

Грокаем алгоритмы

Иллюстрированное пособие
для программистов и любопытствующих

Адитья Бхаргава



«Грокнуть» означает понять так полно, что наблюдатель становится частью объекта наблюдения...

Р. Хайнлайн

ПИТЕР





Оглавление

Предисловие	11
Благодарности	12
О книге	14
Структура книги	15
Как работать с этой книгой	16
Для кого предназначена эта книга	16
Условные обозначения и загружаемые материалы	17
Об авторе	17
От издательства	17
Глава 1. Знакомство с алгоритмами	18
Введение	18
Что вы узнаете об эффективности алгоритмов	19
Что вы узнаете о решении задач	19
Бинарный поиск	20
Более эффективный поиск	23
Упражнения	27
Время выполнения	28
«О-большое»	29
Время выполнения алгоритмов растет с разной скоростью	29

Наглядное представление «О-большое»	32
«О-большое» определяет время выполнения в худшем случае	34
Типичные примеры «О-большого»	35
<i>Упражнения</i>	36
Задача о коммивояжере	37
Шпаргалка	39
Глава 2. Сортировка выбором	40
Как работает память	41
Массивы и связанные списки	43
Связанные списки	45
Массивы	46
Терминология	47
<i>Упражнения</i>	48
Вставка в середину списка	49
Удаление	50
<i>Упражнения</i>	51
Сортировка выбором	53
Пример кода	57
Шпаргалка	58
Глава 3. Рекурсия	59
Рекурсия	60
Базовый случай и рекурсивный случай	63
Стек	65
Стек вызовов	66
<i>Упражнения</i>	68
Стек вызовов с рекурсией	69
<i>Упражнения</i>	73
Шпаргалка	74
Глава 4. Быстрая сортировка	75
«Разделяй и властвуй»	76
<i>Упражнения</i>	85
Быстрая сортировка	85
Снова об «О-большом»	92
Сортировка слиянием и быстрая сортировка	93
Средний и худший случай	95
<i>Упражнения</i>	99
Шпаргалка	99

Глава 5. Хеш-таблицы	100
Хеш-функции	103
Упражнения	107
Примеры использования	107
Использование хеш-таблиц для поиска	108
Исключение дубликатов	110
Использование хеш-таблицы как кэша	112
Шпаргалка	116
Коллизии	116
Быстродействие	119
Коэффициент заполнения	122
Хорошая хеш-функция	124
Упражнения	125
Шпаргалка	126
Глава 6. Поиск в ширину	127
Знакомство с графами	128
Что такое граф?	131
Поиск в ширину	132
Поиск кратчайшего пути	135
Очереди	136
Упражнения	137
Реализация графа	138
Реализация алгоритма	141
Время выполнения	146
Упражнения	147
Шпаргалка	150
Глава 7. Алгоритм Дейкстры	151
Работа с алгоритмом Дейкстры	152
Терминология	157
История одного обмена	160
Ребра с отрицательным весом	167
Реализация	170
Упражнения	181
Шпаргалка	181
Глава 8. Жадные алгоритмы	182
Задача составления расписания	183
Задача о рюкзаке	185

<i>Упражнения</i>	187
Задача о покрытии множества	187
Приближенные алгоритмы	189
<i>Упражнения</i>	196
NP-полные задачи	196
Задача о коммивояжере — шаг за шагом	197
Как определить, что задача является NP-полной?	202
<i>Упражнения</i>	205
Шпаргалка	205

Глава 9. Динамическое программирование 206

Задача о рюкзаке	206
Простое решение	207
Динамическое программирование	208
Задача о рюкзаке: вопросы	217
Что произойдет при добавлении элемента?	217
<i>Упражнения</i>	220
Что произойдет при изменении порядка строк?	220
Можно ли заполнять таблицу по столбцам, а не по строкам?	221
Что произойдет при добавлении меньшего элемента?	221
Можно ли взять часть предмета?	221
Оптимизация туристического маршрута	223
Взаимозависимые элементы	224
Может ли оказаться, что решение требует более двух «подрюкзаков»?	225
Возможно ли, что при лучшем решении в рюкзаке остается пустое место? . .	226
<i>Упражнения</i>	226
Самая длинная общая подстрока	226
Построение таблицы	228
Заполнение таблицы	229
Решение	230
Самая длинная общая подпоследовательность	232
Самая длинная общая подпоследовательность — решение	233
<i>Упражнения</i>	235
Шпаргалка	235

