# Algorytmy grafowe



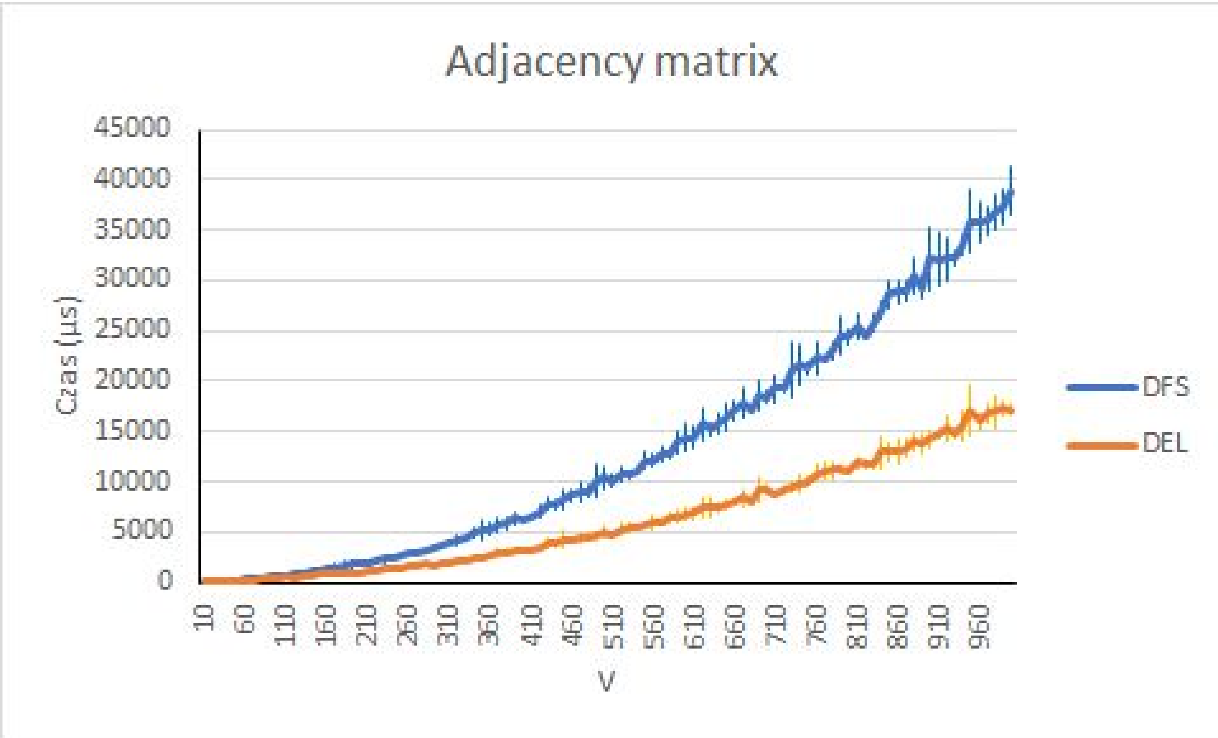
***Wykonanie:***

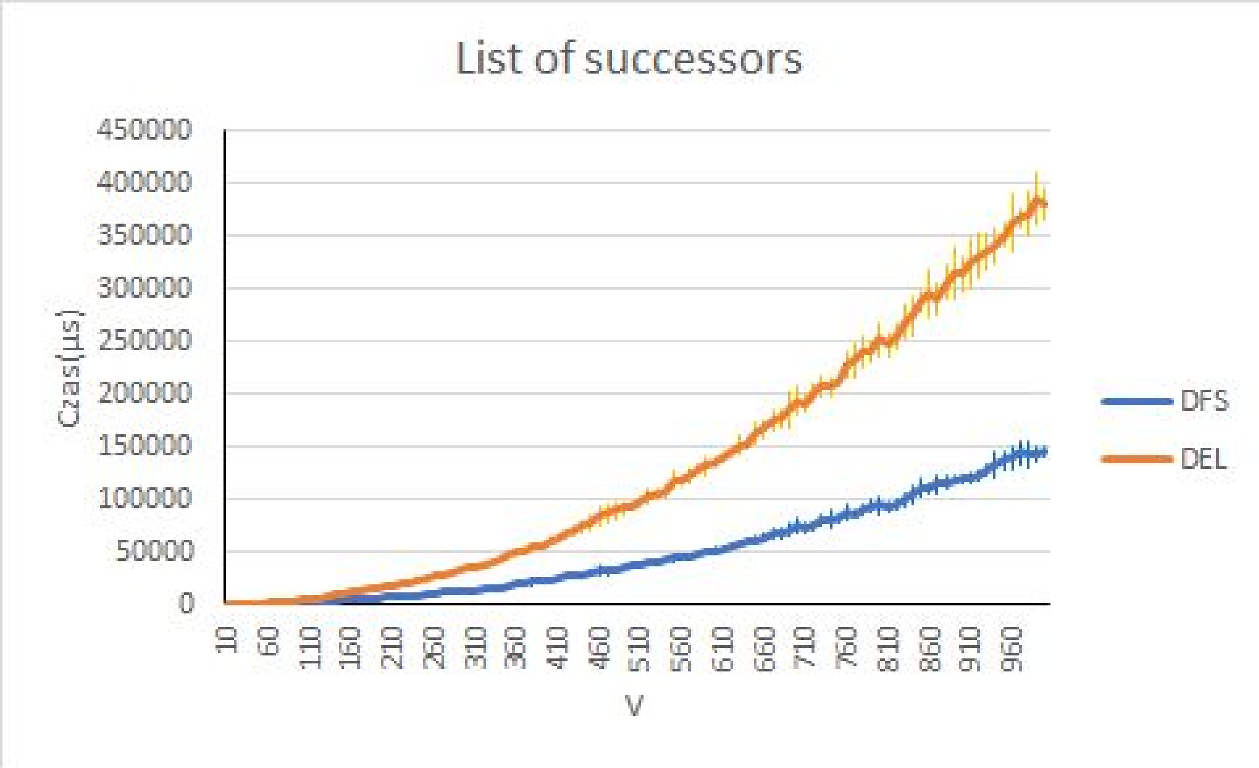
