Лабораторна робота 7

Тема: Реалізація алгоритма Краскала
Ціль: Засвоїти ефективні методи реалізації представлень неорієнтованого графу за допомогою матриці суміжності та за допомогою списків суміжних вершин. Засвоїти метод реалізації алгоритму Краскала розв'язання задачі пошуку остовного дерева найменшої вартості.
Опорні знання: Мови програмування Паскаль, С. Поняття АТД та реалізації АТД. АТД Орієнтований граф. Алгоритм обходу орієнтованого груфу.
Завданння: Ознайомитися з теоретичним матеріалом та виконати завдання, визначені в розділі Хід роботы, підготувати відповіді на конетрольні запитання, оформити протокол виконання роботи.

Хід роботи
Завдання 1 Реалізувати представлення позначеного неорієнтованого графу матрицею суміжності.
Завдання 2. Описати АТД для алгоритму Краскала.
Завдання 3. Реалізувати алгоритму Краскала розв'язання задачі пошуку основного дерева найменшої вартості.

1. Формулювання задачі пошуку основного дерева найменшої вартості
Формулювання задачі:
Задача полягає в реалізації представлення позначеного неорієнтованого графу за допомогою матриці суміжності.

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

    def add_edge(self, vertex1, vertex2, weight):
        self.adjacency_matrix[vertex1][vertex2] = weight
        self.adjacency_matrix[vertex2][vertex1] = weight

    def get_neighbors(self, vertex):
        return [i for i, weight in enumerate(self.adjacency_matrix[vertex]) if weight > 0]

    def get_edge_weight(self, vertex1, vertex2):
        return self.adjacency_matrix[vertex1][vertex2]
```

Формулювання задачі:
Задача полягає в описі АТД для алгоритму Краскала для пошуку остовного дерева найменшої вартості.

2. Опис АТД та структури даних Неорієнтований граф позначений граф.
Опис АТД та структури даних:
АТД (Абстрактний тип даних): Граф
Структура даних:
Graph
Властивості:
vertex_count: Кількість вершин у графі.
adjacency_matrix: Матриця суміжності.
Методи:
add_edge(vertex1, vertex2, weight): Додає ребро між вершинами з вагою.
get_neighbors(vertex): Повертає список сусідів для даної вершини.
get_edge_weight(vertex1, vertex2): Повертає вагу ребра між двома вершинами.

Опис АТД та структури даних:
АТД (Абстрактний тип даних): Краскал

Структура даних:
Kruskal
Властивості:
graph: Граф, для якого виконується алгоритм Краскала.
Методи:
find_parent(subset, i): Знаходить корінь (представника) множини, до якої належить вершина i.
union(subset, i, j): Об'єднує дві множини за їхніми коренями.
kruskal_algorithm(): Виконує алгоритм Краскала для пошуку остовного дерева найменшої вартості.

3. Програмний код з реалізацією алгоритму Краскала.

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

    def add_edge(self, vertex1, vertex2, weight):
        self.adjacency_matrix[vertex1][vertex2] = weight
        self.adjacency_matrix[vertex2][vertex1] = weight

    def get_neighbors(self, vertex):
        return [i for i, weight in enumerate(self.adjacency_matrix[vertex]) if weight > 0]

    def get_edge_weight(self, vertex1, vertex2):
        return self.adjacency_matrix[vertex1][vertex2]


class Kruskal:
    def __init__(self, graph):
        self.graph = graph

    def find_parent(self, subset, i):
        if subset[i] == -1:
            return i
        return self.find_parent(subset, subset[i])

    def union(self, subset, i, j):
        i_root = self.find_parent(subset, i)
        j_root = self.find_parent(subset, j)
        subset[i_root] = j_root

    def kruskal_algorithm(self):
        result = []
        edge_list = []

        for i in range(self.graph.vertex_count):
            for j in self.graph.get_neighbors(i):
                edge_list.append((i, j, self.graph.get_edge_weight(i, j)))

        edge_list = sorted(edge_list, key=lambda edge: edge[2])

        subset = [-1] * self.graph.vertex_count

        for edge in edge_list:
            i, j, weight = edge
            i_root = self.find_parent(subset, i)
            j_root = self.find_parent(subset, j)

            if i_root != j_root:
```

```
                result.append((i, j, weight))
                self.union(subset, i_root, j_root)

        # Виведення результатів на екран
        print("Остовне дерево найменшої вартості (Краскал):")
        total_weight = 0
        for edge in result:
            i, j, weight = edge
            print(f"Ребро {i} - {j}, вага: {weight}")
            total_weight += weight
        print(f"Загальна вага остовного дерева: {total_weight}")

        return result


# Створення графу та виведення матриці суміжності
graph = Graph(5)
graph.add_edge(0, 1, 2)
graph.add_edge(0, 2, 4)
graph.add_edge(1, 2, 1)
graph.add_edge(1, 3, 7)
graph.add_edge(2, 4, 3)

print("Матриця суміжності:")
for row in graph.adjacency_matrix:
    print(row)

# Створення об'єкту для алгоритму Краскала та виклик алгоритму
kruskal = Kruskal(graph)
result = kruskal.kruskal_algorithm()
```

Результатом виконання буде:

```
Матриця суміжності:
[0, 2, 4, 0, 0]
[2, 0, 1, 7, 0]
[4, 1, 0, 0, 3]
[0, 7, 0, 0, 0]
[0, 0, 3, 0, 0]
Остовне дерево найменшої вартості (Краскал):
Ребро 1 - 2, вага: 1
Ребро 0 - 1, вага: 2
Ребро 2 - 4, вага: 3
Ребро 1 - 3, вага: 7
Загальна вага остовного дерева: 13
```

Контрольні запитання:

1. Означення неорієнтованого графа:

Неорієнтований граф - це граф, в якому ребра не мають напрямку, тобто вони не вказують на порядок між двома вершинами.

2. Означення позначеного графа:
Позначений граф - це граф, у якому кожне ребро має асоційований з ним ваговий або деякий інший тип позначення.

3. Формулювання задачі пошуку остовного дерева найменшої вартості та методи розв'язання:
Задача полягає у знаходженні підграфа графа, який є деревом та включає всі вершини початкового графа, але при цьому має мінімальну суму ваг ребер. Найбільш відомими методами розв'язання цієї задачі є алгоритми Краскала та Прима.

4. Дані для представлення неорієнтованого позначеного графу матрицею суміжності:
Вхідні дані включають матрицю суміжності, де елемент на позначеному рядку і стовпці представляє вагу або наявність ребра між відповідними вершинами.

5. Структури даних для реалізації алгоритму Краскала:
Для реалізації алгоритму Краскала зазвичай використовують структури даних, такі як граф (з матрицею суміжності або списками суміжних вершин), структури дерева для представлення лісу підграфів та структури даних Union-Find для ефективного об'єднання множин вершин.

6. АТД Неорієнтований позначений граф:
- *Вершини: Множина вершин графу.*
- *Ребра: Множина ребер, де кожне ребро може мати вагу та позначення.*
- *Додавання вершини та ребра.*
- *Отримання списку суміжних вершин для заданої вершини.*

7. АТД для алгоритму Краскала:
- *Додавання ребра: Додавання ребра з вагою до графу.*
- *Запуск алгоритму: Запуск алгоритму Краскала для знаходження остовного дерева найменшої вартості.*
- *Отримання результату: Отримання ребер остовного дерева та їх ваг.*

8. Оцінка ефективності алгоритму Краскала:
- *Час: O(E log V), де E - кількість ребер, V - кількість вершин.*
- *Пам'ять: O(V + E), де V - кількість вершин, E - кількість ребер.*