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 №7 по курсу "Анализ алгоритмов"

Тема Поиск в словаре

Студент Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Строганов Д.В.

Москва — 2024 г.

Оглавление

Введение

Словарь – структура данных, построенная на основе пар значений. Первое значение пары – ключ для идентификации элементов, второе – собственно сам хранимый элемент. Например, в телефонном справочнике номеру телефона соответствует фамилия абонента. Существует несколько основных различных реализаций словаря: массив, двоичные деревья поиска, хеш-таблицы. Каждая из этих реализаций имеет свои минусы и плюсы, например время поиска, вставки и удаления элементов.

Со временем были разработаны различные алгоритмы поиска в словаре. В данной лабораторной работе будут рассмотрены три из них:

1. поиск в АВЛ–дереве;
2. бинарный поиск;

Целью данной лабораторной работы является изучение и реализация алгоритмов поиска в словаре.

Для достижения поставленной цели необходимо выполнить следующие **задачи**.

1. Изучение основ поиска значения в словаре.
2. Исследование подходов к реализации двух алгоритмов поиска в словаре: поиск в АВЛ–дереве, бинарный поиск.
3. Построение схем разработанных алгоритмов.
4. Описание используемых структур данных.
5. Определение средств программной реализации.
6. Реализация разработанных алгоритмов поиска в словаре.
7. Проведение тестирования реализованных алгоритмов.
8. Проведение сравнительного анализа времени выполнения двух алгоритмов поиска в словаре на основе экспериментальных данных.

9. Проведение сравнительного анализа количества сравнений в худших и лучших случаях работы алгоритмов.

1 Аналитическая часть

В данном разделе будут представлены теоретические сведения о рассматриваемых алгоритов поиска в словаре.

Словари – объекты, записанные парой "ключ-значение". Отличным примером словаря является толковый словарь. В данном словаре ключи – это нужное нам слово, а значения – это значение искомого слова. Ключи в словаре должны быть уникальными. Поиск необходимой информации в словаре – одна из фундаментальных задач программирования.

1.1 Бинарный поиск

Данный алгоритм базируется на том, что словарь изначально упорядочен, что позволяет сравнивать ключ с средним элементом, и, если, он меньше, то продолжать искать в левой части, таким же методом, иначе - в правой.

Алгоритм бинарного поиска можно описать следующим образом:

1. получить значение находящееся в середине словаря и сравнить его с ключом;
2. в случае, если ключ меньше данного значения, продолжить поиск в младшей части словаря, в обратном случае – в старшей части словаря;
3. на новом интервале снова получить значение из середины этого интервала и сравнить с ключом.
4. поиск продолжать до тех пор, пока не будет найден искомый ключ, или интервал поиска не окажется пустым.

Обход словаря данным алгоритм можно представить в виде дерева, поэтому трудоемкость в худшем случае составит $\log_2 N$.

1.2 Поиск в АВЛ–Дереве

АВЛ–дерево – это прежде всего двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Это значит, что для поиска нужного ключа в АВЛ-дереве можно использовать стандартный алгоритм.

Особенностью АВЛ–дерева является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.

Алгоритм поиска в АВЛ–дереве состоит из следующих шагов:

1. Начинаем с корня дерева.
2. . Если ключ, который мы ищем, равен ключу текущего узла, то мы нашли элемент и возвращаем его.
3. Если ключ меньше ключа текущего узла, то мы переходим к левому поддереву и повторяем шаги 2-3 для этого поддерева.
4. Если ключ больше ключа текущего узла, то мы переходим к правому поддереву и повторяем шаги 2-3 для этого поддерева.
5. Если мы дошли до конца дерева и не нашли элемент, то элемент не существует в дереве.

Вывод

В данном разделе были рассмотрены алгоритмы поиска в словаре.

2 Конструкторская часть

В данном разделе приводится схема рассматриваемых алгоритмов, определяются требования к программе.

