**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра АПУ**

курсовой проект

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Хеш-таблицы**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 1371 |  | Ларионов Н.М.,  Степанов Р.О. |
| Преподаватель |  | Ряскова Е.Б. |

Санкт-Петербург

2022

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 3 |
|  | Задание на курсовой проект | 5 |
|  | Описание программы | 6 |
|  | Листинг программы | 7 |
|  | Тестирование программы | 17 |
|  | Заключение | 20 |
|  |

**введение**

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

Хеш-таблицей называется структура данных, предназначенная для реализации ассоциативного массива, такого в котором адресация реализуется посредством хеш-функции.

Хеш-функция, или функция отображения, вычисляет для каждого элемента таблицы по его ключу номер (позицию) элемента в массиве, таким образом, преобразует значения ключей к интервалу [0, m – 1], где m — размер хеш-таблицы, m ≪ |U|.

Количество возможных коллизий можно уменьшить, если выполнить два условия:

1) выбрать размер хеш-таблицы с запасом. Если размер таблицы превышает мощность хранимого множества более чем вдвое, вероятность коллизии становится меньше 0,5. Если мощность множества заранее неизвестна, то выбирают некоторый начальный размер, а когда его оказывается недостаточно, таблицу перестраивают с увеличением размера (обычно вдвое);

2) подобрать функцию отображения (хеш-функцию) такую, чтобы все ячейки таблицы были востребованы по возможности с равной вероятностью, независимо от того какое распределение имеют хранящиеся в таблице ключи.

По способу разрешения коллизий различают хеш-таблицы двух типов:

1) Открытое хеширование (метод цепочек).

2) Закрытое хеширование (с открытой адресацией).

Принцип организации хеш-таблицы методом открытого хеширования заключается в реализации логически связанных цепочек, начинающихся в ячейках хеш-таблицы.

В отличие от открытого хеширования закрытое не требует каких-либо дополнительных структур данных. В ячейках таблицы хранятся не указатели, а элементы исходного массива, доступ к каждому из которых осуществляется по хеш-коду ключа, при этом одна ячейка может содержать только один элемент.

Такие задачи часто встречаются на практике, когда приходится обрабатывать большие массивы данных , при этом необходимо обеспечить быстрый доступ к каждому элементу. За счет того, что хеш-таблица создается на основе массива со временем доступа O(1) и ключ, зачастую, вычисляется за такое же время, обеспечивается достаточная скорость доступа к элементам

**Задание на курсовой проект**

1. Организуйте данные из предметной области в виде хеш-таблицы (*первичный ключ хеш-таблицы выделен курсивом*). Метод хеширования также указан в таблице с вариантами заданий. В качестве алгоритма хеширования использовать *метод деления*. Обратите внимание, что для вычисления хеш-функции, значение ключевого поля должно быть целым (для перевода строковых данных в целые можно воспользоваться функцией atoi). Примеры реализации хеш-таблиц с открытым и закрытым хешированием см.

2. Реализуйте следующие функции:

* Меню для выбора функции (используя конструкцию switch — case).
* Добавление записи (в случае закрытого хеширования необходимо отслеживать заполненность хеш-таблицы)
* Проверку корректности вводимой информации.
* Удаление записи
* Просмотр всех имеющихся записей.
* Удаление всех записей.
* Поиск записи по ключевому полю. Пример реализации см.
* Поиск данных по фрагменту одного из не ключевых полей (см. вариант задания). Для поиска подстроки в строке необходимо использовать *прямой поиск*. Подробнее о прямом поиске см.

Вариант 1

|  |  |  |
| --- | --- | --- |
| Предметная область, характеристики | Метод хеширования | Поиск по  фрагменту строки |
| Данные о покупателе:  Id – строка формата «NNNNNN», где N –  цифры. | Закрытое хеширование с линейным  опробованием | Поиск покупателей  с заданной  фамилией |

**Описание программы**

Программа построена на двух пользовательских классах. Класс HashTable описывает параметры и методы хеш-таблицы покупателей, класс customer описывает параметры и методы покупателя.

Стандартные методы обрабатывают данные и работают с хеш-таблицей. Каждому необходимому методу соответствует квази-метод с характерным названием, выполняющий ввод данных, их проверку, вызов стандартного метода для их обработки и вывод данных. Таким образом, фрагменты программы, занимающиеся логикой и интерфейсом отделены друг от друга для более удобного восприятия и работы. Меню выводится отдельной функцией PrintMenu(), функция main() содержит в себе исключительно вызовы функций, выбираемые в зависимости от введенного с консоли значения.

Обращаться к пунктам меню следует через их номера.

