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 Суханова Евгения Валерьевна
Ссылка на репозиторий:
https://github.com/MUCTR-IKT-CPP/EVSuhanova_30/blob/main/Algoritms/laba5.cpp

Приняли: Пысин Максим Дмитриевич
Краснов Дмитрий Олегович

Дата сдачи: 20.03.2023

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: Матрица смежности
3. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста, необходимо 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 предназначен для получения кратчайших путей между всеми парами вершин существующих в графе (ориентированном или неориентированном).

Результатом работы этого алгоритма является матрица N x N, где N количество вершин. Особенностью которую нужно учитывать, является то, что в матрицах смежности для взвешенного графа нельзя использовать 0 для указания несуществующего пути, лучше использовать максимальное значение.

Поиск всех кратчайших путей Флойда (Матрица смежности)

1. Инициируем 3 счетчика, 2 счетчика для матрицы и 1 для номера промежуточной вершины.
2. Итерируем по счетчику промежуточных вершин до тех пор пока не кончатся вершины
3. Итерируем по первому счетчику вершин, до тех пор пока не кончатся вершины
4. Итерируем по второму счетчику вершин, до тех пор пока не кончатся вершины
5. Считаем длину пути используя промежуточную вершину как узловую точку, т.е. от вершины по первому счетчику до узловой плюс от узловой до вершины по второму счетчику.
6. Проверяем будет ли новый путь меньше предыдущего уже найденного через другую вершину, и записываем новое значение если оно подходит.

Асимптотическая сложность равно $O(n^3)$, что равно n вызову алгоритмов Дейкстры.

Алгоритм можно использовать для определения существования путей между двумя точками на графе.

Выполнение задачи.

1. Для начала нам необходимо задать условия для генерации графа
   - Минимальное количество ребер зависит от количества вершин и минимального количества связей у одной вершины; **Int₊(ver / 2) * min_bind**
   - Максимальное количество ребер зависит от количества ребер; **ver * (ver - 1) / 2**

```
int ver[] = { 10, 20, 50, 100 };
int min_bind[] = { 3, 4, 10, 20 };
int max_reb, min_reb;
bool oriented;
```

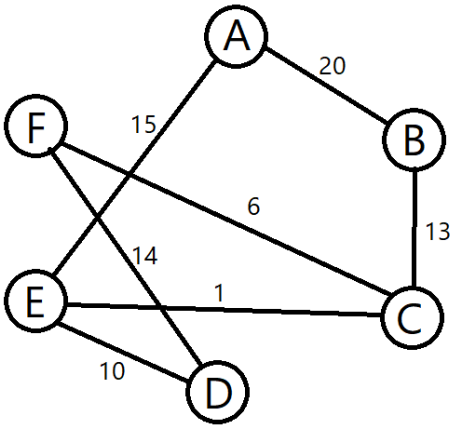
```
gr.ver = ver[i]; // Вершины
cout << "Вершин: " << gr.ver << endl;
min_reb = ceil( x: (double)gr.ver / 2) * min_bind[i];
max_reb = gr.ver * (gr.ver - 1) / 2;
gr.reb = min_reb + rand() % (max_reb - min_reb + 1); // Ребра
oriented = rand() % 2; // Ориентированность
```

2. Далее на основе заданных условий мы генерируем граф (аналогично лабораторной №4)
   - А именно мы генерируем матрицу смежности
   - При ориентированном графе мы расставляем направления ребер
3. После чего, мы расставляем веса на ребра, что делает из обычного графа взвешенный
   - В случае, если связи между вершинами нет, то мы не можем поставить туда 0, так как это значение предполагает путь из вершину в саму себя. Поэтому в таких случаях мы ставит максимальное значение веса плюс один (INF).

```
void weights(Grapth gr, bool oriented) {
    if (oriented) {
        for (int i = 0; i < gr.ver; i++) {
            for (int j = 0; j < gr.ver; j++) {
                if (gr.matrix_smezh[i][j] == 1) {
                    gr.matrix_smezh[i][j] = genWeight();
                }
                else if (i != j) {
                    gr.matrix_smezh[i][j] = INF;
                }
            }
        }
    }
    else {
        for (int i = 0; i < gr.ver; i++) {
            for (int j = i + 1; j < gr.ver; j++) {
                if (gr.matrix_smezh[i][j] == 1) {
                    gr.matrix_smezh[i][j] = gr.matrix_smezh[j][i] = genWeight();
                }
                else {
                    gr.matrix_smezh[i][j] = gr.matrix_smezh[j][i] = INF;
                }
            }
        }
    }
}
```

