МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №3

по дисциплине «Структуры и алгоритмы обработки данных»
по теме
«Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины»

Выполнил:
Студент группы ИКБО-13-22 Козлов Кирилл Игоревич

Проверил: ассистент Муравьёва Е.А.

МОСКВА 2023 г.

Практическая работа № 2

Цель работы

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

Ход работы

Вариант 17

17	Открытый адрес (смещение на 1)	Частотный словарь: слово, количество вхождений в текст.

Задание 1

Формулировка задачи:

Ответьте на вопросы:

1. Расскажите о назначении хеш-фунции: назначение хеш-функции заключается в преобразовании входных данных (ключей) в уникальное числовое значение фиксированной длины, называемое хешем. Основная цель хеш-функции - равномерно распределить данные по хеш-таблице и обеспечить быстрый доступ к данным.

2. Что такое коллизия: коллизия в хеш-таблице возникает, когда двум разным ключам соответствует один и тот же хеш-значение. То есть два различных ключа возвращают одинаковый индекс в хеш-таблице. Это может произойти из-за ограниченного размера хеш-таблицы и свойства хеш-функции.

3. Что такое «открытый адрес» по отношению к хеш-таблице: открытый адрес" в хеш-таблице относится к методу разрешения коллизий, при котором, если возникает коллизия, пробуется найти следующую доступную свободную ячейку (открытое место) в хеш-таблице для вставки элемента с использованием различных стратегий поиска.

4. Как в хеш-таблице с открытым адресом реализуется коллизия: в хеш-таблице с открытым адресом коллизия решается путем поиска следующей доступной свободной ячейки для вставки элемента (пробирования), используя такие стратегии, как линейное пробирование (последовательное исследование

ячеек), квадратичное пробирование (исследование ячеек с помощью квадратичной функции), двойное хеширование (вычисление второй хеш-функции для определения шага пробирования).

5. Какая проблема, может возникнуть после удаления элемента из хештаблицы с открытым адресом и как ее устранить: после удаления элемента из хеш-таблицы с открытым адресом может возникнуть проблема «пустых мест», которые нарушают равномерную расстановку элементов в таблице, что может замедлить производительность поиска. Эту проблему можно решить с помощью специальных маркеров или методов удаления, которые поддерживают интегритет таблицы.

6. Что определяет коэффициент нагрузки в хеш-таблице: коэффициент нагрузки в хеш-таблице определяет отношение количества элементов (заполненных ячеек) к размеру таблицы. Он вычисляется как отношение числа элементов к общему размеру таблицы. Коэффициент нагрузки позволяет оценить заполненность хеш-таблицы и выбрать оптимальный размер таблицы для обеспечения эффективного поиска**.**

7. Что такое «первичный кластер» в таблице с открытым адресом: первичный кластер в таблице с открытым адресом относится к сосредоточению серии занятых ячеек, которые возникают из-за последовательной коллизии хеш-функций. При появлении первичного кластера производительность поиска может снижаться, поскольку требуется больше шагов для поиска свободной ячейки.

8. Как реализуется двойное хеширование: двойное хеширование (double hashing) — это метод разрешения коллизий в хеш-таблице, при котором используется вторичная хеш-функция для определения шага пробирования. Вместо простого увеличения индекса в случае коллизии, вторичная хеш-функция вычисляет шаг пробирования, позволяя элементам распределиться в таблице равномерно и находить свободные ячейки более эффективно.

Задание 2

Формулировка задачи:

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла.

1) Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте.

2) Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.

3) Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.

4) Подготовить тесты (последовательность значений ключей), обеспечивающие:

- вставку ключа без коллизии
- вставку ключа и разрешение коллизии
- вставку ключа с последующим рехешированием
- удаление ключа из таблицы
- поиск ключа в таблице

Примечание. Для метода с открытым адресом подготовить тест для поиска ключа, который размещен в таблице после удаленного ключа, с одним значением хеша для этих ключей.

5) Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.

Решение:

Для решения данной задачи были реализованы следующие функции и структуры: структура записи в бинарном файле, структура элемента хэш-функции, хэш-функция, возвращающая остаток от деления ключа в записи на текущий размер таблицы, рекурсивная функция добавления элемента в хэш-

таблицу, реализующую смещение на 1, если ячейка занята

```
struct wordRecord
{
    char word[20];
    unsigned short int amountOfEntry;

public:
    wordRecord(std::string userWord, unsigned short int userAmountOfEntry)
    {
        for (int i = 0; i < userWord.size(); i++)
        {
            word[i] = userWord[i];
        }
        this->amountOfEntry = userAmountOfEntry;
    }
    wordRecord()
    {
        amountOfEntry = -1;
    }
};
//----Хэш-функция----//
unsigned short hashFunc(unsigned short int key, int tableSize)
{
    return(key % tableSize);
}
//----Структура записи в хэш-таблице----//
struct hashElement
{
    short int amountKey;
    std::streampos offset;

public:
    hashElement(unsigned short int userAmountOfEntry, std::streampos currOffset,
int tableSize)
    {
        this->amountKey = userAmountOfEntry;
        this->offset = currOffset;
    }
    hashElement()
    {
        this->amountKey = -1;
        this->offset = -1;
    }
};
//----Функция вставки элемента в хэш-таблицу----//
void addHashElement(hashElement newHashElement, int newHashIndex, hashElement*
hashTable, int tableSize)
{
    if (hashTable[newHashIndex].offset == -1)
    {
        hashTable[newHashIndex] = newHashElement;
    }
    else if (hashTable[newHashIndex].amountKey != -1)
    {
        addHashElement(newHashElement, newHashIndex + 1, hashTable, tableSize);
    }
}
```

