# 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: Я. С. Поскряков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

**Москва, 2019**

# Лабораторная работа №4

**Задача:** Требуется разработать программу, осуществляющую ввод шаблона и поиск его в последующем тексте используя заданный вариантом алгоритм.

**Вариант алгоритма поиска:** Поиск одного образца-маски: в образце может встречаться «джокер», равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

# 1 Описание

Поиск образца с джокерами преобразуется к множественному поиску с помощью алгоритма Ахо-Корасик с некоторыми нюансами. В дерево ключей нужно поместить не один образец, а все подстроки образца, не содержащие джокеры. Затем построить для дерева связи неудач для каждой вершины, обойдя дерево в ширину. Связь неудач связывает в дереве две вершины, первая из которых это вершина, полученная после прохода по тексту и совпадении символов текста с некоторым образцом до следующей вершины, в которой найдено несовпадение, а вторая – это вершина соответствующая символу другого образца, перфикс которого является максимальным суффиксом уже пройденного пути. Это существенно ускоряет поиск. Препроцессинг для построения связей неудач выполняется за O(n), где n – суммарная длина всех подстрок образца без джокеров. Помимо всего прочего при построении дерева нужно найти длины подстрок образца, не содержащих джокеров, и их позиции начала в образце. Это понадобится в дальнейшем при поиске образца в тексте, для проверки относительного месторасположения подстрок.

Непосредственно при поиске нужно пройти по тексту и найти для каждого подобразца начальные позиции вхождений подобразцов в текст.

Задача сводится к тому, чтобы заполнить 2 вектора – образец и текст, на основе образца построить дерево ключей, создать связи неудач, а затем совершить поиск единственным проходом по тексту.

# 2 Исходный код

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <sstream>
#include <queue>
#include <fstream>

using std::string;
using std::vector;
using std::map;
using std::queue;
using std::fstream;
using std::cout;
using std::cin;
using std::endl;

class TNode {
public:
    unsigned long long key_number;
    TNode * parent;
    map <unsigned long long, TNode *> child;
    TNode * link;
    bool is_list;
    int place_under_string;
    int num_mask;
    vector <int> num_in_pattern;
    int depth;
    TNode() {

    }
    TNode(unsigned long long n_number, TNode * n_parent, int n_depth) {
        key_number = n_number;
        parent = n_parent;
        link = nullptr;
        is_list = false;
        place_under_string = -1;
        num_mask = 0;
        depth = n_depth;
    }
    ~TNode() { }
};

class TTrie {
public:
    TNode * root;
    vector <string> patterns;
```

```
48      vector <int> joker_place;
49      int num_pattern_without_Joker;
50      int num_Joker;
51
52      const string Joker = "?";
53      TTrie() {
54          root = new TNode(0, root, 0);
55          root->link = root;
56          num_Joker = 0;
57          num_pattern_without_Joker = 0;
58      }
59      void Make();
60      void Make_Links();
61      void Find_Occurrence(vector<std::pair<int, int> > answer, vector <unsigned long
            long> check);
62  };
63
64  void TTrie::Make() {
65      map <unsigned long long, TNode *>::iterator it;
66      TNode * now_node = root;
67      int now_length = 0;
68      int now_depth = 0;
69      int index_begin = 0;
70      int i = 0;
71      for (; i < this->patterns.size(); i++) {
72
73          if (patterns[i] == Joker) {
74              num_Joker++;
75              now_node->num_in_pattern.push_back(index_begin);
76              index_begin = i + 1;
77              now_node->is_list = true;
78              now_node = root;
79              now_depth = 0;
80              //num_pattern_without_Joker++;
81          }
82          else {
83              it = now_node->child.find(stoul(patterns[i]));
84              now_depth++;
85              if (it == now_node->child.end()) {
86                  TNode * new_node = new TNode(stoul(patterns[i]), now_node, now_depth);
87                  now_node->child[stoul(patterns[i])] = new_node;
88                  now_node = new_node;
89              }
90              else {
91                  now_node = it->second;
92              }
93              now_length++;
94          }
95      }
```

