Алгоритм Прима

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

Содержание

Идея

Данный алгоритм очень похож на алгоритм Дейкстры. Будем последовательно строить поддерево F ответа в графе G, поддерживая приоритетную очередь Q из вершин $G \setminus F$, в которой ключом для вершины v является $\min\limits_{u \in V(F), uv \in E(G)} w(uv)$ — вес минимального ребра из вершин F в вершины $G \setminus F$. Также для каждой вершины в очереди будем хранить $p(v)$ — вершину u, на которой достигается минимум в определении ключа. Дерево F поддерживается неявно, и его ребра — это пары $(v, p(v))$, где $v \in G \setminus \{r\} \setminus Q$, а r — корень F. Изначально F пусто и значения ключей у всех вершин равны $+\infty$. Выберём произвольную вершину r и присвоим её ключу значение 0. На каждом шаге будем извлекать минимальную вершину v из приоритетной очереди и релаксировать все ребра vu, такие что $u \in Q$, выполняя при этом операцию decreaseKey над очередью и обновление $p(v)$. Ребро $(v, p(v))$ при этом добавляется к ответу.

Реализация

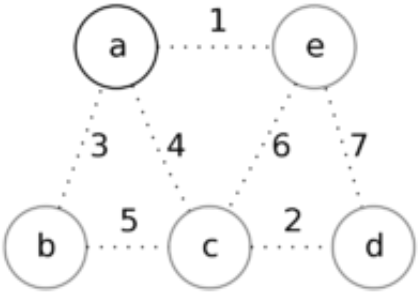
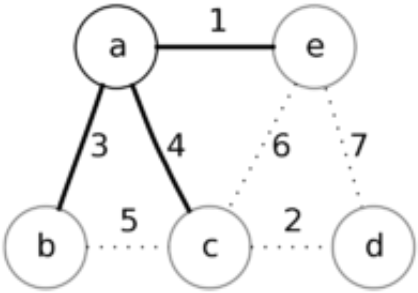
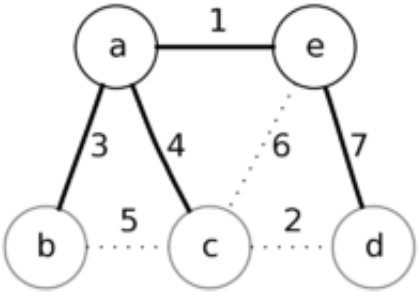
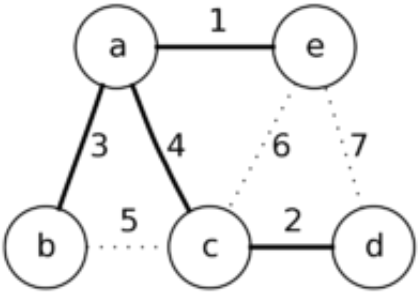
```
// G — исходный граф
// w — весовая функция
function primFindMST() :
    for v ∈ V(G)
        key[v] = ∞
        p[v] = null
    r = произвольная вершина графа G
    key[r] = 0
    Q.push(V(G))
    while not Q.isEmpty()
        v = Q.extractMin()
        for vu ∈ E(G)
            if u ∈ Q and key[u] > w(v, u)
                p[u] = v
```

```
        key[u] = w(v, u)
        Q.decreaseKey(u, key[u])
```

Ребра дерева восстанавливаются из его неявного вида после выполнения алгоритма.
Чтобы упростить операцию decreaseKey можно написать кучу на основе сбалансированного бинарного дерева поиска. Тогда просто удалим вершину и добавим ее обратно уже с новым ключом. Асимптотика таких преобразований $O(\log n)$. Если же делать с бинарной кучей, то вместо операции decreaseKey, будем всегда просто добавлять вершину с новым ключом, если из кучи достали вершину с ключом, значение которого больше чем у нее уже стоит, просто игнорировать. Вершин в куче будет не больше n^2, следовательно, операция extractMin будет выполняться за $O(\log n^2)$, что равно $O(\log n)$. Максимальное количество вершин, которое мы сможем достать, равняется количеству ребер, то есть m, поэтому общая асимптотика составит $O(m \log n)$, что хорошо только на разреженных графах.

Пример

Рассмотрим работу алгоритма на примере графа. Пусть произвольно выбранная вершина — это вершина a.

Изображение	Множество вершин	Описание
	**a b c d e** 0 ∞ ∞ ∞ ∞	Извлечём из множества вершину **a**, так как её приоритет минимален. Рассмотрим смежные с ней вершины **b**, **c**, и **e**. Обновим их приоритеты, как веса соответствующих рёбер **ab**, **ac** и **ae**, которые будут добавлены в ответ.
	a **b c d e** 0 3 4 ∞ 1	Теперь минимальный приоритет у вершины **e**. Извлечём её и рассмотрим смежные с ней вершины **a**, **c**, и **d**. Изменим приоритет только у вершины **d**, так как приоритеты вершин **a** и **c** меньше, чем веса у соответствующих рёбер **ea** и **ec**, и установим приоритет вершины **d** равный весу ребра **ed**, которое будет добавлено в ответ.
	a **b c d** e 0 3 4 7 1	После извлечения вершины **b** ничего не изменится, так как приоритеты вершин **a** и **c** меньше, чем веса у соответствующих рёбер **ba** и **bc**. Однако, после извлечения следующей вершины — **c**, будет обновлён приоритет у вершины **d** на более низкий (равный весу ребра **cd**) и в ответе ребро **ed** будет заменено на **cd**.
	a b c **d** e 0 3 4 2 1	Далее будет рассмотрена следующая вершина — **d**, но ничего не изменится, так как приоритеты вершин **e** и **c** меньше, чем веса у соответствующих рёбер **de** и **dc**. После этого алгоритм завершит работу, так как в заданном множестве не останется вершин, которые не были бы рассмотрены

Корректность

По поддерживаемым инвариантам после извлечения вершины v $(v \neq r)$ из Q ребро $(v, p(v))$ является ребром минимального веса, пересекающим разрез (F, Q). Значит, по лемме о безопасном ребре, оно безопасно. Алгоритм построения *MST*, добавляющий безопасные ребра, причём делающий это ровно $|V| - 1$ раз, корректен.

Оценка производительности

Производительность алгоритма Прима зависит от выбранной реализации приоритетной очереди, как и в алгоритме Дейкстры. Извлечение минимума выполняется V раз, релаксация — $O(E)$ раз.

Структура данных для приоритетной очереди	**Асимптотика времени работы**
Наивная реализация	$O(V^2 + E)$
Двоичная куча	$O(E \log V)$
Фибоначчиева куча	$O(V \log V + E)$

См. также

- Алгоритм Краскала
- 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. — с.653 — 656.— ISBN 978-5-8459-0857-5 (рус.)
- Википедия — Алгоритм Прима (http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0)
- Wikipedia — Prim's algorithm (http://en.wikipedia.org/wiki/Prim%27s_algorithm)
- MAXimal :: algo :: Минимальное остовное дерево. Алгоритм Прима (http://e-maxx.ru/algo/mst_prim)
