КАФЕДРА

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

РАЗРАБОТКА ПРОГРАММЫ

«Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы»

по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № _____

подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Содержание

1. Задание на курсовой проект.

Цель курсового проектирования: изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ.

Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов.

Тема курсового проекта: «Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы на тему регистрации больных в поликлинике».

Вариант задания:

1.Расчет варианта: 3(Регистрация больных в поликлинике)

2.Расчет метода хэширования: 0(Открытое хэширование)

3.Расчет метода сортировки: 1(Шейкерная)

4.Расчет вида списка: 3(Циклический двунаправленный)

5.Расчет метода обхода дерева: 1(Обратный)

6.Расчет алгоритма поиска слова в тексте: 0(Боуера и Мура)

2. Введение.

Реализация программы, описанной выше, предназначена для автоматизации и упрощения процесса регистрации больных в поликлинике. Эта программа обладает функциональностью, которая помогает ведению базы данных клиентов, докторов и направлений, а также предоставляет различные операции для работы с этими данными. Основная задача программы - обеспечить удобный доступ и управление информацией о больных, их регистрационных данных, направлениях на прием, а также информацией о докторах, их расписании и кабинетах.

Реализация такой программы позволяет оптимизировать процессы работы поликлиники, обеспечивает быстрый доступ к информации, минимизирует ошибки при регистрации и облегчает поиск необходимых данных. Это способствует повышению эффективности работы медицинского персонала и обеспечивает более комфортное обслуживание пациентов.

3. Алгоритмы и структуры данных.

1)AVL-дерево.

AVL-дерево является сбалансированным двоичным деревом поиска, которое обеспечивает эффективный поиск, вставку и удаление элементов. В данном проекте оно было использовано для хранения информации о докторах, с ключом, соответствующим ФИО врача. Благодаря сбалансированной структуре AVL-дерева, время выполнения операций поиска, вставки и удаления докторов остается логарифмическим, что обеспечивает высокую производительность и эффективность программы.

2) Хэш-таблица.

Хэш-таблица - структура данных, которая позволяет эффективно хранить и получать доступ к данным с помощью хэш-функции. В данном проекте она была использована для хранения данных о клиентах в поликлинике по регистрационному номеру. Хэш-таблица позволяет быстро определить наличие клиента в базе данных и получить информацию о нем. Это значительно ускоряет процесс выдачи направления и обеспечивает более эффективное управление поликлиникой.

3) Циклический двунаправленный список.

Циклический двунаправленный список является структурой данных, которая позволяет хранить и обрабатывать данные последовательно, включая возможность добавления и удаления элементов в любом месте списка. Этот вид списка позволяет эффективно добавлять и удалять записи о направлениях, а также обеспечивает удобный доступ к данным. Реализация циклического двунаправленного списка включает узлы, каждый из которых содержит данные и ссылки на предыдущий и следующий элементы списка, а также есть ссылка с последнего элемента списка на начало списка и наоборот.

4) Шейкерная сортировка.

Для сортировки списка направлений был использован алгоритм шейкерной сортировки. Шейкерная сортировка представляет собой двунаправленную пузырьковую сортировку. В этом случае алгоритм обрабатывает массив сначала слева направо, перемещая таким образом наибольший элемент в конец массива, а затем справа налево, перемещая наименьший элемент в начало массива. Шейкерная сортировка извлечения проста в реализации и эффективна для небольших списков, так как имеет квадратичную сложность времени выполнения. Этот метод сочетает в себе преимущества сортировки пузырьком и сортировки вставками, что позволяет эффективно упорядочить данные врачей по ФИО.

5) Метод обхода AVL-дерева обратный.

Для вывода списка клиентов на экран в курсовом проекте был использован метод обхода AVL-дерева обратный (обратный обход в глубину). Этот метод позволяет последовательно обойти все узлы дерева, начиная с самых левых узлов и заканчивая самыми правыми.

6) Открытое хэширование:

Для реализации хеш-таблицы, используемой для хранения данных о больных, был выбран метод открытого хеширования. При этом методе каждому ключу (регистрационному номеру больного) сопоставляется индекс в массиве хеш-таблицы. В случае коллизий, когда разным ключам сопоставляется один и тот же индекс, используется линейное пробирование для поиска следующего свободного слота в таблице.

7) Расчет поиска слова в тексте алгоритмом Бойера-Мура:

Для поиска больного по его ФИО был выбран алгоритм Бойера-Мура (БМ). Этот алгоритм позволяет эффективно находить все совпадения заданного фрагмента ФИО в тексте, учитывая особенности этого поиска и обеспечивая быстродействие.

