Оглавление

15.3.3. Как использовать кучи в алгоритме Прима

Кучи обеспечивают невероятно быструю, почти линейно-временную реализацию алгоритма Прима[1].

Теорема 15.4 (время выполнения алгоритма Прима (на основе кучи)). *Для каждого графа $G = (V, E)$ и вещественных реберных стоимостей реализация алгоритма Прима на основе кучи выполняется за время $O((m + n) \log n)$, где $m = |E|$ и $n = |V|$*[2].

Ограничение времени выполнения в теореме 15.4 — это только логарифмический фактор, превышающий время, необходимое для чтения входных данных. Таким образом, задача минимального остовного дерева квалифицируется как «свободный примитив», объединяющий такие элементы, как сортировка, вычисление связных компонентов графа и задача кратчайшего пути с единственным истоком.

БЕСПЛАТНЫЕ ПРИМИТИВЫ

Алгоритм с линейным или почти линейным временем выполнения можно рассматривать как «бесплатный» примитив, потому что объем используемых вычислений едва превышает объем, необходимый только для чтения входных данных. Когда у вас есть примитив, имеющий отношение к вашей задаче, к тому же необычайно быстрый, почему бы им не воспользоваться? Например, вы всегда можете вычислить минимальное остовное дерево для данных неориентированного графа на этапе предобработки, даже если не уверены, чем это обернется в дальнейшем. К слову, одна из целей этой книжной серии — снабдить ваш алгоритмический инструментарий как можно бóльшим числом бесплатных примитивов, готовых к применению в любой момент.

---

[1] Тем, кто уже прочел *часть 2*, вероятно, знакомы реализации алгоритма кратчайшего пути Дейкстры на основе кучи (см. раздел 10.4).

[2] Исходя из предположения о том, что входной граф связан, а m равно по крайней мере $n - 1$, можно упростить $O((m + n) \log n)$ до $O(m \log n)$ (в пределах времени выполнения).

В кучевой реализации алгоритма Прима объекты в куче соответствуют еще необработанным вершинам ($V - X$ в псевдокоде алгоритма `Prim`)[1, 2]. Ключ вершины $w \in V - X$ определяется как минимальная стоимость инцидентного пересекающего ребра (рис. 15.5).

ИНВАРИАНТ

Ключ вершины $w \in V - X$ есть минимальная стоимость ребра (v, w), где $v \in X$ (либо $+\infty$, если такого ребра не существует).

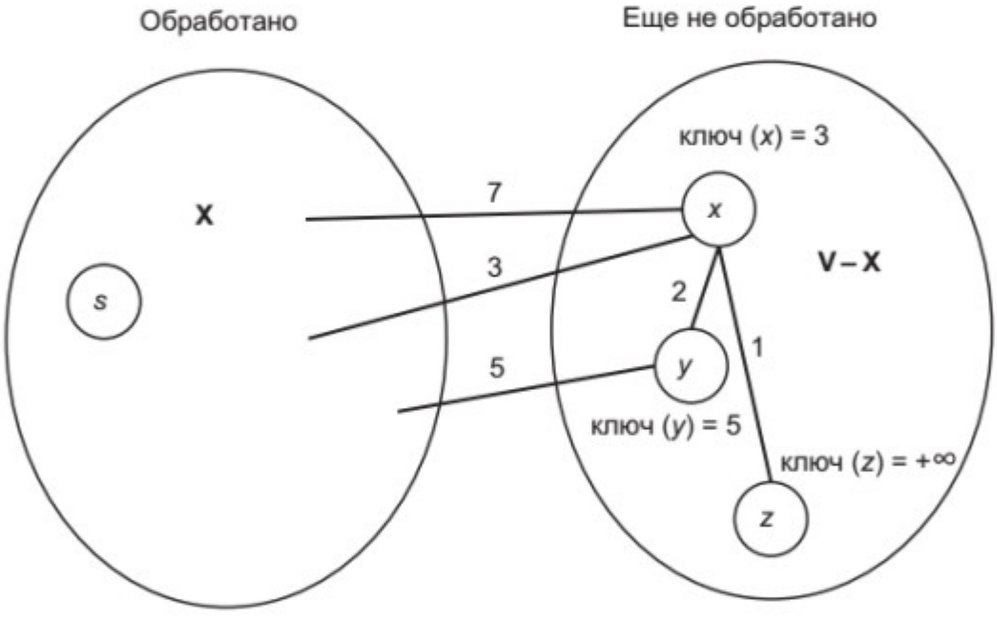


Рис. 15.5. Ключ вершины $w \in V - X$ определяется как минимальная стоимость ребра (v, w), где $v \in X$ (либо $+\infty$, если такого ребра не существует)

Интерпретируя эти ключи, представьте использование двухраундного турнира с выбыванием для выявления минимально-стоимостного ребра (v, w), где $v \in X$ и $w \notin X$. Первый раунд состоит из локального турнира для каждой

[1] Мы ссылаемся на вершины входного графа и соответствующие объекты в куче взаимозаменяемо.

