

Оглавление

Предисловие	12
О чем эта книга	13
Навыки, которые вы приобретете.....	14
В чем особенность книг этой серии.....	16
Для кого эта книга?	17
Дополнительные ресурсы.....	18
Благодарности	20
От издательства	20
Глава 13. Введение в жадные алгоритмы	21
13.1. Парадигма проектирования жадных алгоритмов	22
13.1.1. Парадигмы алгоритмов	22
13.1.2. Темы жадной парадигмы.....	23
13.2. Задача планирования.....	25
13.2.1. Постановка	25
13.2.2. Сроки завершения	26
13.2.3. Целевая функция.....	27
13.2.4. Решение упражнения 13.1	28
13.3. Разработка жадного алгоритма	28
13.3.1. Два частных случая	29
13.3.2. Дуэльные жадные алгоритмы.....	29
13.3.3. Решения упражнений 13.2–13.3	33

13.4. Доказательство правильности.....	34
13.4.1. Случай отсутствия совпадающих значений: высокоуровневый план	35
13.4.2. Обмен работами при последовательной инверсии	36
13.4.3. Анализ стоимости и преимущества.....	38
13.4.4. Обработка совпадений значений	40
13.4.5. Решения упражнений 13.4–13.5	41
Задачи на закрепление материала	44
Задачи по программированию	45

Глава 14. Коды Хаффмана..... 46

14.1. Коды.....	47
14.1.1. Двоичные коды фиксированной длины	47
14.1.2. Коды переменной длины.....	47
14.1.3. Беспрефиксные коды	49
14.1.4. Преимущества беспрефиксных кодов.....	50
14.1.5. Определение задачи.....	51
14.1.6. Решения упражнений 14.1–14.2	52
14.2. Коды в виде деревьев	52
14.2.1. Три примера	53
14.2.2. Какие деревья представляют беспрефиксные коды?	55
14.2.3. Определение задачи (в новой формулировке)	56
14.3. Жадный алгоритм Хаффмана	57
14.3.1. Построение деревьев путем последовательных слияний.....	57
14.3.2. Жадный критерий Хаффмана	60
14.3.3. Псевдокод.....	61
14.3.4. Пример	63
14.3.5. Более крупный пример	64
14.3.6. Время выполнения.....	66
14.3.7. Решение упражнения 14.3.....	67
*14.4. Доказательство правильности.....	67
14.4.1. Высокоуровневый план	68
14.4.2. Подробности	69
Задачи на закрепление материала	76
Сложные задачи	78
Задачи по программированию	78

Глава 15. Минимальные остовные деревья	79
15.1. Определение задачи	80
15.1.1. Графы.....	80
15.1.2. Остовные деревья.....	81
15.1.3. Решение упражнения 15.1	84
15.2. Алгоритм Прима.....	85
15.2.1. Пример	85
15.2.2. Псевдокод.....	88
15.2.3. Простая реализация.....	90
*15.3. Ускорение алгоритма Prim посредством куч	91
15.3.1. В поисках времени выполнения, близкого к линейному	91
15.3.2. Кучевая структура данных	92
15.3.3. Как использовать кучи в алгоритме Прима	93
15.3.4. Псевдокод.....	95
15.3.5. Анализ времени выполнения	97
15.3.6. Решение упражнения 15.3	98
*15.4. Алгоритм Прима: доказательство правильности.....	98
15.4.1. Свойство минимального узкого места	99
15.4.2. Интересные факты об остовных деревьях.....	102
15.4.3. Доказательство теоремы 15.6 (из свойства минимального узкого места следует минимальное остовное дерево)	105
15.4.4. Сводя все воедино	107
15.5. Алгоритм Краскала.....	107
15.5.1. Пример	107
15.5.2. Псевдокод.....	110
15.5.3. Простая реализация.....	111
*15.6. Ускорение алгоритма Краскала с помощью структуры данных Union-Find	112
15.6.1. Структура данных Union-Find	113
15.6.2. Псевдокод.....	115
15.6.3. Анализ времени выполнения	116
15.6.4. Быстрая и приближенная реализация структуры данных Union-Find	117
15.6.5. Решения упражнений 15.5–15.7	123
*15.7. Алгоритм Краскала: доказательство правильности.....	124

15.8. Применение: кластеризация с одиночной связью.....	126
15.8.1. Кластеризация	127
15.8.2. Восходящая кластеризация	128
Задачи на закрепление материала	132
Задачи повышенной сложности	134
Задачи по программированию	136