***Adrian Madajewski*** 145406

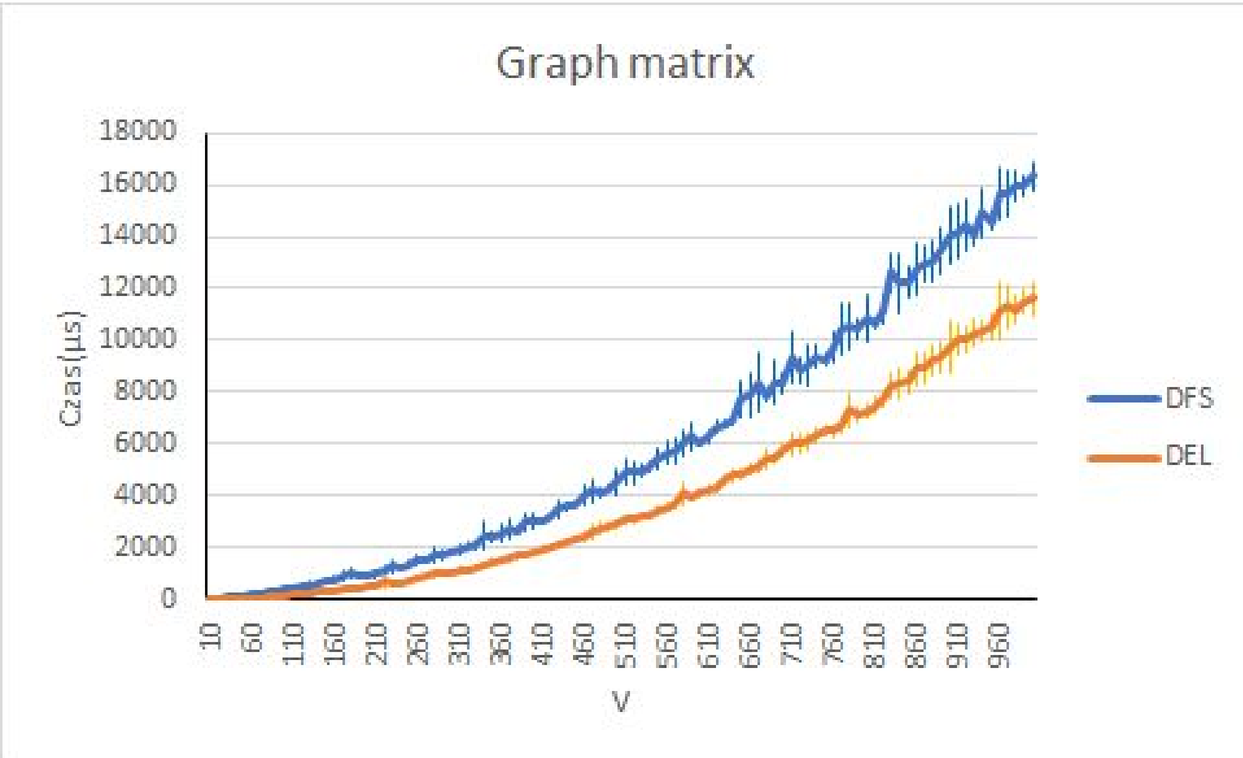
***Michał Kwarta*** 