 Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 14

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы c цепочками»

Выполнил:

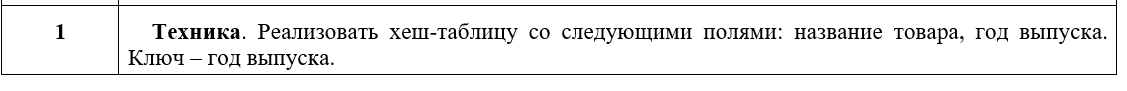
Студент 1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: асс. Андронова М.В.

2024, Минск

Вариант 1



#include <iostream>

#include <Windows.h>

#include <vector>

#include <ctime>

#include <string>

#include <chrono>

#include <random>

using namespace std;

class HashTable {

public:

// Структура для хранения узла в цепочке

struct HashNode {

int year; // год выпуска

string name; // навзание

HashNode\* next; // Указатель на следующий узел в цепочке

// Конструктор для инициализации узла

HashNode(int rn, const string& rt)

: year(rn), name(rt), next(nullptr) {}

};

private:

size\_t tableSize; // Размер хеш-таблицы

vector<HashNode\*> table; // Вектор для хранения указателей на узлы

vector<int> primes = { 29, 31, 37, 41, 43, 47, 53, 59, 61, 67 }; // Список простых чисел для универсального хеширования

int primeIndex = 0; // Индекс для выбора простого числа для хеширования

int a, b; // Коэффициенты для универсального хеширования

int p; // Простое число больше размера таблицы

// Универсальная хеш-функция

int hashFunction(int key) {

return ((a \* key + b) % p) % tableSize;

}

// Генерация случайных параметров для хеш-функции

void generateHashParameters() {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dist(1, p - 1);

a = dist(gen);

b = dist(gen);

}

public:

// Конструктор для инициализации хеш-таблицы

HashTable(size\_t size) : tableSize(size), table(size, nullptr) {

// Находим простое число больше размера таблицы

p = \*upper\_bound(primes.begin(), primes.end(), tableSize);

// Генерируем параметры для хеш-функции

generateHashParameters();

}

// Деструктор для освобождения памяти

~HashTable() {

for (auto node : table) {

while (node) {

HashNode\* temp = node;

node = node->next;

delete temp;

}

}

}

// Вставка элемента в хеш-таблицу

void insert(int year, const string& name) {

int hashIndex = hashFunction(year);

HashNode\* newNode = new HashNode(year, name);

if (!table[hashIndex]) {

table[hashIndex] = newNode;

}

else {

HashNode\* temp = table[hashIndex];

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Удаление элемента из хеш-таблицы

void remove(int year) {

int hashIndex = hashFunction(year);

HashNode\* node = table[hashIndex];

HashNode\* prev = nullptr;

while (node) {

if (node->year == year) {

if (prev) {

prev->next = node->next;

}

else {

table[hashIndex] = node->next;

}

delete node;

cout << "Товар удален." << endl;

return;

}

prev = node;

node = node->next;

}

cout << "Ошибка: товар не найден." << endl;

}

// Поиск элемента в хеш-таблице

void search(int year) {

int hashIndex = hashFunction(year);

HashNode\* node = table[hashIndex];

while (node) {

if (node->year == year) {

cout << "Найден товар: " << node->name << ", хеш: " << hashIndex << endl;

return;

}

node = node->next;

}

cout << "Ошибка: товар не найден." << endl;

}

// Вывод содержимого хеш-таблицы

void display() {

for (size\_t i = 0; i < table.size(); ++i) {

if (table[i]) {

HashNode\* node = table[i];

cout << "Хеш " << i << ": ";

while (node) {

cout << " -> [" << node->year << ": " << node->name << "]";

node = node->next;

}

cout << endl;

}

}

}

};

// Функция для отображения меню

int menu() {

int choice;

cout << "\nМеню:\n";

cout << "1. Добавить товар\n";

cout << "2. Удалить товар\n";

cout << "3. Найти товар\n";

cout << "4. Вывести всЁ\n";

cout << "5. Выйти\n";

cout << "Ваш выбор: ";

cin >> choice;

return choice;

}

int main() {

setlocale(LC\_CTYPE, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Выберите размер таблицы: 16 / 32 / 64 / 128 \n";

size\_t size;

while (true) {

cin >> size;

if (size == 16 || size == 32 || size == 64 || size == 128)

break;

cout << "\nОшибка: попробуйте снова: ";

}

HashTable table(size); // Создание хеш-таблицы

while (true) {

int choice = menu();

if (choice < 1 || choice > 5) {

cout << "Ошибка: неверный выбор." << endl;

continue;

