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: М. А. Бронников
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №4

Задача: 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).

Формат входных данных

Искомый образец задается 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 на количество слов или числ в них, не накладывается.

Формат выходных данных

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

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

Требуется написать реализацию алгоритма *Ахо-Корасик* поиска нескольких образцов в тексте.

Как сказано в [2]: «Суть алгоритма заключена в использовании структуры данных — **бора** и построения по нему **конечного детерминированного автомата**».

Грубо говоря, «бор — это дерево, в котором каждая вершина обозначает какую-то строку (корень обозначает нулевую строку — ε). На ребрах между вершинами написана 1 буква (в этом его принципиальное различие с суффиксными деревьями и др.), 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].

Для преобразования бора в конечный автомат нам понадобятся **суффиксные ссылки**. «Если мы пытаемся выполнить переход по какой-либо букве, а соответствующего ребра в боре нет, то мы тем не менее должны перейти в какое-то состояние. Для этого нам и нужны суффиксные ссылки»[3].

Также мы введем **«хорошие» суффиксные ссылки**, которые указывают на концы образцов, которые соответствуют данному сосотоянию.

При *поиске* по тексту мы считываем символ, котрый определяет следующий узел бора, после чего проверяем этот узел на конец образца, а также проходим по всем «хорошим» суффиксным ссылкам. Если конец образца найден, выводим соответствующую информацию на экран.

2 Исходный код

В каждом узле бора должен располагаться соответствующий символ алфавита, множество указателей на сыновей узла(множество переходов в различные состояния автомата), признак окончания образца(причем моя реализация воспринимает два введёных идентичных образца как разные образцы), а также суффиксную ссылку и «хорошую» суффиксную ссылку. Для этого создадим структуру $KANode$, которая содержит подструктуру $Massive$, которая хранит в себе указатели на всех сыновей узла в отсортированном виде. Структура Bor состоит из указателя на корневой узел и высоты бора(для экономичных проходов по бору). Помимо этого введем вспомогательную структуру $Flag$ для хранения пар целых чисел.

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <string.h>
6
7  #define BUFSIZE 17
8
9  typedef struct Flag{
10   unsigned int lenght;
11   unsigned int number;
12 }Flag;
13
14 typedef struct KANode{
15
16   struct Massive{
17     struct KANode** data;
18     unsigned int size;
19     unsigned int buzy;
20   } dict;
21
22   Flag* flag;
23   char key[BUFSIZE];
24   struct KANode* suffix;
25   struct KANode* goodsuf;
26
27 }KANode;
28
29 typedef struct Massive Massive;
30
31 typedef struct Bor{
32   KANode* head;
33   unsigned int height;
34 }Bor;
35
```

```
36 void Make_KA(Bor* bor);
37 bool Create_Bor(Bor* bor);
38 short unsigned int Make_Bor(Bor* bor);
39 void Destroy_KA(Bor* bor);
40 void Error(short unsigned int i);
41 void Massive_Sort(Massive* mass);
42 KANode* Massive_Search(Massive* mass, char buffer[BUFSIZE]);
43 short unsigned int Text_Search(Bor* bor);
44 KANode* Get_Next(KANode* bor, char buffer[BUFSIZE]);
```

Функции и процедуры программы	
void Make_KA(Bor* bor)	Процедура преобразования бора в конечный автомат
bool Create_Bor(Bor* bor)	Функция начальной инициализации бора при создании
short unsigned int Make_Bor(Bor* bor)	Функция считывания образцов и запонения ими бора
void Destroy_KA(Bor* bor)	Процедура уничтожения конечного автомата
void Error(short unsigned int i)	Примитивный обработчик ошибок с выводом на экран
void Massive_Sort(Massive* mass)	Процудура сортировки вставками массива
KANode* Massive_Search(Massive* mass, char buffer[BUFSIZE])	Функция бинарного поиска указателя в массиве по символу
short unsigned int Text_Search(Bor* bor)	Функция поиска вхождений образцов в тексте с выводом на экран
KANode* Get_Next(KANode* bor, char buffer[BUFSIZE])	Функция поиска следующего состояния для автомата

Поиск образцов использует **очередь** для граммотного вывода результата на экран, однако её реализация основана на статичном массиве(длина которого соответствует высоте бора) и математическими операциями с индексами.
Проходы по бору же осуществляются на стэковой модели прохода в глубину(без рекурсии). **Стэк** реализован также при помощи массивов длины равной высоте бора.

3 Консоль

Тесты для этой лабораторной я генерировал вручную, так как каждый тест имел целью проверить определенные особенности работы программы, которые исследовать программно сгенерированными тестовыми файлами довольно трудно.

```
max@max-X550CC:~/DA/lab4$ ls
prog.c  test  test2
max@max-X550CC:~/DA/lab4$ gcc -std=c99 -Wall prog.c -o run
max@max-X550CC:~/DA/lab4$ cat test
qwert fvss wtrwr
vvs vfvs qwert fvss rt
wtrwr qwert fvss wtrwr

abcd qwert fvss vvs vfvs qwert fvss

wtrwr qwert

fvss

wtrwr qwert fvss wtrwr
max@max-X550CC:~/DA/lab4$ ./run <test
1,6,1
3,1,3
3,2,1
7,1,3
7,2,1
max@max-X550CC:~/DA/lab4$ cat test2
abc AbC aBc ABC
abC
Abc abc
efg abc

abc
ABC
ABC ABc eFg


AbC  eref
abc        abc
```

```
max@max-X550CC:~/DA/lab4$ ./run <test2
1,1,2
1,1,3
2,1,2
2,1,3
3,1,2
1,1,1
3,1,3
3,2,2
3,3,4
4,1,2
5,1,2
5,1,3
5,2,2
max@max-X550CC:~/DA/lab4$ exit
```

4 Выводы

Выполнив *Лабораторную работу №4* по курсу «Дискретный анализ», я получил новые знания и опыт в решении проблемы поиска образца в тексте.
Эта тема не является для меня новой, так как еще при обучении на первом курсе я столкнулся с ней в «Курсе инфрматики», где приводились алгоритмы *Бойера-Мура, Раббена-Карпа, Кнута-Морриса-Прата*. Однако с реализацией алгоритма поика, со всеми её особенностями, я столкнулся впервые только сейчас. К тому же новой для меня оказалась и тема нахождения сразу нескольких образцов в тексте, ведь приведенные выше алгоритмы позволяли находить лишь одну подстроку.

Поэтому изучение и реализация алгоритма *Ахо-Корасик* сослужило мне хорошую службу, дав полезные знания, расширив мой арсенал средств разработки, а также позволив по-новому взглянуть на вопрос поиска подстроки в тексте. Эта лабораторная в очередной раз укрепила в моем сознании значимость роли деревьев в программировании, позволив узнать новые их применения.
Я уверен, что полученные мной знания еще не раз пригодятся мне в будущем на пути к мечте стать хорошим программистом.

Список литературы

[1] *Алгоритм Ахо — Корасик — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Ахо_-_Корасик` (дата обращения: 16.03.2019).

[2] *Алгоритм Ахо — Корасик — Хабр.*
URL: `https://habr.com/ru/post/198682/` (дата обращения: 16.03.2019).

[3] *Алгоритм Ахо — Корасик — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Ахо-Корасик` (дата обращения: 16.03.2019).