Лабораторная работа №3
«ХЭШИРОВАНИЕ»

по дисциплине "Методы программирования"

Выполнил Ковалев Даниил
СКБ171, вариант 12
МИЭМ НИУ ВШЭ

В данной лабораторной проведено сравнение следующих алгоритмов хэширования строк:

- Банальный алгоритм (dummy)
- Алгоритм rot13
- Алгоритм elf
- Алгоритм rot19 — модифицированная версия rot13 для работы с 64-битными хэшами
- Алгоритм std::hash

Алгоритмы применяются для хэширования поля trainer в массиве элементов Entry — данных о футбольных командах, принимающих участие в чемпионате страны по футболу: страна, название клуба, город, год, ФИО главного тренера (сравнение по полям — год, страна количество набранных очков (по убыванию), название клуба).

Считывание данных в программе происходит либо из csv-файла с заголовком country;city;club;trainer;year;score, либо из базы данных SQLite с таблицей entries и полями country (TEXT), city (TEXT), club (TEXT), trainer (TEXT), year (INTEGER), score (INTEGER).

Схема запуска: запускать нужно python run.py, он запустит генерацию массива из 1000000 элементов типа Entry и положит их базу данных data.sqlite:

```
../generate_data/generate_data --size=1000000 --output=data.sqlite
```

После этого необходимо запустить тестирование алгоритмов хэширования, взяв в качестве данных сгенерированный data.sqlite; результаты тестирования времени поиска в хэш-таблице положить в results_time.csv, результаты тестирования процента коллизий — в results_collision.csv. Размеры массивов, на которых тестируются хэши, берутся из sizes.txt, и являются равномерно распределенными на логарифмической оси от 10000 до 1000000.

```
./3_hashes --input=data.sqlite --sizes=sizes.txt --output_time=results_time
    .csv --output_collision=results_collision.csv 2> test_hash.log
```

После этого средствами пакета matplotlib для python на основе results_time.csv и results_collision.csv строятся два графика:

- зависимость времени поиска в хэш-таблице от размера массива данных (обе оси логарифмические)
- зависимость процента коллизий от размера массива данных (горизонтальная ось логарифмическая)

Реализация хэш-таблицы для разрешения коллизий использует принцип корзин: в одной корзине разрешается держать какое-то количество элементов с одинаковыми хэшами. Если размер корзины начинает превышать некоторый установленный, происходит перехэширование с увеличением количества корзин. Для плохих хэшей размер корзин способен увеличиваться довольно сильно, поэтому время поиска при их использовании больше, чем при использовании хороших хэшей.

Структура проекта:

- 🗁 **3rd_party** — см. ЛР1
- 🗁 **entry** — см. ЛР1
- 🗁 **generate_data** — см. ЛР1
- 🗁 **helpers** — см. ЛР1
- 🗁 **lab3**:
  - 🗎 **CMakeLists.txt**
  - 🗎 **dummy.h** — файл с описанием алгоритма хэширования **dummy**
  - 🗎 **dummy.cpp** — файл с реализацией алгоритма хэширования **dummy**
  - 🗎 **rot13.h** — файл с описанием алгоритмов хэширования **rot13** и **rot19**
  - 🗎 **rot13.cpp** — файл с реализацией алгоритмов хэширования **rot13** и **rot19**
  - 🗎 **elf.h** — файл с описанием алгоритма хэширования **elf**
  - 🗎 **elf.cpp** — файл с реализацией алгоритма хэширования **elf**
  - 🗎 **hash_table.h** — файл с реализацией хэш-таблицы
  - 🗎 **main.cpp** — файл с реализацией тестирования алгоритмов хэширования
  - 🗎 **run.py** — скрипт запуска генерации данных и тестирования алгоритмов хэширования
- 🗎 **CMakeLists.txt** — см. ЛР1