Глава 10. Алгоритм k ближайших соседей 236

Апельсины и грейпфруты	236
Построение рекомендательной системы	239
Извлечение признаков	240
<i>Упражнения</i>	245

Регрессия	245
Выбор признаков	248
<i>Упражнения</i>	249
Знакомство с машинным обучением	249
OCR	250
Построение спам-фильтра	251
Прогнозы на биржевых торгах	252
Шпаргалка	252

Глава 11. Что дальше? 254

Деревья	254
Инвертированные индексы	258
Преобразование Фурье	259
Параллельные алгоритмы	260
MapReduce	261
Для чего нужны распределенные алгоритмы?	261
Функция map	261
Функция reduce	262
Фильтры Блума и HyperLogLog	263
Фильтры Блума	265
HyperLogLog	265
Алгоритмы SHA	266
Сравнение файлов	267
Проверка паролей	268
Локально-чувствительное хеширование	269
Обмен ключами Диффи—Хеллмана	270
Линейное программирование	272
Эпилог	273

Ответы к упражнениям 274

5

Хеш-таблицы

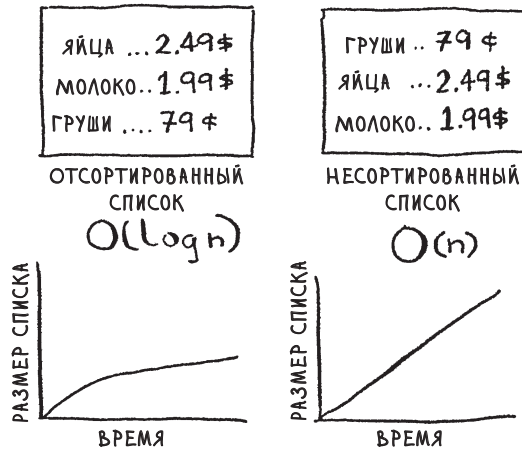


В этой главе

- ✓ Вы узнаете о хеш-таблицах — одной из самых полезных базовых структур данных. Хеш-таблицы находят множество применений; в этой главе рассматриваются распространенные варианты использования.
- ✓ Вы изучите внутреннее устройство хеш-таблиц: реализацию, коллизии и хеш-функции. Это поможет вам понять, как анализируется производительность хеш-таблицы.

Представьте, что вы продавец в маленьком магазинчике. Когда клиент покупает товары, вы проверяете их цену по книге. Если записи в книге не упорядочены по алфавиту, то поиск слова «апельсины» в каждой строке займет слишком много времени. Фактически вам придется проводить простой поиск из главы 1, а для этого нужно проверить каждую запись. Помните, сколько времени это займет? $O(n)$. Если же книга упорядочена по алфавиту, вы сможете воспользоваться бинарным поиском, время которого составляет всего $O(\log n)$.





На всякий случай напомним, что время $O(n)$ и $O(\log n)$ — далеко не одно и то же! Предположим, вы можете просмотреть 10 записей в книге за секунду. В следующей таблице показано, сколько времени займет простой и бинарный поиск.

КОЛИЧЕСТВО ЗАПИСЕЙ В КНИГЕ	$O(n)$	$O(\log n)$
100	10с	1с ← НЕОБХОДИМО ПРОВЕРИТЬ $\log_2 100 = 7$ СТРОК
1000	1.66 мин	1с ← НЕОБХОДИМО ПРОВЕРИТЬ $\log_2 1000 = 10$ СТРОК
10000	16.6 мин	2с ← $\log_2 10000 = 14$ СТРОК = 2 с

Вы уже знаете, что бинарный поиск работает очень быстро. Но поиск данных в книге — головная боль для кассира, даже если ее содержимое отсортировано. Пока вы листаете страницы, клиент потихоньку начинает выходить из себя. Гораздо удобнее было бы завести помощницу, которая помнит все названия товаров и цены. Тогда ничего искать вообще не придется: вы спрашиваете помощницу, а она мгновенно отвечает.



Ваша помощница Мэгги может за время $O(1)$ сообщить цену любого товара, независимо от размера книги. Она работает еще быстрее, чем бинарный поиск.

КОЛИЧЕСТВО ЭЛЕМЕНТОВ В КНИГЕ	ПРОСТОЙ ПОИСК	БИНАРНЫЙ ПОИСК	МЭГГИ
	$O(n)$	$O(\log n)$	$O(1)$
100	10 с	1 с	МГНОВЕННО
1000	1.6 мин	1 с	МГНОВЕННО
10000	16.6 мин	2 с	МГНОВЕННО

Просто чудо, а не девушка! И где взять такую Мэгги?