2.1 Требования к программе

На основе изученной информации можно описать входные и выходные данные. На вход программе должно подаваться число для поиска, массив. На выходе программа должна выдавать индекс найденного числа либо сообщить о его отсутствии.

2.2 Разработка алгоритмов

На рисунке 2.1 приведена схема бинарного поиска в массиве. На рисунке 2.2 приведена схема поиска в AVL-дереве

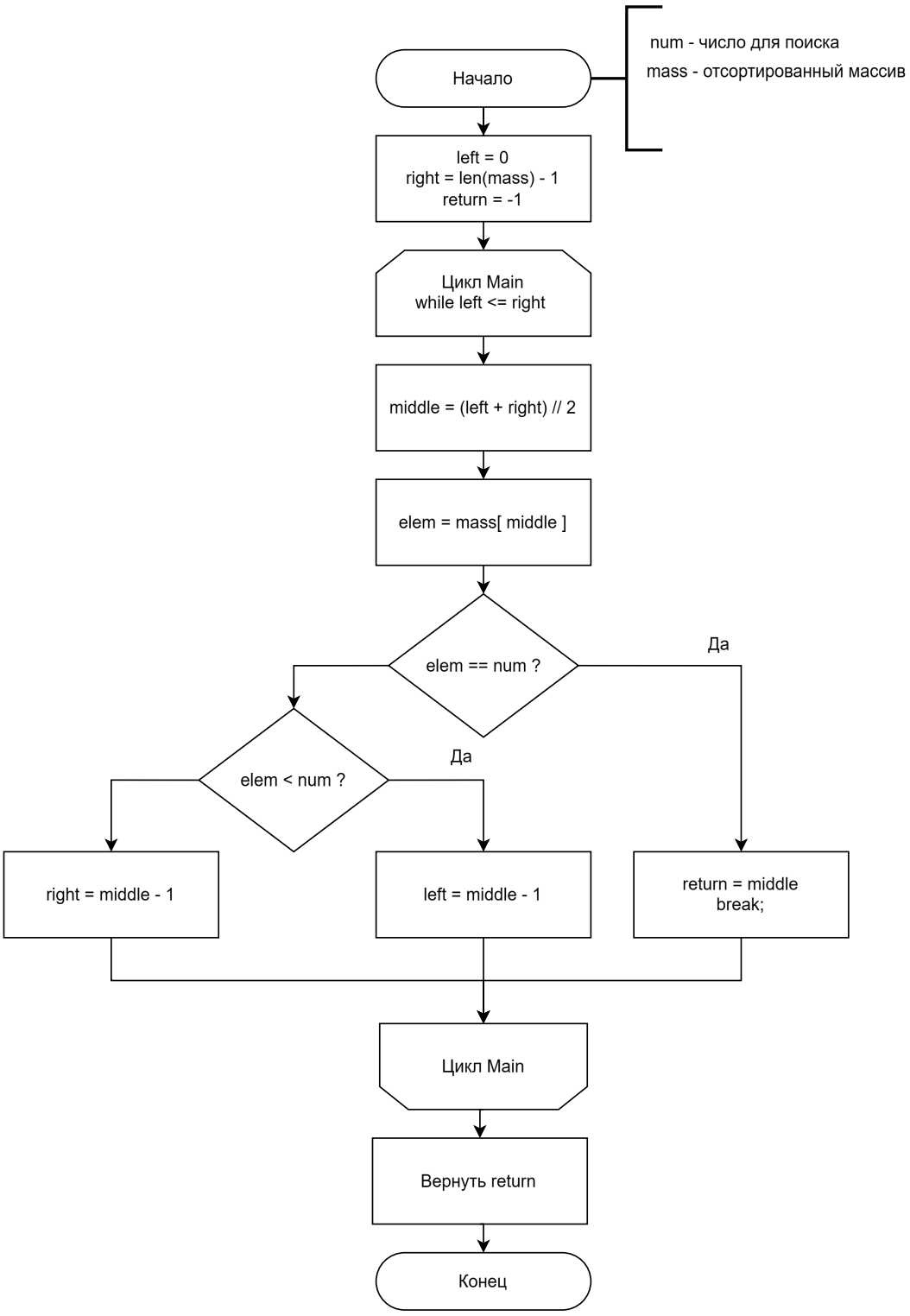

Рисунок 2.1 – Схема бинарного поиска в массиве

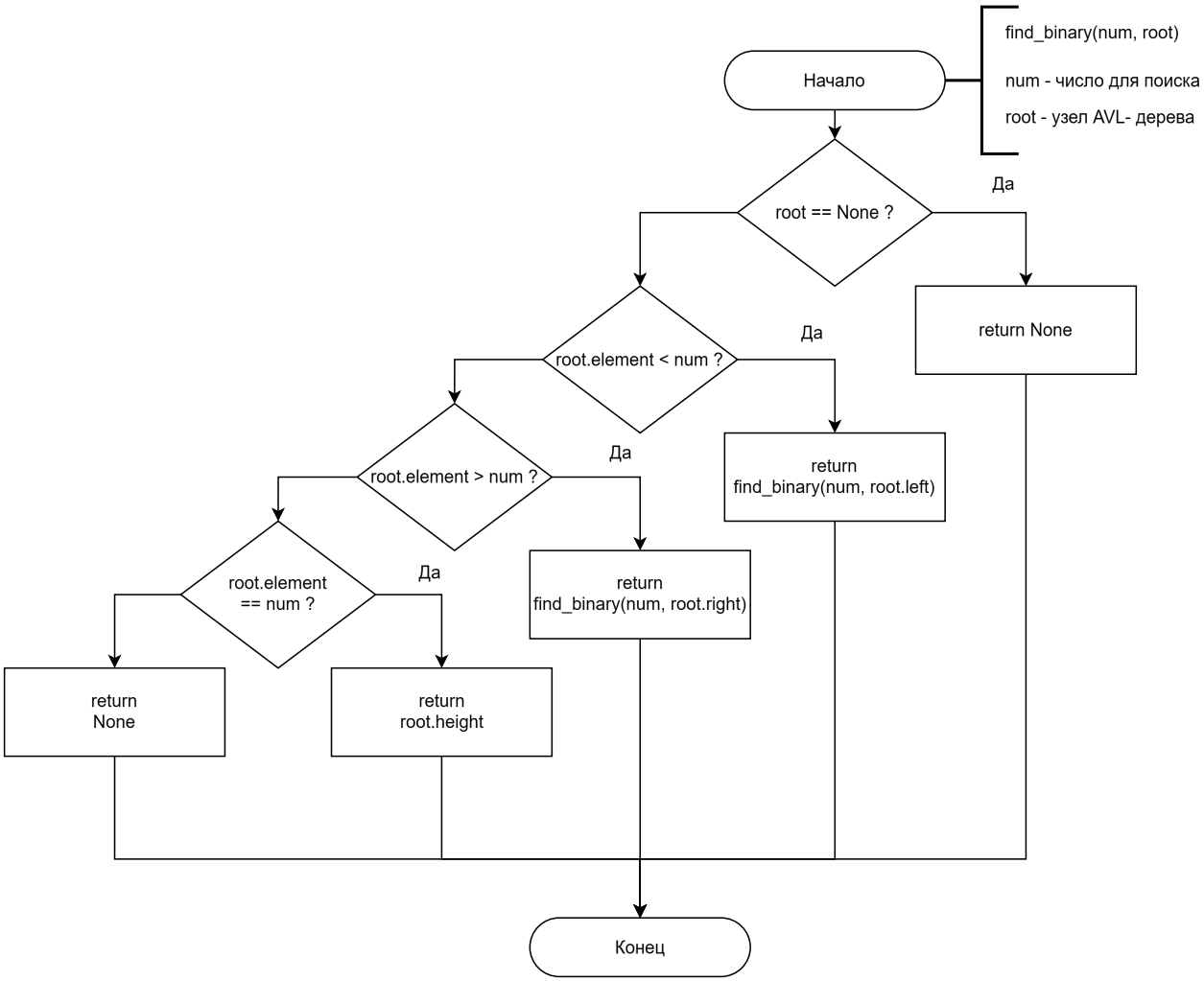

Рисунок 2.2 – Схема поиска в AVL-дереве

Вывод

В данном разделе была рассмотрена схема бинарного поиска в массиве, поиска в AVL-дереве, были приведены требования к программе.

3 Технологическая часть

В данном разделе будут приведены средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python. Выбор обусловлен наличием библиотек для измерения времени, наличием инструментов для работы с массивами.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей: main.py - главный файл программы, в котором располагается вся программа.

3.3 Реализация алгоритмов

В листингах 3.1, 3.2 приведены реализации алгоритмов.

Листинг 3.1 – Поиск в AVL-дереве.

```
def _find(self, key, node):
    if not node:
        return None
    elif key < node.value:
        return self._find(key, node.l)
    elif key > node.value:
        return self._find(key, node.r)
    elif key == node.value:
        return node
    else:
        return None
```

