МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ: _____

ПРЕПОДАВАТЕЛЬ:

Кандидат технических наук, доцент		Матьяш В.А.
(должность, уч. степень, звание)	(подпись, дата)	(инициалы, фамилия)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

«ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ»

ПО КУРСУ: «СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

РАБОТУ ВЫПОЛНИЛ (-А) СТУДЕНТ (-КА): 4033 (номер группы) / Х.В. Сидиропуло (инициалы, фамилия)

/ _____ (подпись студента) / 03.06.2022 (дата отчета)

Санкт-Петербург 2022

СОДЕРЖАНИЕ

1.Задание на курсовой проект

Вариант задания:

1.Предметная область - Обслуживание клиентов оператора сотовой связи

2.Метод хеширования - Закрытое хеширование с двойным хешированием

3.Метод сортировки - Сортировка включением

4.Вид списка - Циклический однонаправленный

5.Метод обхода дерева - Прямой

6.Алгоритм поиска слова в тексте - Боуера и Мура (БМ)

1.1 Обслуживание клиентов оператора сотовой связи

1.6.1. Информационная система для предметной области «Обслуживание клиентов оператора сотовой связи» должна осуществлять
ввод, хранение, обработку и вывод данных о:
– клиентах;
– SIM-картах, принадлежащих оператору сотовой связи;
– выдаче или возврате SIM-карт клиентами.
1.6.2. Данные о каждом клиенте должны содержать:
– Номер паспорта – строка формата «NNNN-NNNNNN», где N – цифры;
– Место и дату выдачи паспорта – строка;
– ФИО – строка;
– Год рождения – целое;
– Адрес – строка.

Примечание: длина строк (кроме номера паспорта) определяется студентом самостоятельно.

1.6.3. Данные о клиентах должны быть организованны в виде АВЛ-дерева поиска, упорядоченного по «номеру паспорта».

1.6.4. Данные о каждой SIM-карте должны содержать:
– Номер SIM-карты – строка формата «NNN-NNNNNNN», где N – цифра;
– Тариф – строка;
– Год выпуска – целое;
– Признак наличия – логическое.

Примечание: длина строк (кроме «номера SIM-карты») определяется студентом самостоятельно.

1.6.5. Данные об SIM-картах должны быть организованны в виде хеш-таблицы, первичным ключом которой является «номер SIMкарты» Метод хеширования определяется вариантом задания.

1.6.6. Данные о выдаче или возврате SIM-карт клиентами должны содержать:
– Номер паспорта – строка, формат которой соответствует аналогичной строке в данных о клиентах;
– Номер SIM-карты – строка, формат которой соответствует аналогичной строке в данных о SIM-картах;
– Дату выдачи – строка;
– Дату окончания действия – строка.

Примечания:

1. Наличие в этих данных записи, содержащей в поле «номер паспорта» значение X и в поле «номер SIM-карты» значение Y, соответственно означает выдачу клиенту с номером паспорта X SIM-карты с номером Y. Отсутствие такой записи означает, что клиенту с номером паспорта X не выдавалась SIM-карта с номером Y.
2. Одному клиенту может быть выдано несколько SIM-карт. Таким образом, могут быть данные, имеющие повторяющиеся значения в своих полях.

1.6.7. Данные о выдаче или возврате SIM-карты клиентов должны быть организованны в виде списка, который упорядочен по первичному ключу – «номер SIM-карты». Вид списка и метод сортировки определяются вариантом задания.

1.6.8. Информационная система «Обслуживание клиентов оператора сотовой связи» должна осуществлять следующие операции:
– регистрацию нового клиента;
– снятие с обслуживания клиента;
– просмотр всех зарегистрированных клиентов;

– очистку данных о клиентах;
– поиск клиента по «номер паспорта». Результаты поиска – все сведения о найденном клиенте и номера SIM-карт, которые ему выданы;
– поиск клиента по фрагментам ФИО или адреса. Результаты поиска – список найденных клиентов с указанием номера паспорта,

ФИО и адреса;
– добавление новой SIM-карты;
– удаление сведений о SIM-карте;
– просмотр всех имеющихся SIM-карт;
– очистку данных о SIM-картах;
– поиск SIM-карты по «номеру SIM-карты».
Результаты поиска – все сведения о найденной SIM-карте, а также ФИО и номер паспорта клиента, которому выдана эта SIM-карта;
– поиск SIM-карты по тарифу. Результаты поиска – список найденных SIM-карт с указанием «номера SIM-карты», тарифа, года выпуска;
– регистрацию выдачи клиенту SIM-карты;
– регистрацию возврата SIM-карты от клиента.

1.6.9. Состав данных о клиенте или SIM-карте, выдаваемых при просмотре всех зарегистрированных клиентов или просмотре всех SIM-карт, определяется студентом самостоятельно, но должен содержать не менее двух полей.

1.6.10. Метод поиска SIM-карты по тарифу определяется студентом самостоятельно. Выбранный метод необходимо сравнить с альтернативными методами.

1.6.11. Поиск клиента по фрагментам ФИО или адреса должен осуществляться путем систематического обхода АВЛ-дерева поиска. Метод обхода определяется вариантом задания.
При поиске клиента по фрагментам ФИО или адреса могут быть заданы как полное ФИО или адрес, так и их части (например, только фамилия клиента без имени
и отчества, только название улицы из адреса).
Для обнаружения заданного фрагмента в полном ФИО или адресе должен применяться алгоритм поиска слова в тексте, указанный в варианте задания.

1.6.12. Регистрация выдачи SIM-карты клиенту должна осуществляться только при наличии SIM-карты у оператора сотовой связи (значение поля «Признак наличия» для соответствующей SIM-карты имеет значение «Истина»).

1.6.13. При регистрации выдачи SIM-карты клиенту или возврата SIM-карты клиентом должно корректироваться значение поля «Признак наличия» для соответствующей SIM-карты.

1.6.14. При удалении сведений о SIM-карте должны быть учтены и обработаны ситуации, когда эта SIM-карта уже выдана клиенту. Аналогичным образом следует поступать и с удалением данных о клиентах.

2.Введение

В настоящее время трудно найти компанию, которая не использует цифровые технологии в рабочем процессе. Для удачной автоматизации программисту требуются определить структуры для хранения реальных данных и подобрать наиболее удачные алгоритмы для их обработки.

Целью данного курсового проекта является разработка приложения для обслуживания клиентов оператора сотовой связи.

Данные о клиентах хранятся в АВЛ-дереве, данные о SIM-картах – в хеш-таблице, а информация о выдаче SIM-карт клиентам – в однонаправленном циклическом списке.

Разрабатываемая система должна обеспечивать функции добавления, удаления и поиска записей во всех указанных структурах.

3.Алгоритмы и структуры данных

3.1. Хеш-таблица. Закрытое хеширование с двойным хешированием.

Данные об SIM-картах должны быть организованны в виде хеш-таблицы. первичным ключом которой является «номер SIM карты».

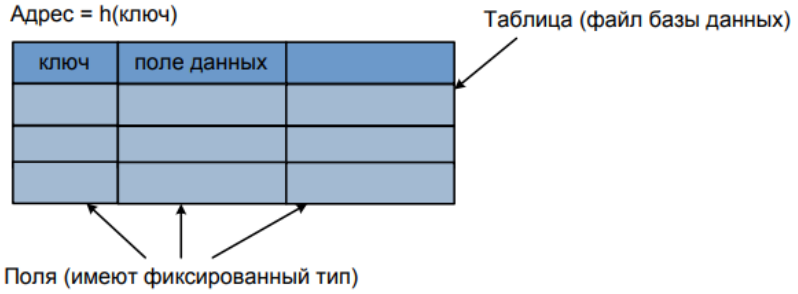
Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией. Хеш-таблица содержит некоторый массив , элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками).

