

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

# РТУ МИРЭА

Отчет

Практическая работа №3

Дисциплина Структуры и алгоритмы обработки данных

Тема. Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины

|  |  |
| --- | --- |
| Выполнил студент | Медведев И. В. |
|  | Фамилия И.О. |
| Группа | ИНБО-08-21 |
|  | Номер группы |

**Москва 2022**

Вариант № 16

**ЗАДАНИЕ 1**

Условие задания

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.

Требования к выполнению

1. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы. Имена заголовочным файлам определите сами. Подключите заголовочные файлы к приложению.

2. Для обеспечения прямого доступа к записи в файле элемент хеш-таблицы должен включать обязательные поля: ключ записи в файле, номер записи с этим ключом в файле. Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.

3. Управление хеш-таблицей.

1) Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте. Определения разместить в соответствующем заголовочном файле. Все операции управления хеш-таблицей размещать в этом заголовочном файле.

2) Тестирование операций выполнять в функции main приложения по мере их реализации.

После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.

Разработать операции по управлению хеш-таблицей.

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

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

5) 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.

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

4. Управление двоичным файлом.

Операции управления двоичным файлом: создание двоичного файла из текстового, добавить запись в двоичный файл, удалить запись с заданным ключом из файла, прочитать запись файла по заданному номеру записи.

Примечание. Эти операции должны быть отлажены в практической работе 2, или уже в этой работе, если их пока нету.

Структура записи двоичного файла и все операции, по управлению файлом, должны быть размещены в соответствующем заголовочном файле.

Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.

5. Управление файлом посредством хеш-таблицы.

В заголовочный файл управления файлом посредством хеш-таблицы подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом. Реализовать поочередно все перечисленные ниже операции в этом заголовочном файле, выполняя их тестирование из функции main приложения. После разработки всех операций выполнить их комплексное тестирование.

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.

Постановка задачи варианта

|  |  |  |
| --- | --- | --- |
| Вариант16 | Цепное хеширование | Расписание занятий группы: номер группы, количество студентов, средний балл группы. |

Реализованная структура:

Структура записи в файле:

struct groupElement {  
 int groupId;  
 double medianScore;  
 int studentCount;  
 int predmetId;  
  
 groupElement();  
  
 groupElement(int groupId);  
  
 groupElement(int groupId, double medianScore, int studentCount, int predmetId);  
};

Реализация методов:

groupElement::groupElement() {  
 srand(time(nullptr));  
 groupId = rand() % 100;  
 medianScore = -1;  
 studentCount = -1;  
 predmetId = -1;  
}  
  
groupElement::groupElement(int groupId) {  
 this->groupId = groupId;  
 this->medianScore = -1;  
 this->studentCount = -1;  
 this->predmetId = -1;  
}  
  
groupElement::groupElement(int groupId, double medianScore, int studentCount, int predmetId) {  
 this->groupId = groupId;  
 this->medianScore = medianScore;  
 this->studentCount = studentCount;   
 this->predmetId = predmetId;  
}

Структура элементов таблицы:

struct tableNode {  
 int groupId;  
 int entryId;  
 tableNode \*next;  
  
 tableNode(int groupId, int entryId, tableNode \*next = nullptr);  
};

Реализация методов:

tableNode::tableNode(int groupId, int entryId, tableNode \*next) {  
 this->groupId = groupId;  
 this->entryId = entryId;  
 this->next = next;  
}

Структура таблицы:

struct HashTable {  
 int length;  
 int filled;  
 tableNode \*\*rows;  
  
 HashTable(int length = 10);  
  
 int insert(groupElement gotElement, int entryId = -1);  
  
 int remove(int groupId);  
  
 tableNode \*search(int groupId, int entryId = -1, int params = 0);  
  
 int rehash();  
  
 int print();  
  
 ~HashTable();  
};

Реализация конструкторов и деструкторов:

HashTable::HashTable(int length) {  
 rows = new tableNode \*[length]{nullptr};  
 this->length = length;  
 this->filled = 0;  
}  
  