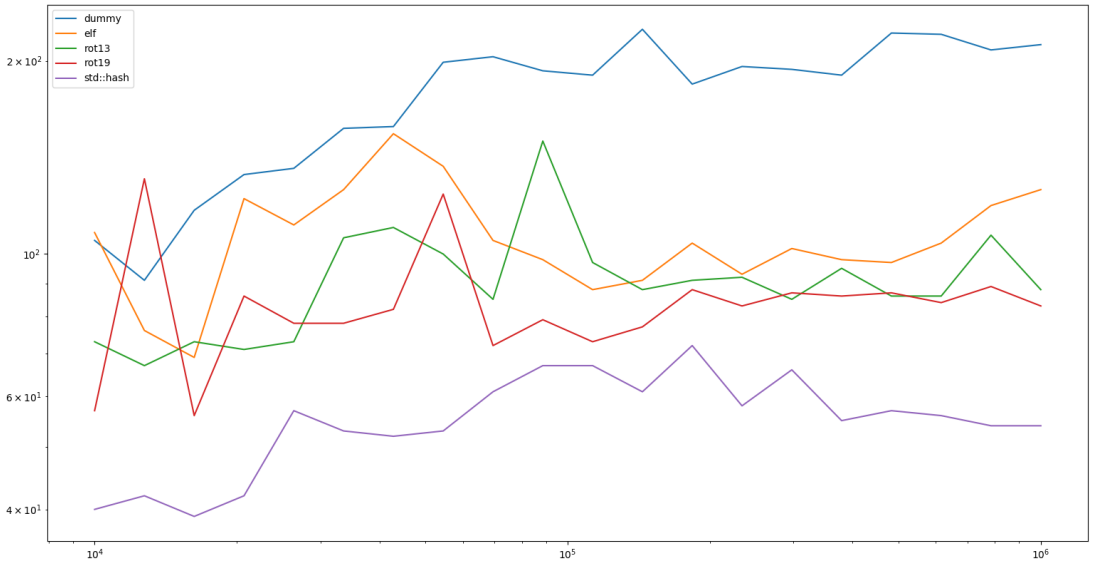
Приведем таблицу, построенную по файлу **results_time.csv**. В ней записано время поиска в хэш-таблице при использовании каждого из 5 тестируемых алгоритмов хэширования в наносекундах на диапазонах данных размерами от 10000 до 1000000.

	10000	12742	16237	20691	26366	33598	42813
dummy	105	91	117	133	136	157	158
rot13	73	67	73	71	73	106	110
elf	108	76	69	122	111	126	154
rot19	57	131	56	86	78	78	82
std::hash	40	42	39	42	57	53	52

54555	69519	88586	112883	143844	183298	233572
199	203	193	190	224	184	196
100	85	150	97	88	91	92
137	105	98	88	91	104	93
124	72	79	73	77	88	83
53	61	67	67	61	72	58

297635	379269	483293	615848	784759	1000000
194	190	221	220	208	212
85	95	86	86	107	88
102	98	97	104	119	126
87	86	87	84	89	83
66	55	57	56	54	54

График, построенный по этой таблице



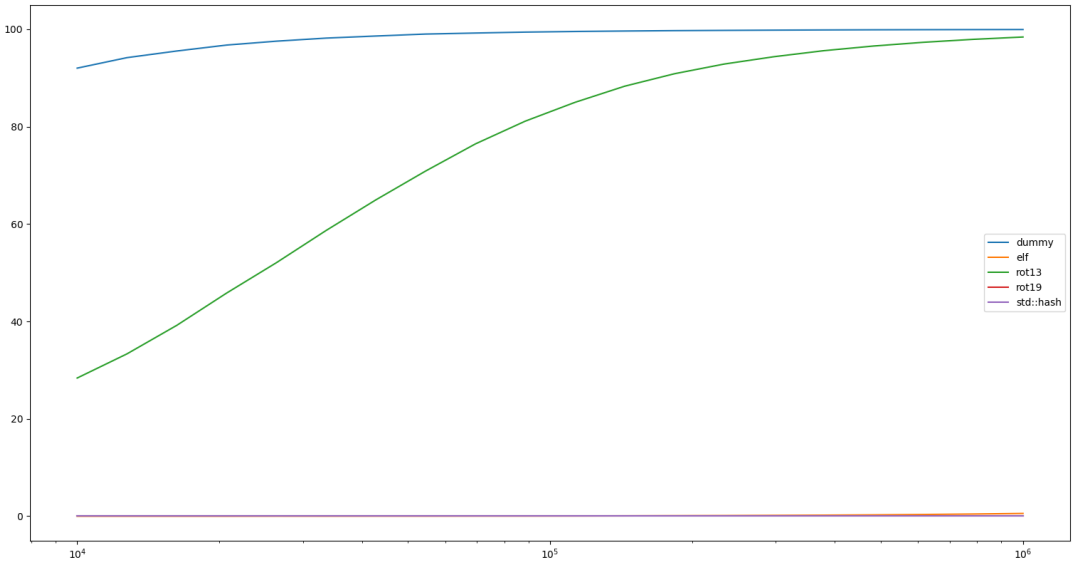
Приведем таблицу, построенную по файлу **results_collision.csv**. В ней записан процент коллизий для каждого из 5 тестируемых алгоритмов хэширования при хэшировании диапазонов данных размерами от 10000 до 1000000.

