Министерство науки и высшего образования Российской Федерации
Российский химико-технологический университет имени Д. И. Менделеева

Факультет информационных технологий и управления
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«Взвешенные графы»

Выполнил студент группы КС-30 Колесников Артем Максимович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/AMKolesnikov_30_ALG

Приняли: аспирант кафедры ИКТ Пысин Максим Дмитриевич
аспирант кафедры ИКТ Краснов Дмитрий Олегович

Дата сдачи: 18.04.2022

Москва
2022

Содержание

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: 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 структуры

Граф – совокупность точек, соединенных линиями. Точки называются вершинами, или узлами, а линии – ребрами, или дугами.

Взвешенным называют 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.

Степень входа вершины – количество входящих в нее ребер, степень выхода – количество исходящих ребер.

Граф, содержащий ребра между всеми парами вершин, является полным.

В ориентированном графе ребра являются направленными, т.е. существует только одно доступное направление между двумя связными вершинами.

В неориентированном графе по каждому из ребер можно осуществлять переход в обоих направлениях.

Граф может быть представлен (сохранен) несколькими способами:

1) Матрица смежности, это двумерная таблица, для которой столбцы и строки соответствуют вершинам, а значения в таблицы соответствуют ребрам, для невзвешенного графа они могут быть просто 1 если связь есть и идет в нужном направлении и 0 если ее нет, а для взвешенного графа будут стоять конкретные значения.
2) Матрица инцидентности, это матрица, в которой строки соответствуют вершинам, а столбцы соответствуют связям, и ячейки ставиться 1 если связь выходит из вершины,

-1 если входит и 0 во всех остальных случаях.

3) Список смежности, это список списков, содержащий все вершины, а внутренние списки для каждой вершины содержат все смежные ей.
4) Список ребер, это список строк, в которых хранятся все ребра вершины, а внутренние значение содержит две вершины к которым присоединено это ребро.

Выполнение задачи

Я реализовал взвешенный граф на языке C++. Моя программа состоит из функций; «GenerateGraphs», «algDijkstra», «main».

Функция «GenerateGraphs» ~ генерирует случайный взвешенный граф с указанными параметрами вершин, диапазона весов, минимального количества связей, ориентированный / неориентированный в виде матрицы смежности.

```
vector<vector<int>> GenerateGraphs(int vertex, int min_num_edge_one_vortex, int min_weight,
int max_weight, bool is_directed) {

	cout << "Число вершин: " << vertex << endl;

	vector<vector<int>> graph(vertex, vector<int>(vertex));

	for (int i = 0; i < vertex; i++)
	{
		int count_existing_edge = 0; //количество существующих ребер
		vector<int> not_adjacency_list; //вектор несмежных вершин

		//проверяем число связей вершины и добавляем не связанные вершины в вектор
		for (int j = 0; j < vertex; j++)
		{
			if (j != i) {

				if (graph[j][i] > 0)
					count_existing_edge++;
				else
					not_adjacency_list.push_back(j);
			}
		}

		int num_edges = numGenerator(min_num_edge_one_vortex, vertex - 1);

		if (count_existing_edge < num_edges)
			for (int k = 0; k < num_edges - count_existing_edge; k++)
			{
				int random_index = rand() % not_adjacency_list.size();
				int v = not_adjacency_list[random_index];
				not_adjacency_list.erase(not_adjacency_list.begin() +
random_index);
				int weight = numGenerator(min_weight, max_weight);

				if (is_directed)
					graph[i][v] = weight;
				else
					graph[i][v] = graph[v][i] = weight;

			}
	}

	cout << "\nМатрица смежности" << endl;

	for (int i = 0; i < vertex; i++) {
		for (int j = 0; j < vertex; j++) {
			cout << setw(5) << graph[i][j] << "\t";
		}

		cout << endl;
	}
```

```
	cout << endl;

	return graph;
	 }
```

