ГУАП

КАФЕДРА № 41

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Н.А. Соловьева |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| АЛГОРИТМЫ ПОИСКА |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4314 |  |  |  | Д. М. Развеев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы:**

Научиться реализовывать алгоритмы поиска.

1. **Вариант 18 (задания 4 и 13)**:

*Задание 4*: Реализуйте структуру данных «Массив», элементами которого выступают экземпляры класса Book (минимум 10 элементов), содержащие следующие поля (автор, издательство, кол-во страниц, стоимость, ISBN). Добавьте метод для любой метод сортировки и метод интерполяционного поиска по полю «стоимость». При вызове поиска убедитесь, что элементы структуры данных отсортированы, в ином случае – выбросите исключение.

*Задание 13*: Реализуйте структуру данных «Массив», элементами которого выступают экземпляры класса Book (минимум 10 элементов), содержащие следующие поля (автор, издательство, кол-во страниц, стоимость, ISBN). Добавьте метод для любой метод сортировки и метод бинарного поиска по полю «ISBN». При вызове поиска убедитесь, что элементы структуры данных отсортированы, в ином случае – выбросите исключение.

1. **Ход работы**

*Задание 4*

Программа реализации:

from typing import List, Optional

class Book:

    def \_\_init\_\_(self, author: str, publisher: str, pages: int, price: float, isbn: str):

        self.author: str = author

        self.publisher: str = publisher

        self.pages: int = pages

        self.price: float = price

        self.isbn: str = isbn

    def \_\_repr\_\_(self) -> str:

        return f"{self.author}, {self.publisher}, {self.price} руб."

class Array:

    def \_\_init\_\_(self):

        self.books: List[Book] = []

    def add\_book(self, book: Book) -> None:

        self.books.append(book)

    def cocktail\_sort(self) -> None:

        left: int = 0

        right: int = len(self.books) - 1

        while left < right:

            for i in range(left, right):

                if self.books[i].price > self.books[i + 1].price:

                    self.books[i], self.books[i + 1] = self.books[i + 1], self.books[i]

            right -= 1

            for i in range(right, left, -1):

                if self.books[i - 1].price > self.books[i].price:

                    self.books[i], self.books[i - 1] = self.books[i - 1], self.books[i]

            left += 1

    def is\_sorted(self) -> bool:

        return all(self.books[i].price <= self.books[i + 1].price for i in range(len(self.books) - 1))

    def interpolation\_search(self, target\_price: float) -> Optional[int]:

        if not self.is\_sorted():

            raise Exception("Элементы массива не отсортированы по цене.")

        low: int = 0

        high: int = len(self.books) - 1

        while low <= high and self.books[low].price <= target\_price <= self.books[high].price:

            if self.books[low].price == self.books[high].price:

                break

            pos: int = low + (target\_price - self.books[low].price) \* (high - low) // (self.books[high].price - self.books[low].price)

            if pos < 0 or pos >= len(self.books):

                break

            if self.books[pos].price == target\_price:

                return pos

            elif self.books[pos].price < target\_price:

                low = pos + 1

            else:

                high = pos - 1

        return None

**Описание кода**

* Данный код реализует структуру данных Array, которая представляет собой массив объектов класса Book. Каждый объект книги содержит информацию о:
* Авторе (author, тип: str)
* Издательстве (publisher, тип: str)
* Количестве страниц (pages, тип: int)
* Стоимости (price, тип: float)
* ISBN (isbn, тип: str)

**Основные методы структуры**Array**:**

* add\_book(book: Book) -> None  
  Добавляет объект книги в массив.
* cocktail\_sort() -> None  
  Реализует алгоритм **шейкерной сортировки (Cocktail Sort)** для упорядочивания книг по полю price (стоимость).
* is\_sorted() -> bool  
  Проверяет, отсортирован ли массив по стоимости.
* interpolation\_search(target\_price: float) -> Optional[int]  
  Реализует **интерполяционный поиск** для нахождения индекса книги с указанной ценой. Если массив не отсортирован, выбрасывается исключение. Возвращает None, если книга не найдена.

