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: И. А. Мариничев
Преподаватель: А. А. Кухтичев
Группа: М8О-408Б-19

Дата:
Оценка:
Подпись:

Москва, 2023

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 длинной 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 ограниченна? Что произойдёт, если объём входных данных увеличится в 10 раз, в 100 раз, в 1000 раз?

1 Описание

По выбранному корпусу документов строятся прямой и обратный индексы. Построение индекса занимает 379853 ms (6.33088333 min), сохранение занимает 1.14806e+06 ms (19.13433333 min).

Прямой индекс имеет следующее представление:

<Documents Total>

<Title 0> <URL 0> <Word Count 0>

<Title 1> <URL 1> <Word Count 1>

...

<Title N> <URL N> <Word Count N>

Обратный индекс имеет следующее представление:

<Documents Total>

<Word 0> <Postings Size 0> <Postings 0>

<Word 1> <Postings Size 1> <Postings 1>

...

<Word N> <Postings Size N> <Postings N>

2 Исходный код

Ниже приведен класс индекса, реализующий функционал построения и сохранения инекса в файл:

```
// index.h
#include <iostream>
#include <fstream>
#include <sstream>
#include <locale>
#include <vector>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <memory>
#include <chrono>

struct Document
{
    std::wstring title;
    std::wstring url;
    uint32_t wordCount;
};

class Index
{
public:
    void Build(std::string &inputFile);

    void Save(std::string &outputFile);
    void SaveIndex(std::string &outputFile);
    void SaveInvertedIndex(std::string &outputFile);

    void Load(std::string &inputFile);
    void LoadIndex(std::string &inputFile);
    void LoadInvertedIndex(std::string &inputFile);

    friend class Query;

private:
    uint32_t docTotal;
    std::vector<Document> docIndex;
    std::unordered_map<std::wstring, std::vector<uint32_t>> invertedIndex;

    bool GetDocInfo(std::wifstream &input);
    void GetDocContents(uint32_t id, std::wifstream &input);
};
```

```
// index.cpp
#include "index.h"
```

```
3
4  void Index::Build(std::string &inputFile)
5  {
6      std::cout << "Building index...\n";
7      auto start = std::chrono::high_resolution_clock::now();
8
9      std::wifstream input((inputFile).c_str());
10     for (size_t id = 1; input; ++id)
11     {
12         if (GetDocInfo(input))
13         {
14             GetDocContents(id, input);
15         }
16     }
17     docTotal = docIndex.size();
18     input.close();
19
20     auto end = std::chrono::high_resolution_clock::now();
21     std::cout << "Successfully built index in "
22               << std::chrono::duration<double, std::milli>(end - start).count() << " ms\
                      n";
23 }
24
25 bool Index::GetDocInfo(std::wifstream &input)
26 {
27     uint32_t length;
28     std::wstring line, url, title;
29     Document doc;
30     if (std::getline(input, line))
31     {
32         uint32_t url_start = line.find(L"url=");
33         if (url_start == std::string::npos)
34         {
35             return false;
36         }
37         uint32_t title_start = line.find(L"title=");
38
39         url_start += 4; // len("url=")
40         length = title_start - url_start - 1;
41         url = line.substr(url_start, length);
42
43         title_start += 6; // len("title=")
44         length = line.length() - title_start - 1;
45         title = line.substr(title_start, length);
46
47         doc.title = title;
48         doc.url = url;
49         docIndex.push_back(doc);
50
```

```
51          return true;
52      }
53      else
54      {
55          return false;
56      }
57  }
58
59  void Index::GetDocContents(uint32_t id, std::wifstream &input)
60  {
61      std::wstring line, word;
62      std::unordered_set<std::wstring> words;
63      uint32_t wordCount = 0;
64      while (std::getline(input, line))
65      {
66          if (line == L"</doc>")
67          {
68              break;
69          }
70
71          for (const auto &c : line)
72          {
73              if (isalnum(c, std::locale()))
74              {
75                  word += towlower(c);
76              }
77              else if (word.length())
78              {
79                  words.insert(word);
80                  wordCount++;
81                  word.clear();
82              }
83          }
84
85          if (word.length())
86          {
87              words.insert(word);
88              wordCount++;
89              word.clear();
90          }
91      }
92
93      for (const auto &w : words)
94      {
95          const auto [it, success] = invertedIndex.insert(
