



# Politechnika Poznańska

Informatyka rok I semestr 2

L10, Piątek 11:45 - 13:15

## Algorytmy i Struktury Danych

Prowadzący: Dominik Piotr Witczak

### Sprawozdanie nr 3

#### Sortowanie topologiczne grafów

**Autor:**

Dominik Fischer 164176

Oliwer Miller 163544

Rok akademicki 2024/2025

# Wprowadzenie

Celem niniejszego grafu jest reprezentacja grafu i jego generacji oraz przedstawienie go w trzech formach: macierz grafu, lista sąsiadów i tabela. Przedstawimy również na wykresach wyniki pomiarów czasowych akcji wykonywanych na grafach, w zależności od liczby wierzchołków.

## Struktura grafu

Klasa *Graph* zawiera odwołanie do wierzchołków, wybranej reprezentacji oraz każdą z tych reprezentacji.

### Terminal

```
class Graph:
    def __init__(self, nodes,
                 representation="list"):
        self.nodes = nodes
        self.representation =
            representation
        self.matrix = [[0] * nodes for _
                       in range(nodes)]
        self.adj_list = [[] for _ in
                        range(nodes)]
        self.table = []
```

## Generacja grafu

Funkcja *generate\_acyclic\_graph()* ma za zadanie wygenerowanie losowego acyklicznego grafu skierowanego z nasyceniem *saturation*, którego wartość jest wprowadzana przy uruchamianiu programu z `--generate`. Tworzona jest lista możliwych krawędzi idących z wierzchołka mniejszego do większego, aby uniknąć tworzenia cykli. Następnie, na podstawie podanego nasycenia, jest wyliczana ilość krawędzi do dodania, które są losowo wybierane z listy i dodawane do grafu.

### Terminal

```
def generate_acyclic_graph(self,
                           saturation):
    possible_edges = [(i, j) for i in
                     range(1, self.nodes) for j in
                     range(i + 1, self.nodes + 1)]
    max_edges = len(possible_edges)
    num_edges_to_add = round((
        saturation / 100) * max_edges)

    selected_edges = random.sample(
        possible_edges, num_edges_to_add
    )

    for u, v in selected_edges:
        self.add_edge(u, v)
```

# Generowanie grafu

Tworzenie grafu przez użytkownika następuje przez użytkownika następuje przez wybór jego reprezentacji, a następnie przypisaniu krawędzi między wybranymi wierzchołkami. Proces jest taki sam dla każdej reprezentacji.

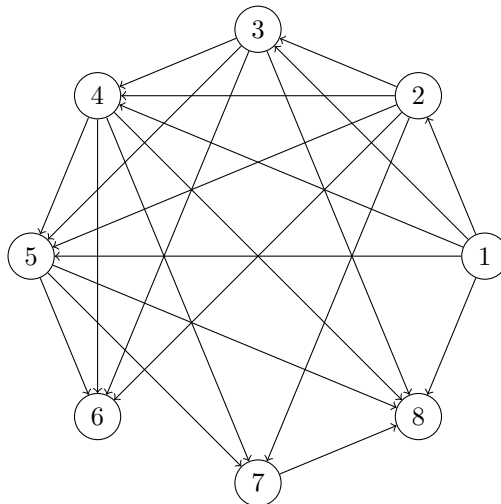
```
oliwer_miller@AsusTufMiller:~/Projekt3$ python3 main.py --user-provided
Dostępne reprezentacje grafu: matrix, list, table
type> matrix
nodes> 8
1> 4 5
2> 3 7
3> 4
4> 8
5> 6
6> 2
7> 8
8> 6
```

Rysunek 1: Tworzenie przez użytkownika

Graf może być również wygenerowany automatycznie. W takim przypadku użytkownik wybiera reprezentację grafu, ilość wierzchołków i wartość nasycenia.

```
oliwer_miller@AsusTufMiller:~/Projekt3$ python3 main.py --generate
Dostępne reprezentacje grafu: matrix, list, table
type> matrix
nodes> 8
saturation (0-100)> 80
Graf wygenerowany!
```

Rysunek 2: Automatyczne generowanie grafu



# Wypisanie grafu

```
action> print
```

	1	2	3	4	5	6	7	8
1	0	0	0	1	1	0	0	0
2	0	0	1	0	0	0	1	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0

(a) Macierz

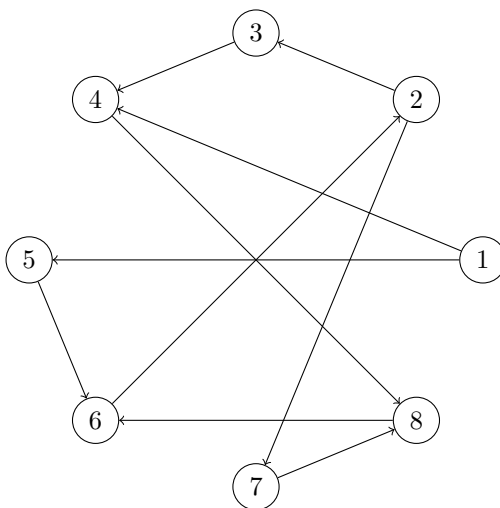
```
action> print
1: 4 5
2: 3 7
3: 4
4: 8
5: 6
6: 2
7: 8
8: 6
```