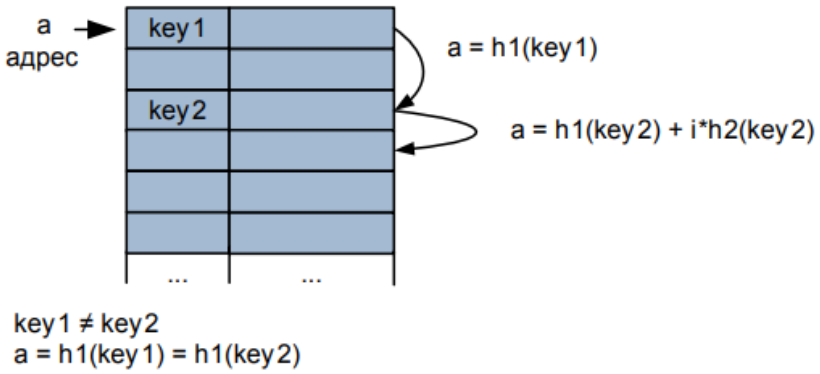
Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение играет роль индекса в массиве . Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превысит 50% (если каждый элемент может равновероятно попасть в любую ячейку) — см. парадокс дней рождения. Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

В некоторых специальных случаях удаётся избежать коллизии полностью. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, использующие подобные хеш-функции, не нуждаются в механизме разрешения коллизий, и называются хеш-таблицами с *прямой адресацией*.



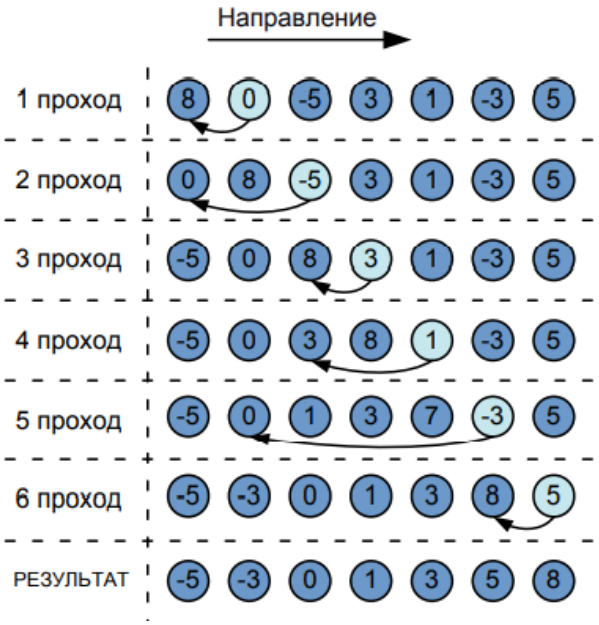
Разрешение коллизий при добавлении элементов методом двойного хеширования выглядит следующих образом:



Метод двойного хеширования основан на том, что в зависимости от значения, которое мы хотим записать в таблицу, высчитывается и величина шага, который будет использован при разрешении коллизий. Величина шага рассчитывается с помощью второй хеш-функции (отличной от основной).

3.2. Метод сортировки. Сортировка вставками

Изначально массив делится на отсортированную и неотсортированную часть. На каждом шаге алгоритма будет выбираться один из элементов неотсортированной части и вставляется на нужную позицию в уже отсортированной части массива до тех пор, пока неотсортированная часть массива данных не будет исчерпана. Метод выбора очередного элемента из исходного массива произволен. Изначально отсортированная часть массива равна одному элементу.

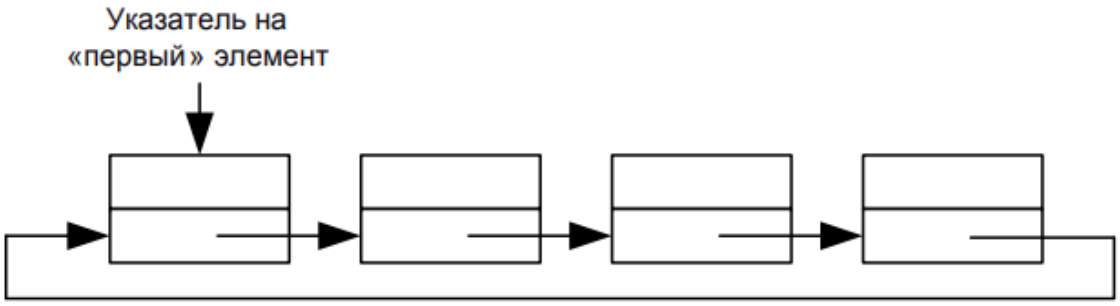


Теоретические временные и пространственные сложности данного метода сортировки приведены ниже:

Метод сортировки	Характеристики			
	T_{max}	T_{mid}	T_{min}	V_{max}
Включением	O(n^2)		O(n)	O(1)

3.3. Вид списка. Циклический однонаправленный список.

Список – это абстрактная структура данных, элементы которой содержат пользовательские данные и указатели на следующий (иногда и на предыдущий) элемент списка.
Идея циклического списка заключается в том, что указатель последнего элемента не является пустым, а указывает на первый элемент. В циклическом однонаправленном списке фактически все элементы являются «средними». В отличие от линейного однонаправленного списка, где удаление «первого» и «последнего» элементов несколько отличается от удаления «среднего», в циклическом однонаправленном списке все элементы могут быть удалены с помощью одного и того же подхода. В циклическом однонаправленном списке понятие «первого» элемента весьма условно. Для полного обхода такого списка достаточно иметь указатель на произвольный элемент.



Хранить указатель на «первый» элемент необязательно, но крайне желательно отслеживать какой-либо указатель на элемент списка, например, это полезно при предотвращении «зацикливания» при поиске элемента.

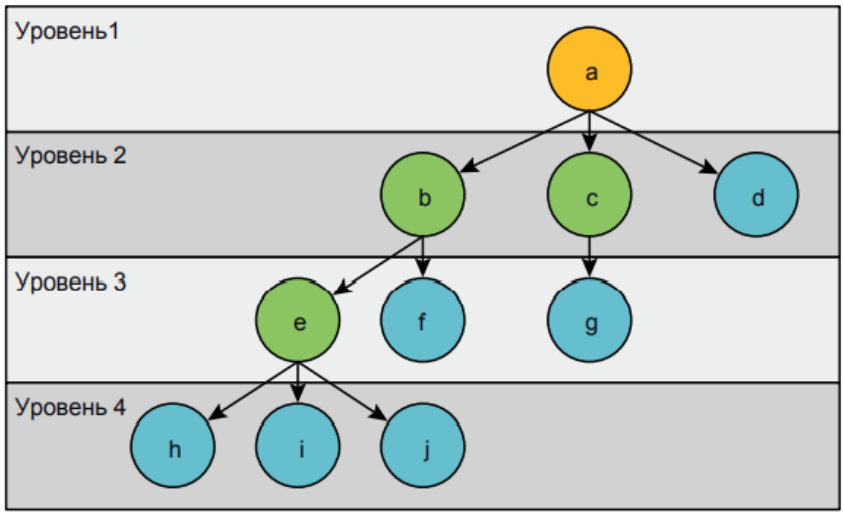
С циклическим однонаправленным списком можно определить следующие операции:
1) вставка (добавление) элемента;
2) просмотр (проход) списка;
3) поиск элемента;
4) удаление элемента.

Наличие только одного указателя позволяет экономить объем памяти для хранения такой структуры данных. Но по аналогии с линейным однонаправленным списком проход списка возможен только в одном направлении. Из плюсов также можно отметить, что операции вставки (добавления) и удаления элементов достаточно упрощены. Использовать циклический список имеет смысл, когда количество «проходов» по списку заранее неизвестно.

3.4. АВЛ дерево. Прямой обход дерева

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

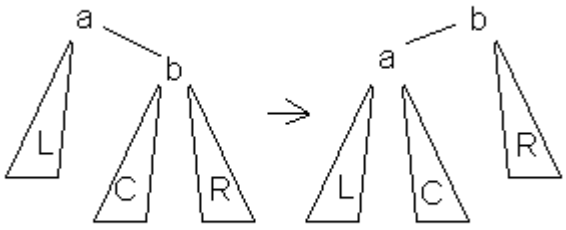


Балансировка

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

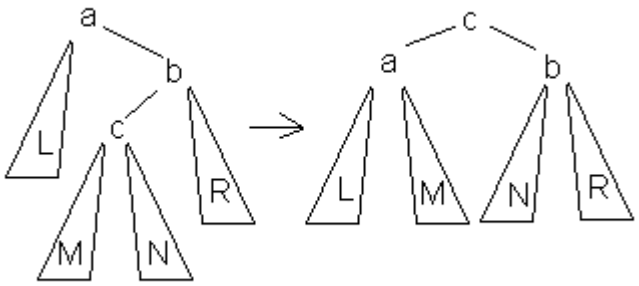
Используются 4 типа вращений:

Малое левое вращение



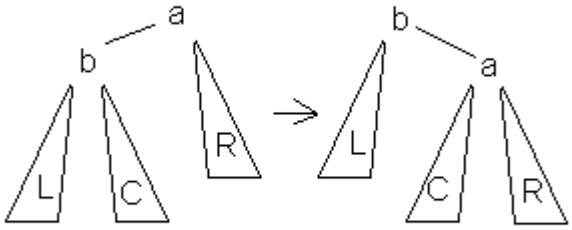
Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева <= высота R.

