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

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4 «Булев поиск»

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:

- [московский авиационный институт]
- [(красный || желтый) 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

Наши запросы представлены в стандартной булевской модели. Она использует алгебру логики и теорию множеств. Булева выражения строятся над множеством, над элементами которого определены три операции – конъюнкция, дизъюнкция и отрицание:

1. Оператор «&&» указывает на то, что оба слова должны присутствовать в статье.

2. Оператор «||» указывает на то, что одно из слов или оба слова должны присутсовать в статье.

3. Оператор «!» указывает на то, что слово не должно присутствовать в статье.

Основные положения при вычислении:

1. Операции выражения в скобках выполняются первыми, т.е. имеют наивысший приоритет. Для выражения, находящегося в скобках, правила вычисления выражения те же, что и для обычного выражения.

2. Наивысший приоритет из трех возможных операций имеет операция отрицания. Затем операция пересечения, затем операция объединения. При отсутствии скобок операции с наибольшим приоритетом выполняются раньше, чем операции с меньшим приоритетом.

3. Операции с равными приоритетами выполняются в порядке их появления в выражении.

На каждом шаге у нас шесть возможных вариантов действий. Для выбора того или иного действия необходимо проверить сочетание состояния вершины стека операций и рассматриваемый символ. Рассмотрим варианты действий.

Для символов «||» и «&&» правила:

1. Если стек пуст, поместим операцию в стек и перейдем к следующему символу.

2. Пока на вершине стека операция с равным или большим приоритетом, выполнить ее, поместить рассматриваемую в данный момент операцию в стек и перейти к следующему символу.

3. Если на вершине стека операция с меньшим приоритетом, поместить рассматриваемую в данный момент операцию в стек.

Для символа «!» правила:

1. Если стек пуст, поместим операцию в стек и перейдем к следующему символу.
2. Пока на вершине стека операция с большим приоритетом, выполнить ее, поместить рассматриваемую в данный момент операцию в стек и перейти к следующему символу.
3. Если на вершине стека операция с меньшим приоритетом, поместить рассматриваемую в данный момент операцию в стек.

В случае левой скобки необходимо добавить символ в стек операций и перейти к следующему символу.

В случае правой скобки возможны следующие варианты:

1. Стек пуст. Следовательно, не хватает левой скобки, т.е. выражение ошибочно.
2. Выполняем операции из вершины стека, пока операция на вершине стека не левая скобка.

И наконец, рассмотрим действия, необходимые для обработки символа конца выражения:

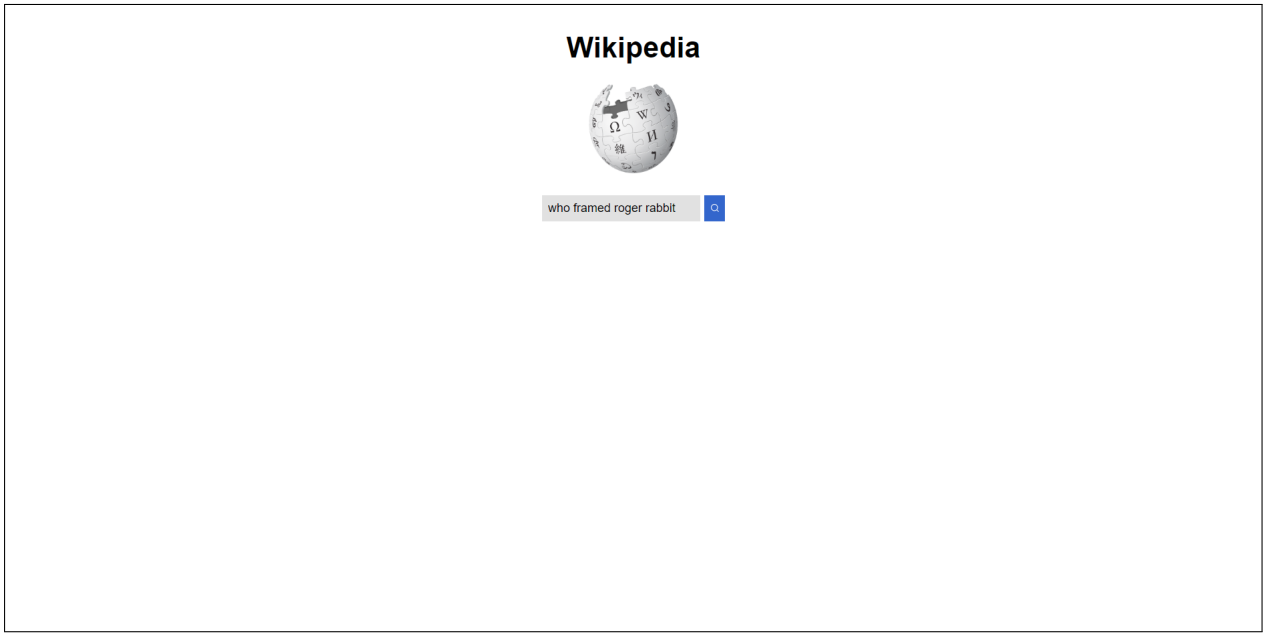
1. Если стек пуст, то выражение вычислено. Следовательно, нужно удалить из стека операндов результат и вывести его.
2. На вершине – левая скобка. Следовательно, левых скобок в выражении было больше, чем правых, т.е. выражение ошибочно.
3. На вершине стека – логическая операция. Необходимо удалить ее из стека и выполнить.