4. Описание программы.

Руководство по использованию программы:

1) Выход: Функция exit() вызывается для завершения выполнения программы и выхода из нее.

2) Вывести список больных: Функция print_clients() вызывается для вывода списка всех зарегистрированных больных из хеш-таблицы.

3) Добавить больного: Функция add_client() вызывается для добавления нового больного в хеш-таблицу. Она запрашивает у пользователя необходимые данные, такие как регистрационный номер, ФИО, возраст и т. д., и добавляет их в хеш-таблицу.

4) Удалить больного: Функция delete_client() вызывается для удаления данных о больном из хеш-таблицы. Она запрашивает у пользователя регистрационный номер больного и удаляет соответствующую запись из хеш-таблицы.

5) Очистить список больных: Функция clear_client() вызывается для полной очистки списка больных в хеш-таблице. Все данные о больных будут удалены.

6) Поиск больного по регистрационному номеру: Функция find_client_by_number() вызывается для поиска больного по заданному регистрационному номеру. Она запрашивает у пользователя регистрационный номер и выводит соответствующую информацию о найденном больном.

7) Поиск больного по его ФИО: Функция find_client_by_name() вызывается для поиска больного по заданному ФИО. Она запрашивает у пользователя ФИО и выводит информацию о всех найденных больных с указанным ФИО.
8) Добавить врача: Функция add_doctor() вызывается для добавления нового врача в АВЛ-дерево. Она запрашивает у пользователя необходимые данные, такие как ФИО, должность и т. д., и добавляет их в АВЛ-дерево.
9) Удалить врача: Функция delete_doctor() вызывается для удаления данных о враче из АВЛ-дерева. Она запрашивает у пользователя ФИО врача и удаляет соответствующую запись из АВЛ-дерева.
10) Вывести список врачей: Функция print_doctors() вызывается для вывода списка всех врачей из АВЛ-дерева.
11) Очистить список врачей: Функция clear_doctors() вызывается для полной очистки списка врачей в АВЛ-дереве. Все данные о врачах будут удалены.
12) Поиск врача по ФИО: Функция find_doctor_by_name() вызывается для поиска врача по заданному ФИО. Она запрашивает у пользователя ФИО и выводит информацию о найденном враче.
13) Поиск врача по должности: Функция find_doctor_by_post() вызывается для поиска врача по заданной должности. Она запрашивает у пользователя должность и выводит информацию о всех найденных врачах с указанной должностью.
14) Регистрация направления: Функция add_direction() вызывается для регистрации направления к врачу. Она запрашивает у пользователя необходимые данные, такие как ФИО врача и данные о направлении, и добавляет их в циклический двунаправленный список.
15) Возврат направления: Функция delete_direction() вызывается для возврата направления к врачу. Она запрашивает у пользователя данные о направлении и удаляет соответствующую запись из циклического двунаправленного списка.
16) Вывести список направлений: Функция print_direction() вызывается для вывода списка всех зарегистрированных направлений к врачу из циклического двунаправленного списка.

5. Тестирование программы.

Исходные данные для тестовых прогонов программы:

```
"{
  "clients": {
    "1735": [
      {
```

```
        "number": "01-123123",
        "name": "тест",
        "date_of_birth": "2002",
        "adress": "дом",
        "work": "строитель"
      }
    ],
    "1834": [
      {
        "number": "02-123123",
        "name": "тест2",
        "date_of_birth": "2002",
        "adress": "дом",
        "work": "тест"
      }
    ],
    "1935": [
      {
        "number": "03-123123",
        "name": "тест3",
        "date_of_birth": "2002",
        "adress": "дом",
        "work": "тест"
      }
    ]
  },
  "direction": [
    {
      "number": "01-123123",
      "name": "тест2",
      "date": "2023",
      "time": "456"
    },
    {
      "number": "02-123123",
      "name": "тест",
      "date": "2023",
      "time": "456"
    },
    {
      "number": "02-123123",
      "name": "тест2",
      "date": "2023",
      "time": "456"
    },
    {
      "number": "01-123123",
      "name": "тест",
      "date": "апсеир",
      "time": "ольрт"
    }
  ],
  "doctors": [
    {
      "name": "тест",
      "post": "qq ww ee",
      "number_cabinet": 1,
      "schedule": "12-13"
    },
    {
      "name": "тест2",
      "post": "qq gg",
      "number_cabinet": 2,
```

```
      "schedule": "1-2"
    }
  ]
}
```