**Листинг программы**

#include <iostream>

#include <windows.h>

#include <string>

#include <cstring>

#define SIZE 100 // Размер хеш-таблицы, константа

using namespace std; // Для удобного использования cout и cin

// Без явного указания пространства имен

string ReadString()

{

string input;

cin.clear();

cin.ignore(256, '\n');

getline(cin, input);

return input;

}

int CheckNums(string& s)

{

for (char const& ch : s) {

if (isdigit(ch) == 0)

return false;

}

return true;

}

class customer // Класс покупателя

{

string id; // Строка с id

string name; // Строка с именем

string address; // Строка с адресом

public: // Публичная секция с методами

customer(string id, string name, string adress);

string GetID();

string GetName();

string GetAddress();

};

class HashTable // Класс хеш-таблиц

{

customer\* customers[SIZE]; // Массив указателей ячеек покупателей

int size; // Текущий размер

public: // Публичная секция с методами

HashTable(); // Базовый конструктор таблицы без параметров

void Push(string id, string name, string adress); // Метод для заполнения таблицы

void KvaziPush(); // Обработка значений перед заполнением таблицы

int HashFunc(string id, int shift); // Хеш-функция

int LineTrial(string id); // Функция линейного опробования

void PrintAllCustomers(); // Фунция вывода всех покупателей

int LineFindCustomer(string name); // Функция линейного поиска покупателя

void KvaziLineFindCustomer(); // Функция обработкии линейного поиска

int HashFindCustomer(string name); // Функция хешированного поиска покупателя

void KvaziHashFindCustomer(); // Функция обработкии хешированного поиска

int ismatch(string first, string second); // Функция для сравнения id и фамилий

void clear(); // Удаление всей таблицы

void dell(string id); // Удаление покупателя по id

void predell();

};

HashTable::HashTable() // Базовый конструктов таблицы без параметров

{

for (int i = 0; i < SIZE; i++) // Цикл стартовой иницилизации массива

{

customers[i] = nullptr; // Присваивание пустой ячейки

}

size = 0; // Стартовый размер равен нулю

}

void HashTable::Push(string id, string name, string adress) // Функция добавления покупателя в таблицу

{

if (size == SIZE) // Проверка таблицы на переполнение

{

cout << "\nТаблица переполнена\n";

return; // Выход из функции

}

int index = HashFunc(id, 0); // Получаем индекс хеш-таблицы

if (!customers[index]) // Решение проблемы коллизии

{ // Проверка ячейки на пустоту

customers[index] = new customer(id, name, adress); // В таблице появляется новая ячейка покупателя

size += 1;

}

else // Если очевидный индекс занят

{ // Вызов функции линейного опробования

index = LineTrial(id);

customers[index] = new customer(id, name, adress); // В таблице появляется новая ячейка покупателя

size += 1;

}

}

void HashTable::KvaziPush() // Квази-функция для обработки информации перед добавленем

{

string id, name, adress;

string second\_name, first\_name, father\_name;

cout << "Введите id покупателя:\n>>";

cin >> id;

if (!CheckNums(id))

{

cout << "\nНекорректный id\n";

return;

}

if (id.size() != 6) // Проверка на размер id

{

cout << "\nНекорректный id\n";

return;

}

if (HashFindCustomer(id) != -1)

{

cout << "\nПокупатель с таким id уже присутствует в таблице\n";

return;

}

cout << "Введите ФИО покупателя\n>>";

cin >> second\_name >> first\_name >> father\_name;

name = second\_name + " " + first\_name + " " + father\_name;

cout << "Введите адрес покупателя\n>>";

adress = ReadString();

Push(id, name, adress);

}

int HashTable::HashFunc(string id, int shift) // Хеш-функция

{ // Однозначно возвращает индекс покупателя в хеш-таблице

// shift - сдвиг, решение проблемы колизии

// сдвиг будет увеличиваться на единицу до обнаружения новой свободной ячейки

int hash = (stoi(id) + shift) % SIZE; // Преобразование string в int

// Хеш - остаток от деления шестизначного id на размер таблицы

return hash;

}

int HashTable::LineTrial(string id) // Функция линейного опробования

{ // Возвращает следующую свободную ячейку

int start = HashFunc(id, 0); // Старт линейного опробования

int shift = 0; // Функция начинает сдвиг с нуля

for (int i = start; i < SIZE; i++) { // Обход таблицы вправо

shift += 1; // Увеличиваем сдвиг на единицу

int index = HashFunc(id, shift); // Вызов хеш-функции со сдвигом

if (!customers[index]) // Если ячейка покупателя в таблице пуста

{

return index; // Вернуть свободный индекс хеш-таблицы

}

}

for (int i = 0; i < start; i++) { // Обход таблицы вправо

shift += 1; // Увеличиваем сдвиг на единицу

int index = HashFunc(id, shift);// Вызов хеш-функции со сдвигом

if (!customers[index]) // Если ячейка покупателя в таблице пуста

{

return index; // Вернуть свободный индекс хеш-таблицы

}

}

}

void HashTable::PrintAllCustomers()

