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”

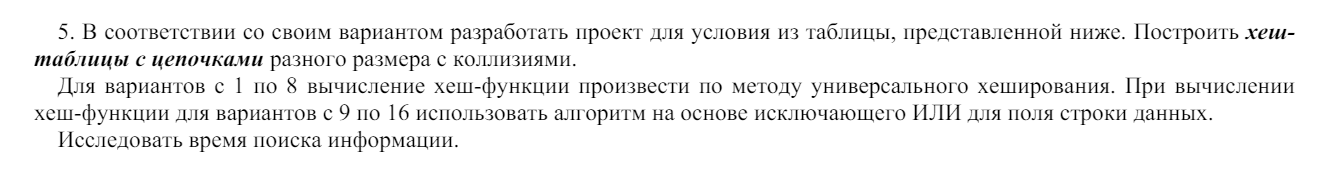
Выполнил:

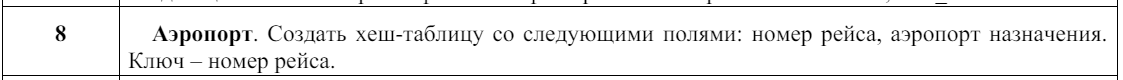
Студент 1 курса 6 группы

Кравченко Сергей Сергеевич

Преподаватель: асс. Андронова М.В.

2024





Main.cpp

#include "Hash\_Chain.h"

#include <iostream>

int main() {

setlocale(LC\_ALL, "RUS");

// Создаем хеш-таблицу с размером 10 и хеш-функцией, использующей значение ключа напрямую

hashTC::Object h1 = hashTC::create(10, [](void\* x) { return \*((int\*)x); });

struct Flight {

int flightNumber;

std::string destinationAirport;

};

int choice;

int flightNumber;

std::string destinationAirport;

Flight result;

while (true) {

std::cout << "1 - Добавить рейс" << std::endl;

std::cout << "2 - Найти рейс по номеру" << std::endl;

std::cout << "3 - Удалить рейс по номеру" << std::endl;

std::cout << "4 - Вывести все рейсы" << std::endl;

std::cout << "0 - Выход" << std::endl;

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice) {

case 0:

std::cout << "Выход из программы." << std::endl;

return 0;

case 1:

std::cout << "Введите номер рейса: ";

std::cin >> flightNumber;

std::cout << "Введите аэропорт назначения: ";

std::cin >> destinationAirport;

h1.insert(&flightNumber); // Добавляем рейс

break;

case 2:

std::cout << "Введите номер рейса для поиска: ";

std::cin >> flightNumber;

if (h1.search(&flightNumber) != nullptr) {

std::cout << "Рейс найден." << std::endl;

}

else {

std::cout << "Рейс не найден." << std::endl;

}

break;

case 3:

std::cout << "Введите номер рейса для удаления: ";

std::cin >> flightNumber;

h1.deleteByData(&flightNumber); // Удаляем рейс

std::cout << "Рейс удален." << std::endl;

break;

case 4:

std::cout << "Все рейсы:" << std::endl;

h1.Scan(); // Выводим все рейсы

std::cout << std::endl;

break;

default:

std::cout << "Неверный выбор. Попробуйте еще раз." << std::endl;

break;

}

}

return 0;

}

Hash\_Chain.h

#pragma once

#include "Lists.h"

#include <random>

namespace hashTC {

// Структура для хранения объектов хеш-таблицы

struct Object {

int size; // Размер хеш-таблицы

int(\*FunKey)(void\*); // Функция для вычисления хеша

listx::Object\* Hash; // Хеш-таблица

// Конструктор для инициализации размера и функции хеширования

Object(int siz, int(\*f)(void\*)) {

size = siz;

FunKey = f;

Hash = new listx::Object[size];

};

// Функция хеширования для вычисления индекса в хеш-таблице

int hashFunction(void\* data);

// Вставка элемента в хеш-таблицу

bool insert(void\* data);

// Поиск элемента в хеш-таблице

listx::Element\* search(void\* data);

// Удаление элемента из хеш-таблицы по данным

bool deleteByData(void\* data);

// Обход всех элементов хеш-таблицы

void Scan();

};

// Создание нового объекта хеш-таблицы

Object create(int size, int(\*f)(void\*));

// Глобальный генератор случайных чисел для использования в хеш-функциях

extern std::default\_random\_engine generator;

// Функция для генерации универсального хеш-значения

int universalHash(void\* data, int size);

}

Lists.h

#pragma once

// Определение пространства имен listx, содержащего структуры и функции для работы с двусвязным списком.

namespace listx {

// Структура Element представляет элемент в двусвязном списке.

struct Element {

// Указатель на предыдущий элемент в списке.