}

if (choice == 5) break;

int routeNumber;

string route;

switch (choice) {

case 1:

cout << "\nВведите год выпуска: ";

cin >> routeNumber;

cin.ignore();

cout << "\nВведите название товара ";

getline(cin, route);

table.insert(routeNumber, route);

break;

case 2:

cout << "\nВведите год ";

cin >> routeNumber;

table.remove(routeNumber);

break;

case 3: {

cout << "\nВведите год ";

cin >> routeNumber;

auto start = chrono::high\_resolution\_clock::now(); // Начало измерения времени

table.search(routeNumber);

auto end = chrono::high\_resolution\_clock::now(); // Конец измерения времени

cout << "Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() \* 1e-9 << " секунд" << endl;

break;

}

case 4:

table.display();

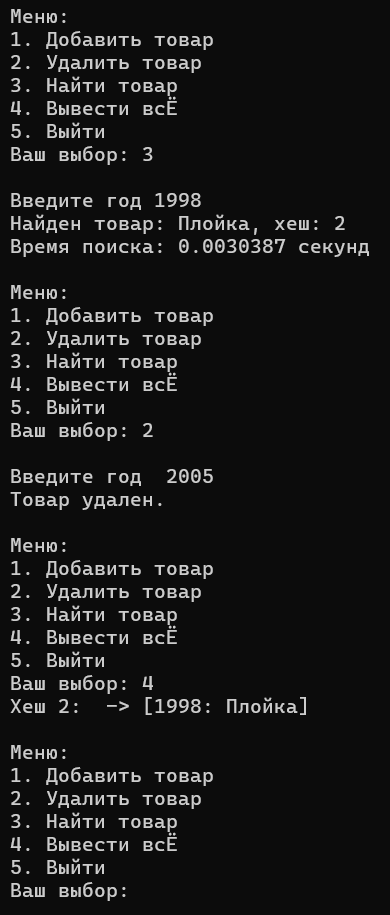
break;

}

}

return 0;

}



Дополнительные задания

1. Составить хеш-таблицу, содержащую буквы и количество их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в хеш-таблице.

#include <iostream>

#include <unordered\_map>

#include <string>

#include <Windows.h>

using namespace std;

// Функция для создания хеш-таблицы, содержащей буквы и количество их вхождений

void createHashTable(unordered\_map<char, int>& hashTable, const string& input) {

for (char ch : input) {

if (isalpha(ch)) { // Проверяем, является ли символ буквой

hashTable[ch]++; // Увеличиваем счетчик вхождений буквы в хеш-таблице

}

}

}

// Функция для вывода хеш-таблицы на экран

void printHashTable(const unordered\_map<char, int>& hashTable) {

for (const auto& entry : hashTable) {

cout << entry.first << ": " << entry.second << endl; // Выводим букву и количество ее вхождений

}

}

// Функция для поиска буквы в хеш-таблице

void searchLetter(const unordered\_map<char, int>& hashTable, char letter) {

auto it = hashTable.find(letter); // Ищем букву в хеш-таблице

if (it != hashTable.end()) {

cout << "Буква '" << letter << "' найдена с частотой: " << it->second << endl;

}

else {

cout << "Буква '" << letter << "' не найдена в хеш-таблице." << endl;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Запрашиваем у пользователя ввод строки

string input;

cout << "Введите строку: ";

getline(cin, input);

// Создаем хеш-таблицу для хранения букв и их частоты

unordered\_map<char, int> hashTable;

// Заполняем хеш-таблицу на основе введенной строки

createHashTable(hashTable, input);

// Выводим построенную хеш-таблицу на экран

cout << "Построенная хеш-таблица:" << endl;

printHashTable(hashTable);

// Осуществляем поиск буквы, введенной пользователем

char letter;

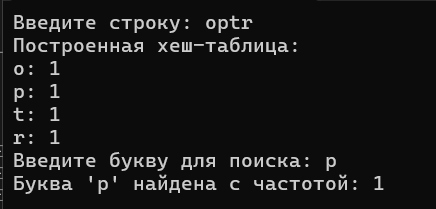
cout << "Введите букву для поиска: ";

cin >> letter;

searchLetter(hashTable, letter);

return 0;

}



В текстовом файле содержатся целые числа. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа в хеш-таблице. Сравнить результаты количества сравнений при различном наборе данных в файле.

