АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 4

Быстрая сортировка

Ход быстрой сортировки зависит от того, как опорный элемент делит данные.

04

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 1 Знакомство с алгоритмами. Сортировка

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

освоить стратегию «разделяй и властвуй» на примере быстрой сортировки, проследить разбиения и связать время работы и глубину рекурсии с выбором опорного элемента.

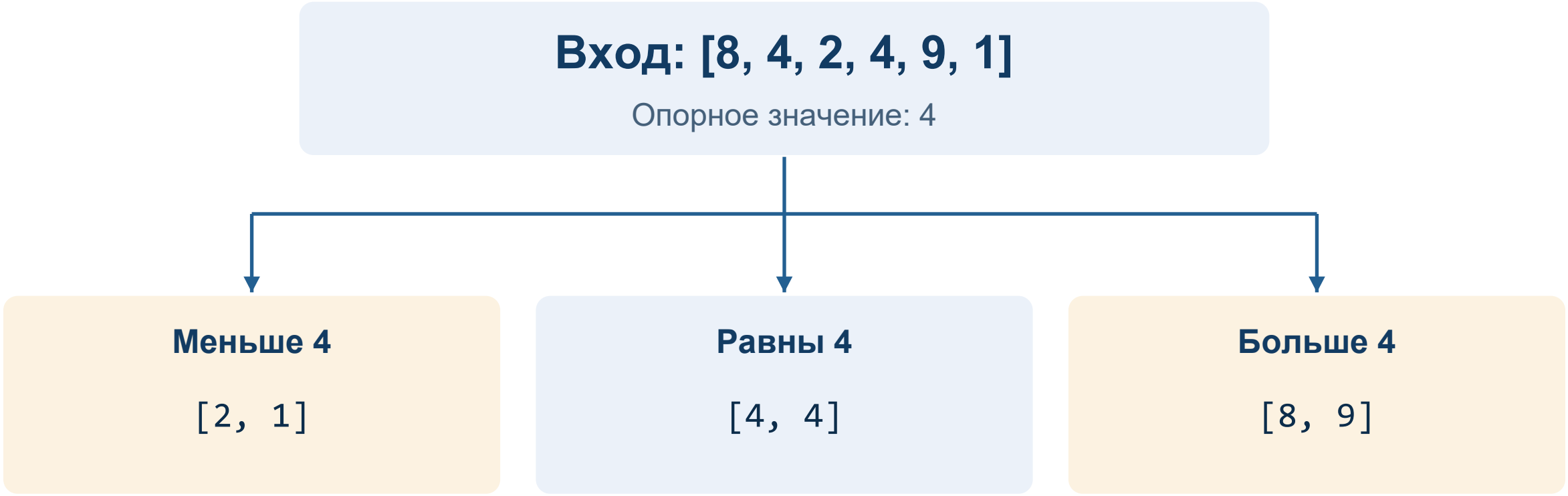
- Применять стратегию «разделяй и властвуй».
- Разбивать элементы относительно опорного.
- Объяснять влияние разбиения на время и память.

ПЛАН ЛЕКЦИИ

1. **Стратегия «разделяй и властвуй»**
2. **Разбиение относительно опорного элемента**
3. **Трассировка с повторами**
4. **Реализация и обоснование корректности**
5. **Время, память и выбор опорного элемента**
6. **Сопоставление с сортировкой слиянием**

Три группы относительно опорного элемента

СХЕМА



> Рекурсивно сортируем крайние группы и соединяем их со средней: [1, 2, 4, 4, 8, 9].

Как превратить сортировку в меньшие задачи

Стратегия **«разделяй и властвуй»** состоит из трёх действий. Разбиваем вход на меньшие части, решаем задачу для каждой части, затем собираем общий результат. Базовые случаи решаем сразу. Для сортировки это список из нуля или одного элемента. Он уже упорядочен по неубыванию.

Остаётся выбрать способ разбиения. Если просто разрезать список на две половины по позиции, после их сортировки придётся определить, как перемешать элементы половин. Именно этот подход использует сортировка слиянием. Быстрая сортировка сначала распределяет элементы по значениям. После разбиения элементы одной части заведомо меньше элементов другой, поэтому достаточно соединить отсортированные части в известном порядке.