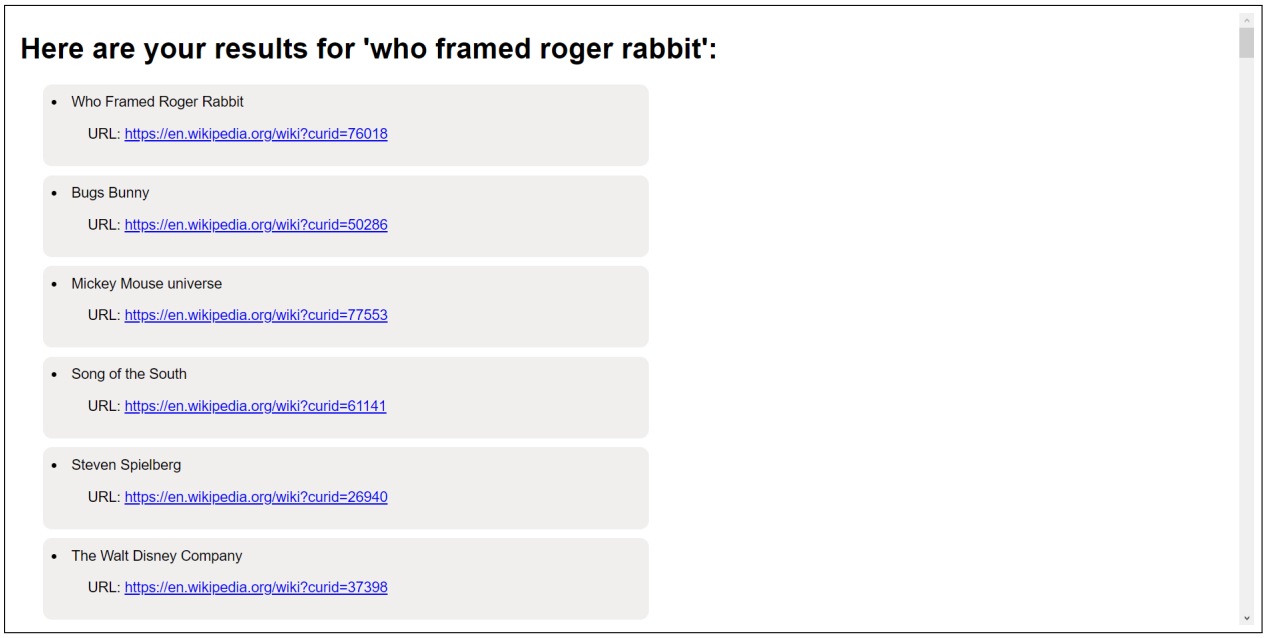
После обработки запроса выводим получившийся ответ.

Ниже приведены несколько запросов и результаты по ним.