[2] Да, можно хранить ребра входного графа в куче для последующей замены минимальных вычислений (по ребрам) в простой реализации с обращением к `ExtractMin`. Но более совершенная реализация хранит вершины в куче.

вершины $w \in V - X$, где участниками являются ребра (v, w) с $v \in X$, и победитель в первом туре является самым дешевым из участников (либо $+\infty$, если нет таких ребер). Победители первого раунда (не более одного на вершину $w \in V - X$) переходят во второй раунд, а окончательный чемпион является самым дешевым победителем первого раунда. Таким образом, ключом вершины $w \in V - X$ является именно стоимость выигрышного ребра в локальном турнире в w. Извлечение вершины с минимальным ключом затем реализует второй раунд турнира и возвращает на серебряном блюдечке с голубой каемочкой следующее дополнение в текущее решение. Поскольку мы платим за музыку и поддерживаем инвариант, сохраняя ключи объектов в актуальном состоянии, мы можем реализовать каждую итерацию алгоритма Прима с помощью одной кучевой операции.

15.3.4. Псевдокод

Тогда псевдокод будет выглядеть так:

PRIM (НА ОСНОВЕ КУЧИ)

Вход: связный неориентированный граф $G = (V, E)$, представленный в виде списков смежности, и стоимость c_e для каждого ребра $e \in E$.

Выход: ребра минимального остовного дерева графа G.

```
// Инициализация
 1 X := {s}, T = ∅, H := пустая куча
 2 for каждый v ≠ s do
 3     if существует ребро (s, v) ∈ E then
 4         key(v) := c_sv, winner(v) := (s, v)
 5     else      // v не имеет пересекающих инцидентных ребер
 6         key(v) := +∞, winner(v) := NULL
 7     Вставить v в H
   // Главный цикл
 8 while H не является пустым do
 9     w* := Извлечь_минимум(H)
```

```
10    добавить w* в X
11    добавить winner(w*) в T
      // обновить ключи для поддержки инварианта
12    for каждое ребро (w*, y) с y ∈ V – X do
13        if c_{w*y} < key(y) then
14            Удалить y из H
15            key(y) := c_{w*y}, winner(y) := (w*, y)
16            Вставить y в H
17  return T
```

Каждая еще не обработанная вершина w записывает в полях *победитель* `winner` и *ключ* `key` данные об идентичности и стоимости победителя своего локального турнира — самого дешевого ребра, инцидентного w, которое пересекает границу (то есть ребра (v, w), где $v \in X$). Строки 2–7 инициализируют эти значения для всех вершин, отличных от s, с целью удовлетворения инварианта, и вставляют эти вершины в кучу. Строки 9–11 реализуют одну итерацию главного цикла алгоритма Прима. Инвариант обеспечивает, чтобы локальный победитель извлеченной вершины был самым дешевым ребром, пересекающим границу, то есть являлся правильным ребром, добавляемым следующим в текущее дерево T. Упражнение 15.3 иллюстрирует, как извлечение может менять границу, требуя обновления ключей вершин, все еще находящихся в $V - X$ с целью обеспечения инварианта.

УПРАЖНЕНИЕ 15.3

На рис. 15.5 предположим, что вершина x извлечена и перемещена в множество X. Какими должны быть новые значения соответственно ключей y и z?

а) 1 и 2

б) 2 и 1

в) 5 и $+\infty$

г) $+\infty$ и $+\infty$

(Решение и пояснение см. в разделе 15.3.6.)

Строки 12–16 псевдокода выполняют необходимые обновления ключей вершин, оставшихся в $V - X$. Когда w^* перемещается из $V - X$ в X, ребра формы (w^*, y), где $y \in V - X$, пересекают границу в первый раз; в локальных турнирах в вершинах $V - X$ имеются новые участники. Мы можем игнорировать тот факт, что ребра формы (u, w^*), где $u \in X$ втягиваются в X и больше не пересекают границу, поскольку мы не несем ответственности за сохранение ключей для вершин в X. Для вершины $y \in V - X$ новым победителем ее локального турнира является либо старый *победитель* (сохраненный в `winner`(y)), либо новый участник (w^*, y). Строка 12 перебирает новых участников[1]. Строка 13 проверяет, является ли ребро (w^*, y) новым победителем локального турнира y; если да, то строки 14–16 обновляют соответственно поля *ключа* и *победителя* y и кучу H[2].

