|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 «Xеш-таблицы»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-13-22 | Тринеев Павел Сергеевич |
| **Проверил:** | ассистент Муравьёва Е.А. |

МОСКВА 2023 г.

**Практическая работа 3**

**Тема:** применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины

**Цель:** получить навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).

**Хеширование для достижения константного времени доступа к записи в таблице**

***Хеширование*** как преобразование исходных данных в выходную битовую строку находит применение в таких сферах, как контроль целостности при передаче данных (контрольные суммы), информационная безопасность (защита паролей, ЭЦП) и некоторые другие.

В том числе хеширование может быть использовано и для организации эффективного (с константным временем О(1)) поиска (также вставки и удаления) элементов данных в ***динамическом множестве.***

Хеш-функция при этом создаёт отображение множества ключевых значений во множество индексов соответствующих записей данных в массиве в виде вспомогательной ***хеш-таблицы*** (рис. 1).

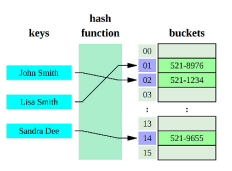


Рис. 1. Индексы элементов динамического множества данных как результат хеширования значения ключевых полей элементов полезных данных

В этом случае при вводе ключа поиска программа вычислит хеш и затем по хештаблице определит индекс искомой записи в массиве полезных данных, что открывает к ней прямой доступ.

Алгоритм хеш-функции может быть основан на делении (модальная арифметика, полиномиальный хеш), умножении (хеширование Фибоначчи), на подходе под названием «универсальное хеширование», а также некоторых других.

Например, для алгоритма, основанного на делении, хеш-функция может быть реализована на основе модальной арифметики:

h = K mod Q, (1)

где К – ключевое значение, Q – наибольшее необходимое количество различных значений хеш-функции (и, как следствие, допустимое количество записей в динамическом множестве). Если К – составное значение (например, строка символов), то его можно представить в виде полинома.

**Примечание:** в рамках данной практической работы целесообразно использовать алгоритмы, основанные на делении.

Одним из свойств хеш-функции является необязательность уникальности значений хеша для различных входных наборов данных. Это объясняет ненулевую вероятность возникновения ***коллизии*** – ситуации, когда по разным ключевым значениям может быть вычислено одинаковое хеш-значение. Таким образом, двум или более наборам данных может быть сопоставлен одинаковый индекс в массиве — а это недопустимо.

Для устранения (разрешения, преодоления) коллизии можно использовать методы ***цепного хеширования и хеш с открытой адресацией.***

Цепным хешированием называется способ разрешения коллизий, когда динамическое множество полезных данных организуется в виде массива ***линейных списков***, состоящих из элементов с одинаковыми хеш-значениями, т.е. индексами в массиве (рис. 2).

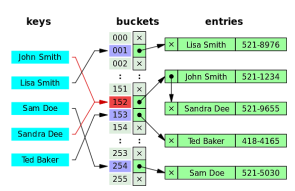


Рис. 2. Схема организации цепного хеширования

При этом в хеш-таблице ключам сопоставляются индексы головных элементов этих списков в массиве.

Массив списков может стать на некотором этапе работы программы неоднородным – несколько длинных списков и множество пустых элементов массива. С одной стороны, массив, даже пустой, занимает память. С другой стороны, время доступа к данным в списке линейное, а не константное, т.е. налицо снижение эффективности поиска.

На практике создают сначала небольшой массив, а по мере заполнения элементами перестраивают его, т.е. увеличивают размер с ***рехешированием*** (пересчетом хешей с новым значением Q).

Критерием необходимости перестройки массива является соотношение n/m – ***коэффициент нагрузки***, где n – это количество уже имеющихся записей, m – длина массива. При достижении значения этого коэффициента 0,75+, следует увеличить длину массива вдвое. Это гарантирует, что длины списков будут относительно небольшими.

Другой способ преодоления коллизий – хеширование с открытой адресацией (рис. 3). Если в массиве в строке с определённым индексом записи нет, то адрес открыт и в соответствующую строку можно поместить новый элемент. Иначе – адрес закрыт (коллизия) и необходимо по некоему алгоритму осуществить ***последовательность проб*** – сместиться относительно закрытого адреса в поисках открытого. Все базовые операции (поиск, вставка, удаление элемента) так или иначе задействуют пробирование, но у каждой свои нюансы.