Element\* prev;

// Указатель на следующий элемент в списке.

Element\* next;

// Указатель на данные, хранящиеся в элементе.

void\* data;

// Конструктор для создания нового элемента с заданными предыдущим и следующим элементами, а также данными.

Element(Element\* pr, void\* dat, Element\* nt) {

prev = pr;

data = dat;

next = nt;

}

// Метод для получения следующего элемента в списке.

Element\* getNext() {

return next;

}

// Метод для получения предыдущего элемента в списке.

Element\* getPrev() {

return prev;

}

};

// Специальный элемент NIL, используемый для обозначения конца списка.

static Element\* NIL = nullptr;

// Структура Object представляет двусвязный список.

struct Object {

// Указатель на первый элемент списка.

Element\* head;

// Конструктор для создания нового списка.

Object() {

head = NIL;

}

// Методы для работы со списком:

// Получение первого элемента списка.

Element\* getFirst();

// Получение последнего элемента списка.

Element\* getLast();

// Поиск элемента по данным.

Element\* search(void\* data);

// Вставка нового элемента в список.

bool insert(void\* data);

// Удаление элемента по указателю на элемент.

bool deleteByElement(Element\* e);

// Удаление элемента по данным.

bool deleteByData(void\* data);

// Просмотр всех элементов списка.

void scan();

};

// Функция для создания нового объекта списка.

Object create();

}

Hash\_Table.cpp

#include "Hash\_Chain.h"

#include <iostream>

// Определение пространства имен hashTC, содержащего реализацию хеш-таблицы с цепочками.

namespace hashTC {

// Реализация хеш-функции для объекта Object.

int Object::hashFunction(void\* data) {

// Вычисление хеш-значения для данных, используя предоставленную функцию FunKey.

// Результат делится на размер хеш-таблицы для получения индекса.

return (FunKey(data) % size);

}

// Вставка данных в хеш-таблицу.

bool Object::insert(void\* data) {

// Вставка данных в цепочку, соответствующую вычисленному хеш-значению.

return (Hash[hashFunction(data)].insert(data));

}

// Удаление данных из хеш-таблицы по данным.

bool Object::deleteByData(void\* data) {

// Удаление данных из цепочки, соответствующей вычисленному хеш-значению.

return (Hash[hashFunction(data)].deleteByData(data));

}

// Поиск данных в хеш-таблице.

listx::Element\* Object::search(void\* data) {

// Поиск данных в цепочке, соответствующей вычисленному хеш-значению.

return Hash[hashFunction(data)].search(data);

}

// Создание нового объекта хеш-таблицы с заданным размером и функцией хеширования.

Object create(int size, int(\*f)(void\*)) {

return Object(size, f);

}

// Генератор случайных чисел для использования в универсальной хеш-функции.

std::default\_random\_engine generator;

// Универсальная хеш-функция, использующая генератор случайных чисел.

int universalHash(void\* data, int size) {

// Создание равномерного распределения в заданном диапазоне.

std::uniform\_int\_distribution<int> distribution(0, size - 1);

// Генерация случайного числа в заданном диапазоне.

return distribution(generator);

}

// Просмотр всех элементов в хеш-таблице.

void Object::Scan() {

// Перебор всех цепочек в хеш-таблице и вывод их содержимого.

for (int i = 0; i < size; i++) {

Hash[i].scan();

std::cout << '\n';

}

}

}

Lists.cpp

#include "Lists.h"

#include <iostream>

// Определение пространства имен listx, содержащего реализацию операций над двусвязным списком.

namespace listx {

// Получение первого элемента списка.

Element\* Object::getFirst() {

return head;

}

// Получение последнего элемента списка.

Element\* Object::getLast() {

// Инициализация указателя на первый элемент списка.

Element\* e = this->getFirst();

// Инициализация указателя на последний найденный элемент.

Element\* rc = this->getFirst();

// Перебор элементов списка до конца.

while (e != NULL) {

rc = e; // Обновление указателя на последний найденный элемент.

e = e->getNext(); // Переход к следующему элементу.

};

return rc; // Возврат указателя на последний элемент.

}

// Поиск элемента в списке по данным.

Element\* Object::search(void\* data) {

// Инициализация указателя на первый элемент списка.

