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 10 группы

Жамойдо Артём Игоревич

Преподаватель: асс. Андронова М.В.

Минск, 2024

В соответствии со своим вариантом разработать проект для условия из таблицы, представленной ниже. Построить ***хеш-таблицы с цепочками*** разного размера с коллизиями.

Для вариантов с 1 по 8 вычисление хеш-функции произвести по методу универсального хеширования. При вычислении хеш-функции для вариантов с 9 по 16 использовать алгоритм на основе исключающего ИЛИ для поля строки данных.

Исследовать время поиска информации.

**Вариант 4**

**Библиотека**. Создать хеш-таблицу со следующими полями: номер книги, название книги. Ключ – номер книги. Номер книги – случайное число.

#include <iostream>

#include <string>

#include <vector>

#include <random>

using namespace std;

// Структура для представления книги

struct Book

{

int bookNumber; // Номер книги

string bookName; // Название книги

};

// Класс хеш-таблицы с цепочками по методу универсального хеширования

class HashTable

{

private:

int tableSize; // Размер таблицы

vector<Book\*> table; // Вектор указателей на книги

vector<int> a, b; // Коэффициенты для метода универсального хеширования

public:

// Конструктор

HashTable(int size) : tableSize(size)

{

// Инициализация векторов коэффициентов случайными значениями

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<int> dis(1, size - 1);

for (int i = 0; i < 2; i++)

{

a.push\_back(dis(gen));

b.push\_back(dis(gen));

}

// Инициализация таблицы

table.resize(tableSize, nullptr);

}

// Метод для вычисления хеша

int hash(int key)

{

return ((a[0] \* key + b[0]) % tableSize + tableSize) % tableSize;

}

// Метод для добавления книги в хеш-таблицу

void addBook(const Book& book)

{

int key = book.bookNumber;

int index = hash(key);

if (table[index] == nullptr)

{

table[index] = new Book(book);

}

else

{

// Предотвращение коллизий

cout << "Конфликт хешей. Книга с номером " << key << " не может быть добавлена." << endl;

}

}

// Метод для поиска книги по номеру

Book\* findBook(int key)

{

int index = hash(key);

if (table[index] != nullptr && table[index]->bookNumber == key)

{

return table[index];

}

return nullptr;

}

// Метод для удаления книги по номеру

void removeBook(int key)

{

int index = hash(key);

if (table[index] != nullptr && table[index]->bookNumber == key)

{

delete table[index];

table[index] = nullptr;

cout << "Книга с номером " << key << " удалена." << endl;

}

else

{

cout << "Книга с номером " << key << " не найдена." << endl;

}

}

// Метод для вывода всех книг в таблице

void printBooks()

{

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

{

if (table[i] != nullptr)

{

cout << "Строка " << i << ": Номер: " << table[i]->bookNumber << ", Название: " << table[i]->bookName << endl;

}

}

}

};

int main()

{

setlocale(0, "ru");

int tableSize = 100; // Размер таблицы

HashTable library(tableSize);

int choice;

do

{

cout << "Меню:" << endl;

cout << "1. Вывести все книги" << endl;

cout << "2. Добавить книгу" << endl;

cout << "3. Найти книгу" << endl;

cout << "4. Удалить книгу" << endl;

cout << "0. Выйти" << endl;

cout << "Выберите пункт меню: ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << "Книги в библиотеке:" << endl;

library.printBooks();

break;

}

case 2:

{

int bookNumber;

string bookName;

cout << "Введите номер книги: ";

cin >> bookNumber;

cin.ignore(); // Очистка буфера ввода перед чтением строки

cout << "Введите название книги: ";

getline(cin, bookName);

Book book;

book.bookNumber = bookNumber;

book.bookName = bookName;

library.addBook(book);

break;

}

case 3:

{

int bookNumber;

cout << "Введите номер книги: ";

cin >> bookNumber;

Book\* foundBook = library.findBook(bookNumber);

if (foundBook != nullptr)

{

cout << "Найдена книга: Номер: " << foundBook->bookNumber << ", Название: " << foundBook->bookName << endl;

}

else

{

cout << "Книга с номером " << bookNumber << " не найдена." << endl;

}

break;

}

case 4:

{

int bookNumber;

cout << "Введите номер книги: ";

cin >> bookNumber;

library.removeBook(bookNumber);

break;

}

case 0:

cout << "Программа завершена." << endl;

break;

default:

cout << "Неверный выбор. Пожалуйста, выберите пункт меню снова." << endl;

break;