```
96      //num_pattern_without_Joker++;
97      now_node->is_list = true;
98      now_node->num_in_pattern.push_back(index_begin);
99      Make_Links();
100 }
101
102 void TTrie::Make_Links() {
103     TNode *now_link, *node, *now_node;
104     unsigned long long number;
105     queue <TNode *> node_queue;
106     now_node = root;
107     node_queue.push(now_node);
108
109     while (!node_queue.empty()) {
110         node = node_queue.front();
111         node_queue.pop();
112
113         for (auto i : node->child) {
114
115             now_node = i.second;
116             if (now_node->is_list) {
117                 for (int k = 0; k < now_node->num_in_pattern.size(); k++)
118                     num_pattern_without_Joker++;
119             }
120             number = i.first;
121             node_queue.push(now_node);
122             now_link = now_node->parent->link;
123             if (now_node->parent == root) {
124                 now_node->link = root;
125             }
126             else {
127                 while (1) {
128                     std::map<unsigned long long, TNode *>::iterator it = now_link->child
                            .find(number);
129                     if (it != now_link->child.end()) {
130                         now_link = it->second;
131                         break;
132                     }
133                     else {
134                         if (now_link == root)
135                             break;
136                         else
137                             now_link = now_link->link;
138                     }
139                 }
140                 now_node->link = now_link;
141             }
142         }
143
```

```
144     }
145 }
146 
147 void TTrie::Find_Occurrence(vector<std::pair<int, int> > answer, vector <unsigned long
        long> check) {
148 
149     map <unsigned long long, TNode *>::iterator it;
150     vector <int> pos_entering;
151 
152     int l = 0;
153     int c = 0;
154     TNode * now_node1 = root;
155     TNode * now_node2;
156 
157     pos_entering.resize(check.size(), 0);
158 
159     do {
160 
161         it = now_node1->child.find(check[c]);
162 
163         while (it != now_node1->child.end()) {
164             now_node2 = it->second;
165 
166             if (now_node2->is_list) {
167                 for (auto i : now_node2->num_in_pattern) {
168                     if (l - i < 0)
169                         break;
170                     pos_entering[l - i]++;
171                 }
172             }
173 
174             // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!1
175             if (now_node2->link != root) {
176                 TNode * now_link = now_node2->link;
177                 do {
178                     if (now_link->is_list) {
179                         int new_str = c - now_link->depth + 1;
180                         for (auto i : now_link->num_in_pattern) {
181                             if (new_str - i < 0)
182                                 break;
183                             pos_entering[new_str - i]++;
184                         }
185                     }
186                     now_link = now_link->link;
187                 } while (now_link != root);
188             }
189             // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
190             c++;
191             if (c >= check.size()) {
```

```
192                 break;
193             }
194             //l++;
195             now_node1 = now_node2;
196             it = now_node1->child.find(check[c]);
197         }
198         if (c >= check.size()) {
199             break;
200         }
201         if (now_node1 == root) {
202             l++;
203             c++;
204         }
205         else {
206             now_node1 = now_node1->link;
207             l = c - now_node1->depth;
208         }
209 
210     } while (c < check.size());
211 
212     if (num_pattern_without_Joker != 0) {
213         for (int j = 0; j < pos_entering.size(); j++) {
214             if (num_pattern_without_Joker == pos_entering[j]) {
215                 //
216                 cout << answer[j].first << " , " << answer[j].second << endl;
217             }
218         }
219     }
220     else {
221         for (int j = 0; j < pos_entering.size(); j++) {
222             if (num_Joker > pos_entering.size() - j)
223                 break;
224             cout << answer[j].first << " , " << answer[j].second << endl;
225         }
226     }
227 
228 }
229 
230 
231 
232 int main()
233 {
234     vector<std::pair<int, int> > answer;
235     TTrie * trie = new TTrie;
236     string enter;
237 
238     vector <unsigned long long> check;
239     string word;
240     //ifstream file("t3.txt");
```