	10000	12742	16237	20691	26366	33598	42813
dummy	92.03	94.1846	95.5472	96.7667	97.565	98.1993	98.6242
rot13	28.38	33.3386	39.1944	45.817	52.0557	58.6821	64.9733
elf	0	0	0	0	0.0151711	0.0119055	0.0233574
rot19	0	0	0	0	0	0	0
std::hash	0	0	0	0	0	0	0

54555	69519	88586	112883	143844	183298	233572
99.0303	99.2391	99.4299	99.5579	99.6503	99.7321	99.7872
70.8954	76.4683	81.1358	85.0022	88.3242	90.8821	92.8746
0.0219962	0.0345229	0.0383808	0.0531524	0.0709102	0.0982007	0.123302
0	0	0	0	0	0	0
0	0	0	0	0	0	0

297635	379269	483293	615848	784759	1000000
99.8344	99.8729	99.9025	99.923	99.941	99.9569
94.3807	95.615	96.5735	97.3378	97.9428	98.4239
0.163959	0.208032	0.268574	0.350086	0.44434	0.572502
0	0	0	0	0	0
0	0	0	0	0	0

График, построенный по этой таблице



Из построенных по этим данным графиков можно сделать следующие выводы:

- Время поиска в хэш-таблице константное и не зависит от размера данных. При этом при использовании хороших хэшей поиск более быстрый, чем при использовании плохих, из-за меньшего размера корзин
- Процент коллизий у **std::hash** и **rot19** нулевой на количестве данных вплоть до 10^6, у **elf** он не превышает 1% при количестве данных 10^6, а у **rot13** и **dummy** он быстро стремится к 100% при увеличении количества данных до 10^6.

Листинги с исходным кодом всех файлов расположены на следующих страницах отчета.

Листинг 1: lab3/CMakeLists.txt

```
set(PROJECT_NAME 3_hashes)

project(${PROJECT_NAME} LANGUAGES CXX)

set(SOURCES main.cpp
            dummy.cpp
            elf.cpp
            rot13.cpp)
set(HEADERS dummy.h
            elf.h
            rot13.h
            hash_table.h)

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

target_include_directories(${PROJECT_NAME} PRIVATE ${Helpers_INCLUDE_DIR} ${
    Entry_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE entry)
target_link_libraries(${PROJECT_NAME} PRIVATE helpers)
target_link_libraries(${PROJECT_NAME} PRIVATE Boost::program_options)

file(COPY run.py DESTINATION ${PROJECT_BINARY_DIR})
```

Листинг 2: lab3/dummy.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий описание простейшей функции хэширования строк
 * @date Февраль 2020
*/
#ifndef DUMMY_H
#define DUMMY_H

#include <cstddef>
#include <string>

namespace my
{

/**
 * Считает простейшую хэш-функцию для строки
 * @param[in] str строка, для которой необходимо посчитать хэш
 * @return значение хэша для заданной строки
 */
std::size_t dummy_hash(const std::string& str);

class DummyHash
{
public:
    std::size_t operator()(const std::string& str) const
    {
        return dummy_hash(str);
    }
};

} // namespace my

#endif // DUMMY_H
```

Листинг 3: lab3/dummy.cpp

```
#include "dummy.h"

namespace my
{
```

```
5
6  std::size_t dummy_hash(const std::string& str)
7  {
8      std::size_t hash = 0;
9      for (char c : str)
10         hash += static_cast<std::uint8_t>(c);
11     return hash;
12 }
13
14 } // namespace my
```

Листинг 4: lab3/rot13.h

```
1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий описание функции хэширования строк rot13
4   * @date Февраль 2020
5  */
6  #ifndef ROT13_H
7  #define ROT13_H
8
9  #include <cstddef>
10 #include <string>
11
12 namespace my
13 {
14
15 /**
16  * Считает хэш-функцию rot13 для строки
17  * @param[in] str строка, для которой необходимо посчитать хэш
18  * @return значение хэша rot13 для заданной строки
19  */
20 std::size_t rot13_hash(const std::string& str);
21
22 /**
23  * Улучшенная версия rot13 для работы с 64-битными хэшами
24  */
25 std::size_t rot19_hash(const std::string& str);
26
27 class Rot13Hash
28 {
29 public:
30     std::size_t operator()(const std::string& str) const
31     {
32         return rot13_hash(str);
33     }
34 };
35
36 class Rot19Hash
37 {
38 public:
39     std::size_t operator()(const std::string& str) const
40     {
41         return rot19_hash(str);
42     }
43 };
44
45 } // namespace my
46
47 #endif // ROT13_H
```

Листинг 5: lab3/rot13.cpp