{

if (size == 0) {

cout << "Таблица с покупателями пуста\n";

return;

}

cout << "Список покупателей:\n";

for (int i = 0; i < SIZE; i++)

{

if (customers[i] != nullptr)

{

cout << "\nID покупателя: ";

cout << customers[i]->GetID();

cout << "\nФИО покупателя: ";

cout << customers[i]->GetName();

cout << "\nАдресс покупателя: ";

cout << customers[i]->GetAddress();

cout << "\n";

}

}

}

int HashTable::LineFindCustomer(string name) // Функция линейного поиска покупателя

{

for (int i = 0; i < SIZE; i++) // Линейный обход таблицы

{

if (!customers[i])

{

continue;

}

string full\_name = customers[i]->GetName(); // Получение полного имени покупателя

// Вычленение фамилии через метод строки

string second\_name = full\_name.substr(0 /\*Начальный индекс\*/, full\_name.find(" ") /\*Количество символов через индекс первого пробела\*/);

if (ismatch(second\_name, name))

{

return i; // Возвращает номер ячейки с совпадающей фамилией

}

}

return -1;

}

void HashTable::KvaziLineFindCustomer()

{

if (size == 0) { //

cout << "Таблица с покупателями пуста\n";

return;

}

string name;

cout << "Введите фамилию покупателя\n>>";

cin >> name;

if (LineFindCustomer(name) == -1)

{

cout << "Покупатель с такой фамилией в таблице отсутствует\n";

return;

}

int index;

index = LineFindCustomer(name);

cout << "ID покупателя: ";

cout << customers[index]->GetID();

cout << "\nФИО покупателя: ";

cout << customers[index]->GetName();

cout << "\nАдресс покупателя: ";

cout << customers[index]->GetAddress();

cout << "\n";

}

int HashTable::HashFindCustomer(string id) // Функция хешированного поиска покупателя

{

int start = HashFunc(id, 0); // Получаем стартовый очевидный id

int shift = 0; // Смещение индекса хеш-таблцицы вправо

for (int i = start; i < SIZE; i++)

{

shift += 1; // Увеличение смещение на единицу вправо

int index = HashFunc(id, shift);

if (!customers[index]) // Если ячейка таблицы с покупателем пустая

{

continue; // Следующая итерация цикла for(i)

}

if (ismatch(customers[index]->GetID(), id)) // Если id совпали

{

return index; // Возвращение индекса покупателя

}

}

for (int i = 0; i < start; i++)

{

shift += 1; // Увеличение смещение на единицу вправо

int index = HashFunc(id, shift);

if (!customers[index]) // Если ячейка таблицы с покупателем пустая

{

continue; // Следующая итерация цикла for(i)

}

if (ismatch(customers[index]->GetID(), id)) // Если id совпали

{

return index; // Возвращение индекса покупателя

}

}

return -1;

}

void HashTable::KvaziHashFindCustomer()

{

if (size == 0) { //

cout << "Таблица с покупателями пуста\n";

return;

}

string id;

cout << "Введите id покупателя\n>>";

cin >> id;

if (!CheckNums(id))

{

cout << "\nНекорректный id\n";

return;

}

if (HashFindCustomer(id) == -1)

{

cout << "Покупатель с таким id в таблице отсутствует\n";

return;

}

int index;

index = HashFindCustomer(id);

cout << "ID покупателя: ";

cout << customers[index]->GetID();

cout << "\nФИО покупателя: ";

cout << customers[index]->GetName();

cout << "\nАдресс покупателя: ";

cout << customers[index]->GetAddress();

cout << "\n";

}

int HashTable::ismatch(string first, string second) { // Функция проверки совпадения данных

if (first == second)

{

return 1;

}

return 0;

}

void HashTable::clear()

{

for (int i = 0; i < SIZE; i++) { // Проход по всей хеш-таблице

if (!customers[i]) // Если ячейка незаполнена ее удаление не требуется

{

continue; // Переход на следующую итерацию цикла for(i)

}

delete customers[i]; // Удаление покупателя

customers[i] = nullptr; // Ячейка хеш-таблицы с покупателем пустая

;

}

size = 0;

}

void HashTable::dell(string id)

