Алгоритм Чу—Лью/Эдмондса поиска остовного ориентированного корневого дерева минимального веса

Ижболдин А.В.
Снигирёв А.А.

Март 2025

Определения

Для начала опишем определения, которые мы будем использовать при описании данного алгоритма.

Остовное ориентированное корневое дерево (Arborescence, арборесценция)
Подграф T графа G, который:

- Является ориентированным деревом (нет циклов, если игнорировать направление).
- Имеет корень r — вершину, из которой достижимы все остальные вершины.
- В каждую вершину $v \neq r$ входит ровно одно ребро (нет входящих рёбер в корень r).

У каждого такого остовного ориентированного корневого дерева определим характеристику:

Вес остовного дерева
Весом остовного дерева T называется сумма весов ребер $e \in E_T$ входящих в него.

$$w(T) = \sum_{e \in E} w(e)$$

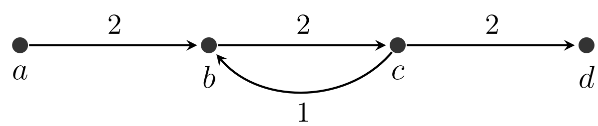
Остовное дерево минимального веса - такое остовное ориентированное корневое дерево в котором вес будет минимальным

Алгоритм Чу—Лью/Эдмондса

Алгоритм Чу—Лью/Эдмондса — это метод поиска остовного ориентированного корневого дерева (известного как арборесценция) минимального веса в ориентированном графе с весами на рёбрах. Такой граф должен содержать корневую вершину, из которой достижимы все остальные вершины. Алгоритм гарантирует построение дерева с минимальной суммой весов входящих рёбер.

Определение
MDST (minimum directed spanning tree) - минимальное остовное корневое дерево минимального веса

В ситуации неориентированных графов MDST обязательно содержит ребро минимального веса (если оно не единственное, то хотя бы одно из них). В ориентированной же ситуации, дуга минимального веса может не содержаться вообще ни в одном остовном дереве. Например на графе снизу:



В такой ситуации MDST будет состоять из $(a, b), (b, c), (c, d)$ и при этом не содержит минимальное ребро (c, b). Это означает, что алгоритмы, основанные на последовательном присоединении к уже построенному промежуточному графу рёбер минимального веса из оставшихся (такие как алгоритмы Прима и Краскала), не могут быть применены в ориентированной ситуации и нужна модификация алгоритма.

Алгоритм, который решает данную проблему предложили независимо сначала Ён-Чин Чу и Чжен-Гон Лью (1965), а затем Джек Эдмондс (1967).

Описание алгоритма

Перед описание введем определение входящих дуг в вершину и подмножество вершин:

Определение
Для вершины $v \in V$ или подмножества вершин $S \subseteq V$ обозначим $\partial^+ v$ и $\partial^+ S$ множество дуг, входящих в v и S соответственно.

Также введем переменную M_v, которая будет хранить вес минимального ребра, входящего в вершину v:

Определение
Для вершины $v \in V$ положим $M_v = \min_{e \in \partial^+ v} w(e)$.

Перейдем к описанию самого алгоритма:

Если хотя бы одна вершина графа G недостижима из r (корня), то требуемое дерево построить нельзя.

1. Для каждой вершины $u \neq v$ графа G построим G' в котором произведём следующую операцию: найдём ребро минимального веса, входящее в u, и вычтем вес этого ребра из весов всех ребер, входящих в u. Назовем ребра входящие в u как $\partial^+ u$

$$M_u = \min_{e \in \partial^+ u} w(e), \quad w'(e) = w(e) - M_u.$$

2. Строим граф $T = (V, E_0)$, где E_0 — множество ребер нулевого веса графа G'. Если этот граф является остовным дерево с корнем в v, то оно и будет искомым.