Запрос	Время (в ms)	Количество найденных файлов
moscow && aviation && institute	12.1651	36
(red || yellow) && automobile	12.0109	356
arms && !legs	8.3885	2997

Внешний вид веб-сервиса:





2 Исходный код

Ниже приведен класс булева поиска, реализующий функционал обработки пользовательских запросов и записи результатов в файл:

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

#include <stack>
#include <set>
#include <algorithm>
#include <cmath>

#define AND L'&'
#define OR L'|'
#define NOT L'!'
#define LEFT_BRACKET L'('
#define RIGHT_BRACKET L')'
#define SPACE L' '


class Query
{
public:
    void GetIndex(std::string &inputFile);
    void ParseQueries(std::string &outputFile);
    void ParseQueriesFromFile(std::string &inputFile, std::string &outputFile);

private:
    Index index;
    std::stack<std::shared_ptr<std::vector<uint32_t>>> operands;
    std::stack<wchar_t> operations;

    const std::vector<uint32_t> &GetDocIndices(std::wstring &word);

    void ProcessingQuery(std::wstring &query);

    bool IsOperation(wchar_t op);
    uint32_t Priority(wchar_t op);

    void ExecuteOperation(wchar_t op);
    void Union();
    void Intersection();
    void Negation();
};
```

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

void Query::GetIndex(std::string &inputFile)
```

```
{
    index.Load(inputFile);
}

void Query::ParseQueries(std::string &outputFile)
{
    std::wstring query;
    bool isFuzzy = false;
    while (std::getline(std::wcin, query))
    {
        if (!query.length())
        {
            break;
        }

        std::wofstream wFileOut(outputFile.c_str());

        isFuzzy = IsFuzzy(query);
        auto start = std::chrono::high_resolution_clock::now();
        if (isFuzzy)
        {
            ProcessingFuzzyQuery(query);
        }
        else
        {
            ProcessingQuery(query);
        }
        std::shared_ptr<std::vector<uint32_t>> result_ptr = operands.top();
        operands.pop();
        if (isFuzzy)
        {
            Ranking(*result_ptr, query);
        }
        auto end = std::chrono::high_resolution_clock::now();

        std::cout << "Found " << (*result_ptr).size() << " result(s) in "
                  << std::chrono::duration<double, std::milli>(end - start).count() << "
                      ms\n";

        wFileOut << (*result_ptr).size() << L'\n';
        for (const auto &i : (*result_ptr))
        {
            wFileOut << index.docIndex[i].title << L' ' << index.docIndex[i].url << L'\
                n';
        }
        wFileOut.close();
    }
}

