

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: В. М. Филиппов  
Преподаватель: А. А. Кухтичев  
Группа: М8О-410Б  
Дата:  
Оценка:  
Подпись:

Москва, 2025

## Лабораторная работа №5 «Булев поиск»

Необходимо реализовать алгоритм булева поиска по корпусу документов при условии, что индекс сдмплен на диск.

# 1 Описание

При выполнении данной лабораторной работы, я реализовал алгоритм булева поиска по документам.

## 1 Булев поиск

Булев поиск — это модель информационного поиска, основанная на формальной логике. В этой модели запрос пользователя представляет собой логическое выражение, а результатом является множество документов, которые либо строго соответствуют условию, либо нет.

Булев поиск строится на трех базовых операторах алгебры логики:

- AND (И): Находит документы, в которых присутствуют оба слова.
- OR (ИЛИ): Находит документы, в которых есть хотя бы одно из слов (или оба сразу).
- NOT (НЕ): Исключает документы, содержащие определенное слово.

Алгоритм поиска работает следующим образом: для каждого токена из запроса мы получаем список документов. Далее преобразуем наш запрос в обратную польскую нотацию при помощи алгоритма сортировочной станции Дейкстры. Затем при помощи стэка операндов и операторов вычисляем предикат. Берем два операнда и один оператор с вершины стэка и выполняем операцию над множествами, кладем результат в стэк. Операции над множествами реализованы через два указателя, что позволяет выполнять пересечение и объединение списков за линейное время относительно их длины.

Если между двумя токенами нет никакого оператора, считается, что между ними стоит оператор AND

Из недостатков можно выделить то, что мы никак не ранжируем результаты поиска.

## 2 Исходный код

Центральным узлом системы является абстрактный класс `ISearcher`, реализующий паттерн "шаблонный метод". Этот паттерн позволяет зафиксировать высокоуровневый алгоритм поиска, делегируя детали реализации конкретных шагов классам-наследникам.

Основные этапы алгоритма поиска документа:

1. Лексический анализ (`parseQuery`). Строка запроса разбивается на токены (слова и операторы). Это позволяет абстрагировать логику обработки текста (например, удаление стоп-слов или стемминг) от логики поиска.
2. Трансформация в ОПН (`sortingStation`). Запрос преобразуется в обратную польскую нотацию. Этот этап необходим для корректной обработки приоритетов логических операций и управления вложенными скобками.
3. Вычисление (`evaluate`). Происходит непосредственное выполнение булевых операций над списками вхождений, полученными из источника данных. Результатом этого этапа является «сырой» список идентификаторов документов, которые соответствуют логическому условию запроса.
4. Пост-процессинг и ранжирование (`processResults`). Это ключевая точка расширения. Метод принимает список найденных документов и исходные токены запроса, возвращая финальную выдачу с весами релевантности.

## 1 Примеры запросов и результаты выдачи

В среднем запрос занимает около 0.01 секунды.

```
1 |
2 | Downloaded: 305134 docs
3 | Finished. Total docs: 305135
4 | Total indexate time: 471.738
5 | Speed indexate: 5776.33
6 | Total downloaded bytes: 10182.4 MB
7 | Total indexed bytes: 2661.05 MB
8 | : 809.048
9 | !
10| 0
11| : manchester united
12| Tokens searching: [manchest, unit, ]
13| 5
14| https://theguardian.com/football/2025/dec/17/premier-league-teams-africa-cup-of-nations-sunderland-morocco-chelsea-arsenal-aston-villa 0
```

15 <https://theguardian.com/football/2025/dec/17/the-football-daily-christmas-awards-2025>  
0

16 <https://theguardian.com/football/2025/dec/17/the-knowledge-football-match-wham-watching-wrote-last-christmas> 0

17 <https://theguardian.com/football/2025/dec/14/ruben-amorim-kobbie-mainoo-loan-manchester-united-bournemouth-premier-league> 0

18 <https://theguardian.com/football/2025/dec/14/brentford-leeds-premier-league-match-report> 0

19 : 0.020981

20 : 6065

21 : epstein

22 Tokens searching: [epstein, ]

23 5

24 <https://theguardian.com/football/2025/dec/12/schmaltz-theatre-and-sharp-teeth-wrexham-reveal-the-hard-truth-about-football> 0

25 <https://theguardian.com/football/2025/dec/08/joey-barton-gets-suspended-prison-sentence-for-offensive-social-media-posts> 0

