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-30.. Лихолат Полина Николаевна
Ссылка на репозиторий: ....https://github.com/MUCTR-IKT-CPP/Likholat_algorithms.git

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

Дата сдачи: 02.04.2023

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.

   (Можно использовать генератор из предыдущей лабораторной работы.)
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:
   - **Вариант 3**. Найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Флойда — Уоршелла.
4. В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график, используя полученные замеры времени, где на ось абсцисс (X) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

Описание метода/модели.

Алгоритм Флойда-Уоршелла - это алгоритм нахождения кратчайших путей между всеми парами вершин в ориентированном или неориентированном взвешенном графе. Алгоритм назван в честь его создателей Роберта Флойда и Стивена Уоршелла, которые впервые опубликовали его в 1962 году.

Основная идея алгоритма Флойда-Уоршелла заключается в использовании динамического программирования. Алгоритм поддерживает матрицу расстояний между всеми парами вершин и на каждой итерации добавляет новые вершины к пути, который может быть сокращен, используя уже рассчитанные расстояния между остальными вершинами.

Алгоритм начинается с матрицы расстояний, которая заполняется весами ребер между соответствующими вершинами графа. Затем на каждой итерации алгоритм пересчитывает матрицу расстояний, добавляя новые вершины к пути и обновляя значения расстояний между всеми парами вершин. Когда все вершины будут добавлены к пути, матрица расстояний будет содержать кратчайшие расстояния между всеми парами вершин.

Алгоритм Флойда-Уоршелла имеет сложность O(n^3), где n - количество вершин в графе. Он может быть использован для решения задач, связанных с нахождением кратчайшего пути между всеми парами вершин, таких как определение кратчайшего пути в транспортных сетях.

Выполнение задачи.

Алгоритм Флойда-Уоршелла для поиска кратчайших путей между всеми парами вершин в графе реализован в виде метода класса Graph. Вот как это работает:

1. Сначала мы копируем матрицу смежности нашего графа в переменную d. Эта матрица будет содержать длины кратчайших путей между всеми парами вершин в графе.
2. Затем мы запускаем три вложенных цикла, которые проходят по всем парам вершин в графе и на каждой итерации пытаются обновить длину кратчайшего пути между этими вершинами.
3. На каждой итерации внешнего цикла мы выбираем вершину k в качестве промежуточной вершины, через которую мы можем попытаться найти более короткий путь между вершинами i и j.
4. Затем мы проверяем, существуют ли ребра между вершинами i и k, и между вершинами k и j. Если оба ребра существуют, мы можем попытаться найти новый путь между вершинами i и j, который проходит через вершину k.
5. Если новый путь короче, чем текущий кратчайший путь между вершинами i и j, мы обновляем значение в матрице d на новое значение.
6. В конце алгоритма мы возвращаем матрицу d, которая содержит длины кратчайших путей между всеми парами вершин в графе.

```
/**
 * Computes the shortest path between every pair of vertices in the graph using the Floyd-
Warshall algorithm.
 *
 * @return a vector of vectors representing the shortest distances between all pairs of
vertices.
 */
vector<vector<int>> Graph::floyd_warshall() {
	// Copy the adjacency matrix.
	vector<vector<int>> d = get_adj_matrix();

	// Apply the Floyd-Warshall algorithm.
	for (int k = 0; k < vertices_; k++) {
		for (int i = 0; i < vertices_; i++) {
			for (int j = 0; j < vertices_; j++) {
				if (d[i][k] != 0 && d[k][j] != 0) {
					if (d[i][j] > d[i][k] + d[k][j]) {
						d[i][j] = d[i][k] + d[k][j];
					}
				}
			}
		}
	}
```

```
    return d;
}
```

Для генерации графов использовался измененный генератор из прошлой лабораторной работы.

```
Graph generate_graph(int num_vertices, int min_edges_per_vertex) {
    Graph g(num_vertices);    // Create a connected graph with num_vertices edges using BFS
    queue<int> q;
    vector<bool> visited(num_vertices, false);
    visited[0] = true;
    q.push(0);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        int num_edges_added = 0;
        for (int v = 0; v < num_vertices; v++) {
            if (u == v || g.get_adj_matrix()[u][v] != 0) {
                continue;
            }
            int weight = rand() % 20 + 1;
            g.add_edge(u, v, weight);
            num_edges_added++;
            if (num_edges_added == num_vertices - 1) {
                break;
            }
        }
        for (int v = 0; v < num_vertices; v++) {
            if (g.get_adj_matrix()[u][v] != 0 && !visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }    // Add additional edges until each vertex has at least min_edges_per_vertex
    for (int u = 0; u < num_vertices; u++) {
        while (g.get_degree(u) < min_edges_per_vertex + 1) {
            int v = rand() % num_vertices;
            if (u == v || g.get_adj_matrix()[u][v] != 0) {
                continue;
            }
            int weight = rand() % 20 + 1;
            g.add_edge(u, v, weight);
        }
    }

    return g;
}
```

Данный алгоритм генерирует случайный взвешенный связный граф с заданным количеством вершин и минимальным количеством ребер на каждую вершину. Алгоритм выполняет следующие шаги:

1. Создает граф с заданным количеством вершин.
2. Используя алгоритм обхода в ширину (BFS), добавляет случайные ребра между вершинами, пока каждая вершина не будет иметь по крайней мере одно ребро.
3. Для каждой вершины добавляет дополнительные ребра до тех пор, пока у каждой вершины не будет по крайней мере заданное минимальное количество ребер.

