Sprawozdanie 3

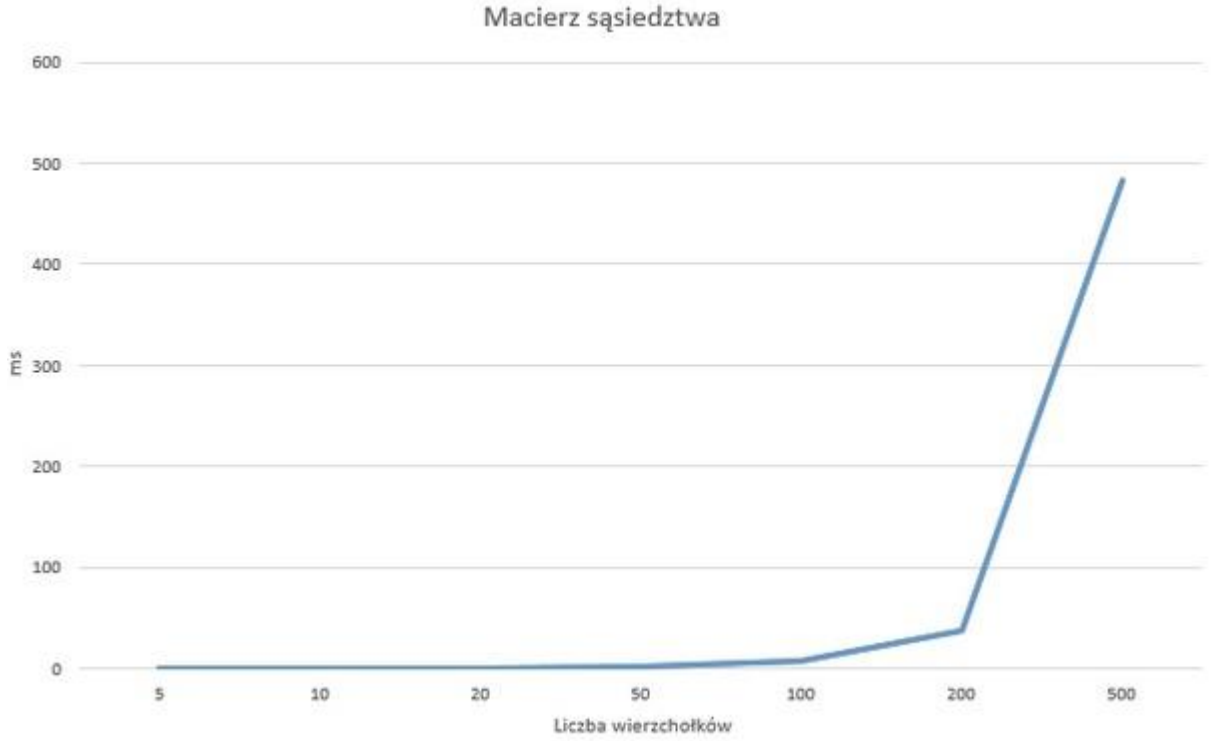
Algorytmy grafowe

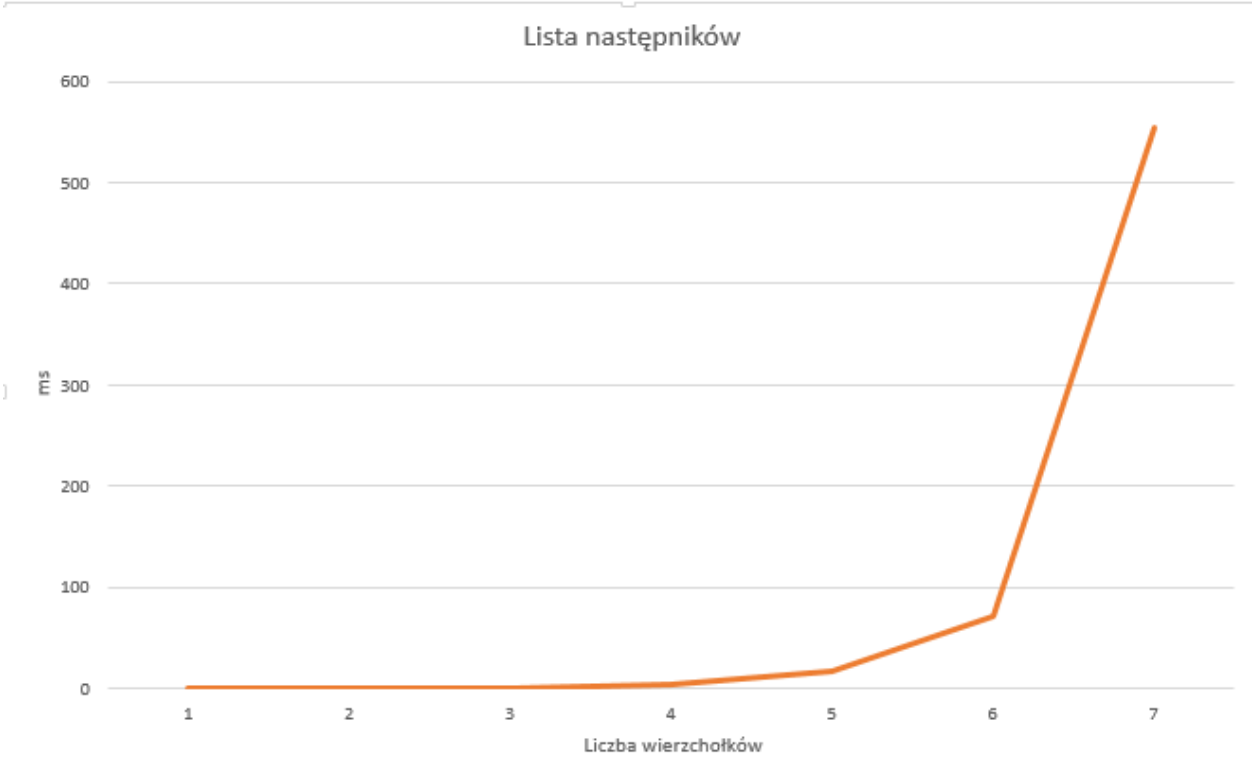
Andrei Staravoitau, nr 150218,

Informatyka I6, II semestr

1. **Graf** – struktura danych składająca się ze zbioru wierzchołków *V* i zbioru krawędzi *E*.
   Graf skierowany – zawiera krawędzie skierowane, *łuki*, z określonym kierunkiem przechodzenia krawędzi.
   Cykl – zamknięta ścieżka między wierzchołkami, która zaczyna i kończy się w tym samym wierzchołku.
   Graf acykliczny – graf który nie zawiera cyklu.
2. **Macierz sąsiedztwa (złożoność pamięciowa O(n*n)) –** macierz, w której każda kolumna i wiersz odpowiada jednemu wierzchołkowi, gdzie wierszy – wierzchołki startowe, a kolumny – końcowe. Otworzenie krawędzi jest **„1”.**
   Tabela krawędzi (złożoność pamięciowa O(2*m)) – tabela z dwóch kolumn, 1 odpowiada wierzchołkowi startowemu, 2 – końcowemu.
   Lista następników – 1 element to początek łuków, wszystkie ostatnie w wierszu – koniec.