26 [https://en.wikipedia.org/wiki/Gatorade\\_shower](https://en.wikipedia.org/wiki/Gatorade_shower) 0

27 [https://en.wikipedia.org/wiki/Sports\\_analytics](https://en.wikipedia.org/wiki/Sports_analytics) 0

28 [https://en.wikipedia.org/wiki/Mechanics\\_of\\_Oscar\\_Pistorius%27s\\_running\\_blades](https://en.wikipedia.org/wiki/Mechanics_of_Oscar_Pistorius%27s_running_blades) 0

29 : 0.00997354

30 : 1479

31 : best football player

32 Tokens searching: [best, footbal, play, ]

33 5

34 <https://theguardian.com/football/2025/dec/17/the-football-daily-christmas-awards-2025>  
0

35 <https://theguardian.com/football/2025/dec/17/the-knowledge-football-match-wham-watching-wrote-last-christmas> 0

36 <https://theguardian.com/sport/2025/dec/15/philip-rivers-indianapolis-colts-nfl-return>  
0

37 <https://theguardian.com/football/2025/dec/15/how-the-guardian-ranked-the-100-best-male-footballers-in-the-world-2025> 0

38 <https://theguardian.com/sport/2025/dec/14/all> 0

39 : 0.0159811

40 : 6711

41 : (manchester & united) | ronaldo

42 Tokens searching: [manchest, unit, ronaldo, ]

43 5

44 <https://theguardian.com/football/2025/dec/17/premier-league-teams-africa-cup-of-nations-sunderland-morocco-chelsea-arsenal-aston-villa> 0

45 <https://theguardian.com/football/2025/dec/17/the-football-daily-christmas-awards-2025>  
0

46 <https://theguardian.com/football/2025/dec/17/brendan-rodgers-saudi-arabia-pro-league-al-qadsiah-aramco> 0

47 <https://theguardian.com/football/2025/dec/17/the-knowledge-football-match-wham-watching-wrote-last-christmas> 0

48 <https://theguardian.com/football/2025/dec/14/ruben-amorim-kobbie-mainoo-loan-manchester-united-bournemouth-premier-league> 0

```

49 : 0.0150467
50 : 6556
51 : !messi | ronaldo & bale
52 Tokens searching: [messi, ronaldo, bal, ]
53 5
54 https://theguardian.com/sport/2025/dec/17/nba-cup-takeaways-spurs-knicks-victor-
    wembanyama 0
55 https://theguardian.com/sport/2025/dec/17/all 0
56 https://theguardian.com/football/2025/dec/17/premier-league-teams-africa-cup-of-
    nations-sunderland-morocco-chelsea-arsenal-aston-villa 0
57 https://theguardian.com/football/2025/dec/17/celtic-chair-peter-lawwell-to-stand-down-
    after-intolerable-abuse-from-fans 0
58 https://theguardian.com/football/2025/dec/17/fifa-50-per-cent-increase-2026-world-cup-
    prize-money-50m-dollars-winners 0
59 : 0.0415131
60 : 303572
61 :

```

```

1  #pragma once
2  #include <cstdlib>
3  #include <cstring>
4  #include <vector>
5
6  #include "index.h"
7  #include "tokenizer.h"
8
9  std::vector<int> intersect_lists(const std::vector<int>& l1, const std::vector<int>&
    l2);
10 std::vector<int> union_lists(const std::vector<int>& l1, const std::vector<int>& l2);
11 std::vector<int> not_list(const std::vector<int>& l, int total_docs);
12
13 class ISearcher {
14 protected:
15     std::shared_ptr<Tokenizer> tokenizer;
16     std::shared_ptr<IIndexSource> source;
17
18 public:
19     ISearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<Tokenizer> tok);
20     virtual ~ISearcher() = default;
21     virtual std::vector<std::pair<std::string, double>> findDocument(const std::string&
        query);
22
23 protected:
24     int getPriority(const std::string& op);
25     bool isOperator(const std::string& token);
26     std::vector<int> evaluate(const std::vector<std::string>& tokens, int total_docs);
27
28     std::vector<int> fetchDocIds(const std::string& term);
29     virtual std::vector<std::pair<std::string, double>> processResults(const std:::
        vector<int>& docIds,

