Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

1. Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин.
    - Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20].
    - Веса ребер задаются случайным значением от 1 до 20.
    - Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой.
2. Выведите получившийся граф в виде матрицы смежности.
3. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста., необходимо:
    - Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Краскала.

Описание метода/модели.

Граф

Граф — это структура данных, состоящая из вершин (узлов) и рёбер (связей между узлами). Граф может быть ориентированным (направленные рёбра) или неориентированным, а также взвешенным (с весами рёбер) или невзвешенным.

Модель алгоритма Краскала:

Алгоритм Краскала используется для поиска минимального остовного дерева (MST) в связном, взвешенном и неориентированном графе.

Принцип работы:

- Отсортировать все рёбра графа по весу (от меньшего к большему).
- Инициализировать пустое множество для рёбер остовного дерева.
- Последовательно добавлять рёбра, если они не образуют цикл (используется структура "Объединение-Представитель" (Union-Find)).
- Повторять, пока в дереве не окажется (V - 1) рёбер (где V — число вершин).

Сложность алгоритма:

Основная сложность — сортировка рёбер O(E log E), а проверки на циклы (Union-Find) работают почти за O(1). Итоговая сложность O(E log V).

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

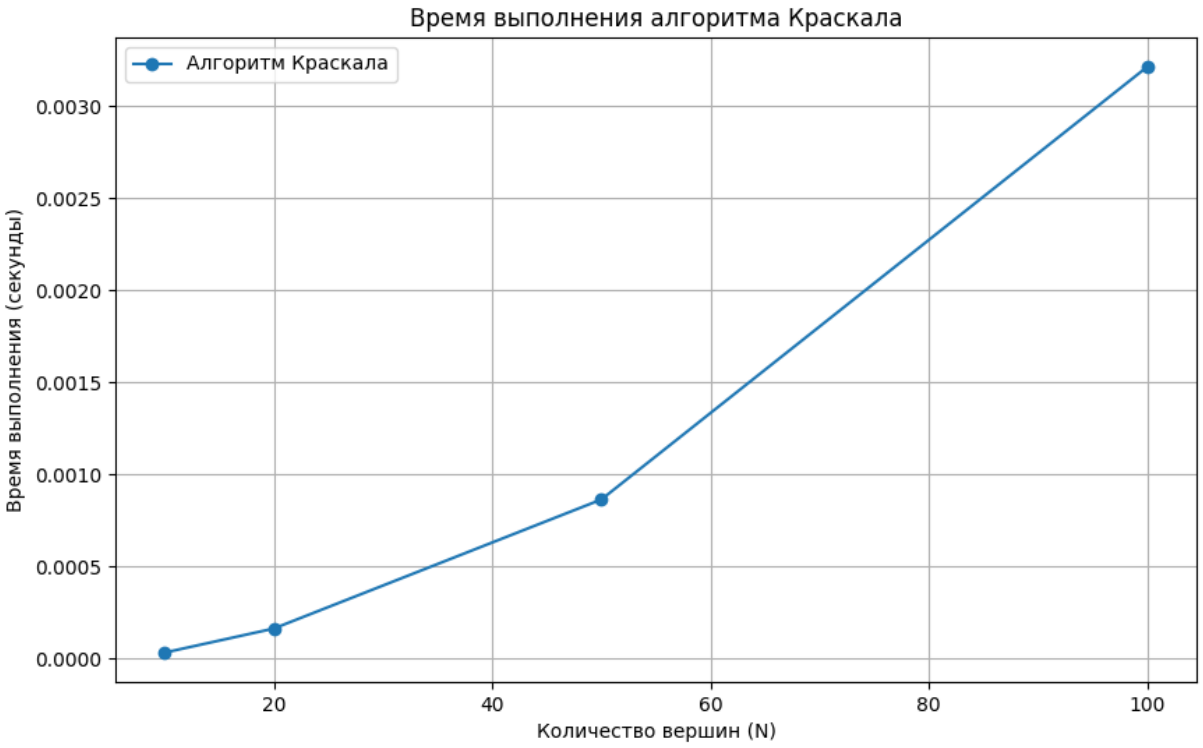
Блок кода отвечающий за реализацию графа и алгоритма Краскала:

```
    class WeightedGraph:
        def __init__(self, vertices: int):
            """Инициализация графа."""
            self.vertices = vertices
            self.edges = []  # Список рёбер (u, v, вес)
            self.adj_matrix = [[0] * vertices for _ in range(vertices)]  # Матрица смежности
        def add_edge(self, u: int, v: int, weight: int):
            """Добавление ребра в граф."""
            self.edges.append((u, v, weight))
            self.edges.append((v, u, weight))  # Для неориентированного графа
            self.adj_matrix[u][v] = weight
            self.adj_matrix[v][u] = weight
        def generate_random_graph(self, min_edges_per_vertex: int, weight_range: tuple = (1, 20)):
            """Генерация случайного связного графа."""
            # Шаг 1: Создаем связный граф (на случай, если вершины не связаны)
            for u in range(self.vertices - 1):
                weight = random.randint(*weight_range)
                self.add_edge(u, u + 1, weight)
            # Шаг 2: Добавляем дополнительные рёбра случайным образом
            for u in range(self.vertices):
                num_edges = random.randint(min_edges_per_vertex, self.vertices - 1)
                connected = set(v for u_, v, _ in self.edges if u_ == u)  # Уже связные вершины
                while len(connected) < num_edges:
                    v = random.randint(0, self.vertices - 1)
                    if u != v and v not in connected:  # Исключаем петли и повторяющиеся рёбра
                        weight = random.randint(*weight_range)
                        self.add_edge(u, v, weight)
                        connected.add(v)
        def print_adjacency_matrix(self):
            """Вывод матрицы смежности для графа."""
            for row in self.adj_matrix:
                print(" ".join(f"{weight:2}" for weight in row))
    # ---------- Алгоритм Краскала ----------
    def kruskal(graph: WeightedGraph):
        """Алгоритм Краскала для построения минимального остовного дерева."""
        # Сортируем рёбра по весу
```

