«Санкт-Петербургский государственный университет»
Факультет «Прикладная математика-процессы управления»

Отчет

по дисциплине: **«Алгоритмы и анализ сложности»**

на тему: **Эмпирический анализ алгоритма**

“Алгоритм Борувки нахождения минимального остовного дерева в графе”

Выполнил: Малиновский Илья Владимирович,

Студент группы 19.Б-13

Санкт-Петербург, 2021

Содержание

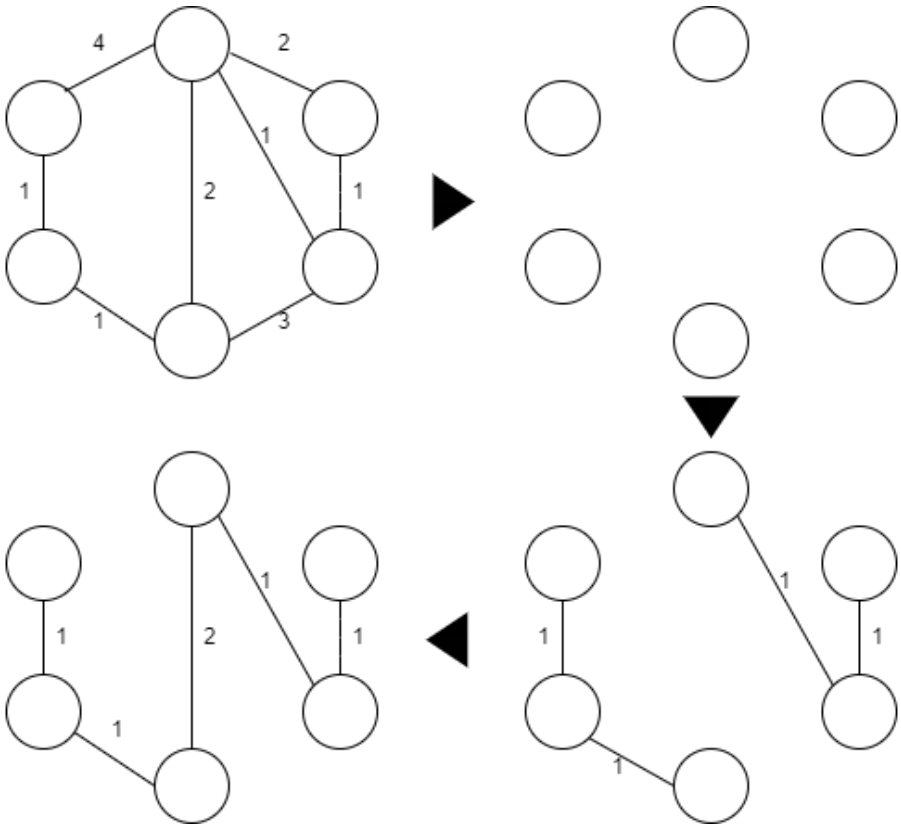
Краткое описание алгоритма

Алгоритм Борувки – это жадный алгоритм, разработанный и опубликованный Отакаром Борувкой, чешским математиком, наиболее знаменитым своей теорией графов.

В этом алгоритме стоит отметить то, что это самый старый из известных алгоритмов для поиска минимального остовного дерева. Борувка придумал его в 1926 году, еще до того, как появились компьютеры в том виде, в каком мы их знаем сегодня. Он был опубликован как метод построения эффективной электросети.

Идея алгоритма довольно проста и интуитивно понятна. Разобьём его на несколько шагов:

1. Инициализируем каждую из вершин как отдельную компоненту.
2. Создадим минимальное остовное дерево **S** как пустое множество (в будущем содержащее решение).
3. Если компонент более одной
    - Найдем для каждой минимальное по весу ребро, которое соединяет ее с другой компонентной.
    - Если это ребро не в **S**, добавим его.
4. Если осталась всего одна компонента – минимальное остовное дерево построено. Иначе – повтор 3-го пункта.