HashTable::~HashTable() {  
 this->length=0;  
 this->filled=0;  
 rows = nullptr;  
}

Декомпозиция и алгоритмы выполнения операций:

1. Открытие файла с проверкой

A

template<class fileStream>

void openFile(fileStream &file, std::string FILE\_NAME, const std::string &dir = "in", const std::string &type = "txt");

void

Входные данные

Результат

void openFile(fileStream &file, std::string FILE\_NAME, const std::string &dir = "in", const std::string &type = "txt");

template<class fileStream>  
void openFile(fileStream &file, string FILE\_NAME, const string &dir, const string &type) {  
 while (!file.is\_open() || !file) {  
 if (type == "binary") {  
 if (dir == "in") file.open(FILE\_NAME, ios::in | ios::binary);  
 else if (dir=="out") file.open(FILE\_NAME, ios::out | ios::binary);  
 else file.open(FILE\_NAME, std::ios::in | std::ios::out | std::ios::app | std::ios::binary);  
 } else file.open(FILE\_NAME);  
 if (!file.is\_open() || !file) {  
 cout << "error occured while openning the file. write any number to try again or write 0 to exit." << endl;  
 int r;  
 cin >> r;  
 if (r == 0)  
 break;  
 } else break;  
 }  
}

1. Создание и заполнение текстового файла

A

const std::string &txtFileName

void

Входные данные

Результат

void createTxtFile(const std::string &txtFileName);

void createTxtFile(const std::string &txtFileName) {  
 ofstream file;   
openFile(file, txtFileName);   
//int groupId; double medianScore; int studentCount; int predmetId;  
file << 12 << endl << 4.2 << endl << 28 << endl << 1 << endl;  
file << 14 << endl << 3 << endl << 15 << endl << 3 << endl;  
file << 112 << endl << 4.2 << endl << 28 << endl << 2 << endl;  
file << 25 << endl << 3.6 << endl << 30 << endl << 2 << endl;  
file << 66 << endl << 4.9 << endl << 25 << endl << 6;  
file << 11 << endl << 2.1 << endl << 7 << endl << 1 << endl;  
file << 3 << endl << 2 << endl << 6 << endl << 3 << endl;  
file << 255 << endl << 2.2 << endl << 5 << endl << 5 << endl;  
file << 666 << endl << 2.3 << endl << 4 << endl << 4 << endl;  
file << 1232 << endl << 2.9 << endl << 3 << endl << 6 << endl;  
file.close();  
}

1. Вывод значений бинарного файла

A

const std::string &binFileName

void

Входные данные

Результат

void printOutBinFile(const std::string &binFileName);

void printOutBinFile(const std::string &binFileName) {  
 fstream file;  
 openFile(file, binFileName, "in", "bin");  
 groupElement temp;  
 while (file.read((char \*) &temp, sizeof(groupElement))) {  
 cout << temp.groupId << endl;  
 }  
 file.close();  
}

1. Запись значения в конец бинарного файла

A

const string &binFileName, groupElement entry = \*new groupElement()

void

Входные данные

Результат

void addEntryInBin(const string &binFileName, groupElement entry = \*new groupElement());

void addEntryInBin(const string &binFileName, groupElement entry) {  
 ofstream writeFile;  
 openFile(writeFile, binFileName, "app", "binary");  
 writeFile.write((char \*) &entry, sizeof(groupElement));  
 writeFile.close();  
}

1. Прочитать значение на определенной позиции в бинарном файле

A

const string &binFileName, int order = 0

groupElement

Входные данные

Результат

groupElement getEntryFromBin(const string &binFileName, int order = 0);

groupElement getEntryFromBin(const string &binFileName, int order) {  
 ifstream readFile;  
 openFile(readFile, binFileName, "in", "binary");  
 groupElement entry;  
 readFile.seekg(sizeof(groupElement) \* order, ios::beg);  
 readFile.read((char \*) &entry, sizeof(groupElement));  
 readFile.close();  
 return entry;  
}