```
241     getline(cin, enter);
242     std::stringstream patterns_string(enter);
243 
244     for (int i = 0; patterns_string >> enter; i++) {
245         trie->patterns.push_back(enter);
246     }
247 
248     trie->Make();
249 
250     int lineCnt = 1;
251     int wordCnt = 1;
252 
253     for (/*int i = 0; i < 1 */; getline(cin, enter);/* ; i++*/) {
254         //getline(cin, enter);
255         std::stringstream check_stream(enter);
256         for (; check_stream >> word;) {
257             check.push_back(stoul(word));
258             answer.push_back(std::make_pair(lineCnt, wordCnt));
259             ++wordCnt;
260         }
261         wordCnt = 1;
262         lineCnt++;
263     }
264 
265     trie->Find_Occurrence(answer, check);
266 
267     return 0;
268 }
```

# 3 Консоль

Тесты генерировались следующим кодом на языке *Python*.

```python
#!usr/bin/python

from random import *
from string import ascii_letters

def get_random_pattern_key():
    return ''.join(choice('123456789') for i in range(randint(1,3)))

def get_random_text_key():
    return ''.join(choice('123456789') for i in range(randint(1,5)))

if __name__ == "__main__":
    pattern = ''
    text = ''
    keys = []
    test_file_name = "tests/{:02d}".format( 10 )
    with open( "{0}.t".format( test_file_name ), 'w' ) as output_file:
        for x in range(5):
            check = randint(1, 100000)
            if check % 2 == 0:
                key = '?'
            else:
                key = get_random_pattern_key()
                if key not in keys:
                    keys.append(key)
            pattern = pattern + ' ' + key
        output_file.write("{0}\n".format( pattern.lstrip() ))

        for i in range(10000): # kolvo strok v file
            if i % 1000 == 0:
                print 'Generate ', i
            text = ''
            for x in range(100): # kolvo chisel v str
                check = randint(1, 100000)
                if check % 2 == 0 and len(keys) > 0:
                    key = choice(keys)
                    text = text + ' ' + key
                else:
                    key = get_random_text_key()
                    text = text + ' ' + key
            output_file.write("{0}\n".format( text.lstrip() ))
```

```
yar@ubuntu:~$ clang++ -pedantic -Wall -std=c++14 -Werror -Wno-sign-compare
-lm main.c -o main --some_long_argument=true
yar@ubuntu:~$ cat input1
```

```
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
yar@ubuntu:~$ ./main <input1
1,3
1,8
yar@ubuntu:~$ cat input2
a a b a a c a a b a a d
a a b a a d a a b a a c a a d a a b a a c a a c a a b a a c a a b a a b a a
b

yar@ubuntu:~$ ./main <input2
yar@ubuntu:~$ cat input3
aaa aa
aaa aaa aa aaa aaa aa
yar@ubuntu:~$ ./main <input3
1,2
1,5
```

# 4 Тест производительности

Для проведения теста производительности было произведено сравнение времени работы написанной программы и метода find стандартного контейнера string. Были использованы тесты с разным количеством символов.

```
50000 символов:
Ахо-Корасик:    30 ms
std::find:  30 ms

5000000 символов:
Ахо-Корасик:    1170 ms
std::find:  1200 ms

50000000 символов:
Ахо-Корасик:    9700 ms
std::find:  99000 ms
```

# 5 Выводы

Для выполнения данной лабораторной работы по курсу «Дискретный анализ», я изучил особенности работы со строками и методы их обработки для повышения эффективности дальнейшей работы с ними. Одним из алгоритмов данной работы является алгоритм Ахо-Корасик. Ахо-Корасик - алгоритм поиска подстроки, реализует поиск множества подстрок из словаря в данной строке. Данный алгоритм довольно широко распространен в системном программном обеспечении, к примеру, используется в утилите поиска grep.

# Список литературы


[1] *Ахо-Корасик - http://e-maxx.ru.*
URL: `http://e-maxx.ru/algo/aho_corasick` (дата обращения: 20.12.2018).

[2] *Ахо-Корасик — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Ахо_-Корасик` (дата обращения: 20.12.2018).

[3] *Алгоритм Ахо-Корасик — ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Ахо-Корасик` (дата обращения: 20.12.2018).