Большое левое вращение



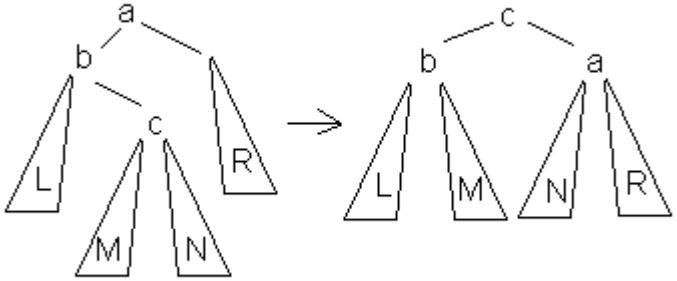
Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева > высота R.

Малое правое вращение



Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота C <= высота L.

Большое правое вращение

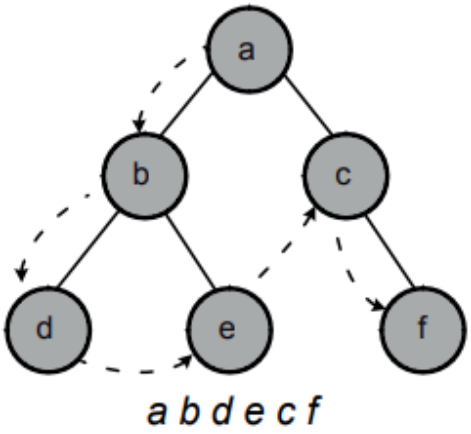


Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота c-поддерева > высота L.

В каждом случае достаточно просто доказать то, что операция приводит к нужному результату и что полная высота уменьшается не более чем на 1 и не может увеличиться. Также можно заметить, что большое левое вращение — это композиция правого малого вращения и левого малого вращения. Из-за условия балансирования высоты дерева O(log(N)), где N — количество вершин, поэтому добавление элемента требует O(log(N)) операций.

Прямой обход дерева

1. Проверяем, не является ли текущий узел пустым или null.
2. Обходим левое поддерево рекурсивно, вызвав функцию центрированного обхода.
3. Показываем поле данных корня (или текущего узла).
4. Обходим правое поддерево рекурсивно, вызвав функцию центрированного обхода.



3.4. Алгоритм поиска в подстроке. Алгоритма Бойра-Мура

В 1975 году Р. Боуер и Д. Мур предложили свой алгоритм поиска слова в тексте.

Предполагается, что данный алгоритм улучшает обработку самого плохого случая. Алгоритм Боуера и Мура (далее – БМ-поиск) предполагает, что символы будут сравниваться с конца слова. Перед началом сравнений необходима предобработка слова, которая преобразует его в некоторую таблицу. В ней помимо символов алфавита (символы, входящие и в слово, и в текст) хранится еще величина, характеризующая расстояние самого правого вхождения символа в слово до правого конца слова.
Если символа в слове вообще нет, то в таблицу заносится величина равная длине слова. Алгоритм будет работать следующим образом:
На первом шаге искомое слово сопоставляется с текстом так, что первый символ слова соотносится с первым символом текста. Далее начинается поэлементное сравнение символов с конца слова. Если все символы совпали, то слово в тексте найдено. Если обнаружено несовпадение символов, то слово можно сместить вправо относительно текста таким образом, чтобы самый правый символ слова оказался на той же позиции, что и последний несовпадающий символ текста. Если несовпадающего символа текста в слове вообще нет, то смещение происходит на длину всего слова относительно текущей позиции.

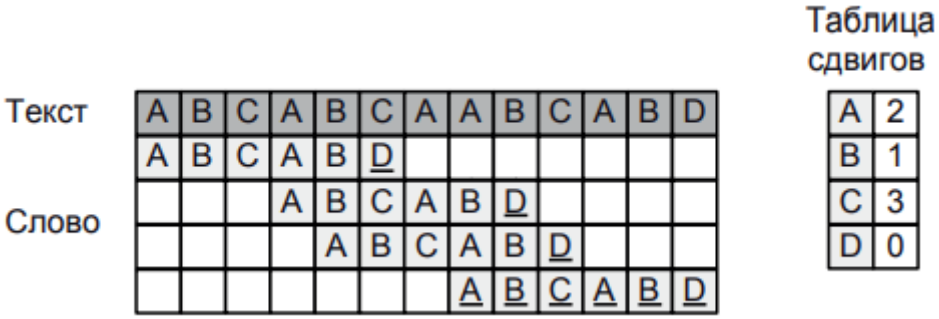


Таблица сдвигов

A	2
B	1
C	3
D	0

К достоинствам данного алгоритма можно отнести то, что смещение слова относительно текста часто происходит сразу на несколько позиций. Данный алгоритм считается наиболее быстрым алгоритмом поиска слова в тексте общего назначения. Существует достаточно большое количество его оптимизаций. К минусам алгоритма стоит отнести то, что его эффективность сильно страдает при «плохих» данных.

4. Описание программы

Программа реализована на языке C++ в виде консольного приложения. В главной функции main() реализовано меню пользователя, в котором каждому действию соответствует определенная цифра.

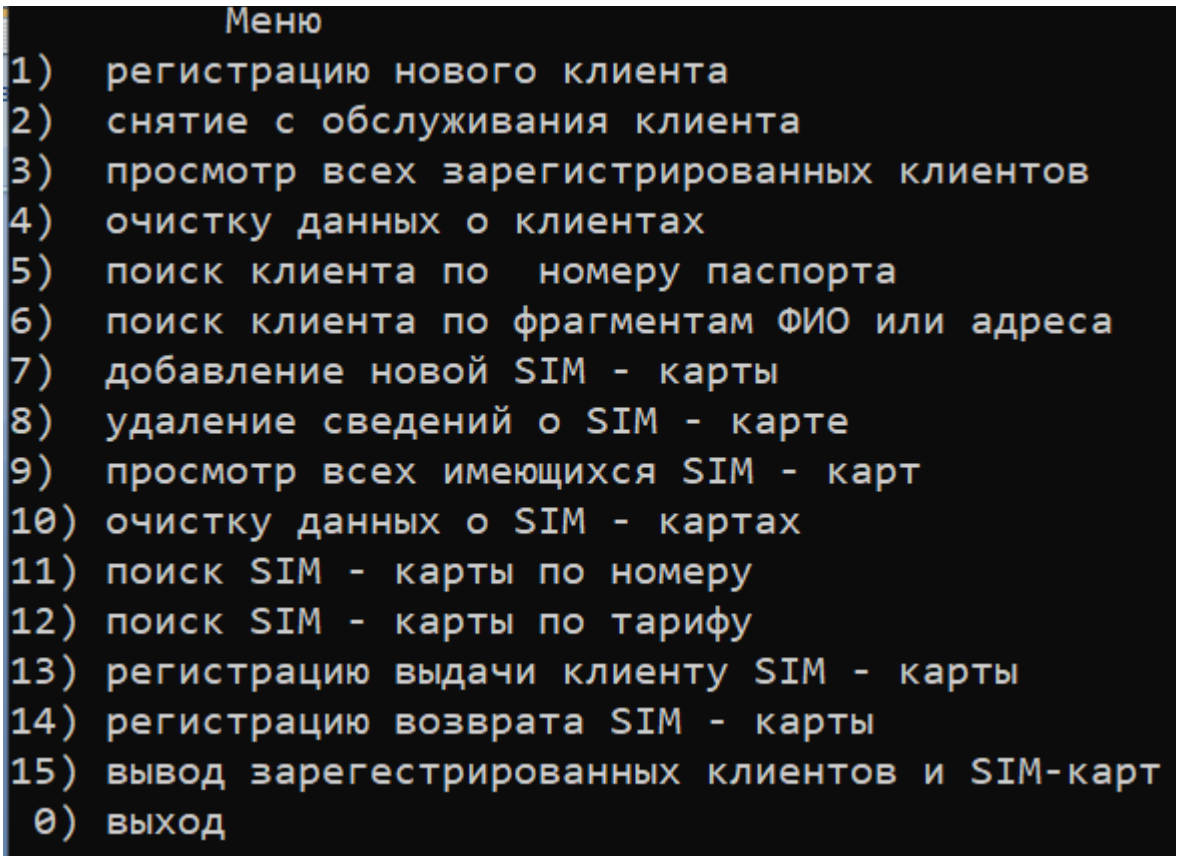
Реализованы следующие функции для работы с клиентами:
1) регистрацию нового клиента
2) снятие с обслуживания клиента
3) просмотр всех зарегистрированных клиентов
4) очистку данных о клиентах
5) поиск клиента по номеру паспорта
6) поиск клиента по фрагментам ФИО или адреса