На каждом шаге добавления ребра между вершинами, генерируется случайный вес ребра, выбирая число в диапазоне от 1 до 20.

Построим график зависимости времени поиска кратчайших путей от количества вершин в графе, используя Python.

```
import numpy as np
import matplotlib.pyplot as plt

# Задаем данные
n = [10, 20, 50, 100]
times = [[0.0941, 0.0758, 0.0712, 0.2697, 0.0805],
         [0.3935, 0.5374, 0.4231, 0.5592, 0.4801],
         [5.6365, 5.7859, 6.2985, 6.5246, 5.2811],
         [44.0427, 42.2208, 42.5902, 42.9874, 46.2117]]

# Среднее время для каждого количества вершин
mean_times = [np.mean(times[i]) for i in range(len(times))]

# Строим график
plt.plot(n, mean_times)
plt.xlabel('Количество вершин')
plt.ylabel('Время (мс)')
plt.title('Зависимость времени от количества вершин')
plt.show()
```

По графику (рис. 1.1) видно, что время выполнения алгоритма растет примерно линейно с ростом количества вершин в графе. Это связано с тем, что при увеличении числа вершин возрастает количество ребер, которые нужно просмотреть для выполнения алгоритма, что в свою очередь занимает больше времени.

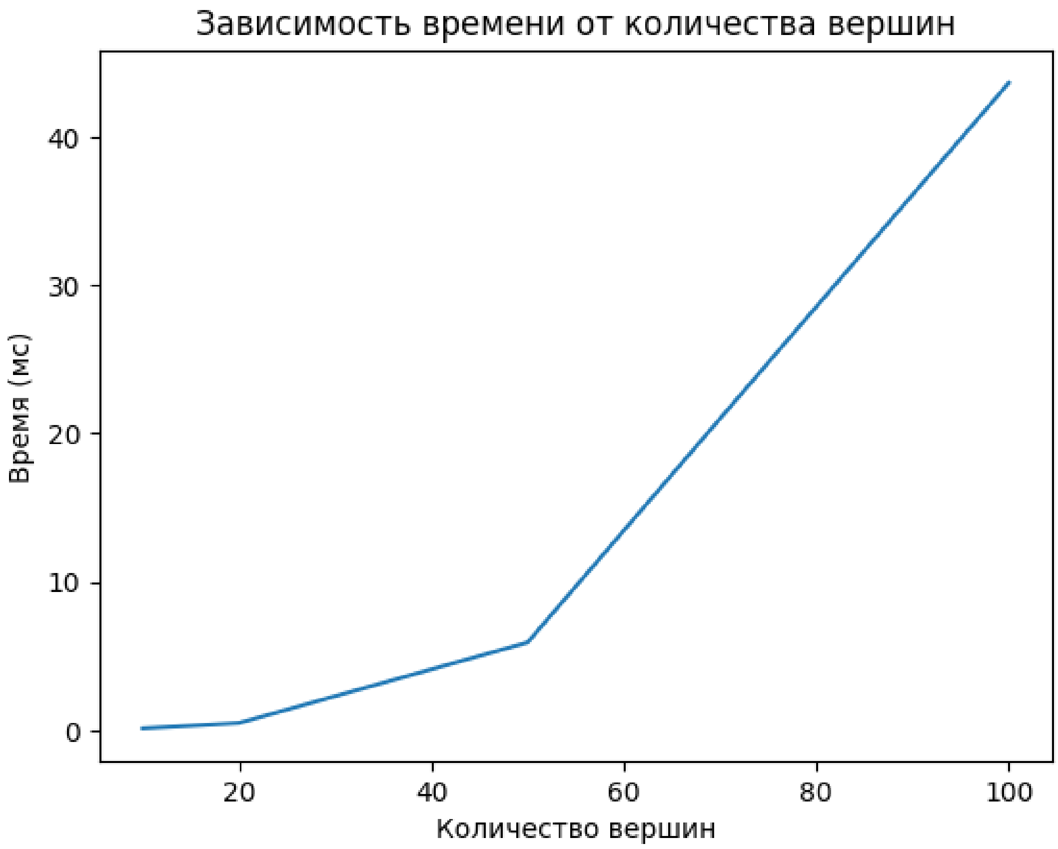

Рисунок 1.1. График зависимости времени поиска от количества вершин

Заключение.

В данной лабораторной работе мы реализовали алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми вершинами графа. Изучив алгоритм, мы можем сделать выводы о его преимуществах и недостатках.

Преимущества алгоритма Флойда-Уоршелла:

- Работает с любыми типами графов, включая графы с отрицательными весами ребер и графы с циклами.
- Гарантирует нахождение кратчайших путей между всеми парами вершин в графе.
- Может быть использован для обнаружения отрицательных циклов в графе.

Недостатки алгоритма Флойда-Уоршелла:

- Имеет кубическую сложность по времени и по памяти, что может привести к проблемам с производительностью при работе с большими графами.
- Может быть неэффективным для решения задач, которые требуют нахождения кратчайших путей только между несколькими парами вершин, так как он вычисляет кратчайшие пути между всеми парами вершин в графе.

В целом, алгоритм Флойда-Уоршелла является хорошим выбором для решения задач нахождения кратчайшего пути между всеми парами вершин в графе, если размер графа не слишком велик. Однако для более сложных задач, может потребоваться использование более эффективных алгоритмов с лучшей производительностью.