Pomiary casu





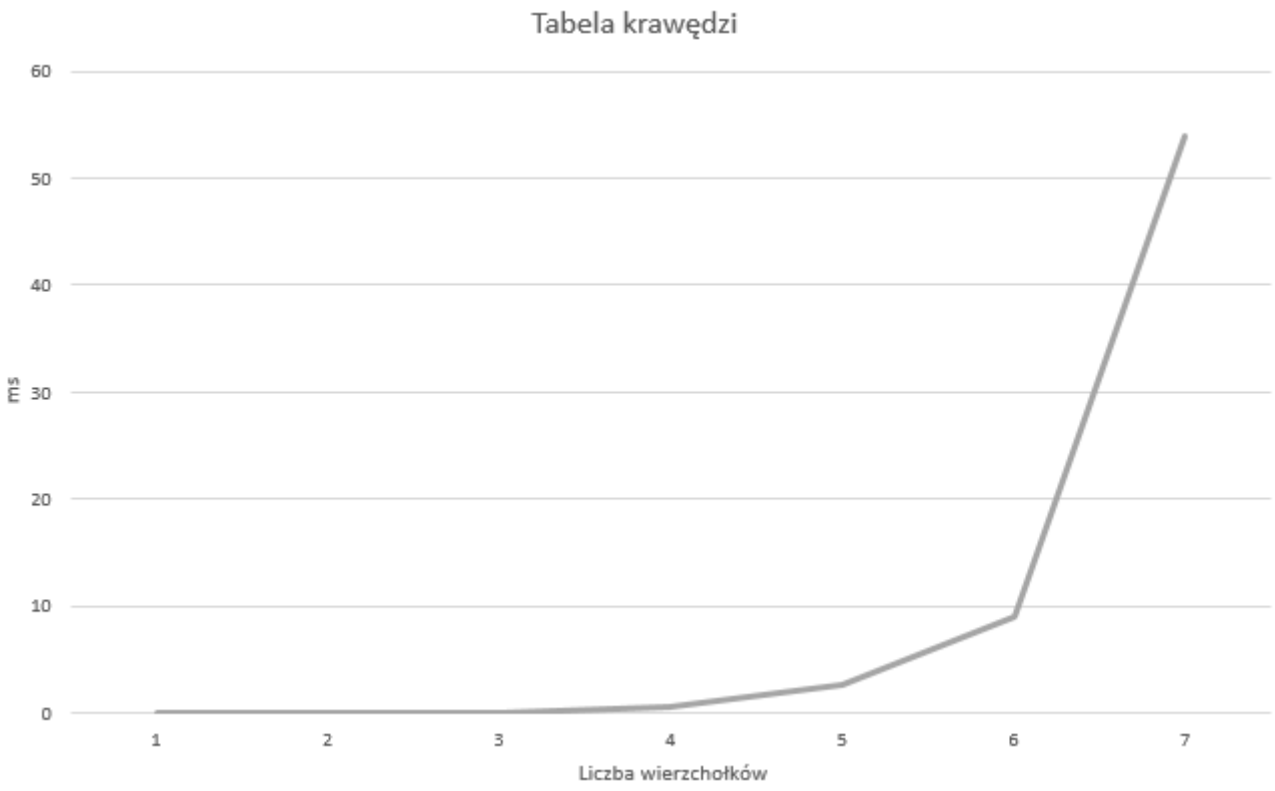


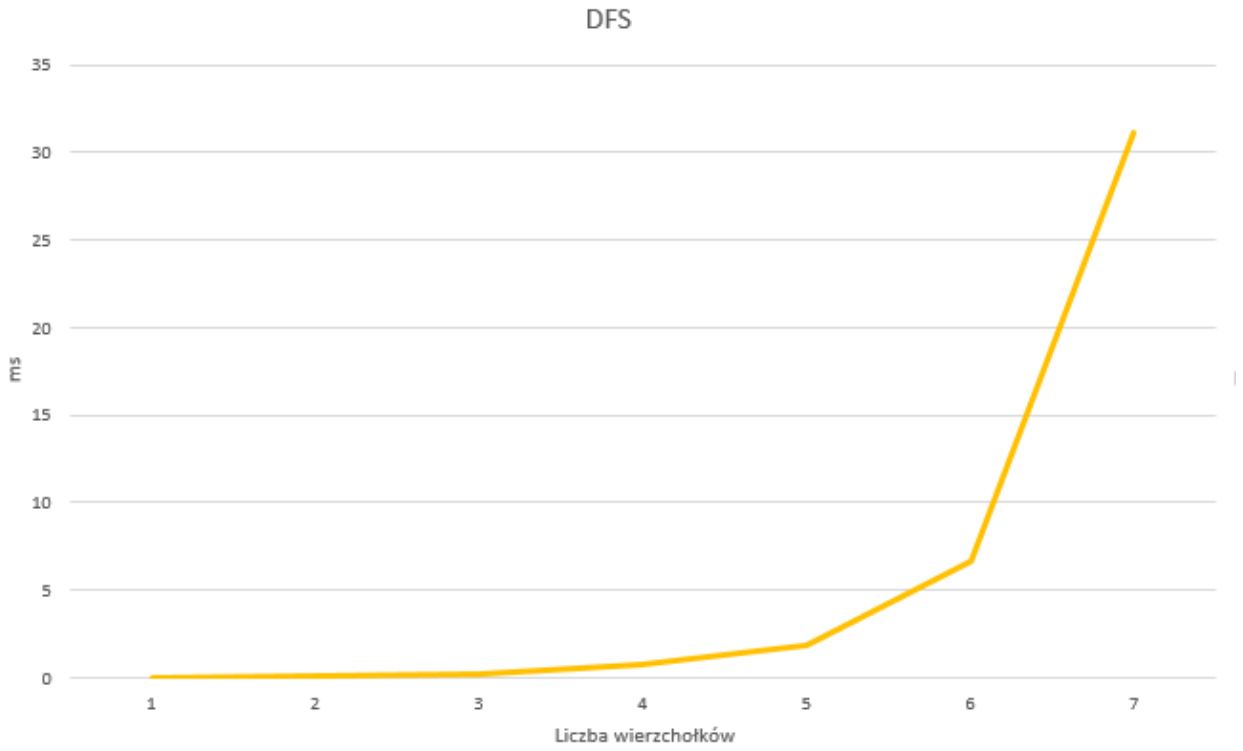
Tabela krawędzi
60
50
40
30
20
10
0
ms
1
2
3
4
5
6
7
Liczba wierzchołków