Element\* rc = head;

// Перебор элементов списка до нахождения элемента с заданными данными или до конца списка.

while ((rc != NULL) && (rc->data != data)) {

rc = rc->next; // Переход к следующему элементу.

}

return rc; // Возврат найденного элемента или NULL, если элемент не найден.

}

// Вставка нового элемента в начало списка.

bool Object::insert(void\* data) {

// Инициализация результата вставки.

bool rc = NULL;

// Если список пуст, создание нового элемента и установка его как головы списка.

if (head == NULL)

head = new Element(NULL, data, head);

else

// В противном случае, создание нового элемента и установка его как нового головы списка.

head = (head->prev = new Element(NULL, data, head));

return rc; // Возврат результата вставки.

}

// Удаление элемента из списка по указателю на элемент.

bool Object::deleteByElement(Element\* e) {

// Инициализация результата удаления.

bool rc = NULL;

// Проверка, что элемент для удаления не NULL.

if (rc = (e != NULL)) {

// Если следующий элемент существует, обновление его предыдущего указателя.

if (e->next != NULL)

e->next->prev = e->prev;

// Если предыдущий элемент существует, обновление его следующего указателя.

if (e->prev != NULL)

e->prev->next = e->next;

else

// Если удаляемый элемент является головой списка, обновление головы списка.

head = e->next;

// Освобождение памяти, занимаемой удаляемым элементом.

delete e;

}

// Вывод сообщения о удалении элемента.

std::cout << "Элемент удален" << std::endl;

return rc; // Возврат результата удаления.

}

// Удаление элемента из списка по данным.

bool Object::deleteByData(void\* data) {

// Удаление элемента, найденного по данным.

return deleteByElement(search(data));

}

// Просмотр всех элементов списка.

void Object::scan() {

// Инициализация указателя на первый элемент списка.

Element\* e = this->getFirst();

// Перебор элементов списка и вывод их данных.

while (e != NULL) {

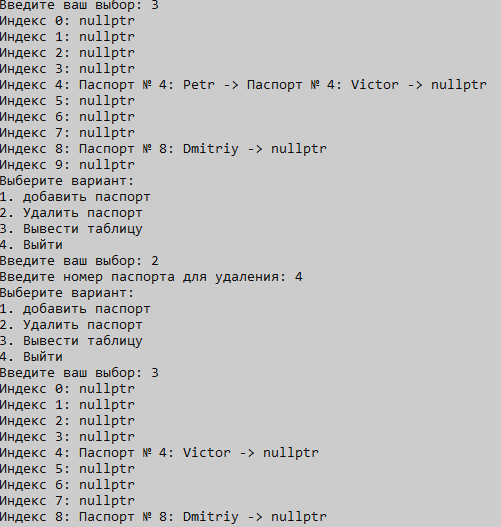
std::cout << ((int)e->data) << " "; // Вывод данных элемента.

e = e->getNext(); // Переход к следующему элементу.

