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-38.. Прилепский Артем Сергеевич
Ссылка на репозиторий: ....github.com/news1d/Algorithms_and_structures

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

Дата сдачи: 31.03.2023

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.

2. Выведите получившийся граф в виде матрицы смежности. Пример вывода данных:
3. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста. Необходимо:
   - **Вариант 1**. Найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Дейкстры.
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.

Взвешенный граф – это такой граф каждое ребро которого сопоставимо с каким либо числовым значением называемым весом графа. Вес ребра в графе может отражать какой-либо параметр в системе которая этим графом описывается. Например, если есть набор пунктов назначения соединённых друг с другом дорогами веса на таком графе могут означать длину этих дорог.

Алгоритм Дейкстры находит кратчайший путь от заданной вершины ко всем другим вершинам графа, включая требуемую конечную вершину t.

Кратчайший-путь-Дейкстры (Граф, начальная, конечная)

Из начальной точки до начальной точки путь занимает 0 единиц.

Итерируем от 1 до n, устанавливаем дистанции для каждой вершины в максимум

Начиная с начальной точки и до тех пор пока есть новый узел:

Начиная с первого ребра от текущей точки и до тех пор пока есть необследованные ребра

Берем вес текущего ребра и номер конечной вершины.

Сравниваем уже вычисленное ранее расстояние до конечной вершины ребра и дистанцию до текущего узла суммированную с весом ребра, выбираем минимальное и записываем как дистанцию до конечной вершины ребра, а так же значение родительского для измененного пути.

Выбираем следующей вершиной ту, которая имеет минимальное расстояние от изначальной.

Важно помнить, что алгоритм Дейкстры не работает в графах с отрицательным значением. Сложность алгоритма $O(n^2)$

Путь из алгоритма получается обратным раскручиванием массива родителей.

Выполнение задачи.

Для реализации программы был выбран язык программирования Python.

1) Генерируем взвешенные графы с нужными параметрами используя функцию.

   Код функции:

```
def generate_graph(vertices, min_edges):
    # Создаем пустой граф
    G = {}
    # Добавляем вершины
    for i in range(vertices):
        G[i] = {}
    # Задаем случайное количество связей каждой вершины
    for i in range(vertices):
        num_edges = random.randint(min_edges, int(vertices / 2))
        # Соединяем вершины случайными ребрами с весами от 1 до 20
        for j in random.sample(range(vertices), num_edges):
            if i != j:
                G[i][j] = random.randint(1, 20)
                G[j][i] = G[i][j]
    return G
```

   Код генерации:

```
# Создаем список графов для каждого количества вершин
graphs_list = []
for i in range(len(num_vertices)):
    graph = generate_graph(vertices=num_vertices[i], min_edges=num_min_edges[i])
    graphs_list.append(graph)
```

2) Выводим получившиеся графы в виде матрицы смежности.

   Код:

```
# Выводим матрицы смежности для каждого графа
for graph in graphs_list:
    print(f"\nМатрица смежности для графа с {len(graph)} вершинами:")
    print(adjacency_matrix(graph), sep='\n')
```

```
Матрица смежности для графа с 10 вершинами:
[[ 0 17  0  2 17  5  0 17  0  2]
 [17  0  0  2  1  0  0  0 15  2]
 [ 0  0  0  0 11  0  2 14  1 17]
 [ 2  2  0  0  0  0  8 20 11  8]
 [17  1 11  0  0 17  0  3  0  5]
 [ 5  0  0  0 17  0 13  9  7  5]
 [ 0  0  2  8  0 13  0 11 20  0]
 [17  0 14 20  3  9 11  0  3  0]
 [ 0 15  1 11  0  7 20  3  0  0]
 [ 2  2 17  8  5  5  0  0  0  0]]
```

Пример вывода

3) Находим кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Дейкстры и проводим серию тестов.