Листинг 1 – функции для решения задания 2

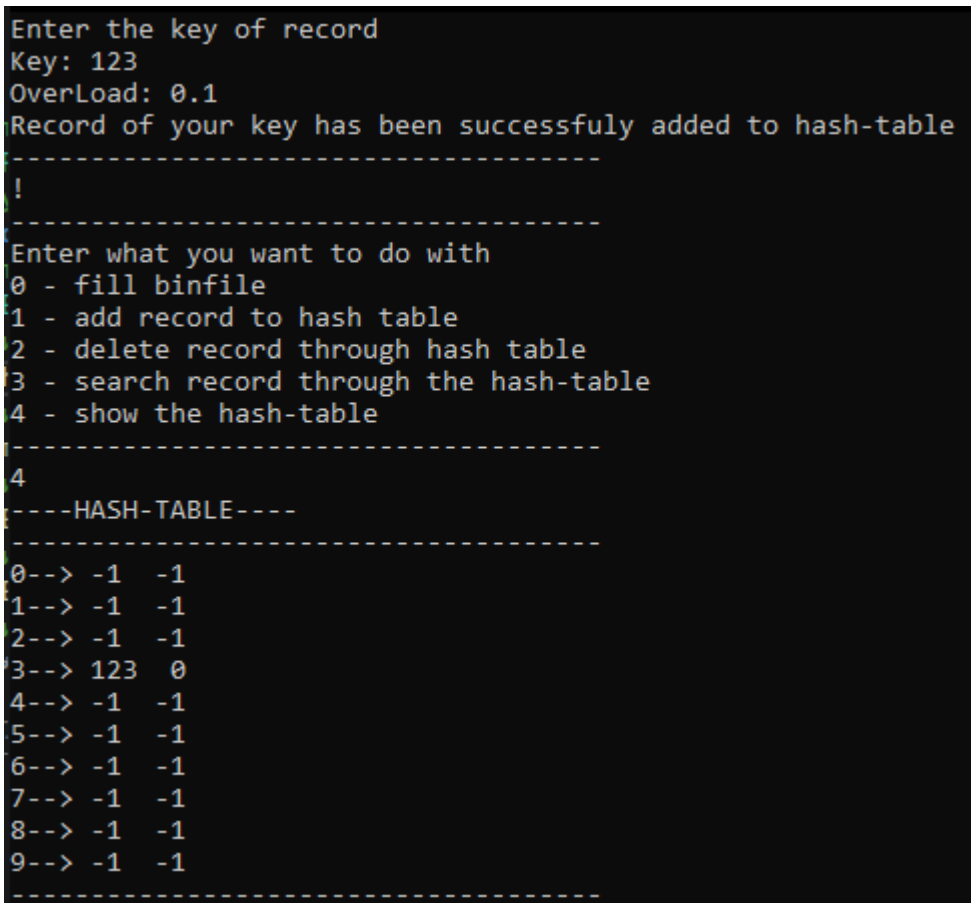

Рисунок 2 – Добавление элемента без коллизии

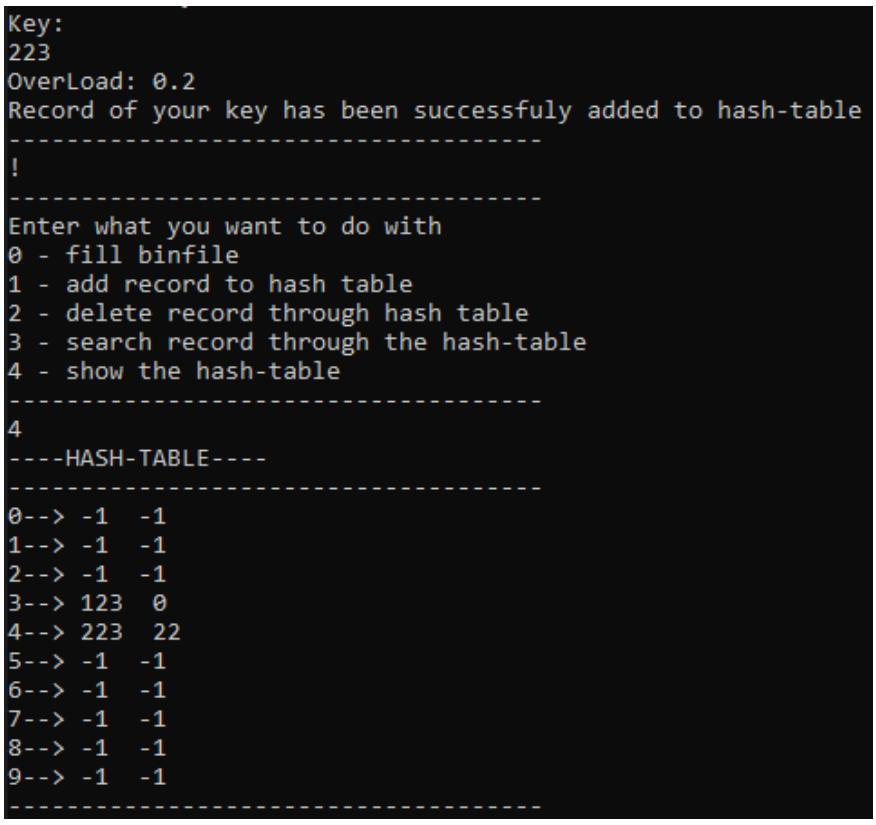

Рисунок 3 – Добавление ключа с решением коллизии

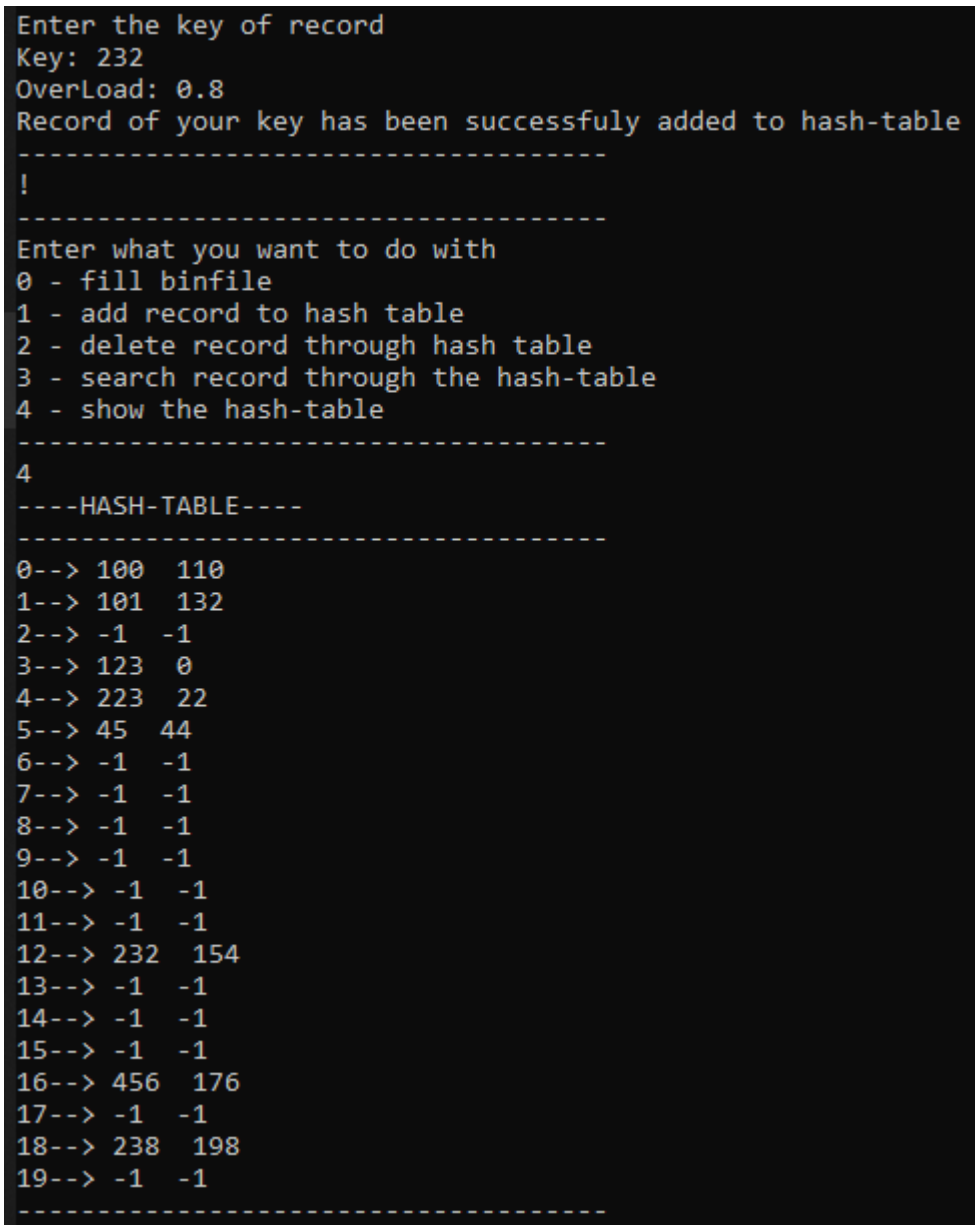

Рисунок 4 – Добавление ключа с последующим рехешированием

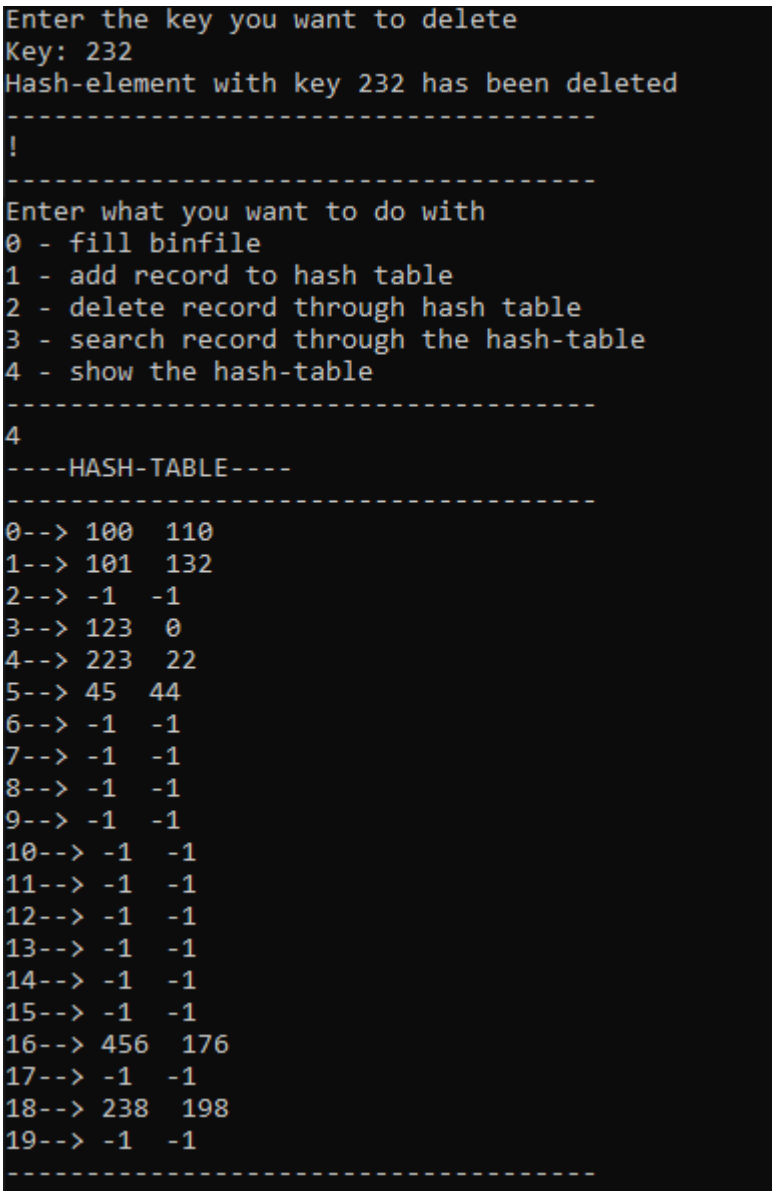

Рисунок 5 – Удаление элемента из хэш-таблицы

```
----HASH-TABLE----
-------------------------------------
0--> 100  110
1--> 101  132
2--> -1  -1
3--> 123  0
4--> 223  22
5--> 45  44
6--> -1  -1
7--> -1  -1
8--> -1  -1
9--> -1  -1
10--> -1  -1
11--> -1  -1
12--> -1  -1
13--> -1  -1
14--> -1  -1
15--> -1  -1
16--> 456  176
17--> -1  -1
18--> 238  198
19--> -1  -1
-------------------------------------
Enter what you want to do with
0 - fill binfile
1 - add record to hash table
2 - delete record through hash table
3 - search record through the hash-table
4 - show the hash-table
-------------------------------------
3
Enter the key you want to search
Key: 238
Hash-element with key 238 has been found
Hash index of element: 18
-------------------------------------
!
-------------------------------------
```

Рисунок 6 – Поиск элемента в хэш-таблице

Полный Листинг задания один представлен ниже.

```cpp
#include <iostream>
#include <fstream>

//WORK WITH HASH-TABLE ONLY//
//WORK WITH HASH-TABLE ONLY//

//----Структура записи в текстовом и бинарном файле----//
struct wordRecord
{
    char word[20];
    unsigned short int amountOfEntry;

public:
    wordRecord(std::string userWord, unsigned short int userAmountOfEntry)
    {
        for (int i = 0; i < userWord.size(); i++)
        {
            word[i] = userWord[i];
        }
        this->amountOfEntry = userAmountOfEntry;
    }
    wordRecord()
    {
        amountOfEntry = -1;
```

```
    }
};
//----Хэш-функция----//
unsigned short hashFunc(unsigned short int key, int tableSize)
{
    return(key % tableSize);
}
//----Структура записи в хэш-таблице----//
struct hashElement
{
    short int amountKey;
    std::streampos offset;

public:
    hashElement(unsigned short int userAmountOfEntry, std::streampos currOffset,
int tableSize)
    {
        this->amountKey = userAmountOfEntry;
        this->offset = currOffset;
    }
    hashElement()
    {
        this->amountKey = -1;
        this->offset = -1;
    }
};
//----Функция вставки элемента в хэш-таблицу----//
void addHashElement(hashElement newHashElement, int newHashIndex, hashElement*