```
1  #include "rot13.h"
2
3  namespace my
4  {
```

```
5
6  std::size_t rot13_hash(const std::string& str)
7  {
8      std::size_t hash = 0;
9      for (char c : str)
10     {
11         hash += static_cast<std::uint8_t>(c);
12         hash -= (hash << 13) | (hash >> 19);
13     }
14     return hash;
15 }
16
17 std::size_t rot19_hash(const std::string& str)
18 {
19     std::size_t hash = 0;
20     for (char c : str)
21     {
22         hash += static_cast<std::uint8_t>(c);
23         hash -= (hash << 19) | (hash >> 43);
24     }
25     return hash;
26 }
27
28 } // namespace my
```

Листинг 6: lab3/elf.h

```
1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий реализацию функции хэширования строк ELF
4   * @date Февраль 2020
5  */
6  #ifndef ELF_H
7  #define ELF_H
8
9  #include <cstddef>
10 #include <string>
11
12 namespace my
13 {
14
15 /**
16  * Считает хэш-функцию elf для строки
17  * @param[in] str строка, для которой необходимо посчитать хэш
18  * @return значение хэша elf для заданной строки
19  */
20 std::size_t elf_hash(const std::string& str);
21
22 class ElfHash
23 {
24 public:
25     std::size_t operator()(const std::string& str) const
26     {
27         return elf_hash(str);
28     }
29 };
30
31 } // namespace my
32
33 #endif // ELF_H
```

Листинг 7: lab3/elf.cpp

```
1  #include "elf.h"
2
3  namespace my
4  {
```

```
5
6  std::size_t elf_hash(const std::string& str)
7  {
8      std::size_t h = 0, high;
9      for (char c : str)
10     {
11         h = (h << 4) + static_cast<std::size_t>(c++);
12         if ((high = (h & 0xF0000000)))
13             h ^= high >> 24;
14         h &= ~high;
15     }
16     return h;
17 }
18
19 } // namespace my
```

Листинг 8: lab3/hash_table.h

```
1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий реализацию простейшей хэш-таблицы
4   * @date Февраль 2020
5  */
6  #ifndef HASH_TABLE_H
7  #define HASH_TABLE_H
8
9  #include <functional>
10 #include <cstddef>
11 #include <vector>
12 #include <forward_list>
13 #include <list>
14 #include <utility>
15 #include <optional>
16 #include <cmath>
17 #include <cassert>
18
19 #ifndef NDEBUG
20 #include <iostream>
21 #endif
22
23 namespace my
24 {
25
26 /**
27  * Простейшая реализация хэш-таблицы (аналога `std::unordered_multimap`)
28  * @tparam Key тип ключа данных
29  * @tparam T тип хранимого значения
30  * @tparam Hash тип, являющийся default-constructible и объект которого можно вызвать
31  * с параметром типа `Key` и получить хэш в виде `std::size_t`
32  */
33 template <typename Key, typename T, typename Hash = std::hash<Key>>
34 class HashTable
35 {
36 public:
37     HashTable()
38         : m_hasher({})
39         , m_data(std::vector<Bucket>(17))
40     {}
41     HashTable(const HashTable&) = default;
42     HashTable& operator=(const HashTable&) = default;
43     HashTable(HashTable&&) noexcept = default;
44     HashTable& operator=(HashTable&&) noexcept = default;
45
46     /**
47      * Добавляет в хэш-таблицу элемент с заданным ключом и значением
48      * @param[in] key ключ добавляемого элемента
49      * @param[in] value значение добавляемого элемента
50      */
51     void emplace(const Key& key, T value)
```

```
52      {
53          emplace(key, std::forward_list<T>{ std::move(value) });
54      }
55
56      /**
57       * Добавляет в хэш-таблицу список элементов элемент с заданным ключом
58       * @param[in] key ключ добавляемых элемента
59       * @param[in] values список, содержащий значения добавляемых элементов
60       */
61      void emplace(const Key& key, std::forward_list<T> values)
62      {
63          std::size_t hash = m_hasher(key);
64          Bucket& bucket = m_data[hash % m_data.size()];
65          emplace(key, std::move(values), bucket, hash);
66      }
67
68      /**
69       * Ищет в хэш-таблице элементы с заданным ключом
70       * @param[in] key ключ, по которому будет осуществляться поиск
71       * @return `std::forward_list`, содержащий значения,
72       * хранящиеся в хэш-таблице по ключу `key`
73       */
74      [[nodiscard]] const std::forward_list<T>& equal_range(const Key& key) const
75      {
76          const Bucket& bucket = m_data[get_index(key)];
77  #ifndef NDEBUG
78          assert(bucket.size() <= m_max_bucket_size);
79  #endif
80          for (const Node& node : bucket)
81              if (node.hash_and_key().key() == key)
82                  return node.values();
83          return m_empty_list;
84      }
85
86  private:
87      class HashAndKey
88      {
89      public:
90          HashAndKey(std::size_t hash, Key key)
91              : m_hash(hash), m_key(std::move(key))
92          {}
93          HashAndKey() = delete;
94          HashAndKey(const HashAndKey&) = default;
95          HashAndKey& operator=(const HashAndKey&) = default;
96          HashAndKey(HashAndKey&&) noexcept = default;
97          HashAndKey& operator=(HashAndKey&&) noexcept = default;
98
99          [[nodiscard]] std::size_t hash() const { return m_hash; }
100
101         [[nodiscard]] const Key& key() const { return m_key; }
102         [[nodiscard]] Key& key() { return m_key; }
103
104     private:
105         std::size_t m_hash;
106         Key m_key;
107     };
108
109     class Node
110     {
111     public:
112         Node(HashAndKey hash_and_key, std::forward_list<T> values)
113             : m_hash_and_key(std::move(hash_and_key)), m_values(std::move(values))
114         {}
115         Node() = delete;
116         Node(const Node&) = default;
117         Node& operator=(const Node&) = default;
118         Node(Node&&) noexcept = default;
119         Node& operator=(Node&&) noexcept = default;
120
121         [[nodiscard]] const HashAndKey& hash_and_key() const { return m_hash_and_key; }
```