{

int index = HashFindCustomer(id); // Получение индекса покупателя с нужным id

delete customers[index]; // Удаление покупателя

customers[index] = nullptr; // Ячейка хеш-таблицы с покупателем пустая

size -= 1;

}

void HashTable::predell()

{

string id;

cout << "\nВведите id покупателя, которого нужно удалить из таблицы:\n>>";

cin >> id;

if (!CheckNums(id))

{

cout << "\nНекорректный id\n";

return;

}

int index = HashFindCustomer(id); // Получение индекса покупателя с нужным id

if (index == -1) { // В случая если покупатель не найден

cout << "Покупатель с таким id не найден\n";

return;

}

dell(id);

}

customer::customer(string id, string name, string adress) // Базовый конструктор покупателя с параметрами

{

// this - указатель на текущий объект (класса)

this->id = id;

this->name = name;

this->address = adress;

// Конструктор при создании покупателя (В момент заполнения таблицы)

}

string customer::GetID()

{ // Метод получения id

return id; // Возвращает id

}

string customer::GetName()

{ // Метод получения имени

return name; // Возвращает имя

}

string customer::GetAddress()

{ // Метод получения адреса

return address; // Возвращает имя

}

void PrintMenu()

{

cout << "\nМеню\n"

"1 - Добавить запись покупателя\n"

"2 - Удалить запись покупателя\n"

"3 - Вывести все записи покупателей в таблице\n"

"4 - Хешированный поиск покупателя по id\n"

"5 - Линейный поиск покупателя через фамилию\n"

"6 - Удалить хеш-таблицу\n"

"0 - Выход из программы\n";

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251); // Установка русской расскладки

HashTable Table;

// Проверка id на шестеру

int console = -1;

string name;

while (console != 0)

{

PrintMenu();

cout << ">>";

cin >> console;

switch (console)

{

case 1:

Table.KvaziPush();

break;

case 2:

Table.predell();

break;

case 3:

Table.PrintAllCustomers();

break;

case 4:

Table.KvaziHashFindCustomer();

break;

case 5:

Table.KvaziLineFindCustomer();

break;

case 6:

Table.clear();

break;

}

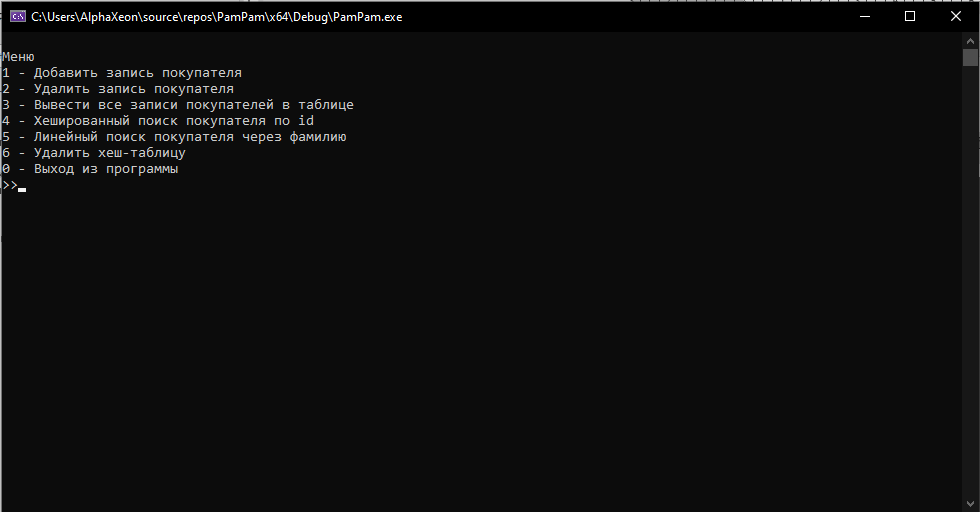
}

return 0;

