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»

Лабораторная работа № 3
по дисциплине «Анализ Алгоритмов»

Тема Поиск по словарю

Студент Пермякова Е. Д.

Группа ИУ7-52Б

Преподаватели Строганов Д. В., Волкова Л. Л

Москва, 2024

Содержание

ВВЕДЕНИЕ

Целью работы является сравнительный анализ алгоритмов поиска элемента в массиве, используя полный перебор и бинарный поиск.

Задачи:

1) Изучить алгоритмы поиска элемента в массиве, используя полный перебор и бинарный поискж
2) Реализовать алгоритмы:
   - поиска элемента в массиве полным перебором;
   - поиска элемента в массиве бинарным поиском;
3) Провести сравнительный анализ по трудоемкости работы алгоритмов;

1 Аналитическая часть

В данной работе будут рассмотрены два стандартных подхода поиска элемента в массиве: полный перебор и бинарный поиск.

1.1 Алгоритм поиска элемента в массиве полным перебором

Поиск полным перебором предполагает сравнение искомого элемента с каждым элементом массива. Возможно (N + 1) случаев: элемент не найден и N возможных случаев расположения жлемента в массиве.

Лучший случай: за одно сравнение элемент найден в начале массива.

Худших случаев два: за N сравнений либо элемент не найден, либо ключ найден на последнем сравнении.

Пусть на старте алгоритм поиска затрачивает k_0 операций, а при каждом сравнении k_1 сравнений. Тогда в лучшем случае будет затрачено $k_0 + k_1$ операций. В случае, если ключ будет найден на второй позиции, будет затрачено $k_0 + 2k_1$, на последней позиции - $k_0 + Nk_1$ операций, столько же если ключ не будет найден вовсе.

Пусть S_1 и S_2 - две строки, длиной *M* и *N* соответственно, над некоторым алфавитом, тогда расстояние Левенштейна d(s_1, s_2) можно подсчитать по следующей рекуррентной формуле d(s_1, s_2) = D(M, N):

1.2 Алгоритма поиска элемента в массиве бинарным поиском

При двоичном поиске обход можно представить деревом, поэтому трудоемкость в худшем случае составит $\log_2 N$ (в худшем случае нужно спуститься по двоичному дереве от корня до листа)

Скорость роста функции $\log_2 N$ меньше чем у N

Вывод

В данном разделе были теоретически разобраны два стандартных подхода поиска элемента в массиве: полный перебор и бинарный поиск.

2 Конструкторская часть

В этом разделе будут представлens схемы алгоритмов поиска элемента в массиве полным перебором и бинарным поиском.

2.1 Описание алгоритмов

На рисунках 2.1-2.3 представлены схемы алгоритмов поиска элемента в массиве полным перебором и бинарным поиском.