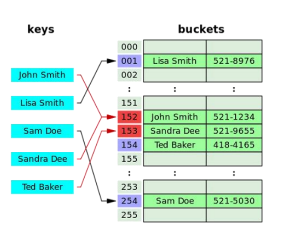


Рис. 3. Пример заполнения массива на основе открытой адресации

Распространённые схемы пробирования: линейное, квадратичное пробирование, двойное хеширование.

В наиболее простой схеме ***линейного*** пробирования смещение относительно адреса коллизии кратно целочисленной константе (эту константу следует задать так, чтобы они с длиной массива были взаимно просты):

адрес=h(x)+ci (2)

где i – номер попытки разрешить коллизию; c – константа, определяющая шаг перебора.

В ***квадратичной схеме*** шаг перебора сегментов нелинейно зависит от номера попытки найти свободный сегмент:

адрес=h(x)+ci+di2 (3)

где i – номер попытки разрешить коллизию, c и d – константы.

В схеме ***двойного хеширования*** смещение относительно закрытого адреса кратно величине второй хеш-функции, схожей, но не эквивалентной основной:

адрес=h(x)+ih2(x) (4)

В случае открытой адресации имеет смысл создать массив сразу наибольшей длины. В противном случае при постепенном заполнении массива записями будет всё более длительной процедура поиска открытого адреса. Затраты времени на перестройку этого массива лишь снизят эффективность всей программы.

**Задание 1.**

Ответьте на вопросы:

1. Расскажите о назначении хеш-фунции.

2. Что такое коллизия?

3. Что такое «открытый адрес» по отношению к хеш-таблице?

4. Как в хеш-таблице с открытым адресом реализуется коллизия?

5. Какая проблема, может возникнуть после удаления элемента из хештаблицы с открытым адресом и как ее устранить?

6. Что определяет коэффициент нагрузки в хеш-таблице?

7. Что такое «первичный кластер» в таблице с открытым адресом?

8. Как реализуется двойное хеширование?

**1. Расскажите о назначении хеш-фунции.**

**Хеш-функция** — это математическая функция, которая преобразует входные данные (обычно переменной длины) в фиксированную строку битов определенной длины. Этот результат, называемый хеш-значением или просто хешем, обычно представляет собой уникальное значение, которое определяется данными входа.

**2. Что такое коллизия?**

**Коллизия** — это ситуация, при которой два разных входных набора данных (например, сообщения, файлы или значения) приводят к одинаковым хеш-значениям при использовании определенной хеш-функции. То есть, двум разным входам соответствует один и тот же хеш-код.

**3. Что такое «открытый адрес» по отношению к хеш-таблице?**

**Открытый адрес** — это одна из стратегий разрешения коллизий в хеш-таблицах. Хеш-таблицы используются для эффективного хранения данных и быстрого поиска по ключу. Однако иногда возникают ситуации, когда нескольким разным ключам соответствует одно и то же местоположение в хештаблице (коллизия). "Открытый адрес" представляет собой метод обработки коллизий, при котором новые элементы вставляются в другое доступное место, если первоначальное место, вычисленное с использованием хеш-функции, уже занято.

**4. Как в хеш-таблице с открытым адресом реализуется коллизия?**

В хеш-таблице с открытым адресом коллизии решаются путем поиска и замены новых ключей, которые имеют коллизию с уже занятыми местами (так называемое "пробирование").

**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 с производительностью и правильностью операций поиска. Это связано с тем, что удаление элемента приводит к освобождению места, которое теперь может быть использовано для вставки новых элементов или для последующих операций поиска.

**6. Что определяет коэффициент нагрузки в хеш-таблице?**

Коэффициент нагрузки (load factor) в хеш-таблице определяет, насколько заполнена таблица данными в процентах. Он рассчитывается как отношение числа элементов, хранящихся в таблице, к общему числу доступных ячеек в таблице.

**7. Что такое «первичный кластер» в таблице с открытым адресом?**

В хеш-таблицах с открытым адресом, термин "первичный кластер" относится к ситуации, когда несколько элементов имеют одинаковый хеш-код и находятся рядом друг с другом в таблице.

**8. Как реализуется двойное хеширование?**

Двойное хеширование (Double Hashing) — это метод разрешения коллизий в хеш-таблицах с открытым адресом. Этот метод позволяет находить новые места для элементов, которые имеют коллизии, путем применения двух хеш-функций вместо одной.