```

```

30         const std::vector<std
           ::string>& terms) =
           0;
31
32     std::vector<std::string> parseQuery(const std::string& query);
33     std::vector<std::string> sortingStation(const std::vector<std::string>& tokens);
34 };
35
36 class BinarySearcher : public ISearcher {
37 public:
38     BinarySearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<Tokenizer> tok);
39
40 private:
41     std::vector<std::pair<std::string, double>> processResults(const std::vector<int>&
           docIds,
42
           const std::vector<std::string
           >& terms) override;
43 };

1  #include "searcher.h"
2
3  #include <fcntl.h>
4  #include <sys/mman.h>
5  #include <sys/stat.h>
6  #include <unistd.h>
7
8  #include <algorithm>
9  #include <cstring>
10 #include <iostream>
11 #include <string>
12
13 #include "tokenizer.h"
14
15 std::vector<int> intersect_lists(const std::vector<int>& l1, const std::vector<int>&
16     l2) {
17     std::vector<int> res;
18     res.reserve(std::min(l1.size(), l2.size()));
19     auto i1 = l1.begin(), i2 = l2.begin();
20     while (i1 != l1.end() && i2 != l2.end()) {
21         if (*i1 < *i2)
22             i1++;
23         else if (*i2 < *i1)
24             i2++;
25         else {
26             res.push_back(*i1);
27             i1++;
28             i2++;
29         }
30     }
31     return res;

```

```

31 }
32
33 std::vector<int> union_lists(const std::vector<int>& l1, const std::vector<int>& l2) {
34     std::vector<int> res;
35     res.reserve(l1.size() + l2.size());
36     auto i1 = l1.begin(), i2 = l2.begin();
37     while (i1 != l1.end() || i2 != l2.end()) {
38         if (i1 == l1.end()) {
39             res.push_back(*i2++);
40             continue;
41         }
42         if (i2 == l2.end()) {
43             res.push_back(*i1++);
44             continue;
45         }
46
47         if (*i1 < *i2)
48             res.push_back(*i1++);
49         else if (*i2 < *i1)
50             res.push_back(*i2++);
51         else {
52             res.push_back(*i1);
53             i1++;
54             i2++;
55         }
56     }
57     return res;
58 }
59
60 std::vector<int> not_list(const std::vector<int>& l, int total_docs) {
61     std::vector<int> res;
62     res.reserve(total_docs - l.size());
63     int current_doc = 0;
64     auto it = l.begin();
65
66     while (current_doc < total_docs) {
67         if (it != l.end() && *it == current_doc) {
68             it++;
69         } else {
70             res.push_back(current_doc);
71         }
72         current_doc++;
73     }
74     return res;
75 }
76
77 int ISearcher::getPriority(const std::string& op) {
78     if (op == "!") return 3;
79     if (op == "&") return 2;

```

```

80     if (op == "|") return 1;
81     return 0;
82 }
83
84 bool ISearcher::isOperator(const std::string& token) {
85     return token == "!" || token == "&" || token == "|" || token == "(" || token == ")"
86         ;
87 }
88 std::vector<std::pair<std::string, double>> ISearcher::findDocument(const std::string&
89     query) {
90     auto tokens = parseQuery(query);
91
92     std::vector<std::string> queryTerms;
93     for (const auto& token : tokens) {
94         if (!isOperator(token)) {
95             queryTerms.push_back(token);
96         }
97     }
98
99     std::cout << "Tokens searching: [";
100    for (const auto& token : queryTerms) {
101        std::cout << token << ", ";
102    }
103    std::cout << "]\n";
104
105    auto rpn = sortingStation(tokens);
106
107    auto docIds = evaluate(rpn, source->getTotalDocs());
108
109    if (docIds.empty()) return {};
110
111    return processResults(docIds, queryTerms);
112 }
113 std::vector<int> ISearcher::evaluate(const std::vector<std::string>& rpn, int
114     total_docs) {
115     std::vector<std::vector<int>> stack;
116     for (const auto& token : rpn) {
117         if (!isOperator(token)) {
118             stack.push_back(fetchDocIds(token));
119         } else {
120             if (token == "!") {
121                 if (stack.empty()) continue;
122                 auto op1 = stack.back();
123                 stack.pop_back();
124                 stack.push_back(not_list(op1, total_docs));
125             } else {
126                 if (stack.size() < 2) continue;