hashTable, int tableSize)
{
    if (hashTable[newHashIndex].offset == -1)
    {
        hashTable[newHashIndex] = newHashElement;
    }
    else if (hashTable[newHashIndex].amountKey != -1)
    {
        addHashElement(newHashElement, newHashIndex + 1, hashTable, tableSize);
    }
}
//----Функция добавления элемента в хэш-таблицу----//
void addElementFunc()
{

}
//----Функция удаления элемента из хэш-таблицы----//
void removeElementFunc()
{

}
//----Функция поиска элемента в хэш-таблице----//
void searchElementFunc()
{

}
//----Функция рехеширования----//
hashElement* reHashFunc(hashElement* hashTable, int tableSize)
{
    hashElement* reHashTable = new hashElement[tableSize*2];
    hashElement reHashElement;
    int newHashIndex;
    for (int i = 0; i < tableSize; i++)
    {
        if(hashTable[i].amountKey != -1)
        {
            reHashElement.amountKey = hashTable[i].amountKey;
```

```
            reHashElement.offset = hashTable[i].offset;
            newHashIndex = hashFunc(hashTable[i].amountKey, tableSize * 2);
            addHashElement(reHashElement, newHashIndex, reHashTable, tableSize *
2);
        }
    }
    return(reHashTable);
}

int main()
{
    double amoutOfOccupiedElements = 0;
    int tableSize = 10;
    int userChoice;
    int newHashIndex;
    double overLoad;
    unsigned short int userAmountOfEntry;
    int userKeyToInsert;
    int userKeyToDelete;
    int userKeyToSearch;
    hashElement* hashTable = new hashElement[tableSize];
    wordRecord toRead;
    std::streampos currOffsetWrite = 0;
    std::streampos currOffsetRead = 0;
    std::string userWord;
    std::string fileName = "wordRecordList";
    std::cout << "Work with hash-table and binary file \n";
    std::cout << "------------------------------------ \n";
    do
    {
        std::cout << "Enter what you want to do with \n";
        std::cout << "0 - fill binfile \n";
        std::cout << "1 - add record to hash table \n";
        std::cout << "2 - delete record through hash table \n";
        std::cout << "3 - search record through the hash-table \n";
        std::cout << "4 - show the hash-table \n";
        std::cout << "------------------------------------ \n";
        std::cin >> userChoice;
        switch (userChoice)
        {
        case 0:
        {
            int recordAmount;
            std::cout << "Enter amount of records \n";
            std::cout << "Amount: ";