```
122         [[nodiscard]] HashAndKey& hash_and_key() { return m_hash_and_key; }
123
124         [[nodiscard]] const std::forward_list<T>& values() const { return m_values; }
125         [[nodiscard]] std::forward_list<T>& values() { return m_values; }
126
127     private:
128         HashAndKey m_hash_and_key;
129         std::forward_list<T> m_values;
130     };
131
132     using Bucket = std::list<Node>;
133
134     Hash m_hasher;
135     std::vector<Bucket> m_data;
136     std::size_t m_max_bucket_size = 3;
137     std::size_t not_empty_count = 0;
138
139     static inline const std::forward_list<T> m_empty_list{};
140
141     [[nodiscard]] std::size_t get_index(const Key& key) const
142     {
143         return m_hasher(key) % m_data.size();
144     }
145
146     void emplace(const Key& key, std::forward_list<T> values, std::size_t hash)
147     {
148         Bucket& bucket = m_data[hash % m_data.size()];
149         emplace(key, std::move(values), bucket, hash);
150     }
151
152     void emplace(const Key& key, std::forward_list<T> values, Bucket& bucket, std::
    size_t hash)
153     {
154         for (Node& node : bucket)
155         {
156             if (node.hash_and_key().key() == key)
157             {
158                 for (T& value : values)
159                     node.values().emplace_front(std::move(value));
160                 return;
161             }
162         }
163         if (bucket.size() != m_max_bucket_size)
164         {
165             if (bucket.empty())
166                 ++not_empty_count;
167             bucket.emplace_front(HashAndKey(hash, key), std::move(values));
168         }
169         else
170         {
171             rehash();
172             emplace(key, std::move(values), hash);
173         }
174     }
175
176     void rehash()
177     {
178         std::size_t new_size, new_max_bucket_size;
179         if (not_empty_count * 7 > m_data.size())
180         {
181             new_size = m_data.size() * 2;
182             new_max_bucket_size = 3;
183         }
184         else
185         {
186             new_size = m_data.size();
187             new_max_bucket_size = m_max_bucket_size + 1;
188         }
189 #ifndef NDEBUG
190         std::cerr << "Rehash: " << m_data.size() << ", " << m_max_bucket_size
```

```
                  << " --> " << new_size << ", " << new_max_bucket_size << std::endl;
#endif
        HashTable new_table;
        new_table.m_max_bucket_size = new_max_bucket_size;
        new_table.m_data.resize(new_size);

        for (Bucket& bucket : m_data)
            for (Node& node : bucket)
                new_table.emplace(std::move(node.hash_and_key().key()), std::move(node.values()), node.hash_and_key().hash());

        std::swap(*this, new_table);
    }
};

} // namespace my

#endif // HASH_TABLE_H
```

Листинг 9: lab3/main.cpp

```
#include "entry.h"
#include "io_operations.h"
#include "dummy.h"
#include "elf.h"
#include "rot13.h"
#include "hash_table.h"
#include <boost/program_options.hpp>
#include <algorithm>
#include <chrono>
#include <fstream>
#include <iostream>
#include <unordered_map>
#include <map>
#include <string>
#include <vector>
#include <cassert>
#include <functional>
#include <random>
#include <type_traits>

using ArraySize = std::size_t;
using HashName = std::string;
using Time = std::int64_t;
using Data = std::vector<Entry>;
using Iterator = Data::iterator;
using SizeToTime = std::map<ArraySize, Time>;
using SizeToPercentage = std::map<ArraySize, double>;
using TestTimeResult = std::map<HashName, SizeToTime>;
using TestCollisionResult = std::map<HashName, SizeToPercentage>;

static std::mt19937 prng(std::random_device{}());

enum class HashAlgorithm
{
    STDHASH,
    DUMMY,
    ROT13,
    ROT19,
    ELF,
};

static const std::map<HashAlgorithm, HashName> hash_names =
{
    { HashAlgorithm::STDHASH, "std::hash" },
    { HashAlgorithm::DUMMY,   "dummy" },
    { HashAlgorithm::ROT13,   "rot13" },
    { HashAlgorithm::ROT19,   "rot19" },
    { HashAlgorithm::ELF,     "elf" },
```