Действия, которые проводились с базой:
1) Добавили нового больного.

```
Регистрационный номер (MM-NNNNNN):УУ-121212
Неправильный формат
Регистрационный номер (MM-NNNNNN):51-121212
ФИО: Тестов Тест Тестович
Дата рождения: 01.01.1999
Адрес проживания: г.Санкт-Петербург, ул. Красная, д.10, кв.9
Место работы (учёбы): г.Санкт-Петербург, ул. Московская, д.43, оф.403
```

2) Вывели список больных.

```
Регистрационный номер: 01-123123
ФИО:тест
Дата рождения:2002
Адрес:дом
Место работы:строитель
--------------------
Регистрационный номер: 02-123123
ФИО:тест2
Дата рождения:2002
Адрес:дом
Место работы:тест
--------------------
Регистрационный номер: 03-123123
ФИО:тест3
Дата рождения:2002
Адрес:дом
Место работы:тест
--------------------
Регистрационный номер: 51-121212
ФИО:Тестов Тест Тестович
Дата рождения:01.01.1999
Адрес:г.Санкт-Петербург, ул. Красная, д.10, кв.9
Место работы:г.Санкт-Петербург, ул. Московская, д.43, оф.403
```

3) Удалили больного.

```
ФИО: тест3
```

4) Нашли больного по регистрационному номеру.

```
Регистрационный номер (MM-NNNNNN):02-123123
Регистрационный номер: 02-123123
ФИО:тест2
Дата рождения:2002
Адрес:дом
Место работы:тест
```

5) Нашли клиента по ФИО.

```
ФИО: Тестов Тест Тестович
Регистрационный номер: 51-121212
ФИО:Тестов Тест Тестович
Дата рождения:01.01.1999
Адрес:г.Санкт-Петербург, ул. Красная, д.10, кв.9
Место работы:г.Санкт-Петербург, ул. Московская, д.43, оф.403
```

6) Добавили нового врача.

```
ФИО доктора: Фокин Артур Сафарович
Должность: терапевт
Номер кабинета: 55
График приема: Пн-Пт 9:00-15:00
```

7) Вывели всех врачей.

```
ФИО:тест
Должность:qq ww ee
Номер кабинета:1
График приёма:12-13
--------------------
ФИО:Фокин Артур Сафарович
Должность:терапевт
Номер кабинета:55
График приёма:Пн-Пт 9:00-15:00
--------------------
ФИО:тест2
Должность:qq gg
Номер кабинета:2
График приёма:1-2
```

8) Удалили врача.

```
ФИО доктора: тест2
```

8) Нашли врача по ФИО.

```
ФИО доктора: Фокин Артур Сафарович
ФИО:Фокин Артур Сафарович
Должность:терапевт
Номер кабинета:55
График приёма:Пн-Пт 9:00-15:00
```

9) Нашли врача по должности.

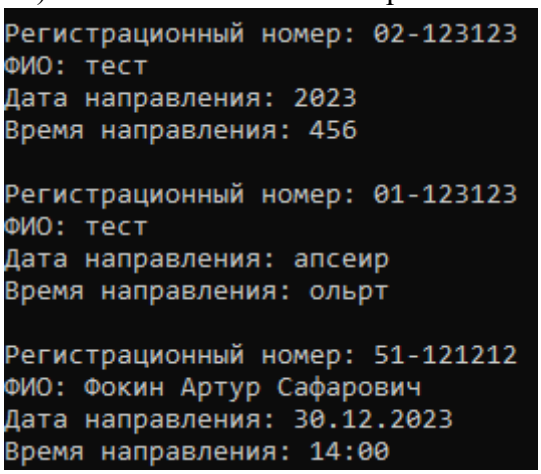
```
Должность: терапевт

ФИО:Фокин Артур Сафарович
Должность:терапевт
Номер кабинета:55
График приёма:Пн-Пт 9:00-15:00
```

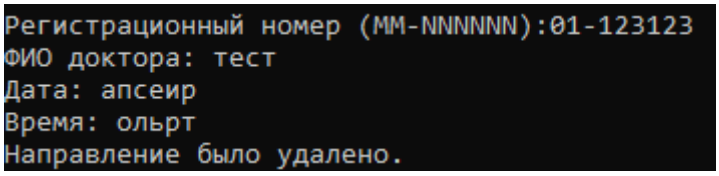
10) Сделали регистрацию направления.

```
Регистрационный номер (MM-NNNNNN):51-121212
ФИО доктора: Фокин Артур Сафарович
Дата: 30.12.2023
Время: 14:00
```

11) Вывели список всех направлений.