            std::cin >> recordAmount;
            std::ofstream fout(fileName, std::ios::binary);
            for (int i = 0; i < recordAmount; i++)
            {
                std::cout << "Enter the word and its aomunt of entry \n";
                std::cout << "Word: ";
                std::cin >> userWord;
                std::cout << "Amount: ";
                std::cin >> userAmountOfEntry;
                wordRecord userWordRecord(userWord, userAmountOfEntry);
                if (!fout.is_open()) { std::cout << "Sorry, can`t find your file
\n"; break; }
                fout.seekp(currOffsetWrite);
                fout.write((char*)&userWordRecord, sizeof(wordRecord));
                currOffsetWrite = fout.tellp();
                std::cout << "CURRENT POS OF WRITE " << currOffsetWrite << "\n";
                std::cout << "Successfuly written to file!! \n";
                std::cout << "------------------------------------ \n";
            }
            fout.close();
```

```
            break;
        }
        case 1:
        {
            std::cout << "Enter the key of record \n";
            std::cout << "Key: ";
            std::cin >> userKeyToInsert;
            std::ifstream fin(fileName, std::ios::binary);
            while (!fin.eof())
            {
                fin.seekg(currOffsetRead);
                fin.read(reinterpret_cast<char*>(&toRead), sizeof(wordRecord));
                if (toRead.amountOfEntry == userKeyToInsert)
                {
                    amoutOfOccupiedElements++;
                    overLoad = amoutOfOccupiedElements / tableSize;
                    std::cout << "OverLoad: " << overLoad << '\n';
                    if (overLoad >= 0.75)
                    {
                        hashTable = (hashElement*)realloc(hashTable,
sizeof(hashElement) * tableSize * 2);
                        hashTable = reHashFunc(hashTable, tableSize);
                        tableSize *= 2;
                    }
                    hashElement toInsert;
                    toInsert.amountKey = toRead.amountOfEntry;
                    toInsert.offset = currOffsetRead;
                    newHashIndex = hashFunc(toInsert.amountKey, tableSize);
                    addHashElement(toInsert, newHashIndex, hashTable, tableSize);
                    currOffsetRead = 0;
                    std::cout << "Record of your key has been successfuly added
to hash-table \n";