3. Иначе в графе T есть сильно связные компоненты. Для этого построим конденсацию (сжатие) графа $G'' = G' \backslash C$, (C - сильно связная компонетна в T), в котором сильно связная компонента будет сжата в "супервершину".
Все рёбра, направленные в любую вершину из C, будут теперь направлены в эту супервершину. При этом, если в G' было несколько рёбер из одной и той же вершины в разные вершины компоненты C, то в G'' из этой вершины проводится только одно ребро — с минимальным весом среди исходных.

   Аналогично, все рёбра, исходящие из вершин компоненты C и ведущие в вершины вне C, в новом графе рассматриваются как рёбра, исходящие из супервершины. Если из разных вершин C есть рёбра в одну и ту же вершину вне C, выбирается ребро с минимальным весом.

4. Повторяем шаги начиная с 1 пока не получим остовное дерево для сжатого графа.

5. Пусть в T построено MDST, теперь каждую "супервершину" заменим деревом из нулевых дуг внутри соответствующей сильно связной компоненты.

6. Полученное дерево T — MDST в графе G.

Примечание: Занулять ребра как в первом пункте для всех вершин не обязательно, можно просто хранить минимальные входящие рёбра в вершины и использовать их при построении дерева, и изменять веса только для "супервершин" в сжатом графе.

Доказательство корретности

Очевидно, что количество циклов внутри графа - конечно, а за одну сжатую вершину мы "устраняем" один цикл, следовательно алгоритм завершиться.

На первом шаге алгоритма мы создаём новый граф G' путём установки $w'(e) \leftarrow w(e) - M_v$ для всех $e \in \partial^+ v$ для каждой вершины $v \in V$. Другими словами, мы вычитаем некоторое значение из веса каждой входящей дуги в вершины так, чтобы была хотя бы одна дуга с весом 0.

Теорема
T является минимальной по весу арборесценцией в $G \Leftrightarrow T$ является минимальной по весу арборесценцией в G'.

Доказательство. Каждая вершина должна иметь ровно одну входящую дугу. Если мы уменьшим вес каждой входящей дуги на M_v, мы также уменьшим вес каждой возможной арборесценции на M_v. Таким образом, мы не влияем на минимальную по весу арборесценцию. □

Каждая вершина имеет хотя бы одну входящую дугу с весом 0. Для каждой вершины мы выбираем входящую дугу с весом 0. Если это арборесценция, то это должна быть минимальная по весу арборесценция, поскольку все веса дуг по-прежнему неотрицательны.

Рассмотрим некоторый цикл C с нулевой стоимостью. На втором шаге алгоритма мы строим новый граф $G'' = G'/C$, который представляет собой G' с циклом C, стянутым в одну вершину, с удалением дуг внутри C и заменой параллельных дуг на самую дешёвую дугу.

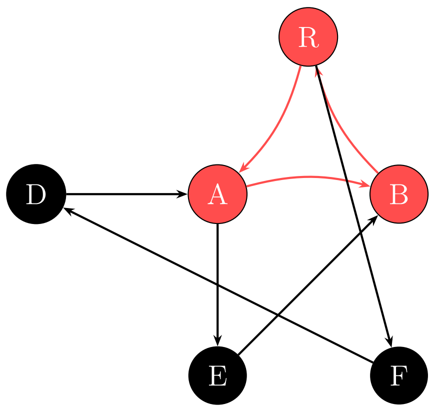


Рис. 1: Граф с циклом

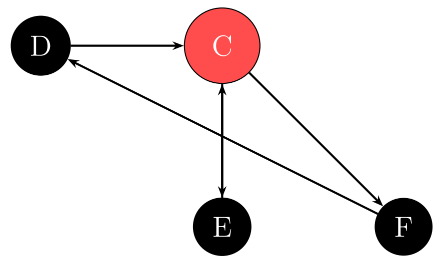


Рис. 2: Конденсированный граф

> **Теорема**
> Пусть $\mathrm{OPT}(G)$ — стоимость MDST в G. Мы утверждаем, что $\mathrm{OPT}(G') = \mathrm{OPT}(G'')$.