На рассмотренном выше примере, первым шагом граф разбился на пять компонент (по числу вершин). Далее, для каждой из компонент было найдено самое дешевое ребро, тем самым образовалось две компоненты. Последним шагом компоненты объединились в одну, добавлением ребра с весом 2 (как самым дешевым для обоих компонент).

Математический анализ алгоритма.

Алгоритм при каждом шаге уменьшает количество компонент связности в два раза. Изначально количество компонент равно числу вершин, следовательно, алгоритм совершит $\log(V)$ шагов.

Поскольку на каждом шаге проверяется E ребер, то внутренний цикл имеет асимптотику $O(E)$.

Окончательная же временная сложность составляет $O(E \log(V))$.

Количество ребер влияет на работу алгоритма сильнее количества вершин. Соответственно, лучшим случаем для алгоритма будет граф с минимальным числом ребер, которые образуют компоненту. Худшим же – полный граф.

Описание характеристик входных данных.

На вход подается сгенерированный неориентированный связный граф. С числом вершин или ребер, равным $N \in (500 \ldots 20000)$ с шагом в 500.

Для достоверности берется среднее время работы 30-и запусков алгоритма.

Трудоемкость оценивается затраченным временем на выполнение. Время считается с помощью средств языка Python.

Описание характеристик генератора для входных данных

Генерация входных данных происходит по средствам функции $generate\_graph(n)$, где n – число, характер которого описан выше.

С помощью встроенной в python функции $randomint()$ между двумя случайными вершинами создается ребро со случайным весом (в диапазоне от 1 до 100).

Чтобы убедится в существовании одной компоненты связности для графа создаются ребра, последовательно соединяющие вершины от 1 до n.

Реализация алгоритма

Реализация алгоритма представлена на языке Python. Сам код находится на GitHub по адресу: (https://github.com/imalinowski/BoruvkaAlgorithm)

Вычислительный эксперимент

Поскольку при анализе алгоритма выяснилось, что сложность алгоритма зависит от количества вершин и ребер в графе, будем производить эксперименты отдельно для каждой величины.

Ради чистоты эксперимента для каждой десятки запусков вычислялось среднее время. На графиках это указано цветом.

Начнем с фиксированного количества вершин. Так как $N \in 500 \dots 20000$, то число вершин, между которыми может находится столько ребер равно

$$\mathrm{n} \approx \sqrt{\mathrm{N} * 2}$$

$$(N = \frac{n(n-1)}{2})$$

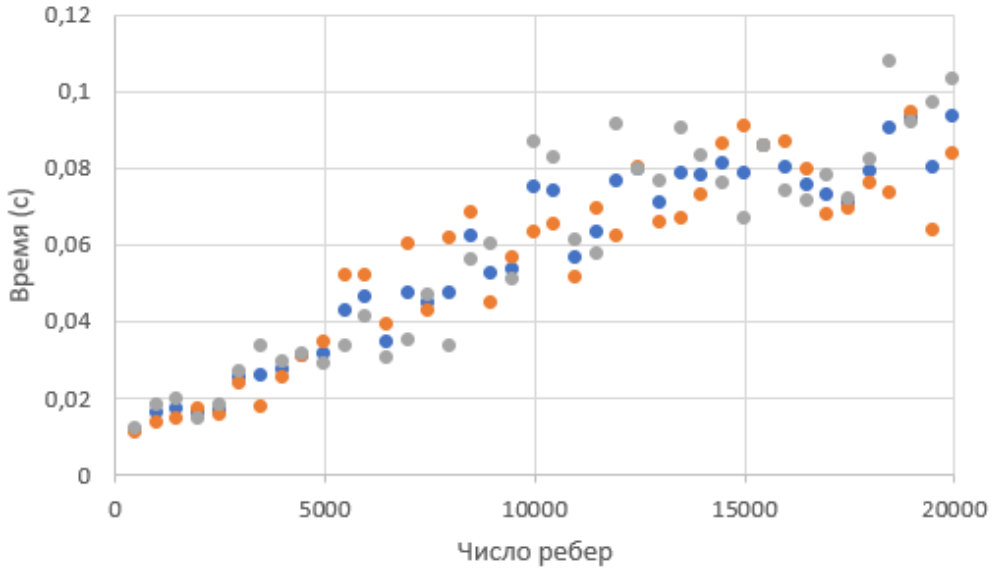
Выберем максимальное $N = 20000$ следовательно, для него $n \approx 150$.