1. Удаление записи из бинарного файла по ключу

A

int groupId, const string &binFileName

int

Входные данные

Результат

int deleteEntryByKey(int groupId, const string &binFileName);

int deleteEntryByKey(int groupId, const string &binFileName) {  
 groupElement entry;  
 groupElement lastEntry;  
 fstream file;  
 file.open(binFileName, ios::in | ios::out | ios::binary);  
 file.seekg(-1 \* sizeof(groupElement), ios::end);  
 int sizeWithoutLast = file.tellg();  
 file.read((char \*) &lastEntry, sizeof(groupElement));  
 file.seekg(0, ios::beg);  
  
 int i = 0;  
 while (file.read((char \*) &entry, sizeof(groupElement)) && entry.groupId != groupId) {  
 i++;  
 }  
  
 file.seekg(i \* sizeof(groupElement), ios::beg);  
 file.write((char \*) &lastEntry, sizeof(groupElement));  
  
 file.seekg(0, std::ios::beg);  
 std::filesystem::resize\_file(binFileName, sizeWithoutLast);  
 file.close();  
  
 return i;  
}

1. Подсче количества записей в файле

A

const string &binFileName

int

Входные данные

Результат

int getFileLength(const std::string &binFileName);

int getFileLength(const string &binFileName) {  
 fstream file;  
 file.open(binFileName, ios::in | ios::out | ios::binary);  
 groupElement entry;  
 int fileLength = 0;  
 while (file.read((char \*) &entry, sizeof(entry))) {  
 fileLength++;  
 }  
 file.close();  
 return fileLength;  
}

1. Вычисление хэша

A

int groupId, int length

int

Входные данные

Результат

int hashIndex(int groupId, int length)

int hashIndex(int groupId, int length) {  
 return groupId % length;  
}

1. Вставка элемента в хэш-таблицу

A

groupElement gotElement, int entryId

int

Входные данные

Результат

int HashTable::insert(groupElement gotElement, int entryId)

int HashTable::insert(groupElement gotElement, int entryId) {  
 int i = hashIndex(gotElement.groupId, this->length);  
 tableNode \*currentNode = this->rows[i]; //head  
 while (currentNode && currentNode->next != nullptr)  
 currentNode = currentNode->next;  
 tableNode \*newNode = new tableNode(gotElement.groupId, entryId);  
 if (currentNode == nullptr)  
 this->rows[i] = newNode;  
 else  
 currentNode->next = newNode;  
 this->filled++;  
 if (this->filled / this->length >= 0.75)  
 this->rehash();  
 return 0;  
}

1. Рехеширование таблицы

A

groupElement gotElement, int entryId

int

Входные данные

Результат

int HashTable::rehash()

int HashTable::rehash() {  
 int oldLength = this->length;  
 tableNode \*\*oldRows = this->rows;  
 this->rows = new tableNode \*[this->length \*= 2]{nullptr};  
 this->filled = 0;  
 for (int i = 0; i < oldLength; i++) {  
 tableNode \*currentNode = oldRows[i];  
 while (currentNode) {  
 this->insert(groupElement(currentNode->groupId), currentNode->entryId);  
 currentNode = currentNode->next;  
 }  
 delete[] currentNode;  
 }  
 delete[] oldRows;  
 return 0;  
}

1. Поиск в хэш-таблице

A

int groupId, int entryId, int params

tableNode \*

Входные данные

Результат

tableNode \*HashTable::search(int groupId, int entryId, int params)

tableNode \*HashTable::search(int groupId, int entryId, int params) {  
 int i = hashIndex(groupId, this->length);  
 tableNode \*node = this->rows[i];  
 while (node && node->groupId != groupId && (params != -1 || node->next->groupId != groupId) &&  
 (entryId == -1 || node->entryId != entryId))  
 node = node->next;  
 if (params == +1)  
 return node->next;  
 if (params == -1 && node->groupId == groupId)  
 return nullptr;  
 if ((node->groupId == groupId || params != 0) && (entryId == -1 || node->entryId == entryId))  
 return node;  
 else return nullptr;  
}

