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 в массиве

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

Преподаватели Волкова Л.Л., Строганов Д.М.

Москва, 2024

Содержание

ВВЕДЕНИЕ

Поиск в массиве – одна из частых задач, встречающихся в программировании. Существует два основных подхода к её решению – поиск полным перебором и двоичный(бинарный) поиск.

Целью данной работы является исследование трудоёмкости алгоритмов поиска в массиве.

Для достижения этой цели требуется решить следующие задачи:

— Рассмотрение алгоритмов бинарного поиска и поиска полным перебором в массиве;

— Разработка алгоритмов бинарного поиска и поиска полным перебором в массиве;

— Реализация разработанных алгоритмов;

— Сравнительный анализ реализаций по трудоёмкости.

1 Аналитическая часть

1.1 Поиск полным перебором

Линейный поиск(полным перебором) – алгоритм поиска в массиве, при котором массив последовательно просматривает поэлементно. Поиск прекращается при одном из двух условий:

1) Искомый элемент найден по i-му индексу;
2) Просмотрен весь массив и искомый элемент не найден.

Линейный поиск не накладывает никаких дополнительных ограничений на массив, а значит универсален, так как можно применить к любому массиву,но он имеет сложность O(N) [1], что является не самым эффективным решением.

1.2 Бинарный поиск

Бинарный поиск(или поиск делением пополам) – более эффективный алгоритм для поиска элемента в отсортированном массиве. Ключевой идеей алгоритма является делением массива пополам на каждой итерации. Для этого берётся середина текущего массива и сравнивается с искомым элементом. Если искомый больше, то он точно правее текущего, иначе левее. За счёт этого получается на каждой итерации откидывать половину оставшегося массива.

Такой алгоритм имеет сложность O(logN) [1], что уже лучше, чем линейный поиск, но такой алгоритм возможно применять только к отсортированным массивам, поэтому он универсальный.

Вывод

В результате аналитического раздела были рассмотрены алгоритмы линейного поиска и бинарного поиска в массивах.

2 Конструкторская часть

2.1 Требования к программному обеспечению

К разрабатываемой программе предъявлен ряд требований:

Входные данные: Массив целых чисел, искомое целое число.

Выходные данные: Индекс искомого числа в массиве.

— Индексация элементов в массиве начинается с 0;

— В случае, если искомого элемента в массиве нет, вместо индекса должно возвращаться значение -1.

2.2 Вариации бинарного поиска

Алгоритм бинарного поиска может быть описан двумя способами – итерационно и рекурсивно.

При итерационной реализации выделяют две переменные – левую и правую границы, в цикле считается средний элемент по границам и в зависимости от него эти границы меняются.

В рекурсивной реализации выбирается серединный элемент по всему массиву, и в зависимости от его значения рекурсивно вызывается поиск либо от правой части массива, либо от левой.

Для данной работы была выбрана итерационная вариация алгоритма, так как в общем случае рекурсивный вызов – трудоёмкая операция, при этом нет сложностей в реализации итерационной вариации.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема разработанного алгоритма линейного поиска. На рисунке 2.2 представлена схема итерационного алгоритма бинарного поиска.