```
};

std::vector<Entry::Trainer> pick_random_elements(Data::const_iterator begin, Data::
    const_iterator end, std::size_t length)
{
    std::ptrdiff_t size = std::distance(begin, end);
    std::uniform_int_distribution<std::ptrdiff_t> dist(0, size - 1);
    std::vector<Entry::Trainer> answer;
    for (std::size_t i = 0; i < length; ++i)
        answer.push_back(std::next(begin, dist(prng))->trainer());
    return answer;
}

template <typename Hash>
SizeToTime test_hash_timings(const Data& data, const std::map<std::size_t, std::vector<
    Entry::Trainer>>& size_to_elements)
{
    SizeToTime answer;
    using namespace std::chrono;
    for (auto& [_size, elements] : size_to_elements)
    {
        std::size_t size = std::min(_size, data.size());
        Data::const_iterator data_size_it = std::next(data.begin(), static_cast<std::
    ptrdiff_t>(size));
        my::HashTable<Entry::Trainer, Entry, Hash> mmap;
        for (Data::const_iterator it = data.begin(); it != data_size_it; ++it)
            mmap.emplace(it->trainer(), *it);
#ifndef NDEBUG
        std::unordered_multimap<Entry::Trainer, Entry, Hash> std_mmap;
        for (Data::const_iterator it = data.begin(); it != data_size_it; ++it)
            std_mmap.emplace(it->trainer(), *it);
#endif
        time_point<high_resolution_clock> start = high_resolution_clock::now();
        for (const Entry::Trainer& element_to_search : elements)
        {
            const std::forward_list<Entry>& found = mmap.equal_range(element_to_search);
#ifndef NDEBUG
            auto [range_begin, range_end] = std_mmap.equal_range(element_to_search);
            assert(std::distance(found.begin(), found.end()) == std::distance(
    range_begin, range_end));
#endif
        }
        time_point<high_resolution_clock> end = high_resolution_clock::now();
        answer[size] = static_cast<Time>(
                            duration_cast<std::chrono::nanoseconds>(end - start).count()
    /
                            static_cast<double>(elements.size())
                        );
    }

    return answer;
}

TestTimeResult test_all_timings(const Data& data, const std::vector<ArraySize>& sizes)
{
    const std::size_t SEARCH_COUNT = 1000;
    TestTimeResult answer;

    std::map<std::size_t, std::vector<Entry::Trainer>> elements_to_search;
    for (ArraySize size : sizes)
    {
        size = std::min(size, data.size());
        if (elements_to_search.contains(size))
            continue;
        Data::const_iterator data_size_it = std::next(data.begin(), static_cast<std::
    ptrdiff_t>(size));
        elements_to_search[size] = pick_random_elements(data.begin(), data_size_it,
    SEARCH_COUNT);
    }

```

```
112     for (auto& [algo, name] : hash_names)
113     {
114         std::cerr << "Testing timings for " << name << "..." << std::endl;
115         switch (algo)
116         {
117         case HashAlgorithm::STDHASH:
118             answer.emplace(name, test_hash_timings<std::hash<Entry::Trainer>>(data,
    elements_to_search));
119             break;
120         case HashAlgorithm::DUMMY:
121             answer.emplace(name, test_hash_timings<my::DummyHash>(data,
    elements_to_search));
122             break;
123         case HashAlgorithm::ROT13:
124             answer.emplace(name, test_hash_timings<my::Rot13Hash>(data,
    elements_to_search));
125             break;
126         case HashAlgorithm::ROT19:
127             answer.emplace(name, test_hash_timings<my::Rot19Hash>(data,
    elements_to_search));
128             break;
129         case HashAlgorithm::ELF:
130             answer.emplace(name, test_hash_timings<my::ElfHash>(data, elements_to_search
    ));
131             break;
132         }
133         std::cerr << "Done!" << std::endl;
134     }
135
136     return answer;
137 }
138
139 template <typename Hash>
140 SizeToPercentage test_hash_collisions(const std::map<std::size_t, std::vector<std::
    string>>& size_to_elements)
141 {
142     SizeToPercentage answer;
143     Hash hasher;
144     for (auto& [size, elements] : size_to_elements)
145     {
146         std::map<std::size_t, std::size_t> collision_count;
147         for (const std::string& element : elements)
148             ++collision_count[hasher(element)];
149         std::size_t collisions = 0;
150         for (auto [hash, number] : collision_count)
151             if (number > 1)
152                 collisions += number;
153         answer[size] = static_cast<double>(collisions) * 100 / size;
154     }
155
156     return answer;
157 }
158
159 TestCollisionResult test_all_collisions(const Data& data, const std::vector<ArraySize>&
    sizes)
160 {
161     std::vector<std::string> strings;
162     strings.reserve(data.size());
163     for (const Entry& entry : data)
164         strings.emplace_back(entry.country() + entry.city() + entry.club() + entry.
    trainer());
165     std::sort(strings.begin(), strings.end());
166     std::vector<std::string>::iterator end_it = std::unique(strings.begin(), strings.end
    ());
167     strings.resize(static_cast<std::size_t>(std::distance(strings.begin(), end_it)));
168     std::shuffle(strings.begin(), strings.end(), prng);
169
170     TestCollisionResult answer;
171     std::map<std::size_t, std::vector<std::string>> elements;
172     for (ArraySize size : sizes)
```

