Грокаем алгоритмы

Иллюстрированное пособие для программистов и любопытствующих

Адитья Бхаргава

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

Р. Хайнлайн

ПИТЕР

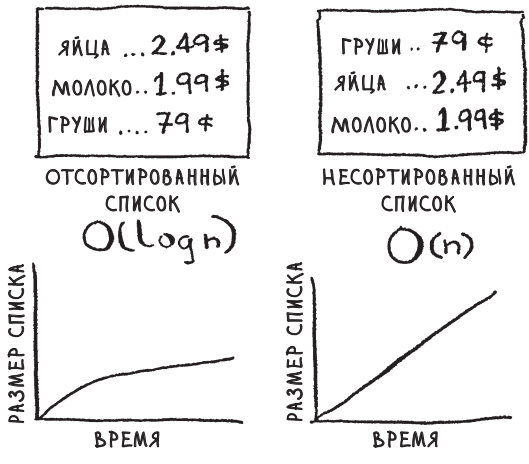
Оглавление

5 Хеш-таблицы

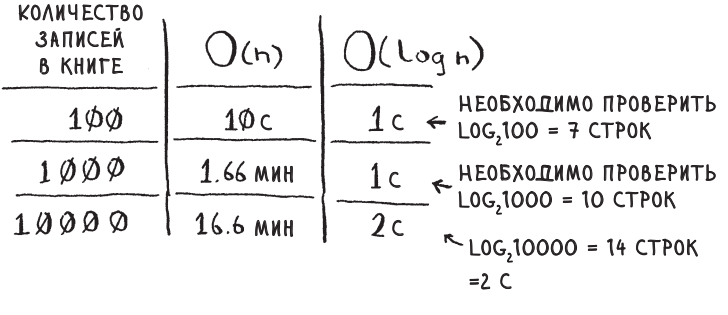
В этой главе

- ✓ 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 напомню, 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.



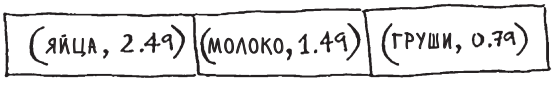
Вы уже знаете, что бинарный поиск работает очень быстро. Но поиск данных в книге — головная боль для кассира, даже если ее содержимое отсортировано. Пока вы листаете страницы, клиент потихоньку начинает выходить из себя. Гораздо удобнее было бы завести помощницу, которая помнит все названия товаров и 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
10 000	1.6 мин	1 с	МГНОВЕННО
100 000	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.



[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 строку[1] 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:

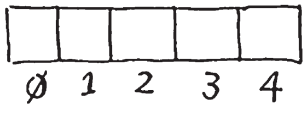
- Она должна быть последовательной. Допустим, вы передали ей строку «апельсины» и получили 4. Это значит, что каждый раз в будущем, передавая ей строку «апельсины», вы будете получать 4. Без этого хеш-таблица бесполезна.
- Разным словам должны соответствовать разные числа. Например, хеш-функция, которая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

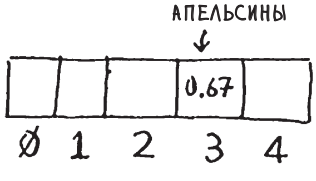
Итак, хеш-функция связывает строки с числами. Зачем это нужно, спросите вы? Так ведь это позволит нам реализовать «Мэгги»!

Начнем с пустого массива:

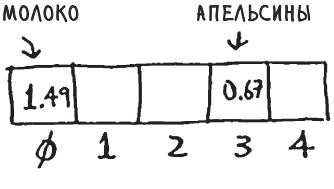


Все цены будут храниться в этом массиве; передадим хеш-функции строку «апельсины».

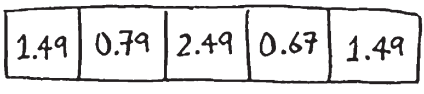
Хеш-функция выдает значение «3». Сохраним цену апельсинов в элементе массива с индексом 3.



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

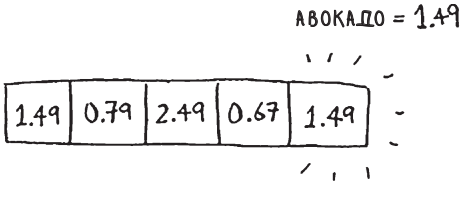


Продолжайте действовать так, и со временем весь массив будет заполнен ценами на товары.



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

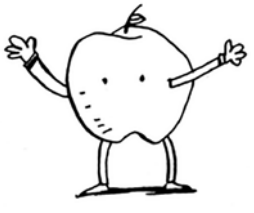
Хеш-таблица состоит из ключей и значений. В хеше book имена продуктов являются ключами, а цены — значениями. Хеш-таблица связывает ключи со значениями.

ХЕШ-ТАБЛИЦА С ЦЕНАМИ НА ПРОДУКТЫ

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

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) = rand()` — **Возвращает случайное число**

5.3 `f(x) = next_empty_slot()` — **Возвращает индекс следующего пустого элемента в хеш-таблице**

5.4 `f(x) = len(x)` — **Возвращает длину полученной строки**

Примеры использования

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

Конец ознакомительного фрагмента