**Задание 2.**

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла. Метод разрешения коллизии представлен в вашем варианте задания в таблице 1.

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.

1. Управление хеш-таблицей.

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

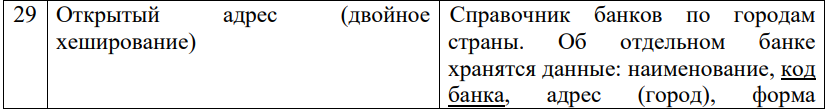
− вставку ключа с последующим рехешированием

− удаление ключа из таблицы

− поиск ключа в таблице

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.

**Вариант 29.**



Листинг двойного хеширования, решение коллизии (Листинг 1)

*Листинг 1*.

|  |
| --- |
| int Hash\_func(int Key, HashTable\* table) { |
| int Hf1 = Key % table->size; |
| int Hf2 = 1 + Key % (table->size - 1); |
| int counter = 0; |
| //cout << "size " << table->size << endl; |
| for (int i = 0; i < table->size; i++) { |
| if (table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == NULL || table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == Key){ |
| return ((Hf1 + (i \* Hf2)) % table->size); |
| } |
| } |
| return -1; |
| } |

Время создания таблицы.

|  |  |
| --- | --- |
| N | T |
| 100 | 0.0004346 |
| 1000 | 0.0005346 |
| 10000 | 0.0049239 |
| 1000000 | 0.504709 |

Время поиска по хеш-таблице

|  |  |
| --- | --- |
| N | T |
| 100 | 2.23626 |
| 1000 | 4.8029 |
| 10000 | 5.18374 |

Полный листинг для 2 задания(Листинг 2)