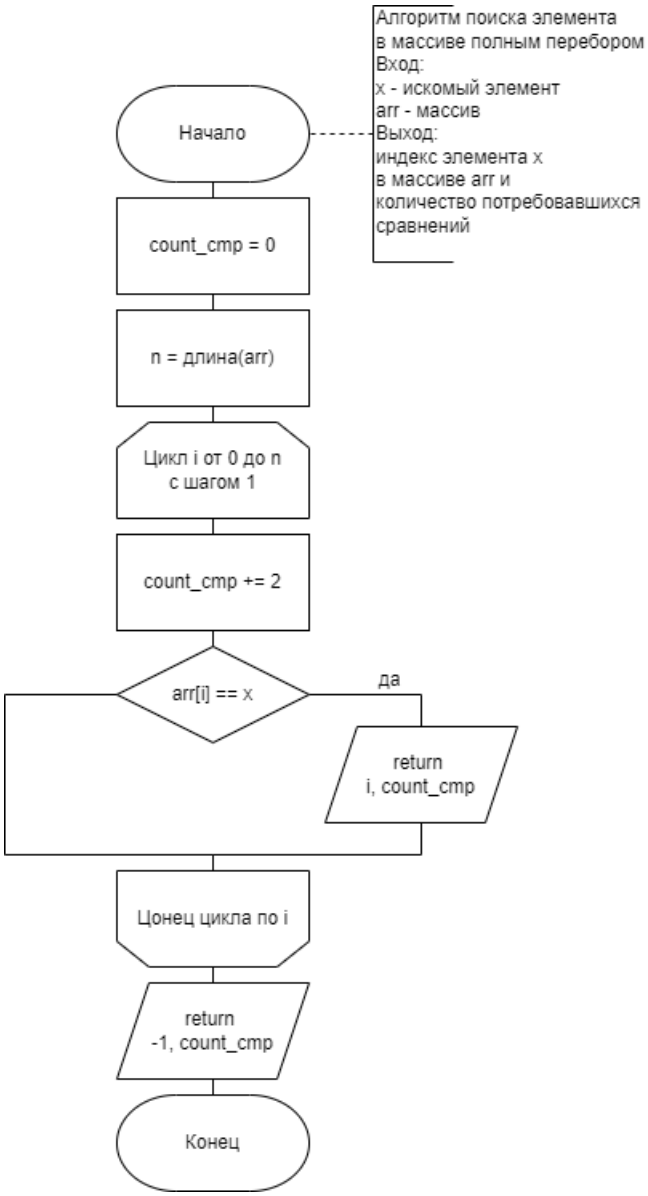

Рисунок 2.1 — Схема алгоритма поиска элемента в массиве полным перебором

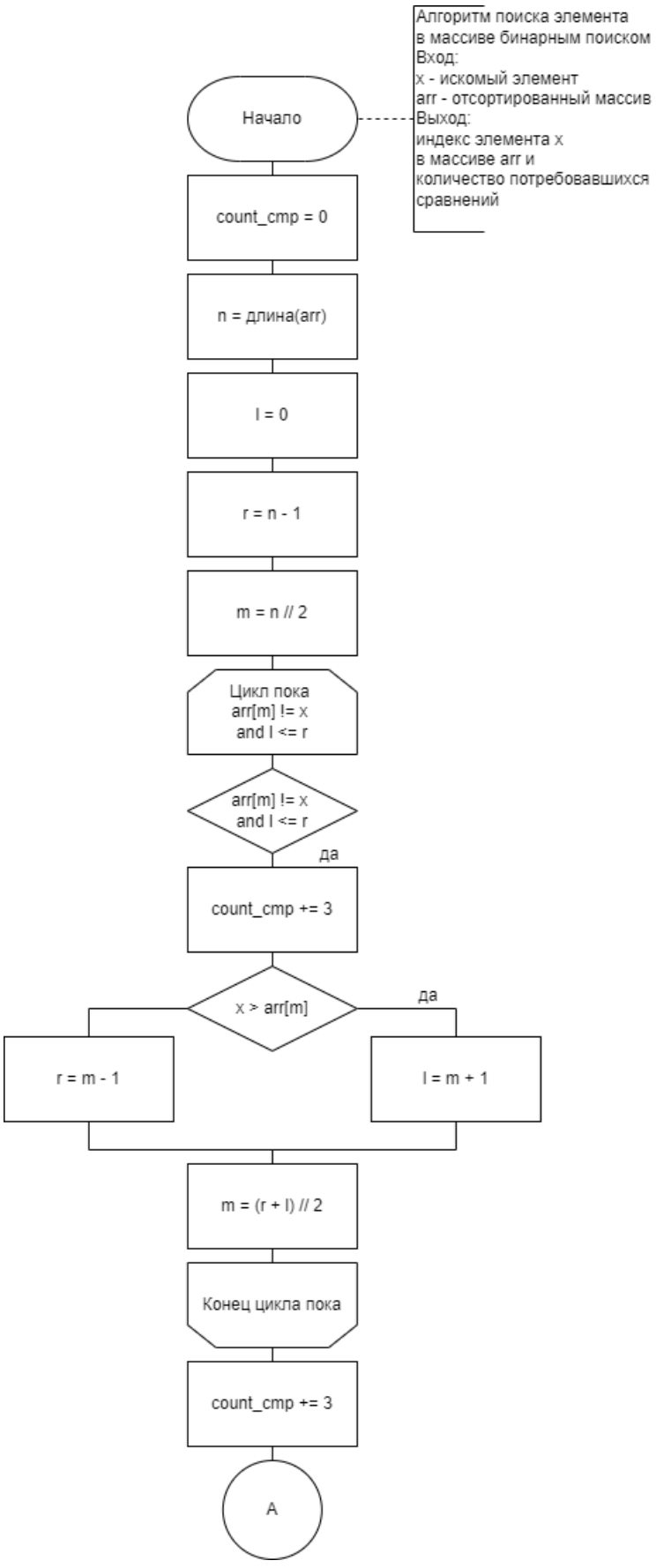

Рисунок 2.2 — Схема алгоритма поиска элемента в массиве бинарным поиском

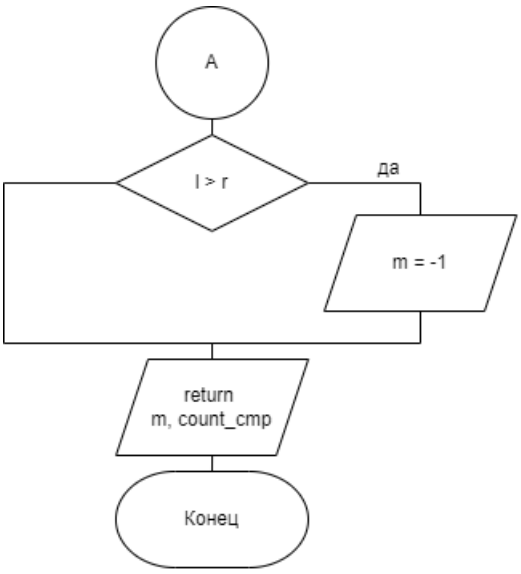

Рисунок 2.3 — Схема алгоритма поиска элемента в массиве бинарным поиском

Вывод

В данном разделе были представлены схемы алгоритмов поиска элемента в массиве полным перебором и бинарным поиском.

3 Технологическая часть

В данном разделе будут приведены средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $Python$ [1], так как он удовлетворяет требованиям лабраторной работы: поддерживает динамические структуры данных, такие как массивы и имеет библиотеку $Matplotlib$ [2] для построения графиков.

3.2 Реализация алгоритмов

В листингах 3.1-3.2 представлены реализации алгоритмов поиска элемента в массиве.

Листинг 3.1 – Алгоритм поиска элемента в массиве полным перебором

```
        def completeSearch(arr, x, printing=False):
            count_cmp = 0
            for i in range(len(arr)):
                count_cmp += 2
                if arr[i] == x:
                    return i, count_cmp
            return -1, count_cmp
```

Листинг 3.2 – Алгоритма поиска элемента в массиве бинарным поиском

```
    def binSearch(arr, x, printing=False):
        arr.sort()
        n = len(arr)
        l, r = 0, n - 1
        m = n // 2

        count_cmp = 0
        while arr[m] != x and l <= r:
            count_cmp += 3
            if x > arr[m]:
```

```
                l = m + 1
            else:
                r = m - 1
            m = (r + l) // 2
    count_cmp += 3
    if l > r:
        return -1, count_cmp
    else:
        return m, count_cmp
```

3.3 Классы эквивалентности тестирования

Для тестирования были выделены следующие классы тестирования:

1) Пустой массив;
2) Искомый элемент – первый;
3) Искомый элемент – последний;
4) Искомый элемент – в середине массива;
5) Искомого элемена – нет;

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов поиска элемента x в массиве $arr = [4, 5, 12, 13, 13, 18, 22, 26, 30, 35]$.

