Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Информационный поиск»

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

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №5 «Поиск цитат, координатный индекс»

В этом задании необходимо расширить язык запросов булева поиска новым элементом – поиском цитат. Синтаксис этого элемента следующий:

- [«что где когда»] – кавычки, включают режим цитатного поиска для терминов внутри кавычек. Этому запросу удовлетворяют документы, содержащие в себе все термины что, где и когда, причём они должны встретиться внутри документа ровно в этой последовательности, без каких либо вкраплений других терминов.
- [«что где когда» / 5] – аналогично предыдущему пункту, но допускаются вкрапления других терминов так, чтобы расстояние от первого термина цитаты до последнего не превышало бы 5.

Новый элемент может комбинироваться с другими стандартными средствами булева поиска, например:

- [«что где когда» && друзь]
- [«что где когда» || квн]
- [«что где когда» && !«хрустальная сова»]

Для реализации цитатного поиска нужно использовать координатный индекс, т.е. для каждого вхождения термина в документ построить и сохранить список позиций внутри документа, где этот термин встречался.

В отчёте нужно описать формат координатного индекса. Привести статистические данные:

- Размер получившегося индекса.
- Время построения индекса.
- Общее количество позиций. Среднее количество позиций на термин и на пару термин-документ.
- Скорость индексации (кб входных данных в секунду)
- Время выполнения поисковых запросов.
- Примеры долго выполняющихся запросов.

Кроме того, нужно привести примеры запросов и результаты их выполнения. В выводах должны быть указаны недостатки работы, приведены примеры их решения. Что можно сделать, чтобы ускорить «долгие» запросы?

1 Описание

По выбранному корпусу документов строится координатный индекс. Построение индекса занимает 379853 ms (6.33088333 min), сохранение занимает 1.14806e+06 ms (19.13433333 min). Загрузка требует 96074.5 ms (1.601241667 min).

Координатный индекс имеет следующее представление:
<Word 0> <Document Id 0> <Coordinates Size 0> <Coordinates 0>
...
<Word 0> <Document Id N> <Coordinates Size N> <Coordinates N>
...
<Word M> <Document Id 0> <Coordinates Size 0> <Coordinates 0>
...
<Word M> <Document Id N> <Coordinates Size N> <Coordinates N>

Алгоритм, учитывающий вкрапления других слов в цитатном поиске, был изменён и имеет другой синтаксис. Общий вид запросов: “word1 [/k1] word2 [/k2] ... wordN”. Алгоритм не ищет цитату так, чтобы можно было указать расстояние именно между первым и последним словом. Вместо этого предлагается искать цитату с вкраплениями между каждой последовательной парой слов. Если расстояние не указано, то оно подразумевается равным единице. Т. е. запрос “whale /3 fish dish” будет подразумевать, что между словами “whale” и “fish” может быть до 2 других слова, а после “fish” обязано идти слово “dish”. Цитата так же участвует в булевом поиске и может компоноваться с другими выражениями. В данном случае цитата представляется как отдельный терм с особым алгоритмом поиска документов, где она встречается.

Ниже приведены несколько запросов и результаты по ним.

Запрос	Время (в ms)	Количество найденных файлов
"fellowship of the ring"	390.703	36
"fellowship /5 of /5 the /5 ring"	21.9868	38
"fellowship of the ring"&& gandalf	3.6879	26
"fellowship of the ring"|| grinch	6.5681	54
"fellowship of the ring"&& !"peter jackson"	524.674	8