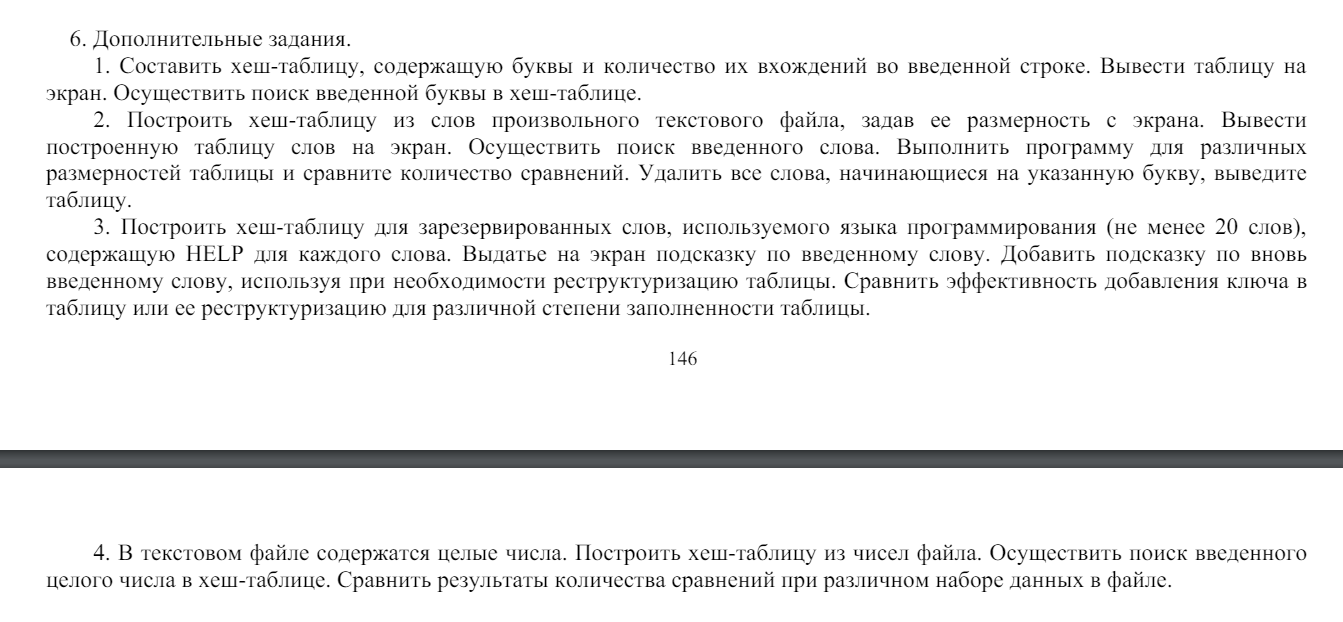
}

}

}



Дополнительные варианты



1.

#include <iostream>

#include <unordered\_map>

#include <string>

#include <cctype> // Для функции std::tolower

using namespace std;

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

unordered\_map<char, int> createHashTable(const string& str) {

unordered\_map<char, int> hashTable;

for (char ch : str) {

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

char key = tolower(ch); // Приводим букву к нижнему регистру

hashTable[key]++;

}

}

return hashTable;

}

int main() {

setlocale(LC\_ALL, "RUS");

string inputString;

cout << "Введите строку: ";

getline(cin, inputString);

// Создаем хеш-таблицу

unordered\_map<char, int> hashTable = createHashTable(inputString);

// Выводим содержимое хеш-таблицы на экран

cout << "Хеш-таблица:\n";

for (const auto& pair : hashTable) {

cout << "'" << pair.first << "': " << pair.second << '\n';

}

// Поиск символа в хеш-таблице

char searchChar;

cout << "Введите символ для поиска: ";

cin >> searchChar;

// Приводим символ к нижнему регистру, так как ключи в хеш-таблице тоже в нижнем регистре

searchChar = tolower(searchChar);

// Проверяем наличие символа в хеш-таблице

if (hashTable.find(searchChar) != hashTable.end()) {

cout << "Символ '" << searchChar << "' найден в строке " << hashTable[searchChar] << " раз\n";

}

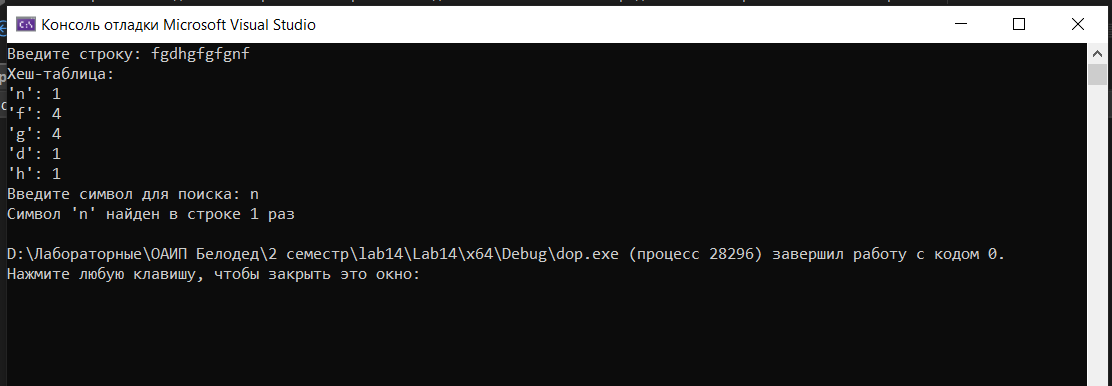
else {

cout << "Символ '" << searchChar << "' не найден в строке\n";

}

return 0;

}



2.

