Содержание

[15]

МАССИВЫ ПОИСКА

Даже начинающие программисты быстро понимают, что массивы позволяют очень легко выполнять одни и те же операции над каждым компонентом однородной группы элементов. Однако они часто не осознают, что массивы можно использовать и менее очевидными способами. Примером могут служить массивы поиска.

Описание задачи

Раньше съезды с автомагистралей нумеровались последовательными целыми числами. Первый съезд (на конкретной автомагистрали) имел номер 1, второй — номер 2 и т. д. Позже нумерация съездов с шоссе была изменена так, чтобы она соответствовала номеру мили (т. е. количеству миль, пройденных от начала шоссе до соответствующего съезда). Таким образом, съезд на отметке 1-й мили имеет номер 1, съезд на отметке 15-й мили имеет номер 15 и т. д. Возникает задача: как «преобразовать» старую нумерацию съездов в новую?

Обзор

Как вы знаете, массивы обладают двумя важными характеристиками. Во-первых, каждый элемент должен быть одного типа (т. е. элементы должны быть *однородными*). Во-вторых, индексы представляют собой последовательные неотрицательные целочисленные значения. Однако, помимо этого, нет никаких ограничений на их использование.

Когда вы впервые знакомитесь с массивами, все примеры, как правило, подразумевают какую-либо «обработку данных». Например, идет ли речь о еженедельных продажах, годовой численности населения, оценках на экзаменах и т. д. — это массивы данных. Индексы используются просто для дифференциации элементов. Однако значения индексов могут иметь смысл сами по себе. Например, в данном контексте индексы могут представлять номера съездов с автомагистрали.

Ход решения

Остается только подумать о том, должны ли индексы обозначать номера съездов по старой системе или по новой. К счастью, учитывая природу старых и новых номеров съездов, правильная схема нумерации очевидна. Индексы массива начинаются с 0, но имеют те же свойства, что и старая система нумерации съездов. Итак, если имеется пять

съездов, вы можете использовать массив длиной шесть элементов (с индексами 0, 1, ..., 5) для хранения информации о каждом съезде. В этом случае информация, которую вы хотите связать с каждым старым номером съезда, — это новый номер съезда, который сам по себе является целочисленным значением. Таким образом, вы можете хранить нужную вам информацию в массиве целочисленных значений длиной в шесть.

Например, если новые номера съездов находятся на отметках 1, 15, 16, 28 и 35 миль, вы можете сохранить их в следующем массиве с именем `NEW_NUMBERS`:

```
NEW_NUMBERS: List[int] = [-1, 1, 15, 16, 28, 35]
```

где первый элемент -1 указывает на отсутствие старого съезда с номером 0. Это показано на рисунке 15.1.

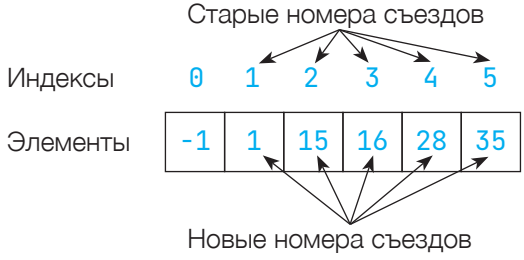

Рис. 15.1. Соответствие между старыми и новыми номерами съездов

Тогда новый номер съезда, соответствующий старому номеру съезда `i`, будет равен `NEW_NUMBERS[i]`. Например, `NEW_NUMBERS[3]` — это новый номер съезда, соответствующий старому номеру съезда 3.

Шаблон

Из этого примера сразу следует закономерность:

1. Создайте массив, в котором индексы соответствуют *ключу*, который будет использоваться для выполнения поиска, а элементы соответствуют *значению*, которое необходимо определить.
2. Создайте функцию, которая проверяет ключ и возвращает соответствующий элемент массива для любого допустимого ключа (и статус ошибки для любого недопустимого ключа).

Примеры

Некоторые другие примеры помогут проиллюстрировать как достоинства, так и ограничения, касающиеся этого шаблона.

