Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: Е. А. Айрапетова
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из выходных строк, необходимо воспользоваться полученным суффисным деревом для решения задания.

Формат выходных данных: Для каждого образца, найденного в тексте, нужно распечатать строку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания

Алфавит строк: Строчные буквы латинского алфавита (т.е. от a до z).

Формат входных данных: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Формат выходных данных: Строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.

Вариант: Найти в заранее известном тексте поступающие на вход образцы.

1 Описание

Требуется реализовать алгоритм Укконена для построения суффиксного дерева, из суффиксного дерева построить суффиксный массив и написать эффективный алгоритм поиска паттерна в тексте при помощи суффиксного массива. Суффиксный массив получается при обходе в глубину дерева в лексикографическом порядке. Поиск паттерна в тексте при помощи суффиксного массива основывается на поиске левой границы, где паттерн не меньше суффикса по длине в массиве и правой границы, где паттерн меньше суффикса в массиве, т.к. суффиксный массив упорядочен лексикографически, то все суффиксы лежащие между левой и правой границей являются вхождениями, вхождение границ определяется реализацией.

2 Исходный код

Основная функция считывает текст, строит по нему суффиксное дерево, по суффиксному дереву строит суффиксный массив, считывает все паттерны и находит вхождения в текст, для каждого паттерна выводит упорядоченные индексы начала каждого вхождения паттерна в текст.

lab5.cpp:

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>

class TSuffArr;

class TNode {
public:
    TNode(std::string::iterator, std::string::iterator);
    ~TNode() {};
    std::map<char, TNode*> v;
    std::string::iterator start;
    std::string::iterator end;
    TNode* suff_link;
};

class TSuffTree {
public:
    TSuffTree(std::string const&);
    ~TSuffTree();
    friend TSuffArr;
private:
    std::string text;
    TNode* root;
    TNode* curr_suff_link;
    TNode* activ_node;
    int remainder;
    int activ_length;
    std::string::iterator activ_edge;
    void Destroy(TNode*);
    void SuffLinkAdd(TNode*);
    void DFS(TNode*, std::vector<int>&, int const&);
    void Create(std::string::iterator);
    bool GoDown(std::string::iterator, TNode*);
};

class TSuffArr {
public:
```

```
41      TSuffArr(TSuffTree* tree);
42      ~TSuffArr() {};
43      std::vector<int> Find(std::string const&);
44  private:
45      int FindLeft(std::string const&);
46      int FindRight(std::string const&);
47      std::string text;
48      std::vector<int> arr;
49  };
50
51  TSuffTree::TSuffTree(std::string const& string) : text(string), root(new TNode(text.
        end(), text.end())), remainder(0) {
52      activ_edge = text.begin();
53      activ_node = root;
54      root->suff_link = root;
55      curr_suff_link = root;
56      activ_length = 0;
57      for(std::string::iterator suff = text.begin(); suff != text.end(); ++suff) {
58          Create(suff);
59      }
60  }
61
62  TNode::TNode(std::string::iterator start, std::string::iterator end) : start(start),
        end(end), suff_link(nullptr) {}
63
64  void TSuffTree::Destroy(TNode* node) {
65      for (std::map<char, TNode*>::iterator it = node->v.begin(); it != node->v.end(); ++
            it) {
66          Destroy(it->second);
67      }
68      delete node;
69  }
70
71  TSuffTree::~TSuffTree() {
72      Destroy(root);
73  }
74
75  void TSuffTree::Create(std::string::iterator pos) {
76      curr_suff_link = root;
77      ++remainder;
78      while(remainder) {
79          if(!activ_length) { activ_edge = pos; }
80          std::map<char, TNode*>::iterator v = activ_node->v.find(*activ_edge);
81          TNode* next;
82          if(v == activ_node->v.end()) {
83              TNode* leaf = new TNode(pos, text.end());
84              activ_node->v[*activ_edge] = leaf;
85              SuffLinkAdd(activ_node);
86          } else {
```

