DOI 10.37882/2223–2966.2023.04.26

О НАХОЖДЕНИИ МИНИМАЛЬНОГО ОСТОВНОГО ДЕРЕВА

Охлупина Ольга Валентиновна
Кандидат физико-математических наук, доцент, Брянский государственный инженерно-технологический университет
helga131081@yandex.ru
Мурашко Дмитрий Сергеевич
Брянский государственный инженерно-технологический университет
murashko100500@gmail.com

ON FINDING THE MINIMAL SPANNING TREE

O. Okhlupina
D. Murashko

Summary. The use of graph theory for the purpose of formalization and data research is widely used in various branches of human activity, such as construction, design, aviation, economics. Finding the minimum spanning tree of an arbitrary undirected weighted graph has been a well-known problem for decades. However, the method of solving this problem still does not lose its relevance in connection with the search for the most effective implementation. The existing algorithms of the solution are being improved for the purpose of universality of use and error tolerance. The paper considers the solution of the problem of finding the minimum spanning tree using the Prima algorithm using html css java script. A convenient tool has been obtained for quickly creating a tree grid and calculating the minimum distance between points by building a path map and visually displaying the result to the user. For convenient operation, an interface has been developed that allows you to manipulate the grid of space.

Keywords: weight, vertex, graph, tree, path, edge, spanning tree, connected graph, undirected graph, space grid.

Аннотация. Использование теории графов с целью формализации и исследования данных находит широкое применение в различных отраслях человеческой деятельности, таких, как строительство, дизайн, авиация, экономика. Поиск минимального остовного дерева произвольного неориентированного взвешенного графа представляет собой широко известную задачу уже не одно десятилетие. Однако, способ решения данной задачи до сих пор не теряет своей актуальности в связи с поиском наиболее эффективной реализации. Существующие алгоритмы решения подвергаются усовершенствованию с целью универсальности использования и устойчивости к ошибкам. В работе рассмотрено решение задачи нахождения минимального остовного дерева с применением алгоритма Прима с использованием html css java script. Получен удобный инструмент для быстрого создания сетки дерева и просчёта минимального расстояния между точками с помощью выстраивания карты путей и визуально удобного отображения результата пользователю. Для удобной работы был разработан интерфейс, позволяющий манипулировать сеткой пространства.

Ключевые слова: вес, вершина, граф, дерево, путь, ребро, остовное дерево, связный граф, неориентированный граф, сетка пространства.

Введение

Модели взвешенных графов находят широкое применение при решении практических задач в различных областях с целью визуализации процессов: от электронных схем до задач планирования. В связи с этим актуальной становится проблема минимизации затрат. При этом веса могут отождествляться как со временем, так и с затратами, либо с иными величинами. Переход к математической модели задачи означает возникновение необходимости поиска наименьшего веса пути, соединяющего точки неориентированного графа.

В первой части работы приводится математическое обоснование поставленной задачи. Вторая часть посвящена реализации алгоритма поиска минимального остовного дерева (МОД) с демонстрацией программного кода на с использованием html css java script.

Задача поиска МОД

Введём следующие обозначения. Пусть $G(V,E)$ — неориентированный взвешенный граф с множеством вершин V и множеством рёбер E. При этом вес ребра определим с помощью функции $\omega: E \to R$, где R — множество действительных чисел.

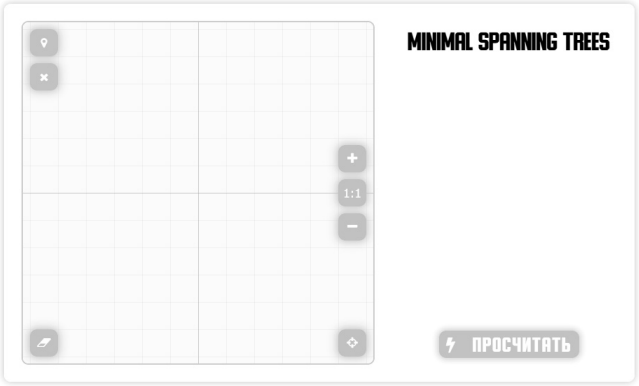

Рис. 1. Общий вид сайта

Маршрутом в графе будем считать чередующуюся последовательность вершин и рёбер графа, в которой произвольные пары соседних элементов инцидентны. Путь — маршрут, в котором рёбра не повторяются. Простой путь — путь, в котором не повторяются вершины. Под циклом будем понимать замкнутый маршрут без повторяющихся рёбер.

