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: П. А. Мохляков
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант дерева: Поиск одного образца-маски: в образце может встречаться «джокер» (представляется символом ? — знак вопроса), 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.

Вариант алгоритма: `Последовательность букв английского алфавита длиной не более 256 символов`

Вариант значения: `Числа в диапазоне от` 0 `до` $2^{32} - 1$

1 Описание

Алогоритм Ахо-Корасик - это алшоритм поиска подстроки в строке. В отсличии от других алгоритмов, алогоритм Ахо-Корасик позволяет деталь поиск сразу несколько паттернов одноврененно, то есть за один проход. Благодаря этому скорость поиска множества паттернов намного выше, чем в других алгоритмах.

Рассмотрим данный алгоритм на примере моего исходного кода.

2 Исходный код

Данный алгоритм можно разбить на несколько основных частей:

- Создание trie из паттернов.
- Создание суффиксных ссылок.
- Поиск в тексте.

Добавление паттерна в trie происходит поэлементно. Стартовой позицией является Root, мы смотрим, есть ли у элемента ребенок в виде нашего введенного элемента. Если он существует, то мы просто переходим в него, иначе создаем, добавляем в таблицу детей и переходим. Дойдя до конца паттерна мы говорим, что это последий элемент и добавляем его позицию с учетом джокеров.

```
void Push(const std::vector<long long> &pattern, int pos){
  node_ptr cur = Root;
  for(auto elem_pattern: pattern){
    node_ptr child = cur->ChildPrt(elem_pattern);
    if(child == nullptr){
      child = new TNode(elem_pattern);
      child->Parent = cur;
      cur->Childs.insert({elem_pattern,child});
    }
    cur = child;
  }
  cur->Last = true;
  cur->Jockers.push_back(pos);
}
```

Далее происходит создание ссылок на суффиксы и концы префиксов, для этого нужно обойти trie в ширину. При поиске суффикса мы переходим в родителя, и далее перезодим по суффиксам родителя до тех пор пока не у суффикса не будет в качестве ребенка наш елемент, в таком случае это суффикс нащего элемента, иначе идем пока не дойдем до нулевого указателя, в таком случае суффикс нашего элемента это Root. Для проверки на концы префиксов мы смотрим является ли наш суффикс концом паттерна или же указывает ли он другой конец префикса.

```
void BorSuf(){
    //BFS
    std::queue<node_ptr> queue;
    queue.push(Root);
    while(!queue.empty()){
        node_ptr elem = queue.front();
        queue.pop();
        for(auto child: elem->Childs)
        {
```

```
            queue.push(child.second);
        }
        //~BFS
        //Suf
        if(elem == Root) //not for Root
        continue;

        node_ptr parent = elem->Parent;
        parent = parent->Suffix;
        while(parent != nullptr && parent->ChildPrt(elem->Sym) == nullptr)
            parent = parent->Suffix;

        if(parent == nullptr)
            elem->Suffix = Root;
        else
            elem->Suffix = parent->ChildPrt(elem->Sym);
        //~Suf
        //SubLast
        if(elem->Suffix->Last)
            elem->SubLast = elem->Suffix;
        else if(elem->Suffix->SubLast != nullptr)
            elem->SubLast = elem->Suffix->SubLast;
        //~SubLast
    }//while
}
```

При поиске мы смотрим на символ и переходим в ребенка с этим символом, если такого не существует, то переходим по суффиксу пока не сможем перейти или пока не упремся в Root. В случае если мы дости элемента конца паттерна или элемента со ссылок на префикса сонца паттерна, то мы добавляем единицу в вектор вхождений в позиции текста минус позиция паттерна с учетом джокеров. После поиска мы смотрим на вектор вхождений, в точках где значение равно колличеству паттернов и есть начальные точки вохождений паттернов.

```
void ThisLast(node_ptr Cur,std::vector<int> &vect_incl,int pos){
    if(Cur->SubLast != nullptr)
        ThisLast(Cur->SubLast,vect_incl,pos);
    for(auto jocker: Cur->Jockers){
        if(pos-jocker >= 0)
            ++vect_incl[pos-jocker];
    }
}

void Find(std::vector<SText> &text,std::vector<int> &pos_incl){
    node_ptr cur = Root;
    int i=0;
    for(auto elem: text){
        while(cur->ChildPrt(elem.Sym)==nullptr && cur->Suffix!=nullptr)
            cur = cur->Suffix;
```

```
        if(cur->ChildPrt(elem.Sym)!=nullptr)
            cur = cur->Childs.at(elem.Sym);
        if(cur->Last)
            ThisLast(cur,pos_incl,i);
        else if(cur->SubLast != nullptr)
            ThisLast(cur->SubLast,pos_incl,i);

        ++i;
    }
}
```