                    std::cout << "------------------------------------ \n";
                    break;
                }
                currOffsetRead = fin.tellg();
            }
            currOffsetRead = 0;
            std::cout << "! \n";
            std::cout << "------------------------------------ \n";
            fin.close();
            break;
        }
        case 2:
        {
            std::cout << "Enter the key you want to delete \n";
            std::cout << "Key: ";
            std::cin >> userKeyToDelete;
            for (int i = 0; i < tableSize; i++)
            {
                if (hashTable[i].amountKey == userKeyToDelete)
                {
                    hashTable[i].amountKey = -1;
                    hashTable[i].offset = -1;
                    std::cout << "Hash-element with key " << userKeyToDelete << "
has been deleted \n";
                    std::cout << "------------------------------------ \n";
                    break;
                }
            }
            std::cout << "! \n";
            std::cout << "------------------------------------ \n";
            break;
        }
        case 3:
```

```
        {
            std::cout << "Enter the key you want to search \n";
            std::cout << "Key: ";
            std::cin >> userKeyToSearch;
            for (int i = 0; i < tableSize; i++)
            {
                if (hashTable[i].amountKey == userKeyToSearch)
                {
                    std::cout << "Hash-element with key " << userKeyToSearch << "
has been found \n";
                    std::cout << "Hash index of element: " << i << '\n';
                    std::cout << "------------------------------------ \n";
                    break;
                }
            }
            std::cout << "! \n";
            std::cout << "------------------------------------ \n";
            break;
            break;
        }
        case 4:
        {
            std::cout << "----HASH-TABLE---- \n";
            std::cout << "------------------------------------ \n";
            for (int i = 0; i < tableSize; i++)
            {
                std::cout << i << "--> " << hashTable[i].amountKey << "  " <<
hashTable[i].offset << '\n';
            }
            std::cout << "------------------------------------ \n";
            break;
        }
        }

    } while (userChoice == 0 || userChoice == 1 || userChoice == 2 || userChoice
== 3 || userChoice == 4);
}
```