Все тесты пройдены успешно.

Таблица 3.1 — Функциональные тесты

			Результат	
№	x	Массив	Поный перебор	Бинарный поиск
1	4	"Пустой массив"	-1	-1
2	4	arr	0	0
3	5	arr	4	4
4	35	arr	9	9
5	18	arr	5	5

Вывод

Были представлены листинги всех описанных ренее алгоритмов поиска элемента в массиве и их тесты.

4 Исследовательская часть

Цель исследования – сравнительный анализ реализованных алгоритмов по трудоемкости.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система – Майкрософт Windows 11 Домашняя для одного языка; Версия – 10.0.22631; Сборка – 22631;

— Установленная оперативная память (RAM) – 16,0 ГБ;

— Процессор – AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16;

4.2 Время выполнения алгоритмов

Трудоемкость алгоритмов определялась в терминаз числа сравнений, которые понадобились на нахождение искомого элемента.

Согласно варианту, была посчитана длина массива по следующей формуле:

$$n = \begin{cases} X \mod 1000, \text{ если } \left(\frac{X}{4} \mod 10\right) = 0, \\ \left(\frac{X}{4} \mod 10\right) \times (X \mod 10) + \left(\frac{X}{2} \mod 10\right), \text{ иначе} \end{cases} \tag{4.1}$$