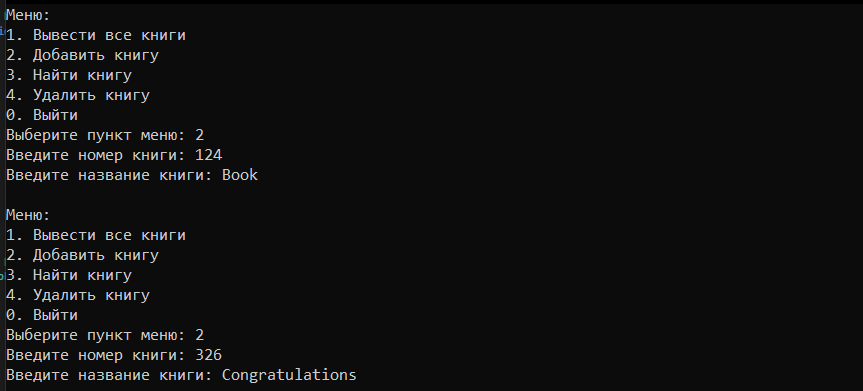
}

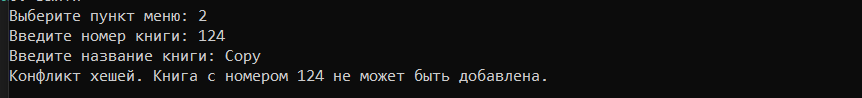
cout << endl;

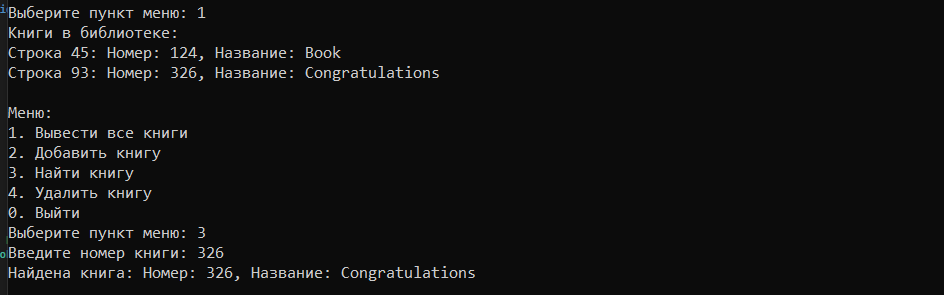
} while (choice != 0);

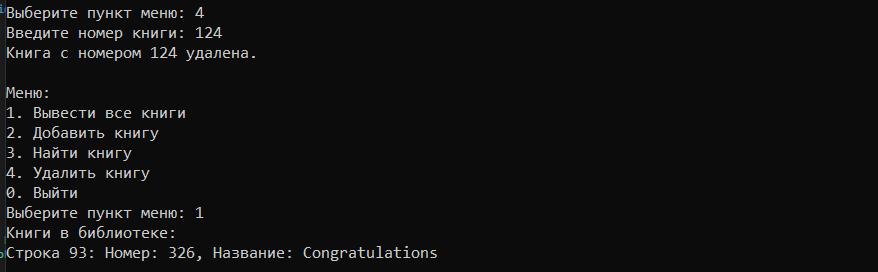
return 0;

}

****

****

****

****

Доп. задачи

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 <string>

using namespace std;

// Структура таблицы

struct LetterCount

{

char letter;

int count;

};

const int TABLE\_SIZE = 26;

// Класс для хеш-таблицы

class HashTable

{

public:

HashTable(); // Конструктор

// Прототипы методов

void insert(char letter);

bool search(char letter);

void print();

private:

LetterCount m\_data[TABLE\_SIZE]; // Массив с данными таблицы

int hash(char letter); // Прототип хеш-функции

};

// Конструктор

HashTable::HashTable()

{

// Заполняем таблицу алфавитом

for (int i = 0; i < TABLE\_SIZE; i++)

{

m\_data[i].letter = 'a' + i;

m\_data[i].count = 0;

}

}

// Функция для вычисления хеша

int HashTable::hash(char letter)

{

return (letter - 'a') % TABLE\_SIZE;

}

// Добавление элемента к хеш-таблицу

void HashTable::insert(char letter)

{

// Получаем хеш и записываем в index

int index = hash(letter);

// Поиск элемента

while (m\_data[index].count > 0 && m\_data[index].letter != letter)

{

index = (index + 1) % TABLE\_SIZE;

}

// Записываем данные и увеличиваем количество

m\_data[index].letter = letter;

m\_data[index].count++;

}

// Метод поиска элемента

bool HashTable::search(char letter)

{

// Получение хеша

int index = hash(letter);

// Перебираем существующие буквы

while (m\_data[index].count > 0)