Листинг 1 – Полный Листинг кода Задания 2

Задание 3

Формулировка задачи:

Управление бинарным файлом посредством хеш-таблицы. Разработать и реализовать операции.

1) Прочитать запись из файла и вставить элемент в таблицу (элемент включает: ключ и номер записи с этим ключом в файле, и для метода с открытой адресацией возможны дополнительные поля).

2) Удалить запись из таблицы при заданном значении ключа и соответственно из файла.

3) Найти запись в файле по значению ключа (найти ключ в хеш-таблице, получить номер записи с этим ключом в файле, выполнить прямой доступ к записи по ее номеру).

4) Подготовить тесты для тестирования приложения:

Заполните файл небольшим количеством записей.

- Включите в файл записи как не приводящие к коллизиям, так и приводящие.
- Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.

Заполните файл большим количеством записей (до 1 000 000).

Определите время чтения записи с заданным ключом: для первой записи файла,

для последней и где-то в середине. Убедитесь (или нет), что время доступа для всех записей одинаково.

Решение:

Для решения Задания 3 были изменены случаи case в оснорвной функции main. Функции хеширования, рехеширования и добавления элемента остались без изменений.

Для реализации поиска были добавлены строчки обращения к записи в

файле напрямую, через смещение, записанное внутри хэш-элемента.

Для реализации удаления записи был разработал алгоритм, в котором создается еще один бинарный файл, он открывается каждый раз пустым и в него записываются все элементы исходного бинарного файла, кроме той записи, которую пользователь ввел для удаления по ключу. Эту запись копирование в новый файл просто игнорирует, при этом смещая курсор на размер записи, чтоб сохранить корректные смещения в самой хэш-таблице.

