Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 3 по курсу "Анализ алгоритмов"

Тема Поиск в словаре

Студент Беляев Н.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Поиск данных в хранилище является ключевой задачей при разработке информационных систем. Обработка данных требует их предварительного поиска и извлечения. Скорость поиска данных критически влияет на скорость работы системы.

Цель работы — описать, реализовать и сравнить алгоритмы линейного и бинарного поиска в словаре.

Для достижения цели необходимо выполнить следующие задачи:

- описать алгоритмы;
- спроектировать, реализовать и протестировать алгоритмы;
- провести замер количества операций сравнения в ходе работы алгоритмов при различных положениях искомого элемента в словаре.

1 Аналитический раздел

Раздел содержит определение используемого в работе словаря и описание алгоритмов поиска в нем.

1.1 Определение словаря

В данной работе под словарем будем подразумевать массив элементов длины N. Массив – упорядоченная коллекция, доступ к элементам которой осуществляется произвольным образом по их индексам.

1.2 Линейный поиск

Алгоритм линейного поиска предполагает последовательный перебор элементов словаря до тех пор, пока искомый не будет найден. Алгоритм не предполагает предварительной обработки элементов, а искомый элемент может располагаться на любой позиции, что требует перебора всех элементов в худшем случае и обеспечивает линейную временную сложность [1].

1.3 Бинарный поиск

Алгоритм бинарного поиска требует упорядоченности элементов словаря по возрастанию. Пусть искомый элемент именуется как x. Из словаря выбирается опорный элемент $pivot$. Упорядоченность элементов гарантирует:

- $x > pivot$, то есть x правее $pivot$ и дальнейшего внимание требует только часть словаря правее $pivot$;
- $x < pivot$, то есть x левее $pivot$ и дальнейшего внимание требует только часть словаря левее $pivot$;
- $x == pivot$, то есть искомый элемент найден.

Учитывая, что в общем случае искомый элемент x может оказаться в любой ячейке словаря равновероятно, в качестве опорного элемента $pivot$ принято выбирать средний на рассматриваемом интервале элемент словаря. При таком выборе опорного элемента на каждой итерации алгоритма размер словаря усекается вдвое, что обеспечивает логарифмическую временную сложность его работы [1].

Графическая интерпретация этапов работы алгоритма представлена на рисунке 1.1:

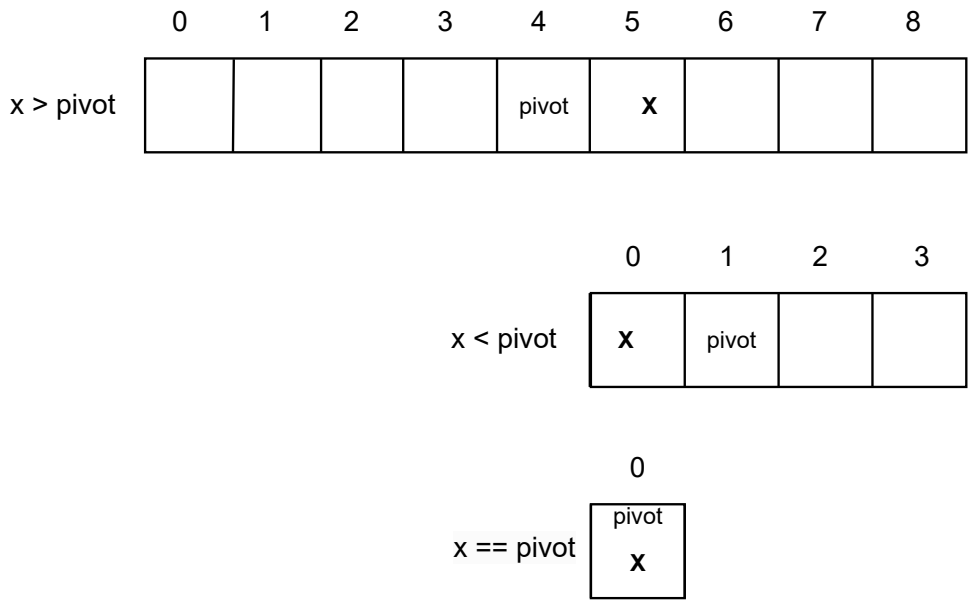

Рисунок 1.1 – Иллюстрация шагов работы алгоритма бинарного поиска

Вывод

В разделе дано определение используемого в работе словаря и описаны алгоритмы линейного и бинарного поиска.