2 Исходный код

Ниже приведены основные методы реализующие работу с координатным индексом:

```
// index.cpp
void Index::SaveCoordinateIndex(std::string &outputFile)
{
    std::wofstream wFileOut((outputFile + "_coordinate").c_str());
    if (wFileOut)
    {
        for (const auto &it : coordinateIndex)
        {
            wFileOut << it.first << L" ";
            wFileOut << it.second.size() << L" ";
            for (size_t i = 0; i < it.second.size(); ++i)
            {
                wFileOut << it.second[i] - 1 << L" ";
            }
            wFileOut << L"\n";
        }
    }
    wFileOut.close();
}

void Index::LoadCoordinateIndex(std::string &inputFile)
{
    std::wifstream wFileIn((inputFile + "_coordinate").c_str());
    std::wstring word, docID;
    uint32_t size;
    while (!wFileIn.eof())
    {
        std::vector<uint32_t> coordinates;
        wFileIn >> word >> docID >> size;

        uint32_t coordinate;
        coordinates.reserve(size);
        for (size_t i = 0; i < size; ++i)
        {
            wFileIn >> coordinate;
            coordinates.push_back(coordinate);
        }

        coordinateIndex.insert(std::make_pair((word + L" " + docID), coordinates));
        word.clear();
    }
    wFileIn.close();
}
```

Ниже приведены основные методы реализующие работу с цитатным поиском:

```
// query.cpp
void Query::ProcessingQuote(std::wstring &quote)
{
    std::wstring word = L"";
    std::wstring wordPrevious;
    std::stack<std::vector<uint32_t>> result;

    size_t i = 0;
    size_t k = 1;
    while (i <= quote.size())
    {
        if ((i == quote.size()) or (quote[i] == SPACE))
        {
            if ((word.size() != 0) and (result.size() == 0))
            {
                wordPrevious = word;
                result.push(GetDocIndices(word));
                word = L"";
            }
            if (word.size())
            {
                std::vector<uint32_t> postings1 = result.top();
                result.pop();
                std::vector<uint32_t> postings2 = GetDocIndices(word);

                result.push(IntersectionForQuote(wordPrevious, word, postings1,
                    postings2, k));
                k = 1;
                wordPrevious = word;
                word = L"";
            }
        }
        else if (quote[i] == SLASH)
        {
            k = 0;
            i++;
            while ('0' <= quote[i] and quote[i] <= '9')
            {
                k *= 10;
                k += (size_t)(quote[i] - '0');
                i++;
            }
            i--;
        }
        else
        {
            word += quote[i];
        }
```

```
48          i++;
49      }
50
51      auto result_ptr = std::make_shared<std::vector<uint32_t>>(
52          result.size() == 1 ? result.top() : std::vector<uint32_t>());
53      operands.push(result_ptr);
54  }
55
56  std::vector<uint32_t> Query::IntersectionForQuote(std::wstring word1, std::wstring
        word2,
57                                                    std::vector<uint32_t> &l, std::vector<
                                                        uint32_t> &r,
58                                                    size_t k)
59  {
60      std::set<uint32_t> result;
61      size_t i = 0, j = 0;
62      while (i < l.size() && j < r.size())
63      {
64          if (l[i] == r[j])
65          {
66              std::vector<uint32_t> coordinates1 = index.coordinateIndex[word1 + L' ' +
                    std::to_wstring(l[i])];
67              std::vector<uint32_t> coordinates2 = index.coordinateIndex[word2 + L' ' +
                    std::to_wstring(r[j])];
68
69              size_t ii = 0, jj = 0;
70              while (ii < coordinates1.size())
71              {
72                  while (jj < coordinates2.size())
73                  {
74                      if ((coordinates2[jj] - coordinates1[ii]) > 0 and
75                          (coordinates2[jj] - coordinates1[ii]) <= k)
76                      {
77                          result.insert(l[i]);
78                      }
79                      else if (coordinates2[jj] > coordinates1[ii])
80                      {
81                          break;
82                      }
83                      jj++;
84                  }
85                  ii++;
86              }
87              i++;
88              j++;
89          }
90          else if (l[i] < r[j])
91          {
92              i++;
```

```
93          }
94          else
95          {
96              j++;
97          }
98      }
99
100     return std::vector<uint32_t>(result.begin(), result.end());
101 }
```

3 Выводы

Выполнив пятую лабораторную работу по курсу «Информационный поиск», я построил свой цитатный поиск, который использует координатный индекс. Координатный индекс занимает значительно больше памяти в отличие от прямого и обратного, но позволяет сохранить информацию о расположении слов в документах друг относительно друга.

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.))