Глава 16. Введение в динамическое программирование..... 137

16.1. Задача о взвешенном независимом множестве	138
16.1.1. Определение задачи.....	139
16.1.2. Естественный жадный алгоритм оказывается безуспешным... 141	
16.1.3. Подход «разделяй и властвуй»?.....	142
16.1.4. Решения упражнений 16.1–16.2	143
16.2. Линейно-временной алгоритм для взвешенного независимого множества на путях.....	144
16.2.1. Оптимальная подструктура и рекуррентное соотношение	144
16.2.2. Наивный рекурсивный подход.....	147
16.2.3. Рекурсия с кэшем.....	148
16.2.4. Восходящая итеративная реализация	149
16.2.5. Решения упражнений 16.3–16.4	151
16.3. Алгоритм реконструкции.....	152
16.4. Принципы динамического программирования	155
16.4.1. Трехшаговый рецепт.....	155
16.4.2. Желаемые свойства подзадач	156
16.4.3. Повторяемый мыслительный процесс.....	157
16.4.4. Динамическое программирование против «разделяй и властвуй».....	157
16.4.5. Почему «динамическое программирование»?.....	159
16.5. Задача о ранце	160
16.5.1. Определение задачи.....	160
16.5.2. Оптимальная подструктура и рекуррентность.....	161
16.5.3. Подзадачи	164
16.5.4. Алгоритм динамического программирования	165
16.5.5. Пример	167
16.5.6. Реконструкция	168
16.5.7. Решения упражнений 16.5–16.6	169

Задачи на закрепление материала	171
Задачи повышенной сложности	173
Задачи по программированию	174
Глава 17. Расширенное динамическое программирование.....	175
17.1. Выравнивание последовательностей	176
17.1.1. Актуальность.....	176
17.1.2. Определение задачи	177
17.1.3. Оптимальная подструктура	179
17.1.4. Рекуррентное соотношение	182
17.1.5. Подзадачи.....	183
17.1.6. Алгоритм динамического программирования.....	184
17.1.7. Реконструкция.....	185
17.1.8. Решение упражнений 17.1–17.3.....	186
*17.2. Оптимальные бинарные деревья поиска	187
17.2.1. Обзор бинарного дерева поиска.....	188
17.2.2. Среднее время поиска.....	190
17.2.3. Определение задачи	192
17.2.4. Оптимальная подструктура	193
17.2.5. Рекуррентные соотношения	197
17.2.6. Подзадачи.....	198
17.2.7. Алгоритм динамического программирования.....	199
17.2.8. Улучшение времени выполнения.....	202
17.2.9. Решения упражнений 17.4–17.5.....	203
Задачи на закрепление материала	204
Задачи повышенной сложности	206
Задачи по программированию	207
Глава 18. Кратчайшие пути повторно.....	208
18.1. Кратчайшие пути с отрицательными длинами ребер	209
18.1.1. Задача о кратчайшем пути с единственным истоком.....	209
18.1.2. Отрицательные циклы	211
18.1.3. Решение упражнения 18.1	214
18.2. Алгоритм Беллмана—Форда	214
18.2.1. Подзадачи	215
18.2.2. Оптимальная подструктура	217

18.2.3. Рекуррентия	219
18.2.4. Когда следует остановиться?	220
18.2.5. Псевдокод.....	222
18.2.6. Пример	223
18.2.7. Время выполнения	226
18.2.8. Маршрутизация интернета.....	227
18.2.9. Решения упражнений 18.2–18.3	228
18.3. Задача о кратчайшем пути для всех пар	229
18.3.1. Определение задачи.....	229
18.3.2. Сведение до кратчайших путей с единственным истоком.....	230
18.3.3. Решение упражнения 18.4	231
18.4. Алгоритм Флойда—Уоршелла.....	231
18.4.1. Подзадачи	231
18.4.2. Оптимальная подструктура	233
18.4.3. Псевдокод.....	236
18.4.4. Обнаружение отрицательного цикла.....	239
18.4.5. Резюме и открытые вопросы.....	240
18.4.6. Решения упражнений 18.5–18.6	241
Задачи на закрепление материала	243
Задачи повышенной сложности	244
Задачи по программированию	245
Эпилог: руководство по разработке алгоритмов.....	246
Подсказки и решения избранных задач	248

15.3.3. Как использовать кучи в алгоритме Прима

Кучи обеспечивают невероятно быструю, почти линейно-временную реализацию алгоритма Прима¹.

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 большим 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).