Некоторые из результатов работы алгоритма представлены в таблице.

Таблица 1. Результаты вычислительного эксперимента, n – число ребер, t – среднее время в секундах

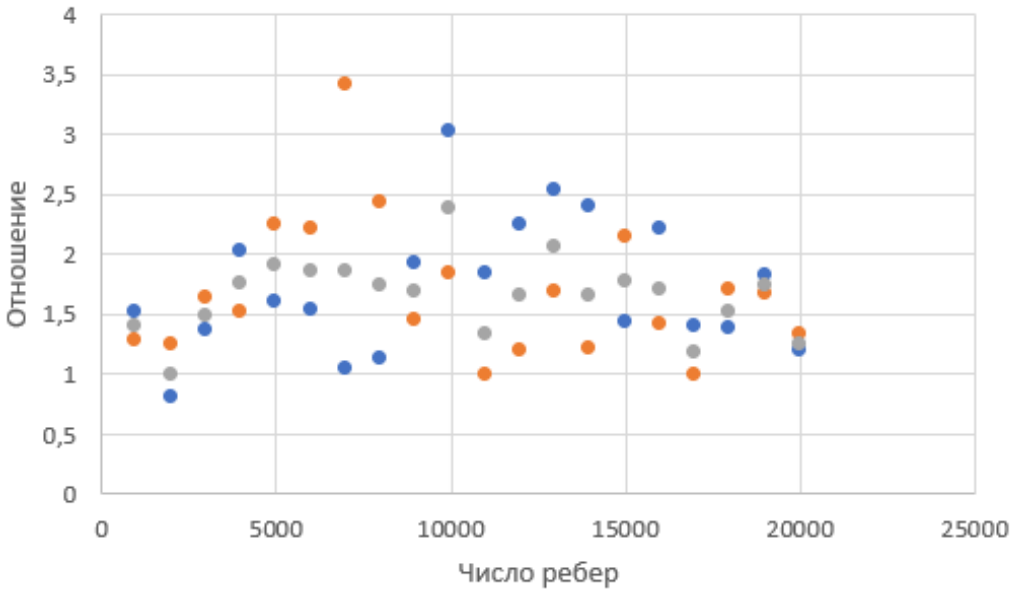
N	t
500	0,011211952
1000	0,01571854
7500	0,044310609
10000	0,074777404
12500	0,079581102
15000	0,078409394
20000	0,093218247

Рис.1 График зависимости времени работы t от числа ребер n



По графику видно, что время выполнения алгоритма ограничено $O(E)$ (при фиксированном количестве вершин).

Рис.1.2 Отношение M(2E)/M(E)



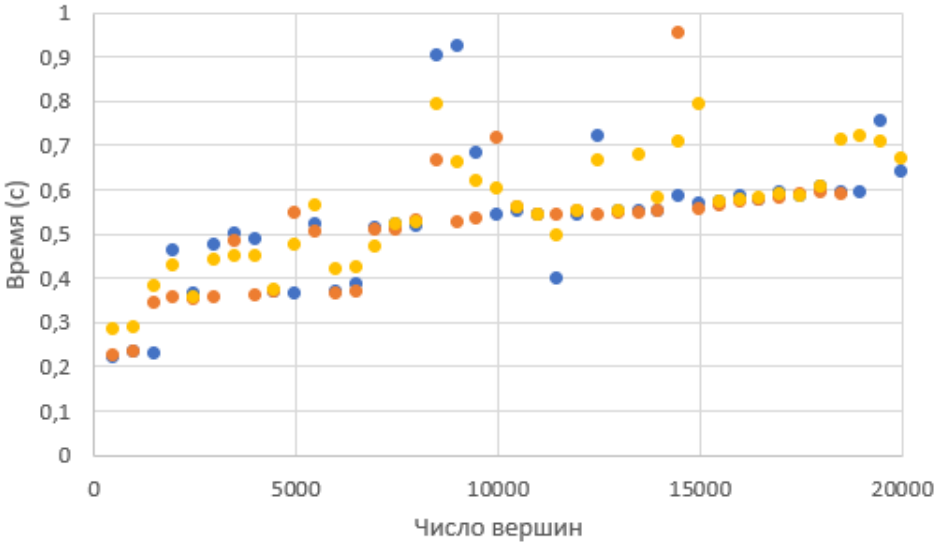
Из (рис 1.2) видно, что отношение $M(2E)/M(E)$ стремится к величине от 1.25 до 2.