Пример с нумерацией съездов

Продолжая пример с номерами съездов, вы можете реализовать этот шаблон следующим образом:

```
NEW_NUMBERS: List[int] = [-1, 1, 15, 16, 28, 35]

def newExitNumberFor(oldExitNumber: int) -> int:

    if ((oldExitNumber > 5)):
        return -1

    else:
        return NEW_NUMBERS[oldExitNumber]
```

Эта функция возвращает значение -1, если старый номер недействителен, в противном случае она возвращает новый номер съезда `NEW_NUMBERS[oldExitNumber]`.

Нецелые значения

Конечно, хотя индексы должны быть целыми числами (из-за природы массивов)[1], значения не обязательно должны быть целыми. Это легко проиллюстрировать на примере, где имя съезда соответствует старому номеру съезда:

```
NAMES: List[str] = ["", "Willow Ave.", "Broad St.", \
"Downtown", "North End", "Lake Dr."]
```

1 В главе 17, посвященной конформным массивам, обсуждается один из способов обойти это ограничение.

[ЧАСТЬ IV]

Шаблоны, требующие углубленных знаний о массивах

Часть IV содержит шаблоны программирования, которые требуют более глубокого понимания массивов и понимания массивов массивов (иногда называемых многомерными массивами). В частности, эта часть книги содержит следующие шаблоны программирования.

Подмассивы. Решение задач, в которых вычисления необходимо выполнять для всех элементов массива, расположенных между двумя индексами.

Окрестности. Решение задач, в которых вычисления необходимо выполнять для всех элементов массива, находящихся «в окрестности» определенного индекса.

Оба этих шаблона предполагают выполнение вычислений над подмножеством элементов массива. Они различаются способом определения подмножества.

[19]

ПОДМАССИВЫ

Описание задачи

Большинство примеров с массивами, которые вы видели, включают в себя перебор всех элементов. Однако во многих ситуациях вам нужно перебрать только некоторые элементы массива. Поскольку в этой книге вы сталкивались в основном с примерами, в которых это не так, вы, возможно, могли начать использовать шаблон, который усложняет решение поставленной задачи (поэтому его иногда называют *антишаблоном*).

Обзор

Предположим, вас попросили написать функцию, которой передается массив целочисленных значений, а возвращается общая сумма элементов массива. Вероятно, вы бы использовали аккумулятор (см. главу 13):

```
def total(data: List[int]) -> int:
    result: int = 0

    for i in range(len(data)):
        result += data[i]
    return result
```

Эта функция предполагает, что количество итераций определено, и использует цикл `for`.

Проблема этой реализации в том, что она не позволяет найти сумму подмножества элементов. Например, если индексы представляют месяцы, а элементы представляют данные о продажах, то вам может потребоваться найти общий объем продаж только за второй квартал (т. е. апрель, май и июнь; номера месяцев 3, 4, 5 отсчитываются от 0).

Ход решения

Очевидно, что вам нужно добавить в функцию формальные параметры, описывающие интересующее подмножество. Например, вы можете передать еще один массив, содержащий рассматриваемые индексы. Или вы можете передать конформный массив логических

[ЧАСТЬ V]

Шаблоны, требующие знания строковых объектов

Часть V содержит шаблоны программирования, требующие понимания строковых объектов. Эта часть книги содержит следующие шаблоны программирования.

Центрирование. Решение проблем, связанных с центрированием содержимого (разных видов) в контейнерах (разных видов).

Разделение строк. Решение задач, аналогичных задаче расстановки запятых между словами в списке.

Динамическое форматирование. Решение проблем, требующих динамического определения формата строки (т. е. во время выполнения, а не во время компиляции).

Множественность. Решение проблемы образования как правильных, так и неправильных форм множественного числа.

Первые три шаблона в этой части книги используют аккумулятор и некоторую другую логику для решения связанных с ними задач. Модель множественности на самом деле представляет собой не что иное, как умелое использование методов.

[ЧАСТЬ VI]

Шаблоны, требующие знания ссылок

Часть VI содержит шаблоны программирования, требующие понимания объектов, ссылок, а также передачи параметров. В частности, эта часть книги содержит следующие шаблоны программирования.