145192

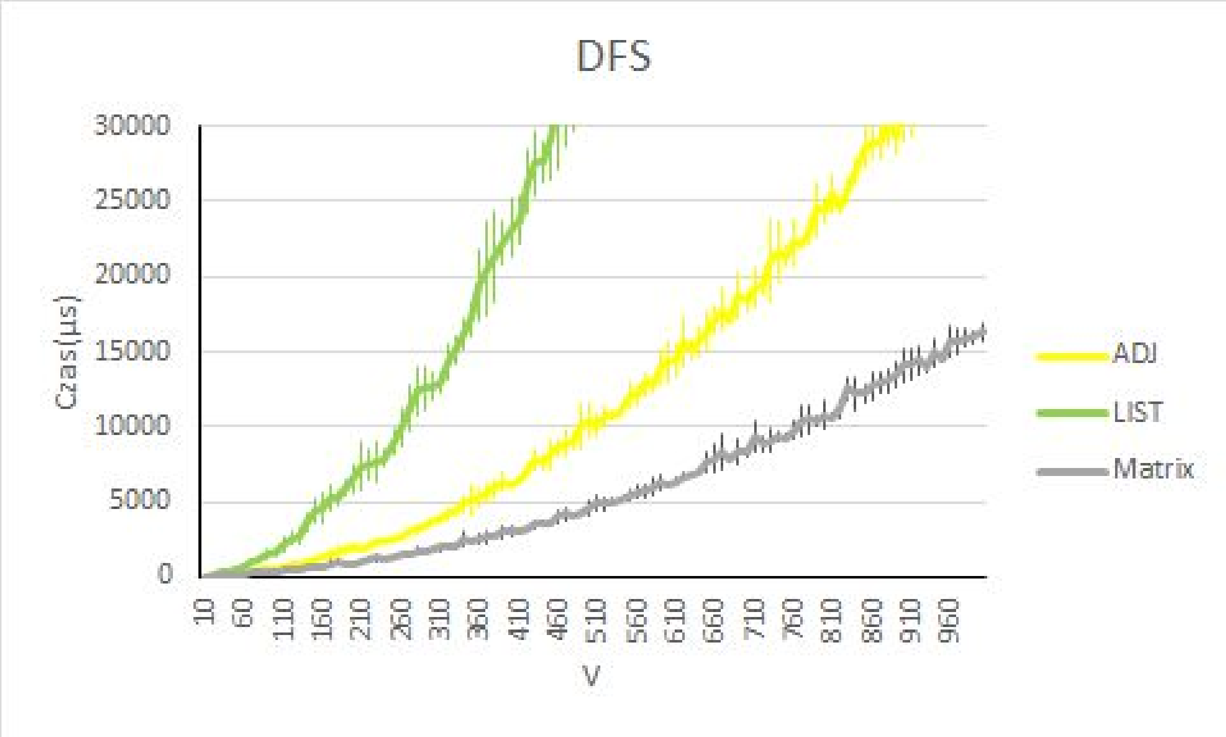
# Wykresy czasu od ilości elementów dla wybranych procedur:

