АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 1

Эффективность алгоритмов. Бинарный поиск

Отсортированный массив позволяет исключать половину кандидатов за шаг.

01

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 формулировать условия задачи поиска, оценивать число операций в зависимости от размера входа и применять бинарный поиск к отсортированной последовательности.

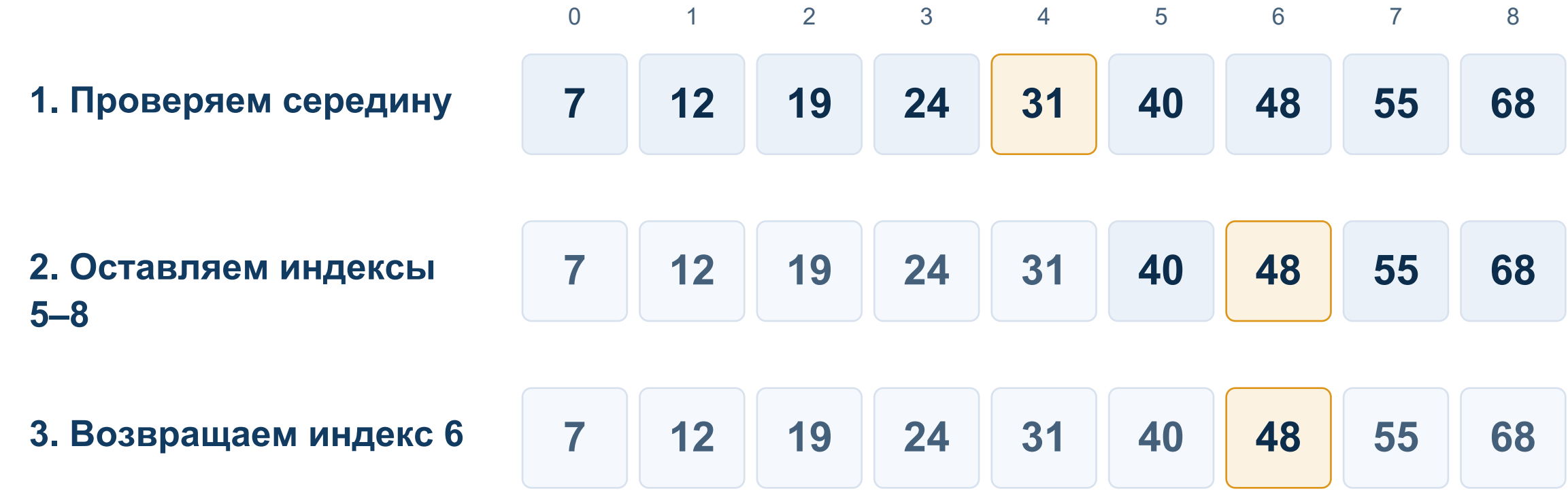
- Сравнивать время и дополнительную память алгоритмов.
- Объяснять, почему бинарный поиск требует упорядоченности.
- Вручную трассировать поиск и проверять граничные случаи.

ПЛАН ЛЕКЦИИ

1. **Задача поиска и линейный просмотр**
2. **Рост времени работы и обозначение O**
3. **Бинарный поиск и трассировка его границ**
4. **Обоснование корректности и реализация на Python**
5. **Условия применения и крайние случаи**

Один запрос, три шага

СХЕМА



Ищем 48. После сравнения с 31 можно отбросить позиции 0–4 целиком.

Что измеряет эффективность алгоритма

Алгоритм задаёт конечную последовательность действий для решения задачи. При поиске входом служат последовательность `values` и искомое значение `target`. Результатом будем считать индекс любого элемента, равного `target`, либо `None`, если подходящего элемента нет. Индексы начинаются с нуля. Допустим пустой вход. Если одинаковых элементов несколько, выбор конкретной позиции оставляем алгоритму.

