Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Информационный поиск»

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

Москва, 2021

Лабораторная работа №1 «Добыча корпуса документов»

Нужно реализовать ввод поисковых запросов и их выполнение над индексом, получение поисковой выдачи. Синтаксис поисковых запросов:

- Размер «сырых» данных.
- Пробел или два амперсанда, «&&», соответствуют логической операции «И».
- Две вертикальных «палочки», «||» – логическая операция «ИЛИ»
- Восклицательный знак, «!» – логическая операция «НЕТ»
- Могут использоваться скобки.

Парсер поисковых запросов должен быть устойчив к переменному числу пробелов, максимально толерантен к введённому поисковому запросу.

Примеры запросов:

ионный институт

тый) автомобиль

руки !ноги

Для демонстрации работы поисковой системы должен быть реализован веб-сервис, реализующий базовую функциональность поиска из двух страниц:

- Начальная страница с формой ввода поискового запроса.
- Страница поисковой выдачи, содержащая в себе форму ввода поискового запроса, 50 результатов поиска в виде текстов заголовков документов и ссылок на эти документы, а так же ссылку на получение следующих 50 результатов.

Так же должна быть реализована утилита командной строки, загружающая индекс и выполняющая поиск по нему для каждого запроса на отдельной строчке входного файла. В отчёте должно быть отмечено:

- Скорость выполнения поисковых запросов.
- Примеры сложных поисковых запросов, вызывающих длительную работу.
- Каким образом тестировалась корректность поисковой выдачи

1 Описание

Требуется разработать поиск термов по индексированным документам. Алгоритм поиска состоит в следующем: каждый индекс будем загружать в память и в нем искать термы, которые заданы пользователем. Затем реализуем объединение или пересечение множеств, в зависимости от запроса пользователя.

В качестве примера работы я решил найти документы, которые содержат в себе токены "pharmacy "period". Всего было найдено 80 документов. Время поиска - 1343 мс. Если выполнить поиск таких термов в Visual Studio Code, то поиск займет порядка 20 секунд.

Реализованная индексация и поиск имеют некоторые недостатки, но для некоторых задач он вполне приемлемый и может показывать довольно быстрые результаты.

2 Исходный код

```
#include <iostream>
#include <common.h>
#include <boost/filesystem.hpp>
#include <boost/range/iterator_range.hpp>
#include <boost/serialization/unordered_map.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/archive/binary_iarchive.hpp>
#include <unordered_map>
#include <vector>
#include <utility>
#include <tuple>

#include <utils.hpp>
#include "engine.h"

int SearchEngine::save_index_to_file(const fs::path& path_to_idx, const dictionary&
    dict)
{
  std::ofstream f(path_to_idx.c_str(), std::ios::binary);
  if (f.fail())
    return -1;
  boost::archive::binary_oarchive oa(f);
  oa << dict;
  return 0;
}

int SearchEngine::load_index_from_file(const std::string& path_to_idx, dictionary&
    dict)
{
  std::ifstream f(path_to_idx.c_str(), std::ios::binary);
  if (f.fail())
    return -1;
  boost::archive::binary_iarchive ia(f);
  ia >> dict;
  return 0;
}

void SearchEngine::build_index(const fs::path& work_folder)
{
  utils::recreate_dir_safely(indexed_data_folder_path);

  dictionary dict;
  ull doc_count = 0;
  ull index_count = 0;
  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(
          indexed_data_folder_path), {}))
87    {
88      load_index_from_file(index_path.path().string(), dict);
89      for (int i = 0; i < terms.size(); ++i)
90      {
91        auto it = dict.find(terms[i]);
92        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 }
```

```
  1 #pragma once
  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
 13 #include <utils.hpp>
 14
 15 namespace fs = boost::filesystem;
 16 using ull = unsigned long long;
 17
 18 constexpr ull FilesInIndex = 1000;
 19
 20 struct dictionary
 21 {
 22   std::unordered_map<std::string, std::vector<ull>> umap;
 23
 24   auto find(const std::string& s)
 25   {
 26     return umap.find(s);
 27   }
 28
 29   auto insert(const std::pair<std::string, std::vector<ull>>&& val)
 30   {
 31     return umap.insert(val);
 32   }
 33
 34   auto end()
 35   {
 36     return umap.end();
 37   }
 38
 39   auto clear()
 40   {
 41     return umap.clear();
 42   }
 43
 44   template <typename Archive>
 45   void serialize(Archive& ar, const unsigned int version)
 46   {
```

```
47      ar& umap;
48    }
49
50  };
51
52  class SearchEngine
53  {
54  public:
55    std::vector<int> search(const std::vector<std::string> terms);
56    void build_index(const fs::path& work_folder);
57
58  private:
59    int save_index_to_file(const fs::path& path_to_idx, const dictionary& dict);
60    int load_index_from_file(const std::string& path_to_idx, dictionary& dict);
61
62  private:
63    const fs::path work_folder = LR"(C:\Users\Alexey\Desktop\IS\2020-03-13)";
64    const fs::path tokenized_data_folder_path = work_folder / tokenized_data_folder;
65    const fs::path indexed_data_folder_path = work_folder / indexed_data_folder;
66  };
```

3 Выводы

Выполнив четвертую лабораторную работу по курсу «Информационный поиск», я реализовал поиск термов по индексам. В процессе работы были проанализированы существующие реализации такого поиска и из них были извлечены некоторые полезные детали. Подводя итоги, можно сказать, что эта и предыдущие работы дали много практических и теоретических знаний в предметной области Information Retrieval.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))

[2] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008