Для работы с сим-картами:
7) добавление новой SIM - карты
8) удаление сведений о SIM - карте
9) просмотр всех имеющихся SIM - карт
10) очистку данных о SIM - картах
11) поиск SIM - карты по номеру
12) поиск SIM - карты по тарифу

Для работы с регистрацией сим-карты клиенту
13) регистрацию выдачи клиенту SIM - карты
14) регистрацию возврата SIM - карты
15) вывод зарегистрированных клиентов и SIM-карт
0) выход

5. Тестирование программы

Вывод главного меню



Регистрация нового клиента и проверки

```
Введите ФИО
Сидиропуло
Неверно!
Введите ФИО заново:
Сидиропуло Хетаг Владимирович
Введите паспорт
9016-2
Неверно!
Введите серию и номер паспорта заново:
9016-207777
Введите год рождения
второй
Неверно!
Введите дату заново:
2002
Введите адрес
ВО 47
Введите место выдачи паспорта
Владикавказ
Клиент добавлен
```

Далее добавляем еще 2 клиента

Просмотр всего списка клиентов

```
---------------------------------------------
ФИО                          Паспорт           Год рождения        Адрес        Место выдачи паспорта


Павлюх Павел Валерьевич      9016-207666       2002                фрунзенская        ханты-мансийск
Яковлев Владислав Сергеевич  9016-207333       2002                купчино            санкт-петербург
Сидиропуло Хетаг Владимирович 9016-207777      2002                ВО 47              Владикавказ
---------------------------------------------
```

Удаление клиента

```
Введите паспорт клиента:9016-207666
 Клиент удален
```

```
---------------------------------------------
ФИО                          Паспорт          Год рождения        Адрес        Место выдачи паспорта

Сидиропуло Хетаг Владимирович 9016-207777       2002               ВО 47             Владикавказ
Яковлев Владислав Сергеевич   9016-207333       2002               купчино           санкт-петербург
---------------------------------------------
```

Очистка списка клиентов

Без клиентов

```
4
Клиентов нет
```

С клиентами

```
4
Клиенты очищены
```

При существовании выданной сим-карты клиенту

```
4
Существуют зарегистрированные клиенты. Очистка не может быть произведена.
```

Добавление сим-карты

```
Введите номер
999-0692535553535
Неверно!
Введите номер сим-карты заново:
999-888665
Неверно!
Введите номер сим-карты заново:
988-8324665
Введите тариф
безлимит
Введите год выпуска
двадцать второй
Неверно!
Введите дату заново:
2018
Введите наличие (1 - да; 0 - нет)
1
```

После добавляем еще 2 номера

Просмотр всего списка сим-карт

```
Номер Sim                       Тариф           Год выпуска         Наличие
999-0691225                     безлимит        2021                1
988-8324665                     безлимит        2018                1
928-8993246                     мега            2020                1
```

Удаление сим-карты

```
Введите номер SIM для удаления
988-8324665
SIM с номером - 988-8324665 удалена.
```

Если сим-карта выдана клиенту:

```
Введите номер SIM для удаления
999-0691225
Сим карта выдана. Удалить невозможно.
```

Очистка данных о сим-картах:

Если выданных сим-карт нет:

```
10
Сим-карты очищены
```

Если существует выданная клиенту сим-карта:

```
10
Существуют зарегестрированные клиенты. Удаление невозможно.
```

Поиск клиента по номеру паспорта

```
Введите номер паспорта:9016-207777
ФИО клиента: Сидиропуло Хетаг Владимирович
Паспорт клиента: 9016-207777
Год рождения: 2002
Город проживания: ВО 47
Место выдачи пасспорта: Владикавказ


Сим-карт нет
```

Если существует зарегистрированная сим-карта:

```
5
Введите номер паспорта:9016-207777
ФИО клиента: Сидиропуло Хетаг Владимирович
Паспорт клиента: 9016-207777
Год рождения: 2002
Город проживания: ВО 47
Место выдачи пасспорта: Владикавказ


988-8324665
```

Поиск по фрагментам ФИО или адреса:

```
6
Введите строку :
Сидиропуло
ФИО: Сидиропуло Хетаг Владимирович      Паспорт: 9016-207777    Год рождения: 2002
Место жительства: ВладикавказМесто выдачи: ВО 47
```

```
Введите строку :
ВО 47
ФИО: Сидиропуло Хетаг Владимирович      Паспорт: 9016-207723    Год рождения: 2002      Место жительства: Владикавказ
Место выдачи: ВО 47
```

Регистрация выдачи сим-карты

```
13
Введите номер паспорта: 9016-207777
Введите номер сим карты: 999-0691225
Введите дату начала активации:
Год: 2022
Месяц: 3
День: 6
Введите дату деактивации:
Год: 2022
Месяц: 5
День: 6
Сим-карта выдана:
```

Просмотр зарегистрированных сим-карт

```
15
 | Номер сим   |   Паспорт   |   Дата начала   |   Дата окончания  |
 | 999-0691225 | 9016-207777 |   06.03.2022    |    06.05.2022     |
```

Регистрация возврата сим-карт

```
14
Введите паспорт:
9016-207777
Возврат оформлен
```

Просмотр выданных клиентам сим-карт после снятия с обслуживания:

```
15
База данных пуста.
```

Поиск номеров по тарифу

Исходные сим-карты:

```
9

Номер Sim                       Тариф           Год выпуска     Наличие
999-0691225                     безлимит        2022            1
988-8324665                     безлимит        2020            1
928-8324665                     мега            2021            1
```

Поиск по тарифу “безлимит” :

```
12
Введите тариф для поиска
безлимит

Номер Sim                       Тариф           Год выпуска     Наличие
999-0691225                     безлимит        2022            1
988-8324665                     безлимит        2020            1
```

Поиск по тарифу “мега” :

```
12
Введите тариф для поиска
мега

Номер Sim                       Тариф           Год выпуска     Наличие
928-8324665                     мега            2021            1
```

Поиск сим-карты по номеру:

```
11
Введите номер SIM для поиска
999-0691225
Номер СИМ 999-0691225
Тариф безлимит
Год выпуска 2022
Наличие 1
```

6. Заключение

В ходе выполнения курсового проекта была разработана система для обслуживания клиентов оператора сотовой связи, удовлетворяющая всем указанным требованиям.

Во время выполнения работы были закреплены знания о структурах данных и алгоритмах их обработки, а также получены практические навыки их использования при разработке программ.

Для организации данных использовались такие структуры данных, как хеш-таблицы, АВЛ-деревья, односвязные списки.

7. Список использованной литературы

1. Матьяш В.А., Рогачев С.А. Алгоритмы и структуры данных: Учебное пособие / СПбГУАП. СПб., 2021.
2. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.
3. Седжвик Р. Алгоритмы на C++: анализ структуры данных, сортировка, поиск, алгоритмы на графах. М.: Вильямс, 2014. 1056 с
4. Матьяш В. А., Путилов В. А., Фильчаков В. В. Структуры и алгоритмы обработки данных. Апатиты, 2000. 80 с
5. Кнут Д. Искусство программирования: в 3 т. Т. 1. Основные алгоритмы. М.: Вильямс, 2014. 720 с.

