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: А. В. Семин
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-20

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №5

Задача: Реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для заданных данных, воспользоваться им для решения своего варианта.
Вариант: Найти самую длинную общую подстроку двух строк.

Формат ввода
Две строки.

Формат вывода
На первой строке нужно распечатать длину максимальной общей подстроки, затем перечислить все возможные варианты общих подстрок этой длины в порядке лексикографического возрастания без повторов.

1 Описание

Решение состоит из двух частей:

1. Построение суффиксного дерева для конкатенации двух строк с помощью алгоритма Укконена за линейное время.
2. Поиск по дереву общих подстрок с помощью обхода в глубину.

Описание алгоритма Укконена:
Изначально стоит отметить, что структура дерева состоит из активной точки - тройки элементов:

- активная вершина - вершина, из которой происходит вставка элемента;
- активное ребро - ребро, по которому происходит поиск вставляемого элемента;
- активная длина - позиция символа на ребре, которую необходимо проверять.

Базово реализация этого алгоритма состоит из трех основных правил:
Правило 1. (применяется всякий раз, когда активная вершина является корневой)

- активная веришна остается корнем;
- активное ребро становится первым символом нового суффикса, который нужно вставить;
- активная длина уменьшается на 1.

Правило 2.
Если ребро разделяется и вставляется новая вершина, и если это не первая вершина, созданная на текущем шаге, ранее вставленная вершина и новая вершина соединяются через специальный указатель - суффиксную ссылку.

Правило 3.
После разделения ребра из активной вершины, которая не является корнем, переходим по суффиксной ссылке, выходящей из этой вершины, если таковая имеется. Активная вершина устанавливается вершиной, на которую она указывает. Если суффиксная ссылка отсутствует, активная вершина устанавливается корнем. Активное ребро и активная длина остаются без изменений.

2 Исходный код

Заголовочный файл tree.hpp:

```
#pragma once
#include <bits/stdc++.h>

const char FIRST_SEP = '#';
const char SECOND_SEP = '$';

struct TNode {
  std::map<char, TNode*> Next;
  TNode* suffixLink;

  int begin, end;

  TNode(int begin, int end);
  ~TNode();
};

class TSuffixTree {
public:
  std::string text;
  TNode* root;
  TNode* curNode;
  int end;
  int remainder;
  int curLen;
  int curEdge;

  void Search();
  TSuffixTree(const std::string& textA, const std::string& textB);
  ~TSuffixTree();
private:
  int indFirst_Sep;
  int DFS(std::vector<std::pair<int,int>>& ans, int& maxLen, TNode* node, int len, int
       begin);
  void AddLetter(int i);
  void printVector(const std::vector<std::pair<int, int>>& ans);
};
```

Реализация алгоритма Укконена:

```
void TSuffixTree::AddLetter(int i){
  TNode* prevAdded = root;
  ++remainder;

  if (remainder == 1) {
    curEdge = i;
```

```
7     }
8
9     while (remainder > 0) { // ,
10      auto it = curNode->Next.find(text[curEdge]);
11      TNode* nextNode = nullptr;
12      if (it != curNode->Next.end()) {
13        nextNode = it->second;
14      }
15
16      if (nextNode == nullptr) { //       curEdge
17        curNode->Next[text[curEdge]] = new TNode(i, -1);
18        if (prevAdded != root) { //
19          prevAdded->suffixLink = curNode;
20        }
21        prevAdded = curNode;
22      } else { //   curEdge
23        int edgeLen;
24        if (nextNode->end == -1) { //
25          edgeLen = end - nextNode->begin + 1;
26        } else { //
27          edgeLen = nextNode->end - nextNode->begin;
28        }
29
30        if (curLen >= edgeLen) { //
31          // std::cout << "curLen >= edgeLen\n";
32          // std::cout << "I = " << i << '\n';
33          curNode = nextNode;
34          curLen -= edgeLen;
35          curEdge += edgeLen;
36          continue; //
37        }
38
39        if (text[nextNode->begin + curLen] == text[i]) { //         (curLen)
40          curLen++;
41          if (prevAdded != root) {
42  //    std::cout << "suff2\n";
43            prevAdded->suffixLink = curNode;
44          }
45          break;
46        }
47
48        TNode* midNode = new TNode(nextNode->begin, nextNode->begin + curLen);
49        curNode->Next[text[curEdge]] = midNode;
50        midNode->Next[text[i]] = new TNode(i, -1);
51        nextNode->begin += curLen;
52        midNode->Next[text[nextNode->begin]] = nextNode;
53
54        if (prevAdded != root) {
55  //    std::cout << "suff3\n";
```

```
56          prevAdded->suffixLink = midNode;
57        }
58        prevAdded = midNode;
59      }
60
61
62      if (curNode == root && curLen > 0) {
63        curEdge++;
64        curLen--;
65      } else if (curNode != root) {
66        curNode = curNode->suffixLink;
67      }
68
69      --remainder;
70    }
71  }
```

3 Консоль

console input:

```
xabay
xabcbay
```

console output:

```
3
bay
xab
```

4 Тест производительности

Тест производиительности представляет собой сравнение реализованного мной алгоритма Укконена и наивного алгоритма поиска наибольшей подстроки за сложность $O(n^2)$.

Сравнение производится путем запуска запуска теста на двух длинных строках.

В результате работы *benchmark.cpp* видны следующие результаты:

```
a@WIN-THNQL51M105:~/Desktop/DA/lab5$ make
-O2 tree.o benchmark.o -o solution
a@WIN-THNQL51M105:~/Desktop/DA/lab5$ ./solution <randomtest.txt
3
dow
My Solution : 100us
Default Solution : 1269us
```

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», мною были изучены суффиксные деревья, а также различные виды их построения и использования для решения прикладных задач.

Суффиксные деревья позволяют обрабатывать текст за линейное время O(n), где n - длина искомого текста. В целом суффиксные деревья сильно оптимизируют работу с поиском по тексту и различными другими задачами.

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.))