Разбиение на три группы

Выбираем одно из значений списка как **опорное**, или `pivot`. Одним проходом распределяем элементы по трём спискам. В `less` помещаем значения меньше опорного, в `equal` — равные, в `greater` — большие. Классификация не зависит от исходной позиции элемента. Отдельная группа `equal` нужна, чтобы сохранить каждое вхождение опорного значения и сразу завершить обработку этих элементов.

Затем сортируем `less` и `greater` рекурсивно и возвращаем их вместе с `equal` в порядке `less`, `equal`, `greater`. В учебной программе опорным служит первый элемент, чтобы ручной расчёт можно было повторить. Если все значения равны, обе рекурсивные части пусты. Если опорное значение минимальное или максимальное и встречается один раз, одна подзадача имеет размер `n - 1`. Значит, выбор опорного меняет форму вычислений.

Ручная трассировка

Пусть требуется отсортировать `[23, 8, 31, 16, 23, 12, 42, 27]`. В первом разбиении опорное значение 23. Левую группу `[8, 16, 12]` сортируем отдельно. Она разбивается относительно 8 на пустую группу меньших и группу `[16, 12]`. Справа от исходного опорного окажутся числа `[31, 42, 27]` и их собственное разбиение относительно 31.

Список в вызове	Опорное	Меньше	Равны	Больше
`[23, 8, 31, 16, 23, 12, 42, 27]`	23	`[8, 16, 12]`	`[23, 23]`	`[31, 42, 27]`
`[8, 16, 12]`	8	`[]`	`[8]`	`[16, 12]`
`[16, 12]`	16	`[12]`	`[16]`	`[]`
`[31, 42, 27]`	31	`[27]`	`[31]`	`[42]`

Ручная трассировка

03 ПРОДОЛЖЕНИЕ

В таблице перечислены вызовы, выполняющие разбиение. Списки длины 0 и 1 возвращаются сразу. При сборке `[16, 12]` даёт `[12, 16]`, а весь левый список даёт `[8, 12, 16]`. Правый список даёт `[27, 31, 42]`. Обе копии числа 23 оказываются между этими частями. Если во время ручного расчёта одна копия пропала, значит, при разбиении не учли кратность значений.

Реализация и корректность

04 ПРИМЕР PYTHON

04_01.py

```
def quicksort(values):
    if len(values) < 2:
        return values.copy()

    pivot = values[0]
    less, equal, greater = [], [], []
    for value in values:
        if value < pivot:
            less.append(value)
        elif value > pivot:
            greater.append(value)
        else:
            equal.append(value)

    return quicksort(less) + equal + quicksort(greater)


values = [23, 8, 31, 16, 23, 12, 42, 27]
print(quicksort(values))
print(quicksort([5, 5, 5]))
print(values)
```

РЕЗУЛЬТАТ ЗАПУСКА

```
[8, 12, 16, 23, 23, 27, 31, 42]
[5, 5, 5]
[23, 8, 31, 16, 23, 12, 42, 27]
```

Реализация и корректность

04 ПРОДОЛЖЕНИЕ

Функция принимает список значений с согласованным отношением порядка, например список целых чисел. Она возвращает новый список и сохраняет исходный порядок входных данных. Для каждого входного элемента ровно одна ветка цикла добавляет его в одну из групп. Опорный элемент попадает в `equal`, поэтому обе рекурсивные задачи строго меньше текущей. Так можно обосновать завершение.

Для корректности проверяем два свойства результата. Во-первых, он содержит все исходные элементы с их кратностями, потому что разбиение и объединение лишь перераспределяют элементы. Во-вторых, он упорядочен. Для базового случая это очевидно. Если рекурсивные вызовы правильно упорядочивают меньшие списки, все элементы левой части меньше опорного, а правой — больше. Соединение этих частей с группой равных сохраняет порядок на обеих границах и внутри каждой части.