2 Конструкторский раздел

В разделе приведены схемы алгоритмов линейного и бинарного поиска.

2.1 Схемы алгоритмов

2.1.1 Линейный поиск

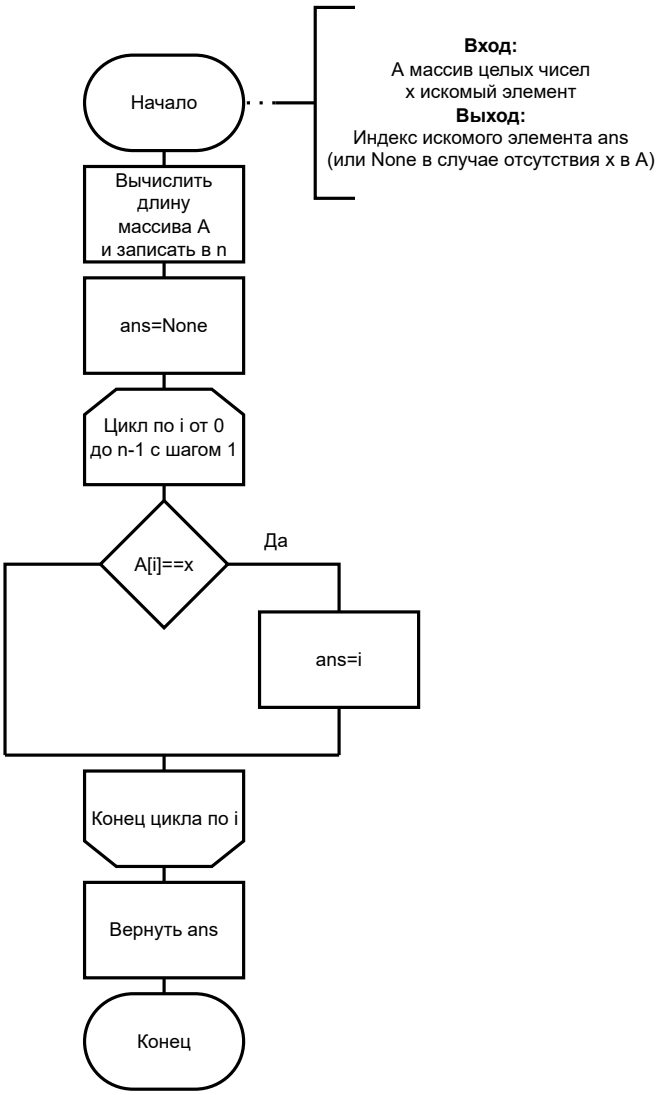

Рисунок 2.1 – Схема алгоритма линейного поиска

2.1.2 Алгоритм бинарного поиска

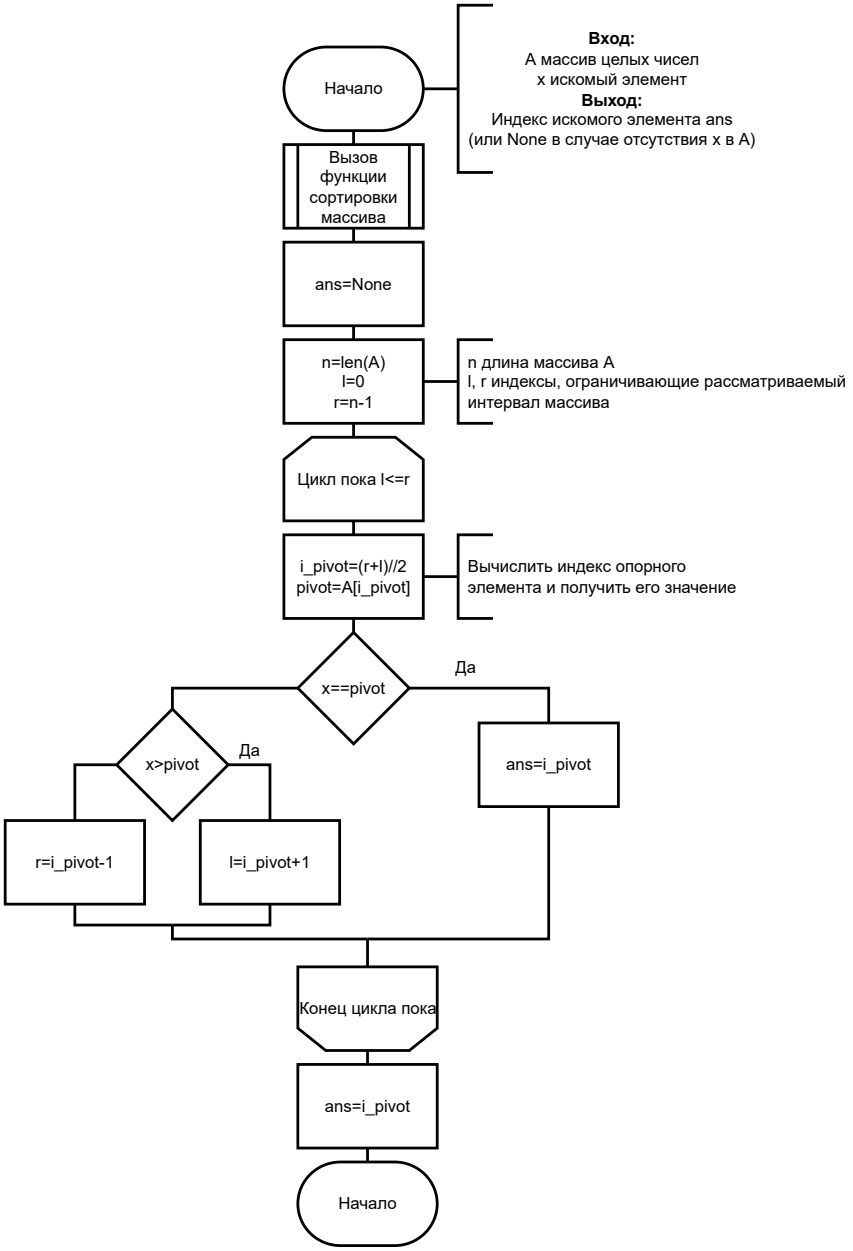

Рисунок 2.2 – Схема алгоритма бинарного поиска

Вывод

В разделе были приведены схемы алгоритмов линейного и бинарного поиска.

3 Технологический раздел

Раздел содержит описание средств реализации программы, листинги кода алгоритмов и функциональные тесты.

3.1 Средства реализации

Для реализации программы выбран язык программирования *Python* [2].

3.2 Реализация алгоритмов

Листинги 3.1 и 3.2 содержат реализации рассматриваемых алгоритмов поиска:

Листинг 3.1 – Алгоритм линейного поиска

```
def linear_search(data, x):
    n = len(data)
    ans=None
    for i in range(n):
        if data[i]==x:
            ans=i
            break
    return ans
```

Листинг 3.2 – Алгоритм бинарного поиска

```
def binary_search(data, x):