Подграфом графа $G(V,E)$ является граф $G_1(V_1,E_1)$, у которого $V_1 \subseteq V$, $E_1 \subseteq E$.

Дерево — связный граф, не содержащий циклов.

Остовным деревом (ОД) связного графа $G(V,E)$ назовём его произвольный подграф $G_1(V_1,E_1)$, содержащий все вершины графа G и являющийся деревом. Каждый связный граф содержит ОД.

Вес ОД складывается из суммы весов рёбер ОД. Минимальное ОД (МОД) взвешенного графа — это ОД, вес которого не больше веса произвольного ОД данного графа.

Таким образом, возникает задача построения ОД с минимальным весом.

Замечание: в случае равенства весов рёбер графа может возникнуть ситуация существования нескольких МОД. Минимизации подвергается не само дерево, а вес. В случае существования нескольких рёбер минимального веса минимальным при применении алгоритма может являться любое из этих рёбер.

Представим теперь себе сформулированную выше задачу таким образом: возьмём за основу вершины исходного графа без рёбер. Нам необходимо соединить их между собой так, чтобы можно было оказаться из одной вершины в другой без возникновения циклов и минимальным суммарным весом задействованных рёбер.

Идея алгоритма Прима состоит в том, что, отталкиваясь от подграфа, содержащего единственную произвольную вершину исходного графа, мы должны достичь МОД. Перебор рёбер, инцидентных выбранной изначально вершине, происходит так, что определяется ребро с минимальным весом, связывающее две различные компоненты связности, одной из которых является имеющийся подграф. Процесс повторяется до нахождения МОД.

Реализация алгоритма

В качестве алгоритма работы, был выбран алгоритм Прима. Весь интерфейс сайта [1] был написан на языках программирования html и css, все алгоритмические части были написаны на JavaScript. Код представлен в [2].

Сайт представляет из себя платформу с передвигаемым полем и элементами управления. Имеется возможность добавлять, удалять точки дерева, очищать, приближать и отдалять поле. Также, нельзя не отметить, кнопку вызова алгоритма просчёта дерева.

В первую очередь, после расстановки точек и нажатия на кнопку, вызывается функция traceButton:

```
function traceButton () {
clearTools ();
clearLines ();
let obj = new MinTree (arrayPoints);
obj.trace ();
obj.calculate ();
}
```

В ней, в первую очередь, очищаются старые данные, со старых расчётов. Далее происходит создание объекта класса MinTree по массиву точке с координатами плоскости, после чего вызывается метод trace, для трассировки линий в памяти. Происходит это, путём

создания массива линий от каждой точки к каждой, перемешивания этого массива и, с помощью функции, отсечения одной из двух пересекающихся линий, после чего, происходит рендеринг этих линий на поле сайта. Следующим на очереди, как раз, идёт метод calculate, в котором и заключён весь алгоритм Прима, готовый работать по заранее подготовленному дереву.

```
calculate () {
function pushLineIds (line) {
selected_peaks.add (line.ids.id_1);
selected_peaks.add (line.ids.id_2);
unselected_peaks.delete (line.ids.id_1);
unselected_peaks.delete (line.ids.id_2);
}

let lines_act;
let lines_result = [];
let unselected_peaks = new Set ();
let selected_peaks = new Set ();

selected_peaks.add (this.points [0].id);
unselected_peaks.delete (this.points [0].id);
this.points.sort (() => Math.random () — 0.5);
this.points.forEach ((res) => unselected_peaks.add (res.id));

while (lines_act = this.getLinesIdWithIdSet (selected_peaks)) {
let short_line = this.getMinimalLenLine (lines_act);
lines_result.push (short_line);
pushLineIds (short_line);
}
for (let line of lines_result) {
let first_point_svg = document.querySelector ('.fieldPoint');
let line_body = document.getElementById (`line-${line.id}`);
line_body.style.strokeWidth = '6';
line_body.style.stroke = '#444';
first_point_svg.insertAdjacentHTML ('beforebegin', line_body.outerHTML);
line_body.remove ();
}
for (let point of document.querySelectorAll ('.fieldPoint')) {
point.classList.add ('calculate-mode-point');
}
}
```