- Таким образом, мы получили матрицу смежности для неориентированного графа:

```
Вершин: 6
Ребер: 7
Минимальное количество связей: 2
Ориентированный: 0
Матрица смежности
     |   A    B    C    D    E    F
  A  |   0   20  INF  INF   15  INF
  B  |  20    0   13  INF  INF  INF
  C  | INF   13    0  INF    1    6
  D  | INF  INF  INF    0   10   14
  E  |  15  INF    1   10    0  INF
  F  | INF  INF    6   14  INF    0
```



4. Далее с помощью поиска в глубину реализованного в прошлой лабораторной работе, мы проверяем граф на связность. При отсутствии связности мы выбираем по две вершины от каждого куска графа и соединяем их, сохраняя заданное количество ребер.

```
void connected(int* nodes, Grapth gr) {
    int not_bind_1 = 0, with_bind_1 = 0, not_bind_2 = 0, with_bind_2 = 0;

    for (int i = 0; i < gr.ver; i++) {
        if (nodes[i] == 0) not_bind_1 = i;
        else with_bind_1 = i;
    }

    for (int i = 0; i < gr.ver; i++) {
        if (gr.matrix_smezh[not_bind_1][i] == 1) not_bind_2 = i;
        if (gr.matrix_smezh[with_bind_1][i] == 1) with_bind_2 = i;
    }

    if (gr.oriented) {
        gr.matrix_smezh[with_bind_1][with_bind_2] = 0;
        gr.matrix_smezh[with_bind_1][not_bind_1] = genWeight();
        gr.matrix_smezh[not_bind_1][not_bind_2] = 0;
        gr.matrix_smezh[not_bind_2][with_bind_2] = genWeight();
    }
    else {
        gr.matrix_smezh[with_bind_1][with_bind_2] = gr.matrix_smezh[with_bind_2][with_bind_1] = 0;
        gr.matrix_smezh[with_bind_1][not_bind_1] = gr.matrix_smezh[not_bind_1][with_bind_1] = genWeight();
        gr.matrix_smezh[not_bind_1][not_bind_2] = gr.matrix_smezh[not_bind_2][not_bind_1] = 0;
        gr.matrix_smezh[not_bind_2][with_bind_2] = gr.matrix_smezh[with_bind_2][not_bind_2] = genWeight();
    }
}
```

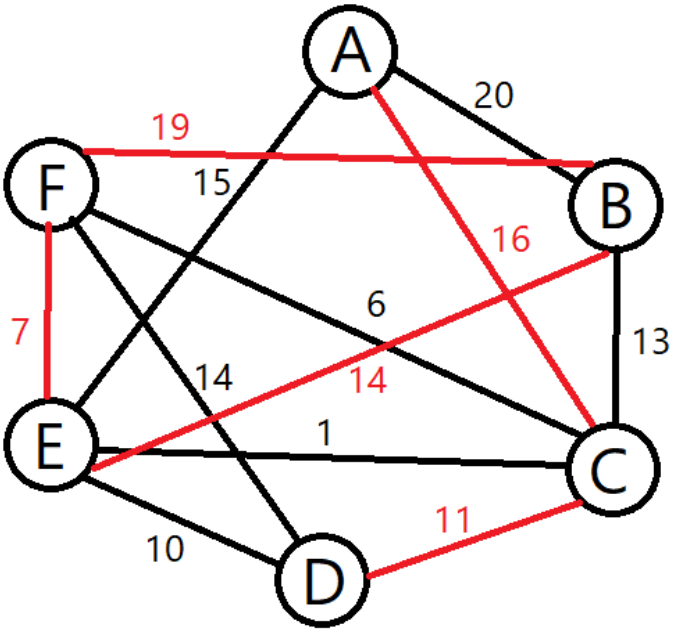
5. После выполнения условия связности графа, программа переходит к алгоритму Флойда-Уоршелла.
    - Берутся вершин по порядку
    - Находим значение пути через узловые вершины
    - Находим новое значение ребра равное минимальному между старым ребром и суммой ребер

```
void Floyd_Warshall(Grapth gr) {
    for (int k = 0; k < gr.ver; k++) {
        for (int i = 0; i < gr.ver; i++) {
            for (int j = 0; j < gr.ver; j++) {
                gr.matrix_smezh[i][j] =
                    min(_Left: gr.matrix_smezh[i][j], _Right: gr.matrix_smezh[i][k] + gr.matrix_smezh[k][j]);
            }
        }
    }
}
```