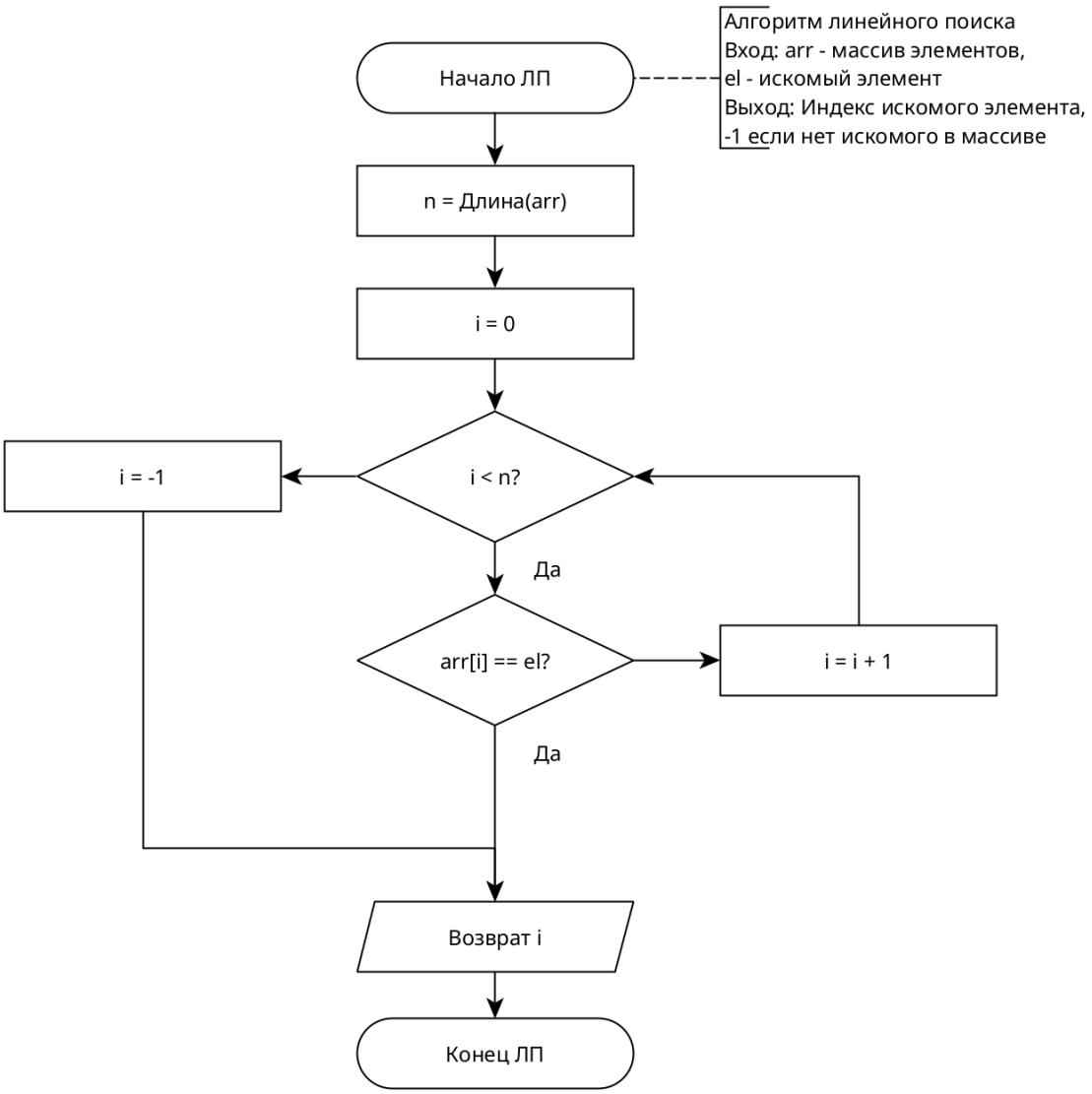

Рисунок 2.1 — Схема алгоритма линейного поиска

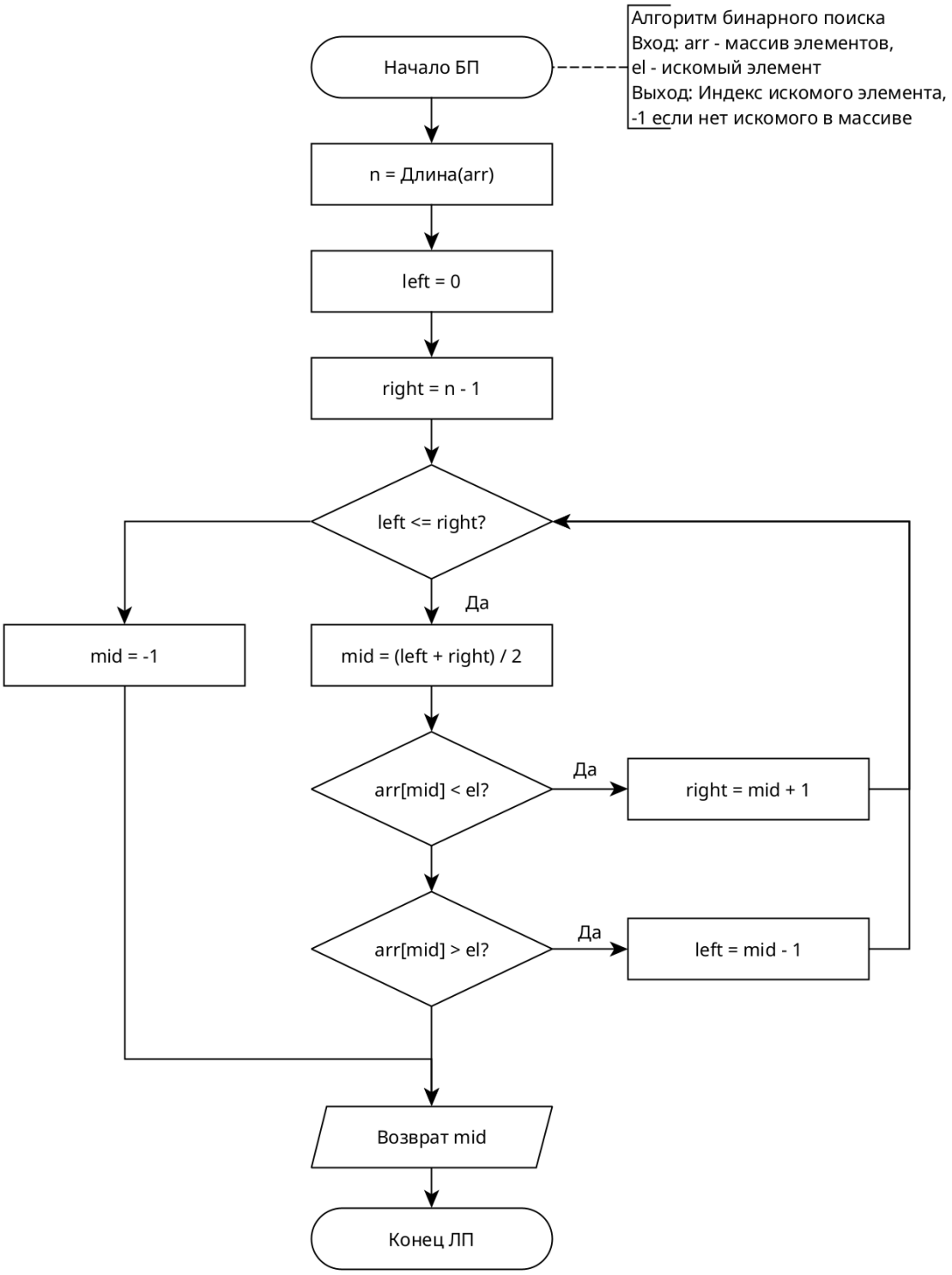

Рисунок 2.2 — Схема алгоритма итерационного бинарного поиска

2.4 Вывод

В результате конструкторской части были определены требования к ПО, а также разработаны схемы алгоритмов линейного поиска и бинарного поиска.

3 Технологическая часть

3.1 Средства разработки

В качестве языка программирования был выбран python3 [2], так как данный язык обладает множеством инструментов для визуализации данных и работы с ними.

Для основного файла был выбран инструмент jupyter notebook [3], так как он позволяет организовать код в удобные блоки, а также выводить данные и графики прямо в нём, что позволяет легко продемонстрировать все исследования.

Для построения графиков использовалась библиотека plotly [4].

3.2 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма линейного поиска, а в листинге 3.2 – реализация бинарного поиска. В каждый из этих алгоритмов был добавлен счётчик сравнений в теле цикла, при этом в линейном поиске это счётчик увеличивает на 1 за итерацию, а в бинарном поиске может увеличиться на 1 или 2, в зависимости от отношения текущего элемента с искомым.

Листинг 3.1 — Алгоритм линейного поиска

```
def SimpleSearch(array: list[int], element: int) -> tuple[int, int]:
  comparisonCount = 0
  for i in range(len(array)):
    comparisonCount += 1
    if array[i] == element:
      return i, comparisonCount
return -1, comparisonCount
```

Листинг 3.2 — Алгоритм бинарного поиска

```
def BinarySearch(array: list[int], element: int) -> tuple[int, int]:
  left = 0
  right = len(array) - 1
  comparisonCount = 0

  while left <= right:
    mid = (left + right) // 2

    if array[mid] < element:
      comparisonCount += 1
      left = mid + 1
    elif array[mid] > element:
      right = mid - 1
```

```
        comparisonCount += 2
      else:
        comparisonCount += 2
        return mid, comparisonCount

  return -1, comparisonCount
```

3.3 Функциональные тесты

При тестирования было выделено три вида тестов:

1) Массив пустой;

2) Искомого элемента в массиве нет;

3) Искомы элемент есть в массиве.

Для тестирования пунктов 1 и 2 были написаны отдельные тесты, а для пункта 3 использовалась следующая система: создавался массив целых чисел размером 1000 элементов с элемента $1, \ldots, 1000$ в порядке возрастания. Затем создавалась копия этого массива и перемешивалась случайным образом. Затем каждый элемент из перемешанной копии искался в исходном массиве линейным поиском и бинарным поиском, при этом значения функций сравнивались с методом list.index [5] из стандартной библиотеки python3, который ищет индекс элемента в массиве.

Для перемешивания массива была использована функция shuffle из стандартного модуля random [6].

Код тестирования приведён на листинге 3.3.