```

```cpp
void Query::ParseQueriesFromFile(std::string &inputFile, std::string &outputFile)
{
    std::wifstream wFileIn(inputFile.c_str());
    std::wofstream wFileOut(outputFile.c_str());

    std::wstring query;
    bool isFuzzy = false;
    while (std::getline(wFileIn, query))
    {
        if (!query.length())
        {
            break;
        }

        isFuzzy = IsFuzzy(query);
        auto start = std::chrono::high_resolution_clock::now();
        if (isFuzzy)
        {
            ProcessingFuzzyQuery(query);
        }
        else
        {
            ProcessingQuery(query);
        }
        std::shared_ptr<std::vector<uint32_t>> result_ptr = operands.top();
        operands.pop();
        if (isFuzzy)
        {
            Ranking(*result_ptr, query);
        }
        auto end = std::chrono::high_resolution_clock::now();

        std::cout << "Found " << (*result_ptr).size() << " result(s) in "
                  << std::chrono::duration<double, std::milli>(end - start).count() << "
                      ms\n";

        wFileOut << (*result_ptr).size() << L'\n';
        for (const auto &i : (*result_ptr))
        {
            wFileOut << index.docIndex[i].title << L' ' << index.docIndex[i].url << L'\
                n';
        }
    }
    wFileIn.close();
    wFileOut.close();
}

const std::vector<uint32_t> &Query::GetDocIndices(std::wstring &word)
{
```

```
 99      static const std::vector<uint32_t> empty(0);
100
101      auto it = index.invertedIndex.find(word);
102      if (it != index.invertedIndex.end())
103      {
104          return it->second;
105      }
106      else
107      {
108          return empty;
109      }
110  }
111
112  void Query::ProcessingQuery(std::wstring &query)
113  {
114      std::wstring word;
115      std::wstring quote;
116      for (size_t i = 0; i < query.length(); ++i)
117      {
118          wchar_t c = query[i];
119
120          if (query.substr(i, 4) == L" && ")
121          {
122              c = AND;
123              i += 3;
124          }
125          else if (query.substr(i, 4) == L" || ")
126          {
127              c = OR;
128              i += 3;
129          }
130          else if (c == SPACE)
131          {
132              c = AND;
133          }
134
135          if (c == QUOTE)
136          {
137              do
138              {
139                  i++;
140                  c = query[i];
141                  quote += towlower(c);
142              } while (query[i + 1] != QUOTE);
143              i++;
144              ProcessingQuote(quote);
145              quote.clear();
146              continue;
147          }
```

```
148
149         if ((IsOperation(c) || c == LEFT_BRACKET || c == RIGHT_BRACKET) && word.length
                ())
150         {
151             auto postings = std::make_shared<std::vector<uint32_t>>(GetDocIndices(word)
                    );
152             operands.push(postings);
153             word.clear();
154         }
155
156         if (c == LEFT_BRACKET)
157         {
158             operations.push(LEFT_BRACKET);
159         }
160         else if (c == RIGHT_BRACKET)
161         {
162             while (operations.top() != LEFT_BRACKET)
163             {
164                 ExecuteOperation(operations.top());
165                 operations.pop();
166             }
167             operations.pop();
168         }
169         else if (IsOperation(c))
170         {
171             while (!operations.empty() && (((c != NOT) && (Priority(operations.top())
                        >= Priority(c))) ||
172                                            ((c == NOT) && (Priority(operations.top()) >
                                                Priority(c)))))
173             {
174                 ExecuteOperation(operations.top());
175                 operations.pop();
176             }
177             operations.push(c);
178         }
179         else
180         {
181             word += towlower(c);
182         }
183     }
184
185     if (word.length())
186     {
187         auto postings = std::make_shared<std::vector<uint32_t>>(GetDocIndices(word));
188         operands.push(postings);
189     }
190
191     while (!operations.empty())
192     {
```

```
193         wchar_t op = operations.top();
194         ExecuteOperation(op);
195         operations.pop();
196     }
197 }
198 
199 bool Query::IsOperation(wchar_t op)
200 {
201     return (op == AND) || (op == OR) || (op == NOT);
202 }
203 
204 uint32_t Query::Priority(wchar_t op)
205 {
206     switch (op)
207     {
208     case OR:
209         return 1;
210     case AND:
211         return 2;
212     case NOT:
213         return 3;
214     default:
215         return 0;
216     }
217 }
218 
219 void Query::ExecuteOperation(wchar_t op)
220 {
221     switch (op)
222     {
223     case OR:
224         Union();
225         break;
226     case AND:
227         Intersection();
228         break;
229     case NOT:
230         Negation();
231         break;
232     default:
233         break;
234     }
235 }
236 
237 void Query::Union()
238 {
239     std::vector<uint32_t> result;
240     std::shared_ptr<std::vector<uint32_t>> postings1_ptr = operands.top();
241     operands.pop();
```

```
242     std::shared_ptr<std::vector<uint32_t>> postings2_ptr = operands.top();
243     operands.pop();
244     std::set_union((*postings1_ptr).begin(), (*postings1_ptr).end(),
245                    (*postings2_ptr).begin(), (*postings2_ptr).end(),
246                    std::back_inserter(result));
247     auto result_ptr = std::make_shared<std::vector<uint32_t>>(result);
248     operands.push(result_ptr);
249 }
250
251 void Query::Intersection()
252 {
253     std::vector<uint32_t> result;
254     std::shared_ptr<std::vector<uint32_t>> postings1_ptr = operands.top();
255     operands.pop();
256     std::shared_ptr<std::vector<uint32_t>> postings2_ptr = operands.top();
257     operands.pop();
258     std::set_intersection((*postings1_ptr).begin(), (*postings1_ptr).end(),
259                           (*postings2_ptr).begin(), (*postings2_ptr).end(),
260                           std::back_inserter(result));
261     auto result_ptr = std::make_shared<std::vector<uint32_t>>(result);
262     operands.push(result_ptr);
263 }
264
265 void Query::Negation()
266 {
267     std::vector<uint32_t> result;
268     std::shared_ptr<std::vector<uint32_t>> postings_ptr = operands.top();
269     operands.pop();
270     size_t j = 0;
271     for (uint32_t i = 0; i < index.docTotal; ++i)
272     {
273         if (j == (*postings_ptr).size() || i < (*postings_ptr)[j])
274         {
275             result.push_back(i);
276         }
277         else if (i == (*postings_ptr)[j])
278         {
279             ++j;
280         }
281     }
282     auto result_ptr = std::make_shared<std::vector<uint32_t>>(result);
283     operands.push(result_ptr);
284 }
```

3 Выводы

Выполнив четвертую лабораторную работу по курсу «Информационный поиск», я посторил свой булев поиск, который обрабатывает выражения в скобках и реализует все три оператора: «&&», «||», «!». Кроме того был реализован веб-сервис, напсанный при помощи HTML и Flask.

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.))