Листинг:

```
#pragma once

#include <vector>
#include <unordered_map>
#include <queue>
#include <iostream>

struct SText
{
  long long Sym;
  int str;
  int word;
};


class TAhoKorasik{
protected:
class TNode{
  public:
  using node_ptr = TNode*;
  long long Sym; //Data
  bool Last;
  std::vector<int> Jockers;
  node_ptr Parent; //Ptrs
  node_ptr Suffix;
  node_ptr SubLast;
  std::unordered_map<long long,node_ptr> Childs;

  TNode(long long inSym = -1,bool inLast = false):Sym(inSym),Last(inLast){
    Parent = nullptr;
    Suffix = nullptr;
    SubLast = nullptr;
  }

  ~TNode(){
    for(auto Child: Childs){
      delete(Child.second);
```

```
38      }
39    }
40
41    node_ptr ChildPrt(long long &inSym){
42      std::unordered_map<long long,node_ptr>::iterator child = Childs.find(inSym);
43      if(child == Childs.end())
44        return nullptr;
45      else
46        return child->second;
47    }
48  };//TNode
49  using node_ptr = TNode::node_ptr;
50
51  node_ptr Root;
52
53  public:
54
55  TAhoKorasik(){
56    Root = new TNode();
57  }
58  ~TAhoKorasik(){
59    delete(Root);
60  }
61
62  void Push(const std::vector<long long> &pattern, int pos){
63    node_ptr cur = Root;
64    for(auto elem_pattern: pattern){
65      node_ptr child = cur->ChildPrt(elem_pattern);
66      if(child == nullptr){
67        child = new TNode(elem_pattern);
68        child->Parent = cur;
69        cur->Childs.insert({elem_pattern,child});
70      }
71      cur = child;
72    }
73    cur->Last = true;
74    cur->Jockers.push_back(pos);
75  }
76
77  void BorSuf(){
78    //BFS
79    std::queue<node_ptr> queue;
80    queue.push(Root);
81    while(!queue.empty()){
82      node_ptr elem = queue.front();
83      queue.pop();
84      for(auto child: elem->Childs)
85      {
86        queue.push(child.second);
```

```
87      }
88      //~BFS
89      //Suf
90      if(elem == Root) //not for Root
91        continue;
92
93      node_ptr parent = elem->Parent;
94      parent = parent->Suffix;
95      while(parent != nullptr && parent->ChildPrt(elem->Sym) == nullptr)
96        parent = parent->Suffix;
97
98      if(parent == nullptr)
99        elem->Suffix = Root;
100     else
101       elem->Suffix = parent->ChildPrt(elem->Sym);
102     //~Suf
103     //SubLast
104     if(elem->Suffix->Last)
105       elem->SubLast = elem->Suffix;
106     else if(elem->Suffix->SubLast != nullptr)
107       elem->SubLast = elem->Suffix->SubLast;
108     //~SubLast
109   }//while
110 }
111
112 void ThisLast(node_ptr Cur,std::vector<int> &vect_incl,int pos){
113   if(Cur->SubLast != nullptr)
114     ThisLast(Cur->SubLast,vect_incl,pos);
115   for(auto jocker: Cur->Jockers){
116     if(pos-jocker >= 0)
117       ++vect_incl[pos-jocker];
118   }
119 }
120
121 void Find(std::vector<SText> &text,std::vector<int> &pos_incl){
122   node_ptr cur = Root;
123   int i=0;
124   for(auto elem: text){
125     while(cur->ChildPrt(elem.Sym)==nullptr && cur->Suffix!=nullptr)
126       cur = cur->Suffix;
127     if(cur->ChildPrt(elem.Sym)!=nullptr)
128       cur = cur->Childs.at(elem.Sym);
129     if(cur->Last)
130       ThisLast(cur,pos_incl,i);
131     else if(cur->SubLast != nullptr)
132       ThisLast(cur->SubLast,pos_incl,i);
133
134     ++i;
135   }
```

```
136 }
137
138 };//TAhoKorasik
```

3 Консоль

```
pavel@DESKTOP-VBSMFB3:~/Projects/mai/2_course/DA/LB4/second$ cat ../test
1 ? 02
0001
1
02
2
pavel@DESKTOP-VBSMFB3:~/Projects/mai/2_course/DA/LB4/second$ ./solution <../test
1,1
2,1
```

4 Тест производительности

Сравнение производительности будет производится с std::find, представленным в стандартной библиотеке шаблонов C++.

	Длина паттерна	Количество строк	Количество слов в строке
test1.txt	10	10	100
test2.txt	1000	10000	100
test3.txt	10 где джокеры и сиволы через один	10000	10

```
pavel@DESKTOP-VBSMFB3:~/solution$ make
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test1.txt
Aho-AhoKorasik: 1ms
std::find: 1ms
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test2.txt
Aho-AhoKorasik: 5ms
std::find: 12ms
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test3.txt
Aho-AhoKorasik: 7
std::find: 11
```

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я реализовал алгоритм поиска подстроки в строке Ахо-Корасик.

Данный алгоритм отличается быстротой поиска множества паттернов, за счет того, что поиски всех паттернов происходят одновременно за один проход по тексту. Тем не менее при высокой энтропии текста и малом количестве паттернов этот алгоритм может быть сравним по скорости с наивным алгоритмом.