    ans=None
    data.sort()

    l = 0
    r = len(data) - 1

    while l <= r:
        i_pivot = (l + r) // 2
        pivot = data[i_pivot]

        if x == pivot:
            ans=i_pivot
            break
        elif x > pivot:
            l = i_pivot + 1
        else:
            r = i_pivot - 1

    return ans
```

3.3 Функциональное тестирование

В таблицах 3.1 и 3.2 приведены результаты функционального тестирования реализаций алгоритмов.

3.3.1 Алгоритм линейного поиска

Таблица 3.1 – Описание функционального тестирования алгоритма линейного поиска

Тест	Входные данные	Ожидаемый выход	Фактический выход
1	[1, 2, 3, 4, 5], 3	2	2
2	[1, 2, 3, 4, 5], 6	None	None
3	[ ], 1	None	None
4	[5, 4, 3, 2, 1], 1	4	4

3.3.2 Алгоритм бинарного поиска

Таблица 3.2 – Описание функционального тестирования алгоритма бинарного поиска

Тест	Входные данные	Ожидаемый выход	Фактический выход
1	[1, 2, 3, 4, 5], 3	2	2
2	[1, 2, 3, 4, 5], 6	None	None
3	[ ], 1	None	None
4	[5, 4, 3, 2, 1], 1	0	0

Все тесты пройдены успешно.