(b) Lista

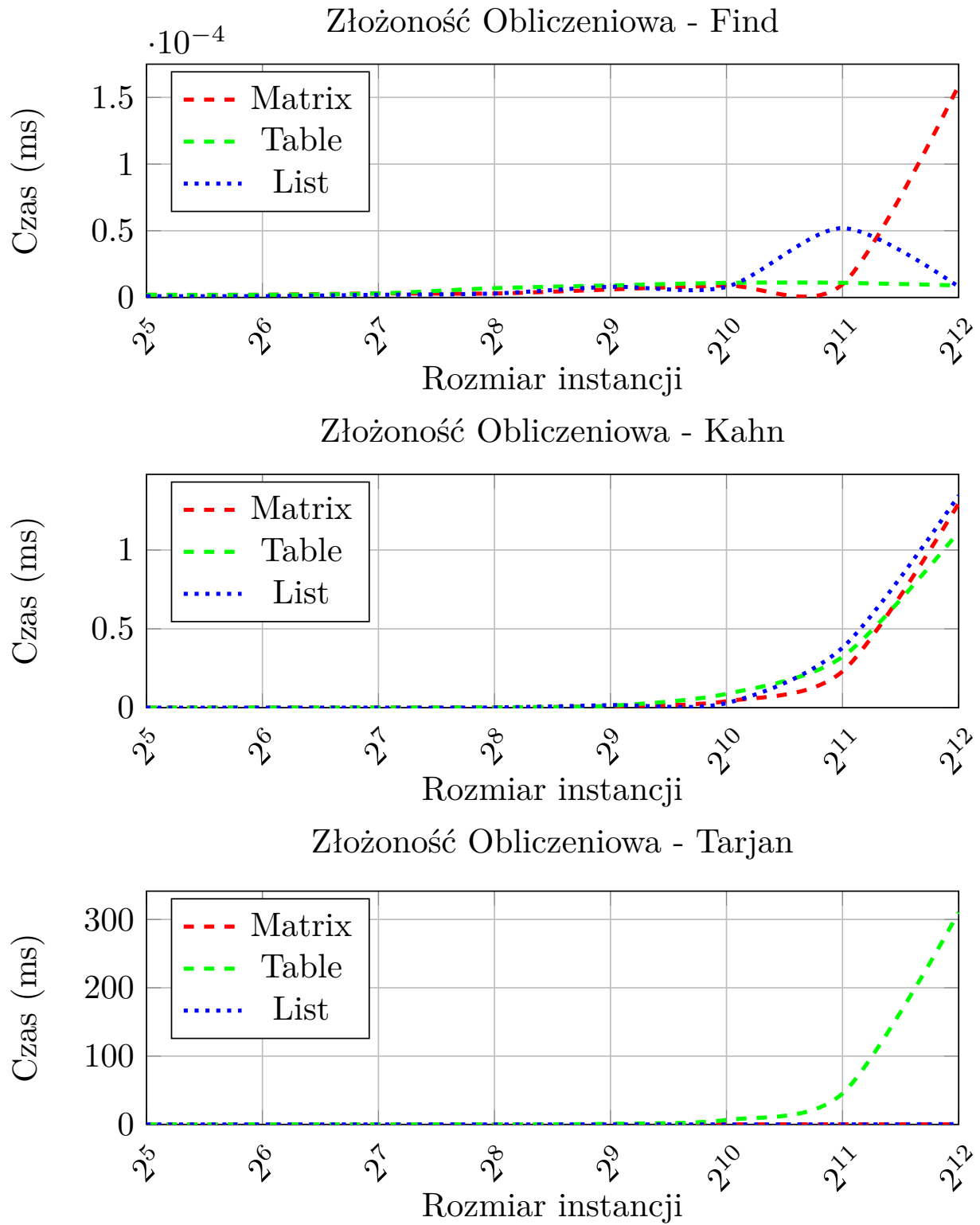
```
action> print
From -> To
1 -> 4
1 -> 5
2 -> 3
2 -> 7
3 -> 4
4 -> 8
5 -> 6
6 -> 2
7 -> 8
8 -> 6
```

(c) Tabela

Rysunek 3: Trzy reprezentacje grafu



# Porównanie czasów wykonania



Rysunek 4: Porównanie czasów wykonania dla 3 operacji i 3 reprezentacji grafów

# Wnioski

Akcja Find ma za zadanie sprawdzić czy istnieje krawędź między dwoma danymi wierzchołkami. Wszystkie reprezentacje grafu wypadają tak samo dobrze do wielkości danych około  $2^{10}$ . Przy większych danych złożoność obliczeniowa Find w macierzy grafu znacznie wzrasta. W przypadku listy złożoność też wzrasta, jednak nie tak gwałtownie, a dla tablicy złożoność pozostaje taka sama.

W sortowaniu topologicznym algorytmem Kahna wszystkie reprezentacje wypadają bardzo podobnie, jedynie w dla danych o wielkości około  $2^{11}$  dana akcja wykonuje się szybciej dla macierzy grafu.

W sortowaniu topologicznym algorytmem Tarjana złożoność obliczeniowa dla tablicy wzrasta od rozmiaru danych wynoszącemu  $2^{10}$  elementów. Złożoność obliczeniowa dla macierzy i listy pozostaje stała.