Листинг 3.2 – Бинарный поиск.

```
def BinarySearch(key, list_keys):
    l, r = 0, len(list_keys) - 1
    k = 0
    while l <= r:
        middle = (r + l) // 2
        elem = list_keys[middle]
        k += 1
        if elem == key:
            return k
        elif elem < key:
            l = middle + 1
        elif elem > key:
            r = middle - 1
        else:
            break


    return -1
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов.

Таблица 3.1 – Тестирование функций

Массив	Число для поиска	Ожидаемый результат
$2, 3, 4, 7, 9, 10, 12$	3	1
$2, 3, 4, 7, 9, 10, 12$	9	4
$2, 3, 4, 7, 9, 10, 12$	13	-1
$2, 3, 4, 7, 9, 10, 12$	8	-1

Алгоритмы прошли проверку.

Вывод

Были разработан и протестирован алгоритм поиска в AVL–дереве, алгоритм бинарного поиска.

4 Исследовательская часть

В данном разделе будет приведена постановка замера времени и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замер времени:

Операционная система: 64-разрядная операционная система, процессор x64.

Память: 16 Гб.

Процессор: Intel(R) Core(TM) i7–4700HQ CPU @ 2.40 ГГц.

Во время замера времени ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Алгоритмы замерялись при помощи функции $time.perf\_counter()$ из библиотеки *time* языка *Python*. Данная функция возвращает значение в долях секунды счетчика производительности, то есть часов с наибольшим доступным разрешением для измерения короткой длительности.

Замеры времени для каждого выполнения проводились 100000 раз. В качестве результата взято среднее время работы алгоритма.

Из таблицы 4.1 можно сделать вывод, что бинарный поиск затрачивает меньше времени в 500 раз. Поиск по AVL-дереву и бинарный поиск требуют подготовки данных. Для первого алгоритма должно быть построено бинарное сбалансированное дерево, для бинарного поиска массив должен быть изначально отсортирован.

Лучшим случаем при нахождении элемента является его нахождение в корне AVL–дерева, или же в середине массива для бинарного поиска.

Таблица 4.1 – Замеры времени при нахождении элемента в массиве

Кол-во элементов	AVL-дерево(ns)	Бинарный поиск(ns)
128	952.03200	2.43218
256	1020.54900	2.67411
512	1464.10500	2.99134
1024	1673.71200	3.51576
2048	1743.52000	3.62115

Таблица 4.2 – Замеры времени при отсутствии элемента в массиве

Кол-во элементов	AVL-дерево(ns)	Бинарный поиск(ns)
128	763.34930	2.32894
256	1007.96921	3.56399
512	1686.90563	3.95372
1024	1950.73587	5.04567
2048	2137.42351	6.51642

Лучшим случаем при отсутствии элемента является выявление его отсутствия при втором сравнении элементов во время выполнения алгоритма.

Худшим случаем при нахождении элемента является его нахождение на максимальной высоте AVL–дерева или на грани массива. Худшим случаем при отсутствии элемента является прохождение всей ветви AVL–дерева или же всех сравнений при бинарном поиске т.е. элемент должен быть за гранью разброса величин, находящихся в массиве.

При лучших случаях с нахождением элемента, логично, будет произведено только одно сравнение. При лучших случаях с отсутствием элемента

будет произведено 5 сравнений.

