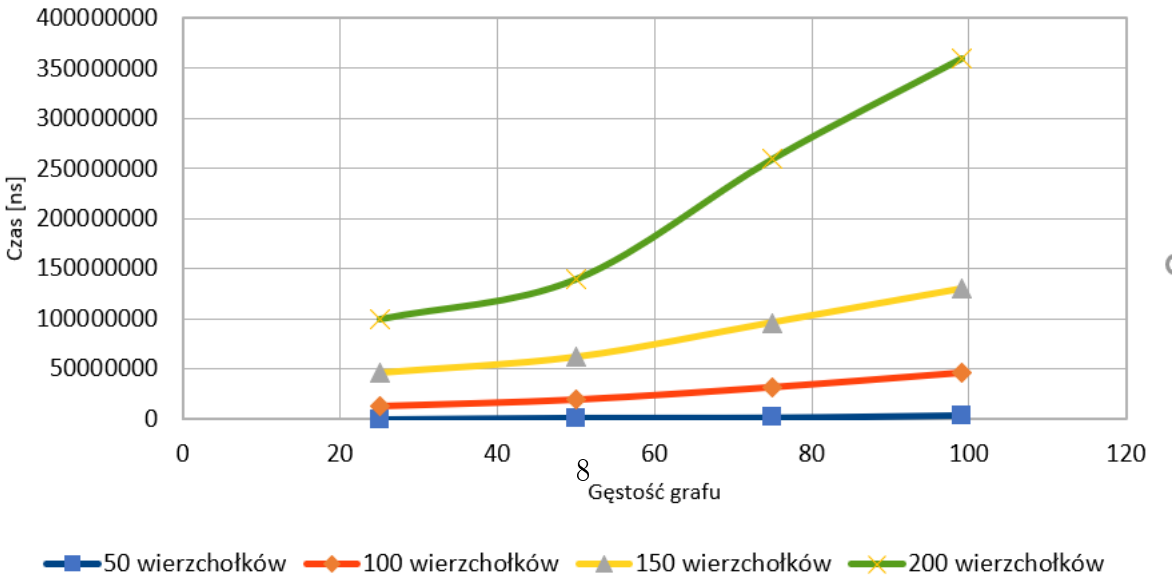
Rysunek 3: Algorytm Prima w implementacji macierzowej



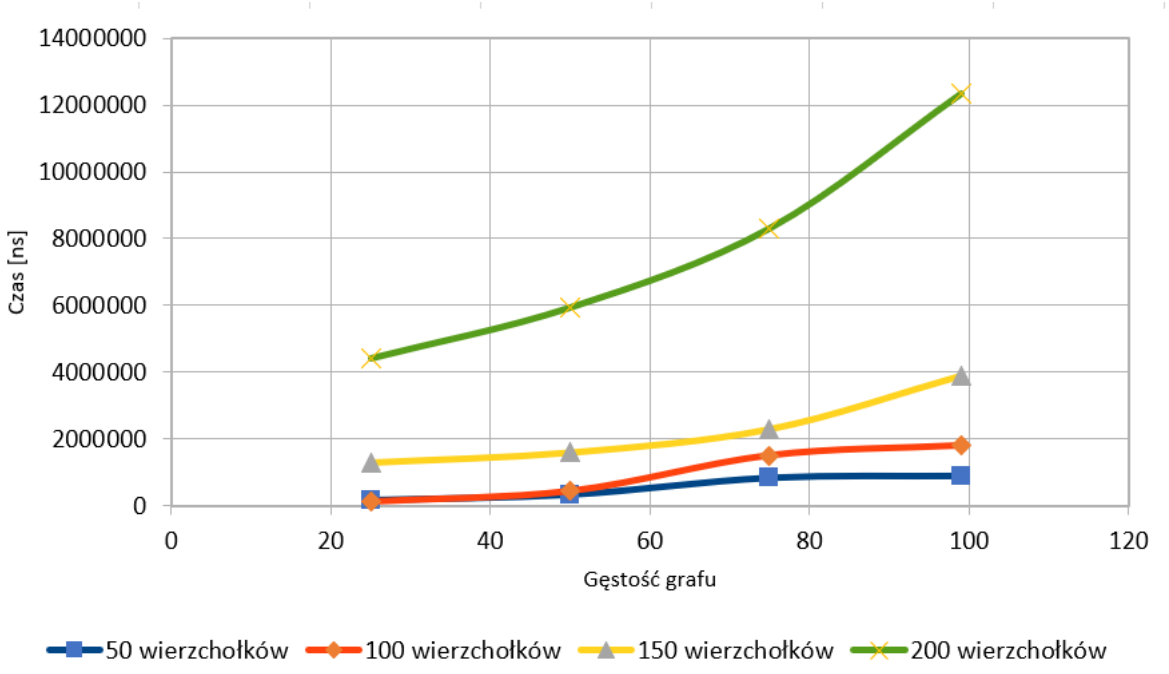
Rysunek 4: Algorytm Prima w implementacji listowej



