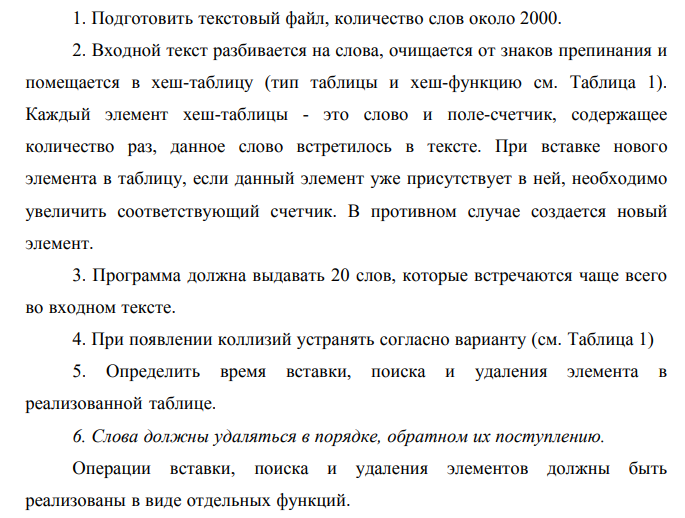
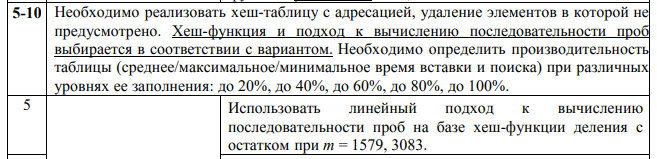
**Лабораторная робота №6**

Структуры данных. Хеш-таблицы

**Мета роботы**: Научиться создавать алгоритм быстрого поиска элементов множества путем создания оптимальной хеш-функции. Анализ частоты встречаемости слов во входном тексте с применением хеш-таблиц.

*Задание*





**Ход работы**

1. **Текст программы**

#include <iostream>

#include <fstream>

#include <algorithm>

#include <vector>

#include <string>

#include <regex>

#include <sstream>

#include "Map.hpp"

std::vector<std::string> readWordsFromFile(const std::string url, int count);

dg::Map getMapOfWords(std::vector<std::string> &words);

std::vector<std::pair<std::string, int>> getWordsFrequency(std::vector<std::string>& words);

void printTopOfWords(std::vector<std::string> &words, int wordsInTop = 20);

void printHello();

void run();

template <typename T>

T prompt(const char label[]);

int main() {

printHello();

run();

system("pause");

return 0;

}

void run() {

auto words = readWordsFromFile("loremText2000.txt", 2000);

const int wordsInTop = prompt<unsigned short>("Enter the number of words in the TOP: ");

std::cout << "The most common words in the text: \n";

printTopOfWords(words, wordsInTop);

}

void printTopOfWords(std::vector<std::string>& words, int wordsInTop) {

const auto frequency = getWordsFrequency(words);

const auto topSize = std::min(wordsInTop, static\_cast<int>(frequency.size()));

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

std::cout << " - " << frequency[i].first << ": " << frequency[i].second << std::endl;

}

std::cout << std::endl;

}

std::vector<std::pair<std::string, int>> getWordsFrequency(std::vector<std::string> &words) {

auto mapOfWords = getMapOfWords(words);

auto frequency = mapOfWords.toVector();

static auto comparator = [](

const std::pair<std::string, int> &a,

const std::pair<std::string, int> b

) {

return a.second > b.second;

};

std::sort(frequency.begin(), frequency.end(), comparator);

return frequency;

}

dg::Map getMapOfWords(std::vector<std::string>& words) {

dg::Map mapOfWords;

for (auto& word : words) {

mapOfWords.set(word, mapOfWords.get(word) + 1);

}

return mapOfWords;

}

std::vector<std::string> readWordsFromFile(const std::string url, int count) {

static const std::regex wordRegex("[.,;-\?!]");

std::vector<std::string> words(count);

std::ifstream fin(url);

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

throw std::runtime\_error("Can't open the file");

}

std::string fileText(

(std::istreambuf\_iterator<char>(fin)),

std::istreambuf\_iterator<char>()

);

fileText = std::regex\_replace(fileText, wordRegex, "");

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

std::istringstream textStream(fileText);

std::string next;

for (auto& word : words) {

if (!(textStream >> next)) break;

word = next;

}

fin.close();

return words;

}

void printHello() {

std::cout << "\* \* \* Algorithms: Hash Tables \* \* \*\n\n";

}

template <typename T>

T prompt(const char label[]) {

std::cout << label;

while (true) {

T val;

std::cin >> val;

if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Wrong. Try again: ";

}

else {

std::cin.ignore(32767, '\n');

return val;

}

}

}

***Map.hpp***

#pragma once

#include <string>

#include <vector>

namespace dg {

class Map {

public:

typedef std::pair<std::string, int> Slot;

Map();

Map(int capacity);

void set(std::string &key, int val);

int get(std::string &key);

std::vector<Slot> toVector();

~Map();

private:

int m\_capacity;

std::vector<Slot> m\_slots;

int hash(std::string &key, int i) const;

static void checkKey(std::string &key);

};

};

***Map.cpp***

#include "Map.hpp"

#include <stdexcept>

namespace dg {

Map::Map() :

Map(1579)

{

}

Map::Map(int capacity) :

m\_capacity(capacity),

m\_slots(m\_capacity)

{

}

void Map::set(std::string &key, int val) {

checkKey(key);

for (int i = 0; i < m\_capacity; i++) {

auto& item = m\_slots[hash(key, i)];

if (item.first == key || item.first.empty()) {

item.first = key;

item.second = val;

return;

}

}

throw std::runtime\_error("Map is full");

}

int Map::get(std::string &key) {

checkKey(key);

for (int i = 0; i < m\_capacity; i++) {

const auto& item = m\_slots[hash(key, i)];

if (item.first == key || item.first.empty()) {

return item.second;

}

}

return 0;

}

int Map::hash(std::string& key, int i) const {

static int d = 256;

int hash = 0;

for (auto c : key) {

hash += c \* d;

}

hash = hash % m\_capacity;

return (hash + i) % m\_capacity;

}

std::vector<Map::Slot> Map::toVector() {

std::vector<Slot> vec;

for (auto &slot : m\_slots) {

if (slot.first.empty()) continue;

vec.push\_back(slot);

}

return vec;

}

void Map::checkKey(std::string& key) {

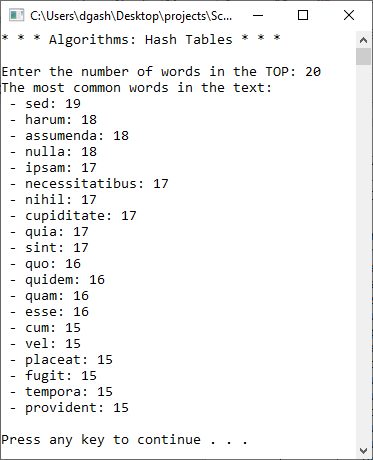
if (key.empty()) throw std::runtime\_error("The key can't be empty");

}

Map::~Map() = default;

}

1. **Результат выполнения программы**



**Итог:** на этой лабораторной работе я научился создавать алгоритм быстрого поиска элементов множества путем создания оптимальной хеш-функции; проанализировал частоты встречаемости слов во входном тексте с применением хеш-таблиц.