1. Вывод значений хэш-таблицы

A

int

Входные данные

Результат

int HashTable::print()

int HashTable::print() {  
 cout << "print:\n";  
 for (int i = 0; i < this->length; i++) {  
 tableNode \*currentElement = rows[i];  
 cout << i << ":";  
 while (currentElement) {  
 cout << " (groupId: " << currentElement->groupId << " entryId: " << currentElement->entryId << ')';  
 currentElement = currentElement->next;  
 }  
 cout << endl;  
 delete[] currentElement;  
 }  
 return 0;  
}

1. Удаление значения из хэш-таблицы

A

Int groupId

int

Входные данные

Результат

int HashTable::remove(int groupId)

int HashTable::remove(int groupId) {  
 tableNode \*prevNode = this->search(groupId, -1, -1);  
 tableNode \*nextNode = this->search(groupId, -1, +1);  
 if (!prevNode) {  
 int i = hashIndex(groupId, this->length);  
 this->rows[i] = nextNode;  
 } else prevNode->next = nextNode;  
 return 0;  
}

1. Заполнение хэш-таблицы значениями бинарного файла

A

HashTable &table, const std::string &binFileName

void

Входные данные

Результат

void fillTableFromBin(HashTable &table, const std::string &binFileName)

void fillTableFromBin(HashTable &table, const std::string &binFileName) {  
 groupElement entry;  
 for (int i = 0; i < getFileLength(binFileName); i++) {  
 entry = getEntryFromBin(binFileName, i);  
 table.insert(entry, i);  
 }  
}

1. Найти последний элемент в таблице

A

HashTable &table

tableNode\*

Входные данные

Результат

tableNode \*findLastElementInTable(HashTable &table)

tableNode \*findLastElementInTable(HashTable &table) {  
 tableNode \*last = new tableNode(0, -1, nullptr);  
 for (int i = 0; i < table.length; i++) {  
 tableNode \*currentElement = table.rows[i];  
 while (currentElement) {  
 if (currentElement->entryId > last->entryId) {  
 last = currentElement;  
 }  
 currentElement = currentElement->next;  
 }  
 delete[] currentElement;  
 }  
 return last;  
}

1. Удаление записи из файла и из таблицы

A

HashTable &table, const std::string &binFileName, int groupId

void

Входные данные

Результат

void removeEntry(HashTable &table, const std::string &binFileName, int groupId)

void removeEntry(HashTable &table, const std::string &binFileName, int groupId) {  
 tableNode \*node = findLastElementInTable(table);  
 table.print();  
 table.remove(groupId);  
 int newEntryId = deleteEntryByKey(groupId, binFileName);  
 node->entryId = newEntryId;  
 table.print();  
}

1. Нахождение записи в бинарном файле по ключу

A

HashTable &table, int groupId

groupElement

Входные данные

Результат

groupElement findInBinByKey(HashTable &table, int groupId)

groupElement findInBinByKey(HashTable &table, int groupId) {  
 tableNode \*foundInTable = table.search(groupId);  
 groupElement res(foundInTable->groupId);  
 return res;  
}

1. Проверка поиска при миллионе элементов

void milTest(HashTable &table,const std::string &binFileName)

void milTest(HashTable &table,const std::string &binFileName) {  
 HashTable newTable(1000000);  
 for (int i=0;i<1000000;i++){  
 table.insert(groupElement(i,0,0,0),i);  
 }  
 unsigned int start\_time = clock(); // начальное время  
 tableNode \*node = table.search(0);  
 unsigned int end\_time = clock(); // конечное время  
 unsigned int search\_time1 = end\_time - start\_time;  
 start\_time = clock(); // начальное время  
 node = table.search(1000000-1);  
 end\_time = clock(); // конечное время  
 unsigned int search\_time2 = end\_time - start\_time;  
  
 cout << "first element find in " << search\_time1 << "ms" << endl;  
 cout << "last element find in " << search\_time2 << "ms" << endl;  
}