6. После выполнения алгоритма выводим новую матрицу смежности (новый граф)

Матрица смежности

	A	B	C	D	E	F
A	0	20	16	INF	15	INF
B	20	0	13	INF	14	19
C	16	13	0	11	1	6
D	INF	INF	11	0	10	14
E	15	14	1	10	0	7
F	INF	19	6	14	7	0



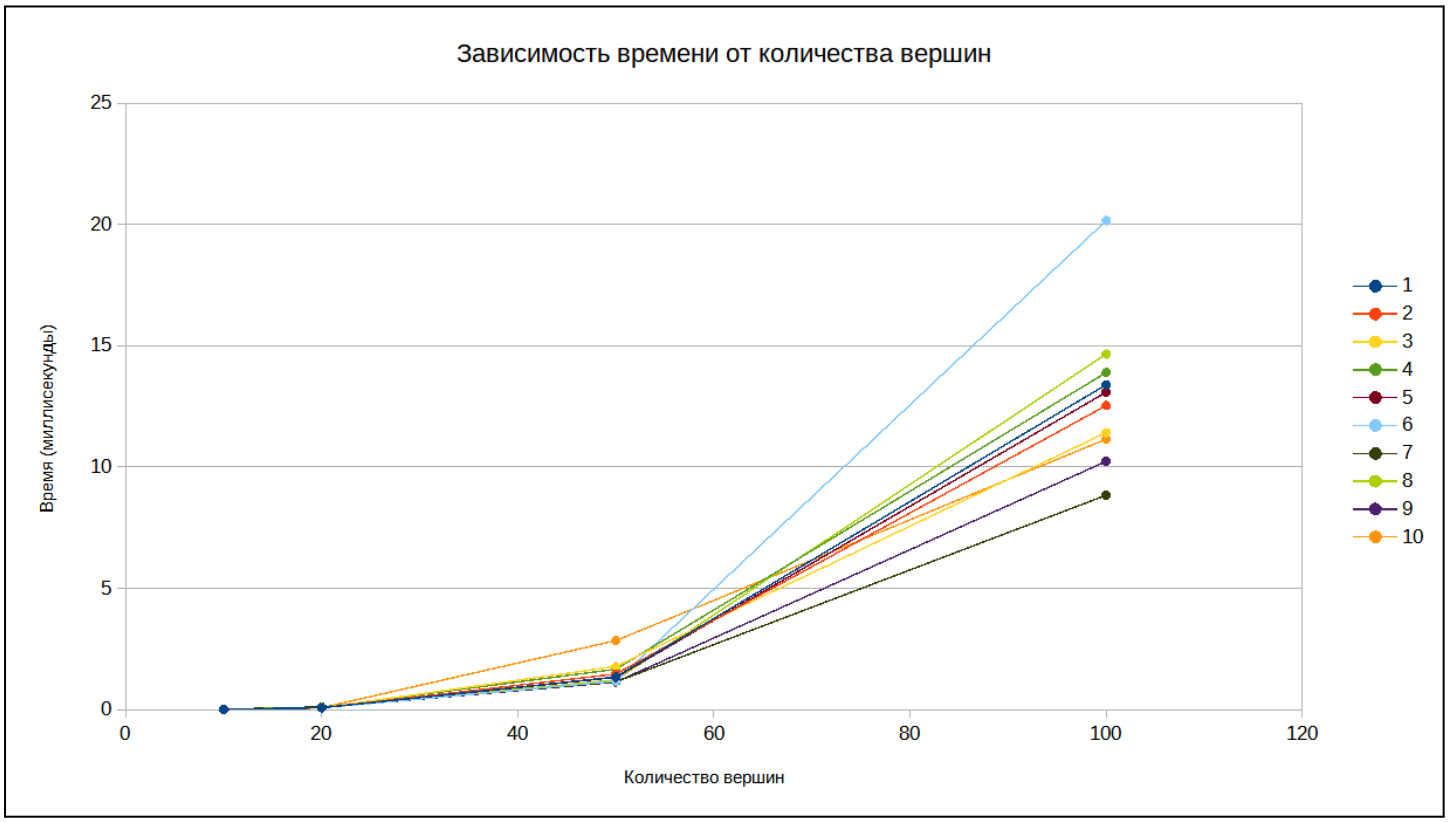
Тестирование алгоритма

Было проведено по 10 тестов на графы с различным количеством вершин (10, 20, 50, 100) и различным минимальным количеством связей у одной вершины (3, 4, 10, 20).

