АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 3

Рекурсия. Базовый случай и рекурсивный случай. Стек

Рекурсия завершается, когда каждый вызов приближает задачу к базовому случаю.

03

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 1 Знакомство с алгоритмами. Сортировка

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

научиться выделять случаи рекурсивного решения, отслеживать ход вычислений в стеке вызовов и оценивать глубину рекурсии и затраты памяти.

- Формулировать базовый и рекурсивный случаи.
- Отслеживать кадры стека при вызове и возврате.
- Оценивать глубину и выбирать между циклом и рекурсией.

ПЛАН ЛЕКЦИИ

1. **Рекурсия как решение через меньшую задачу**
2. **Два случая и условие завершения**
3. **Стек как структура данных и стек вызовов**
4. **Трассировка рекурсии на Python**
5. **Оценка сложности и типичные ошибки**

Как стек хранит незавершённые вызовы

СХЕМА

1

Вызов

f(3) ждёт результат f(2)

2

Вызов

f(2) ждёт результат f(1)

3

База

f(1) возвращает 1

4

Возврат

f(2) = 2, затем f(3) = 6

В примере f(1) = 1, а при n > 1: f(n) = n · f(n − 1). Последний вызов завершается первым.

Как свести задачу к меньшей

01 ТЕМА

Функция называется **рекурсивной**, если прямо или через другие функции вызывает саму себя. Прямую рекурсию удобно рассмотреть на задаче подсчёта совпадений. Пусть `readings = [14, 9, 14, 3]`, а цель равна 14. Требуется вернуть число позиций с этим значением. Если рассмотреть первую позицию отдельно, от неё получается одно совпадение, а остальное можно подсчитать в хвосте списка. Хвост задаёт ту же задачу с меньшим числом элементов.

Подзадачу будем задавать начальным индексом `index`. Значение функции означает число совпадений от `index` до конца списка. При `index = 0` мы работаем со всем списком. При `index = len(readings)` рассматриваем пустой суффикс. Все вызовы получают один список, а начало рассматриваемой части задаёт индекс. Так можно повторно использовать входные данные без копирования элементов.

Два случая и доказательство завершения

Базовый случай возвращает ответ без рекурсивного вызова. Когда `index == len(readings)`, доступных позиций нет, поэтому ответ равен нулю. Условие проверяется до чтения `readings[index]`. При пустом списке оно срабатывает сразу.

Рекурсивный случай использует решение меньшей задачи. Если `index < len(readings)`, сравниваем текущий элемент с целью. Получаем вклад 0 или 1 и прибавляем число совпадений на позициях, начиная с `index + 1`. Каждое обращение решает одну и ту же задачу с новой границей.

Два случая и доказательство завершения

02 ПРОДОЛЖЕНИЕ

Чтобы обосновать завершение, введём меру размера `len(readings) - index`. При корректном начальном индексе она целая и неотрицательная. Каждый рекурсивный вызов уменьшает её ровно на единицу, а при нуле срабатывает базовый случай. Для доказательства корректности начнём с пустого суффикса. Его ответ равен 0. Если вызов с `index + 1` правильно считает совпадения в меньшем суффиксе, то прибавление вклада текущего элемента даёт верный результат для суффикса с `index`. Так доказательство проходит от простого случая к исходной задаче.

При проектировании рекурсии полезно проговорить смысл каждого параметра, ответ для самого маленького допустимого входа и изменение, приближающее к этому входу. Наличие ветки с `return` помогает только тогда, когда цепочка вызовов действительно может до неё дойти.

Стек и стек вызовов

Стек — структура, в которой добавление и удаление элементов происходят с одного конца, называемого вершиной. Последний добавленный элемент извлекается первым. Этот принцип называют LIFO. В Python список можно использовать как стек с `append` для добавления и `pop()` для извлечения последнего элемента. Перед извлечением стоит проверить, что стек непуст. Операция `pop(0)` работает с началом списка и сдвигает остальные ссылки, поэтому для вершины используем конец списка.

Стек вызовов поддерживает порядок выполнения функций. При вызове появляется кадр с параметрами, локальными переменными и сведениями о том, где продолжить исполнение после возврата. Если функция вызывает другую, вызывающий кадр остаётся в ожидании. Когда вызванная функция возвращает значение, её кадр снимается, а приостановленная функция получает результат и продолжает вычисление. Это относится и к обычным вложенным вызовам. В рекурсии на разных уровнях стека могут находиться кадры одной функции с разными значениями параметров.

