Отчет по лабораторной работе №1

Введение в алгоритмы. Сложность. Поиск

Дата: 2025-10-03 Семестр: 3 курс 1 полугодие - 5 семестр Группа: ПИЖ-б-о-23-2(2) Дисциплина: Анализ сложности алгоритмов Студент: Мусхажиев Игорь Александрович

Цель работы

Освоить понятие вычислительной сложности алгоритма. Получить практические навыки реализации и анализа линейного и бинарного поиска. Экспериментально подтвердить теоретические оценки сложности O(n) и O(log n).

Теоретическая часть

Алгоритмическая сложность характеризует количество ресурсов (времени и памяти), необходимых алгоритму для обработки входных данных размера *n*.

- **Асимптотический анализ** — анализ поведения алгоритма при $n \to \infty$, что позволяет абстрагироваться от аппаратных особенностей и констант.
- **O-нотация** («O-большое») — верхняя асимптотическая оценка роста функции, описывающая наихудший сценарий работы алгоритма.
- **Линейный поиск (Linear Search)** — последовательный перебор всех элементов массива до нахождения целевого значения. Сложность: **O(n)**.
- **Бинарный поиск (Binary Search)** — поиск элемента в отсортированном массиве путём деления интервала поиска пополам. Сложность: **O(log n)**. Требует предварительной сортировки массива.

Практическая часть

Выполненные задачи

- [x] Реализована функция линейного поиска `linear_search(arr, target)`.
- [x] Реализована функция бинарного поиска `binary_search(arr, target)`.
- [x] Проведён теоретический анализ сложности функций.

- [x] Замерено время выполнения алгоритмов на массивах различных размеров.
- [x] Результаты визуализированы в виде графиков с линейной и логарифмической шкалой.
- [x] Добавлены характеристики ПК и проведён анализ расхождений теории и практики.

Ключевые фрагменты кода

```
# Линейный поиск O(n)
def linear_search(arr, target):
    for i in range(len(arr)):          # O(n)
        if arr[i] == target:           # O(1)
            return i                   # O(1)
    return -1                          # O(1)

# Бинарный поиск O(log n)
def binary_search(arr, target):
    left, right = 0, len(arr) - 1       # O(1)
    while left <= right:                # O(log n)
        mid = (left + right) // 2       # O(1)
        if arr[mid] == target:          # O(1)
            return mid                  # O(1)
        elif arr[mid] < target:         # O(1)
            left = mid + 1              # O(1)
        else:
            right = mid - 1             # O(1)
    return -1                           # O(1)
```

Результаты выполнения

Размер массива | Линейный поиск (с) | Бинарный поиск (с)

```
    1000 | 0.00001915 | 0.00000110
    2000 | 0.00004108 | 0.00000088
    5000 | 0.00010450 | 0.00000122
   10000 | 0.00019329 | 0.00000119
   20000 | 0.00038755 | 0.00000131
   50000 | 0.00100343 | 0.00000145
  100000 | 0.00198772 | 0.00000143
```

Характеристики ПК для тестирования

```
Процессор: 12th Gen Intel(R) Core(TM) i5-12400F
Оперативная память: 16 GB DDR4
ОС: Windows 10
Python: 3.10.10
```

Экспериментальные данные

```
Размер массива | Линейный поиск (с) | Бинарный поиск (с)
-----------------------------------------------------
        1000 | 0.00001915 | 0.00000110
        2000 | 0.00004108 | 0.00000088
        5000 | 0.00010450 | 0.00000122
       10000 | 0.00019329 | 0.00000119
       20000 | 0.00038755 | 0.00000131
       50000 | 0.00100343 | 0.00000145
      100000 | 0.00198772 | 0.00000143
```

- **time_vs_n_linear.png** — график зависимости времени поиска от размера массива в линейной шкале.
- **time_vs_n_loglog.png** — график зависимости времени поиска в логарифмической шкале (log-log).

Графики показывают линейный рост времени работы линейного поиска и логарифмический — бинарного поиска, что подтверждает теоретическую асимптотику.

Тестирование

- [x] Поиск проверен на элементе в конце массива (наихудший случай для линейного поиска).
- [x] Замеры времени проведены на массивах от 1 000 до 100 000 элементов.
- [x] Для каждого размера массива выполнено 10 повторных измерений, и рассчитано среднее значение времени.

Выводы

1.Линейный поиск демонстрирует рост времени выполнения, пропорциональный размеру массива (n), что соответствует теоретической сложности O(n). 2.Бинарный поиск показывает почти постоянное время выполнения даже при увеличении n в 100 раз, что подтверждает его сложность O(log n). 3.Экспериментальные результаты согласуются с теоретическими оценками. Незначительные отклонения объясняются влиянием констант, накладными расходами Python и особенностями архитектуры процессора.

Ответы на контрольные вопросы

1. **Что такое асимптотическая сложность алгоритма и зачем она нужна?**
   Асимптотическая сложность — это оценка ресурсов алгоритма в зависимости от размера

входных данных n. Она позволяет сравнивать эффективность алгоритмов и выбирать оптимальные решения для больших объёмов данных.

2. **Разница между O(1), O(n) и O(log n). Примеры:**
   - O(1) — время выполнения не зависит от n (например, доступ к элементу массива по индексу).
   - O(n) — время растёт линейно с ростом n (линейный поиск).
   - O(log n) — время растёт логарифмически (бинарный поиск в отсортированном массиве).

3. **Отличие линейного поиска от бинарного. Условия бинарного поиска:**
   Линейный поиск последовательно проверяет каждый элемент (O(n)). Бинарный поиск работает только с отсортированным массивом и на каждом шаге уменьшает область поиска вдвое (O(log n)).

4. **Почему на практике время может отличаться от теоретической оценки:**
   Из-за скрытых констант в O-нотации, особенностей реализации языка (например, Python), архитектуры процессора (кэширование, предсказание ветвлений) и конкретного набора входных данных.

5. **Как экспериментально подтвердить сложность:**
   Необходимо замерить время выполнения алгоритма на входах разного размера, построить график зависимости времени от n и сравнить его с теоретическими кривыми (линейной, логарифмической и т.д.).

Приложения

- Исходный код программы
- Результаты замеров времени (results.txt)
- Графики: ![График зависимости времени поиска от размера массива в линейной шкале - time_vs_n_linear.png]