Повторим эксперимент, на этот раз фиксируем число ребер. Так как для существования минимального остовного дерева необходима одна компонента связности, нужно чтобы фиксированное число ребер могло обеспечить это условие. Следовательно, оно должно быть достаточно большим. Например, 125000 – достаточное число ребер для 20000 вершин.

Таблица 2. Результаты вычислительного эксперимента, n – число вершин, t – среднее время в секундах.

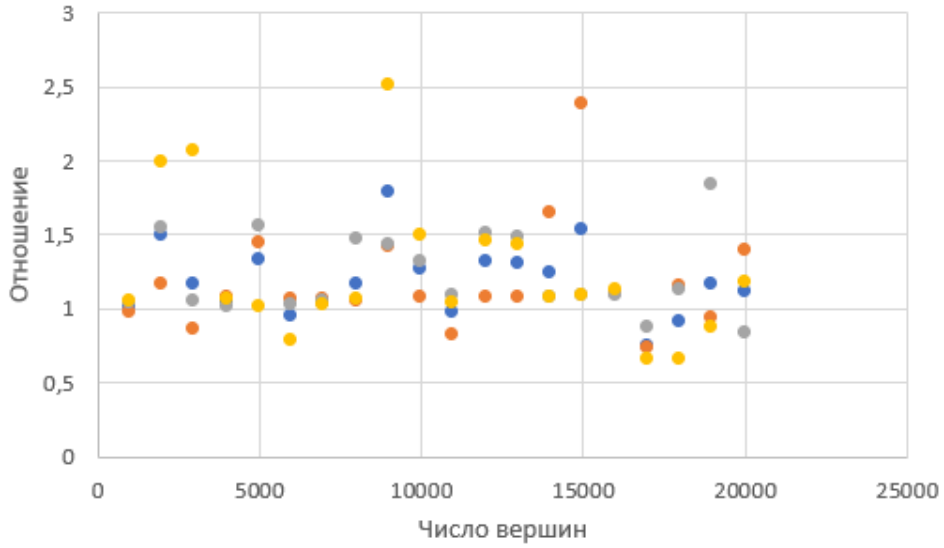
N	t
500	0,28398935
1000	0,286043167
7500	0,518503904
10000	0,601590157
12500	0,662014325
15000	0,790776253
20000	0,667101463

Рис.2 График зависимости времени работы t от числа вершин n



Как видно из графика, время выполнения ограничено $O(\log(N))$.

Рис.2.2 Отношение M(2E)/M(E)



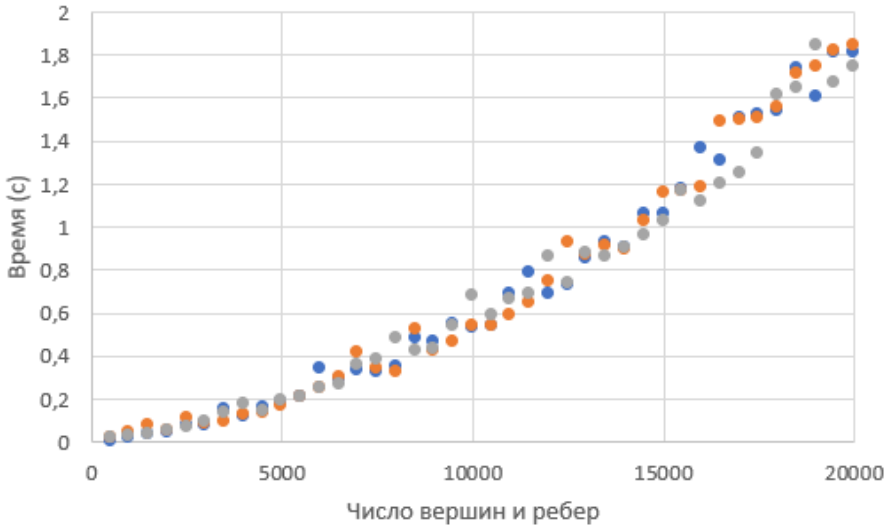
При вычислении отношения $\frac{M(2V)}{M(V)}$ получены следующие результаты (рис 2.2). Величина стремится к значению между 1 и 1.25 .