Функция «algDijkstra» ~ ищет кратчайшие пути между всеми вершинами взвешенного графа и их длину, и выводит все это в консоль:

```
void algDijkstra(vector<vector<int>> graph, int start_ind, int finish_ind) {

	int num_vertex = graph.size();
	int max_value = 2e5;
	vector<bool> visited(num_vertex, false); // посещенные вершины
	vector<int> min_distance(num_vertex, max_value); // минимальное расстояние
(изначально равно "максимуму")
	int temp, minindex, min;

	min_distance[start_ind] = 0;
	// Шаг алгоритма
	do {
		minindex = max_value;
		min = max_value;
		for (int i = 0; i < num_vertex; i++)
		{ // Если вершину ещё не обошли и вес меньше min
			if (!visited[i] && (min_distance[i] < min))
			{ // Переприсваиваем значения
				min = min_distance[i];
				minindex = i;
			}
		}
		// Добавляем найденный минимальный вес
		// к текущему весу вершины
		// и сравниваем с текущим минимальным весом вершины
		if (minindex != max_value)
		{
			for (int i = 0; i < num_vertex; i++)
			{
				if (graph[minindex][i] > 0)
				{
					temp = min + graph[minindex][i];
					if (temp < min_distance[i])
					{
						min_distance[i] = temp;
					}
				}
			}
			visited[minindex] = true;
		}
	} while (minindex < max_value);

	// Вывод кратчайших расстояний до вершин
	cout << "Кратчайшие расстояния до вершин:" << endl;
	for (int i = 0; i < num_vertex; i++)
		cout << setw(5) <<  min_distance[i] << "\t";

	// Восстановление пути
	vector<int> visited_vertex(num_vertex);
	int end = finish_ind;
	visited_vertex[0] = finish_ind;
	int k = 1; // индекс предыдущей вершины
	int weight = min_distance[finish_ind]; // вес конечной вершины

	while (end != start_ind) // пока не дошли до начальной вершины
```

```
	{
		for (int i = 0; i < num_vertex; i++) // просматриваем все вершины
			if (graph[i][end] != 0)   // если связь есть
			{
				int temp = weight - graph[i][end]; // определяем вес пути из 
предыдущей вершины
				if (temp == min_distance[i]) // если вес совпал с рассчитанным
				{                 // значит из этой вершины и был переход
					weight = temp; // сохраняем новый вес
					end = i;       // сохраняем предыдущую вершину
					visited_vertex[k] = i; // и записываем ее в массив
					k++;
				}
			}
	}
	// Вывод пути (начальная вершина оказалась в конце массива из k элементов)
	cout << "\nВывод кратчайшего пути:" << endl;
	for (int i = k - 1; i >= 0; i--)
		cout << setw(5) << visited_vertex[i] << "\t";

	cout << endl;
	 }
```

Функция «main» ~ состоит из цикла в котором происходит расчет минимального, максимального, среднего времени, затраченного на поиски кратчайших путей алгоритмом Дейкстры в сгенерированном взвешенном графе, состоящем из `NUM_VERTEX = { 10, 20, 50, 100, 250, 500 }` вершин, каждое из которых рассчитывается 10 раз. На выходе мы получаем файл «Time.txt» с результатами работы нашего алгоритма.

```
int main()
{
	srand(time(0));
	setlocale(LC_ALL, "Rus");

	const int NUM_TEST = 10;
	const int MIN_WEIGHT = 1;
	const int MAX_WEIGHT = 20;

	vector<int> NUM_VERTEX = { 10, 20, 50, 100, 250, 500 };
	vector<int> MIN_NUM_ADJ = { 3, 4, 10, 20, 50, 100 };

	double time[NUM_TEST] = { 0 };

	ofstream tout("Time.txt");

	if (tout.is_open()) {
		for (int i = 0; i < NUM_VERTEX.size(); i++)
		{
			double sum_time = 0;

			tout << "Число вершин: " << NUM_VERTEX[i] << endl;
			tout << "Время поиска кратчайших путей (мс):" << endl;

			for (int j = 0; j < NUM_TEST; j++)
			{
				vector<vector<int>> v = GenerateGraphs(NUM_VERTEX[i], 
MIN_NUM_ADJ[i], MIN_WEIGHT, MAX_WEIGHT, false);

				chrono::high_resolution_clock::time_point start = 
chrono::high_resolution_clock::now();
				algDijkstra(v, 0, v.size() - 1);
```

```
                        chrono::high_resolution_clock::time_point end =
chrono::high_resolution_clock::now();
                        chrono::duration<double, milli> milli_diff = end - start;

                        time[j] = milli_diff.count();
                        tout << time[j] << " ; ";

                        sum_time += time[j];

                        cout << "Время поиска кратчайших путей (мс): " << time[j] <<
endl;
                        cout << endl << endl;

                }

                tout << endl;

                double min_t = time[0];
                double max_t = time[0];

                for (int j = 0; j < NUM_TEST; j++) {
                        if (max_t < time[j]) max_t = time[j];
                        if (min_t > time[j]) min_t = time[j];
                }

                tout << endl << "Минимальное: " << min_t << endl << "Максимальное: "
<< max_t << endl << "Среднее: " << sum_time / NUM_TEST << endl << endl;
                tout << endl << endl;
        }

        tout.close();
    }

}
```

Один из результатов работы программы:

```
Число вершин: 10

Матрица смежности
    0      18       4       9       1      20      13       0       1       7
   18       0       8      17      18       4       4      14       9      14
    4       8       0      12       1       0       0      16       8      13
    9      17      12       0      12      10       2       2       1      20
    1      18       1      12       0       0       7       8       2       0
   20       4       0      10       0       0       0       0      14       8
   13       4       0       2       7       0       0       0      10       1
    0      14      16       2       8       0       0       0       0       0
    1       9       8       1       2      14      10       0       0       0
    7      14      13      20       0       8       1       0       0       0

Кратчайшие расстояния до вершин:
    0       8       2       2       1      12       4       4       1       5
Вывод кратчайшего пути:
    0       8       3       6       9
Время поиска кратчайших путей (мс): 19.0984
```