{

// Сравниваем букву из таблицы с введенной

if (m\_data[index].letter == letter)

{

return true;

}

// Смещаем индекс

index = (index + 1) % TABLE\_SIZE;

}

return false;

}

// Метод вывода таблицы

void HashTable::print()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

cout << m\_data[i].letter << ": " << m\_data[i].count << endl;

}

}

int main()

{

setlocale(0, "ru");

string str;

cout << "Введите строку" << endl;

getline(cin, str);

HashTable table;

for (char c : str)

{

table.insert(c);

}

table.print(); // Вывод таблицы

char targetLetter;

cout << "Введите букву ";

cin >> targetLetter;

if (table.search(targetLetter))

{

cout << "Буква " << targetLetter << " найдена" << endl;

}

else

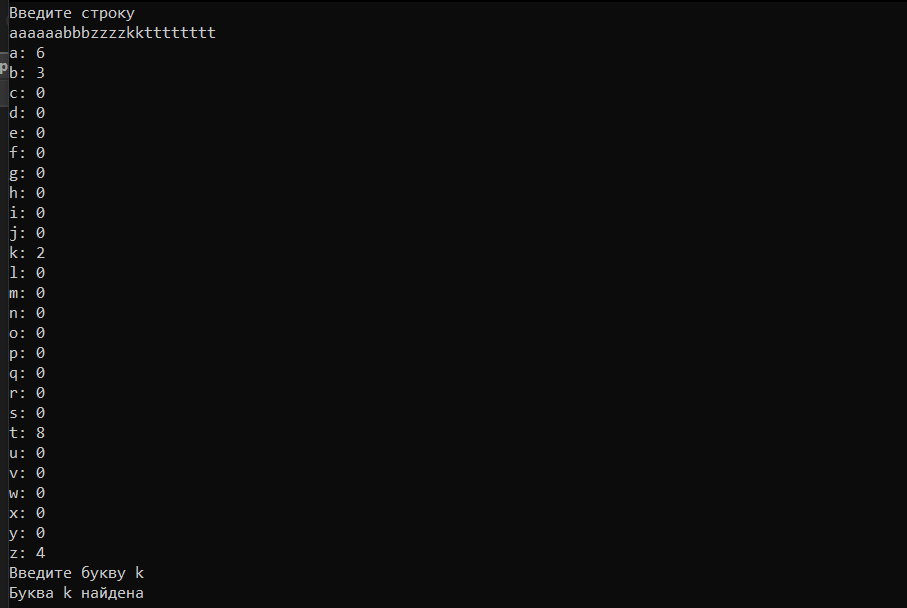
{

cout << "Буква " << targetLetter << " не найдена" << endl;

}

return 0;

}



2. 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 <string>

using namespace std;

struct HashNode

{

string key;

HashNode\* next;

HashNode(string k) : key(k), next(nullptr) {}

};

// Класс хеш-таблицы

class HashTable

{

public:

HashTable(const int&); // Конструктор

~HashTable(); // Диструктор

// Прототипы методов

void insert(string key);

void remove(char s);

void print();

bool search(string key);

private:

HashNode\*\* m\_data; // Массив с даными таблицы

int TABLE\_SIZE;

// Метод хеширования

int hash(string key)

{

int sum = 0;

for (char c : key)

{

sum ^= c;

}

return sum % TABLE\_SIZE;

}

};

// Конструктор

HashTable::HashTable(const int& size)

{

TABLE\_SIZE = size;

m\_data = new HashNode \* [TABLE\_SIZE];

// Заполнение таблицы пустыми указателями

for (int i = 0; i < TABLE\_SIZE; i++)

{

m\_data[i] = nullptr;

}

}

// Диструктор

HashTable::~HashTable()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

HashNode\* node = m\_data[i];

while (node != nullptr)

{

HashNode\* prev = node;

node = node->next;

delete prev;

}

}

delete[] m\_data;

}

// Метод вывода таблицы

void HashTable::print()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

if (m\_data[i] != nullptr)

{

cout << m\_data[i]->key << endl;

}

}

}

// Метод добавления элемента

void HashTable::insert(string key)

{

// Получение хеша по ключу

int index = hash(key);

// Создание нового элемента

HashNode\* node = new HashNode(key);

// Проверка на наличие элементов с полученным хешем

if (m\_data[index] != nullptr)

{

// Запись следующего значения в таблицу

node->next = m\_data[index];

}

m\_data[index] = node;

}

// Функция поиска элемента

bool HashTable::search(string key)

{

int index = hash(key);

// Временная переменная

HashNode\* node = m\_data[index];

// Смещение до последнего элемента

while (node != nullptr)