#include <iostream>

#include <fstream>

#include <unordered\_set>

#include <vector>

#include <cstdlib>

// Функция для создания хеш-таблицы из чисел текстового файла

std::unordered\_set<int> createHashTable(const std::string& filename) {

std::ifstream file(filename);

if (!file.is\_open()) {

std::cerr << "Ошибка при открытии файла.\n";

return {};

}

std::unordered\_set<int> hashTable;

int number;

while (file >> number) {

hashTable.insert(number);

}

return hashTable;

}

// Функция для поиска числа в хеш-таблице

bool searchNumber(const std::unordered\_set<int>& hashTable, int number) {

return hashTable.find(number) != hashTable.end();

}

int main() {

setlocale(LC\_ALL, "RUS");

std::string filename;

std::cout << "Введите имя текстового файла: ";

std::cin >> filename;

// Создаем хеш-таблицу из чисел текстового файла

std::unordered\_set<int> hashTable = createHashTable(filename);

// Выводим количество элементов в хеш-таблице

std::cout << "Количество элементов в хеш-таблице: " << hashTable.size() << std::endl;

// Поиск числа в хеш-таблице

int numberToSearch;

std::cout << "Введите целое число для поиска: ";

std::cin >> numberToSearch;

if (searchNumber(hashTable, numberToSearch)) {

std::cout << "Число найдено в хеш-таблице.\n";

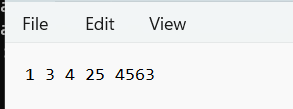
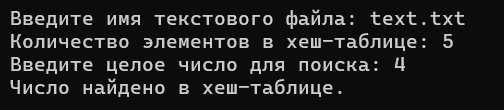
}

else {

std::cout << "Число не найдено в хеш-таблице.\n";

}

return 0;

}

Построить хеш-таблицу из слов произвольного текстового файла, задав ее размерность с экрана. Вывести построенную таблицу слов на экран. Осуществить поиск введенного слова. Выполнить программу для различных размерностей таблицы и сравните количество сравнений. Удалить все слова, начинающиеся на указанную букву, выведите таблицу.

#include <iostream>

#include <fstream>

#include <unordered\_map>

#include <string>

using namespace std;

// Функция для создания хеш-таблицы из слов текстового файла

void createHashTable(unordered\_map<string, int>& hashTable, const string& filename) {

ifstream file(filename); // Открываем файл для чтения

if (!file.is\_open()) {

cerr << "Не удалось открыть файл." << endl;

return;

}

string word;

while (file >> word) {

hashTable[word]++; // Увеличиваем счетчик вхождений слова в хеш-таблице

}

file.close(); // Закрываем файл

}

// Функция для вывода хеш-таблицы на экран

void printHashTable(const unordered\_map<string, int>& hashTable) {

for (const auto& entry : hashTable) {

cout << entry.first << ": " << entry.second << endl; // Выводим слово и количество его вхождений

}

}

// Функция для поиска слова в хеш-таблице

void searchWord(const unordered\_map<string, int>& hashTable, const string& word) {

auto it = hashTable.find(word); // Ищем слово в хеш-таблице

if (it != hashTable.end()) {

cout << "Слово '" << word << "' найдено с частотой: " << it->second << endl;

}

else {

cout << "Слово '" << word << "' не найдено в хеш-таблице." << endl;

}

}

// Функция для удаления всех слов, начинающихся на указанную букву

void deleteWordsStartingWith(unordered\_map<string, int>& hashTable, char letter) {

for (auto it = hashTable.begin(); it != hashTable.end(); ) {

if (it->first[0] == letter) {

it = hashTable.erase(it); // Удаляем слово и получаем итератор на следующий элемент

}

else {

++it; // Переходим к следующему элементу

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

// Задаем размерность таблицы (не используется напрямую, так как unordered\_map управляет размером сам)

int tableSize;

cout << "Введите предполагаемое количество различных слов в файле (размерность хеш-таблицы): ";

cin >> tableSize;

// Создаем хеш-таблицу для хранения слов и их частоты

unordered\_map<string, int> hashTable;

hashTable.reserve(tableSize); // Резервируем место в хеш-таблице для повышения эффективности

// Считываем слова из файла и создаем хеш-таблицу

string filename;

cout << "Введите имя файла: ";

cin >> filename;

createHashTable(hashTable, filename);

// Выводим построенную хеш-таблицу на экран

cout << "Построенная хеш-таблица:" << endl;

printHashTable(hashTable);

// Осуществляем поиск слова, введенного пользователем

string word;

cout << "Введите слово для поиска: ";

cin >> word;

searchWord(hashTable, word);

// Удаляем все слова, начинающиеся на указанную букву

char letter;

cout << "Введите букву для удаления всех слов, начинающихся на нее: ";

cin >> letter;

deleteWordsStartingWith(hashTable, letter);

// Выводим хеш-таблицу после удаления слов

cout << "Хеш-таблица после удаления слов, начинающихся на букву '" << letter << "':" << endl;

printHashTable(hashTable);

return 0;

}