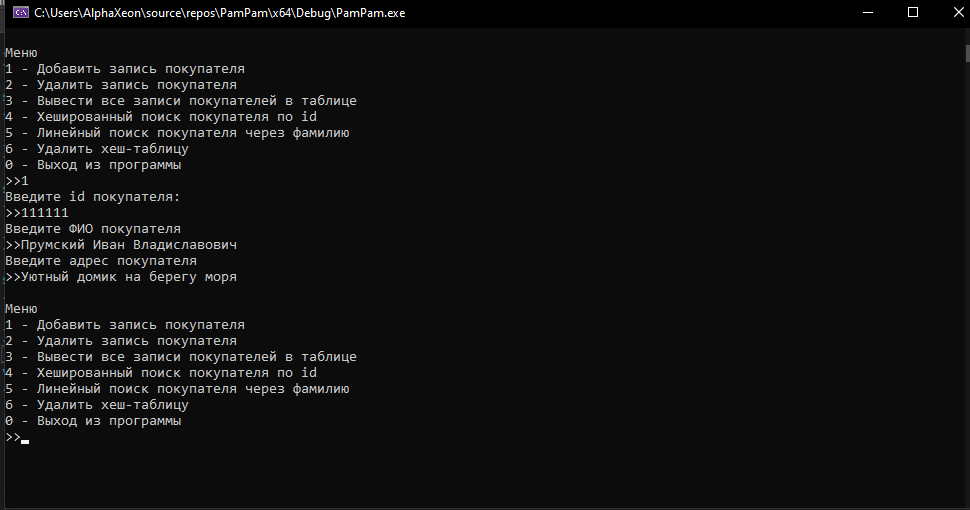
}

**Тестирование программы**

Меню программы



Добавление покупателя



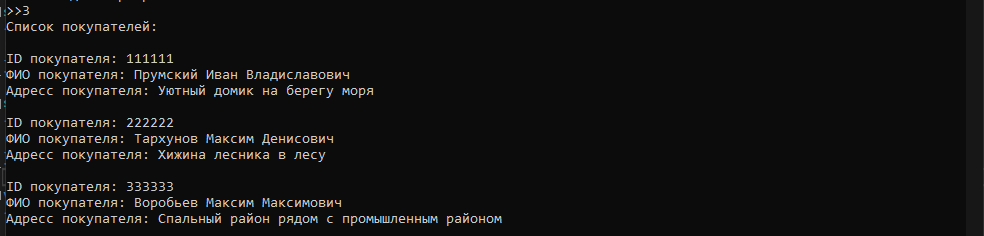
Проверка ввода



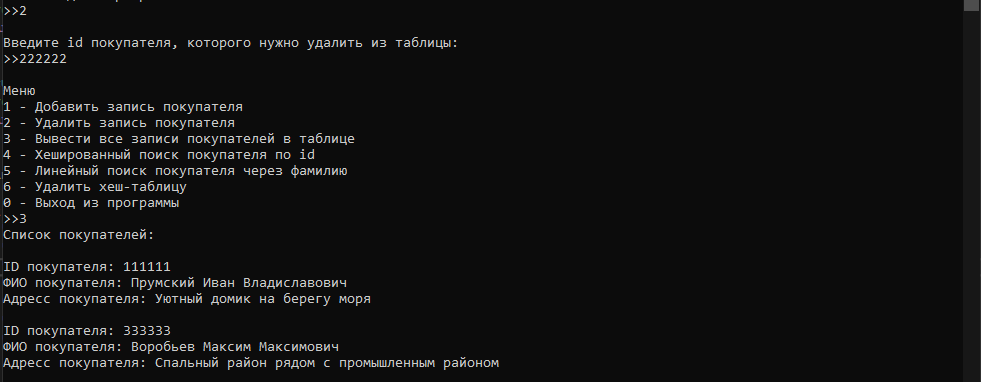




Вывод всех покупателей



Удаление покупателя



Хешированный поиск по id покупателя



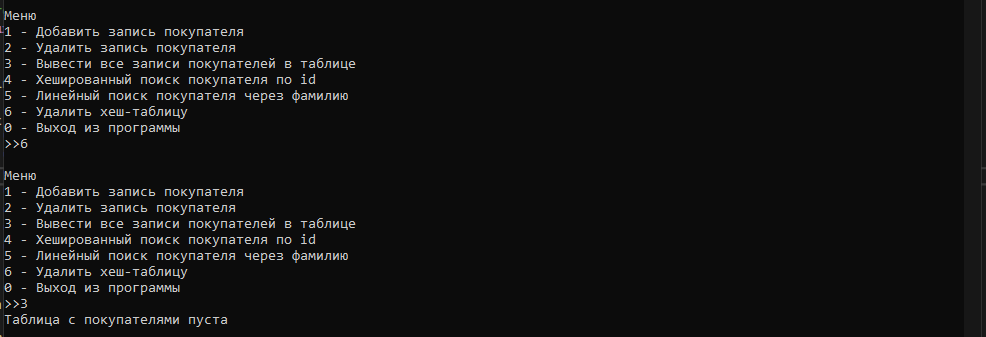
Линейный поиск по фамилии покупателя



(Также с проверкой ввода)

…

Удаление таблицы покупателей



**Заключение**

По итогу курсового проектирования были разработаны две программы с закрытым хешированием и линейным опробованием для данных о покупателей. Была проведена работа над изучением логики и принципом построения хеш-таблиц с различными методами хеширования.