Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: А. А. Пустовалова
Преподаватель: А. А. Кухтичев
Группа: 8О-206
Дата:
Оценка:
Подпись:

Москва, 2016

Лабораторная работа №4

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск одного образца-маски, в образце может встречаться «джокер», равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$.

1 Описание

Требуется реализовать алгоритм Ахо-Корасик поиска всех вхождений заданных строк в текст. Необходимо построить такую структуру данных как бор для всех заданных строк. Элемент бора состоит из указателей на вершины дерева по символам, следующим за данным в заданных строках. Для работы алгоритма Ахо-Корасик нужно для каждой вершины определить связь неудач и связь выхода. Связь неудач ведёт в вершину с префиксом равным наибольшему суффиксу списка элементов от корня до текущего элемента. Связь выхода ведёт в терминальную вершину с наибольшим суффиксом равным суффиксу списка пройденных элементов до текущего. При поиске образцов в тексте достаточно перемещаться двумя указателями по тексту и по бору. Если из текущего элемента бора есть указатель по следующему элементу текста, переходим по нему, опускаемся по связям выхода, пока они существуют, и помечаем вхождение строк. Иначе переходим к родителю, затем по связи неудач, пока не окажемся в корне или не будет существовать указатель на следующий элемент по данному символу текста. Связи неудач и связи выхода строятся за $O(n)$. Полное время работы алгоритма: $O(n)$ на подготовку и $O(m+k)$ на поиск, т.е. $O(n+m+k)$.

2 Исходный код

Будем считывать первую строку текста посимвольно, считать числа и записывать в вектор. Если встретили знак вопроса, запишем в бор полученный вектор чисел. При вставке начнём идти по бору от корня. Если есть переход из текущей вершины по элементу массива, переходим. Иначе создадим новую вершину, создадим указатель в текущей вершине на новую по элементу массива. Когда дойдём до последнего элемента массива, добавим порядковый номер числа в шаблоне в вектор терминальных состояний вершины дерева. Вызовем функцию расстановки связей неудач и выхода. У корня связь неудач указывает на себя, у потомков корня связь неудач указывает на корень. Далее, обходя бор в ширину, расставляем связи следующим образом: для вершины i переходим к её предку, пока не дойдем до корня или пока не встретим указатель из текущей вершины по значению вершины i , будем переходить по связям неудач. Если нашли вершину, имеющую указатель по значению вершины i, делаем связь неудач из вершины i в найденную. Иначе связь неудач будет вести в корень. Одновременно расставляем связи выхода. Если найденная вершина для связи неудач является последней для некоторой строки шаблона, создадим связь выхода на эту вершину, в противном случае, если у найденной вершины имеетсся связь выхода на какую-либо вершину, создадим связь выхода на эту же вершину. Будем посимвольно считывать текст, записывать числа в вектор, сохранять местоположение каждого числа в тексте. Одновременно перемещаясь по тексту и бору будем помечать все вхождения строк шаблона в текст. Если для данного числа количество найденных слов шаблона, удаленных от него на порядковый номер в строке шаблона, равно количеству слов в шаблоне, то мы нашли вхождение шаблона в текст.

1.c	
void Bor(TItem* root, vector<unsigned int>& n, unsigned int leaf)	Создание бора
void MakeLinks(TItem* root)	Расстановка связей неудач и выхода

```
class TItem {
public:
    map<unsigned int, TItem*> next;
    vector<int> leaf;
    TItem* parent;
    TItem* s1;
    TItem* s2;
    unsigned int v;
    TItem(char v1) {
        s1 = NULL;
        s2 = NULL;
        v = v1;
    }
};
```

3 Консоль

```
2 3 ? ? 5 1 2 3 ? 5 1 2 3 6 ? 2 3 ? 1 5 1 2 3 4

2 3 1 5 2 3 2 3 5 12 5 1 2 3 15 5 1
2 3 12 141 5 1 2 3 189 5 1 2 3 6 342
2 3 234242 1 5 1 2 3 4 5 1 2 3 6 1241 2 3 121
1 5 1 2 3 4 121 12312
3,1
4,1
```

4 Тест производительности

Тест производительности представляет из себя следующее: алгоритм Ахо-Корасик сравнивается с наивной реализацией поиска образцов в тексте, время измеряется в тактах процессора.

Входные данные состоят из 50000 чисел или знаков вопроса в шаблоне и из 1000 строк по 1000 случайных чисел в каждой строке текста.

```
Ахо-Корасик: 4596602
Наивная реализация: 295829510
```

Тестирование показало, что алгоритм Ахо-Корасик в несколько раз эфективнее наивной реализации поиска в тексте.

5 Выводы

Выполнив третью лабораторную работу, я научилась реализовывать алгоритм Ахо-Корасик для поиска в тексте, находить все вхождения шаблона с «джокерами» в тексте с его помощью. Также я провела сравнение скорости работы алгоритма с наивной реализацией решения.