Код функции:

```
def dijkstra(graph):
    # Находим кратчайшие пути между всеми парами вершин
    for i in range(len(graph)):
        dist = {j: float('inf') for j in graph}
        dist[i] = 0
        prev = {j: None for j in graph}
        visited = list()
        while len(visited) < len(graph):
            # Находим вершину с минимальным расстоянием до начальной вершины
            min_dist = float('inf')
            min_vertex = None
            for vertex in graph:
                if vertex not in visited and dist[vertex] < min_dist:
                    min_dist = dist[vertex]
                    min_vertex = vertex
            # Обновляем расстояния до смежных вершин
            for neighbor, weight in graph[min_vertex].items():
                if dist[min_vertex] + weight < dist[neighbor]:
                    dist[neighbor] = dist[min_vertex] + weight
                    prev[neighbor] = min_vertex
            visited.append(min_vertex)
        # Выводим длину кратчайшего пути и пройденные вершины
        for j in range(i+1, len(graph)):
            path = [j]
            vertex = j
            while prev[vertex] is not None:
                path.append(prev[vertex])
                vertex = prev[vertex]
            path.reverse()
            print(f"Кратчайшим путем между вершинами {i} и {j} является {path} и
равен {dist[j]}")
```

Код тестов:

```
# Задаем количество тестов
num_tests = 5
for i in range(len(num_vertices)):
    for j in range(num_tests):
        graph = generate_graph(vertices=num_vertices[i],
min_edges=num_min_edges[i])
```

```
        print(f'\nТест {j + 1}/{num_tests} для графа с {len(graph)} вершинами')
        dijkstra(graph)
```

```
Тест 1/5 для графа с 10 вершинами
Кратчайшим путем между вершинами 0 и 1 является [0, 9, 1] и он равен 4
Кратчайшим путем между вершинами 0 и 2 является [0, 3, 6, 2] и он равен 12
Кратчайшим путем между вершинами 0 и 3 является [0, 3] и он равен 2
Кратчайшим путем между вершинами 0 и 4 является [0, 9, 1, 4] и он равен 5
Кратчайшим путем между вершинами 0 и 5 является [0, 5] и он равен 5
Кратчайшим путем между вершинами 0 и 6 является [0, 3, 6] и он равен 10
Кратчайшим путем между вершинами 0 и 7 является [0, 9, 1, 4, 7] и он равен 8
Кратчайшим путем между вершинами 0 и 8 является [0, 9, 1, 4, 7, 8] и он равен 11
Кратчайшим путем между вершинами 0 и 9 является [0, 9] и он равен 2
Кратчайшим путем между вершинами 1 и 2 является [1, 4, 7, 8, 2] и он равен 8
Кратчайшим путем между вершинами 1 и 3 является [1, 3] и он равен 2
Кратчайшим путем между вершинами 1 и 4 является [1, 4] и он равен 1
...
Кратчайшим путем между вершинами 4 и 8 является [4, 7, 8] и он равен 6
Кратчайшим путем между вершинами 4 и 9 является [4, 1, 9] и он равен 3
Кратчайшим путем между вершинами 5 и 6 является [5, 8, 2, 6] и он равен 10
Кратчайшим путем между вершинами 5 и 7 является [5, 7] и он равен 9
Кратчайшим путем между вершинами 5 и 8 является [5, 8] и он равен 7
Кратчайшим путем между вершинами 5 и 9 является [5, 9] и он равен 5
Кратчайшим путем между вершинами 6 и 7 является [6, 2, 8, 7] и он равен 6
Кратчайшим путем между вершинами 6 и 8 является [6, 2, 8] и он равен 3
Кратчайшим путем между вершинами 6 и 9 является [6, 3, 1, 9] и он равен 12
Кратчайшим путем между вершинами 7 и 8 является [7, 8] и он равен 3
Кратчайшим путем между вершинами 7 и 9 является [7, 4, 1, 9] и он равен 6
Кратчайшим путем между вершинами 8 и 9 является [8, 7, 4, 1, 9] и он равен 9
```