Для начала рассмотрим **линейный поиск**. Он сравнивает `target` с первым элементом, затем со вторым и так далее, пока не встретит совпадение или не просмотрит всю последовательность. В списке `[31, 16, 24, 42]` число 24 будет найдено на третьем сравнении, а для 19 потребуются все четыре. Этот способ работает при любом порядке элементов. Удачное расположение искомого значения даёт одно сравнение. При отсутствии значения или его единственном вхождении в конце приходится проверить все `n` элементов.

Что измеряет эффективность алгоритма

01 ПРОДОЛЖЕНИЕ

Время на часах зависит от компьютера, реализации и характера данных. Для первого анализа выберем единицу работы, которую легко подсчитать, — проверку одного элемента. Считаем сравнение и чтение по индексу операциями постоянной стоимости, как в нашей учебной модели для списка целых чисел. Затем задаём вопрос о том, как растёт число проверок, когда n увеличивается. Дополнительную память оцениваем отдельно и не включаем в неё уже переданный вход.

Как читать обозначение O

02 ТЕМА

Запись `O(f(n))` указывает асимптотическую верхнюю границу роста. Для достаточно больших `n` число операций ограничено сверху постоянным множителем, умноженным на `f(n)`. Например, если вспомогательные действия добавляют к проходу `3n + 2` операций, оценка остаётся `O(n)`. Обозначение применимо к разным случаям работы алгоритма, поэтому нужно назвать выбранный случай. Здесь сравниваем верхние границы числа проверок в худшем случае.

`O(1)` описывает постоянную верхнюю границу, `O(n)` — линейную, `O(log n)` — логарифмическую. Двоичный логарифм отвечает на вопрос, в какую степень нужно возвести 2, чтобы получить `n`. Если каждый шаг оставляет не более половины кандидатов, при `n = 1024` достаточно десяти делений пополам, чтобы дойти до одного кандидата. Затем проверяем и его. В асимптотической записи основание логарифма можно опустить, поскольку смена основания меняет значение на постоянный множитель.

Как читать обозначение O

02 ПРОДОЛЖЕНИЕ

Число элементов	Максимум проверок при линейном поиске	Максимум проверок при бинарном поиске
8	8	4
64	64	7
1024	1024	11
1 048 576	1 048 576	21

Например, миллион записей и два миллиона записей дают почти двукратную разницу для линейного поиска. Для бинарного поиска удвоение добавляет примерно один шаг. Этот результат касается роста числа проверок. Для очень короткого списка фактическое время зависит также от постоянных издержек выполнения цикла и сравнения границ.

Как бинарный поиск сужает диапазон

Предположим, что список упорядочен по неубыванию. Храним левую и правую границы возможных позиций, обе включительно. В начале это `left = 0` и `right = len(values) - 1`. Пока границы не пересеклись, берём индекс середины `middle = (left + right) // 2` и проверяем элемент на этой позиции.

Если он совпал с целью, возвращаем индекс. Если он меньше цели, то на всех позициях до `middle` включительно стоят значения, меньшие цели. Значит, новая левая граница равна `middle + 1`. Если значение больше цели, то все позиции от `middle` до правой границы содержат слишком большие значения. Новая правая граница равна `middle - 1`.

Рассмотрим поиск числа 48 в списке `[7, 12, 19, 24, 31, 40, 48, 55, 68]`.

Как бинарный поиск сужает диапазон

03 ПРОДОЛЖЕНИЕ

Шаг	left	right	middle	Значение	Решение
1	0	8	4	31	Оставляем позиции 5–8
2	5	8	6	48	Возвращаем 6

Если ищем 47, первый шаг такой же, а второй оставляет только позицию 5 со значением 40. На третьем шаге получаем `left = 6`, `right = 5`. Диапазон пуст, поэтому возвращаем `None`. Проверка условия `left <= right` позволяет рассмотреть и диапазон, где остался ровно один кандидат.

Корректность и реализация

04 ТЕМА

Свойство, которое остаётся верным в начале каждой итерации, называют **инвариантом цикла**. Для бинарного поиска он звучит так. Если цель есть во входном списке и совпадение ещё не возвращено, то хотя бы одно вхождение находится между текущими границами. До первой итерации диапазон охватывает весь список. После каждого сравнения сортировка позволяет исключить только те позиции, где цели быть не может. При совпадении найденный индекс удовлетворяет условию задачи. При выходе из цикла допустимых позиций больше нет, и инвариант обосновывает ответ `None`. Завершение гарантировано, поскольку после каждой проверки без совпадения середина удаляется из диапазона, а длина диапазона строго уменьшается.