Наконец, проведем эксперимент без фиксирования ребер или вершин. Пусть их число равно N.

Таблица 3. Результаты вычислительного эксперимента, n - число вершин и ребер, t – среднее время в секундах

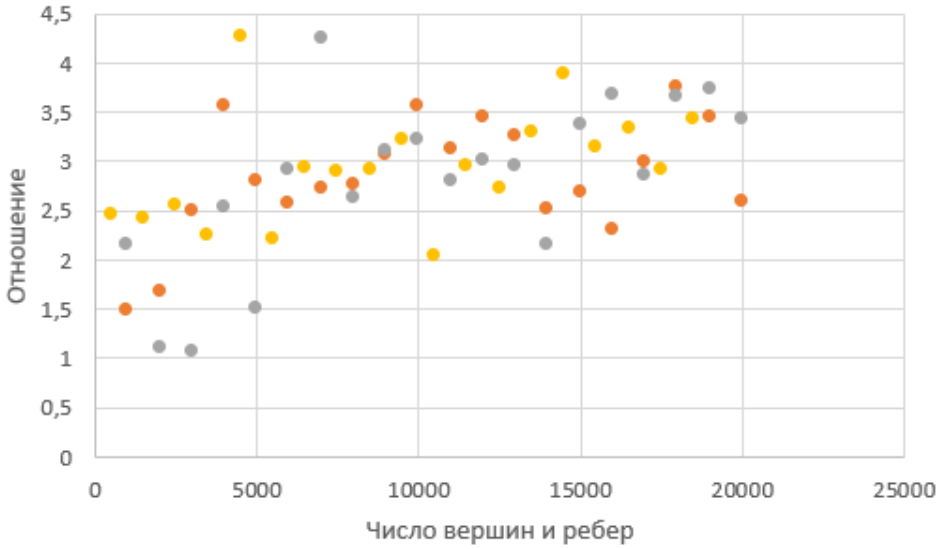
n	t
500	0,020239353
1000	0,043648481
7500	0,482725308
10000	0,664385619
12500	0,85060253
15000	1,040450616
20000	1,428771238

Рис.3 График зависимости времени работы t от числа вершин и ребер n



На графике по оси Y указано время в секундах, а по оси X – n. Для чистоты эксперимента для каждой десятки запусков вычислялось среднее время. На графиках это указано цветом. По графику видно, что время выполнения алгоритма ограничено $O(nlog(n))$.

Рис.2.2 Отношение M(2N)/M(N)



По графику видно, что величина $M(2N)/M(N)$ стремится к значению между 3 и 4.

Теоретически отношение M(2N)/M(N) должно стремиться к $\frac{2log(2n)}{log(n)}$.

Однако на практике мы видим погрешность ввиду того, что N – мало, а на больших данных алгоритм работает медленно.

Характеристики вычислительной среды

Для выполнения работы использовался ноутбук

- процессор Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- оперативная память 8,00 ГБ

Вычисления производились в терминале Windows 10, версия Python 3.10

Источники

1. Алгоритм Борувки — Алговики (algowiki-project.org)
2. Алгоритм Борувки — Викиконспекты (ifmo.ru)
3. Алгоритмы и введение в разработку А. Левитин.
4. Recitation10.pdf (buffalo.edu)