Приложение А.

Текст программы:

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <iostream>
#include <iomanip>
#include <locale.h>
#include <string>
#include <regex>
using std::to_string;
using std::cin;
using std::regex;
using std::getline;
using std::cout;
using std::string;
using std::endl;
using std::setw;
using std::left;
regex nameing("([А-Я])([а-я]{0,45})([\\s])(([А-Я])[а-я]{0,45})([\\s])(([А-Я])[а-я]{0,45})");
//регулярное выражение ФИО
regex pasporting("([1-9])([0-9]{3})[-]([0-9]{6})");                    //регулярное
выражение номера паспорта
regex number("([1-9])([0-9])([0-9])[-]([0-9]){7}");                    //регулярное
выражение номера симки



string getDateRight() {
    string one, d, e, f;
    int a, b, c;
    cout << "Год: ";
    do {
        cin >> a;
        if ((a <= 2020) || (cin.fail()) || (a > 2022)) { //проверка на ввод года
            cout << "Неправильно введен год. Повторите ввод." << endl;
            cin.clear();
            while (cin.get() != '\n');
        }
        else {
            f = to_string(a);
            break;
        }
    } while (true);
    cout << "Месяц: ";
    do {
        cin >> b;
        if (b <= 0 || cin.fail() || b > 12) { //проверка на месяц
            cout << "Неправильно введен месяц. Повторите ввод." << endl;
```

```
            cin.clear();
            while (cin.get() != '\n');
        }
        else {
            e = to_string(b);
            if (e.length() == 1) {
                e = '0' + e;
            }
            break;
        }
    } while (true);
    cout << "День: ";
    do {
        cin >> c;
        if (((cin.fail() || c <= 0 || c > 31) && (b == 1 || b == 3 || b == 5 || b == 7 || b == 8 || b == 10 || b == 12)) || ((cin.fail() || c <= 0 || c > 30) && (b == 4 || b == 6 || b == 9 || b == 11)) || ((cin.fail() || c <= 0 || c > 29) && (b == 2) && (a % 4 == 0)) || ((cin.fail() || c <= 0 || c > 28) && (b == 2) && (a % 4 != 0))) {
            cout << "Неправильно введен день. Повторите ввод." << endl; //проверка дня по месяцу и году
            cin.clear();
            while (cin.get() != '\n');
        }
        else {
            d = to_string(c);
            if (d.length() == 1) {
                d = '0' + d;
            }
            one = d + '.' + e + '.' + f;
            return one;
            break;
        }
    } while (true);
}
string trim(const string& str)
{
    size_t first = str.find_first_not_of(' ');
    if (string::npos == first)
    {
        return str;
    }
    size_t last = str.find_last_not_of(' ');
    return str.substr(first, (last - first + 1));
}
// проверка имени
string namecheck(string name) {
    name = trim(name);
    while (cin.fail() || ((regex_match(name, nameing) == false))) {
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail(), '\n');
        cout << "Неверно!" << endl;
        cout << "Введите ФИО заново: " << endl;
```

```
        getline(cin, name);
        name = trim(name);
    }
    return name;
}
// проверка паспорта
string pasportcheck(string pasport) {
    pasport = trim(pasport);
    while (cin.fail() || ((regex_match(pasport, pasporting) == false))) {
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail(), '\n');
        cout << "Неверно!" << endl;
        cout << "Введите серию и номер паспорта заново: " << endl;
        getline(cin, pasport);
        pasport = trim(pasport);
    }
    return pasport;
}
string numbercheck(string sim) {
    sim = trim(sim);
    while (cin.fail() || ((regex_match(sim, number) == false))) {
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail(), '\n');
        cout << "Неверно!" << endl;
        cout << "Введите номер сим-карты заново: " << endl;
        getline(cin, sim);
        sim = trim(sim);
    }
    return sim;
}
// проверка года
int yearcheck(int year) {
    while (cin.fail() || (year <= 1915)) {
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail(), '\n');
        cout << "Неверно!" << endl;
        cout << "Введите дату заново:" << endl;
        cin >> year;
    }
    return year;
}
bool Dateeq(string in, string out) {
    if (in[9] < out[9]) {
        return 1;
    }
    else if (in[9] > out[9]) {
        cout << "Неправильное сочетание дат выдачи - окончания действия.\n Регистрация
клиента пропущена." << endl;
        return 0;
    }
    else {
        if (in[3] < out[3]) {
```

```
            return 1;
        }
        else if (in[3] > out[3]) {
            cout << "Неправильное сочетание дат выдачи - окончания действия.\n Регистрация
клиента пропущена." << endl;
            return 0;
        }
        else if (in[3] == out[3]) {
            if (in[4] < out[4]) {
                return 1;
            }
            else if (in[4] > out[4]) {
                cout << "Неправильное сочетание дат выдачи - окончания действия.\n
Регистрация клиента пропущена." << endl;
                return 0;
            }
            else if (in[4] == out[4]) {
                if (in[0] < out[0]) {
                    return 1;
                }
                else if (in[0] > out[0]) {
                    cout << "Неправильное сочетание дат выдачи - окончания действия.\n
Регистрация клиента пропущена." << endl;
                    return 0;
                }
                else if (in[0] == out[0]) {
                    if (in[1] < out[1]) {
                        return 1;
                    }
                    else if (in[1] > out[1]) {
                        cout << "Неправильное сочетание дат выдачи - окончания действия.\n
Регистрация клиента пропущена." << endl;
                        return 0;
                    }
                    else if (in[1] == out[1]) {
                        cout << "Неправильное сочетание дат выдачи - окончания действия.\n
Регистрация клиента пропущена." << endl;
                        return 0;
                    }
                }
            }
        }
    }
    ////// return
    return 0;
}
// для АВЛ дерева упорядоченному по паспорту
struct client {
        string name; //ФИО
        string passport; //паспорт
        int year; //год рождения
        string adds; //адрес
```

```
        string place; // место и дата выдачи паспорта
        client(string a, string b, string c, string d, int g);
        client();
};

client::client() {
        name = "";
        passport = "";
        year = NULL;
        adds = "";
        place = "";
}

client::client(string a, string b, string c, string d, int g) {
        name = a;
        passport = b;
        year = g;
        adds = c;
        place = d;
}

client addclient() {

    string n, p, a, pl;
    int y;
    cout << "Введите ФИО" << endl;
    getline(cin, n, '\n');
    n = namecheck(n);
    cout << "Введите паспорт" << endl;
    getline(cin, p, '\n');
    p = pasportcheck(p);
    cout << "Введите год рождения" << endl;
    cin >> y;
    y = yearcheck(y);
    cin.ignore();
    cout << "Введите адрес" << endl;
    getline(cin, a, '\n');
    cout << "Введите место выдачи паспорта" << endl;
    getline(cin, pl, '\n');
    client k(n, p, a, pl, y);
    return k;


}
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

struct node // структура для представления узлов дерева
{
    client key;
    unsigned char height;
```

```
   node* left;
   node* right;
   node(client k) { key = k; left = right = 0; height = 1; }
};

unsigned char height(node* p)
{
   return p ? p->height : 0;
}

int bfactor(node* p)
{
   return height(p->right) - height(p->left);
}

void fixheight(node* p)
{
   unsigned char hl = height(p->left);
   unsigned char hr = height(p->right);
   p->height = (hl > hr ? hl : hr) + 1;
}

void print_tree(node* p, int level) //рекурсивная функция вывода дерева
{
   if (p)
   {
      print_tree(p->left, level + 1);
      for (int i = 0; i < level; i++)
         cout << "   ";
      print_tree(p->right, level + 1);
   }
}

void walk(node* p)
{
   if (!p)
   {
      return;
   }

   else
   {
      cout << setw(30) << p->key.name << setw(19) << p->key.passport << setw(21) << p-
>key.year;

      cout << setw(20) << p->key.adds << p->key.place << endl;
      walk(p->left);
      walk(p->right);
   }
}

node* rotateright(node* p) // правый поворот вокруг p
```

```
{
    node* q = p->left;
    p->left = q->right;
    q->right = p;
    fixheight(p);
    fixheight(q);
    return q;
}

node* rotateleft(node* q) // левый поворот вокруг q
{
    node* p = q->right;
    q->right = p->left;
    p->left = q;
    fixheight(q);
    fixheight(p);
    return p;
}

node* balance(node* p) // балансировка узла p
{
    fixheight(p);
    if (bfactor(p) == 2)
    {
        if (bfactor(p->right) < 0)
            p->right = rotateright(p->right);
        return rotateleft(p);
    }
    if (bfactor(p) == -2)
    {
        if (bfactor(p->left) > 0)
            p->left = rotateleft(p->left);
        return rotateright(p);
    }
    return p; // балансировка не нужна
}

node* insert(node* p, client k) // вставка ключа k в дерево с корнем p
{
    if (!p) return new node(k);
    if (k.passport < p->key.passport)
        p->left = insert(p->left, k);
    else if (k.passport > p->key.passport) {
        p->right = insert(p->right, k);
    }
    else
    {
        cout << "Такой номер паспорта уже существует." << endl;
    }
    return balance(p);
}
```