А теперь проведем тесты и замер производительности:

import time

def test\_books() -> None:

    array: Array = Array()

    books: List[Book] = [

        Book("Ivanov", "Publisher1", 300, 500, "ISBN1"),

        Book("Petrov", "Publisher2", 250, 150, "ISBN2"),

        Book("Sidorov", "Publisher3", 400, 300, "ISBN3"),

        Book("Fedorov", "Publisher4", 200, 200, "ISBN4"),

        Book("Aleksandrov", "Publisher5", 350, 100, "ISBN5"),

        Book("Veselov", "Publisher6", 500, 600, "ISBN6")

    ]

    for book in books:

        array.add\_book(book)

    print("Книги до сортировки:")

    print(array.books)

    array.cocktail\_sort()

    print("\nКниги после сортировки:")

    print(array.books)

    # Тест поиска

    price\_to\_find: float = 300

    print(f"\nПоиск книги с ценой {price\_to\_find}:")

    index: Optional[int] = array.interpolation\_search(price\_to\_find)

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

    price\_to\_find = 1000

    print(f"\nПоиск книги с ценой {price\_to\_find}:")

    index = array.interpolation\_search(price\_to\_find)

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

def benchmark\_books() -> None:

    array: Array = Array()

    books: List[Book] = [

        Book(f"Author{i}", f"Publisher{i}", 300 + i, i \* 10, f"ISBN{i}") for i in range(10000)

    ]

    for book in books:

        array.add\_book(book)

    start: float = time.time()

    array.cocktail\_sort()

    index: Optional[int] = array.interpolation\_search(5000)

    end: float = time.time()

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

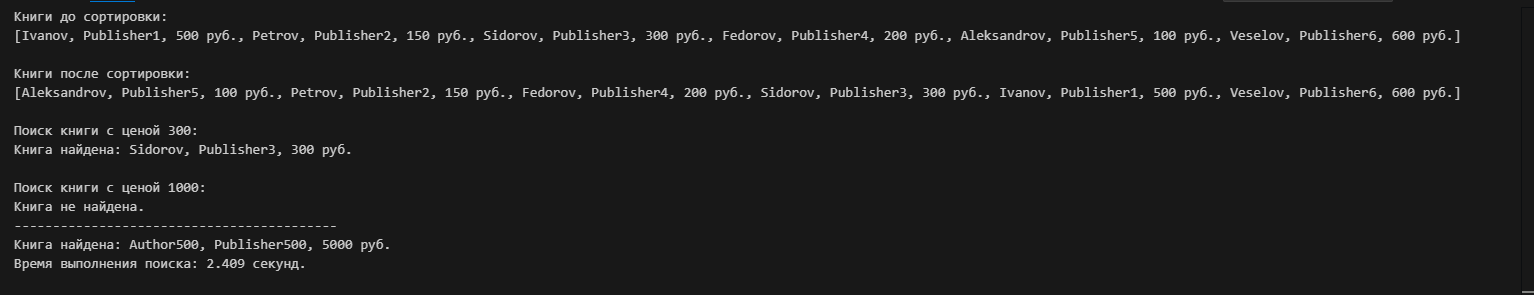
    print(f"Время выполнения поиска: {end - start:.3f} секунд.")

if \_\_name\_\_ == "\_\_main\_\_":

    test\_books()

    print("------------------------------------------")

    benchmark\_books()



Видно, что класс верно выполнил все свои функции, значит код работает корректно.

*Задание 13*

Программа реализации:

from typing import List, Optional

class Book:

    def \_\_init\_\_(self, author: str, publisher: str, pages: int, price: float, isbn: str):

        self.author: str = author

        self.publisher: str = publisher

        self.pages: int = pages

        self.price: float = price

        self.isbn: str = isbn

    def \_\_repr\_\_(self) -> str:

        return f"{self.author}, {self.publisher}, {self.isbn}"

class Array:

    def \_\_init\_\_(self):

        self.books: List[Book] = []

    def add\_book(self, book: Book) -> None:

        self.books.append(book)

    def cocktail\_sort(self) -> None:

        left: int = 0

        right: int = len(self.books) - 1

        while left < right:

            for i in range(left, right):

                if self.books[i].isbn > self.books[i + 1].isbn:

                    self.books[i], self.books[i + 1] = self.books[i + 1], self.books[i]

            right -= 1

            for i in range(right, left, -1):

                if self.books[i - 1].isbn > self.books[i].isbn:

                    self.books[i], self.books[i - 1] = self.books[i - 1], self.books[i]

            left += 1

    def is\_sorted(self) -> bool:

        return all(self.books[i].isbn <= self.books[i + 1].isbn for i in range(len(self.books) - 1))

    def binary\_search(self, target\_isbn: str) -> Optional[int]:

        if not self.is\_sorted():

            raise Exception("Элементы массива не отсортированы по ISBN.")

        low: int = 0

        high: int = len(self.books) - 1

        while low <= high:

            mid: int = (low + high) // 2

            if self.books[mid].isbn == target\_isbn:

                return mid

            elif self.books[mid].isbn < target\_isbn:

                low = mid + 1

            else:

                high = mid - 1

        return None

**Описание кода**

Данный код реализует структуру данных Array, которая представляет собой массив

объектов класса Book. Каждый объект книги содержит информацию о:

* **Авторе** (author, тип: str)
* **Издательстве** (publisher, тип: str)
* **Количестве страниц** (pages, тип: int)
* **Стоимость** (price, тип: float)
* **ISBN** (isbn, тип: str)

**Основные методы структуры**Array**:**

* add\_book(book: Book) -> None  
  Добавляет объект книги в массив.
* cocktail\_sort() -> None  
  Реализует алгоритм **шейкерной сортировки (Cocktail Sort)** для упорядочивания книг по полю isbn.
* is\_sorted() -> bool  
  Проверяет, отсортирован ли массив по isbn.
* binary\_search(target\_isbn: str) -> Optional[int]  
  Реализует **бинарный поиск** для нахождения индекса книги с указанным isbn.  
  Если массив не отсортирован, выбрасывается исключение.  
  Возвращает None, если книга не найдена.