Корректность и реализация

04 ПРИМЕР PYTHON

01_01.py

```
def binary_search(values, target):
    left, right = 0, len(values) - 1
    while left <= right:
        middle = (left + right) // 2
        value = values[middle]
        if value == target:
            return middle
        if value < target:
            left = middle + 1
        else:
            right = middle - 1
    return None


readings = [7, 12, 19, 24, 31, 40, 48, 55, 68]
print(binary_search(readings, 48))
print(binary_search(readings, 47))
print(binary_search([], 48))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
6
None
None
```

Корректность и реализация

04 ПРОДОЛЖЕНИЕ

Для непустого списка верхняя граница числа проверок равна целой части $\log_2(n)$ плюс единица. В лучшем случае совпадение находится на первом шаге. Поскольку реализация хранит только несколько переменных и не копирует элементы, ей требуется $O(1)$ дополнительной памяти.

Когда применять бинарный поиск

05 ТЕМА

Если данные не отсортированы, сравнение в середине не позволяет исключить половину. В списке `[1, 9, 3]` поиск числа 3 увидит в середине 9 и ошибочно отбросит правую позицию. Если исходный набор отсортировать специально, нужно учитывать стоимость сортировки, а возвращаемый индекс будет относиться к новому порядку. Для одного запроса по неупорядоченному списку обычно достаточно последовательного просмотра. При подготовке к серии запросов можно отсортировать набор один раз, а затем выполнять бинарный поиск для каждой цели.

В нашей реализации предполагается быстрый доступ к произвольной позиции. Если поиск середины сам требует последовательного обхода, число сравнений и общее время могут расти по-разному. Перед использованием функции также согласуем правила сравнения и сортировки. Например, поиск по фамилиям в алфавитном списке должен учитывать те же правила регистра, по которым список упорядочен.

Когда применять бинарный поиск

05 ПРОДОЛЖЕНИЕ

Для проверки функции подходят пустой список, список с одним элементом, поиск крайних значений, отсутствующее значение между соседями и список с повторами. Проверяйте результат через `index is not None`. Индекс `0` соответствует первому элементу и в логическом условии `if index` дал бы ложное значение. Если задача требует найти самое левое совпадение, договорённость о результате меняется и после совпадения потребуется продолжить поиск слева.

Что нужно уметь после лекции

ИТОГИ

Линейный поиск просматривает до `n` кандидатов. Бинарный поиск в упорядоченном списке с доступом по индексу проверяет середину и оставляет не более половины позиций, поэтому его худшее время составляет `O(log n)`. Для обоснования результата достаточно отслеживать допустимые границы, инвариант и уменьшение диапазона.

> Отсортированный массив позволяет исключать половину кандидатов за шаг.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем размер входа отличается от числа операций?
2. Почему указание `O(n)` само по себе не говорит, какой случай работы анализируется?
3. Какое свойство отсортированного списка позволяет исключить левую половину?
4. Что произойдёт, если обновлять границу значением `middle` вместо `middle + 1` или `middle - 1`?
5. Как отличить успешный поиск первого элемента от отсутствия цели?

Самостоятельная работа

ПРАКТИКА

1. Найдите вручную 63 и 2 в списке `[4, 10, 16, 23, 31, 47, 63, 75]`. Запишите границы, середину и решение на каждом шаге.

2. Напишите линейный поиск с тем же соглашением о результате. Сравните число проверок двух алгоритмов при поиске последнего элемента и отсутствующего значения в списке `list(range(1000))`.

3. Измените бинарный поиск, чтобы для `[2, 2, 2, 5, 8]` и цели 2 он возвращал индекс первого совпадения. Сформулируйте условие, которое остаётся верным после нахождения очередного совпадения.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 1. Знакомство с алгоритмами. С. 18–39.

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