1. Тестирование основного функционала программы

A

const std::string &binFileName, const std::string &textFileName

int

Входные данные

Результат

int testHashT(const std::string &binFileName, const std::string &textFileName);

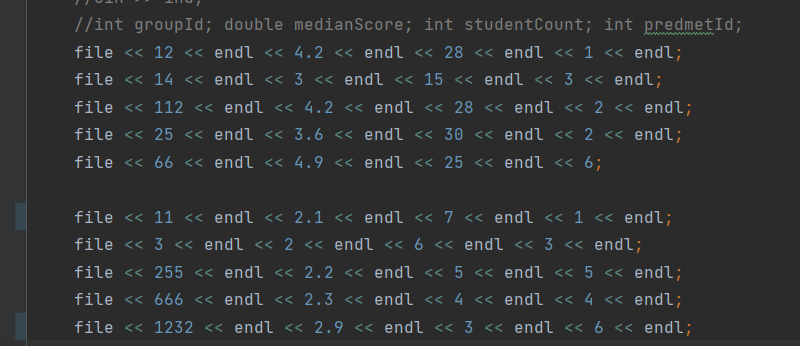
int testHeshT(const std::string &BIN\_FILE\_NAME, const std::string &FILE\_NAME) {  
 HashTable table(10);  
 tableNode \*node;  
 groupElement \*elem;  
  
 int num;  
 while (true) {  
 cout << " Operations:" << endl;  
 cout << " 1. Create default TXT file" << endl;  
 cout << " 2. Create BIN file from TXT file" << endl;  
 cout << " 3. Create hashtable" << endl;  
 cout << " 4. Fill hashtable from BIN file" << endl;  
 cout << " 5. Print hashtable" << endl;  
 cout << " 6. Insert in hashTable" << endl;  
 cout << " 7. Remove from hashTable" << endl;  
 cout << " 8. Rehash hashTable" << endl;  
 cout << " 9. Find element in hashTable" << endl;  
 cout << " 10. Remove from hashTable and BIN" << endl;  
 cout << " 11. Find element by key" << endl;  
 cout << " 12. Print BIN file" << endl;  
 cout << " 13. Insert in hashTable and BIN" << endl;  
 cout << " 14. Get hash" << endl;  
 cout << " 15. Test with 1000000 records" << endl;  
 cout << " 0. EXIT" << endl;  
 cout << "numPunkt=";  
 cin >> num;  
 switch (num) {  
 case 1:  
 createTxtFile(FILE\_NAME);  
 break;  
 case 2: {  
 createBinFromTxt(FILE\_NAME, BIN\_FILE\_NAME);  
 break;  
 }  
 case 3:  
 cout << "Введите размер таблицы или 0 для автоматического создания" << endl;  
 int tableLength;  
 cin >> tableLength;  
 if (tableLength <= 0)  
 tableLength = getFileLength(BIN\_FILE\_NAME);  
 if (tableLength <= 0) {  
 tableLength = 10;  
 }  
 table = HashTable(tableLength);  
 break;  
 case 4:  
 fillTableFromBin(table, BIN\_FILE\_NAME);  
 break;  
 case 5:  
 table.print();  
 break;  
 case 6:  
 cout << "write groupId, medianScore, studentCount, predmetId" << endl;  
 int groupId1;  
 double medianScore;  
 int studentCount;  
 int predmetId;  
 cin >> groupId1 >> medianScore >> studentCount >> predmetId;  
 elem = new groupElement(groupId1, medianScore, studentCount, predmetId);  
 table.insert(\*elem);  
 break;  
 case 7:  
 int groupId2;  
 cin >> groupId2;  
 table.remove(groupId2);  
 break;  
 case 8:  
 table.rehash();  
 break;  
 case 9:  
 int groupId3;  
 cin >> groupId3;  
 node = table.search(groupId3);  
 cout << node->groupId << " " << node->entryId << endl;  
 break;  
 case 10:  
 int groupId4;  
 cin >> groupId4;  
 removeEntry(table, BIN\_FILE\_NAME, groupId4);  
 break;  
 case 11:  
 int groupId5;  
 cin >> groupId5;  
 \*elem = findInBinByKey(table, groupId5);  
 cout << elem->groupId << endl;  
 break;  
 case 12:  
 printOutBinFile(BIN\_FILE\_NAME);  
 break;  
 case 13:  
 addEntry(table, BIN\_FILE\_NAME);  
 break;  
 case 14:  
 cout << "write groupId" << endl;  
 int groupId6;  
 cin >> groupId6;  
 cout << hashIndex(groupId6, table.length) << endl;  
 break;  
 case 15:  
 cout << "Test with many records" << endl;  
 milTest(table, BIN\_FILE\_NAME);  
 break;  
 case 0:  
 exit(0);  
 default:  
 cout << "unknown number. write 0 to exit." << endl;  
 break;  
 }  
 }  
}