Для каждого теста было замерено время работы алгоритма Флойда-Уоршелла в миллисекундах. Результаты были записаны в файл для удобства использования.

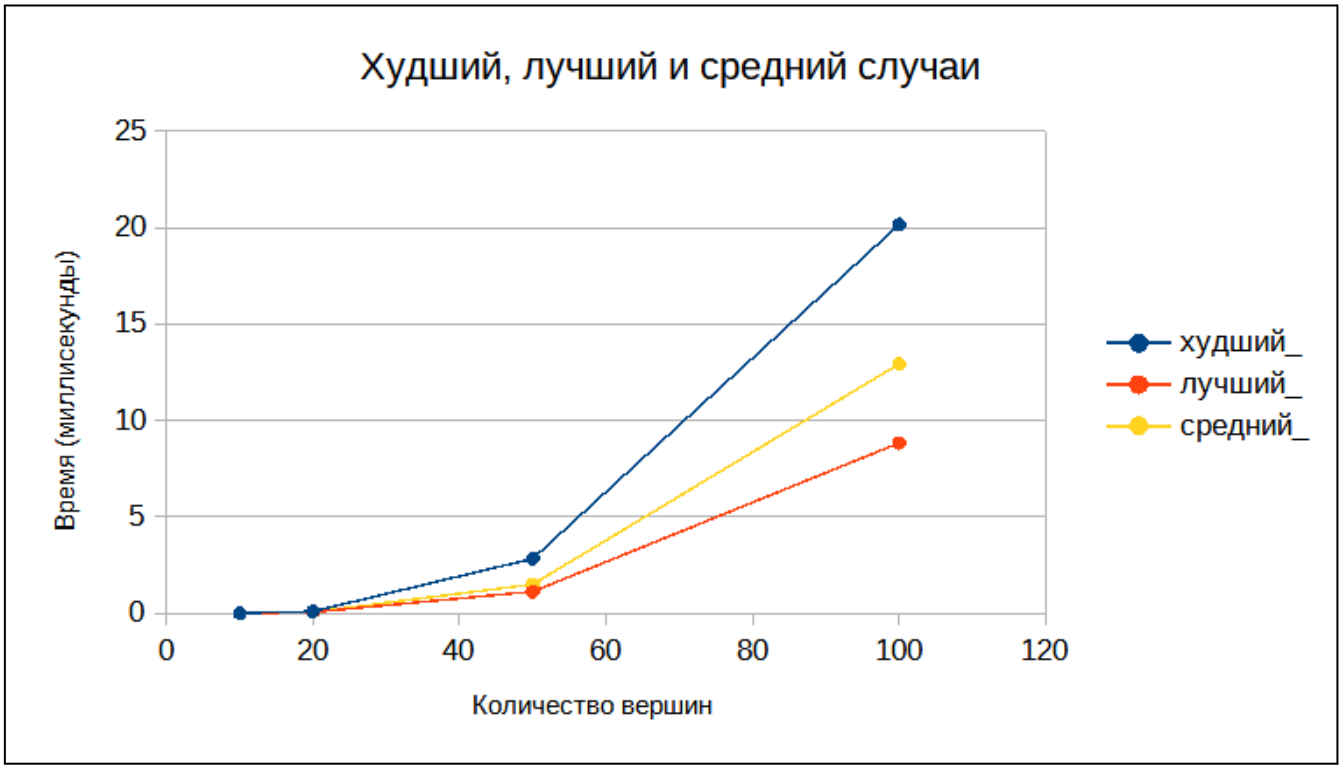
Таким образом, мы получили следующие результаты:

	Время (миллисекунды)			
	10	20	50	100
1	0,0095	0,0686	1,3421	13,3881
2	0,0096	0,0683	1,4634	12,5283
3	0,0096	0,0701	1,7743	11,4138
4	0,0099	0,0825	1,6616	13,8895
5	0,0083	0,0709	1,326	13,0791
6	0,009	0,0694	1,1674	20,1516
7	0,0096	0,1067	1,1364	8,8312
8	0,0089	0,071	1,217	14,6491
9	0,0093	0,0785	1,1297	10,2279
10	0,0096	0,0712	2,8399	11,1414
Асимптота	0,05	0,4	6,25	50
Худший	0,0099	0,1067	2,8399	20,1516
Лучший	0,0083	0,0683	1,1297	8,8312
Средний	0,00933	0,07572	1,50578	12,93
C = 0,00005				



Таким образом, мы получили зависимость времени работы программы от количества вершин у графа.

