Алгоритм Краскала

Алгоритм Краскала (англ. *Kruskal's algorithm*) — алгоритм поиска минимального остовного дерева (англ. *minimum spanning tree*, *MST*) во взвешенном неориентированном связном графе.

Содержание

Идея

Будем последовательно строить подграф F графа G ("растущий лес"), пытаясь на каждом шаге достроить F до некоторого MST. Начнем с того, что включим в F все вершины графа G. Теперь будем обходить множество $E(G)$ в порядке неубывания весов ребер. Если очередное ребро e соединяет вершины одной компоненты связности F, то добавление его в остов приведет к возникновению цикла в этой компоненте связности. В таком случае, очевидно, e не может быть включено в F. Иначе e соединяет разные компоненты связности F, тогда существует $\langle S, T \rangle$ разрез такой, что одна из компонент связности составляет одну его часть, а оставшаяся часть графа — вторую. Тогда e — минимальное ребро, пересекающее этот разрез. Значит, из леммы о безопасном ребре следует, что e является безопасным, поэтому добавим это ребро в F. На последнем шаге ребро соединит две оставшиеся компоненты связности, полученный подграф будет минимальным остовным деревом графа G. Для проверки возможности добавления ребра используется система непересекающихся множеств.

Реализация

```
// G — исходный граф
// F — минимальный остов
function kruskalFindMST():
   F ← V(G)

   for vu ∈ E(G)
      if v и u в разных компонентах связности F

   return F
```

Задача о максимальном ребре минимального веса

Легко показать, что максимальное ребро в MST минимально. Обратное в общем случае неверно. Но MST из-за сортировки строится за $O(E \log E)$. Однако из-за того, что необходимо минимизировать только максимальное ребро, а не сумму всех рёбер, можно предъявить алгоритм, решающий задачу за линейное время.

С помощью алгоритма поиска k-ой порядковой статистики найдем ребро-медиану за $O(E)$ и разделим множество ребер на два равных по мощности так, чтобы ребра в первом не превосходили по весу ребер во втором. Проверим образуют ли ребра из первого подмножества остов графа, запустив обход в глубину.

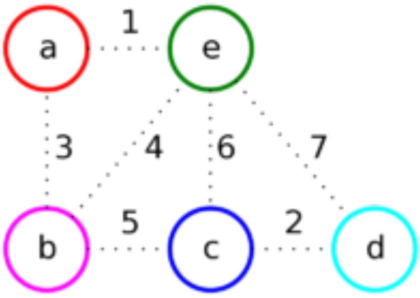
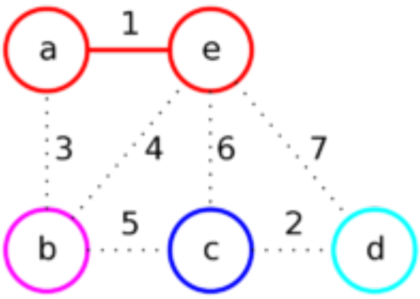
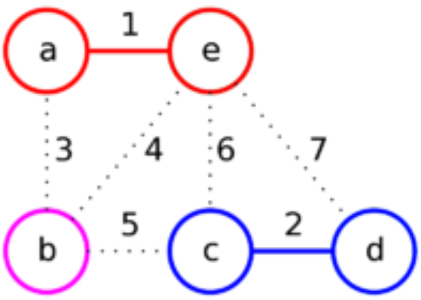
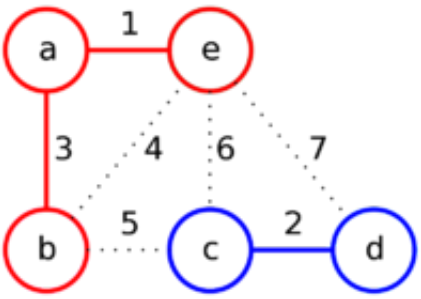
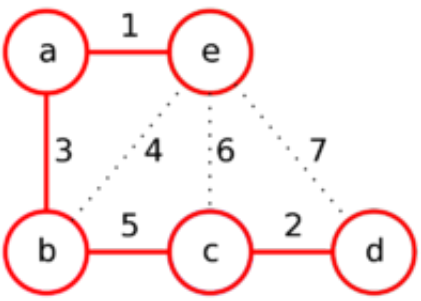
- Если да, то рекурсивно запустим алгоритм от него.
- В противном случае сконденсируем получившиеся несвязные компоненты в супервершины и рассмотрим граф с этими вершинами и ребрами из второго подмножества.

На последнем шаге останутся две компоненты связности и одно ребро в первом подмножестве — это максимальное ребро минимального веса.

На каждом шаге ребер становится в два раза меньше, а все операции выполняются за время пропорциональное количеству ребер на текущем шаге, тогда время работы алгоритма $O(E + \frac{E}{2} + \frac{E}{4} + \ldots + 1) = O(E)$.

Пример

Рёбра *(в порядке их просмотра)*	ae	cd	ab	be	bc	ec	ed
Веса рёбер	1	2	3	4	5	6	7

Изображение	Описание
	Первое ребро, которое будет рассмотрено — **ae**, так как его вес минимальный. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**a** — красное и **e** — зелёное). Объединим красное и зелёное множество в одно (красное), так как теперь они соединены ребром.
	Рассмотрим следующие ребро — **cd**. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**c** — синее и **d** — голубое). Объединим синее и голубое множество в одно (синее), так как теперь они соединены ребром.
	Дальше рассмотрим ребро **ab**. Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**a** — красное и **b** — розовое). Объединим красное и розовое множество в одно (красное), так как теперь они соединены ребром.
	Рассмотрим следующие ребро — **be**. Оно соединяет вершины из одного множества, поэтому перейдём к следующему ребру **bc** Добавим его к ответу, так как его концы соединяют вершины из разных множеств (**b** — красное и **c** — синее). Объединим красное и синее множество в одно (красное), так как теперь они соединены ребром.
	Рёбра **ec** и **ed** соединяют вершины из одного множества, поэтому после их просмотра они не будут добавлены в ответ Всё рёбра были рассмотрены, поэтому алгоритм завершает работу. Полученный граф — минимальное остовное дерево

Асимптотика

Сортировка E займет $O(E\log E)$.
Работа с СНМ займет $O(E\alpha(V))$, где α — обратная функция Аккермана, которая не превосходит 4 во всех практических приложениях и которую можно принять за константу.
Алгоритм работает за $O(E(\log E + \alpha(V))) = O(E\log E)$.

См. также

- Алгоритм Прима
- Алгоритм Борувки

Источники информации

- Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн — Алгоритмы: построение и анализ, 2-е издание. Пер. с англ. — М.:Издательский дом "Вильямс", 2010. — 1296 с.: ил. — Парал. тит. англ. — ISBN 978-5-8459-0857-5 (рус.)
- Википедия — Функция Аккермана (http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%90%D0%BA%D0%BA%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B0)
- Википедия — Алгоритм Крускала (http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB%D0%B0)
- Wikipedia — Kruskal's algorithm (http://en.wikipedia.org/wiki/Kruskal's_algorithm)
- MAXimal :: algo :: Минимальное остовное дерево. Алгоритм Крускала (http://e-maxx.ru/algo/mst_kruskal)