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. С. Пищик
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 своего варианта задания.

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

Вариант:

Найти в заранее известном тексте поступающие на вход образцы.

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

Формат результата: 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.

1 Описание

Требуется реализовать алгоритм Укконена для построения суффиксного дерева, из суффиксного дерева построить суффиксный маассив 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 $>=$ суффикса в массиве и правой границы, где паттерн $<$ 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

st.cpp	
TSuffTree::TSuffTree(std::string const& string)	Конструктор с 1 аргументом - строкой, который производит инициализацию.
TNode::TNode(std::string::iterator start, std::string::iterator end)	Конструктор для инициализации вершины суффиксного дерева, который принимает 2 аргумента - итераторы на начало и на конец строки.
void TSuffTree::Destroy(TNode* node)	Функция удаления вершины дерева и всех его потомков.
TSuffTree::∼TSuffTree()	Деструктор суффиксного дерева.
void TSuffTree::Create(std::string::iterator pos)	Функция вставки в дерево суффикса начинающегося с pos.
bool TSuffTree::GoDown(std::string:: iterator pos, TNode* node)	Функция спуска.
void TSuffTree::SuffLinkAdd(TNode* node)	Функция добавления суффиксной ссылки на node.
void TSuffTree::DFS(TNode* node, std::vector<int>& result, int const& deep)	Поиск в глубину в дереве в лексикографическом порядке.
TSuffArr::TSuffArr(TSuffTree* tree)	Конструктор построения суффиксного массива из суффиксного дерева tree.
int TSuffArr::FindLeft(std::string const& pattern)	Функция поиска левого индекса суффиксного массива в котором суффикс содержит паттерн.
int TSuffArr::FindRight(std::string const& pattern)	Функция поиска правого индекса суффиксного массива в котором суффикс содержит паттерн.
std::vector<int> TSuffArr::Find(std::string const& pattern)	Функция возвращающая вектор упорядоченных индексов начала вхождения паттерна в текст.

main.cpp	
int main()	Основная функция, собирающая все части программы в единое целое. 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.

```
1  class TNode
2  {
3  public:
4      TNode(std::string::iterator, std::string::iterator);
5      ~TNode() {};
6      std::map<char, TNode*> v;
7      std::string::iterator start;
8      std::string::iterator end;
9      TNode* suff_link;
10 };
11
12 class TSuffTree
13 {
14 public:
15     TSuffTree(std::string const&);
16     ~TSuffTree();
17     friend TSuffArr;
18 private:
19     std::string text;
20     TNode* root;
21     TNode* curr_suff_link;
22     TNode* activ_node;
23     int remainder;
24     int activ_length;
25     std::string::iterator activ_edge;
26     void Destroy(TNode*);
27     void SuffLinkAdd(TNode*);
28     void DFS(TNode*, std::vector<int>&, int const&);
29     void Create(std::string::iterator);
30     bool GoDown(std::string::iterator, TNode*);
31 };
32
33 class TSuffArr
34 {
35 public:
```

```
36      TSuffArr(TSuffTree* tree);
37      ~TSuffArr() {};
38      std::vector<int> Find(std::string const&);
39  private:
40      int FindLeft(std::string const&);
41      int FindRight(std::string const&);
42      std::string text;
43      std::vector<int> arr;
44  };
```

3 Консоль

```
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ cat test.txt
abcgfhdeghheababctcehjtceghjtcehjdadeabcghheghhhheghhdeabcacbabchhjdetcehj
abc
de
ghhe
tcehj
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ ./solution<test.txt
1: 1,15,38,56,62
2: 7,36,54,68
3: 9,41
4: 18,29,70
```

4 Тест производительности

Тест состоит из 400 тыс. поисков паттерна в тексте для моего поиска с использованием суффиксного массива и для стандартного метода строк find, который модифицирован для поиска нескольких образцов в тексте. Паттерны повторяются по 4 штуки - p1: «abc», p2: «de», p3: «ghhe», p4: «tcehj», p5: «abc» и т.д.
Текст: «abcgfhdeghheababctcehjtceghjtcehjdadeabcghheghhhheghhdeabcacbabchhjdetcehj»

```
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ ./benchmark
Time for create suffix tree and suffix array: 0.0007106 seconds
Time for my find: 6.08816 seconds
Time for standart find: 1.66517 seconds
```

Как можно увидеть, время работы на построение суффиксново дерева и массива крайне мало, а поиск в несколько раз медленнее, но если например нам важно сэкономить память, но не так критично время, то поиск с использованием суффиксного массива вполне подходит.

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмом Укконена построения суффиксного дерева, построения суффиксного массива из суффиксного дерева, поиска подстроки в строке с использованием суффиксного массива.

Список литературы

[1] habr алгоритм Укконена

[2] e-maxx алгоритм Укконена

[3] e-maxx суффиксный масси