

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Ахо-Корасик

Студентка гр. 9383

Сергиенкова
А.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург
2021

Цель работы

Изучение алгоритма Ахо-Корасик и его реализация на языке программирования C++.

Задание 1

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст ($T, 1 \leq |T| \leq 100000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1, \dots, p_n\}$ $1 \leq |p_i| \leq 75$

Все строки содержат символы из алфавита $\{A, C, G, T, N\}$

Выход:

Все вхождения образцов из P в T .

Каждое вхождение образца в текст представить в виде двух чисел - i p

Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1).

Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Sample Input:

```
NTA
G3
TAG
T
TAG
T
```

Sample Output:

```
2 2
2 3
```

Задание 2

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу P необходимо найти все вхождения P в текст T .

Например, образец $ab??c?$ с джокером $?$ встречается дважды в тексте $xabvccbababcsax$.

Символ джокер не входит в алфавит, символы которого используются в T . Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита $\{A, C, G, T, N\}$

Вход:

Текст ($T, 1 \leq |T| \leq 1000000$)

Шаблон ($P, 1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA

A\$\$A\$

\$

Sample Output:

1

Основные теоретические положения.

Алгоритм Ахо-Корасик реализует эффективный поиск всех вхождений всех строк-образцов в заданную строку. На вход алгоритму поступают строка и несколько строк шаблонов. Задача алгоритма – найти все возможные вхождения строк шаблонов в строку.

Для работы алгоритма необходимо реализовать бор и автомат, основанный на боре.

Бор – это дерево, в котором каждая вершина обозначает какую-то строку (коренная вершина обозначает пустую строку). При создании нового бора, в нём находится только корень. Рёбра этого дерева обозначают буквы строки. Для построения автомата на основе бора, потребуются суффиксные ссылки.

Суффиксная ссылка – это наибольший суффикс строки. Для коренной вершины суффиксная ссылка – это петля. Чтобы построить суффиксную ссылку нужно найти вершину, в которую ведёт ребро с символом из суффиксной ссылки вершины предка. Конечная суффиксная ссылка – это суффиксная ссылка конечной

вершины.

Описание алгоритма

На первом шаге алгоритма рассматриваем каждый шаблон в наборе. Строим бор. Проверяем, существует ли вершина, в которую можно перейти по ребру, помеченному символом из шаблона, если такой вершины нет – создаём такую вершину и ребро. Переходим по бору.

На втором шаге строим автомат на основе бора. Проверяем текущую вершину: если корень, то суффиксная ссылка тоже корень, иначе суффиксная ссылка – это вершина, в которую ведёт ребро с данным символом из суффиксной ссылки родительской вершины. При этом, если ссылка ищется для вершины, следующей за корнем, то для неё ссылка будет корнем.

Если вершина, в которую осуществлён переход, терминальная, то добавляем информации о вхождении встроку соответствующего ей шаблона в список. Если для этой же вершины конечная ссылка не пустая, то переходим по этим ссылкам до тех пор, пока они не пустые.

Чтобы реализовать алгоритм Ахо-Корасик с джокером, делаем всё тоже самое, но бор строится не для шаблона, а для безджокерных подшаблонов, находящихся в нём. На основе бора строим автомат.

Если в строке нашелся подшаблон(любой), то ячейку массива(который заранее был заполнен нулями) по адресу, который образован разностью номера начального символа данного подшаблона в строку и его смещения относительно начала первого подшаблона, инкрементируем.

В конечном итоге индексы ячеек массива, значение которых будет равно кол-ву подшаблонов в исходном шаблоне, будут индексами вхождения заданного шаблона в строку.

Сложность алгоритма.

1. Алгоритм Ахо-Корасик (поиск вхождений безмасочных шаблонов).

Сложность по памяти.

Поскольку автомат хранится как красно-чёрное дерево, его сложность по памяти – $O(n)$, где n – суммарная длина шаблонов.

Сложность по времени.

Сложность алгоритма по времени в данном случае – $O((T + n)\log(s) + k)$, где T – длина строки, в которой ищутся вхождения, s – размер алфавита, k – общее количество вхождений шаблонов в текст, так как время на построение бора (и автомата) сокращается по сравнению с другими реализациями (лишние символы не добавляются), что влияет и на время обработки строки.

2. Алгоритм Ахо-Корасик (поиск вхождений шаблона с маской).

Сложность по памяти.