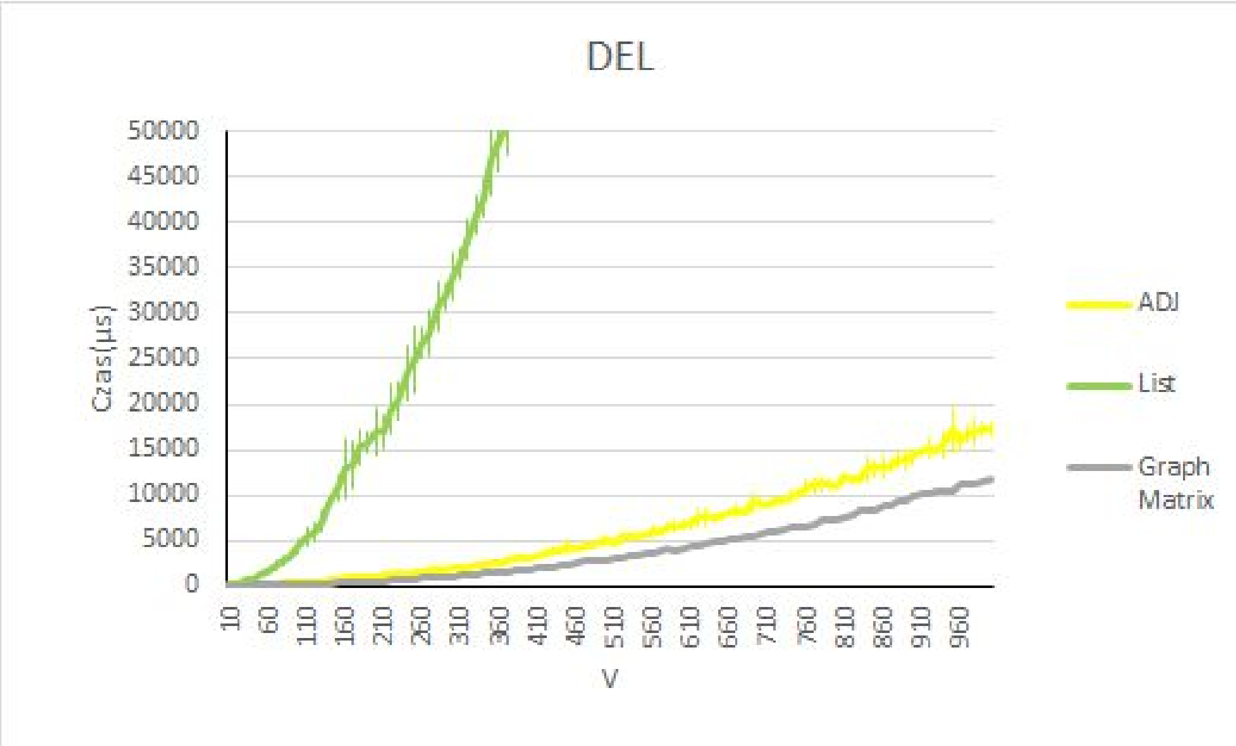
Poniżej przedstawione wykresy są wynikiem $7$ testów, każdy dla $V$ z przedziału $[10, 20, ..., 1000]$. Wygenerowane zostały grafy *DAG* z gęstością $50\%$. Podczas generowania grafów zostało przyjęte założenie, że dla każdej pary $(i, j)$ spełniony jest warunek $i < j$. Dla każdego wierzchołka generowana była losowa liczba krawędzi aż do spełnienia warunku gęstości. **Pomiary nie uwzględniają czasu tworzenia struktury.** Czas na wykresach jest wyrażonych w mikrosekundach.




