АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 2

Сортировка выбором. Массивы

Структура данных задаёт цену доступа, а сортировка — порядок действий.

02

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 1 Знакомство с алгоритмами. Сортировка

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

сравнить стоимость операций над массивом и односвязным списком, выполнить сортировку выбором вручную и обосновать её корректность и сложность.

- Сравнивать массив и связный список по операциям.
- Трассировать сортировку выбором и считать сравнения.
- Объяснять, когда обмен нарушает устойчивость.

ПЛАН ЛЕКЦИИ

1. **Массив, индекс и динамический список Python**
2. **Узлы и связи в односвязном списке**
3. **Выбор структуры по операциям над данными**
4. **Сортировка выбором и ручная трассировка**
5. **Реализация на Python, сложность и устойчивость**

Минимум занимает своё место

СХЕМА



После первого обмена число 2 стоит на своём месте. Второй проход ищет минимум только в хвосте.

Как устроен массив

01 ТЕМА

Массив хранит элементы в последовательных ячейках. Если известны адрес начала и размер ячейки, адрес позиции `i` можно вычислить как адрес начала плюс `i`, умноженное на размер ячейки. Поэтому чтение или замена по известному индексу занимает `O(1)`. Поиск значения в произвольном неупорядоченном массиве требует до `n` проверок. Индекс и искомое значение задают две разные задачи.

Динамический массив может расти. Например, в CPython встроенный `list` хранит массив ссылок на объекты. Сами объекты могут находиться в разных местах памяти. Пусть список содержит `[18, 27, 35, 44]`. При вставке 31 на позицию 2 результатом будет `[18, 27, 31, 35, 44]`. Для этой вставки ссылки на 35 и 44 сдвигаются вправо. При удалении элемента с позиции 2 они сдвигаются влево. В общем случае число сдвигов пропорционально длине хвоста, поэтому вставка и удаление в начале или середине могут занять `O(n)`.

Как устроен массив

01 ПРОДОЛЖЕНИЕ

При добавлении в конец сдвигов нет. Если есть свободная ячейка, `append` записывает ссылку за постоянное время. Когда места больше нет, динамический массив увеличивает ёмкость и может перенести все прежние ссылки. Отдельное расширение может стоить `O(n)`. Для длинной последовательности добавлений средняя стоимость в расчёте на операцию ограничена константой. Такую оценку называют **амортизированной**. Здесь она учитывает суммарную работу всей серии, даже если отдельные операции существенно различаются по времени.

В Python `values.insert(i, x)` добавляет значение `x` перед текущей позицией `i`. Вызов `values.pop(i)` удаляет элемент с позиции `i` и возвращает его. Вызов `values.remove(x)` сначала находит первое вхождение значения `x`, а потом удаляет его. Для последней операции учитываем и поиск, и сдвиг. Если значения повторяются, выбор операции зависит от того, знаем ли мы нужную позицию или хотим удалить первое совпадение.

Как устроен односвязный список

В **односвязном списке** каждый узел хранит значение и ссылку на следующий узел. `head` указывает на первый узел. Последний узел ссылается на `None`. Пустой список обозначается `head = None`. Чтобы получить элемент по индексу 2, нужно перейти от головы ко второму узлу и затем к третьему. В общем случае доступ по индексу занимает `O(n)`.

```
До вставки  head → [18] → [27] → [35] → [44] → None
После       head → [18] → [27] → [31] → [35] → [44] → None
```

Если ссылка на узел со значением 27 уже известна, для вставки достаточно направить ссылку нового узла на узел со значением 35, а затем связать 27 с новым узлом. Значения других узлов остаются на своих местах. Если переставить эти действия без сохранения старой ссылки, можно потерять доступ к хвосту. При удалении узла 31 из показанной цепочки ссылка узла 27 должна снова указывать на узел 35. Для такого удаления нужна ссылка на предшественника. Поиск предшественника от головы может потребовать линейного обхода.

Как устроен односвязный список

02 ПРИМЕР PYTHON

02_01.py

```
class Node:
    def __init__(self, value, next_node=None):
        self.value = value
        self.next = next_node


def to_list(head):
    values = []
    current = head
    while current is not None:
        values.append(current.value)
        current = current.next
    return values


previous = Node(27, Node(35, Node(44)))
head = Node(18, previous)
previous.next = Node(31, previous.next)
print(to_list(head))
previous.next = previous.next.next
print(to_list(head))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
[18, 27, 31, 35, 44]
[18, 27, 35, 44]
```

Как устроен односвязный список

02 ПРОДОЛЖЕНИЕ

Функция `to_list` создаёт отдельный список для печати. Вставка в цепочку меняет только ссылки, а преобразование для вывода обходит всю цепочку. При оценке времени этих операций учитываем их отдельно.