Блоки кода отвечающие за тестирование:

```
def test_kruskal():
    vertex_sets = [10, 20, 50, 100]  # Количество вершин для графов
    min_edges = [3, 4, 10, 20]
    iterations = 5  # Количество тестов на один граф
    results = []
    for vertices, min_edges_per_vertex in zip(vertex_sets, min_edges):
        times = []
        for _ in range(iterations):
            # Генерация графа
            graph = WeightedGraph(vertices)
            graph.generate_random_graph(min_edges_per_vertex)
            # Печать графа (опционально включить для отладки)
            print(f"\nГраф с {vertices} вершинами:")
            graph.print_adjacency_matrix()
            # Замер времени на выполнение алгоритма Краскала
            start_time = time.time()
            mst = kruskal(graph)
            elapsed_time = time.time() - start_time
            times.append(elapsed_time)
            # Печать результата остовного дерева
            print(f"\nОстовное дерево (MST): {mst}")
            print(f"Время выполнения: {elapsed_time:.5f} сек")
        # Записываем среднее время
        average_time = sum(times) / len(times)
        results.append((vertices, average_time))
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже, а так же в терминале выводятся графы и их остовые деревья (данные так же приводятся ниже в виде файла).



Фрагмент вывода данных:

```
  Граф                  с                  10                  вершинами:
 0         5     17     12          3          3          0     10          0          0
 5         0          9          9          0     14          0          0          0          1
17          9          0          7          0     14     13          0          0     20
12          9          7          0          6          9     18     13     14     10
 3          0          0          6          0          2          4     10     17     18
 3     14     14          9          2          0          4          0     10     13
 0          0     13     18          4          4          0          4          9          2
10          0          0     13     10          0          4          0     12     19
 0          0          0     14     17     10          9     12          0     10
 0          1     20     10     18     13          2     19     10          0

Остовное дерево (MST): [(1, 9, 1), (4, 5, 2), (9, 6, 2), (0, 4, 3), (5, 6, 4),
(6,    7,    4),    (3,    4,    6),    (2,    3,    7),    (6,    8,    9)]
Время выполнения: 0.00052 сек
```

Файл с выводом данных находится в репозитории в директории Lab_5 и называется output.txt: https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git

Заключение.

Алгоритм Краскала эффективно строит **минимальное остовное дерево (MST)**, особенно на **разреженных графах**, благодаря использованию **сортировки рёбер и структуры Union-Find**.

- **Производительность:** Основное время занимает сортировка рёбер (**O(E log E)**), а объединение и поиск сжатия пути в **Union-Find** выполняются почти за **O(1)**. Итоговая сложность **O(E log V)**, что эффективно при малом числе рёбер.
- **Тестирование:** Важно проверять на различных графах, включая разреженные, плотные, с изолированными вершинами и уже остовные.
- **Альтернативы:** Для **плотных графов** алгоритм **Прима** может быть предпочтительнее (**$O(V^2)$** или **O(E + V log V)** с кучей).

В целом, Краскал — мощный алгоритм для MST, особенно при **хранении рёбер списком**.