*Листинг 2.*

|  |
| --- |
| #include <iostream> |
| #include <fstream> |
| #include <string> |
| #include <chrono> |
| #include <set> |
| #include <vector> |
| #include <algorithm> |
| #define NULL 0 |
| #define CAPACITY 1000001 |
| #define DEL -1 |
|  |
| using namespace std; |
|  |
|  |
|  |
| struct Ht\_item |
| { |
| string name; |
| int bank\_code; //ключь |
| string address; |
| string property; |
| }; |
|  |
| struct HashTable { |
| Ht\_item\* items; |
| int size; |
| int count; |
| }; |
|  |
| //создвние |
|  |
| //создание хеш таблици |
| HashTable\* create\_table(int size) { |
| auto start = chrono::high\_resolution\_clock::now(); |
| HashTable\* table; |
| table = new HashTable; |
| table->size = size; |
| table->count = 0; |
| table->items = new Ht\_item[table->size]; |
| for (int i = 0; i < table->size; i++) { |
| table->items[i].bank\_code = NULL; |
| } |
| auto end = chrono::high\_resolution\_clock::now(); |
| chrono::duration<double> duration = end - start; |
| cout << "Время выполнения создания таблици: " << duration.count() << " секунд." << endl; |
| return table; |
| } |
|  |
| //двойная хеш функция |
| int Hash\_func(int Key, HashTable\* table) { |
| int Hf1 = Key % table->size; |
| int Hf2 = 1 + Key % (table->size - 1); |
| int counter = 0; |
| //cout << "size " << table->size << endl; |
| for (int i = 0; i < table->size; i++) { |
| if (table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == NULL || table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == Key){ |
| return ((Hf1 + (i \* Hf2)) % table->size); |
| } |
| } |
| return -1; |
| } |
|  |
| int Insert(int Key, HashTable\* table); |
|  |
| //функция рехеширования |
| HashTable\* ReHach(HashTable\* table) { |
| HashTable\* ReHachTable = new HashTable; |
|  |
| int size = table->size; |
|  |
| ReHachTable = create\_table(size \* 2); |
| int posistion; |
|  |
| for (int i = 0; i < size; i++) { |
| if(table->items[i].bank\_code != NULL) |
| { |
| posistion = Insert(table->items[i].bank\_code, ReHachTable); |
| ReHachTable->items[posistion].address = table->items[i].address; |
| ReHachTable->items[posistion].bank\_code = table->items[i].bank\_code; |
| ReHachTable->items[posistion].name = table->items[i].name; |
| ReHachTable->items[posistion].property = table->items[i].property; |
| } |
| } |
| delete[] table; |
| return ReHachTable; |
| } |
|  |
| //вставка |
| int Insert(int Key, HashTable\* table) { |
|  |
|  |
|  |
| int position = Hash\_func(Key, table); |
| //cout << "count " << table->count << endl; |
|  |
| if (position == -1) { return -1;} |
| cout << "position " << position << " Key " << Key << endl; |
|  |
|  |
|  |
| //cout << "bank\_code " << table->items[position].bank\_code << endl; |
| return position; |
| } |
|  |
|  |
|  |
| int main() { |
|  |
| setlocale(LC\_ALL, "ru"); |
|  |
| HashTable\* table; |
|  |
| table = create\_table(CAPACITY); |
|  |
| ofstream fileTextIN("Text.txt"); |
| if (!fileTextIN) |
| { |
| cout << "file is not open" << endl; |
| return 1; |
| } |
|  |
| ofstream fileBinaryIN("Bin.bin", ios::binary); |
| if (!fileBinaryIN) { |
| cout << "bin\_file is not open" << endl; |
| return 1; |
| } |
|  |
| set<int> unique; |
| Ht\_item data; |
| char spisok\_1[3][20] = {"Тинькоф", "Сбербанк", "Втб"}; |
| char spisok\_2[3][35] = { "Москва", "Санкт-Петербург", "Екатеринбург" }; |
| int a\_1, a\_2; |
|  |
| for (int i = 0; i < 1000; ++i) |
| { |
| //----------------------------------------------------------------------- |
| do { |
| data.bank\_code = rand() % (1000 - 1 + 1) + 1; |
| } while (unique.count(data.bank\_code) > 0); |
| unique.insert(data.bank\_code); |
| //----------------------------------------------------------------------- |
| a\_1 = rand() % 3; |
| data.name = spisok\_1[a\_1]; |
| //----------------------------------------------------------------------- |
| a\_2 = rand() % 3; |
| data.address = spisok\_2[a\_2]; |
| //----------------------------------------------------------------------- |
| data.property = "Коммерческий"; |
|  |
| table->count++; |
| int Flag = Insert(data.bank\_code, table); |
|  |
| table->items[Flag].name = data.name; |
| table->items[Flag].bank\_code = data.bank\_code; |
| table->items[Flag].address = data.address; |
| table->items[Flag].property = data.property; |
|  |
| fileTextIN << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << "\n"; |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data), sizeof(data)); |
| cout << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << " " << Flag << "\n"; |
|  |
| } |
|  |
| fileTextIN.close(); |
| fileBinaryIN.close(); |
|  |
| while (true) |
| { |
| int c; |
| cout << "Выберете дейстивие: " << endl; |
| cout << "1:создать новый элемент хеш таблици" << endl; |
| cout << "2:найти элемент по ключу" << endl; |
| cout << "3:удалить элемент по ключу" << endl; |
| cin >> c; |
| switch (c) |
| { |
| case 1: |
| { |
|  |
| ofstream fileTextIN("Text.txt"); |
| if (!fileTextIN) |
| { |
| cout << "file is not open" << endl; |
| return 1; |
| } |
|  |
| ofstream fileBinaryIN("Bin.bin", ios::binary); |
| if (!fileBinaryIN) { |
| cout << "bin\_file is not open" << endl; |
| return 1; |
| } |
|  |
| cout << "Создается новый обьект..." << endl; |
|  |
| //----------------------------------------------------------------------- |
| do { |
| data.bank\_code = rand() % (99 - 1 + 1) + 1; |
| } while (unique.count(data.bank\_code) > 0); |
| unique.insert(data.bank\_code); |
| //----------------------------------------------------------------------- |
| a\_1 = rand() % 3; |
| data.name = spisok\_1[a\_1]; |
| //----------------------------------------------------------------------- |
| a\_2 = rand() % 3; |
| data.address = spisok\_2[a\_2]; |
| //----------------------------------------------------------------------- |
| data.property = "Коммерческий"; |
|  |
| table->count++; |
|  |
| if (table->count > table->size) { |
| cout << "переполнение, нужно сделать рехеширование" << endl; |
| table = ReHach(table); |
| } |
|  |
| int Flag = Insert(data.bank\_code, table); |
|  |
| if (Flag != -1) |
| { |
| fileTextIN << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << "\n"; |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data), sizeof(data)); |
| cout << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << " " << Flag << "\n"; |
| } |
| else { |
| cout << "такого нет" << endl; |
| } |
|  |
| for (int i = 0; i < 10; i++) { |
| cout << table->items[i].bank\_code << endl; |
| } |
|  |
| fileTextIN.close(); |
| fileBinaryIN.close(); |
|  |
| break; |
| } |
| case 2: |
| { |
| auto start = chrono::high\_resolution\_clock::now(); |
| cout << "Введите банковский код" << endl; |
| int Key; |
| cin >> Key; |
| int position = Hash\_func(Key, table); |
| if (position == -1) { cout << "Eror" << endl; break; } |
| cout << table->items[position].name << " " << table->items[position].bank\_code << " " << table->items[position].address << " " << table->items[position].property << endl; |
| auto end = chrono::high\_resolution\_clock::now(); |
| chrono::duration<double> duration = end - start; |
| cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl; |
| break; |
| } |
| case 3: |
| { |
| cout << "Введите банковский код" << endl; |
| int Key; |
| cin >> Key; |
| int position = Hash\_func(Key, table); |
| if (position == -1) { cout << "Eror" << endl; break; } |
| table->items[position].bank\_code = DEL; |
| cout << "удалено" << endl; |
| break; |
| } |
| } |
| } |
| } |

