

**Московский авиационный институт  
(национальный исследовательский университет)**

**Факультет информационных технологий и прикладной  
математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №3 по курсу «Информационный поиск»**

Студент: А. Т. Бахарев  
Преподаватель: А. А. Кухтичев  
Группа: М8О-108М  
Дата:  
Оценка:  
Подпись:

**Москва, 2021**

## Лабораторная работа №3 «Булев индекс»

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов. Требования к индексу:

- Самостоятельно разработанный, бинарный формат представления данных. Формат необходимо описать в отчёте, в побайтовом (или побитовом) представлении.
- Формат должен предполагать расширение, т.к. в следующих работах он будет меняться под требования новых лабораторных работ.
- Использование текстового представления или готовых баз данных не допускается.
- Кроме обратного индекса, должен быть создан «прямой» индекс, содержащий в себе как минимум заголовки документов и ссылки на них (понадобятся для выполнения ЛР4, при генерации страницы поисковой выдачи).
- Для термов должна быть как минимум понижена капитализация.

В отчёте должно быть отмечено как минимум:

- Выбранное внутренне представление документов после токенизации.
- Выбранный метод сортировки, его достоинства и недостатки для задачи индексации.

Среди результатов и выводов работы нужно указать:

- Количество термов.
- Средняя длина терма. Сравнить со средней длиной токена, вычисленной в ЛР1 по курсу ОТЕЯ. Объяснить причину отличий.
- Скорость индексации: общую, в расчёте на один документ, на килобайт текста.
- Оптимальна ли работа индексации? Что можно ускорить? Каким образом? Чем она ограничена? Что произойдёт, если объём входных данных увеличится в 10 раз, в 100 раз, в 1000 раз?

# 1 Описание

Требуется разработать поисковый индекс, который затем будет использоваться для булевого поиска по документом. Элемент индекса в моей реализации - это пара ключ/значение, где ключом является токен, а значением - вектор с номерами документов, где этот токен встречается. Для записи и чтение такого индекса я использую библиотеку Boost, которая предоставляют удобный интерфейс для сериализации и десериализации контейнеров из STL.

Для построения индекса использовался алгоритм SPIMI. Для того, чтобы не отслеживать объем потребляемой оперативной памяти при построении индекса, я ограничил число документов одним индексом(10000 штук). Таким образом, количество индексов можно изменять.

## 2 Исходный код

```
1 | #include <iostream>
2 | #include <common.h>
3 | #include <boost/filesystem.hpp>
4 | #include <boost/range/iterator_range.hpp>
5 | #include <boost/serialization/unordered_map.hpp>
6 | #include <boost/serialization/vector.hpp>
7 | #include <boost/archive/binary_oarchive.hpp>
8 | #include <boost/archive/binary_iarchive.hpp>
9 | #include <unordered_map>
10 | #include <vector>
11 | #include <utility>
12 | #include <tuple>
13 |
14 | #include <utils.hpp>
15 | #include "engine.h"
16 |
17 | int SearchEngine::save_index_to_file(const fs::path& path_to_idx, const dictionary&
    dict)
18 | {
19 |     std::ofstream f(path_to_idx.c_str(), std::ios::binary);
20 |     if (f.fail())
21 |         return -1;
22 |     boost::archive::binary_oarchive oa(f);
23 |     oa << dict;
24 |     return 0;
25 | }
26 |
27 | int SearchEngine::load_index_from_file(const std::string& path_to_idx, dictionary&
    dict)
28 | {
29 |     std::ifstream f(path_to_idx.c_str(), std::ios::binary);
30 |     if (f.fail())
31 |         return -1;
32 |     boost::archive::binary_iarchive ia(f);
33 |     ia >> dict;
34 |     return 0;
35 | }
36 |
37 | void SearchEngine::build_index(const fs::path& work_folder)
38 | {
39 |     utils::recreate_dir_safely(indexed_data_folder_path);
40 |
41 |     dictionary dict;
42 |     ull doc_count = 0;
43 |     ull index_count = 0;
44 |     for (auto& entry : boost::make_iterator_range(fs::directory_iterator(
        tokenized_data_folder_path), {}))
```

```

45 {
46     if (doc_count < FilesInIndex)
47     {
48         std::string filename_str = entry.path().stem().string();
49         ull filename_int = std::stoi(filename_str);
50
51         std::fstream file(entry.path().string());
52         std::string term;
53         while (file >> term)
54         {
55             auto it = dict.find(term);
56             if (it == dict.end())
57             {
58                 dict.insert(std::move(std::make_pair(term, std::vector<ull>{filename_int})));
59             }
60             else
61             {
62                 if (it->second.back() != filename_int)
63                     it->second.emplace_back(filename_int);
64             }
65         }
66
67         ++doc_count;
68     }
69     else
70     {
71         std::string index_file = std::to_string(index_count) + ".idx";
72         save_index_to_file(indexed_data_folder_path / fs::path(index_file), dict);
73         dict.clear();
74
75         ++index_count;
76         doc_count = 0;
77     }
78 }
79 }
80
81 std::vector<int> SearchEngine::search(const std::vector<std::string> terms)
82 {
83     std::vector<std::vector<int>> search_result(terms.size());
84
85     dictionary dict;
86     for (auto& index_path : boost::make_iterator_range(fs::directory_iterator(
87         indexed_data_folder_path), {}))
88     {
89         load_index_from_file(index_path.path().string(), dict);
90         for (int i = 0; i < terms.size(); ++i)
91         {
92             auto it = dict.find(terms[i]);
93             if (it != dict.end())

```

```

93     {
94         search_result[i].insert(search_result[i].end(), it->second.begin(), it->second.
           end());
95     }
96 }
97 }
98
99 for (auto& arr : search_result)
100 {
101     std::sort(arr.begin(), arr.end());
102 }
103
104
105 std::vector<int> res;
106
107 for (int i = 0; i < search_result.size() - 1; ++i)
108 {
109     std::set_intersection(search_result[i].begin(), search_result[i].end(),
110                           search_result[i + 1].begin(), search_result[i + 1].end(),
111                           std::back_inserter(res));
112 }
113
114 return res;
115 }
116
117 int main()
118 {
119     bool is_build = false;
120     SearchEngine se;
121
122     if (is_build)
123     {
124         se.build_index(indexed_data_folder);
125     }
126     else // search
127     {
128         std::vector<std::string> terms = { "pharmacy", "period" };
129
130         auto res = se.search(terms);
131         std::sort(res.begin(), res.end());
132
133         for (auto& i : res)
134         {
135             std::cout << i << " ";
136         }
137         std::cout << std::endl;
138
139         std::cout << "TOTAL: " << res.size();
140     }

```

```
141 ||  
142 || return 0;  
143 || }
```

### 3 Выводы

Выполнив третью лабораторную работу по курсу «Информационный поиск», я разработал и реализовал простейший инвертированный индекс для поиска термов по документам. Так же, я изучил еще одну возможность библиотеки Boost - запись и чтение стандартных контейнеров на диск.



## Список литературы

- [1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Ключина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))
- [2] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008