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: К. А. Калугин
Преподаватель: А. А. Кухтичев
Группа: М8О-307Б

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.

1 Описание

Основная идея алгоритма состоит в построении суффиксного дерева - префиксного дерева (trie), содержащего все суффиксы текста.Для этого используется алгоритм Укконена[1]. Он основан на наивной вставке каждого суффикса каждого префикса текста и использует некоторые ускорители, которые позволяют ему работать за линейное время. Эти ускорители - хранение индексов символов в узлах, использование суффиксных ссылок и «skip/count»'ов.

2 Исходный код

Lab5.3.cpp	
int GetRight (TNode* node)	Функция получения правого края узла
void AddToNode (TNode* currentNode, int newStrIdx)	Функция добавления в дерево нового листа
void SplitArc (TNode* currentNode, int arcNumber, int newNodePos, int newStrRight)	Функция добавления новой внутренней вершины
SuffixRefTransition (TNode* currentNode, int strToSkipLength, int strToSkipBeginIdx, int newStrIdx)	Функция реализации skip/count'а.
void ApplyExtensionRules (TNode* currentNode, int arcNumber, int posInArc, int newStrIdx)	Функция применения правил продолжения (2-го и 3-го, тк первое реализуется автоматически, благодаря использованию end'а).
void ApplyExtensionRules (TNode* currentNode, int newStrIdx)	Функция применения правил продолжения. используется только для листов.
void AddNewString (TNode* currentNode, int newStrLeft, int newStrRight)	Функция добавления новой строки в дерево.
void CleanRecursive (TNode* currentNode)	Функция рекурсивной очистки дерева.
int RecCounter (TNode* curr)	Функция рекурсивного подсчета листов в потомках для каждой вершины. Являетсся препроцессингом для поиска.
void BuildTree (string & newText)	Функция построения суффиксного дерева по тексту.
void FindPattern (string & pattern, TNode* curr, int pb, int c)	Функция рекурсивного поиска паттерна по суффиксному дереву.

```
1  struct TNode {
2      int left, right;
3      int leafNumber;
4      vector <int> leaves;
5      vector <TNode*> arcs;
6      TNode* suffixRef= nullptr;
7      TNode () {}
8      TNode (int newLeft, int newRight) {}
9      TNode (int newLeft, int newRight, int newLeafNumber) {}
10 };
```

```
class TSuffixTree {
private:
    struct TNode {};
    string text;
    int end;
    TNode* root;
    int leafCounter;
    TNode* savedSuffixRef;
    int GetRight (TNode* node) {}
    void AddToNode (TNode* currentNode, int newStrIdx) {}
    void SplitArc (
        TNode* currentNode, //
        int arcNumber, //
        int newNodePos, //
        int newStrRight //     (  )
        ) {}
    void SuffixRefTransition (TNode* currentNode, int strToSkipLength, int
        strToSkipBeginIdx, int newStrIdx) {}
    void ApplyExtensionRules (TNode* currentNode, int arcNumber, int posInArc, int
        newStrIdx) {}
    void ApplyExtensionRules (TNode* currentNode, int newStrIdx) {}
    void AddNewString (TNode* currentNode, int newStrLeft, int newStrRight) {}
    void CleanRecursive (TNode* currentNode) {}
public:
    TSuffixTree () {}
    ~TSuffixTree () {}
    int RecCounter (TNode* curr) {}
    void BuildTree (string & newText) {}
    void FindPattern (string & pattern, TNode* curr, int pb, int c) {}
```

```
    int main () {
    string text;
    string pattern;
    TSuffixTree* suffixTree = new TSuffixTree;
    vector <int> result;
    cin >> text;

    suffixTree->BuildTree (text);
    unsigned int timer = clock ();
    int i = 0;
    while (cin >> pattern) {
        i ++;
        if (pattern.size() <= text.size ()) {
            suffixTree->FindPattern (pattern, nullptr, 0, i);
        }
    }
    delete suffixTree;
    cout << "Time: " << clock () - timer << std::endl;
    return 0;
}
```

3 Консоль

```
PS C:\VSC\DA>.\a.exe
abxabyabz
a
1: 1,4,7
v
b
3: 2,5,8
ab
4: 1,4,7
x
5: 3
y
6: 6
```

4 Тест производительности

Сравним реализованный алгоритм и поиск подстроки в строке с помощью std::string.find (). Тесты содержат текст размером от 50 до 14000 символов и количество паттернов равное 5000, 50000 и 500000.

```
PS C:\VSC\DA>g++ .\gen-5-02.cpp -o gen; .\gen.exe >test5; Get-Content test5
| .\a.exe >.\res.txt; Get-Content test5 | .\checker.exe >.\standart.txt; .\comp.exe
<Time: 14>
<Time: 12>
PS C:\VSC\DA>g++ .\gen-5-02.cpp -o gen; .\gen.exe >test5; Get-Content test5
| .\a.exe >.\res.txt; Get-Content test5 | .\checker.exe >.\standart.txt; .\comp.exe
<Time: 1754>
<Time: 2051>
PS C:\VSC\DA>g++ .\gen-5-02.cpp -o gen; .\gen.exe >test5; Get-Content test5
| .\a.exe >.\res.txt; Get-Content test5 | .\checker.exe >.\standart.txt; .\comp.exe
<Time: 23915>
<Time: 25358>
```

Как видно, на малом количестве паттернов стандартная библиотека выигрывает у суффиксного дерева, однако с возрастанием количества поисковых запросов время, потраченное на построение суффиксного дерева окупается и время работы нашего алгоритма становится меньше, чем у стандартного.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился следующим вещам:

Во-первых, иногда использование привычных алгоритмов (в моем случае - реализацию дерева с помощью ссылок и узлов), а не более эффективных, но менее привычных (реализация дерева в виде графа с матрицей смежности) оказывается предпочтительнее, тк упрощается работа с кодом и алгоритмом, а следовательно - уменьшается количество трудноотлавливаемых ошибок.

Во-вторых, попытки оптимизировать уже работающий алгоритм при помощи добавления if‘ов с краевыми условиями (например, поиск патерна, при равенстве длин текста и патерна) почти никогда не заканчивается чем-то хорошим.

Список литературы

[1] Гасфилд Д. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология.* / Пер. с англ. И. В. Романовского. — СПб.: Невский Диалект; БХВ-Петербург, 2003. — 654 с.: ил.

[2] *Trie, или нагруженное дерево.*
URL: `https://habr.com/ru/post/111874/` (дата обращения: 27.11.2021).

[3] *Простое суффиксное дерево.*
URL: `https://habr.com/ru/post/258121/` (дата обращения: 27.11.2021).