|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Практическое задание №7 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **кафедра теоретической и прикладной информатики** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-03 |
| Бригада: | Место для ввода текста. |
| Студенты: | Сидоров Даниил, |
|  | Малыгин Сергей |
|  |  |
| Преподаватели: | Щукин Георгий Анатольевич |
|  |  |
| Новосибирск | | |
| 2021 | | |
|  | | |

**1.Результаты замеров:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | Dictionary | | | | | | Unordered\_map | |
| hash | | hash\_1 | | hash\_2 | | Вставка | Поиск |
| Вставка | Поиск | Вставка | Поиск | Вставка | Поиск |
| 1000 | 0.0002622 | 0.0001282 | 0.0002029 | 9.92e-05 | 0.000404 | 0.0001952 | 0.0001642 | 0 |
| 10 000 | 0.0020994 | 0.0009905 | 0.0047182 | 0.0028023 | 0.0032802 | 0.0023154 | 0.0837461 | 1e-07 |
| 100 000 | 0.327338 | 0.25224 | 2.04237 | 1.52039 | 0.0793462 | 0.0621752 | 0.0329948 | 0 |
| 500 000 | 12.9349 | 11.9589 | 68.4912 | 68.1199 | 0.505482 | 0.461987 | 0.191883 | 1e-07 |

**2.Программа:**

**dictionary.h**

**#pragma once**

**#include <string>**

**#include <functional>**

**#include <vector>**

**#include <list>**

**using HashFunction = std::function<int(const std::string&)>;**

**/\***

**\* Dictionary that maps strings to other strings.**

**\*/**

**class Dictionary**

**{**

**public:**

**Dictionary(int num\_of\_buckets = 1000, HashFunction hash = std::hash<std::string> {});**

**~Dictionary();**

**/// Set/replace value for the key.**

**void set(const std::string &key, const std::string &value);**

**/// Get value for the key or empty string if there is no such key.**

**std::string get(const std::string &key) const;**

**/// Get number of items (keys) in the dictionary.**

**int size() const;**

**private:**

**using KeyValue = std::pair<std::string, std::string>;**

**std::vector<std::list<KeyValue>> table;**

**HashFunction hash\_function;**

**// Add other private methods and fields here.**

**};**

**dictionary.cpp**

**#include "dictionary.h"**

**#include <iostream>**

**Dictionary::Dictionary(int num\_of\_buckets, HashFunction hash) :**

**table(num\_of\_buckets), hash\_function(hash)**

**{**

**}**

**Dictionary::~Dictionary()**

**{**

**}**

**void Dictionary::set(const std::string& key, const std::string& value)**

**{**

**// Insert key-value pair into the dictionary.**

**// If such key already exists, replace old value with the new one.**

**int KEY = hash\_function(key) % table.size();**

**if (!table[KEY].empty())**

**{**

**for (auto& PAIR : table[KEY])**

**{**

**if (PAIR.first == key)**

**{**

**PAIR.second = value;**

**return;**

**}**

**}**

**table[KEY].push\_back(KeyValue(key, value));**

**return;**

**}**

**else**

**{**

**std::list<KeyValue> LIST;**

**LIST.push\_back(KeyValue(key, value));**

**table[KEY] = LIST;**

**return;**

**}**

**}**

**std::string Dictionary::get(const std::string &key) const**

**{**

**// Return value for the key.**

**// Return empty string if there is no such key.**

**int KEY = hash\_function(key) % table.size();**

**if (!table[KEY].empty())**

**{**

**for (auto& PAIR : table[KEY])**

**if (PAIR.first == key)**

**return PAIR.second;**

**}**

**return "";**

**}**

**int Dictionary::size() const**

**{**

**// Return size of the dictionary, i.e. number of key-value pairs.**

**int size = 0;**

**for (const auto &bucket: table)**

**{**

**size += bucket.size();**

**}**

**return size;**

**}**

**main.cpp**

**#include "dictionary.h"**

**#include "random\_string.h"**

**#include <unordered\_map>**

**#include <iostream>**

**#include <chrono>**

**using namespace std;**

**#define CATCH\_CONFIG\_RUNNER**

**#include "catch.hpp"**

**const int MAX = 10000000;**

**const int MIN = 10000;**

**int hash\_1(std::string str)**

**{**

**int result(0);**

**for (int i(0); i < str.length(); i++)**

**result += int(str[i]);**

**return result;**

**}**

**int hash\_2(std::string str)**

**{**

**int result(0);**

**const int a = 37;**

**for (int i(0); i < str.length(); i++)**

**result += int(str[i] \* pow(a, i));**

**return result;**

**}**

**int main(int argc, char \*\*argv)**

**{**

**int result = Catch::Session().run(argc, argv);;**

**for (int i(MIN); i <= MAX; i \*= 10)**

**{**

**Dictionary D;**

**Dictionary D1(1000, hash\_1);**

**Dictionary D2(1000, hash\_2);**

**unordered\_map<string, string> umap;**

**vector<string> KEY(i);**

**vector<string> VALUE(i);**

**for (int k(0); k < i; k++)**

**{**

**KEY[k] = random\_string(5);**

**VALUE[k] = random\_string(10);**

**}**

**auto t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D.set(KEY[j], VALUE[j]); ///////D**

**auto t2 = std::chrono::high\_resolution\_clock::now();**

**auto seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D1.set(KEY[j], VALUE[j]); ///////D1**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D1 (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D2.set(KEY[j], VALUE[j]); ///////D2**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D2 (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**umap[KEY[j]] = VALUE[j]; ///////unordered\_map**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "unordered\_map (i=" << i << ") ::" << seconds << endl;**

**cout << endl << "//////////////FIND////////////////" << endl << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D.get(KEY[j]); ///////D**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D1.get(KEY[j]); ///////D1**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D1 (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**D2.get(KEY[j]); ///////D2**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "D2 (i=" << i << ") ::" << seconds << endl;**

**t1 = std::chrono::high\_resolution\_clock::now();**

**for (int j(0); j < i; j++)**

**umap.find(KEY[j]); ///////unordered\_map**

**t2 = std::chrono::high\_resolution\_clock::now();**

**seconds = std::chrono::duration<double>(t2 - t1).count();**

**cout << "unordered\_map (i=" << i << ") ::" << seconds << endl;**

**cout << endl << "-----------+++-----------" << endl;**

**}**

**return result;**

**}**