**Задание 3.**

Управление бинарным файлом посредством хеш-таблицы.

В заголовочный файл подключить заголовочные файлы: управления хештаблицей, 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 (программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе).

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.

Листинг реализации 3 задания(Листинг 3).

*Листинг 3.*

|  |
| --- |
| ifstream fileTextOUT("Text.txt"); |
| ofstream fileText2IN("Text2.txt"); |
| ofstream fileBinaryIN("Bin.bin", ios::binary, ios\_base::trunc); |
| Ht\_item data1; |
| cout << "Введите банковский код" << endl; |
| int Key; |
| cin >> Key; |
| int position = Hash\_func(Key, table); |
| if (position == -1) { cout << "Eror" << endl; break; } |
| table->items[position].bank\_code = DEL; |
| cout << "удалено" << endl; |
|  |
| char nam[50]; |
| char bank[50]; |
| char add[50]; |
| char prp[50]; |
| string Key\_string = to\_string(Key); |
| int Flag; |
|  |
| for (int i = 0; i < table->size; i++) { |
| fileTextOUT >> nam; |
| fileTextOUT >> bank; |
| if (bank == Key\_string) { |
| Flag = -1; |
| } |
| fileTextOUT >> add; |
| fileTextOUT >> prp; |
| if (Flag != -1) { |
| fileText2IN << nam << " " << bank << " " << add << " " << prp << " " << '\n'; |
| data1.name = nam; |
| data1.bank\_code = (int)bank; |
| data1.address = add; |
| data1.property = prp; |
| } |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data1), sizeof(data1)); |
| cout << "удалено из файла" << endl; |
| } |
| fileText2IN.close(); |
| fileTextOUT.close(); |
| fileBinaryIN.close(); |
|  |
| char buff; |
|  |
| ifstream fin("Text2.txt"); |
| ofstream fil("Text.txt"); |
| while (fin >> buff) |
| { |
| fil << buff; |
| } |
| break; |

Полный листинг программы(Листинг 4.)