В первую очередь, подготавливаются переменные, для хранения результативных и буферных данных. В первую выбранную вершину добавляется случайная точка плоскости, т.е. точка с нулевым индексом, в невыбранные добавляются все оставшиеся вершины. Следом массив точек случайно перемешивается. После чего, в ход вступает бесконечный цикл с перебором всех возможных линий, через функцию getLinesIdWithIdSet, для данной, конечной, комбинации вершин из переменной.

```
getLinesIdWithIdSet (point_set) {
let arrayLines = [];
for (let point of point_set) {
for (let line of this.lines) {
if (line.has (point) &&!line.inSet (point_set)) arrayLines.push (line);
}
}
if (arrayLines.length) {
return arrayLines;
} else {
return false;
}
}
```

Создаётся пустой массив, для итогового хранения ссылок на объекты линий, которые включают в себя только одну из конечных точек. Если линия соответствует требованиям, то она добавляется в массив и результативный массив отправляется обратно в метод calculate. В случае, если массив окажется пустой, функция вернёт значение false, что приведёт к выходу из цикла в методе calculate.

После получения списка возможных линий, которые могут являться частью минимального пути, с помощью функции getMinimalLenLine, находится минимальная по длине линия.

```
getMinimalLenLine (array_lines) {
array_lines.sort ((a, b) => a.len > b.len? 1: a.len < b.len? –1: 0)
return array_lines [0];
}
```

После чего самая короткая линия добавляется в результирующий массив пути минимального остовного дерева, также в выбранные вершины добавляются точки этой линии и эти же точки удаляются из невыбранных вершин. Таким образом происходит вычисление минимального остовного пути. После чего, остаётся только отрисовать всю информацию из результирующего массива.

Заключение

В результате разработки данного веб приложения, мы имеем удобный инструмент для быстрого создания сетки дерева и просчёта минимального расстояния между точками с помощью выстраивания карты путей

и визуально удобного отображения результата пользователю. Использование алгоритма Прима позволяет создать красивый быстрый и безотказный алгоритм, который будет устойчив к ошибкам начальных данных.

Для удобной работы был разработан интерфейс, позволяющий манипулировать сеткой пространства, на которой, в свою очередь, имеется возможность как размещения точек дерева, так и удаления. Так как порядок соединения рёбер между точками определяет алгоритм случайного соединения, то путём простого нажатия на кнопку расчёта, можно заново сгенерировать сетку уже с другими рёбрами, получив совсем другое, уникальное дерево.

ЛИТЕРАТУРА

1. Разработанный сайт [Электронный ресурс]. — Режим доступа: https://lorexiq.github.io/MinimalSpanningTrees/
2. Рабочий код [Электронный ресурс]. — Режим доступа: https://github.com/LorexIQ/MinimalSpanningTrees
3. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. Спб.: БХВ-Петербург, 2003. 1104 с.
4. Кристофидес Н. Теория графов. Алгоритмический подход. М. ДМК Пресс, 2003. 356 с.
5. Современный учебник JavaScript [Электронный ресурс]. — Режим доступа: https://learn.javascript.ru/ (дата обращения: 20.11.2022)
6. O.V. Ohlupina, D.S. Murashko, Applying a probabilistic algorithm to spam filtering // International Journal of Open Information Technologies ISSN: 2307–8162 vol. 10, no. 5, 2022. pp. 17–20.

© Охлупина Ольга Валентиновна (helga131081@yandex.ru), Мурашко Дмитрий Сергеевич (murashko100500@gmail.com).
Журнал «Современная наука: актуальные проблемы теории и практики»

г. Брянск