```
node* findmin(node* p) // поиск узла с минимальным ключом в дереве p
{
   return p->left ? findmin(p->left) : p;
}

node* removemin(node* p) // удаление узла с минимальным ключом из дерева p
{
   if (p->left == 0)
      return p->right;
   p->left = removemin(p->left);
   return balance(p);
}

node* remove(node* p, string k) // удаление ключа k из дерева p
{
   if (!p) return 0;
   if (k < p->key.passport)
      p->left = remove(p->left, k);
   else if (k > p->key.passport)
      p->right = remove(p->right, k);
   else
   {
      node* q = p->left;
      node* r = p->right;
      delete p;
      if (!r) return q;
      node* min = findmin(r);
      min->right = removemin(r);
      min->left = q;
      return balance(min);
   }
   return balance(p);
}

node* removeall(node* p) // удаление ключа k из дерева p
{
   // node *h
   while ((p->left != NULL) && (p->right != NULL)) {
      removemin(p);
   }
   delete p;
   p = NULL;
   cout << "Клиенты очищены " << endl;
   return p;
}


void findclient(node* p, string a) {
   if (p != NULL) {
      if (p->key.passport == a) {
         cout << "ФИО клиента: " << p->key.name << endl;
         cout << "Паспорт клиента: " << p->key.passport << endl;
```

```
            cout << "Год рождения: " << p->key.year << endl;
            cout << "Город проживания: " << p->key.adds << endl;
            cout << "Место выдачи пасспорта: " << p->key.place << endl;
        }

        findclient(p->left, a);
        findclient(p->right, a);
    }
}
// поиск
void BMSearch(node* p, string text)
{
    static int flagos = 0;
    if (!p)
    {
        return;
    }
    else
    {
        int N = text.length();
        int sample[30] = { 0 };
        int lastChar;
        int shiftIndex;
       string txt = p->key.adds;
        int textLen = txt.length();
        for (int i = N - 2; i >= 0; i--)
        {
            sample[i] = N - i - 1;
            for (int j = N - 2; j > i; j--)
            {
                if (text[i] == text[j])
                {
                    sample[i] = sample[j];
                    break;
                }
            }
        }
        sample[N - 1] = N;
        for (int i = N - 1; i > -1; i--)
        {
            if (text[N - 1] == text[i])
            {
                sample[N - 1] = sample[i];
                break;
            }
        }

        int i = 0;
        int j = N - 1;
        lastChar = N - 1;
        while (j + i < textLen)
        {
```

```
            if (txt[j + i] != text[j] || j == 0)
            {
               shiftIndex = -1;
               for (int k = 0; k < N; k++)
               {
                  if (lastChar == text[k])
                  {
                     shiftIndex = k;
                     break;
                  }
               }
               if (shiftIndex == -1)
               {
                  i += N;
               }
               else
               {
                  i += sample[shiftIndex];
               }
               j = N - 1;
               if (j + i < textLen)
               {
                  lastChar = txt[j + i];
               }
               else
               {
                  break;
               }
            }
            else
            {
               j--;
               if (j == 0)
               {
                  cout << "ФИО: " << p->key.name << "\t" << "Паспорт: " << p->key.passport <<
"\t" << "Год рождения: " << p->key.year << "\t" << "Место жительства: " << p->key.place <<
"\t" << "Место выдачи: " << p->key.adds <<endl;
                  flagos++;
               }
            }
         }
      }
      BMSearch(p->left, text);
      BMSearch(p->right, text);
}
//////////////////////////////////////////////////////////////////////////////

bool check_tree_pass(string a, node* p) {
   while (p != NULL) {
      if (a < p->key.passport)
         p = p->left;
      else if (a > p->key.passport) {
```

```
            p = p->right;
        }
        else {
            if (a == p->key.passport) {
                return 1;
            }
        }
    }
    return 0;
}

// для ХЭШ таблицы
struct simcard {

            string number; // номер
            string price; // тариф
            int year; // год выпуска
            bool hasSim; // признак наличия
    simcard(string a, string b, int d, bool g);
    simcard();
};
simcard::simcard() {
    number = "";
    price = "";
    year = NULL;
    hasSim = NULL;
}
simcard::simcard(string a, string b, int d, bool g) {
    number = a;
    price = b;
    year = d;
    hasSim = g;

}

simcard addsim() {

    string n, p,   pl;
    bool a;
    int y;
     cout << "Введите номер" << endl;
    getline(cin, n, '\n');
    n = numbercheck(n);
    cout << "Введите тариф" << endl;
    getline(cin, p, '\n');
    cout << "Введите год выпуска" << endl;
    cin >> y;
    y = yearcheck(y);
    cin.ignore();
    cout << "Введите наличие (1 - да; 0 - нет) " << endl;
    cin >> a;
    cin.ignore();
```

```
    simcard newsim(n, p, y, a);
    return newsim;

}
int HashFunc(string a) {
    int size = 500;
    int value = 1;
    for (int i = 0; i < 4; i++) {
        value += (pow(int(a[i]), 2) * sqrt(value / 2)) / (size / 2);
    }
    value = value % size;
    return value;
}
int Hash_Function2(string a, int i) {
    int sum = 0;
    int size = 500;
    for (int j = 0; j < 4; j++) {
        sum += pow(int(a[j]), 3);
        if (i % 2 == 0)
            sum = sum % (i / 2);
    }
    sum %= size;
    return (i * sum);
}

struct HashCell {
    simcard a;
    HashCell(simcard g);
};

HashCell::HashCell(simcard g) {
    a.number = g.number;
    a.price = g.price;
    a.hasSim = g.hasSim;
    a.year = g.year;
}
// хэш таблица
struct HashTable {
    HashCell** table;
public:
    HashTable();
    void addcell(simcard a);
    void printsims(string a);
    void deleteelem(string a);
    void clear();
    void print();
    simcard findnumber(string a);
};

// поиск номера
simcard HashTable::findnumber(string a)
```

```
{
    int s = 0;
    int g = HashFunc(a);
    int temp = g;
    while (table[g] != NULL)
    {
        if (table[g]->a.number == a)
        {
            return table[g]->a;
        }
        else
        {
                g = ((temp + Hash_Function2(a, g) + 1) % 500);
        }
    }
    simcard A;
    return A;
}
// поиск тарифа

// очистка таблицы
void HashTable::clear() {
    for (int i = 0; i < 500; i++) {
        table[i] = NULL;
    }
    cout << "Сим-карты очищены" << endl;
}

void HashTable::print() {
    int s = 0;
    for (int i = 0; i < 500; i++) {
        if (table[i] != NULL && table[i]->a.number != "DEL") {
            s++;
        }
    }
    if (s != 0) {
        cout << endl;
        cout << setw(40) << left << "Номер Sim ";
        cout << setw(19) << "Тариф ";
        cout << setw(21) << "Год выпуска";
        cout << setw(25) << "Наличие";

        for (int i = 0; i < 500; i++) {
            if (table[i] != NULL && table[i]->a.number != "DEL") {
                cout << setw(40) << left << table[i]->a.number
                    << setw(19) << table[i]->a.price
                    << setw(21) << table[i]->a.year
                    << setw(25) << table[i]->a.hasSim << endl;
            }
        }
    }
    else {
```