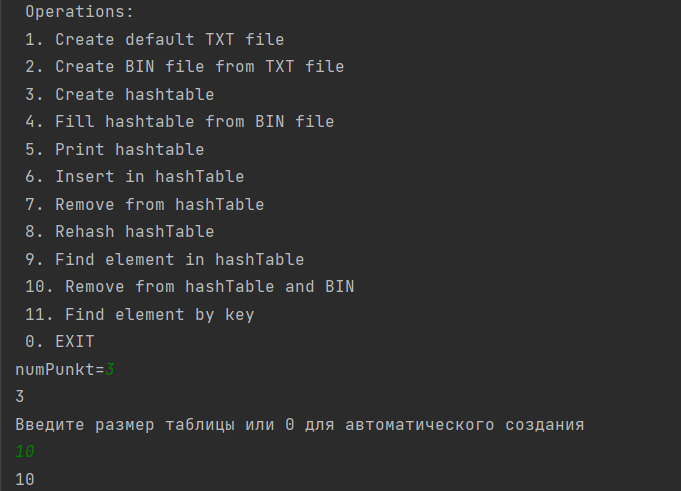
**Функция main**

const string FILE\_NAME = "hash.txt";  
const string BIN\_FILE\_NAME = "bin\_hash.bin";  
  
int main() {  
 system("chcp 65001");  
  
 testHeshT(BIN\_FILE\_NAME, FILE\_NAME);  
  
 return 0;  
}

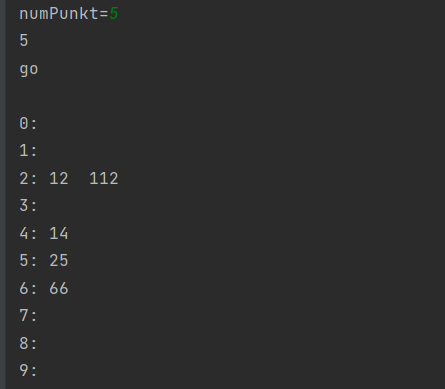
**Тестирование программы**

**Начальные значения текстового файла**

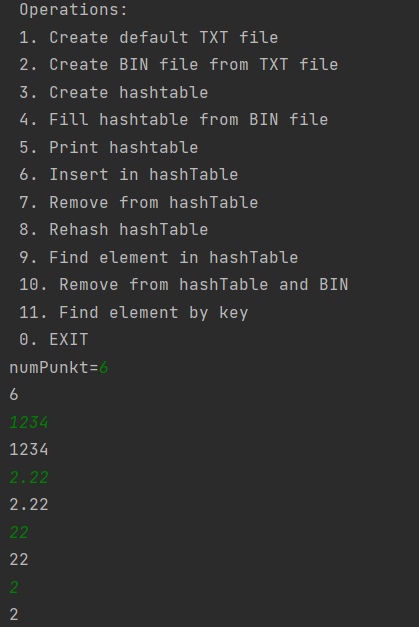
****

****

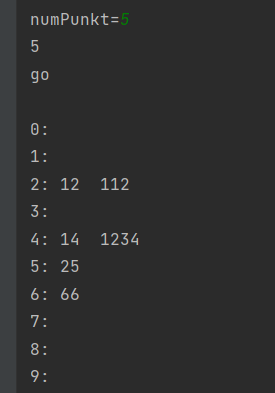
Старт программы, создание текстового файла, бинарного и заполнение таблицы.