Теперь проведем тесты и посмотрим на производительность

import time

def test\_books() -> None:

    array: Array = Array()

    books: List[Book] = [

        Book("Ivanov", "Publisher1", 300, 500, "ISBN1"),

        Book("Petrov", "Publisher2", 250, 150, "ISBN5"),

        Book("Sidorov", "Publisher3", 400, 300, "ISBN3"),

        Book("Fedorov", "Publisher4", 200, 200, "ISBN2"),

        Book("Aleksandrov", "Publisher5", 350, 100, "ISBN4"),

        Book("Veselov", "Publisher6", 500, 600, "ISBN6")

    ]

    for book in books:

        array.add\_book(book)

    print("Книги до сортировки:")

    print(array.books)

    array.cocktail\_sort()

    print("\nКниги после сортировки:")

    print(array.books)

    # Тест поиска

    isbn\_to\_find: str = "ISBN3"

    print(f"\nПоиск книги с ISBN {isbn\_to\_find}:")

    index: Optional[int] = array.binary\_search(isbn\_to\_find)

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

    isbn\_to\_find = "ISBN100"

    print(f"\nПоиск книги с ISBN {isbn\_to\_find}:")

    index = array.binary\_search(isbn\_to\_find)

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

def benchmark\_books() -> None:

    array: Array = Array()

    books: List[Book] = [

        Book(f"Author{i}", f"Publisher{i}", 300 + i, i \* 10, f"ISBN{i:05d}") for i in range(10000)

    ]

    for book in books:

        array.add\_book(book)

    start: float = time.time()

    array.cocktail\_sort()

    index: Optional[int] = array.binary\_search("ISBN05000")

    end: float = time.time()

    if index is not None:

        print(f"Книга найдена: {array.books[index]}")

    else:

        print("Книга не найдена.")

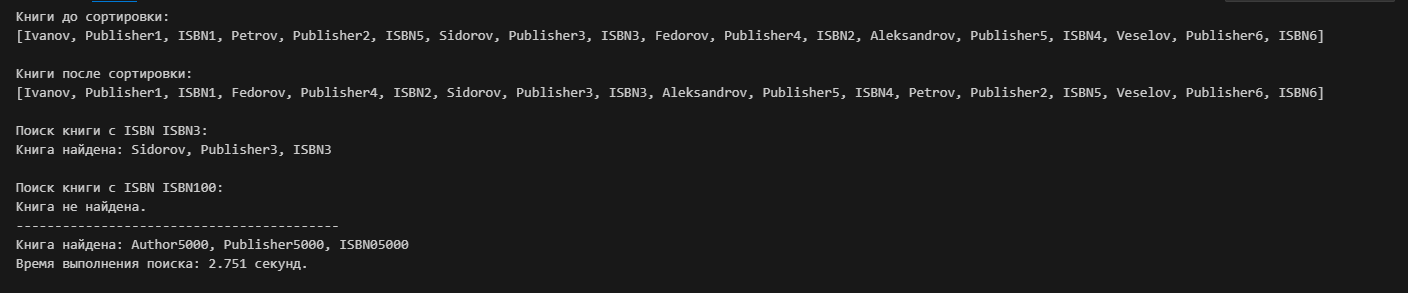
    print(f"Время выполнения поиска: {end - start:.3f} секунд.")

if \_\_name\_\_ == "\_\_main\_\_":

    test\_books()

    print("------------------------------------------")

    benchmark\_books()



Можно заметить, что класс верно выполнил все функции, значит код работает корректно

1. **Вывод**

В ходе лабораторной работы была реализована структура данных «Массив», а также различные методы поиска.

Интерполяционный поиск: данный алгоритм поиска используется для нахождения элемента в отсортированном массиве на основе линейной интерполяции. Он предполагает, что элементы массива равномерно распределены, и пытается предсказать позицию искомого значения для более быстрой навигации по массиву (в данном случае по полю «стоимость»).

Бинарный поиск: данный алгоритм поиска используется для нахождения элемента в отсортированном массиве. Он последовательно делит диапазон поиска пополам, что обеспечивает высокую эффективность при поиске книги по полю ISBN.