12) Удалили направление.



6. Заключение.

Реализация программы была выполнена с использованием вычислительных машин, которые обеспечивают хранение данных и выполнение операций. Корректное функционирование программы зависит от правильного ввода данных и соблюдения инструкций программы.

Результатом выполнения проекта является полнофункциональная информационная система, способная выполнять операции регистрации, удаления, просмотра и поиска данных о больных, врачах и направлениях к врачу. Проект предоставляет удобный интерфейс, основанный на меню, который обеспечивает простоту и эффективность использования системы.

В результате выполнения данного курсового проекта были получены навыки разработки программного обеспечения, применения алгоритмов и структур данных, а также улучшены навыки в программировании и организации проекта.

7. Список литературы.

1)Алгоритмы и структуры данных: учеб. пособие /
2) Структуры и алгоритмы обработки данных / Павлов, Первова:, Лань,2021
3) Алгоритмы/ Вазирани Умеш, Дасгупта Санджой

8. Приложение.

8.1 Листинг программы:

client.py

```
import random

class Client:
    def __init__(self, number, name, date_of_birth, adress, work):
        self.number = number                                    # Регистрационный
номер (строка ММ-NNNNNN)
        self.name = name                                        # ФИО(строка)
        self.date_of_birth = date_of_birth                      # дата
рождения(строка)
        self.adress = adress                                    # адрес(целое)
        self.work = work                                        # место
работы(строка)

    def print(self):
        print("Регистрационный номер: %s\nФИО:%s\nДата
рождения:%s\nАдрес:%s\nМесто работы:%s" % (self.number, self.name,
self.date_of_birth, self.adress, self.work))

class HashTable:
    def __init__(self,  count_sigments=2000):
        self.count_sigments = count_sigments
        self.hash_dict = {}

    def generate_hash(self, key):
        out = 1
        for el in key:
            out += ord(el) ** 2

        return int(out % self.count_sigments)

    def add(self, client):
        hash = self.generate_hash(client.number)
        while True:
            if hash in self.hash_dict:
                self.hash_dict[hash].append(client)

            else:
                self.hash_dict[hash] = [client]
                break
```

```
    def get_by_hash(self, hash):
        return self.hash_dict[hash]

    def remove(self, hash, id):
        return self.hash_dict[hash].pop(id)

    def get_by_hash(self, hash):
        return self.hash_dict[hash]

    def clear_table(self):
        self.hash_dict = {}

    def print_table(self):
        for hash in self.hash_dict:
            # print("%d\t" % el, end="")
            for el in self.hash_dict[hash]:
                print("-" * 20)
                el.print()

##########################################################################
###

    def find_by_name(self, name):
        for hash in self.hash_dict:
            for i, el in enumerate(self.hash_dict[hash]):
                if el.name == name:
                    return hash, i, el

        return None, None, None

    def find_by_number(self, number):
        for hash in self.hash_dict:
            for i, el in enumerate(self.hash_dict[hash]):
                if el.number == number:
                    return hash, i, el

        return None, None, None
```

direction.py

```
class Direction:
    def __init__(self, number, name, date, time):
        self.number = number
        self.name = name
        self.date = date
        self.time = time

    def print(self):
        print("Регистрационный номер: %s\nФИО: %s\nДата направления:
%s\nВремя направления: %s" % (self.number, self.name, self.date, self.time))

class Element_list:
    def __init__(self, item, head=None):
        self.item = item

        self.next = None
        self.previous = None
        self.head = (self if (head == None) else head)
        self.tail = None
```

```
    def add(self, item):
        next = self.head
        while (next.next != None):
            next = next.next

        next.next = Element_list(item, self.head)
        next.next.previous = next.next

        self.head.tail = next.next
        next.next.head = self.head
        return next.next

    def get_item(self):
        return self.item

    def get_head(self):
        return self.head

    def get_all(self):
        result = []
        head = self.head
        while (head != None):
            result.append(head.item)
            head = head.next

        return result

    def get_current_item(self, id):
        head = self.head
        for _ in range(id):
            head = head.next

        return head

    def delete_element(self, id):
        if (id < 0 or id > (self.get_length() - 1)):
            return False

        if id == 0:
            self.head = self.head.next
            return True

        elem_curr = self.head
        elem_prev = None
        elem_next = None
        for i in range(id):
            elem_prev = elem_curr
            elem_curr = elem_curr.next
            elem_next = elem_curr.next

        elem_prev.next = elem_next
        # elem_next.previous = elem_prev

        return True

    def get_length(self):
        head = self.head
        count = 0
        while (head != None):
            head = head.next
            count += 1

        return count
```

```
##########################################################################
###

    def search_by_name(self, name):
        head = self.head
        count = 0
        out = []
        while (head != None):
            if (head.item.name == name):
                out.append(count)

            head = head.next
            count += 1

        return out

    def search_by_number(self, number):
        head = self.head
        count = 0
        out = []
        while (head != None):
            if (head.item.number == number):
                out.append(count)

            head = head.next
            count += 1

        return out

    def search_by_name_number_and_datetime(self, number, name, date, time):
        head = self.head
        count = 0
        while (head != None):
            if (head.item.name == name) and (head.item.number == number) and
(head.item.date == date) and (head.item.time == time):
                return count

            head = head.next
            count += 1

        return None
```