Рисунок 1 - Результат из консоли

```
Число вершин: 10
Время поиска кратчайших путей (мс):
7.5502 ; 11.6725 ; 12.746 ; 10.5354 ; 11.8346 ; 12.8043 ; 24.947 ; 14.3768 ; 20.2699 ; 15.5736 ;

Минимальное: 7.5502
Максимальное: 24.947
Среднее: 14.231



Число вершин: 20
Время поиска кратчайших путей (мс):
21.3913 ; 20.6142 ; 22.8729 ; 23.8847 ; 23.8413 ; 20.2064 ; 12.3883 ; 12.7488 ; 13.3362 ; 26.7947 ;

Минимальное: 12.3883
Максимальное: 26.7947
Среднее: 19.8079



Число вершин: 50
Время поиска кратчайших путей (мс):
36.8436 ; 38.3462 ; 55.2598 ; 57.542 ; 57.584 ; 34.2575 ; 36.6983 ; 36.8822 ; 58.0068 ; 63.3924 ;

Минимальное: 34.2575
Максимальное: 63.3924
Среднее: 47.4813
```

Рисунок 2 - Результат из файла "Time.txt"

Время (мс)			
Вершин	**мин**	**сред**	**макс**
10	7,550	14,231	24,947
20	12,388	19,808	26,795
50	34,258	47,481	63,392
100	61,571	105,696	119,865
250	197,821	274,657	374,879
500	547,551	651,702	822,323

Таблица 1 - Максимальное, минимальное, среднее время (в миллисекундах) поиска кратчайших путей алгоритмом Дейкстры во взвешенном графе, состоящем из N вершин.

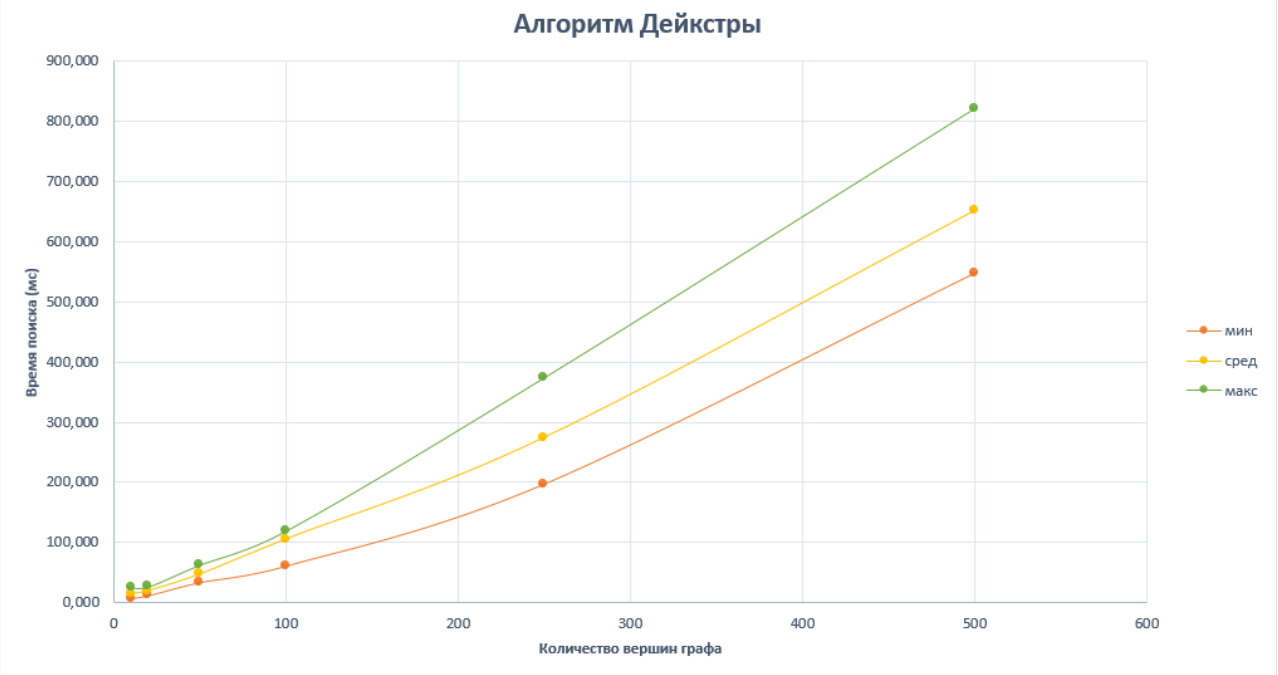

Рисунок 3 - График минимального, среднего, максимального времени поиска кратчайшего пути алгоритмом Дейкстры относительно количества вершин графа.

Заключение

В этой лабораторной работе я познакомился с такой структурой, как взвешенный граф. Мне удалось реализовать его на языке C++. Я реализовал алгоритм Дейкстры и могу сделать вывод о том, что он прекрасно справляется с нахождением кратчайших путей между всеми вершинами. Сложность алгоритма Дейкстры зависит от способа нахождения вершины, а также способа хранения множества не посещённых вершин и способа обновления длин. Анализируя график, можно увидеть, что временная сложность моего алгоритма - $O(n * \log(n))$. Главный недостаток алгоритма заключается в том, что он не работает с графами, которые содержат ребра с отрицательным весом.