Как выбрать представление данных

Операция	Динамический массив	Односвязный список
Чтение по индексу	`O(1)`	До `O(n)`
Поиск значения при произвольном порядке	До `O(n)`	До `O(n)`
Вставка в начало	`O(n)`	`O(1)`
Вставка после известного узла или позиции	До `O(n)` из-за сдвига	`O(1)` при наличии ссылки на узел
Последовательный обход	`O(n)`	`O(n)`

Как выбрать представление данных

03 ПРОДОЛЖЕНИЕ

Для ежедневных показаний датчика, где регулярно нужно читать запись за конкретный день по номеру, удобен массив. Для цепочки задач, где программа уже хранит ссылку на узел и часто вставляет новую задачу сразу за ним, удобен односвязный список. Если позицию всякий раз задают числовым индексом, сначала придётся найти узел. Для серии операций полезно оценить и этот поиск, и само изменение ссылок. Кроме того, у связного списка каждый узел хранит дополнительную ссылку, а последовательное расположение ячеек массива обычно помогает при обходе.

Сортировка выбором по шагам

Разделим массив на отсортированный префикс и оставшийся суффикс. Поначалу префикс пуст. На шаге `start` ищем минимальный элемент на позициях от `start` до конца, запоминаем его индекс и меняем этот элемент местами с `values[start]`. Граница префикса сдвигается на одну позицию. Последний элемент уже окажется на месте после обработки всех предыдущих.

Шаг	Необработанная часть до шага	Минимум	Состояние массива после шага
1	`[52, 19, 37, 19, 26]`	19 с позиции 1	`[19, 52, 37, 19, 26]`
2	`[52, 37, 19, 26]`	19 с позиции 3	`[19, 19, 37, 52, 26]`
3	`[37, 52, 26]`	26 с позиции 4	`[19, 19, 26, 52, 37]`
4	`[52, 37]`	37 с позиции 4	`[19, 19, 26, 37, 52]`

Сортировка выбором по шагам

04 ПРОДОЛЖЕНИЕ

Инвариант внешнего цикла состоит в том, что префикс упорядочен и содержит столько наименьших элементов исходного массива, сколько позиций обработано. До начала сортировки это верно для пустого префикса. Каждый поиск минимума расширяет префикс следующим по величине элементом. Обмен сохраняет все исходные элементы, включая повторяющиеся значения. Когда остаётся только одна позиция, весь массив упорядочен.

Реализация, сложность и устойчивость

05 ПРИМЕР PYTHON

02_02.py

```
def selection_sort(values):
    for start in range(len(values) - 1):
        minimum_index = start
        for index in range(start + 1, len(values)):
            if values[index] < values[minimum_index]:
                minimum_index = index
        if minimum_index != start:
            values[start], values[minimum_index] = (
                values[minimum_index], values[start]
            )


values = [52, 19, 37, 19, 26]
selection_sort(values)
print(values)
```

РЕЗУЛЬТАТ ЗАПУСКА

```
[19, 19, 26, 37, 52]
```

Реализация, сложность и устойчивость

05 ПРОДОЛЖЕНИЕ

Функция меняет исходный список. Чтобы сохранить его порядок, передайте копию `values.copy()`. Для пустого списка и списка из одного элемента внешний цикл не выполняется. На первом шаге сравниваем `n - 1` пар, на следующем `n - 2`, и так до одного сравнения. Всего получается `n(n - 1) / 2` сравнений и не более `n - 1` обменов при `n ≥ 1`. Даже упорядоченный массив требует тех же сравнений. Время составляет `O(n²)`, дополнительная память — `O(1)`.

Устойчивая сортировка сохраняет взаимный порядок записей с одинаковым ключом. При сортировке оценок записи `(4, А)` и `(4, Б)` можно различать по именам, хотя ключ у них одинаков. Если исходный порядок `[4А, 4Б, 2В]`, то после обмена первого элемента с минимумом получится `[2В, 4Б, 4А]`. Обычная сортировка выбором с обменом неустойчива. Строгое сравнение `<` помогает выбрать первое вхождение минимума в суффиксе, но обмен может изменить порядок других равных записей.

Что нужно уметь после лекции

ИТОГИ

Массив позволяет быстро читать по индексу, а односвязный список — менять связи после известного узла. Сортировка выбором последовательно помещает минимум оставшейся части в префикс. Её время определяется квадратичным числом сравнений, а память нужна только для нескольких переменных.

> Структура данных задаёт цену доступа, а сортировка — порядок действий.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему добавление через `append` имеет амортизированную стоимость `O(1)`?
2. Какая информация нужна для удаления узла из середины односвязного списка за `O(1)`?
3. Сколько сравнений выполнит сортировка выбором для шести элементов?
4. Как инвариант префикса помогает обосновать корректность алгоритма?
5. Почему отсутствие равенства в условии `<` ещё не гарантирует устойчивость?

Самостоятельная работа

ПРАКТИКА

1. Покажите сдвиги в массиве `[6, 15, 28, 34, 41]` при вставке 23 на позицию 2 и удалении элемента с позиции 4 в полученном массиве. Повторите операции со связной цепочкой и обозначьте изменённые ссылки.

2. Отсортируйте вручную `[17, 6, 9, 6, 12]`. Затем добавьте в программу счётчики сравнений и обменов. Сверьте программу с ручным расчётом на этом входе и на уже упорядоченном списке.

3. Напишите вариант сортировки выбором для записей `(оценка, имя)` по оценке. Протестируйте вход, где два равных ключа меняют порядок.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 2. Сортировка выбором. С. 40–58.

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