Цепочечные мутаторы. Решение задачи последовательного вызова нескольких различных методов для одного объекта.

Исходящие параметры. Решение задачи, когда необходимо вернуть несколько фрагментов информации из одного метода.

Отсутствующие значения. Решение проблемы отличия пропущенных значений от фактических, чтобы их можно было по-разному обрабатывать при выполнении различного рода вычислений.

Контрольные списки. Решение задачи проверки соответствия определенному набору критериев (назначенных во время выполнения).

Шаблон цепочечных мутаторов предполагает возврат ссылки, а шаблон исходящих параметров предполагает передачу ссылок. Шаблон пропущенных значений использует преимущества «специальной» ссылки `null`. Наконец, шаблон контрольных списков является обобщением шаблона битовых флагов, который позволяет создавать набор критериев динамически (т. е. во время выполнения).

[25]

ЦЕПОЧЕЧНЫЕ МУТАТОРЫ

Существуют примеры, когда метод вызывается для значения, возвращаемого другим методом, без присвоения возвращаемого значения промежуточной переменной. Мнения программистов всех уровней по поводу эффективности этой практики расходятся. Тем не менее, она довольно распространена. Поэтому важно учитывать такое поведение при реализации методов, которые могут быть объединены в цепочку таким образом.

Описание задачи

Предположим, вам нужно создать адрес электронной почты из строковой переменной, названной `user`, которая содержит имя пользователя, и строковой переменной, названной `university`, которая содержит название университета. Для повышения эффективности вы хотите использовать внешнюю библиотеку `StringBuilder` вместо конкатенации строк.

Один из способов реализации этого заключается в следующем:

```
sb = StringBuilder()
sb.append(user)
sb.append("@")
sb.append(university)
sb.append(".edu")
```

В этой реализации каждый вызов метода `append()` просто изменяет состояние `StringBuilder` по мере необходимости.

Этот способ работает, но он несколько сложнее, чем использование конкатенации строк, поскольку требует нескольких операторов. Вместо этого вы могли бы воспользоваться преимуществами класса `StringBuilder` без дополнительных сложностей. В принципе, этого можно добиться, используя цепочку вызовов следующим образом:

```
sb = StringBuilder()
sb.append(user).append("@").append(university).\
append(".edu")
```

Однако это решение будет работать только в том случае, если метод `append()` реализована с учетом подобной специфики.

Шаблон

Если вы хотите использовать цепочку вызовов для изменения объекта, то методы мутатора, которые должны быть объединены в цепочку, должны возвращать что-то, для чего может быть вызван следующий метод в цепочке. Однако он не может просто возвращать что угодно; он должен возвращать объект соответствующего типа (т. е. объект того же типа, что и объект-владелец). Но даже этого недостаточно — он должен фактически вернуть сам объект-владелец. То есть мутатор должен возвращать ссылку на сам объект-владелец.

Класс `StringBuilder` использует эту идею именно по этой причине. Методы `append()`, `delete()`, `insert()`, `replace()` и `reverse()` все возвращают ссылку на объект-владелец, что позволяет связать их вызовы в цепочку. Итак, в примере метод `sb.append(user)` возвращает ссылку на `sb`, затем `sb` вызывает метод `append(«@»)` и так далее.

Пример

Предположим, вы хотите создать инкапсуляцию объекта `Robot`, которая хранит его местоположение в одном или нескольких атрибутах и может перемещаться в четырех направлениях (вперед, назад, вправо и влево). Вам, очевидно, потребуется один или несколько методов-мутаторов для обработки перемещений. Предположим далее, что вы решили создать метод-мутатор для каждого направления с названиями `moveBackward()`, `moveForward()`, `moveLeft()` и `moveRight()`.

Если бы вы не были заинтересованы в поддержке цепочки вызовов, то эти методы были бы `void` (т. е. «пустыми», они бы ничего не возвращали), и их можно было бы использовать, как в следующем примере:

```
bender: Robot = Robot()
bender.moveForward()
bender.moveForward()
bender.moveRight()
bender.moveForward()
```