```
        cout << "Список SIM пуст." << endl;
    }
    return;
}
void HashTable::printsims(string a) {
    int s = 0;
    for (int i = 0; i < 500; i++) {
        if (table[i] != NULL && table[i]->a.price != "DEL") {
            s++;
        }
    }
    if (s != 0) {
        cout << endl;
        cout << setw(40) << left << "Номер Sim ";
        cout << setw(19) << "Тариф ";
        cout << setw(21) << "Год выпуска";
        cout << setw(25) << "Наличие";

        for (int i = 0; i < 500; i++) {
            if (table[i] != NULL && table[i]->a.price != "DEL" and table[i]->a.price == a) {
                cout << setw(40) << left << table[i]->a.number
                    << setw(19) << table[i]->a.price
                    << setw(21) << table[i]->a.year
                    << setw(25) << table[i]->a.hasSim << endl;
            }
        }
    }
    else {
        cout << "Список SIM пуст." << endl;
    }
    return;
}
// добавление
void HashTable::addcell(simcard a) {
    int g = 0;
    int s = HashFunc(a.number);
    int temp = s;
    while (true) {
        if (table[s] == NULL) {
            table[s] = new HashCell(a);
            return;
        }
        else {
            g++;
            s = ((temp + g* Hash_Function2(a.number, g) + 1) % 500);
        }

    }
}
// удаление элемента
void HashTable::deleteelem(string a) {
    int s = 0;
```

```
    int g = HashFunc(a);
    while (table[g] != NULL) {
        if (table[g]->a.number == a) {
            cout << "SIM с номером - " << table[g]->a.number << " удалена." << endl;
            table[g]->a.number = "DEL";
            table[g]->a.price = "DEL";
            table[g]->a.year = NULL;
            table[g]->a.hasSim = NULL;
            return;
        }
        s++;
        g = g + 2 * s + 3 * pow(s, 2);
    }
    return;
}

HashTable::HashTable() {
    table = new HashCell * [500];
    for (int i = 0; i < 500; i++) {
        table[i] = NULL;
    }
}
bool checknum(string a, HashTable b) {
    for (int i = 0; i < 500; i++) {
        if (b.table[i] != NULL) {
            if (b.table[i]->a.number == a) {
                return 1;
            }
        }
    }
    return 0;
}
struct data_sim {

    string npass;
    string nsim;
    string datestart;
    string datestop;
    data_sim(string a, string b, string c, string d);
};

data_sim::data_sim(string a, string b, string c, string d) {
    npass = a;
    nsim = b;
    datestart = c;
    datestop = d;
}
// список Данные о вселении или выселении
struct list {
public:
    data_sim* one;
    struct list* next;
```

```
public:
   list();
};

struct simlist {
public:
   list* head;
   int count;
public:
   simlist();
   void add(data_sim a);
   void sort();
   void printsims();
   void deletelem(string a);
   int findnum(string a);
   string findpass(string a);
   string findname(string a);
};

simlist::simlist() {
   count = 0;
   head = NULL;
}

void simlist::add(data_sim a) {
   data_sim* one = new data_sim(a);
   one->npass = a.npass;
   one->nsim = a.nsim;
   one->datestart = a.datestart;
   one->datestop = a.datestop;
   if (head == NULL) {
      list* temp;
      temp = (struct list*)malloc(sizeof(struct list));
      head = (struct list*)malloc(sizeof(struct list));
      head->one = one;
      head->next = NULL;
      count++;
   }
   else {
      // *f
      list* p, * t,   * temp;
      temp = (struct list*)malloc(sizeof(struct list));
      t = (struct list*)malloc(sizeof(struct list));
      if (count >= 2) {
         t = head;
      }
      while (head->next != NULL) {
         head = head->next;
      }
      p = head->next;
      head->next = temp;
      temp->next = NULL;
```

```
        temp->one = one;
        if (count >= 2) {
            head = t;
        }
        count++;
    }
}
// sort включением
void simlist::sort() {
    list* temp, * func, * s;
    temp = (struct list*)malloc(sizeof(struct list));
    func = (struct list*)malloc(sizeof(struct list));
    temp = head;
    string a, b;
    int c = 1, j = 0;

    while (true) {
        while (head->next != NULL && head->one->nsim < head->next->one->nsim) {
            head = head->next;
            c++;
        }
        if (c == count) {
            head = temp;
            break;
        }
        else {
            func = head->next;
            head = temp;

            while (func->one->nsim  > head->next->one->nsim) {
                head = head->next;
                c = 1;
            }
            if (func->one->nsim < head->one->nsim) {
                head = temp;
                s = head;
                head = func;
                temp = func;
                func->next = s;
                head = temp;
                for (int i = 0; i < count - 1; i++) {
                    head = head->next;
                }
                head->next = NULL;
                head = temp;
                break;
            }
            head->next = func;
            c = 1;
            head = temp;
            for (int i = 0; i < count - 1; i++) {
                head = head->next;
```

```
            }
            head->next = NULL;
            head = temp;
        }
    }
    while (head->next != NULL) {
        head = head->next;
    }
    head = temp;
    char l;
    int r = 1;
    l = head->one->nsim[0];
    while (true) {
        while ((head->next != NULL) && (l == head->next->one->nsim[0])) {
            r++;
            head = head->next;
        }
        if (head->next != NULL) {
            if (head->next->one->nsim[0] != l) {
                l = head->next->one->nsim[0];
                func = head->next;
                r++;
            }
        }
        if (r == count) {
            head = temp;
            break;
        }
    }
}

void simlist::printsims() {
    struct list* temp;
    int i = 1; //номер вывода для таблички
    temp = head;   //передача указателя на голову
    if (count > 0 and temp != nullptr) {    //если размер БД > 0, вывод возможен
        cout << " | Номер сим   |   Паспорт   |   Дата начала   |   Дата окончания  |" << endl;
        do {
            cout << " | " << temp->one->nsim << " | " << temp->one->npass << " |   " << temp->one-
>datestart << "    |    " << temp->one->datestop << "     | " << endl;
            temp = temp->next;
        } while (temp != NULL);   //пробег по списку и вывод
    }
    else {    //вывод при пустой БД
        cout << "База данных пуста." << endl;
    }
}

void simlist::deletelem(string a) {
    struct list* temp, * p;
    p = (struct list*)malloc(sizeof(struct list));
    temp = head;
```

```
    if (head->one->npass == a) {
        head = head->next;
        return;
    }
    while (head != NULL) {
        if (head->one->npass == a) {
            p = head->next;
        }
        head = head->next;
    }
    head = temp;
    while (head != NULL) {
        // head -> next -> one
        if (head->one->npass == a) {
            head->next = p;
            head = temp;
            count--;
            return;
        }
    }
}

string simlist::findpass(string a) {
    list* temp;
    temp = head;
    int s = 0;
    while (head != NULL) {
        if (head->one->npass == a) {
            return  head->one->nsim;
            cout << "Номер" << head->one->nsim << "удален" << endl;
        }
        head = head->next;
    }
    head = temp;
    return "";
}

string simlist::findname(string a) {
    list* temp;
    temp = head;
    int s = 0;
    while (head != NULL) {
        if (head->one->npass == a) {
            return head->one->nsim;
        }
        head = head->next;
    }
    head = temp;
    return "";
}

int simlist::findnum(string a) {
```

```
    list* temp;
    temp = head;
    int s = 0;
    while (head != NULL) {
        if (head->one->nsim == a) {
            s++;
        }
        head = head->next;
    }
    head = temp;
    return s;
}

bool check_life(node* p, simlist h, string a) {
    while (p != NULL) {
        if (a < p->key.name)
            p = p->left;
        else if (a > p->key.name) {
            p = p->right;
        }
        else {
            if (a == p->key.name) {

                return 1;
            }
                else {
                    return 0;
                }
            }
        }
    return 0;
    }


bool checknumbersim (string a, HashTable b) {
    for (int i = 0; i < 500; i++) {
        if (b.table[i] != NULL) {
            if (b.table[i]->a.number== a) {
                return 1;
            }
        }
    }
    return 0;
}


void menu() {
            cout << "\t Меню\n";
    // Клиент - АВЛ
    // SIM - хэщ
    // выдача в список
            cout << "1)  регистрацию нового клиента"  << endl;
```