doctors.py

```
from app.search import *

def str_to_key(str):
    out = ""
    for el in str:
        if el in "1234567890":
            out += el
        else:
            out += "%d" % ord(el)
    return int(out)

class Doctor:
    def __init__(self, name, post, number_cabinet, schedule):
        self.name = name                                        # ФИО
доктора(строка)
```

```
        self.post = post                                    #
должность(строка)
        self.number_cabinet = number_cabinet                # номер
кабинета(целое)
        self.schedule = schedule                            # График
приема(строка)
        self.name_key = str_to_key(name)

    def print(self):
        print("ФИО:%s\nДолжность:%s\nНомер кабинета:%s\nГрафик приёма:%s" %
(self.name, self.post, self.number_cabinet, self.schedule))

class AVLTree:
    def __init__(self):
        self.root = None
        self.client = None
        self.left = None
        self.right = None
        self.height = 1

    def clear(self):
        self.root = None
        self.client = None
        self.left = None
        self.right = None

    def insert(self, node, client):
        if node == None:
            node = AVLTree()
            node.client = None

        if node.client == None:
            node.client = client
            return node

        elif client.name_key < node.client.name_key:
            node.left = self.insert(node.left, client)

        else:
            node.right = self.insert(node.right, client)

        node.height = 1 + max(self.get_height(node.left),
self.get_height(node.right))

        balance = self.get_balance(node)

        if balance > 1 and client.name_key < node.left.client.name_key:
            return self.right_rotate(node)

        if balance < -1 and client.name_key > node.right.client.name_key:
            return self.left_rotate(node)

        if balance > 1 and client.name_key > node.left.client.name_key:
            node.left = self.left_rotate(node.left)
            return self.right_rotate(node)

        if balance < -1 and client.name_key < node.right.client.name_key:
            node.right = self.right_rotate(node.right)
            return self.left_rotate(node)

        return node

    def left_rotate(self, node):
```

```
        right = node.right
        left = right.left

        right.left = node
        node.right = left

        node.height = 1 + max(self.get_height(node.left),
self.get_height(node.right))
        right.height = 1 + max(self.get_height(right.left),
self.get_height(right.right))

        return right

    def right_rotate(self, node):
        left = node.left
        right = left.right

        left.right = node
        node.left = right

        node.height = 1 + max(self.get_height(node.left),
self.get_height(node.right))
        left.height = 1 + max(self.get_height(left.left),
self.get_height(left.right))

        return left

    def get_height(self, node):
        if node is None:
            return 0

        return node.height

    def get_balance(self, node):
        if node is None:
            return 0

        return self.get_height(node.left) - self.get_height(node.right)

    def search(self, node, number):
        if node is None:
            return None

        if number == node.client.number:
            return node.client

        if number < node.client.number:
            return self.search(node.left, number)

        return self.search(node.right, number)

    def post_order(self, node):
        if (node):
            self.post_order(node.left)
            self.post_order(node.right)
            print("-" * 20)
            if node.client != None:
                node.client.print()

    def post_order_return(self, node):
        out = []
        if (node):
            out += self.post_order_return(node.left)
```

```
            out += self.post_order_return(node.right)
            if node.client != None:
                out.append(node.client)

        return out


#############################################################################
###
    def search_by_number(self, passport, current = None, first = True):
        if (current == None) and first:
            current = self

        if current == None:
            return None

        if current.client.passport == passport:
            return current.client

        a = self.search_by_number(passport,current.left,False)
        b = self.search_by_number(passport,current.right,False)
        if a != None:
            return a
        else:
            return b

    def search_by_name(self, name):
        name_key = str_to_key(name)
        current = self
        while True:
            if current == None:
                return None

            if current.client.name_key == name_key:
                return current.client

            elif name_key < current.client.name_key:
                current = current.left

            else:
                current = current.right

#############################################################################
###
    def search_by_post(self, text, current=None):
        if current == None:
            current = self

        result = []
        if boyerMurSearch(text, current.client.post):
            result.append(current.client)

        if current.left != None:
            result += self.search_by_post(text, current.left)

        if current.right != None:
            result += self.search_by_post(text, current.right)

        return result

    def delete_by_number(self, licence):
        name_key = sum([ord(i) for i in licence])
        current = self
```

```
        go_left = False
        postv = None
        while True:
            print(current.client.passport)
            if current == None:
                return False

            if current.client.licence == licence:
                current.client = None
                if postv != None:
                    if go_left:
                        postv.left = None
                    else:
                        postv.right = None

                else:
                    postv = self

                left = current.left
                right = current.right
                if left != None:
                    self.insert(postv, left.client)

                if right != None:
                    self.insert(postv, right.client)

                return True

            else:
                postv = current
                if name_key < current.client.name_key:
                    current = current.left
                    go_left = True

                else:
                    current = current.right
                    go_left = False
```