*Листинг 4.*

|  |
| --- |
| #include <iostream> |
| #include <fstream> |
| #include <string> |
| #include <chrono> |
| #include <set> |
| #include <vector> |
| #include <algorithm> |
| #define NULL 0 |
| #define CAPACITY 1000001 |
| #define DEL -1 |
|  |
| using namespace std; |
|  |
|  |
|  |
| struct Ht\_item |
| { |
| string name; |
| int bank\_code; //ключь |
| string address; |
| string property; |
| }; |
|  |
| struct HashTable { |
| Ht\_item\* items; |
| int size; |
| int count; |
| }; |
|  |
| //создвние |
|  |
| //создание хеш таблици |
| HashTable\* create\_table(int size) { |
| auto start = chrono::high\_resolution\_clock::now(); |
| HashTable\* table; |
| table = new HashTable; |
| table->size = size; |
| table->count = 0; |
| table->items = new Ht\_item[table->size]; |
| for (int i = 0; i < table->size; i++) { |
| table->items[i].bank\_code = NULL; |
| } |
| auto end = chrono::high\_resolution\_clock::now(); |
| chrono::duration<double> duration = end - start; |
| cout << "Время выполнения создания таблици: " << duration.count() << " секунд." << endl; |
| return table; |
| } |
|  |
| //двойная хеш функция |
| int Hash\_func(int Key, HashTable\* table) { |
| int Hf1 = Key % table->size; |
| int Hf2 = 1 + Key % (table->size - 1); |
| int counter = 0; |
| //cout << "size " << table->size << endl; |
| for (int i = 0; i < table->size; i++) { |
| if (table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == NULL || table->items[((Hf1 + (i \* Hf2)) % table->size)].bank\_code == Key){ |
| return ((Hf1 + (i \* Hf2)) % table->size); |
| } |
| } |
| return -1; |
| } |
|  |
| int Insert(int Key, HashTable\* table); |
|  |
| //функция рехеширования |
| HashTable\* ReHach(HashTable\* table) { |
| HashTable\* ReHachTable = new HashTable; |
|  |
| int size = table->size; |
|  |
| ReHachTable = create\_table(size \* 2); |
| int posistion; |
|  |
| for (int i = 0; i < size; i++) { |
| if(table->items[i].bank\_code != NULL) |
| { |
| posistion = Insert(table->items[i].bank\_code, ReHachTable); |
| ReHachTable->items[posistion].address = table->items[i].address; |
| ReHachTable->items[posistion].bank\_code = table->items[i].bank\_code; |
| ReHachTable->items[posistion].name = table->items[i].name; |
| ReHachTable->items[posistion].property = table->items[i].property; |
| } |
| } |
| delete[] table; |
| return ReHachTable; |
| } |
|  |
| //вставка |
| int Insert(int Key, HashTable\* table) { |
|  |
|  |
|  |
| int position = Hash\_func(Key, table); |
| //cout << "count " << table->count << endl; |
|  |
| if (position == -1) { return -1;} |
| cout << "position " << position << " Key " << Key << endl; |
|  |
|  |
|  |
| //cout << "bank\_code " << table->items[position].bank\_code << endl; |
| return position; |
| } |
|  |
|  |
|  |
| int main() { |
|  |
| setlocale(LC\_ALL, "ru"); |
|  |
| HashTable\* table; |
|  |
| table = create\_table(CAPACITY); |
|  |
| ofstream fileTextIN("Text.txt"); |
| if (!fileTextIN) |
| { |
| cout << "file is not open" << endl; |
| return 1; |
| } |
|  |
| ofstream fileBinaryIN("Bin.bin", ios::binary); |
| if (!fileBinaryIN) { |
| cout << "bin\_file is not open" << endl; |
| return 1; |
| } |
|  |
| set<int> unique; |
| Ht\_item data; |
| char spisok\_1[3][20] = {"Тинькоф", "Сбербанк", "Втб"}; |
| char spisok\_2[3][35] = { "Москва", "Санкт-Петербург", "Екатеринбург" }; |
| int a\_1, a\_2; |
|  |
| for (int i = 0; i < 1000; ++i) |
| { |
| //----------------------------------------------------------------------- |
| do { |
| data.bank\_code = rand() % (1000 - 1 + 1) + 1; |
| } while (unique.count(data.bank\_code) > 0); |
| unique.insert(data.bank\_code); |
| //----------------------------------------------------------------------- |
| a\_1 = rand() % 3; |
| data.name = spisok\_1[a\_1]; |
| //----------------------------------------------------------------------- |
| a\_2 = rand() % 3; |
| data.address = spisok\_2[a\_2]; |
| //----------------------------------------------------------------------- |
| data.property = "Коммерческий"; |
|  |
| table->count++; |
| int Flag = Insert(data.bank\_code, table); |
|  |
| table->items[Flag].name = data.name; |
| table->items[Flag].bank\_code = data.bank\_code; |
| table->items[Flag].address = data.address; |
| table->items[Flag].property = data.property; |
|  |