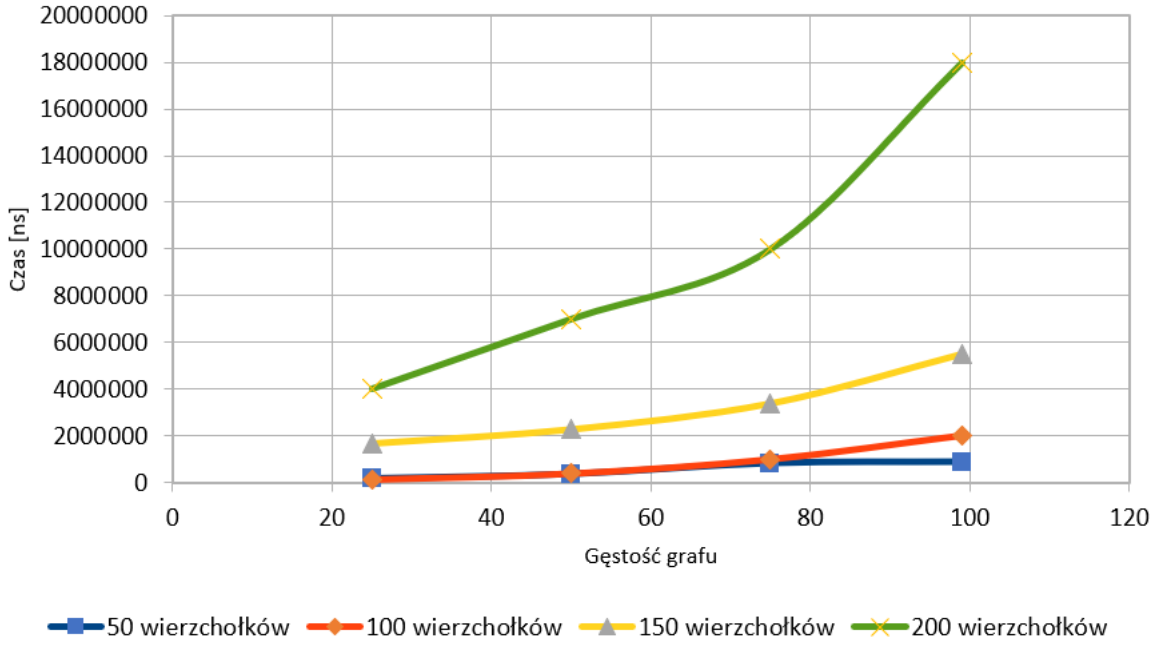
Rysunek 5: Algorytm Kruskala w implementacji listowej



Rysunek 6: Algorytm Kruskala w implementacji macierzowej



Rysunek 7: Algorytm Bellmana-Forda w implementacji listowej



Rysunek 8: Algorytm Bellmana-Forda w implementacji macierzowej

6 Wnioski

Złożoność algorytmów zaprezentowana w punkcie pierwszym wydaje się być zgodna z otrzymanymi wynikami
Różnice pomiędzy otrzymanymi wynikami, a wynikami zakładanymi wynikają z niedokładności pomiarów lub potencjalnych błędów/braku optymalizacji algorytmów
Czas wykonywania wszystkich algorytmów grafowych jest ściśle zależny od ilości wierzchołków i krawędzi badanego grafu
Czasy działań algorytmów można by znacząco skrócić poprzez optymalne implementacje algorytmów z wykorzystaniem kolejki priorytetowej oraz szybszych, a zarazem bardziej skomplikowanych bibliotek operujących na strukturach danych.