```

```

126         auto right = stack.back();
127         stack.pop_back();
128         auto left = stack.back();
129         stack.pop_back();
130         if (token == "&")
131             stack.push_back(intersect_lists(left, right));
132         else if (token == "|")
133             stack.push_back(union_lists(left, right));
134     }
135 }
136 }
137 return stack.empty() ? std::vector<int>{} : stack.back();
138 }
139
140 std::vector<std::string> ISearcher::parseQuery(const std::string& query) {
141     std::vector<std::string> rawTokens = tokenizer->getRawTokens(query);
142
143     std::vector<std::string> processedTokens;
144
145     auto addTokenWithImplicitAnd = [&](const std::string& newToken) {
146         if (!processedTokens.empty()) {
147             const std::string& last = processedTokens.back();
148             if ((!isOperator(last) || last == ")") && (!isOperator(newToken) ||
149                 newToken == "(" || newToken == "!")) {
150                 processedTokens.push_back("&");
151             }
152             processedTokens.push_back(newToken);
153         };
154
155         for (const auto& t : rawTokens) {
156             if (isOperator(t)) {
157                 addTokenWithImplicitAnd(t);
158             } else {
159                 tokenizer->tokenize(t);
160                 auto subtokens = tokenizer->getTokens();
161                 for (const std::string& sub : subtokens) {
162                     addTokenWithImplicitAnd(sub);
163                 }
164             }
165         }
166         return processedTokens;
167     }
168
169     ISearcher::ISearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<Tokenizer> tok
170         )
171         : source(std::move(src)), tokenizer(std::move(tok)) {}
172
173     std::vector<std::string> ISearcher::sortingStation(const std::vector<std::string>&

```

```

tokens) {
173     std::vector<std::string> outputQueue;
174     std::vector<std::string> operatorStack;
175
176     for (const auto& token : tokens) {
177         if (!isOperator(token)) {
178             outputQueue.push_back(token);
179         } else if (token == "(") {
180             operatorStack.push_back("(");
181         } else if (token == ")") {
182             while (!operatorStack.empty() && operatorStack.back() != "(") {
183                 outputQueue.push_back(operatorStack.back());
184                 operatorStack.pop_back();
185             }
186             if (!operatorStack.empty()) operatorStack.pop_back();
187         } else {
188             while (!operatorStack.empty() && operatorStack.back() != "(" &&
189                 getPriority(operatorStack.back()) >= getPriority(token)) {
190                 outputQueue.push_back(operatorStack.back());
191                 operatorStack.pop_back();
192             }
193             operatorStack.push_back(token);
194         }
195     }
196
197     while (!operatorStack.empty()) {
198         outputQueue.push_back(operatorStack.back());
199         operatorStack.pop_back();
200     }
201     return outputQueue;
202 }
203
204 std::vector<int> ISearcher::fetchDocIds(const std::string& term) {
205     std::vector<TermInfo> postings = source->getPostings(term);
206
207     std::vector<int> ids;
208     ids.reserve(postings.size());
209
210     for (const auto& entry : postings) {
211         ids.push_back(entry.doc_id);
212     }
213     return ids;
214 }
215
216 BinarySearcher::BinarySearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<
    Tokenizer> tok) : ISearcher(src, tok) {}
217
218 std::vector<std::pair<std::string, double>> BinarySearcher::processResults(const std::
    vector<int>& docIds,

```

```

219 |
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |
228 |
229 |
230 |

```

```

const std::vector<
    std::string>&
    terms) {

std::vector<std::pair<std::string, double>> result_urls;
result_urls.reserve(docIds.size());

for (int id : docIds) {
    std::string url = source->getUrl(id);
    if (!url.empty()) {
        result_urls.push_back({url, 0.});
    }
}
return result_urls;
}

```

### 3 Выводы

За время выполнения этой лабораторной работы, я узнал что такое булев поиск, реализовал алгоритмы пересечения и объединения множеств методом двух указателей. Это обеспечило линейную сложность обработки списков вхождений, что критически важно для производительности системы. Вспомнил как работает алгоритм сортировочной станции Дейкстры.

## Список литературы

- [1] Маннинг К. Д., Рагхаван П., Шютце Х. *Введение в информационный поиск* : пер. с англ. — М. : ООО «И.Д. Вильямс», 2011. — 528 с.
- [2] Польская нотация или как легко распарсить алгебраическое выражение [Электронный ресурс] // Хабр. — 2021. — URL: <https://habr.com/ru/articles/596925/> (дата обращения: 17.12.2025).
- [3] Boolean search для чайников и кофейников [Электронный ресурс] // Хабр. — 2023. — URL: <https://habr.com/ru/articles/716968/> (дата обращения: 17.12.2025).