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.

2. 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.

3. 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()
```

```
Книги до сортировки:
[Ivanov, Publisher1, 500 руб., Petrov, Publisher2, 150 руб., Sidorov, Publisher3, 300 руб., Fedorov, Publisher4, 200 руб., Aleksandrov, Publisher5, 100 руб., Veselov, Publisher6, 600 руб.]

Книги после сортировки:
[Aleksandrov, Publisher5, 100 руб., Petrov, Publisher2, 150 руб., Fedorov, Publisher4, 200 руб., Sidorov, Publisher3, 300 руб., Ivanov, Publisher1, 500 руб., Veselov, Publisher6, 600 руб.]

Поиск книги с ценой 300:
Книга найдена: Sidorov, Publisher3, 300 руб.

Поиск книги с ценой 1000:
Книга не найдена.
----------------------------------------
Книга найдена: Author500, Publisher500, 5000 руб.
Время выполнения поиска: 2.409 секунд.
```

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)

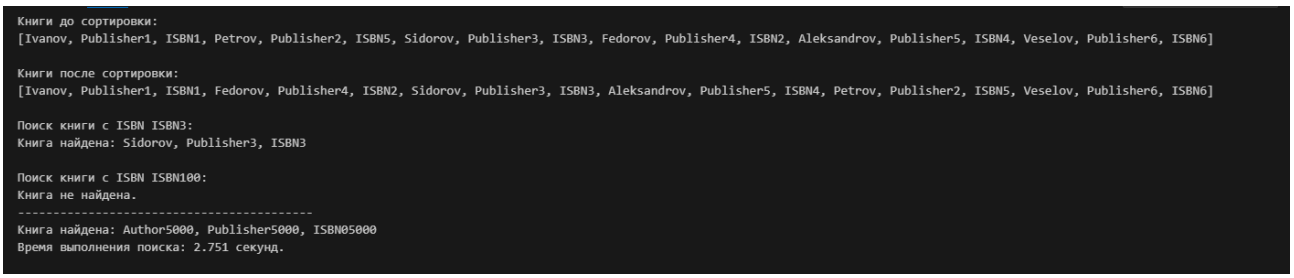
Основные методы структуры Array**:**

- 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

4. 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.