Таблица 4.3 – Количество сравнений элементов худший случай при нахождении элемента

Кол-во элементов	AVL-дерево	Бинарный поиск
128	22	19
256	25	22
512	28	25
1024	31	28
2048	34	31

Таблица 4.4 – Количество сравнений элементов худший случай при отсутствии элемента

Кол-во элементов	AVL-дерево	Бинарный поиск
128	24	21
256	27	24
512	30	27
1024	33	30
2048	36	33

Из таблиц 4.3, 4.4 видно, что количество сравнений при отсутствии элемента в массиве больше. Также можно заметить, что количество сравнений при поиске в AVL–дереве больше, чем при бинарном поиске.

Вывод

Из проведенного анализа можно сделать следующие выводы: количество сравнений при отсутствии элемента в массиве больше на 2. Количество сравнений при поиске в AVL–дереве больше, чем при бинарном поиске.

Следовательно наихудший вариант развития событий – отсутствие элемента в массиве, при этом элемент должен быть вне диапазона чисел, присутствующих в массиве.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

1. изучены основ поиска значения в словаре;
2. исследованы подходы к реализации алгоритмов поиска в АВЛ–дереве, бинарный поиск;
3. построены схем разработанных алгоритмов;
4. определены средства программной реализации;
5. реализованы алгоритмы поиска в словаре;
6. проведены тестирования реализованных алгоритмов;
7. проведены сравнительные анализы времени выполнения двух алгоритмов поиска в словаре на основе экспериментальных данных;
8. проведен сравнительный анализ количества сравнений в худших и лучших случаях работы алгоритмов.

Из таблицы 4.1 можно сделать вывод, что бинарный поиск затрачивает меньше времени в 500 раз. Поиск по AVL-дереву и бинарный поиск требуют подготовки данных. Для первого алгоритма должно быть построено бинарное сбалансированное дерево, для бинарного поиска массив должен быть изначально отсортирован.

Лучшим случаем при нахождении элемента является его нахождение в корне AVL–дерева, или же в середине массива для бинарного поиска. Лучшим случаем при отсутствии элемента является выявление его отсутствия при втором сравнении элементов во время выполнения алгоритма.

Худшим случаем при нахождении элемента является его нахождение на максимальной высоте AVL–дерева или на грани массива. Худшим случаем при отсутствии элемента является прохождение всей ветви AVL–дерева или же всех сравнений при бинарном поиске т.е. элемент должен быть за гранью разброса величин, находящихся в массиве.

При лучших случаях с нахождением элемента, логично, будет произведено только одно сравнение. При лучших случаях с отсутствием элемента будет произведено 5 сравнений.

Из проведенного анализа можно сделать следующие выводы: количество сравнений при отсутствии элемента в массиве больше на 2. Количество сравнений при поиске в AVL–дереве больше, чем при бинарном поиске.

Следовательно наихудший вариант развития событий – отсутствие элемента в массиве, при этом элемент должен быть вне диапазона чисел, присутствующих в массиве.

Список использованных источников

[1] Волосова А. В. Параллельные методы и алгоритмы. – Москва: МАДИ, 2020.

[2] МакКоннелл Дж. Основы современных алгоритмов. - М.: Техносфера, 2004. - 368 с.