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: А. В. Тимофеев
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Вариант №5-1 Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск большого количества образцов при помощи алгоритма Ахо-Корасик.

Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые)

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

Формат входных данных: Искомый образец задаётся на первой строке входного файла.

В случае, если в задании требуется найти несколько образцов, они задаются по одному на строку вплоть до пустой строки.

Затем следует текст, состоящий из слов или чисел, в котором нужно найти заданные образцы.

Никаких ограничений на длину строк, равно как и на количество слов или чисел в них, не накладывается.

Формат результата В выходной файл нужно вывести информацию о всех вхождениях искомых образцов в обрабатываемый текст: по одному вхождению на строку.

Для заданий, в которых требуется найти только один образец, следует вывести два числа через запятую: номер строки и номер слова в строке, с которого начинается найденный образец. В заданиях с большим количеством образцов, на каждое вхождение нужно вывести три числа через запятую: номер строки; номер слова в строке, с которого начинается найденный образец; порядковый номер образца.

Нумерация начинается с единицы. Номер строки в тексте должен отсчитываться от его реального начала (то есть, без учёта строк, занятых образцами).

Порядок следования вхождений образцов несущественен.

1 Описание

Алгоритм Ахо-Корасик был предложен Альфредом Ахо и Маргарет Корасик. Он реализует оптимальный поиск всех вхождений всех строк-образцов в заданную строку.

Суть алгоритма заключена в использование структуры данных — бора и построения по нему конечного детерминированного автомата.

«Грубо говоря, бор — это дерево, в котором каждая вершина обозначает некую строку (корень обозначает нулевую строку). На ребрах между вершинами написана 1 буква, таким образом, добираясь по ребрам из корня в какую-нибудь вершину и контангенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине»[2].

Из определения бора как дерева вытекает также единственность пути между корнем и любой вершиной, следовательно — каждой вершине соответствует ровно один паттерн. Под словом паттерн будем понимать строку, образующуюся при прохождении от корня до конкретного листа. Строить бор будем последовательным добавление исходных паттернов. Изначально у нас есть 1 вершина, корень (root) — пустая строка. Добавление паттерна происходит так: начиная в корне, двигаемся по нашему дереву, выбирая каждый раз ребро, соответствующее очередному слову паттерна. Если такого ребра нет, то мы создаем его вместе с вершиной.

Как сказано в [1]:«Конечный автомат — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных».

Состояние автомата — это одна из вершин бора. Переход из состояний осуществляется по двум параметрам — текущей вершине v и символу ch. По которому нам надо найти вершину u, которая обозначает наидлиннейшую строку, состоящую из суффикса строки v (возможно нулевого) + символа ch. Если такая вершина не найдена, то идем в корень.

Задача поиска подстроки в строки тривиально решается за квадратичное время, поэтому для оптимальной работы важно, чтоб все части Ахо-Корасика асимптотически не превосходили линию относительно длинны строк. С помощью Ахо-Корасик мы сможем для каждого паттерна из набора найти, входит ли он в текст и, указать номер первого слова в строке, с которого начинается паттерн, за время $O(|T| + |S|)$, где |T| — суммарная длина текста, а |S| — суммарная длина паттерна.

2 Исходный код

В каждом узле бора располагается слово длиной не более 16 символов, указатели на дочерние узлы, объединенные в структуре unordered map, указатели на суффиксную ссылку, на «хорошую» суффиксную ссылку и нулевой указатель на пару, хранящую номер паттерна и количество слов в нем. Для определения конца паттерна я использую терминальный узел, в нем хранится терминальный символ и ненулевой указатель на пару, хранящую номер паттерна и количество слов в нем. Все это в себе объединяет класс TTrieNode. Класс TTrie содержит в себе указатель на корень типа TTrieNode и итератор типа TTrieNode*, а так же методы для управления структурой.

main.ccp	
main()	Основная функция, в ней происходит ввод и распознавание данных, определяется, соответствуют ли слова алфавиту, также происходит управление классом TTrie.
TTrie.h	
TTrie(const unsigned int n, T elem)	Конструктор, создает объект.
TTrie.h	
TTrie()	Диструктор, разрушает объект.
TTrie.h	
void nonTerminal (unsigned long long numStr, unsigned long long len);	Функция создает терминальный узел и делает его дочерним относительно узла, на который указывает итератор.
TTrie.h	
void myFree();	Функция рекурсивной отчистки памяти класса TTrieNode.
TTrie.h	
TTrieNode* CreateChilde(const std::string simbol);	Функция создает дочерний узел для узла на который указывает итератор itTTrie.
TTrie.h	
void Create(const std::string &symbols);	Функция проверяет условия добавления слова и передвигает итератор на найденное слово, либо запускает CreateChilde.
TTrie.h	