file.py

```
import os
import json

def save_dict(dict, name):
    json.dump(dict, open(str(name) + '.json','w',encoding='utf-8'), indent=2,
ensure_ascii=False)

def read_dict(name):
    with open(str(name) + '.json', encoding='utf-8') as fh:
        data = json.load(fh)
    return data

class Data_file:
    def __init__(self, file_name="data"):
        self.file_name = file_name

        self.clients = {}
        self.direction = []
        self.doctors = []

        self.read()
```

```
    def clear(self):
        self.clients = {}
        self.direction = []
        self.doctors = []

    def save(self):
        data = {
            "clients": self.clients,
            "direction": self.direction,
            "doctors": self.doctors
        }
        save_dict(data, self.file_name)

    def read(self):
        if not os.path.exists(self.file_name + '.json'):
            self.save()
            self.read()
            return

        data = read_dict(self.file_name)

        self.clients = data["clients"]
        self.direction = data["direction"]
        self.doctors = data["doctors"]

    def get_all(self):
        return self.clients, self.direction, self.doctors
```

main.py

```
import datetime

from app.structs.client import *
from app.structs.direction import *
from app.structs.doctors import *
from app.file import *
from app.sort import *
from app.menu import *
from app.search import *

###############################################################################
###

data_file = Data_file("data")

list = None
tree = AVLTree()
hash_table = HashTable()

###############################################################################
###

def read_file():
    global list
    global tree
    global hash_table
    for el in data_file.doctors:
        tree = tree.insert(tree, Doctor(**el))

    for key in data_file.clients:
```

```
        for el in data_file.clients[key]:
            hash_table.add(Client(**el))

    for el in data_file.direction:
        obj = Direction(**el)
        if list == None:
            list = Element_list(obj)
        else:
            list.add(obj)

def save_to_file():
    global list
    global tree
    global hash_table
    data_file.clear()
    tree_return = tree.post_order_return(tree)
    for el in tree_return:
        data = {}
        data["name"] = el.name
        data["post"] = el.post
        data["number_cabinet"] = el.number_cabinet
        data["schedule"] = el.schedule
        data_file.doctors.append(data)

    for key in hash_table.hash_dict:
        data_file.clients[key] = []
        for el in hash_table.hash_dict[key]:
            data = {}
            data["number"] = el.number
            data["name"] = el.name
            data["date_of_birth"] = el.date_of_birth
            data["adress"] = el.adress
            data["work"] = el.work
            data_file.clients[key].append(data)

    if list != None:
        out = list.get_all()
        for el in out:
            data = {}
            data["number"] = el.number
            data["name"] = el.name
            data["date"] = el.date
            data["time"] = el.time
            data_file.direction.append(data)

    data_file.save()

read_file()

##############################################################################
###

def print_clients():
    hash_table.print_table()

def add_client():
    global hash_table
    data = {}
    data['number'] = check_format("Регистрационный номер (ММ-NNNNNN):", "ММ-
NNNNNN")
    data['name'] = input("ФИО: ")
    data['date_of_birth'] = input("Дата рождения: ")
    data['adress'] = input("Адрес проживания: ")
```

```
    data['work'] = input("Место работы (учёбы): ")
    hash_table.add(Client(**data))

def delete_client():
    global hash_table
    name = input("ФИО: ")
    hash, id, obj = hash_table.find_by_name(name)
    if hash == None:
        print("Больного с таким именем нет.")
        return

    hash_table.remove(hash, id)

    arr = list.search_by_number(obj.number)
    for i in range(len(arr)-1, -1, -1):
        list.delete_element(arr[i])

def clear_client():
    global hash_table
    hash_table.clear()
    print("Таблица больных очищена.")

def find_client_by_number():
    number = check_format("Регистрационный номер (ММ-NNNNNN):", "ММ-NNNNNN")
    hash, id, obj = hash_table.find_by_number(number)
    if hash == None:
        print("Больного с таким номером нет.")
        return

    obj.print()

def find_client_by_name():
    name = input("ФИО: ")
    hash, id, obj = hash_table.find_by_name(name)
    if hash == None:
        print("Больного с таким именем нет.")
        return

    obj.print()

##############################################################################
###

def add_doctor():
    global tree
    data = {}
    data['name'] = input("ФИО доктора: ")
    data['post'] = input("Должность: ")
    data['number_cabinet'] = int(input("Номер кабинета: "))
    data['schedule'] = input("График приема: ")
    tree = tree.insert(tree, Doctor(**data))

def delete_doctor():
    global tree
    global list
    name = input("ФИО доктора: ")
    tree_return = tree.post_order_return(tree)
    tree.clear()
    deleted_flag = False
    for el in tree_return:
        if el.name != name:
            tree.insert(tree, el)
        else:
```

```
            deleted_flag = True

    if not deleted_flag:
        print("Доктора с таким именем не существует.")

    arr = list.search_by_name(name)
    for i in range(len(arr)-1, -1, -1):
        list.delete_element(arr[i])

def print_doctors():
    global tree
    tree.post_order(tree)

def clear_doctors():
    global tree
    tree.clear()
    print("Дерево с докторами было очищенно.")

def find_doctor_by_name():
    global tree
    name = input("ФИО доктора: ")
    obj = tree.search_by_name(name)
    if obj == None:
        print("Такого доктора нет.")
        return

    obj.print()

def find_doctor_by_post():
    global tree
    post = input("Должность: ")
    obj_list = tree.search_by_post(post)

    for el in obj_list:
        print()
        el.print()

##########################################################################
###

def add_direction():
    global list
    data = {}
    data['number'] = check_format("Регистрационный номер (ММ-NNNNNN):", "ММ-
NNNNNN")
    hash, id, obj = hash_table.find_by_number(data['number'])
    if hash == None:
        print("Больного с таким номером нет.")
        return

    data['name'] = input("ФИО доктора: ")
    obj = tree.search_by_name(data['name'])
    if obj == None:
        print("Такого доктора нет.")
        return

    data['date'] = input("Дата: ")
    data['time'] = input("Время: ")

    obj = Direction(**data)
    if list == None:
        list = Element_list(obj)
    else:
```

```
        list.add(obj)

def delete_direction():
    global list
    data = {}
    data['number'] = check_format("Регистрационный номер (ММ-NNNNNN):", "ММ-
NNNNNN")
    hash, id, obj = hash_table.find_by_number(data['number'])
    if hash == None:
        print("Больного с таким номером нет.")
        return

    data['name'] = input("ФИО доктора: ")
    obj = tree.search_by_name(data['name'])
    if obj == None:
        print("Такого доктора нет.")
        return

    data['date'] = input("Дата: ")
    data['time'] = input("Время: ")

    id = list.search_by_name_number_and_datetime(**data)
    if id == None:
        print("Такого направления нет.")
        return

    list.delete_element(id)
    print("Направление было удалено.")

def print_direction():
    global list
    if list == None:
        return

    for i in range(list.get_length()):
        print()
        obj = list.get_current_item(i)
        obj.item.print()

##############################################################################
###

menu = Menu([
    ["Выход\n", exit],
    ['[HashTable] Вывести список больных', print_clients],
    ['[HashTable] Добавить больного', add_client],
    ['[HashTable] Удалить больного', delete_client],
    ['[HashTable] Очистить список больных', clear_client],
    ['[HashTable] Поиск больного по регистрационному номеру',
find_client_by_number],
    ['[HashTable] Поиск больного по его ФИО\n', find_client_by_name],
    ['[AVLTree] Добавить врача', add_doctor],
    ['[AVLTree] Удалить врача', delete_doctor],
    ['[AVLTree] Вывести список врачей', print_doctors],
    ['[AVLTree] Очистить список врачей', clear_doctors],
    ['[AVLTree] Поиск врача по ФИО', find_doctor_by_name],
    ['[AVLTree] Поиск врача по должности\n', find_doctor_by_post],
    ['[list] Регистрация направления', add_direction],
    ['[list] Возврат направления', delete_direction],
    ['[list] Вывести списко направлений', print_direction]
], save_to_file)

menu.run()
```