Листинг 3.3 — Тестирование функций поиска

```
testSize = 1000

array = [i for i in range(1, testSize + 1)]

# Поиск в пустом массиве
if SimpleSearch([], 2)[0] != -1:
  print("Error on empty array with SimpleSearch!")
  exit(1)

if BinarySearch([], 2)[0] != -1:
  print("Error on empty array with BinarySearch!")
  exit(1)

print("Empty array test passed!")

# Поиск несуществующего элемента
```

```
if SimpleSearch(array, testSize + 1000)[0] != -1:
  print("Error on not existing element with SimpleSearch!")
  exit(1)

if BinarySearch([], testSize + 1000)[0] != -1:
  print("Error on not existing element with BinarySearch!")
  exit(1)

print("Not existing element test passed!")

# Поиск всех элементов массива
arrcp = array.copy()
shuffle(arrcp)
for el in arrcp:
  index, _ = SimpleSearch(array, el)
    if index != array.index(el):
    print(f"Error on element {el} with SimpleSearch!")
    break
  index, _ = BinarySearch(array, el)
  if index != array.index(el):
    print(f"Error on element {el} with BinarySearch!")
    break
else:
  print("All positive tests passed!")
```

Все тесты пройдены успешно.

Вывод

В ходе технологической части работы были разработаны алгоритмы линейного поиска и бинарного поиска на языке python3, а также проведено их функциональное тестирование.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства,на котором проводились исследования:

— Процессор: AMD Ryzen 7 5800H (16) @ 4.46 GHz;

— Оперативная память: 16 Г;

— Операционная система: Arch Linux x86_64.

4.2 Расчёт размера массива

Расчёт размера массива по варианту производится по формуле:

$$N = \frac{X}{8} + \begin{cases} X\%1000, & \text{если } \frac{X}{4}\%10 == 0, \\ (\frac{X}{4}\%10) * (X\%10) + (\frac{X}{2}\%10), & \text{иначе,} \end{cases} \tag{4.1}$$

где $X = 8117$ – номер задачи в redmine.

Для моего варианта $N = 1085$

4.3 Трудоёмкость алгоритмов

Для полученного размера было проведено исследование количества операций сравнения в телах циклов в зависимости от индекса искомого элемента в массиве. Результаты приведены в виде гистограмм, для линейного поиска рисунок 4.1, для бинарного поиска рисунки 4.2 и 4.3. На последнем рисунке столбцы отсортированы по высоте. На всех графиках индекс -1 означает ситуацию, когда искомого элемента в массиве нет.

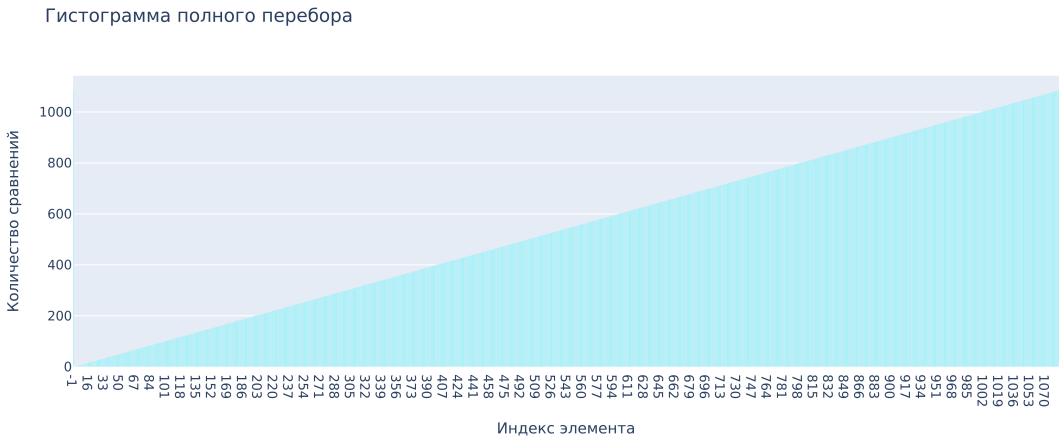

Рисунок 4.1 — Гистограмма зависимости количества сравнений от индекса элемента в линейном поиске