```
173     {
174         size = std::min(size, strings.size());
175         if (elements.contains(size))
176             continue;
177         std::vector<std::string>::const_iterator strings_size_it = std::next(strings.
    begin(), static_cast<std::ptrdiff_t>(size));
178         elements[size] = std::vector<std::string>(strings.cbegin(), strings_size_it);
179     }
180 
181     for (auto& [algo, name] : hash_names)
182     {
183         std::cerr << "Testing collisions for " << name << "..." << std::endl;
184         switch (algo)
185         {
186         case HashAlgorithm::STDHASH:
187             answer.emplace(name, test_hash_collisions<std::hash<std::string>>(elements))
    ;
188             break;
189         case HashAlgorithm::DUMMY:
190             answer.emplace(name, test_hash_collisions<my::DummyHash>(elements));
191             break;
192         case HashAlgorithm::ROT13:
193             answer.emplace(name, test_hash_collisions<my::Rot13Hash>(elements));
194             break;
195         case HashAlgorithm::ROT19:
196             answer.emplace(name, test_hash_collisions<my::Rot19Hash>(elements));
197             break;
198         case HashAlgorithm::ELF:
199             answer.emplace(name, test_hash_collisions<my::ElfHash>(elements));
200             break;
201         }
202         std::cerr << "Done!" << std::endl;
203     }
204 
205     return answer;
206 }
207 
208 int main(int argc, char* argv[]) try
209 {
210     std::ios::sync_with_stdio(false);
211     std::cin.tie(nullptr);
212 
213     namespace po = boost::program_options;
214     po::options_description desc("Allowed options");
215     desc.add_options()
216         ("help,H", "Print this message")
217         ("sizes,S", po::value<std::string>()->required(), "Text file with data sizes to
    be testes in the following format:\n"
218                                                           "size_0 size_1 size_2 ...
    size_n")
219 
220         ("input,I", po::value<std::string>()->required(), "File (csv or sqlite) with
    football clubs data. Format:\n"
221                                                           "* if csv: country;club;city;
    trainer;year;score\n"
222                                                           "* if sqlite: table 'entries'
    with columns 'country', 'club', 'city', 'trainer', 'year', 'score'")
223         ("format,F", po::value<std::string>(), "Input file format (csv or sqlite)")
224         ("output_time,T", po::value<std::string>()->required(), "csv file to write test
    timing results, the format is:\n"
225                                                                 "algo_name;
    result_for_size_0;...;result_for_size_n")
226         ("output_collision,C", po::value<std::string>()->required(), "csv file to write
    test collision results, the format is:\n"
227                                                                     "algo_name;
    result_for_size_0;...;result_for_size_n")
228         ;
229 
230     po::variables_map vm;
231     try
```