menu.py

```
import sys
from sys import platform
import os

def clear_screen():
    if platform == "linux" or platform == "linux2":
        os.system("clear")
    elif platform == "darwin":
        os.system("clear")
    elif platform == "win32":
        os.system("cls")

def check_format(promt, template):
    while True:
        str = input(promt)
        if len(str) != len(template):
            print("Неправильный формат")
            continue

        for i in range(len(str)):
            if template[i] in ["N", "M"] and not str[i].isdigit():
                break
            if template[i] == "-" and not (str[i] == "-"):
                break
        else:
            break
        print("Неправильный формат")
    return str

def read_bool(promt=""):
    while True:
        read = input(promt + " (y/n): ")
        if read in ["y", "Y", "n", "N", "д", "Д", "н", "Н"]:
            return (read in ["y", "Y", "д", "Д"])

class Menu:
    def __init__(self, elements, function=None):
        self.elements = elements
        self.function = function

    def run(self):
        #clear_screen()
        while True:
            print("-" * 30)
            for i in range(len(self.elements)):
                print("%2d) %s" % (i, self.elements[i][0]))

            # try:
            menu_input = int(input("\n >> "))
          # clear_screen()
            self.elements[menu_input][1]()
            if self.function != None:
                self.function()
```

search.py

```
def boyerMurSearch(str1, str2):
    t = str1
    # Этап 1: формирование таблицы смещений
    S = set()  # уникальные символы в образе
    M = len(t) # число символов в образе
    d = {}     # словарь смещений
    for i in range(M-2, -1, -1): # итерации с предпоследнего символа
        if t[i] not in S:        # если символ еще не добавлен в таблицу
            d[t[i]] = M-i-1
            S.add(t[i])
    if t[M-1] not in S:     # отдельно формируем последний символ
        d[t[M-1]] = M
    d['*'] = M              # смещения для прочих символов
    # print(d)
    # Этап 2: поиск образа в строке
    a = str2
    N = len(a)
    if N >= M:
        i = M-1       # счетчик проверяемого символа в строке
        while(i < N):
            k = 0
            j = 0
            flBreak = False
            for j in range(M-1, -1, -1):
                if a[i-k] != t[j]:
                    if j == M-1:
                        off = d[a[i]] if d.get(a[i], False) else d['*']  #
смещение, если не равен последний символ образа
                    else:
                        off = d[t[j]]   # смещение, если не равен не
последний символ образа

                    i += off    # смещение счетчика строки
                    flBreak = True  # если несовпадение символа, то flBreak =
True
                    break
                k += 1          # смещение для сравниваемого символа в строке

            if not flBreak:          # если дошли до начала образа, значит,
все его символы совпали
                # print(f"образ найден по индексу {i-k+1}")
                return True
                break
        else:
            # print("образ не найден")
            return False
    else:
        # print("образ не найден")
        return False
```

sort.py

```
def shaker_sort(array):
    length = len(array)
    swapped = True
    start_index = 0
    end_index = length - 1

    while (swapped == True):

        swapped = False
```

```
        # проход слева направо
        for i in range(start_index, end_index):
            arr_buf = sum([ord(i) for i in array[i].name])
            arr_buf2 = sum([ord(i) for i in array[i + 1].name])
            if (arr_buf > arr_buf2) :
                # обмен элементов
                array[i], array[i + 1] = array[i + 1], array[i]
                swapped = True

        # если не было обменов прерываем цикл
        if (not(swapped)):
            break

        swapped = False

        end_index = end_index - 1

        #проход справа налево
        for i in range(end_index - 1, start_index - 1, -1):
            arr_buf = sum([ord(i) for i in array[i].name])
            arr_buf2 = sum([ord(i) for i in array[i + 1].name])
            if (arr_buf > arr_buf2):
                # обмен элементов
                array[i], array[i + 1] = array[i + 1], array[i]
                swapped = True

        start_index = start_index + 1

    return array
```