

**Московский авиационный институт
(национальный исследовательский университет)**

**Факультет информационных технологий и прикладной
математики**

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: В. Р. Орусский
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4

Задача: Требуется разработать программу, осуществляющую ввод паттернов и текста, после чего для каждого вхождения паттерна в текст вывести информацию в следующем формате: номер строки и номер слова в строке, с которого начинается найденный образец.

Вариант алгоритма: Поиск большого количества образцов при помощи алгоритма Ахо-Корасик..

Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые) .

1 Описание

Основная идея алгоритма Ахо-Корасика - использования trie структуры данных, в которую мы помещаем ноды, по которым мы переходим в зависимости от того, какой элемент рассматривается в тексте. Для того чтобы эффективно использовать алгоритм, мы также строим связи неудач и связи выхода. Первые отвечают за то, чтобы мы откатились не к самому началу поиска, а к какой-то уже найденной части другого паттерна. Вторая связь отвечает за то, чтобы не потерять более мелкие паттерны, которые являются подстроками других паттернов. Время выполнения: $O(n + k + m)$

Память: $O(n)$

Где, n - суммарная длина паттернов m - длина текста k - количество появлений паттернов в тексте

2 Исходный код

Для реализации этого алгоритма, будем использовать trie дерево, в которое будем помещать ноды, с ссылками на них по словам. После чего посчитает все ссылки выхода и ошибок, а дальше при считывании текста будем передавать информацию о позиции взятой строки (чтобы вывести подробную информацию о нахождении подстроки).

```
1 struct Vertex {
2     std::map<std::string, Vertex *> links;
3     Vertex *sufflink = nullptr;
4     std::vector<int> success; //pattern id
5
6     bool CheckKey(const std::string &key) {
7         return links.find(key) != links.end();
8     }
9 };
10
11 using TrieNode = std::pair<std::string, Vertex *>;
12
13 class AhoTrie {
14     Vertex *root;
15     int patternSize;
16     int pieces;
17
18 public:
19     std::vector<int> pieceIndex;
20
21     AhoTrie() : patternSize(0), pieces(0) {
22         root = new Vertex;
23         root->sufflink = root;
24     }
25
26     Vertex *Move(Vertex *item, std::string &letter) {
27         if (item->CheckKey(letter)) item = item->links.at(letter);
28         else {
29             if (item == root)
30                 item = root;
31             else
32                 item = Move(item->sufflink, letter);
33         }
34         return item;
35     }
36
37     static Vertex *GetNextNode(Vertex *item, std::string letter) {
38         if (!item)
39             return nullptr;
40         if (item->CheckKey(letter)) item = item->links.at(letter);
41         else item = nullptr;
42         return item;
43 }
```

```

43 }
44
45 void Linkate() {
46     Vertex *node, *child;
47     std::queue<Vertex *> queue;
48     queue.push(root);
49
50     while (!queue.empty()) {
51         node = queue.front();
52         queue.pop();
53         for (auto &ChildPair : node->links) {
54             child = ChildPair.second;
55             queue.push(child);
56             child->sufflink = FindSuffixLinks(child, node, ChildPair.first);
57
58             child->success.insert(child->success.end(),
59                                   child->sufflink->success.begin(),
60                                   child->sufflink->success.end());
61             child->success.shrink_to_fit();
62         }
63     }
64 }
65
66 Vertex *FindSuffixLinks(Vertex *child, Vertex *parent, const std::string &letter) {
67     Vertex *linkup = parent->sufflink, *check;
68
69     while (true) {
70         check = GetNextNode(linkup, letter);
71         if (check) return (check != child) ? check : root;
72         if (linkup == root) return root;
73
74         linkup = linkup->sufflink;
75     }
76 }
77
78 void AddString(const std::vector<std::string> &str) {
79
80     Vertex *bohr = root, *next;
81     for (auto &word : str) {
82         next = GetNextNode(bohr, word);
83         if (!next) {
84             next = new Vertex;
85             next->sufflink = root;
86             bohr->links.insert(TrieNode(word, next));
87         }
88         bohr = next;
89     }
90     bohr->success.push_back(pieces);
91     ++pieces;

```

```

92 }
93
94 void Search(std::vector<std::string> &text,
95             std::vector<std::pair<int, int> > &placeInfo) {
96     Linkate();
97
98     size_t textLen = text.size();
99     Vertex *node = root;
100    int successIdx;
101    int wordNumber;
102
103   for (wordNumber = 0; wordNumber <= textLen; ++wordNumber) {
104
105     for (auto &i : node->success) {
106         successIdx = wordNumber - pieceIndex[i];
107         std::cout << placeInfo[successIdx].first << ", "
108             << placeInfo[successIdx].second << ", "
109             << i + 1 << '\n';
110
111     }
112     if (wordNumber < textLen) node = Move(node, text[wordNumber]);
113   }
114 }
115
116 };

```

3 Консоль

```
slava@DESKTOP-9JJF73M MINGW64 /d/Slavik/Coding/C++/DiskretAnalysis/Lab#4 (master)
$ g++ main.cpp
```

```
slava@DESKTOP-9JJF73M MINGW64 /d/Slavik/Coding/C++/DiskretAnalysis/Lab#4 (master)
$ ./a.exe <tests/02.t
3,1,1
62,1,1
118,1,1
180,4,1
283,1,1
384,3,1
```

4 Тест производительности

Тест производительности представляет из себя следующее: поиск всех вхождений паттернов в текст Вариант алгоритма: Поиск большого количества образцов при помощи алгоритма Ахо-Корасик. Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые).

Сравнение происходит между поиска подстроки «search» из std. Для тестов будем использовать файлы с различным кол-вом паттернов и строк в тексте.

```
slava@DESKTOP-9JJF73M MINGW64 /d/Slavik/Coding/C++/DiskretAnalysis/Lab#4 (master)
$ ./a.exe <./tests/01.t
Aho search time: 151619
STL search time: 398067
```

```
slava@DESKTOP-9JJF73M MINGW64 /d/Slavik/Coding/C++/DiskretAnalysis/Lab#4 (master)
$ ./a.exe <./tests/01.t
Aho search time: 137632
STL search time: 321141
```

Видно, что Ахо-Корасик работает быстрее.. Поскольку сложность этого алгоритма лучше, чем сложность «std::search».

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я изучил работу многих функций из STL, таких как «`std::transform`», «`vector.shrink_to_fit`», вспомнил работу потоков ввода/вывода. Подробнее изучил работу алгоритма Ахо-Корасик и считаю его одним из самых эффективных для множественного поиска подстрок в тексте.

Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))
- [2] *Aho-Korasick — Википедия.*
URL: https://ru.wikipedia.org/wiki/Алгоритм_Ахо_--_Корасик.
- [3] *Aho-Korasick. E-MAXX.*
URL: https://e-maxx.ru/algo/aho_korasick.