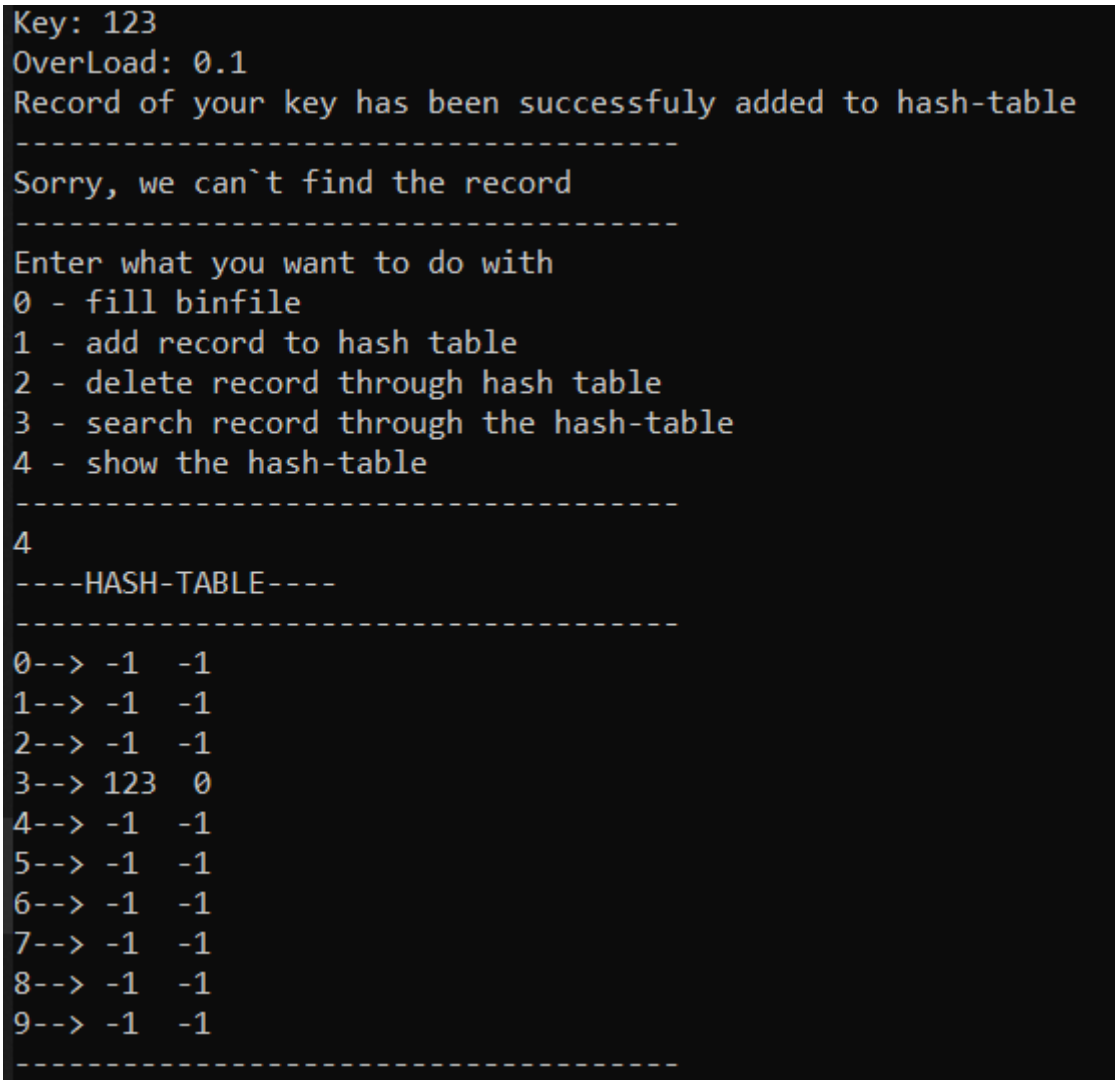

Рисунок 7 – Чтение записи из бинарного файла и добавление его в хэш-таблицу

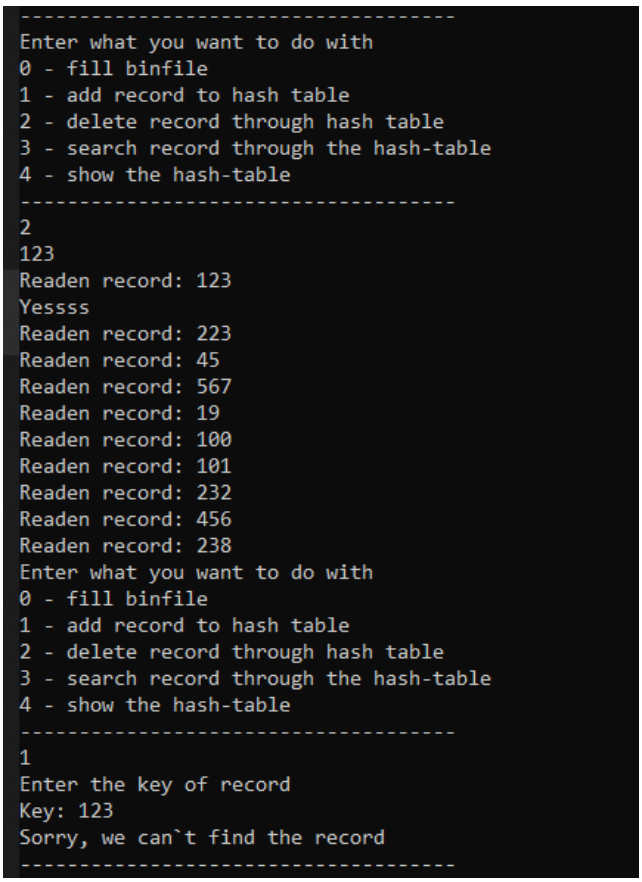

Рисунок 9 – Удаление элемента из хэш таблицы и бинарного файла

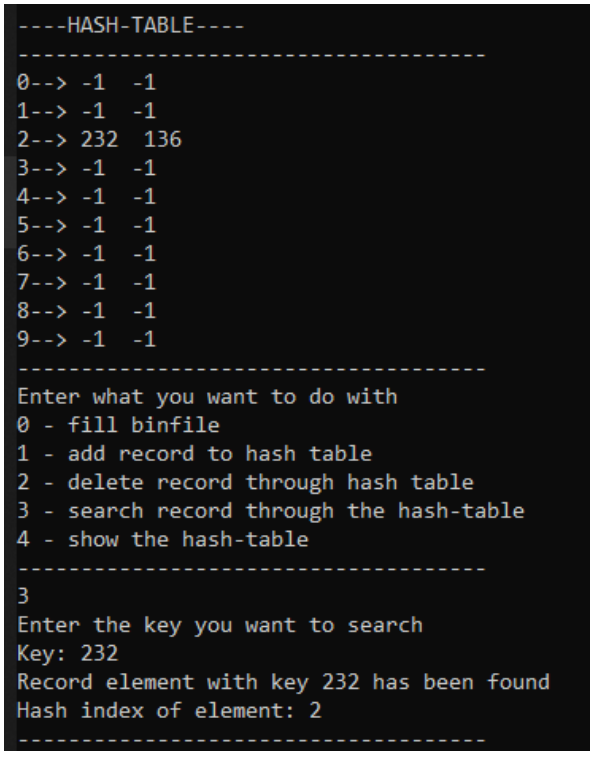

Рисунок 10 – Поиск элемента в хэш-таблице и обращение к нему в бинарном файле напрямую

```
Enter the key you want to search
Key: 1000041
Time: 9e-07 sec.
Record element with key 1000041 has been found
Hash index of element: 1
------------------------------------
```

Рисунок 11 – Время поиска первого элемента в хэш-таблице при 30000 записей в бинарном файле

```
Enter the key you want to search
Key: 1005574
Time: 9e-07 sec.
Record element with key 1005574 has been found
Hash index of element: 4
------------------------------------
```

Рисунок 12 - Время поиска серединного элемента в хэш-таблице при 30000 записей в бинарном файле

```
Enter the key you want to search
Key: 1029326
Time: 9e-07 sec.
Record element with key 1029326 has been found
Hash index of element: 6
------------------------------------
Unfortunately, can`t find element
------------------------------------
```