| fileTextIN << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << "\n"; |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data), sizeof(data)); |
| cout << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << " " << Flag << "\n"; |
|  |
| } |
|  |
| fileTextIN.close(); |
| fileBinaryIN.close(); |
|  |
| while (true) |
| { |
| int c; |
| cout << "Выберете дейстивие: " << endl; |
| cout << "1:создать новый элемент хеш таблици" << endl; |
| cout << "2:найти элемент по ключу" << endl; |
| cout << "3:удалить элемент по ключу" << endl; |
| cin >> c; |
| switch (c) |
| { |
| case 1: |
| { |
|  |
| ofstream fileTextIN("Text.txt"); |
| if (!fileTextIN) |
| { |
| cout << "file is not open" << endl; |
| return 1; |
| } |
|  |
| ofstream fileBinaryIN("Bin.bin", ios::binary); |
| if (!fileBinaryIN) { |
| cout << "bin\_file is not open" << endl; |
| return 1; |
| } |
|  |
| cout << "Создается новый обьект..." << endl; |
|  |
| //----------------------------------------------------------------------- |
| do { |
| data.bank\_code = rand() % (99 - 1 + 1) + 1; |
| } while (unique.count(data.bank\_code) > 0); |
| unique.insert(data.bank\_code); |
| //----------------------------------------------------------------------- |
| a\_1 = rand() % 3; |
| data.name = spisok\_1[a\_1]; |
| //----------------------------------------------------------------------- |
| a\_2 = rand() % 3; |
| data.address = spisok\_2[a\_2]; |
| //----------------------------------------------------------------------- |
| data.property = "Коммерческий"; |
|  |
| table->count++; |
|  |
| if (table->count > table->size) { |
| cout << "переполнение, нужно сделать рехеширование" << endl; |
| table = ReHach(table); |
| } |
|  |
| int Flag = Insert(data.bank\_code, table); |
|  |
| if (Flag != -1) |
| { |
| fileTextIN << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << "\n"; |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data), sizeof(data)); |
| cout << data.name << " " << data.bank\_code << " " << data.address << " " << data.property << " " << Flag << "\n"; |
| } |
| else { |
| cout << "такого нет" << endl; |
| } |
|  |
| for (int i = 0; i < 10; i++) { |
| cout << table->items[i].bank\_code << endl; |
| } |
|  |
| fileTextIN.close(); |
| fileBinaryIN.close(); |
|  |
| break; |
| } |
| case 2: |
| { |
| auto start = chrono::high\_resolution\_clock::now(); |
| cout << "Введите банковский код" << endl; |
| int Key; |
| cin >> Key; |
| int position = Hash\_func(Key, table); |
| if (position == -1) { cout << "Eror" << endl; break; } |
| cout << table->items[position].name << " " << table->items[position].bank\_code << " " << table->items[position].address << " " << table->items[position].property << endl; |
| auto end = chrono::high\_resolution\_clock::now(); |
| chrono::duration<double> duration = end - start; |
| cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl; |
| break; |
| } |
| case 3: |
| { |
| ifstream fileTextOUT("Text.txt"); |
| ofstream fileText2IN("Text2.txt"); |
| ofstream fileBinaryIN("Bin.bin", ios::binary, ios\_base::trunc); |
| Ht\_item data1; |
| cout << "Введите банковский код" << endl; |
| int Key; |
| cin >> Key; |
| int position = Hash\_func(Key, table); |
| if (position == -1) { cout << "Eror" << endl; break; } |
| table->items[position].bank\_code = DEL; |
| cout << "удалено" << endl; |
|  |
| char nam[50]; |
| char bank[50]; |
| char add[50]; |
| char prp[50]; |
| string Key\_string = to\_string(Key); |
| int Flag; |
|  |
| for (int i = 0; i < table->size; i++) { |
| fileTextOUT >> nam; |
| fileTextOUT >> bank; |
| if (bank == Key\_string) { |
| Flag = -1; |
| } |
| fileTextOUT >> add; |
| fileTextOUT >> prp; |
| if (Flag != -1) { |
| fileText2IN << nam << " " << bank << " " << add << " " << prp << " " << '\n'; |
| data1.name = nam; |
| data1.bank\_code = (int)bank; |
| data1.address = add; |
| data1.property = prp; |
| } |
| fileBinaryIN.write(reinterpret\_cast<const char\*>(&data1), sizeof(data1)); |
| cout << "удалено из файла" << endl; |
| } |
| fileText2IN.close(); |
| fileTextOUT.close(); |
| fileBinaryIN.close(); |
|  |
| char buff; |
|  |
| ifstream fin("Text2.txt"); |
| ofstream fil("Text.txt"); |
| while (fin >> buff) |
| { |
| fil << buff; |
| } |
| break; |
| } |
| } |
| } |
| } |

**Вывод.**

Были получены навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).