```
87              next = v->second;
88              if(GoDown(pos, next)) { continue; }
89              if(*(next->start + activ_length) == *pos) {
90                  ++activ_length;
91                  SuffLinkAdd(activ_node);
92                  break;
93              }
94              TNode* split = new TNode(next->start, next->start + activ_length);
95              TNode* leaf = new TNode(pos, text.end());
96              activ_node->v[*activ_edge] = split;
97              split->v[*pos] = leaf;
98              next->start += activ_length;
99              split->v[*next->start] = next;
100             SuffLinkAdd(split);
101         }
102         --remainder;
103         if(activ_node == root && activ_length) {
104             --activ_length;
105             activ_edge = pos - remainder + 1;
106         } else {
107             activ_node = (activ_node->suff_link) ? activ_node->suff_link : root;
108         }
109     }
110 }
111
112 bool TSuffTree::GoDown(std::string::iterator pos, TNode* node) {
113     int length = 0;
114     if(pos + 1 < node->end) {
115         length = pos + 1 - node->start;
116     } else {
117         length = node->end - node->start;
118     }
119     if(activ_length >= length) {
120         activ_edge += length;
121         activ_length -= length;
122         activ_node = node;
123         return true;
124     }
125     return false;
126 }
127
128 void TSuffTree::SuffLinkAdd(TNode* node) {
129     if (curr_suff_link != root) {
130         curr_suff_link->suff_link = node;
131     }
132     curr_suff_link = node;
133 }
134
135 void TSuffTree::DFS(TNode* node, std::vector<int>& result, int const& deep) {
```

```
136     if (node->v.empty()) {
137         result.push_back(text.size() - deep);
138         return;
139     }
140     for(std::map<char, TNode*>::iterator it = node->v.begin(); it != node->v.end(); ++
            it) {
141         int tmp = deep;
142         tmp += it->second->end - it->second->start;
143         DFS(it->second, result, tmp);
144     }
145 }
146 
147 TSuffArr::TSuffArr(TSuffTree* tree) : text(tree->text), arr() {
148     tree->DFS(tree->root, arr, 0);
149 }
150 
151 
152 int TSuffArr::FindLeft(std::string const& pattern) {
153     int left = 0;
154     int right = text.size() - 1;
155     int length = pattern.size();
156     while(left <= right) {
157         int mid = (left + right) / 2;
158         std::string tmp = text.substr(arr[mid], length);
159         if(pattern > tmp) {
160             left = mid + 1;
161         }
162         else {
163             right = mid - 1;
164         }
165     }
166     return left;
167 }
168 
169 int TSuffArr::FindRight(std::string const& pattern) {
170     int left = 0;
171     int right = text.size() - 1;
172     int length = pattern.size();
173     while(left <= right) {
174         int mid = (left + right) / 2;
175         std::string tmp = text.substr(arr[mid], length);
176         if(pattern >= tmp) {
177             left = mid + 1;
178         }
179         else {
180             right = mid - 1;
181         }
182     }
183     return left;
```

```
184 }
185
186 std::vector<int> TSuffArr::Find(std::string const& pattern) {
187     int left = FindLeft(pattern);
188     int right = FindRight(pattern);
189     std::vector<int> result;
190     for(int i = left; i < right; ++i) {
191         result.push_back(arr[i]);
192     }
193     std::sort(result.begin(), result.end());
194     return result;
195 }
196
197 int main() {
198     std::string text, pattern;
199     std::cin >> text;
200     TSuffTree tree(text + "$");
201     TSuffArr sa(&tree);
202     size_t test_number = 1;
203     while(std::cin >> pattern) {
204         std::vector<int> result = sa.Find(pattern);
205         if (!result.empty()) {
206             std::cout << test_number << ": ";
207             for(size_t i = 0; i < result.size(); ++i) {
208                 std::cout << result[i] + 1;
209                 if(i < result.size() - 1) { std::cout << ", "; }
210             }
211             std::cout << "\n";
212         }
213         ++test_number;
214     }
215     return 0;
216 }
```

3 Консоль

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$ g++ -pedantic -Wall -std=c++11 -Werror lab5.cpp
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$ cat test.txt
abcdabc
abcd
bcd
bc
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$ ./a.out <test.txt
1: 1
2: 2
3: 2,6
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$
```

4 Тест производительности

Тест производительности представляет из себя следующее: поиск 30 образцов в строке «abcgfhdeghheababctcehjtceghjtcehjdadeabcghheghhhheghhdeabcacbabchhjdetcehj» с помощью суффиксного массива.

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$ g++ -pedantic -Wall -std=c++11 -Werror benchmark.cpp
jane@Evgenia:/mnt/c/Files/ДА/ЛР5$ ./a.out <testb.txt >out.txt
all 3 ms
```

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я попрактиковалась в построении суффиксных деревьев и массивов, а также поиске подстроки в строке с помочью суффиксных массивов. Такой способ поиска подстроки в строке бывает более полезным в случае, когда нужно обработать очень большое количество разных шаблонов.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн.

[2] Суффиксные деревья — Википедия.
URL: http://ru.wikipedia.org/wiki/Суффиксное_дерево (дата обращения: 26.04.2021).

[3] Суффиксные массивы — Википедия.
URL: http://ru.wikipedia.org/wiki/Суффиксный_массив (дата обращения: 26.04.2021).