Обратимся к структурам данных. Пока вам известны две структуры данных: массивы и списки. (О стеках я не говорю, потому что нормальный поиск в стеке невозможен.) Книгу можно реализовать в виде массива.

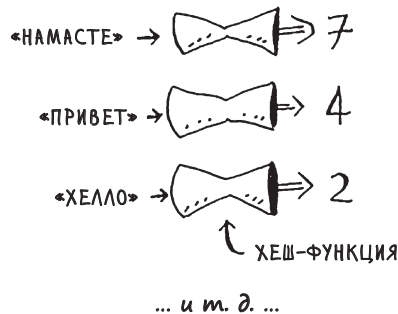
(яйца, 2.49)	(молоко, 1.49)	(груши, 0.79)
--------------	----------------	---------------

¹ В русском переводе apple переведено как апельсин, а не как яблоко, чтобы слово начиналось на букву «а». — Примеч. пер.

Каждый элемент массива на самом деле состоит из двух элементов: названия товара и его цены. Если отсортировать массив по имени, вы сможете провести по нему бинарный поиск для определения цены товара. Это означает, что поиск будет выполняться за время $O(\log n)$. Но нам нужно, чтобы поиск выполнялся за время $O(1)$ (другими словами, вы хотите создать «Мэгги»). В этом вам помогут хеш-функции.

Хеш-функции

Хеш-функция представляет собой функцию, которая получает строку¹ и возвращает число:



В научной терминологии говорят, что хеш-функция «отображает строки на числа». Можно подумать, что найти закономерности получения чисел для подаваемых на вход строк невозможно. Однако хеш-функция должна соответствовать некоторым требованиям:

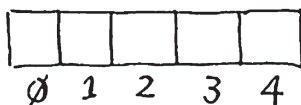
- ❑ Она должна быть последовательной. Допустим, вы передали ей строку «апельсины» и получили 4. Это значит, что каждый раз в будущем, передавая ей строку «апельсины», вы будете получать 4. Без этого хеш-таблица бесполезна.
- ❑ Разным словам должны соответствовать разные числа. Например, хеш-функция, которая возвращает 1 для каждого полученного слова, никуда

¹ Под «строкой» в данном случае следует понимать любые данные — последовательность байтов.

не годится. В идеале каждое входное слово должно отображаться на свое число.

Итак, хеш-функция связывает строки с числами. Зачем это нужно, спросите вы? Так ведь это позволит нам реализовать «Мэгги»!

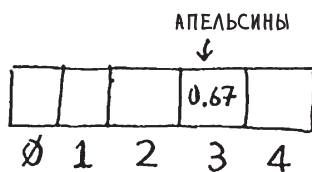
Начнем с пустого массива:



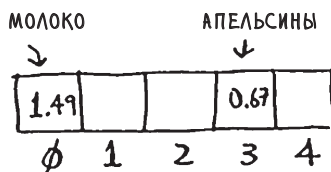
Все цены будут храниться в этом массиве; передадим хеш-функции строку «апельсины».



Хеш-функция выдает значение «3». Сохраним цену апельсинов в элементе массива с индексом 3.



Добавим молоко. Передадим хеш-функции строку «молоко».



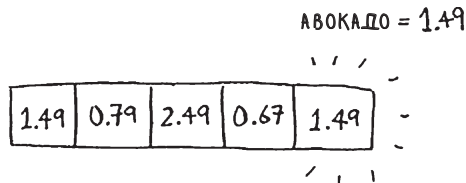
Продолжайте действовать так, и со временем весь массив будет заполнен ценами на товары.

1.49	0.79	2.49	0.67	1.49
------	------	------	------	------

А теперь вы спрашиваете: сколько стоит авокадо? Искать в массиве ничего не нужно, просто передайте строку «авокадо» хеш-функции.



Результат показывает, что значение хранится в элементе с индексом 4. И оно, конечно, там и находится!



Хеш-функция сообщает, где хранится цена, и вам вообще не нужно ничего искать! Такое решение работает, потому что:

- ❑ Хеш-функция неизменно связывает название с одним индексом. Каждый раз, когда она вызывается для строки «авокадо», вы получаете обратно одно и то же число. При первом вызове этой функции вы узнаете, где следует сохранить цену авокадо, а при последующих вызовах она сообщает, где взять эту цену.
- ❑ Хеш-функция связывает разные строки с разными индексами. «Авокадо» связывается с индексом 4, а «молоко» — с индексом 0. Для каждой строки находится отдельная позиция массива, в которой сохраняется цена этого товара.