Вывод

В разделе были описаны средства реализации алгоритмов, приведены листинги кода и описание функционального тестирования.

4 Исследовательский раздел

Раздел содержит описание замера зависимости числа сравнений для поиска элемента в словаре от позиции элемента.

4.1 Исследование зависимости количества сравнений от позиции искомого элемента

Для алгоритмов линейного и бинарного поиска был проведен замер количества необходимых для поиска элемента сравнений в зависимости от позиции элемента. Замер проводился 10 раз, в качестве итогового значения выбиралось среднее арифметическое. Результаты замера приведены на гистограммах 4.1 и 4.2:

Рисунок 4.1 – Зависимость числа сравнений от позиции элемента для алгоритма линейного поиска

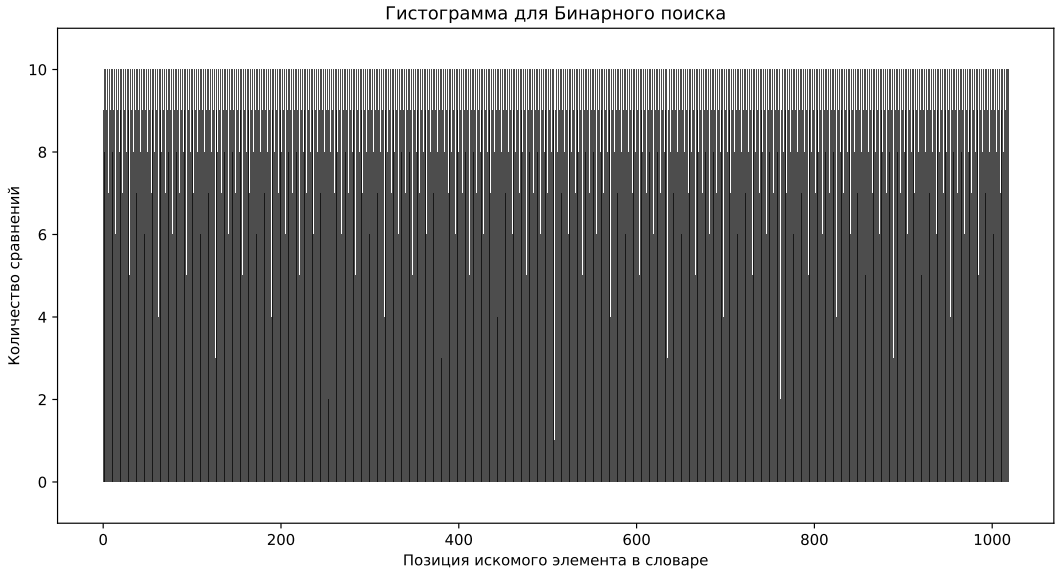

Рисунок 4.2 – Зависимость числа сравнений от позиции элемента для алгоритма бинарного поиска

В случае линейного поиска сравнений тем больше, чем дальше искомый элемент располагается от начала словаря. В лучшем случае алгоритм требует одного сравнения, если искомый элемент первый. В худшем случае алгоритм производит $N + 1$ сравнений – перебирает все элементы массива, но так и не находит искомый и выходит из цикла.

В случае бинарного поиска количество сравнений не превышает $log(n)$ – это худший случай, когда элемент дальше всего отстоит от середины словаря. Лучший случай – одно сравнение, когда искомый элемент является средним в словаре.

В среднем алгоритм бинарного поиска является более эффективным, хоть и требует предварительной сортировки элементов.

Вывод

В разделе был описан замер количества сравнений при поиске элемента в словаре.

ЗАКЛЮЧЕНИЕ

Задачи лабораторной работы выполнены:

– алгоритмы описаны, спроектированы и реализованы;

– проведен замер количества сравнений, необходимых для поиска элемента в словаре при различных позициях элемента в словаре.

Цель лабораторной работы достигнута: алгоритмы описаны и реализованы. Проведено измерение количества сравнений в ходе работа алгоритма в зависимости от позиции искомого элемента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Скиена С.* Алгоритмы. Руководство по разработке. — БХВ, 2023.
2. Python Documentation [Электронный ресурс]. — Режим доступа: `https://www.python.org` (дата обращения: 20.10.2024).