Стек и стек вызовов

03 ПРОДОЛЖЕНИЕ

При подсчёте показаний каждый вызов сначала вычисляет свой вклад, а затем ожидает ответ для оставшегося суффикса. Перед возвратом из базового случая стек выглядит так, если показать только кадры функции подсчёта.

```
Вершина  index = 4, пустой суффикс, возврат 0
         index = 3, вклад 0, ждёт результат для 4
         index = 2, вклад 1, ждёт результат для 3
         index = 1, вклад 0, ждёт результат для 2
Дно      index = 0, вклад 1, ждёт результат для 1
```

Трассировка и реализация

index	Элемент	Собственный вклад	Возвращаемое значение
4	Пустой суффикс	—	0
3	3	0	`0 + 0 = 0`
2	14	1	`1 + 0 = 1`
1	9	0	`0 + 1 = 1`
0	14	1	`1 + 1 = 2`

Вызовы идут от индекса 0 к индексу 4, а значения в таблице возвращаются в обратном порядке. Например, кадр с индексом 2 знает свой вклад, но не может завершить сложение, пока не вернётся кадр с индексом 3.

Трассировка и реализация

04 ПРИМЕР PYTHON

03_01.py

```
def count_matches(values, target, index=0):
    if index == len(values):
        return 0
    contribution = 1 if values[index] == target else 0
    remaining_matches = count_matches(values, target, index + 1)
    return contribution + remaining_matches


readings = [14, 9, 14, 3]
print(count_matches(readings, 14))
print(count_matches(readings, 7))
print(count_matches([], 14))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
2
0
0
```

Трассировка и реализация

04 ПРОДОЛЖЕНИЕ

Внешний вызов использует индекс по умолчанию 0. При явной передаче индекса нужен диапазон от 0 до `len(values)` включительно. Отрицательный индекс в Python обращается к позиции с конца и нарушает смысл подзадачи. Индекс больше длины вызывает ошибку при обращении к списку. Для запуска примера достаточно Python 3, поскольку используются только встроенные операции.

Сколько стоит рекурсивное решение

При `n` элементах выполняется `n` сравнений с целью и `n + 1` вызов функции, включая базовый. Для рассматриваемых целых чисел и доступа по индексу время составляет `O(n)`. До первого возврата все `n + 1` кадров находятся в стеке, поэтому дополнительная память составляет `O(n)`. Цикл, который проходит по значениям и хранит текущий счётчик, решает ту же задачу за `O(n)` времени с `O(1)` дополнительной памятью.

Рекурсивная передача `values[1:]` вместо индекса создаёт срез и копирует ссылки на элементы на каждом шаге. Сумма длин создаваемых срезов при таком подходе растёт квадратично. Другой источник ошибки — передача того же индекса, из-за чего размер задачи перестаёт уменьшаться. При слишком глубокой рекурсии Python может выдать `RecursionError` даже у алгоритма, который теоретически завершается. Линейный обход длинной последовательности удобно записывать циклом. Рекурсивная запись особенно наглядна там, где исходную задачу естественно делить на несколько меньших частей.

Сколько стоит рекурсивное решение

05 ПРОДОЛЖЕНИЕ

Чтобы увидеть работу стека в программе, выведите `index` в начале функции и непосредственно перед каждым `return`. Первая серия сообщений покажет входы от 0 до длины списка, вторая — завершение вызовов в обратном порядке. Проследите отдельно, какое значение возвращает каждый вызов. Если забыть `return` в базовом случае, выполнение может дойти до чтения за границей списка. Если забыть его в конце рекурсивной ветки, вызывающий кадр получит `None` и не сможет прибавить его к целому числу.

Что нужно уметь после лекции

ИТОГИ

Рекурсия задаёт ответ через ответы на меньшие задачи и останавливается на базовом случае. Для проверки алгоритма нужно доказать достижимость базы и правильность перехода от меньшего случая. Стек вызовов объясняет порядок возврата значений и позволяет оценить дополнительную память по максимальной глубине.

> Рекурсия завершается, когда каждый вызов приближает задачу к базовому случаю.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что возвращает `count_matches` при произвольном допустимом значении `index`?
2. Почему базовый случай проверяется перед чтением элемента?
3. Какой кадр находится на вершине стека перед первым возвратом?
4. Чем число вызовов отличается от максимальной глубины стека?
5. Что меняется в оценке работы при передаче среза вместо индекса?

Самостоятельная работа

ПРАКТИКА

1. Для `[8, 6, 8, 8, 2]` и цели 8 нарисуйте стек перед первым возвратом. Запишите возвращаемое значение каждого вызова.
2. Напишите рекурсивную функцию суммы списка без срезов. Определите базовый случай, меру размера и сложность. Проверьте пустой список и список с отрицательными числами.
3. Реализуйте подсчёт совпадений циклом. Сравните результаты двух версий на пустом входе и на списках, где совпадений нет или совпадают все элементы.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 3. Рекурсия. С. 59–74.

> При чтении сначала вручную пройдите один пример, затем запустите код и сравните результат.