- ❑ Хеш-функция знает размер массива и возвращает только действительные индексы. Таким образом, если длина массива равна 5 элементам, хеш-функция не вернет 100, потому что это значение не является действительным индексом в массиве.

Поздравляю: вы создали «Мэгги»! Свяжите воедино хеш-функцию и массив, и вы получите структуру данных, которая называется *хеш-таблицей*. Хеш-таблица станет первой изученной вами структурой данных, с которой связана дополнительная логика. Массивы и списки напрямую отображаются на адреса памяти, но хеш-таблицы устроены более умно. Они определяют место хранения элементов при помощи хеш-функций.

Вероятно, хеш-таблицы станут самой полезной из сложных структур данных, с которыми вы познакомитесь. Они также известны под другими названиями: «ассоциативные массивы», «словари», «отображения», «хеш-карты» или просто «хеши». Хеш-таблицы исключительно быстро работают! Помните описание массивов и связанных списков из главы 2? Обращение к элементу массива происходит мгновенно. А хеш-таблицы используют массивы для хранения данных, поэтому при обращении к элементам они не уступают массивам.

Скорее всего, вам никогда не придется заниматься реализацией хеш-таблиц самостоятельно. В любом приличном языке существует реализация хеш-таблиц. В Python тоже есть хеш-таблицы; они называются *словарями*. Новая хеш-таблица создается функцией `dict`:

```
>>> book = dict()
```



ПУСТАЯ
ХЕШ-ТАБЛИЦА

`book` — новая хеш-таблица. Добавим в `book` несколько цен:

```
>>> book["apple"] = 0.67      <.....    Апельсины стоят 67 центов
>>> book["milk"] = 1.49      <.....    Молоко стоит 1 доллар 49 центов
>>> book["avocado"] = 1.49
>>> print book
{'avocado': 1.49, 'apple': 0.67, 'milk': 1.49}
```

Пока все просто! А теперь запросим цену авокадо:

```
>>> print book["avocado"]
1.49      <----- Цена авокадо
```

Хеш-таблица состоит из ключей и значений. В хеше `book` имена продуктов являются ключами, а цены — значениями. Хеш-таблица связывает ключи со значениями.

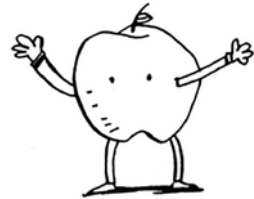
В следующем разделе приведены примеры, в которых хеш-таблицы приносят большую пользу.



ХЕШ-ТАБЛИЦА С ЦЕНАМИ
НА ПРОДУКТЫ

Упражнения

Очень важно, чтобы хеш-функции были последовательными, то есть неизменно возвращали один и тот же результат для одинаковых входных данных. Если это условие будет нарушено, вы не сможете найти свой элемент после того, как он будет помещен в хеш-таблицу!



Какие из следующих функций являются последовательными?

- 5.1 $f(x) = 1$ <----- Возвращает "1" для любых входных значений
- 5.2 $f(x) = \text{rand}()$ <----- Возвращает случайное число
- 5.3 $f(x) = \text{next_empty_slot}()$ <----- Возвращает индекс следующего пустого элемента в хеш-таблице
- 5.4 $f(x) = \text{len}(x)$ <----- Возвращает длину полученной строки

Примеры использования

Хеш-таблицы повсеместно применяются на практике. В этом разделе представлены некоторые примеры.

Конец ознакомительного фрагмента

Текст предоставлен ООО «ЛитРес».

Прочитайте книгу «**Грокаем алгоритмы**» целиком, купив полную легальную версию на ЛитРес. Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.

КУПИТЬ ПОЛНУЮ ВЕРСИЮ КНИГИ

КАК ЧИТАТЬ КНИГУ ПОСЛЕ ПОКУПКИ



Смартфон,
планшет



Компьютер,
ноутбук



Ридер

Онлайн-курс

Алгоритмы и структуры данных



GeekBrains

- Эффективные решения вычислительных задач
- Видеозаписи всех онлайн-занятий
- Сертификат об окончании обучения

⌚ 1 месяц

💻 8 занятий

Записаться на курс

monster-book.com