

## Лабораторна робота 6

**Тема:** Реалізація представлень орграфа. Реалізація пошуку в глибину.

**Ціль:** Засвоїти ефективні методи реалізації представлень орієнтованого графу за допомогою матриці суміжності та за допомогою списків суміжних вершин. Засвоїти метод реалізації обходу графа у глибину.

**Опорні знання:** Мови програмування Паскаль, С. Поняття АД та реалізації АД. АД Орієнтований граф. Алгоритм обходу орієнтованого груфу.

**Завдання:** Ознайомитися з теоретичним матеріалом та виконати завдання, визначені в розділі Хід роботи, підготувати відповіді на конетрольні запитання, оформити протокол виконання роботи.

### Хід роботи

*Завдання 1 Реалізувати представлення орієнтованого графу матрицею суміжності.*

*Завдання 2. Реалізувати представлення орієнтованого графу списками суміжних вершин.*

*Завдання 3. На базі представлення графу списками суміжних рершин реалізувати алгоритм обходу графа у глибину.*

### 1. Опис АД та структури даних Орієнтований граф:

- АД `OrientedGraph` представляє орієнтований граф.
- Структура даних `adjacency_matrix` - матриця суміжності.

```
class OrientedGraph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjacency_matrix = [[0] * vertices for _ in range(vertices)]

    def add_edge(self, start, end):
        self.adjacency_matrix[start][end] = 1

# Приклад використання
graph = OrientedGraph(4)
graph.add_edge(0, 1)
graph.add_edge(1, 2)

# Виведення матриці суміжності на екран
for row in graph.adjacency_matrix:
    print(row)
```

**Результатом виконання буде:**

```
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
```

- АД `OrientedGraph` представляє орієнтований граф
- Структура даних `adjacency_list` - списки суміжних вершин.

```
from collections import defaultdict
```

```
class OrientedGraph:
    def __init__(self):
```

```

self.adjacency_list = defaultdict(list)

def add_edge(self, start, end):
    self.adjacency_list[start].append(end)

# Приклад використання
graph = OrientedGraph()
graph.add_edge(0, 1)
graph.add_edge(1, 2)

# Виведення списків суміжних вершин на екран
for vertex, neighbors in graph.adjacency_list.items():
    print(f"{vertex}: {neighbors}")

```

**Результатом виконання буде:**

```

0: [1]
1: [2]

```

## 2. Програмний код з реалізацією алгоритму обходу графа у глибину:

```

from collections import defaultdict

class OrientedGraph:
    def __init__(self):
        self.adjacency_list = defaultdict(list)

    def add_edge(self, start, end):
        self.adjacency_list[start].append(end)

    def dfs(self, start, visited=None):
        if visited is None:
            visited = set()

        visited.add(start)
        print(start, end=' ')

        for neighbor in self.adjacency_list[start]:
            if neighbor not in visited:
                self.dfs(neighbor, visited)

# Приклад використання
graph = OrientedGraph()
graph.add_edge(0, 1)
graph.add_edge(1, 2)
graph.add_edge(2, 0)
graph.add_edge(2, 3)

# Виведення графа
print("Граф (списки суміжних вершин):")
for vertex, neighbors in graph.adjacency_list.items():
    print(f"{vertex}: {neighbors}")

# Виведення результату обходу графа у глибину
print("\nDFS Result:")
graph.dfs(0)

```

Результатом виконання буде:

```
0: [1]
1: [2]
2: [0, 3]

DFS Result: 0 1 2 3
```

**Контрольні запитання:**

1. Дайте означення орієнтованого графа.
  - Граф, в якому кожне ребро має напрямок, тобто вказує на те, що одна вершина є початковою, а інша - кінцевою.
2. Як формулюється задача обходу графа? Які методи обходів графу є найбільш розповсюдженими?
  - Задача полягає в відвіданні всіх вершин або ребер графа. Найбільш розповсюджені методи обходу - у глибину (DFS) та у ширину (BFS).
3. Які дані є вхідними та вихідними для представлення графу матрицею суміжності?
  - Вхідні - матриця булевих значень, де `graph[i][j]` вказує на наявність ребра між вершинами `i` та `j`. Вихідні - граф.
4. Які дані є вхідними та вихідними для представлення списками суміжних вершин?
  - Вхідні - список списків цілих чисел, де `graph[i]` містить вершини, з'єднані ребрами з вершиною `i`. Вихідні - граф.
5. Які структури даних треба реалізувати для реалізації алгоритму обходу графа у глибину?
  - Стек (або рекурсія для реалізації рекурсивного DFS).
6. Опишіть АДТ Орієнтований граф.
  - Абстрактний тип даних, який представляє собою орієнтований граф з вершинами та напрямленими ребрами.
7. Опишіть алгоритм обходу орієнтованого графа у глибину.
  - Використовує рекурсивний підхід або стек для відвідування вершин у глибину.
8. Опишіть алгоритм обходу орієнтованого графа у ширину.
  - Використовує чергу для відвідування всіх сусідніх вершин на поточному рівні перед переходом до наступного рівня.
9. Оцініть ефективність алгоритму обходу графа у глибину за часом та пам'яттю.
  - DFS має часову складність  $O(V + E)$ , де  $V$  - кількість вершин,  $E$  - кількість ребер, та використовує  $O(V)$  додаткової пам'яті для зберігання інформації про відвідані вершини.