От чего зависят время и память

05 ТЕМА

Разбиение списка длины `n` требует `O(n)` операций. Соединение списков в данном коде также копирует ссылки и занимает `O(n)` для одного вызова. При примерно равных разбиениях глубина порядка `log n`. На одном уровне каждый элемент принадлежит не более чем одной обрабатываемой подзадаче, поэтому уровень стоит `O(n)`, а все уровни — `O(n log n)`. Для первого элемента в качестве опорного на случайной перестановке различных ключей среднее время составляет `O(n log n)`. Выбор опорного случайным образом на каждом шаге даёт такое же ожидаемое время для фиксированного входа с различными ключами.

В худшем случае один вызов оставляет подзадачу длины `n - 1`, следующий — `n - 2`, и так далее. Суммарная работа тогда растёт как `n + (n - 1) + ... + 2`, то есть `O(n²)`. Для показанной программы таким входом будет уже отсортированный список различных чисел. Если все значения равны, все они попадают в `equal`, и работа ограничивается одним проходом с последующей сборкой.

От чего зависят время и память

05 ПРОДОЛЖЕНИЕ

При примерно равных разбиениях стек содержит `O(log n)` кадров. В нашей версии также хранятся промежуточные списки. В сбалансированном случае их общий объём одновременно в памяти составляет `O(n)`, поскольку размеры вдоль активного пути убывают геометрически. В худшем случае удерживаются списки размеров `n - 1`, `n - 2` и так далее. Пиковая дополнительная память тогда может вырасти до `O(n²)`, а глубина стека — до `O(n)`. Существуют версии с перестановками внутри массива, где отдельные списки не создаются. Их оценки памяти следует выводить из конкретной реализации разбиения и рекурсии.

Сопоставление с сортировкой слиянием

06 ТЕМА

Сортировка слиянием делит список пополам по позиции независимо от значений. После сортировки половин алгоритм сравнивает их первые ещё не взятые элементы, переносит меньший в результат и продвигает указатель в соответствующей половине. Например, для `[8, 16]` и `[12, 23]` берём 8, затем 12, затем 16, после чего дописываем 23. Каждая ссылка переносится в результат один раз, поэтому слияние двух половин занимает линейное время по их общей длине.

У сортировки слиянием глубина разбиения логарифмическая для любого входа, поскольку части всегда почти равны. Поэтому время `O(n log n)` сохраняется и в худшем случае. Для быстрой сортировки основная работа на уровне происходит при разбиении, а размер подзадач зависит от опорного значения. В лабораторном сравнении фиксируйте, что именно считаете. Можно считать сравнения ключей, переносы ссылок или время запуска. Эти измерения характеризуют разные стороны алгоритма.

Что нужно уметь после лекции

ИТОГИ

Быстрая сортировка разносит значения по группам относительно опорного, сортирует меньшую и большую группы и соединяет результаты. Качество разбиений определяет глубину и время работы. Различие между памятью стека и памятью промежуточных списков важно для оценки программы на Python.

Ход быстрой сортировки зависит от того, как опорный элемент делит данные.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие три действия составляют стратегию «разделяй и властвуй»?
2. Зачем выделять группу `equal`?
3. Почему каждое рекурсивное обращение получает меньшую задачу?
4. Какой вход даёт $O(n^2)$ для выбора первого элемента опорным?
5. Из чего складывается дополнительная память реализации со списками?

Самостоятельная работа

ПРАКТИКА

1. Выполните быструю сортировку `[20, 11, 8, 20, 13, 7]` вручную. Покажите группы в каждом разбиении и итоговый результат.
2. Добавьте счётчик числа элементов, просмотренных при разбиениях. Сравните уже отсортированный список, его случайную перестановку и список из одинаковых чисел той же длины.
3. Замените первый опорный элемент случайным и проверьте несколько запусков на одном списке. Сравните результаты и число просмотренных элементов.
4. Вручную слейте `[6, 15, 22]` и `[2, 15, 19]`. Сколько сравнений понадобилось и какое правило вы применили к двум равным значениям?

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 4. Быстрая сортировка. С. 75–99.

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