Доказательство. Сначала покажем, что $\mathrm{OPT}(G') \leq \mathrm{OPT}(G'')$. Предположим, что у нас есть минимальная по весу арборесценция T'' в G''. Существует некоторая вершина v_c, которая представляет некоторый цикл в G'. Мы можем построить арборесценцию T' в G' путём разворачивания цикла и удаления одного ребра в цикле (ребро у вершины у которой степень входящих ребер > 1 и принадлежит к сильно связной компоненте) Поскольку цикл имеет вес 0 на всех своих рёбрах, T' имеет тот же вес, что и T''.

Теперь покажем, что $\mathrm{OPT}(G'') \leq \mathrm{OPT}(G')$. Предположим, что у нас есть минимальная по весу арборесценция T' в G'. После стягивания некоторых вершин в G' для получения G'', если мы посмотрим на рёбра в T', для каждой получившейся «супервершины» v_C всё ещё остаётся ориентированный путь от корня r до v_C. Следовательно, мы можем удалить некоторые рёбра (нулевые ребра внутри компоненты сильной связности или же ребра которые входили в разные вершины компоненты сильной связности извне, но больше чем минимальное ребро входящие в вершину компоненты). чтобы создать арборесценцию в G''. Поскольку веса рёбер неотрицательны, мы можем только уменьшить стоимость, удаляя рёбра. Следовательно, $\mathrm{OPT}(G'') \leq \mathrm{OPT}(G')$. □

Вышеприведённое доказательство даёт алгоритм для нахождения минимальной по весу арборесценции в G' при наличии минимальной по весу арборесценции в G'' путём разворачивания стянутого цикла. Поскольку G'' имеет строго меньше вершин, чем G', мы можем теперь запустить алгоритм с начала на G'', и это индуктивно даёт алгоритм для нахождения минимальной по весу арборесценции в G.

Время работы алгоритма составляет $O(VE)$. Каждый шаг стягивания занимает $O(E)$ времени и уменьшает количество вершин по крайней мере на одну. Поскольку вершин V, то имеется не более V итераций.

Оптимизации

Также хочется отметить, что оценка $O(VE)$ не является наименьшей и алгоритм можно оптимизировать до $O(\min(E\log V, V^2))$ (реализация Тарьяна). В 1986 Габов, Галиль, Спенсер, Комптон и Тарьян предложили более быструю реализацию со временем работы $O(E + V\log V)$. В кратце раскажем про каждую из реализаций:

В реализации Тарьяна используется стек (для DFS-обхода), структура данных Union-Find (непересекающиеся множества) для отслеживания компонент сильной связности/сжатия циклов, бинарная/биномиальная куча для хранения минимальных входящих рёбер в вершины.

В реализации Габова, Галиля, Спенсера, Комптона и Тарьяна используется фибоначчиева куча вместо биноминальной кучи для хранения и быстрого обновления минимальных входящих рёбер, стек для обхода, улучшенный Union-Find (с эвристиками сжатия пути и объединения по рангу).

Применения

Конечно же встает вопрос, а для чего и как можно использовать данный алгоритм. Вот несколько его приложений:

- Проводка электроэнергии (Сам Эдмондс предложил оптимальный план проводки электричества для своего города)
- Оптимизация сети для ускорения доставки данных
- В логистических системах часто требуется спроектировать оптимальные маршруты, исходящие из центрального распределительного склада.

Это всего несколько примеров, но на самом деле алгоритм Чу-Лью/Эдмондса находит применение в самых разных областях, где требуется оптимизация ориентированных графов.

Существующие реализации

Существует множество реализаций алгоритма Чу-Лью/Эдмондса, как в виде библиотек, так и в виде отдельных программ. Вот некоторые из них:

- **Boost Graph Library (BGL)** - популярная библиотека для работы с графами на C++, которая включает реализацию алгоритма Чу-Лью/Эдмондса.
- **NetworkX** - библиотека для работы с графами на Python, которая также предоставляет реализацию данного алгоритма.