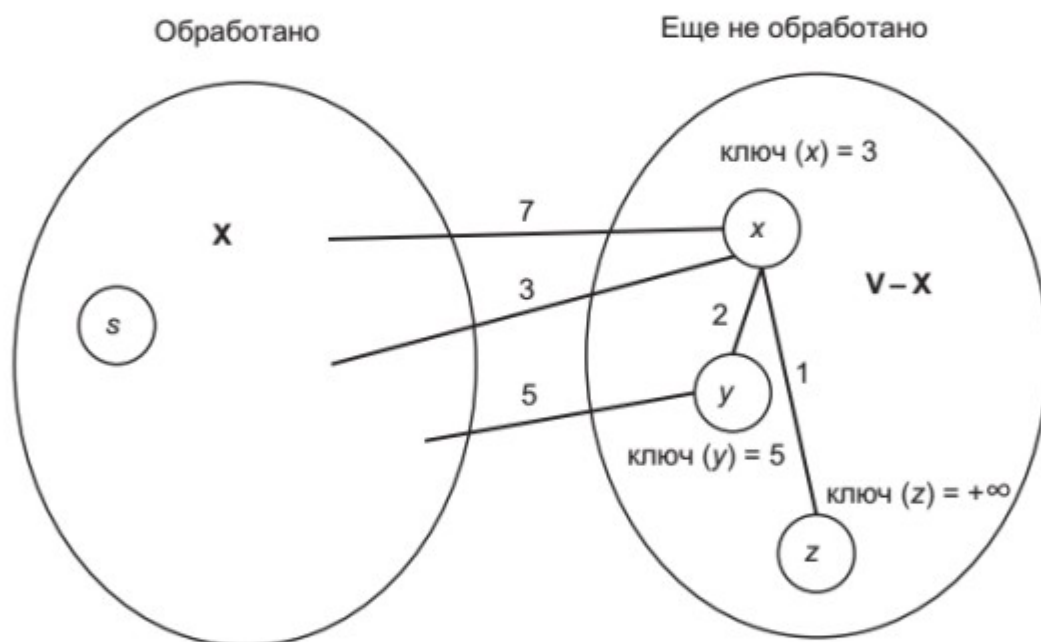


Рис. 15.5. Ключ вершины $w \in V - X$ определяется как минимальная стоимость ребра (v, w) , где $v \in X$ (либо $+\infty$, если такого ребра не существует)

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 $X := \{s\}$, $T = \emptyset$, $H :=$ пустая куча

2 **for** каждый $v \neq s$ **do**

3 **if** существует ребро $(s, v) \in E$ **then**

4 $key(v) := c_{sv}$, $winner(v) := (s, v)$

5 **else** // v не имеет пересекающих инцидентных ребер

6 $key(v) := +\infty$, $winner(v) := NULL$

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  $y \in V - X$  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 $\text{winner}(y)$), либо новый участник (w^*, y) . Строка 12 перебирает новых участников¹. 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:

$O(m + n)$ кучевых операций + $O(m + n)$ дополнительной работы.

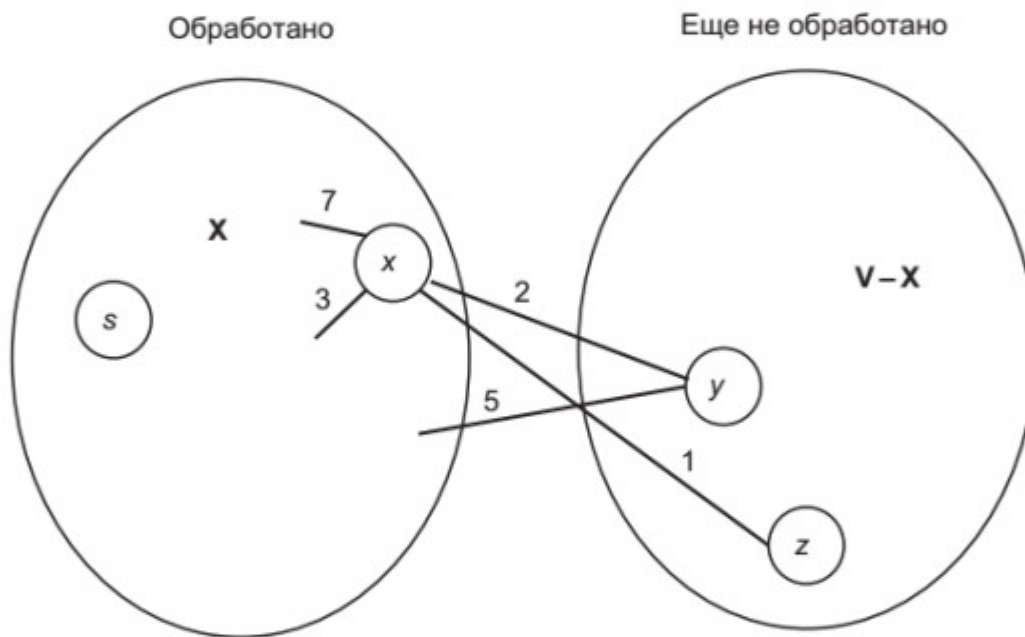
¹ На этом шаге весьма удобно представлять входной граф через списки смежности, поскольку к краям формы (w^*, y) можно обращаться напрямую через массив падающих ребер w^* .

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-