96              std::make_pair(w, std::vector<uint32_t>()));
97          it->second.push_back(id - 1);
98      }
99
```

```
100     docIndex[id - 1].wordCount = wordCount;
101 }
102
103 void Index::Save(std::string &outputFile)
104 {
105     std::cout << "Saving index...\n";
106     auto start = std::chrono::high_resolution_clock::now();
107
108     SaveIndex(outputFile);
109     SaveInvertedIndex(outputFile);
110
111     auto end = std::chrono::high_resolution_clock::now();
112     std::cout << "Successfully saved index in "
113               << std::chrono::duration<double, std::milli>(end - start).count() << " ms\
                      n";
114 }
115
116 void Index::SaveIndex(std::string &outputFile)
117 {
118     std::wofstream wFileOut(outputFile.c_str());
119     if (wFileOut)
120     {
121         wFileOut << docIndex.size() << L"\n";
122         for (const auto &doc : docIndex)
123         {
124             wFileOut << doc.title << L" " << doc.url << L" \"" << doc.wordCount << L"
                          \"\n";
125         }
126     }
127     wFileOut.close();
128 }
129
130 void Index::SaveInvertedIndex(std::string &outputFile)
131 {
132     std::wofstream wFileOut((outputFile + "_inverted").c_str());
133     if (wFileOut)
134     {
135         wFileOut << docTotal << L"\n";
136         for (const auto &it : invertedIndex)
137         {
138             wFileOut << it.first << L" ";
139             wFileOut << it.second.size() << L" ";
140             for (size_t i = 0; i < it.second.size(); ++i)
141             {
142                 wFileOut << it.second[i] << L" ";
143             }
144             wFileOut << L"\n";
145         }
146     }
```

```
147     wFileOut.close();
148 }
149
150 void Index::Load(std::string &inputFile)
151 {
152     std::cout << "Loading index...\n";
153     auto start = std::chrono::high_resolution_clock::now();
154
155     LoadIndex(inputFile);
156     LoadInvertedIndex(inputFile);
157
158     auto end = std::chrono::high_resolution_clock::now();
159     std::cout << "Successfully loaded index in "
160               << std::chrono::duration<double, std::milli>(end - start).count() << " ms\
                      n";
161 }
162
163 void Index::LoadIndex(std::string &inputFile)
164 {
165     std::wifstream wFileIn(inputFile.c_str());
166     std::wstring docLine, size;
167     uint32_t reserveSize;
168
169     std::getline(wFileIn, size);
170     std::wstringstream wSizeIn(size);
171     wSizeIn >> reserveSize;
172
173     Document doc;
174     docIndex.reserve(reserveSize);
175
176     size_t pos = 0;
177     std::wstring delimiter = L"\" \"";
178     while (std::getline(wFileIn, docLine))
179     {
180         pos = docLine.find(delimiter);
181         doc.title = docLine.substr(0, pos + 1);
182         docLine.erase(0, pos + delimiter.length() - 1);
183
184         pos = docLine.find(delimiter);
185         doc.url = docLine.substr(0, pos + 1);
186         docLine.erase(0, pos + delimiter.length());
187
188         std::wstringstream wWordCountIn(docLine.substr(0, docLine.length() - 1));
189         wWordCountIn >> doc.wordCount;
190
191         docIndex.push_back(doc);
192
193         docLine.clear();
194     }
```

```
195     wFileIn.close();
196 }
197
198 void Index::LoadInvertedIndex(std::string &inputFile)
199 {
200     std::wifstream wFileIn((inputFile + "_inverted").c_str());
201     uint32_t size;
202     std::wstring word;
203
204     wFileIn >> docTotal;
205     while (!wFileIn.eof())
206     {
207         std::vector<uint32_t> postings;
208         wFileIn >> word;
209         wFileIn >> size;
210
211         uint32_t posting;
212         postings.reserve(size);
213         for (size_t i = 0; i < size; ++i)
214         {
215             wFileIn >> posting;
216             postings.push_back(posting);
217         }
218
219         invertedIndex.insert(std::make_pair(word, postings));
220         word.clear();
221     }
222     wFileIn.close();
223 }
```

3 Выводы

Выполнив третью лабораторную работу по курсу «Информационный поиск», я познакоимлся с двумя основными структурами данных в этой области:

- **Прямой индекс** — список документов, в котором можно найти этот документ по его id. Иными словами, прямой индекс — это массив строк (вектор документов), где id документа — это его индекс.
- **Обратный индекс** — список слов, которые мы «выпотрошили» из всех документов. За каждым словом закреплён список отсортированных id документов (posting list), в которых это слово встретилось.

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.))