15.3.5. Анализ времени выполнения

Фаза инициализации (строки 1–7) выполняет $n - 1$ кучевых операций (по одной операции Вставить в вершину, отличную от s) и $O(m)$ дополнительной работы, где n и m обозначают соответственно число вершин и ребер. Существует $n - 1$ итераций главного цикла `while` (строки 8–16), поэтому строки 9–11 вносят в суммарное время выполнения $O(n)$ кучевых итераций и $O(n)$ дополнительной работы. Ограничение суммарного времени, проводимого в строках 12–16, является заковыристой частью; ключевое значение имеет то, что *каждое ребро графа G рассматривается в строке 12 только один раз*, в итерации, в которой первая из его конечных точек всасывается в X (то есть играет роль w^*). Когда ребро исследуется, алгоритм выполняет две кучевые операции (в строках 14 и 16) и $O(1)$ дополнительной работы, поэтому суммарный вклад строк 12–16 во время выполнения (по всем итерациям цикла) составляет $O(m)$ кучевых операций плюс $O(m)$ дополнительной работы. Окончательный перечень показателей выглядит так:

$$O(m + n) \text{ кучевых операций} + O(m + n) \text{ дополнительной работы.}$$

---

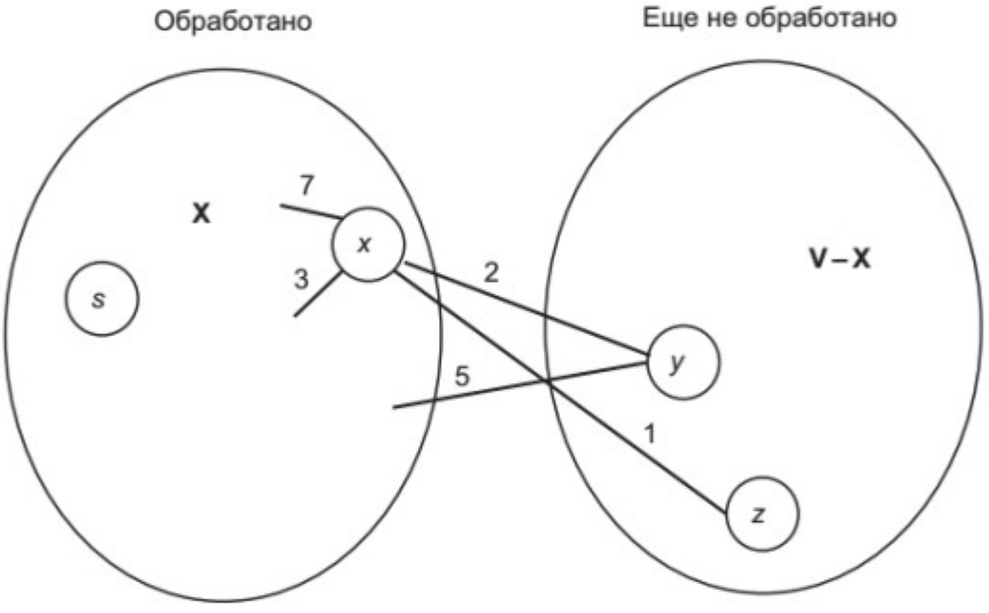
[1] На этом шаге весьма удобно представлять входной граф через списки смежности, поскольку к краям формы (w^*, y) можно обращаться напрямую через массив падающих ребер w^*.

[2] Некоторые реализации кучи экспортируют операцию `DecreaseKey`, что позволяет реализовать строки 14–16 с помощью всего одной кучевой операции, а не двух.

Куча никогда не хранит более $n - 1$ объектов, поэтому каждая кучевая операция выполняется за время $O(\log n)$ (теорема 15.3). Суммарное время выполнения составляет $O((m + n) \log n)$, согласно теореме 15.4. *Ч. Т. Д.*

15.3.6. Решение упражнения 15.3

Правильный ответ: (б). После перемещения вершины x из $V - X$ в X новое изображение выглядит так:



Ребра формы (v, x), где $v \in X$, всасываются в X и больше не пересекают границу (как и в случае с ребрами со стоимостями 3 и 7). Другие ребра, инцидентные x, (x, y) и (x, z), частично выдергиваются из $V - X$ и теперь пересекают границу. Как для y, так и для z эти новые инцидентные пересекающие ребра дешевле, чем все их старые. С целью поддержания инварианта оба их ключа должны быть обновлены: ключ y из 5 в 2, а ключ z из $+\infty$ в 1.

*15.4. Алгоритм Прима: доказательство правильности

Выполнить доказательство правильности алгоритма Прима (теорема 15.1) проще, когда все реберные стоимости различны (см. упражнение 15.5). До-