```
        cout << "2)  снятие с обслуживания клиента " << endl;
        cout << "3)  просмотр всех зарегистрированных клиентов" << endl;
        cout << "4)  очистку данных о клиентах" << endl;
        cout << "5)  поиск клиента по  номеру паспорта " << endl;
        cout << "6)  поиск клиента по фрагментам ФИО или адреса" << endl; // доработать
нормально
   // начинается ХЭШ
        cout << "7)  добавление новой SIM - карты" << endl;
        cout << "8)  удаление сведений о SIM - карте" << endl;
        cout << "9)  просмотр всех имеющихся SIM - карт" << endl;
        cout << "10) очистку данных о SIM - картах" << endl;
        cout << "11) поиск SIM - карты по номеру  " << endl;
        cout << "12) поиск SIM - карты по тарифу" << endl;
   // список
        cout << "13) регистрацию выдачи клиенту SIM - карты" << endl;
        cout << "14) регистрацию возврата SIM - карты " << endl;
   cout << "15) вывод зарегестрированных клиентов и SIM-карт " << endl;
        cout << " 0) выход " << endl;
   // 5  Результаты поиска – все сведения о найденном клиенте и номера SIM - карт,
которые ему выданы;
   // 6  Результаты поиска – список найденных клиентов с указанием номера
паспорта,ФИО и адреса;
   //11  Результаты поиска – все сведения о найденной SIM - карте, а также ФИО и номер
паспорта клиента, которому выдана эта SIM - карта;
   //
   //12 Результаты поиска – список найденных SIM - карт с указанием «номера SIM -
карты», тарифа, года выпуска;
}

int main() {
   node* tree = NULL;
   HashTable Table;
   client clientS;

   simcard num;
   simcard find;

   simlist simns;

   string cl_name;
   string cl_pass;
   string cl_find;
   string str_a, str_b, str_c, str_d;
   string delsim;
   string simpass;

   bool Ldate;
        setlocale(0, "rus");
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        int select;
        bool check = true;
```

```
    while (check) {
            menu();
            cin >> select;
            cin.ignore();

switch (select) {
case 0:
    cout << "The end";
    check = false;
    break;
    ////////////////////////////////////////////////////////////////////////////////
case 1:
    clientS = addclient();
    tree = insert(tree, clientS);
    cout << "Клиент добавлен" << endl;
    break;

    ////////////////////////////////////////////////////////////////////////////////
case 2:

    cout << "Введите паспорт клиента:";
    getline(cin, cl_name, '\n');
    cl_name = pasportcheck(cl_name);
    str_b = simns.findpass(cl_name);
    if (str_b != "" and simns.head != NULL) {
        cout << "Клиени зарегистрирован. Удалить невозможно." << endl;
        continue;
    }
    else {
        tree = remove(tree, cl_name);

        cout << " Клиент удален " << endl;
    }
    break;
    ////////////////////////////////////////////////////////////////////////////////
case 3:
    cout << "-----------------------------------------------" << endl;
    cout << setw(30) << left << "ФИО";
    cout << setw(19) << "Паспорт";
    cout << setw(21) << "Год рождения";
    cout << setw(13) << "Адрес";
    cout << setw(100) << "Место выдачи паспорта" << endl << endl;
    walk(tree);
    cout << "-----------------------------------------------" << endl << endl;
    break;
    ////////////////////////////////////////////////////////////////////////////////
case 4:
    // сделано : нужна коррекция кода для списка
    if (tree == NULL) {
        cout << "Клиентов нет" << endl;
    }
```

```
            else {
                if (simns.count != 0) {
                    cout << "Существуют зарегистрированные клиенты. Очистка не может быть
произведена." << endl;
                    continue;
                }
                tree = removeall(tree);
            }
            break;
            ////////////////////////////////////////////////////////////////////////////
        case 5:

            if (tree == NULL) {

                cout << "Клиенты отсутствуют." << endl;
            }
            else {
                cout << "Введите номер паспорта:";
                getline(cin, cl_pass, '\n');
                cl_pass = pasportcheck(cl_pass);
                findclient(tree, cl_pass);
                str_d = simns.findpass(cl_pass);
                cout << endl << endl;

                    if (str_d != "") {

                        cout << str_d << endl;
                    }
                    else {
                        cout << "Сим-карт нет" << endl;
                    }


            }
            cout << endl;
            break;
            ////////////////////////////////////////////////////////////////////////////
        case 6:

            ////////////////////////////////////////////////////////////////////////////
            cout << "Введите строку :" << endl;
            getline(cin, cl_find, '\n');
            BMSearch(tree, cl_find);
            break;
            ////////////////////////////////////////////////////////////////////////////

        case 7:
            num = addsim();
            if (checknum(num.number, Table) == 1)
            {
                cout << "Ошибка! симкарта  уже есть в базе. Добавление пропущено." << endl;
            }
```

```
            else {
               Table.addcell(num);
            }
            break;
            ////////////////////////////////////////////////////////////////////////////////
         case 8:
            cout << "Введите номер SIM для удаления" << endl;
            getline(cin, delsim, '\n');
            delsim = numbercheck(delsim);
            str_b = simns.findpass(delsim);
            if (str_b != "" && simns.head != NULL) {
               cout << "Сим карта выдана. Удалить невозможно." << endl;
               continue;
            }
            else {
               Table.deleteelem(delsim);
            }
            break;
            ////////////////////////////////////////////////////////////////////////////////
         case 9:
            Table.print();
            break;
            ////////////////////////////////////////////////////////////////////////////////
         case 10:

            if (simns.count != 0) {
               cout << "Существуют зарегестрированные клиенты. Удаление невозможно." <<
endl;
               continue;
            }
            else {

            Table.clear();
         }
            break;
            ////////////////////////////////////////////////////////////////////////////////
                     case 11:
            // доработка со списком - поиск клиента с этим номером
            cout << "Введите номер SIM для поиска" << endl;
            getline(cin, delsim, '\n');
            delsim = numbercheck(delsim);

            find = Table.findnumber(delsim);
            if (num.number == "" and  num.year== 0) {
               cout << "симкарты не существует" << endl;
            }
            else {
               cout << "Номер СИМ " << find.number << "\nТариф " << find.price << "\nГод
выпуска " << find.year << "\nНаличие " << find.hasSim << endl;

            }
```

```
break;
////////////////////////////////////////////////////////////////////////////////
        case 12:
cout << "Введите тариф для поиска" << endl;
getline(cin, delsim, '\n');
Table.printsims(delsim);
if (num.number == "" and num.year == 0) {
   cout << "симкарта не найдена" << endl;

}
break;
// список
////////////////////////////////////////////////////////////////////////////////
        case 13:


cout << "Введите номер паспорта: ";
getline(cin, str_a, '\n');
str_a = pasportcheck(str_a);
if (check_tree_pass(str_a, tree) == 0) {
   cout << "Клиент не зарегистрирован. Выдача отменена." << endl;
   continue;
}

cout << "Введите номер сим карты: ";
getline(cin, str_b, '\n');
str_b = numbercheck(str_b);
find = Table.findnumber(str_b);
if (num.number == "") {
   cout << "Сим карта отсутствует. Выдача отменена." << endl;
   continue;
}

cout << "Введите дату начала активации: " << endl;
str_c = getDateRight();
cout << "Введите дату деактивации: " << endl;
str_d = getDateRight();

  Ldate = Dateeq(str_c, str_d);
if (Ldate == 0) {
   continue;

}
else {
   bool o = check_life(tree, simns, str_b);
      data_sim Database(str_a, str_b, str_c, str_d);
      simns.add(Database);
      cout << "Сим-карта выдана: " << endl;
      simns.sort();

}
```

```
            break;
            ////////////////////////////////////////////////////////////////////////////////
                    case 14:
            cout << "Введите паспорт: " << endl;
            getline(cin, str_a, '\n');
            str_a = pasportcheck(str_a);
            str_b = simns.findpass(str_a);
            if (str_b == "") {
               cout << "У клиента нет сим-карт" << endl;
            }
            else {
               simns.deletelem(str_a);
               cout << "Возврат оформлен" << endl;
               if (simns.head != NULL) {
                 // simns.sort();
               }
            }
                            break;
         case 15:
            simns.printsims();

                    }
            }

   return 0;
}
```