List of successors
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
Czas(µs)
10
60
110
160
210
260
310
360
410
460
510
560
610
660
710
760
810
860
910
960
V
DFS
DEL


Graph matrix
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Czas(µs)
10
60
110
160
210
260
310
360
410
460
510
560
610
660
710
760
810
860
910
960
V
DFS
DEL


DFS
30000
25000
20000
15000
10000
5000
0
Czas(µs)
10
60
110
160
210
260
310
360
410
460
510
560
610
660
710
760
810
860
910
960
V
ADJ
LIST
Matrix


DEL
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Czas(µs)
10
60
110
160
210
260
310
360
410
460
510
560
610
660
710
760
810
860
910
960
V
ADJ
List
Graph Matrix

# Wnioski:

**Macierz sąsiedztwa:**
Sortowanie *DFS* iteruje po wierzchołkach aż znajdzie biały, wtedy uruchamia procedurę *DFSUtil,* która z kolei iteruje po kolumnach. Sprawia to, że procedura sprawdza wszystkie komórki w macierzy o rozmiarze $V \times V$ aby sprawdzić wszystkie krawędzie co sprawia, że złożoność sprowadza się do $O(V^2)$.

Sortowanie z użyciem algorytmu *Kahna* , działa nieco wydajniej. Najpierw następuje sprawdzenie wszystkich komórek macierzy i przypisanie do wierzchołka jego stopnia wejściowego, jednak dzieje się to odwrotnie niż w przypadku *DFS* - zagnieżdżona pętla iteruje po wierszach, co pozwala sprawdzić poprzedniki, a nie następniki. Następnie tworzona jest kolejka ze wszystkimi wierzchołkami ze stopniem 0. Po wyciągnięciu wierzchołka z kolejki, stopień wejściowy pozostałych wierzchołków jest aktualizowany, jednak ta procedura już nie wymaga sprawdzenia całej macierzy, a istnienia krawędzi, więc jej złożoność wynosi $O(V)$, gdzie $V$ to pozostałe wierzchołki w grafie.

Macierz sąsiedztwa nadaje się idealnie do sprawdzania połączenia każdego z wierzchołków, w czasie $O(1)$, jednak charakteryzuje się najgorszą złożonością przejrzenia wszystkich krawędzi - $O(V^2)$.

**Lista następników:**
Sortowanie *DFS* przypisuje każdemu wierzchołkowi kolor biały, następnie sprawdza sąsiadujące wierzchołki (następniki) - operacja ta zajmuje $O(V_i \cdot E_{V_i})$, gdzie $E_v$ to liczba następników wierzchołka $V$ . Każdy następnik zostaje sprawdzony czy nie został już wcześniej odwiedzony. Nie wymaga to jak w przypadku algorytmu *Kahn'a* przejrzenie wszystkich krawędzi, ponieważ wiemy kiedy zakończyliśmy przeszukiwanie danego wierzchołka zaznaczając go jako odwiedzony - na czarno,  dlatego operacja ta jest szybsza od algorytmu *Kahna*.

Sortowanie z użyciem algorytmu *Kahna* dla listy następników przypisuje każdemu wierzchołkowi stopień wejściowy ze złożonością $O(V_i \cdot E_{V_i})$, gdzie $E_v$ to liczba krawędzi każdego wierzchołka $V_i$ . Następnie znowu dla każdego wierzchołka wykonywana jest kolejna procedura, która z kolejki wierzchołków obniża stopień obecnie badanego wierzchołka. Algorytm wypada gorzej dla listy następników od *DFSa* ponieważ wymaga sprawdzenia *wszystkich* następników danego wierzchołka - każdy stopień wejściowy musi zostać zredukowany do $0$, aby możliwe, było wykonanie sortowania.

Lista następników najlepiej sprawdza się do przeglądania krawędzi każdego z wierzchołków, ponieważ wykonuje to w czasie $O(E)$, gdzie $E$ to liczba wszystkich krawędzi, ale nie nadaje się ona do sprawdzania istnienia jednej krawędzi, bo robi to ze złożonością $O(V)$.