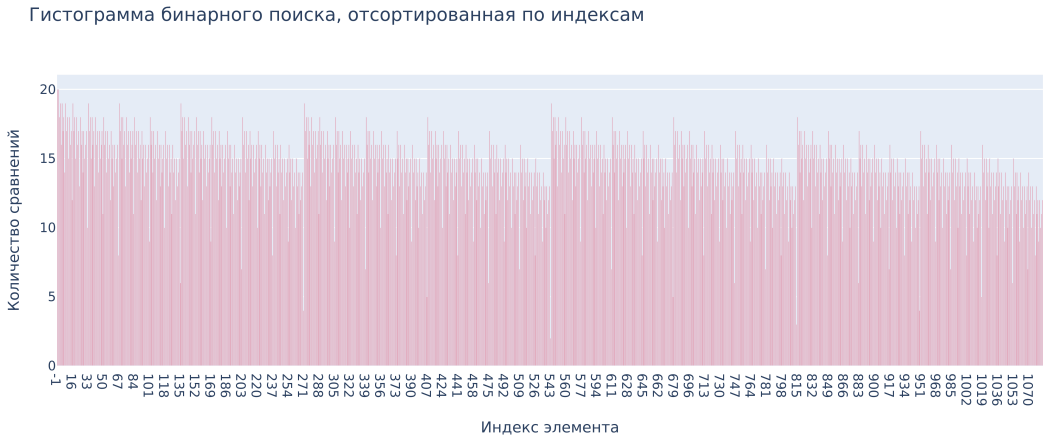

Рисунок 4.2 — Гистограмма зависимости количества сравнений от индекса элемента в бинарном поиске

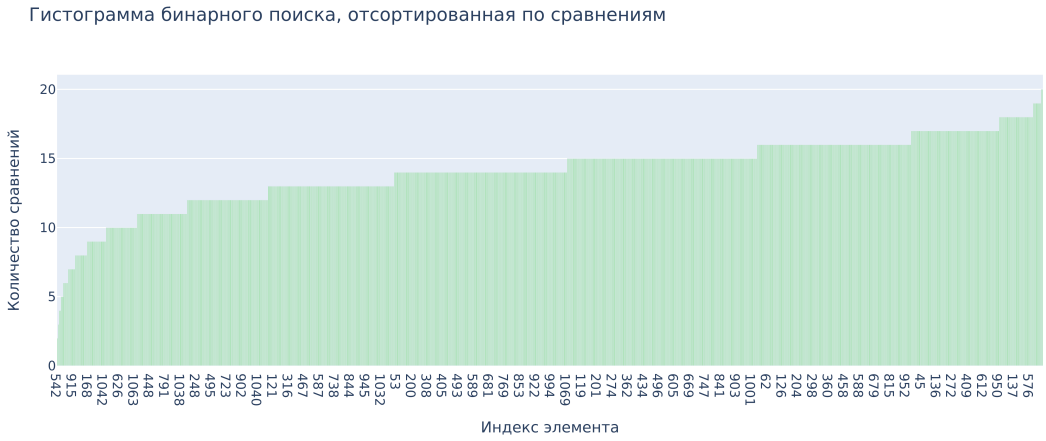

Рисунок 4.3 — Гистограмма зависимости количества сравнений от индекса элемента в бинарном поиске, отсортировано по количеству сравнений

4.4 Вывод

В данном разделе было проведено исследование трудоёмкости алгоритмов поиска в массиве.

Как видно из рисунков 4.1–4.3, линейный поиск существенно уступает бинарному по количеству операций сравнения. При длине массива 1085 элементов максимальное число сравнений для линейного поиска – 1085, а для бинарного поиска – 20.

Также, как можно отметить на рисунке 4.3, кривая трудоёмкости бинарного поиска имеет форму логарифмической прямой, что подтверждает его логарифмическую сложность O(logN).

ЗАКЛЮЧЕНИЕ

В результате исследования было выявлено, что бинарный поиск работает на несколько порядков быстрее, а также имеет трудоёмкость, кривая которой имеет логарифмическую форму.

Цель и все задачи лабораторной работы были выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Никлаус Вирт Алгоритмы и структуры данных. Новая версия для Оберона. [Текст] / Никлаус Вирт. — М.: ДМК Пресс, 2016 — 272 с.

2. Python / [Электронный ресурс] // Python : [сайт]. — URL: https://www.python.org/ (дата обращения: 20.09.2024).

3. Jupyter Notebook: The Classic Notebook Interface / [Электронный ресурс] // Jupyter : [сайт]. — URL: https://jupyter.org/ (дата обращения: 20.09.2024).

4. Plotly Open Source Graphing Library for Python / [Электронный ресурс] // Plotly : [сайт]. — URL: https://plotly.com/python/ (дата обращения: 20.09.2024).

5. Data structures — more on lists / [Электронный ресурс] // Python3 docs : [сайт]. — URL: https://docs.python.org/3/tutorial/datastructures.html#more-on-lists (дата обращения: 01.10.2024).

6. random — Functions for sequences / [Электронный ресурс] // Python3 docs : [сайт]. — URL: https://docs.python.org/3/library/random.html#functions-for-sequences (дата обращения: 01.10.2024).