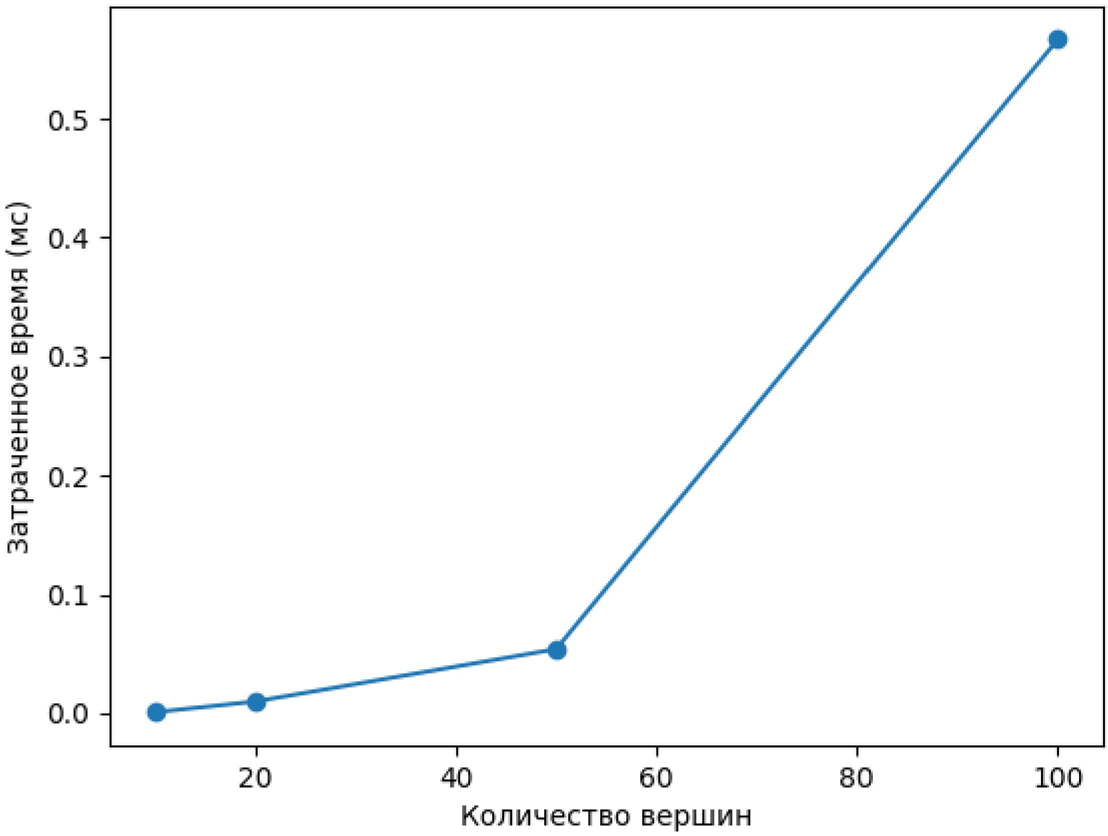
Пример вывода кратчайших путей

4) Замеряем время потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин и строим график, где на ось абсцисс (X) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

<u>Код:</u>

```
# Создаем список для хранения времени
times = []
# Запускаем тесты для каждого графа и сохраняем время выполнения
for graph in graphs_list:
    start_time = time.time()
    dijkstra(graph)
    end_time = time.time()
    times.append(end_time - start_time)

# Строим график
plt.plot(num_vertices, times, 'o-')
plt.xlabel('Количество вершин')
plt.ylabel('Затраченное время (мс)')
plt.show()
```



Полученный график

Заключение.

Алгоритм Дейкстры хорошо работает на графах без отрицательных весов ребер и где отсутствуют циклы отрицательного веса. В таких случаях он находит кратчайшие пути без ошибок и быстро.