Поскольку ищется один шаблон, то сложность по памяти составит $O(n + T)$, где n – суммарная длина всех подшаблонов в шаблоне с маской, а T – длина строки (на сложность влияет добавление массива индексов).

Сложность по времени.

Поскольку время тратится также и на заполнение массива индексов, сложность по памяти составляет $O((T + n)\log(s) + k \cdot p)$, где s – размер алфавита, где k – общее количество вхождений шаблонов в текст, p – суммарное количество сдвигов подшаблонов относительно исходного шаблона.

3. Поиск длин самых длинных цепочек из суффиксных и конечных ссылок.

Сложность по памяти.

Сложность по памяти данного алгоритма – $O(n)$, где n – суммарная длина всех шаблонов, поскольку рассматриваются вершины бора, и для каждой хранится пара значений с наибольшими длинами.

Сложность по времени.

Для каждой вершины рассматриваются цепочки из суффиксных и конечных ссылок (первых в боре не больше, чем количество символов в самом длинном шаблоне – a , а последних – не больше, чем шаблонов – b), поэтому сложность по времени данного алгоритма составляет $O(n \cdot (a + b))$.

Описание функций и СД

В структуре Node реализованы поля, которые позволяют хранить всю информацию для описания бора, суффиксных ссылок.

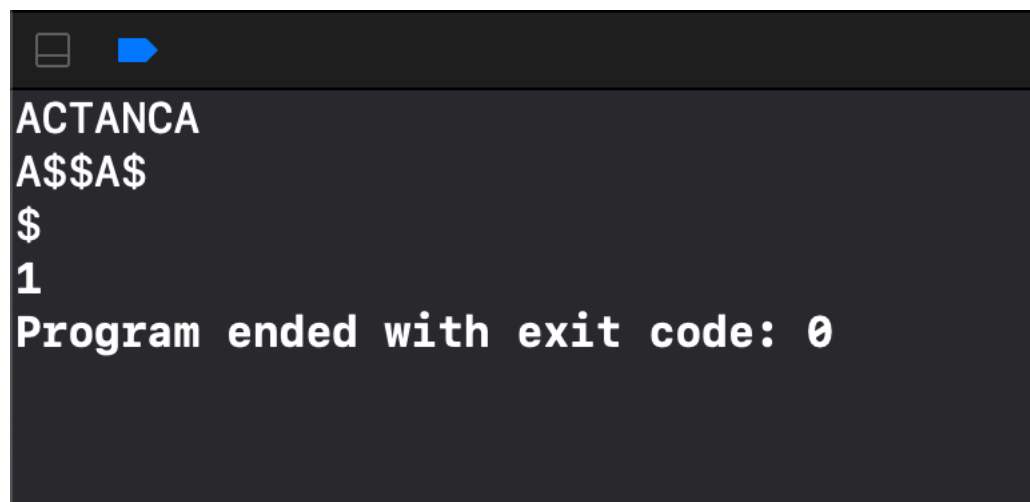
- `std::unordered_map<char, Node*> m_nextEdges` – в этом контейнере хранятся все рёбра, по которым можно перейти.
- `std::vector<int> m_shiftSubstr` – в этом контейнере хранятся сдвиги подстрок в шаблоне, для реализации алгоритма с джокером.
- `std::vector<int> m_patternNumber` – в этом контейнере хранятся номера шаблонов, в которые входит символ, по которому мы пришли.
- `char toParent` – ребро, по которому мы пришли из родительской вершины.
- `bool isTerminal` – флаг, который показывает: является-ли вершина терминальной.
- `int termPatternNumber` – номер шаблона конечной вершины.
- `Node* GetLink(const char &c)` – функция для поиска следующей вершины при поиске в строке
- `void FindSuffixLinks(Node* bohr)` – поиск суффиксных и конечных ссылок
- `std::pair<int, int> LenAllLinks(Node *bohr, node *root, int &depth)` – вычисление длин наибольших цепочек из суффиксных и конечных ссылок

Тестирование



```
NTAG
3
TAGT
TAG
T
2 2
2 3
Program ended with exit code: 0
```

Рисунок 1 – Тестирование с входными данными №1.



```
ACTANCA
A$$$A$
$
1
Program ended with exit code: 0
```

Рисунок 2 – Тестирование с входными данными №2.

Выводы.

Был изучен алгоритм Ахо-Корасик, а так же была произведена реализация данного алгоритма на языке программирования C++