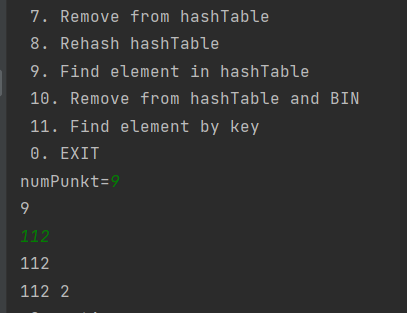
Вывод таблицы



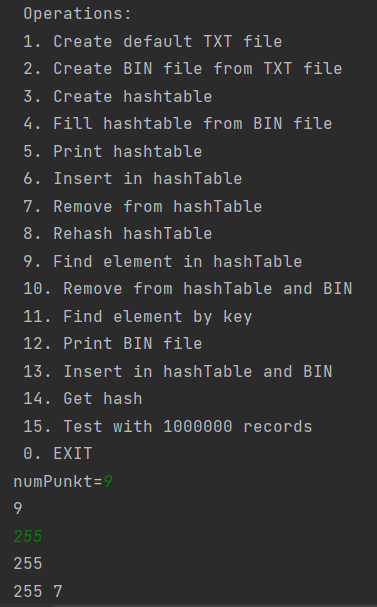
Вставка значения в таблицу

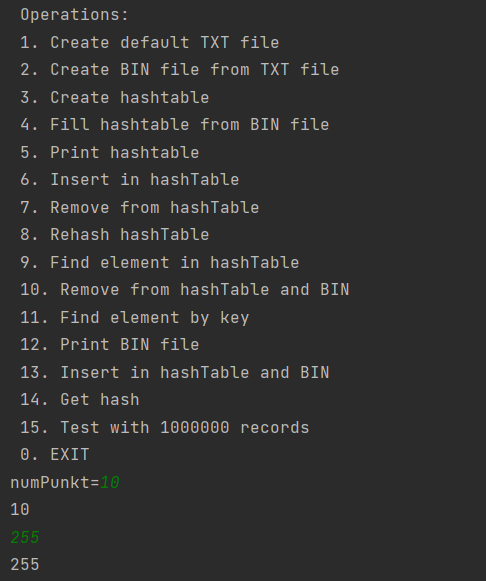


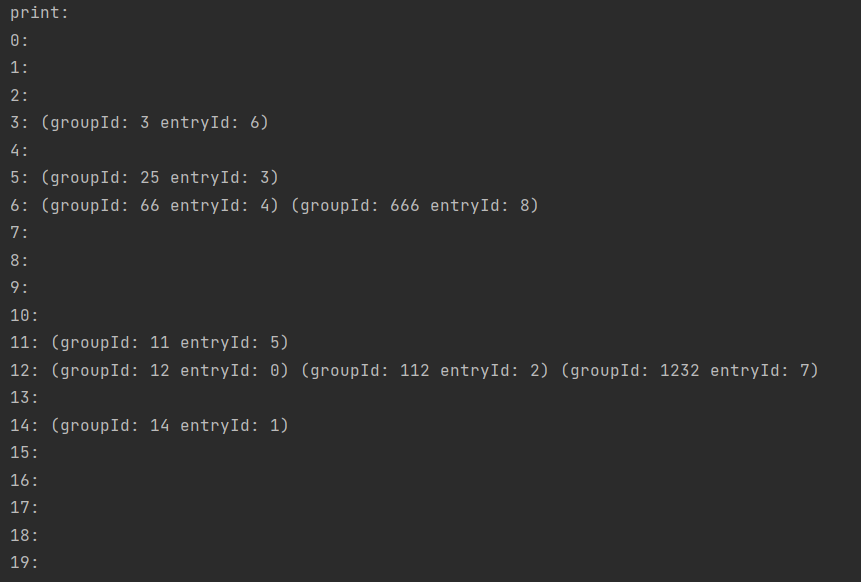
Проверка вставки значения в таблицу



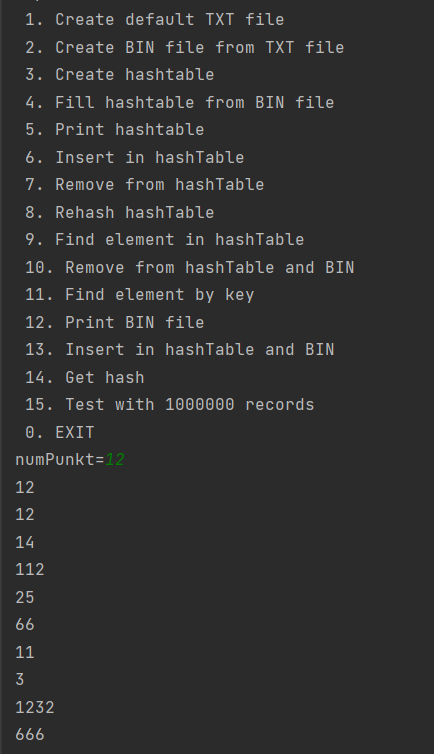
Поиск значения в таблице



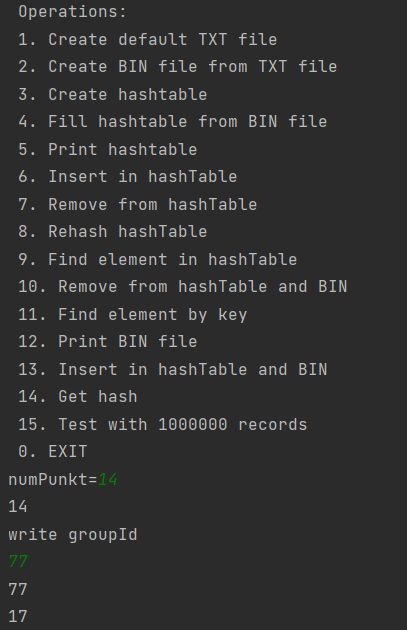




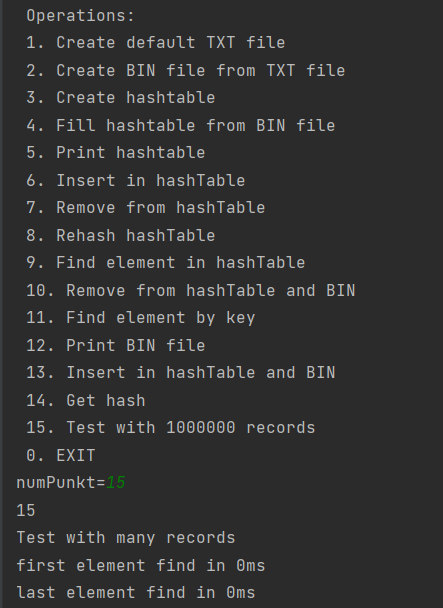
Удаление значения из таблицы



Вывод значений бинарного файла



Тестирование хэш-функции



Тестирование с заполнением большого количества записей

Все тесты выполнены успешно, полученные результаты соответствуют ожидаемым.

Выводы:

При выполнении практической работы я научился:

* Выполнять операции над бинарными файлами
* Контроллировать открытие и закрытие файлов
* Работать со структурами данных
* Управлять двоичными файлами
* Работать с хэш-таблицами
* Записывать и считывать данные из хэш-таблиц
* Организовывать работу между хэш-таблицами и файлами