{

// Проверка на совпадение

if (node->key == key)

{

return true;

}

node = node->next;

}

return false;

}

// Метод удаления элемента

void HashTable::remove(char s)

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

if (m\_data[i] != nullptr)

{

if (m\_data[i]->key[0] == s)

{

delete m\_data[i];

m\_data[i] = nullptr;

}

}

}

}

// Функция чтения слов из файла

void readStr(const string& filename, HashTable& table)

{

// Открытие файла

ifstream file(filename);

if (!file)

{

cerr << "Ошибка открытия файла " << filename << endl;

return;

}

string str;

// Считывание слов из файла

while (file >> str)

{

// Заполнение таблицы

table.insert(str);

}

file.close();

}

int main()

{

setlocale(0, "ru");

string path = "file.txt";

int size;

cout << "Введите размер таблицы: ";

cin >> size;

HashTable table(size);

readStr(path, table);

table.print(); // Вывод таблицы

string str;

cout << "Введите слово: ";

cin >> str;

if (table.search(str))

{

cout << "Элемент " << str << " найден" << endl;

}

else

{

cout << "Элемент " << str << " не найден" << endl;

}

char symbol;

cout << "Введите символ: ";

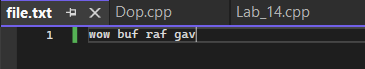
cin >> symbol;

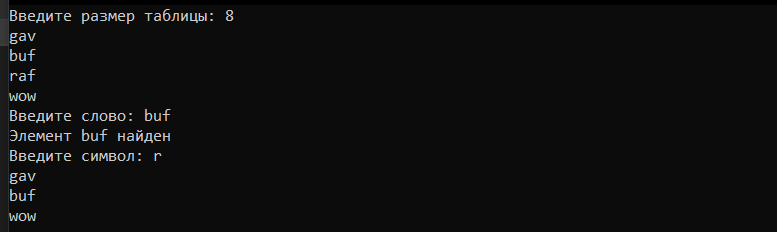
table.remove(symbol); // Удаление слова

table.print(); // Вывод таблицы

return 0;

}





4. 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 <string>

using namespace std;

const int TABLE\_SIZE = 100; // Размер таблицы

struct HashNode

{

int key;

HashNode\* next;

HashNode(int k) : key(k), next(nullptr) {}

};

// Класс хеш-таблицы

class HashTable

{

public:

HashTable(); // Конструктор

~HashTable(); // Диструктор

// Прототипы методов

void insert(int key);

void print();

bool search(int key);

private:

HashNode\*\* m\_data; // Массив хеш-таблицы

// функция хеширования

int hash(int key)

{

return key % TABLE\_SIZE;

}

};

// Конструктор

HashTable::HashTable()

{

m\_data = new HashNode \* [TABLE\_SIZE];

// Заполнение всех элементов таблицы пустыми ссылками

for (int i = 0; i < TABLE\_SIZE; i++)

{

m\_data[i] = nullptr;

}

}

// Диструктор

HashTable::~HashTable()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

HashNode\* node = m\_data[i];

while (node != nullptr)

{

HashNode\* prev = node;

node = node->next;

delete prev;

}

}

delete[] m\_data;

}

// Метод вывода таблицы

void HashTable::print()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

if (m\_data[i] != nullptr)

{

cout << m\_data[i]->key << endl;

}

}

}

// Метод добавления элементов

void HashTable::insert(int key)

{

int index = hash(key);

// Создание нового элемента по ключу

HashNode\* node = new HashNode(key);

if (m\_data[index] != nullptr)

{

node->next = m\_data[index];

}

m\_data[index] = node;

}

// Метод поиска

bool HashTable::search(int key)

{

int index = hash(key);

// Создание временной переменнной

HashNode\* node = m\_data[index];

// Проверка всех элементов списка

while (node != nullptr)

{

if (node->key == key)

{

return true;

}

}

return false;

}

// Фукнция чтения чисел из файла

void readNumbers(const string& filename, HashTable& table)

{

ifstream file(filename); // Открытие файла

if (!file)

{

cerr << "Ошибка открытия файла " << filename << endl;

return;

}

int number;

while (file >> number)

{

table.insert(number);

}

file.close(); // Закрытие файла

}

int main()

{

setlocale(0, "ru");

string path = "file.txt";

HashTable table;

readNumbers(path, table);

table.print(); // Вывод таблицы

int num;

cout << "Введите число: ";

cin >> num;

if (table.search(num))

{

cout << "Элемент " << num << " найден" << endl;

}

else

{

cout << "Элемент " << num << " не найден" << endl;

}

return 0;

}