Рисунок 13 - Время поиска последнего элемента в хэш-таблице при 30000 записей в бинарном файле

```cpp
#include <iostream>
#include <fstream>
#include <set>

//----Структура записи в текстовом и бинарном файле----//
struct wordRecord
{
    char word[5];
    unsigned short int amountOfEntry;

public:
    wordRecord(std::string userWord, unsigned short int userAmountOfEntry)
    {
        for (int i = 0; i < userWord.size(); i++)
        {
            word[i] = userWord[i];
        }
        this->amountOfEntry = userAmountOfEntry;
    }
    wordRecord()
```

```
    {
        amountOfEntry = -1;
    }
};
//----Хэш-функция----//
unsigned short hashFunc(unsigned short int key, int tableSize)
{
    return(key % tableSize);
}
//----Структура записи в хэш-таблице----//
struct hashElement
{
    short int amountKey;
    std::streampos offset;

public:
    hashElement(unsigned short int userAmountOfEntry, std::streampos currOffset,
int tableSize)
    {
        this->amountKey = userAmountOfEntry;
        this->offset = currOffset;
    }
    hashElement()
    {
        this->amountKey = -1;
        this->offset = -1;
    }
};
//----Функция вставки элемента в хэш-таблицу----//
void addHashElement(hashElement newHashElement, int newHashIndex, hashElement*
hashTable, int tableSize)
{
    if (hashTable[newHashIndex].offset == -1)
    {
        hashTable[newHashIndex] = newHashElement;
    }
    else if (hashTable[newHashIndex].amountKey != -1)
    {
        addHashElement(newHashElement, newHashIndex + 1, hashTable, tableSize);
    }
}
//----Функция добавления элемента в хэш-таблицу----//
void addElementFunc()
{

}
//----Функция удаления элемента из хэш-таблицы----//
void removeElementFunc()
{

}
//----Функция поиска элемента в хэш-таблице----//
void searchElementFunc()
{

}
//----Функция рехеширования----//
hashElement* reHashFunc(hashElement* hashTable, int tableSize)
{
    hashElement* reHashTable = new hashElement[tableSize * 2];
    hashElement reHashElement;
    int newHashIndex;
    for (int i = 0; i < tableSize; i++)
    {
        if(hashTable[i].amountKey != -1)
```

```
        {
            reHashElement.amountKey = hashTable[i].amountKey;
            reHashElement.offset = hashTable[i].offset;
            newHashIndex = hashFunc(hashTable[i].amountKey, tableSize * 2);
            addHashElement(reHashElement, newHashIndex, reHashTable, tableSize *
2);
        }
    }
    return(reHashTable);
}
//----Функция генерации случайного числа----//

int main()
{
    double amoutOfOccupiedElements = 0;
    int tableSize = 10;
    int userChoice;
    int newHashIndex;
    double overLoad;
    unsigned short int userAmountOfEntry;
    int userKeyToInsert;
    int userKeyToDelete;
    int userKeyToSearch;
    hashElement* hashTable = new hashElement[tableSize];
    wordRecord toRead;
    std::streampos currOffsetWrite = 0;
    std::streampos currOffsetRead = 0;
    std::string userWord;
    std::string fileName = "wordRecordList.bin";
    std::cout << "Work with hash-table and binary file \n";
    std::cout << "------------------------------------ \n";
    do
    {
        std::cout << "Enter what you want to do with \n";
        std::cout << "0 - fill binfile \n";
        std::cout << "1 - add record to hash table \n";
        std::cout << "2 - delete record through hash table \n";
        std::cout << "3 - search record through the hash-table \n";
        std::cout << "4 - show the hash-table \n";
        std::cout << "------------------------------------ \n";
        std::cin >> userChoice;
        switch (userChoice)
        {
        case 0:
        {
            int recordAmount;
            std::cout << "Enter amount of records \n";
            std::cout << "Amount: ";
            std::cin >> recordAmount;
            std::ofstream fout(fileName, std::ios::binary);
            for (int i = 0; i < recordAmount; i++)
            {
                std::cout << "Enter the word and its aomunt of entry \n";
                std::cout << "Word: ";
                std::cin >> userWord;
                std::cout << "Amount: ";
                std::cin >> userAmountOfEntry;
                wordRecord userWordRecord(userWord, userAmountOfEntry);
                if (!fout.is_open()) { std::cout << "Sorry, can`t find your file
\n"; break; }
                fout.seekp(currOffsetWrite);
                fout.write((char*)&userWordRecord, sizeof(wordRecord));
                currOffsetWrite = fout.tellp();
                std::cout << "CURRENT POS OF WRITE " << currOffsetWrite << "\n";
                std::cout << "Successfuly written to file!! \n";
```

```
                std::cout << "------------------------------------ \n";
            }
            fout.close();
            break;
        }
        case 1:
        {
            std::cout << "Enter the key of record \n";
            std::cout << "Key: ";
            std::cin >> userKeyToInsert;
            std::ifstream fin(fileName, std::ios::binary);
            while (!fin.eof())
            {
                fin.seekg(currOffsetRead);
                fin.read(reinterpret_cast<char*>(&toRead), sizeof(wordRecord));
                if (toRead.amountOfEntry == userKeyToInsert)
                {
                    amoutOfOccupiedElements++;
                    overLoad = amoutOfOccupiedElements / tableSize;
                    std::cout << "OverLoad: " << overLoad << '\n';
                    if (overLoad >= 0.75)
                    {
                        hashTable = (hashElement*)realloc(hashTable,
sizeof(hashElement) * tableSize * 2);
                        hashTable = reHashFunc(hashTable, tableSize);
                        tableSize *= 2;
                    }
                    hashElement toInsert;
                    toInsert.amountKey = toRead.amountOfEntry;
                    toInsert.offset = currOffsetRead;
                    newHashIndex = hashFunc(toInsert.amountKey, tableSize);
                    addHashElement(toInsert, newHashIndex, hashTable, tableSize);
                    currOffsetRead = 0;
                    std::cout << "Record of your key has been successfuly added
to hash-table \n";
                    std::cout << "------------------------------------ \n";
                    break;
                }
                currOffsetRead = fin.tellg();
            }
            currOffsetRead = 0;
            std::cout << "Sorry, we can`t find the record \n";
            std::cout << "------------------------------------ \n";
            fin.close();
            break;
        }
        case 2:
        {
            int ignore;
            std::cin >> ignore;
            for(int i = 0; i < tableSize; i++)
            {
                if(hashTable[i].amountKey == ignore)
                {
                    std::ifstream finRead(fileName, std::ios::binary);
                    std::ofstream foutWrite("temp.bin", std::ios::binary |
std::ios::trunc);
                    wordRecord tempRecord;
                    finRead.read((char*)&tempRecord, sizeof(wordRecord));
                    while (!finRead.eof())
                    {
                        std::cout << "Readen record: " <<
tempRecord.amountOfEntry << '\n';
                        if (ignore == tempRecord.amountOfEntry)
                        {
```

```
                            std::cout << "Yessss \n";
                            currOffsetWrite += sizeof(wordRecord);
                            foutWrite.seekp(currOffsetWrite);
                        }
                        else
                        {
                            foutWrite.write((char*)&tempRecord,
sizeof(wordRecord));
                            currOffsetWrite = foutWrite.tellp();
                        }
                        finRead.read((char*)&tempRecord, sizeof(wordRecord));
                    }
                    currOffsetWrite = 0;
                    finRead.close();
                    foutWrite.close();
                    remove("wordRecordList.bin");
                    rename("temp.bin", "wordRecordList.bin");
                    hashTable[i].amountKey = -1;
                    hashTable[i].offset = -1;
                    break;
                }
            }
            break;
        }
        case 3:
        {
            std::cout << "Enter the key you want to search \n";
            std::cout << "Key: ";
            std::cin >> userKeyToSearch;
            for (int i = 0; i < tableSize; i++)
            {
                if (hashTable[i].amountKey == userKeyToSearch)
                {
                    std::ifstream finSearch(fileName, std::ios::binary);
                    finSearch.seekg( hashTable[i].offset);
                    finSearch.read((char*)&toRead, sizeof(wordRecord));
                    std::cout << "Record element with key " <<
toRead.amountOfEntry << " has been found \n";
                    std::cout << "Hash index of element: " << i << '\n';
                    std::cout << "------------------------------------ \n";
                    currOffsetRead = 0;
                    finSearch.close();
                    break;
                }
            }
            std::cout << "Unfortunately, can`t find element \n";
            std::cout << "------------------------------------ \n";
            break;
            break;
        }
        case 4:
        {
            std::cout << "----HASH-TABLE---- \n";
            std::cout << "------------------------------------ \n";
            for (int i = 0; i < tableSize; i++)
            {
                std::cout << i << "--> " << hashTable[i].amountKey << "  " <<
hashTable[i].offset << '\n';
            }
            std::cout << "------------------------------------ \n";
            break;
        }
        }
```

```
    } while (userChoice == 0 || userChoice == 1 || userChoice == 2 || userChoice
== 3 || userChoice == 4);
}
```

Листинг 2 – листинг кода к заданию 3

Вывод

Все задачи были решены. Также при выполнении этих задач познакомились с хещ-таблицами и работой с ними, а также применили их для быстрого доступа к записям в бинарном файле. Хотя хеш-таблица работает очень быстро, но добиться всегда одного и того-же времени поиска очень сложно. Также двойное хеширование как алгоритм очень хорошо работает, когда в таблице много места для записей, но начинает делать много проходок при малом пространстве и требует очень строго контроля при удалении чтобы цепочка переходов не затерлась.