**Macierz grafu:**
Sortowanie *DFS* wykonuje się podobnie jak w przypadku macierzy sąsiedztwa, jednak zamiast sprawdzać w matrycy czy wartość w komórce jest równa $1$, sprawdzane jest czy wartość mieści się w przedziale $(0;\ V]$, jednak w tym przypadku mamy bezpośredni dostęp do listy następników z kolumny dodatkowej, w ten sposób tak naprawdę łączymy wydajność dwóch poprzednich struktur uzyskując tym samym jeszcze lepsze wyniki sortowania, ponieważ algorytm wie, jakie są kolejne następniki danego wierzchołka - (zapisane w matrycy) - przeszukujemy ją tylko raz, każdy wiersz dla każdego wierzchołka - odpowiednio oznaczając odwiedzane wierzchołki kolorami.

Sortowanie *Kahna* w przypadku naszej implementacji jest to tak samo jak w poprzednim - najpierw nadanie każdemu wierzchołkowi stopnia wejściowego - poprzez listę poprzedników - gdzie, ponieważ w przypadku macierzy grafu mamy do niej dostęp - a każdy kolejny element jest odpowiednio zaadresowany w matrycy. Tym sposobem wyznaczenie stopnia wejściowego każdego wierzchołka wykonuje się w czasie $O(V\ +\ S_v)$ - gdzie $S_v$ oznacza złożoność odwiedzenia poprzedników danego wierzchołka $V$. Usuwanie wierzchołków działa na dokładnie takiej samej zasadzie jak algorytm usuwania dla listy następników - najpierw dodajemy wierzchołki o stopniu zerowym na stos - następnie sprawdzamy połączenie każdego wierzchołka macierzy za pomocą listy następników - odwiedzając dany wierzchołek zmniejszamy jego stopień, dopóki nie osiągniemy $0$ dla każdego z wierzchołków - wtedy kończymy sortowanie.

Macierz grafu łączy w sobie zalety dwóch poprzednich reprezentacji - złożoność sprawdzenia istnienia jednej krawędzi - $O(1)$ i złożoność przejrzenia wszystkich krawędzi - $O(E)$. Jedyną wadą macierzy grafu jest większa złożoność pamięciowa w stosunku do listy następników, jednakże dla grafów gęstych obie reprezentacje sprowadzają się do podobnej złożoności pamięciowej.

**Sortowanie z wykorzystaniem *DFS*:**
Lista następników wypada najgorzej dla tego algorytmu, ponieważ wymaga od nas przeszukania każdego następnika każdego wierzchołka, natomiast w przypadku macierzy sąsiedztwa oraz macierzy grafu wystarczy, że informacja o takim połączeniu widnieje w komórce macierzy i dla obydwu tych algorytmów wystarczy sprawdzić warunek istnienia połączenia wierzchołka $i$ z wierzchołkiem $j$ poprzez wartość komórki $M[i,\ j]$ gdzie $M$ jest danym typem matrycy.

**Sortowanie z wykorzystaniem algorytmu Kahna:**
Najlepszą reprezentacją danych dla wydajnego korzystania z tego algorytmu jest tak jak w przypadku poprzednim macierz grafu, oraz macierz sąsiedztwa - ponieważ aby sprawdzić istnienie następnika w macierzy sąsiedztwa sprawdzamy odpowiednią komórkę z macierzy. W przypadku macierzy grafu - sprawdzamy tylko każdy jeden wiersz tej macierzy i odpowiednie jego połączenie - w ten sposób obliczamy ilość stopni wejściowych i wykonujemy sortowanie, natomiast dla listy następników musimy przejrzeć każdy możliwy wierzchołek podwójnie, najpierw w celu obliczenia stopnia wejściowego, a następnie w celu sprawdzenia połączenia i obniżenia stopnia obecnie badanego wierzchołka.

# Informacje dodatkowe

## Link do arkusza z pomiarami czasu:

https://1drv.ms/x/s!AmnqNWF6AFGwgYVJ8eDHfsQjz7UpjA?e=SJtlpM

## Platforma testowa

| | |
|---|---|
| procesor: | Intel i7 7700k 4.2 GHz |
| ram: | 8 GB RAM DDR4 CL16 |
| karta graficzna: | Geforce GTX 1060 6GB |
| system operacyjny: | Windows 7 64-bit |
| płyta główna: | MSI Z270-A PRO |
| dysk: | HDD 1000 GB |
| j.programowania | C++, -std=c++1z |