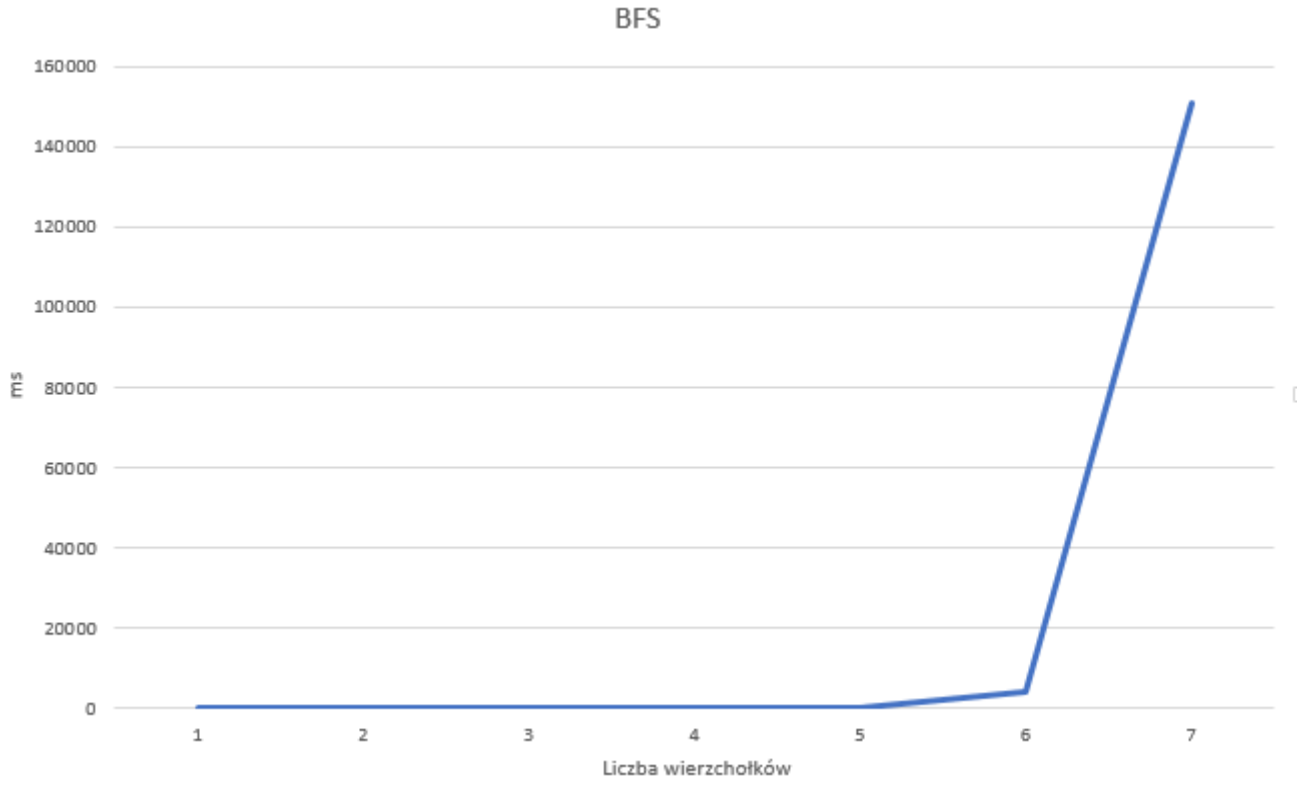
DFS i BFS

DFS (depth first search) – przechodzenie grafu w głąb. Jeden ze sposobów przechodzenia grafu i odwiedzenie wszystkich wierzchołków w grafie. Wybierany jest wierzchołek startowy i odwiedzamy jego następnika i kontynuujemy dla tego następnika dopóki są kolejne nieodwiedzone wierzchołki.

BFS (breadth first search) – przechodzenie grafu wszerz. Sposób przechodzenie grafu. Wybierany wierzchołek startowy jest wrzucamy do kolejki, odwiedzamy wszystkich jego nieodwiedzonych następników i kontynuujemy dla każdego z wierzchołków z kolejki dopóki wierzchołki się nie skończą.