void Search(const std::vector<std::pair<unsigned long long int, unsigned long long int» & strWordVec, const std::string & symbols);	Функция поиска.
TTrie.h	
void CreateLinks();	Функция создания суффиксных ссылок и хороших суффиксных ссылок.
TTrie.h	
void ResetItTTrie ();	Переставляет итератор itTTrie на корень.
TTrie.h	
TTrieNode* GetIter();	Возвращает итератор.
TTrie.h	
bool Go(const std::string & symbols);	Функция ищет строку, переданную в функцию. Если находит, передвигает итератор на нее, если не находит передвигает итератор на root и возвращает значение false.
TTrie.h	
void TTrie::Answer(const std::vector<std::pair<unsigned long long int, unsigned long long int» &strWordVec, std::unordered$_map < std :: string, TTrieNode* >:: iterator&iter$);	Функция проверяет, есть ли совпадение с паттерном и печатает ответ.

```
1  class TTrieNode{
2  public:
3      friend class TTrie;
4      TTrieNode(){
5          numLengt = nullptr ;
6          linkFail = nullptr;
7          exitPoint = nullptr;
8      }
9      void nonTerminal (unsigned long long numStr, unsigned long long len);
10     ~TTrieNode() {
11     };
12 private:
13     void myFree();
14     TTrieNode* CreateChilde(const std::string simbol);
15     std::unordered_map<std::string, TTrieNode *> children;
16     TTrieNode *linkFail;
17     TTrieNode *exitPoint;
18     std::pair<unsigned long long, unsigned long long> *numLengt;
```

```
19 };
20
21 class TTrie{
22 public:
23     TTrie(){
24         this->root = new TTrieNode;
25         itTTrie = this->root;
26     }
27     void Create(const std::string &symbols);
28     void Search(const std::vector<std::pair<unsigned long long int, unsigned long long
           int>> &strWordVec, const std::string &symbols);
29     void CreateLinks();
30     void ResetItTTrie ();
31     ~TTrie() {
32         this->root->myFree();
33         this->itTTrie = nullptr;
34         delete this->root;
35     };
36     TTrieNode* GetIter(){
37         return this->itTTrie;
38     }
39 private:
40     TTrieNode *root;
41     TTrieNode *itTTrie;
42     bool Go(const std::string & symbols);
43     void Answer(const std::vector<std::pair<unsigned long long int, unsigned long long
           int>> &strWordVec, std::unordered_map<std::string,TTrieNode *>::iterator &iter)
           ;
44 };
```

3 Консоль

Тесты для этой лабораторной работы я делал как вручную, так и с помощью самописного генератора тестов. Вручную я проверял конкретные нетривиальные случаи, а генератором создавал большие однообразные тесты для оценки производительности. Пример урезанного теста, полученного с помощью генератора, приведен ниже.

```
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/laba4/solution$ make
g++ -O2 -g -pedantic -std=c++17 -Wall -Werror -c main.cpp
g++ -O2 -g -pedantic -std=c++17 -Wall -Werror main.o -o solution
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/laba4/solution$ ./solution
<test >output
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/laba4/solution$ cat output
1,4,3
1,10,3
2,3,2
2,9,3
4,7,2
5,4,2
6,3,1
6,9,3
dude@DESKTOP-545VSUH:/mnt/d/education/education/DA/laba4/solution$ cat test
dfgdfgje qwetrytyud dfgdfgje
dfgdfg dfgdfgje dfgdfgje
qwetrytyud qwetrytyud dfgdfgje

dfgdfgje dfgdfg dfgdfgje qwetrytyud qwetrytyud dfgdfgje dfgdfgje dfgdfg dfgdfg
qwetrytyud qwetrytyud dfgdfgje
dfgdfg dfgdfg dfgdfg dfgdfgje dfgdfgje dfgdfgje dfgdfg qwetrytyud qwetrytyud
qwetrytyud dfgdfgje
dfgdfg qwetrytyud dfgdfgje dfgdfgje dfgdfgje dfgdfgje qwetrytyud qwetrytyud
qwetrytyud dfgdfg dfgdfgje
dfgdfg dfgdfgje dfgdfg dfgdfg dfgdfg qwetrytyud dfgdfg dfgdfgje dfgdfgje dfgdfg
dfgdfgje
dfgdfg qwetrytyud dfgdfg dfgdfg dfgdfgje dfgdfgje qwetrytyud qwetrytyud dfgdfg
qwetrytyud dfgdfgje
dfgdfg dfgdfg dfgdfgje qwetrytyud dfgdfgje dfgdfg qwetrytyud qwetrytyud qwetrytyud
qwetrytyud dfgdfgje
dfgdfg qwetrytyud dfgdfg dfgdfg qwetrytyud dfgdfg dfgdfgje qwetrytyud qwetrytyud
dfgdfg dfgdfgje
```

4 Выводы

Поиск подстрок в тексте относительно частая задача, она встречается, пожалуй, в любом приложении работающем с текстом. В лабораторной работе мне было предложено реализовать алгоритм Ахо-Карасик. Знания, которые я приобрел в ходе лабораторной работы, очень полезны, так как алгоритм Ахо-Карасик выполняет поиск множества подстрок в тексте быстро за линейное время O(n+m), где n - сумма длин всех образцов, а m - длинна текста. Также алгоритм Ахо-Карасик способен решать и другие задачи педставленные в других вариантах лабораторных работ, например поиск с помощью вариации алгоритма Ахо-Карасик для образца с джокерами.

Эта лабораторная работа укрепила в моем сознании значимость роли древовидных структур в программировании, позволив по-новому на них взглянуть и узнать их новое применение. Я уверен, что полученные мной знания еще не раз пригодятся мне в будущем.

Список литературы

[1] *Конечный автомат — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Конечный_автомат` (дата обращения: 28.08.2021).

[2] *Алгоритм Ахо-Корасик — Habr.*
URL: `https://habr.com/ru/post/198682/` (дата обращения: 27.08.2021).

[3] *Алгоритм Ахо-Корасик — neerc.ifmo.ru.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Ахо-Корасик` (дата обращения: 27.08.2021).