#include <iostream>

#include <fstream>

#include <unordered\_map>

#include <string>

#include <vector>

#include <algorithm>

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

std::unordered\_map<std::string, int> createHashTable(const std::string& filename, size\_t tableSize) {

std::ifstream file(filename);

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

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

return {};

}

std::unordered\_map<std::string, int> hashTable(tableSize);

std::string word;

while (file >> word) {

// Убираем знаки препинания с конца слова

word.erase(std::remove\_if(word.begin(), word.end(), ispunct), word.end());

// Приводим слово к нижнему регистру

std::transform(word.begin(), word.end(), word.begin(), ::tolower);

hashTable[word]++;

}

return hashTable;

}

// Функция для вывода содержимого хеш-таблицы слов на экран

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

std::cout << "Хеш-таблица слов:\n";

for (const auto& pair : hashTable) {

std::cout << "'" << pair.first << "': " << pair.second << '\n';

}

}

// Функция для удаления слов, начинающихся на указанную букву, из хеш-таблицы

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

std::vector<std::string> wordsToRemove;

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

if (it->first.front() == letter) {

wordsToRemove.push\_back(it->first);

}

}

for (const auto& word : wordsToRemove) {

hashTable.erase(word);

}

}

int main() {

setlocale(LC\_ALL, "RUS");

std::string filename;

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

std::cin >> filename;

size\_t tableSize;

std::cout << "Введите размерность хеш-таблицы: ";

std::cin >> tableSize;

// Создаем хеш-таблицу из слов текстового файла

std::unordered\_map<std::string, int> hashTable = createHashTable(filename, tableSize);

// Выводим содержимое хеш-таблицы на экран

printHashTable(hashTable);

// Поиск слова в хеш-таблице

std::string searchWord;

std::cout << "Введите слово для поиска: ";

std::cin >> searchWord;

if (hashTable.find(searchWord) != hashTable.end()) {

std::cout << "Слово '" << searchWord << "' найдено в тексте " << hashTable[searchWord] << " раз\n";

}

else {

std::cout << "Слово '" << searchWord << "' не найдено в тексте\n";

}

// Удаление слов, начинающихся на указанную букву, из хеш-таблицы

char letterToRemove;

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

std::cin >> letterToRemove;

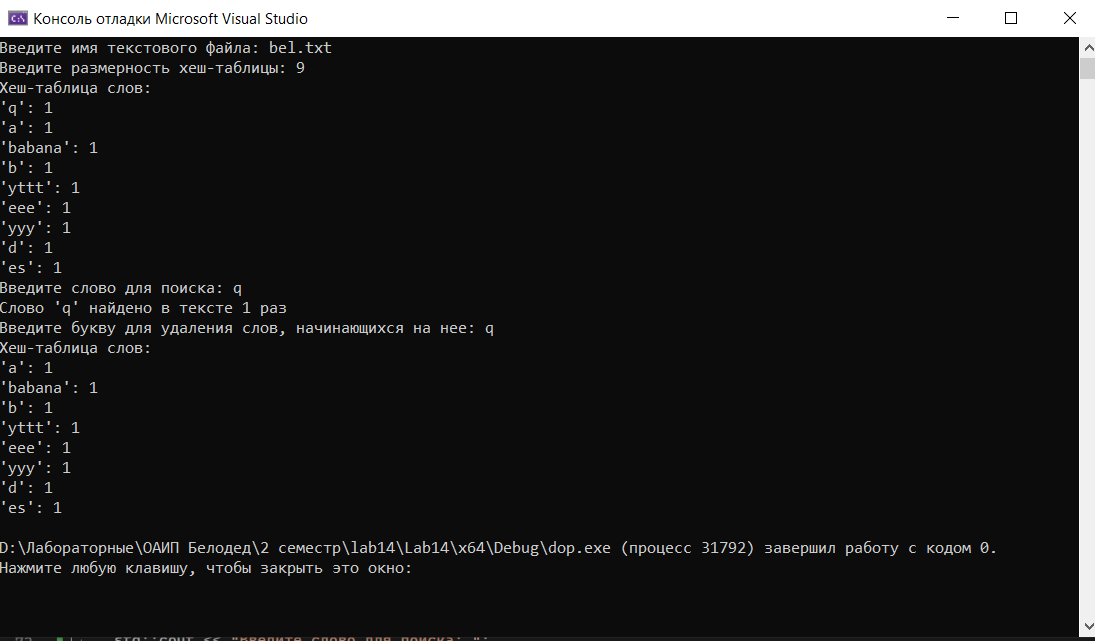
removeWordsStartingWithLetter(hashTable, letterToRemove);

// Выводим обновленную хеш-таблицу на экран

printHashTable(hashTable);

return 0;

}



4.

#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;

}