Pomiary casu





Code

```
import random
from timeit import default_timer as timer



def DFS(start, i):
    print(start+1, end = " ")
    wierzch[start] = 1
    for j in range(i):
        if macierz_sasiedstwa[start][j] != 0 and wierzch[j] ==0:
            DFS(j, i)


def BFS(zwiedzony, graf, wierzch):
    zwiedzony.append(wierzch)
    kolej.append(wierzch)
    while kolej:
        start = kolej.pop(0)
        print(start, end = ' ')
        for i in range(len(graf)):
            for j in range(len(graf[i])-1):
                if graf[i][j+1] not in zwiedzony:
                    zwiedzony.append(graf[i][j+1])
                    kolej.append(graf[i][j+1])


print("Liczba wierzchołków: ")
n=int(input())
macierz_sasiedstwa = []
lista_nastepnikow = []
tabela_krawedzi = []
print("1 - random\n2 - macierz sąsiedstwa\n0 - exit")
a = int(input())
```

```
if a==1:
    luki = n*(n-1)/4
    ind = 1

    start_time = timer()
    while luki>=1:
        if len(lista_nastepnikow)<n-1:
            lista_nastepnikow.append([ind, ind+1])
            ind += 1
        else:
            n1=random.randint(1, n-2)
            n2=random.randint(1, n)
            while n1>=n2 or n2 in lista_nastepnikow[n1]:
                n1 = random.randint(1,n-2)
                n2 = random.randint(1,n)
            lista_nastepnikow[n1].append(n2)
        luki-=1
    b = 0
    stop_time = timer()
    list_time = (stop_time - start_time) * (1000)

    start_time = timer()
    for i in range(len(lista_nastepnikow)):
        lista_nastepnikow[i].sort()
        tabela_krawedzi.append([lista_nastepnikow[i][0]])
        for j in range(len(lista_nastepnikow[i]) - 1):
            tabela_krawedzi[b].append(lista_nastepnikow[i][j + 1])
            b += 1
            if j < len(lista_nastepnikow[i]) - 2:
                tabela_krawedzi.append([lista_nastepnikow[i][0]])
    stop_time = timer()
    tabela_time = (stop_time - start_time) * (1000)
```

```
    start_time = timer()
    for i in range(n-1):
        macierz_sasiedstwa.append([])
        for j in range(n):
            if j+1 in lista_nastepnikow[i] and j+1 != lista_nastepnikow[i][0]:
                macierz_sasiedstwa[i].append(1)
            else:
                macierz_sasiedstwa[i].append(0)
    macierz_sasiedstwa.append([])
    for i in range(n):
        macierz_sasiedstwa[n-1].append(0)
    stop_time = timer()
    macierz_time = (stop_time - start_time) * (1000)

elif a==2:
    print("Proszę podać wiersze macierzy: ")
    b=0
    for i in range(n):
        c = False
        macierz_sasiedstwa.append(list(map(int, input().split())))

        start_time = timer()
        for j in range(n):
            if macierz_sasiedstwa[i][j]==1 and c == False:
                lista_nastepnikow.append([])
                lista_nastepnikow[b].append(i + 1)
                c = True
            if macierz_sasiedstwa[i][j]==1:
                lista_nastepnikow[b].append(j+1)
        if c:
            b += 1
```

```
        stop_time = timer()
        list_time = (stop_time - start_time) * (1000)
    b=0
    start_time = timer()
    for i in range(len(lista_nastepnikow)):
        tabela_krawedzi.append([lista_nastepnikow[i][0]])
        for j in range(len(lista_nastepnikow[i])-1):
            tabela_krawedzi[b].append(lista_nastepnikow[i][j+1])
            b+=1
            if j < len(lista_nastepnikow[i])-2:
                tabela_krawedzi.append([lista_nastepnikow[i][0]])
    stop_time = timer()
    tabela_time = (stop_time - start_time) * (1000)


print("1 - Wyświetl macierz sąsiedstwa\n2 - Wyświetl listę następników\n3 - Wyświetl tabelę krawędzi\n4 - Przejście grafu w głąb\n5 - Przejście grafu wszerz\n6 - main menu")
a = int(input())
while a != 6:
    if a == 1:
        print("Macierz sąsiedsta: ", macierz_sasiedstwa)
        print("Czas wykonania: %s ms " % macierz_time)
    elif a == 2:
        print("Lista następników: ", lista_nastepnikow)
        print("Czas wykonania: %s ms " % list_time)
    elif a == 3:
        print("Tabela krawędzi: ", tabela_krawedzi)
        print("Czas wykonania: %s ms", tabela_time)
    elif a == 4:
        print("Przejście w głąb: ", end='')
        start_time = timer()
        wierzch = []
        for i in range(n):
```

```
            wierzch.append(0)
        DFS(0, n)
        stop_time = timer()
        print("\nCzas wykonania: %s ms" % ((stop_time - start_time) * (1000)))
    elif a == 5:
        print("Przejście wszerz: ", end='')
        start_time = timer()
        odwiedzone=[]
        kolej = []
        BFS(odwiedzone, lista_nastepnikow, 1)
        stop_time = timer()
        print("\nCzas wykonania: %s ms" % ((stop_time - start_time) * (1000)))
    elif a==0:
        break
    print("1 - Wyświetl macierz sąsiedstwa\n2 - Wyświetl listę następników\n3 - Wyświetl tabelę krawędzi\n4 - Przejście grafu w głąb\n5 - Przejście grafu wszerz\n6 - exit")
    a = int(input())
```