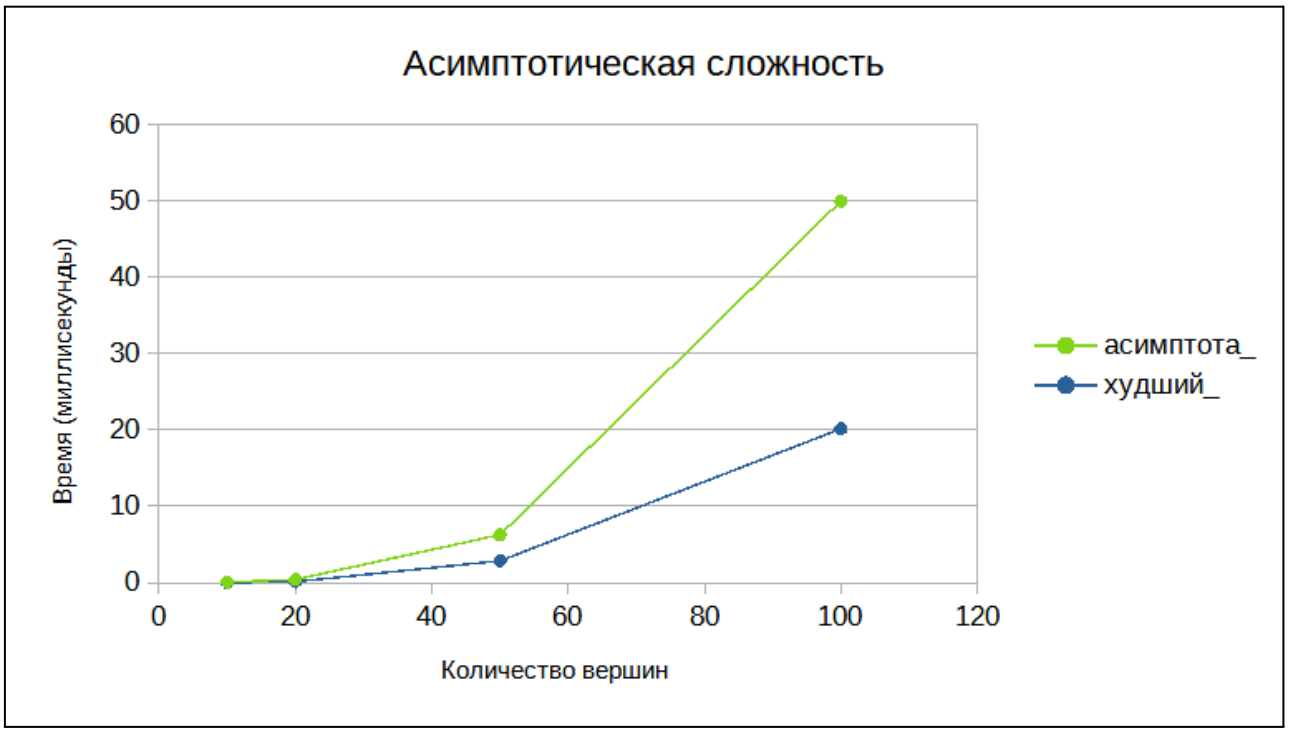
Также был построен график лучшего, среднего и худшего случаев в зависимости от количества вершин графа:



На основе графика худшего случая можно построить график асимптотической сложности:

c * g(N^3)

0,00005*10^3



Заключение.

В заключении, можно заметить, что скорость выполнения алгоритма Флойда-Уоршелла зависит от количества вершин у генерируемого графа. А именно асимптотическая сложность данного алгоритма равна c * g(N^3). Данный алгоритм позволяет нам определить кратчайшие пути из одной вершины в другую. Таким образом, в результате мы имеем оптимальный путь из выбранной вершины до другой вершины, заранее просчитав кратчайший путь от одной к другой и записав результат в матрицу смежности.