```
232     {
233         po::store(parse_command_line(argc, argv, desc), vm);
234         if (vm.contains("help"))
235         {
236             std::cout << desc << "\n";
237             return 0;
238         }
239         po::notify(vm);
240     }
241     catch (const po::error& error)
242     {
243         std::cerr << "Error while parsing command-line arguments: "
244                   << error.what() << "\nPlease use --help to see help message\n";
245         return 1;
246     }
247 
248     std::string input_filename = vm["input"].as<std::string>();
249     std::string sizes_filename = vm["sizes"].as<std::string>();
250     std::string output_time_filename = vm["output_time"].as<std::string>();
251     std::string output_collision_filename = vm["output_collision"].as<std::string>();
252     std::string format;
253     if (vm.contains("format"))
254     {
255         format = vm["format"].as<std::string>();
256     }
257     else
258     {
259         if (input_filename.ends_with(".csv"))
260         {
261             format = "csv";
262         }
263         else if (input_filename.ends_with(".sqlite"))
264         {
265             format = "sqlite";
266         }
267         else
268         {
269             std::cerr << "Invalid format. Please either specify format manually with "
270                          "--format or use --input with extension .csv or .sqlite.\n"
271                          "Please use --help to see detailed help message";
272         }
273     }
274 
275     if (format != "csv" && format != "sqlite")
276     {
277         std::cerr << "Invalid format. Please use --help see help message\n";
278         return 1;
279     }
280 
281     std::cerr << "Reading data..." << std::endl;
282     Data data;
283     if (format == "csv")
284         data = read_data_from_csv(input_filename);
285     else if (format == "sqlite")
286         data = read_data_from_sqlite(input_filename);
287     std::vector<ArraySize> sizes = read_sizes(sizes_filename);
288     shrink_sizes(sizes, data.size());
289     std::cerr << "Done!" << std::endl;
290 
291     // timings
292     {
293         // csv header
294         std::ofstream output(output_time_filename);
295         output << "name";
296         for (ArraySize size : sizes)
297             output << ';' << size;
298         output << '\n';
299 
300         TestTimeResult results = test_all_timings(data, sizes);
301         std::cerr << "Timings:\n";
```

```
302         for (auto& [name, timings] : results)
303         {
304             std::cerr << std::endl << "Algorithm: " << name << std::endl;
305             for (auto [size, time] : timings)
306                 std::cerr << size << ": " << time << std::endl;
307             print_timings_csv_line(output, name, timings);
308         }
309     }
310 
311     // collisions
312     {
313         // csv header
314         std::ofstream output(output_collision_filename);
315         output << "name";
316         for (ArraySize size : sizes)
317             output << ';' << size;
318         output << '\n';
319 
320         TestCollisionResult results = test_all_collisions(data, sizes);
321         std::cerr << "Collisions:" << std::endl;
322         for (auto& [name, percentage] : results)
323         {
324             std::cerr << std::endl << "Algorithm: " << name << std::endl;
325             for (auto [size, time] : percentage)
326                 std::cerr << size << ": " << time << std::endl;
327             print_collisions_csv_line(output, name, percentage);
328         }
329     }
330 
331     return 0;
332 }
333 catch (const std::exception& e)
334 {
335     std::cerr << e.what() << '\n';
336     return 1;
337 }
```

Листинг 10: lab3/run.py

```
1  import subprocess
2  import os
3  import numpy as np
4  import pandas as pd
5  from matplotlib import pyplot as plt
6 
7  try:
8      print("Generating data...")
9      if not os.path.isfile("data.sqlite"):
10         subprocess.call("../generate_data/generate_data --size=1000000 --output=data.
       sqlite", shell=True)
11     print("Done!")
12 
13     if not os.path.isfile("sizes.txt"):
14         f = open("sizes.txt", "w")
15         f.write(" ".join(np.logspace(4, 6, 20).astype(int).astype(str)) + "\n")
16         f.close()
17 
18     print("Running tests...")
19     if not os.path.isfile("results_time.csv") or not os.path.isfile("results_collision.
       csv"):
20         subprocess.call("./3_hashes --input=data.sqlite --sizes=sizes.txt --output_time=
       results_time.csv --output_collision=results_collision.csv 2> test_hash.log", shell=
       True)
21     print("Done!")
22 
23     raw_time = pd.read_csv("results_time.csv", sep=';')
24     data_time = dict()
25     columns_time = list(raw_time.columns[1:].astype(int))
26     for i in range(raw_time.shape[0]):
```

```
27          data_time[raw_time.iloc[i]["name"]] = list(raw_time.iloc[i][1:])
28      plt.figure(figsize=(12, 8))
29      for name, timings in data_time.items():
30          plt.plot(columns_time, timings, label=name)
31          plt.xscale('log')
32          plt.yscale('log')
33          plt.legend()
34      plt.show()
35
36      raw_collision = pd.read_csv("results_collision.csv", sep=';')
37      data_collision = dict()
38      columns_collision = list(raw_collision.columns[1:].astype(int))
39      for i in range(raw_collision.shape[0]):
40          data_collision[raw_collision.iloc[i]["name"]] = list(raw_collision.iloc[i][1:])
41      plt.figure(figsize=(12, 8))
42      for name, collisions in data_collision.items():
43          plt.plot(columns_collision, collisions, label=name)
44          plt.xscale('log')
45          plt.legend()
46      plt.show()
47  except Exception as e:
48      print(e)
```