где X = 8119

Длина массив = 1014 элементов

Были полученны гистограммы 4.1-4.3 для алгоритмов поиска элемента в массиве

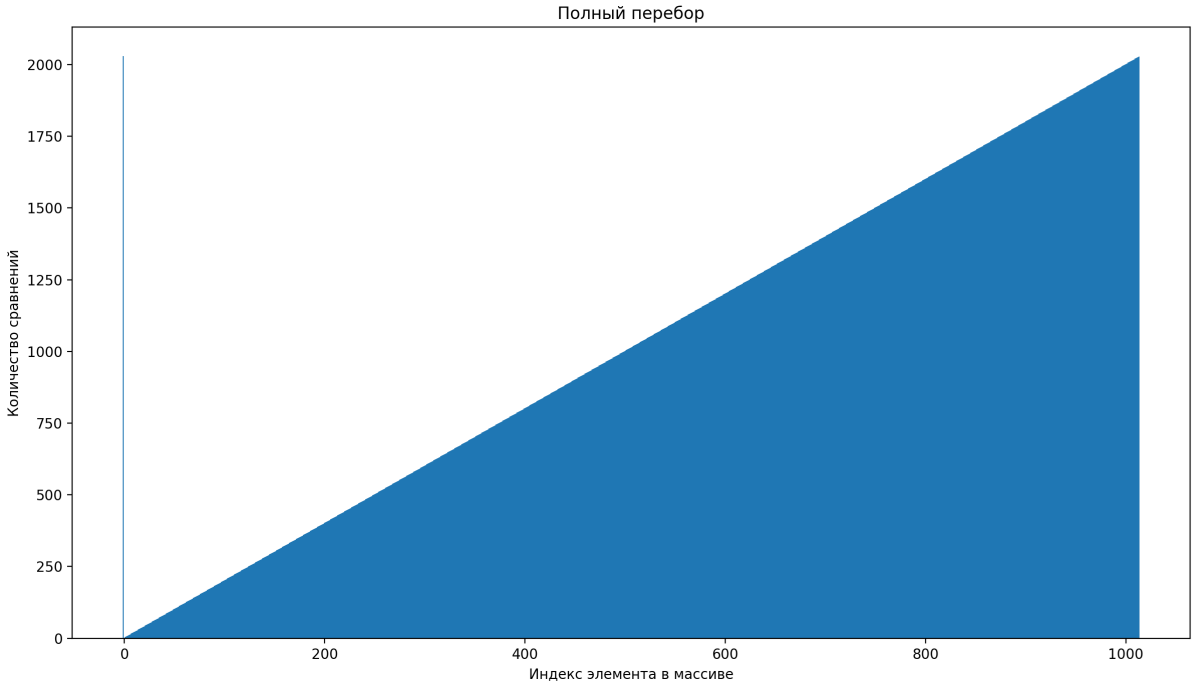

Рисунок 4.1 — Алгоритм поиска элемента в массиве полным перебором

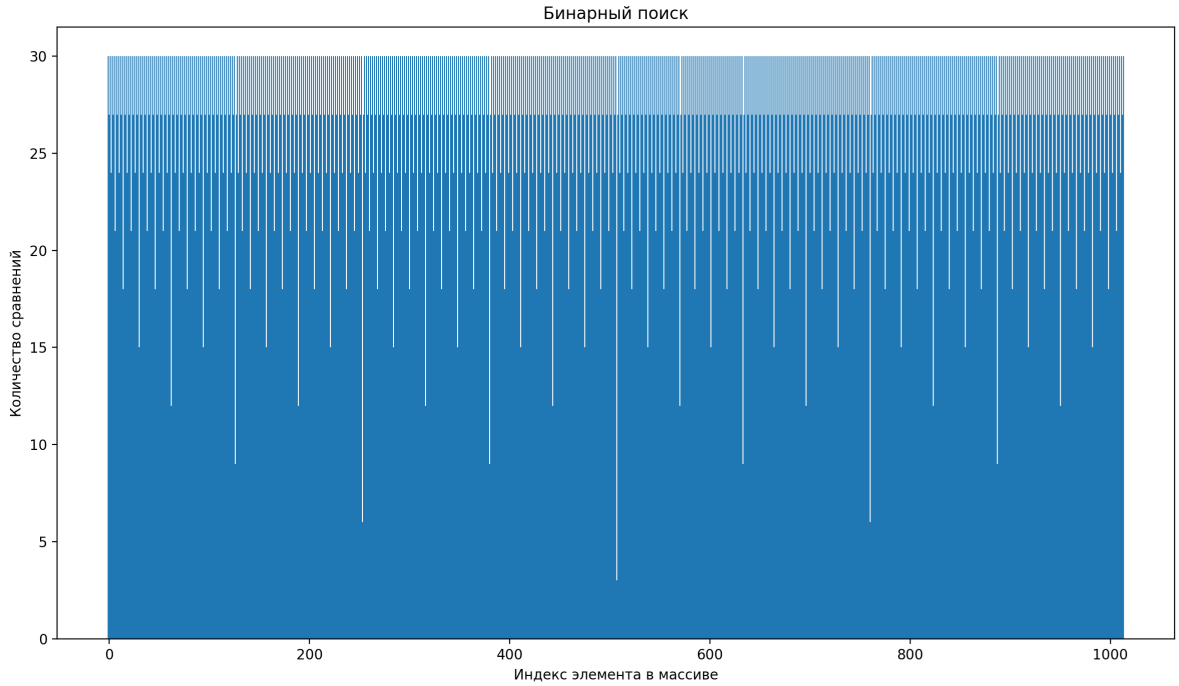

Рисунок 4.2 — Алгоритма поиска элемента в массиве бинарным поиском

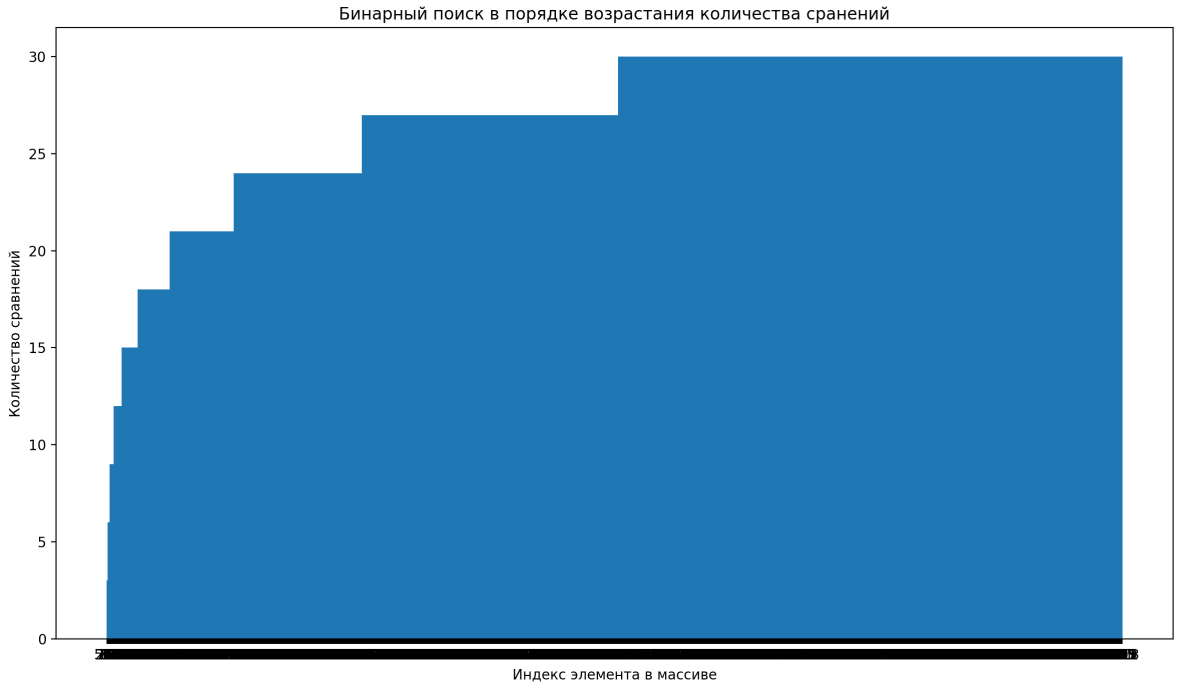

Рисунок 4.3 — Алгоритма поиска элемента в массиве бинарным поиском в порядке возрастания количества сранений

4.3 Вывод

Алгоритм бинарного поиска требует значительно меньше сравнений чем алгоритм использующий полный перебор. Для массива длиной 1014 максимальное количество сравнений в алгоритме бинарного поиска = 30, а в алгоритме полного перебора = 2000.

Но при этом в случаях когда искомых элемент находится в начале массива (индекс до 30) алгоритм полного перебора требует меньше сравнений.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы была выполнена поставленнная цель, которая заключалась в сравнительном анализе алгоритмов поиска элемента в массиве, используя полный перебор и бинарный поиск.

Были выполнены следующие задачи:

1) Изучить алгоритмы поиска элемента в массиве, используя полный перебор и бинарный поиск;
2) Реализовать алгоритмы:
   - поиска элемента в массиве полным перебором;
   - поиска элемента в массиве бинарным поиском;
3) Провести сравнительный анализ по трудоемкости работы алгоритмов;

Основываясь на проведенном исследовании можно сделать следующий вывод. Алгоритм бинарного поиска треубет значительно меньше сравнений чем алгоритм использующий полный перебор. Для массива длиной 1014 максимальное количество сравнений в алгоритме бинарного поиска = 30, а в аглритме полного перебора = 2000.

Но при этом в случаях когда искомых элемент находится в начале массива алгоритм полного перебора требует меньше сравнений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org`. (дата обращения: 30.09.2024).

2. Matplotlib 3.9.2 documentation [Электронный ресурс]. Режим доступа: `https://matplotlib.org/stable/index.html`. (дата обращения: 30.09.2024).