УДК 004.021

АЛГОРИТМ КНУТА-МОРРИСА-ПРАТТА НА ЯЗЫКЕ PYTHON

А.П. ГАЙДЕЛЬ
(*Представлено: канд. физ.-мат. наук, доц. О.Н. ПЕТРОВИЧ*)

В данной статье рассматриваются реализация алгоритма поиска подстроки в строке кнута-морриса-пратта на языке python

Введение. Часто в задачах поиска информации одной из важнейших задач является поиск точно заданной подстроки в строке. Дело в том, что очевидный способ, который все формулирует как «взять и поискать», является отнюдь не самым эффективным, а для такой низкоуровневой и сравнительно частовызываемой функции это немаловажно. Поэтому в данной статье рассмотрены некоторые варианты реализации поиска подстроки в строке, в частности, алгоритм Кнута-Морриса-Пратта.

Основной раздел. Примитивный алгоритм поиска подстроки в строке основан на переборе всех подстрок, длина которых равна длине шаблона поиска, и посимвольном сравнении таких подстрок с шаблоном поиска. По традиции шаблон поиска или образец принято обозначать как needle (англ. «иголка»), а строку, в которой ведётся поиск — как haystack (англ. «стог сена»). На языке Python примитивный алгоритм выглядит так:

```
index = -1
for i in xrange(len(haystack)-len(needle)+1):
    success = True

    for j in xrange(len(needle)):
        if needle[j]<>haystack[i+j]:
            success = False
            break

    if success:
        index = i
        break
print index
```

Листинг 1. – Простой алгоритм поиска

Обозначим n=|haystack|, m=|needle|. Простейший алгоритм поиска даже в лучшем случае проводит n–m+1 сравнений; если же есть много частичных совпадений, скорость снижается до O(n*m).

Рассматриваемый далее алгоритм хотя и имеет невысокую скорость на «хороших» данных, но это компенсируется отсутствием регрессии на «плохих». Алгоритм Кнута-Морриса-Пратта является одним из первых алгоритмов с линейной оценкой в худшем случае. Прежде чем перейти к описанию алгоритма, необходимо рассмотреть понятие префикс-функции.

Префикс-функция строки $\pi(S,i)$ – это длина наибольшего префикса строки S[1..i], который не совпадает с этой строкой и одновременно является ее суффиксом. Проще говоря, это длина наиболее длинного начала строки, являющегося также и ее концом. Для строки S удобно представлять префикс функцию в виде вектора длиной |S|-1. Можно рассматривать префикс-функцию длины |S|, положив $\pi(S,1)=0$ (табл.).

Таблица. – Пример префикс функции для строки «abcdabcabcdabcdab»

S[i]	a	b	c	d	a	b	c	a	b	c	d	a	b	c	d	a	b
$\pi(S,i)$	0	0	0	0	1	2	3	1	2	3	4	5	6	7	4	5	6

Предположим, что $\pi(S,i)=k$. Отметим следующие свойства префикс-функции.

1. Если $S[i+1]=S[k+1]$, то $\pi(S,i+1)=k+1$.
2. $S[1..\pi(S,k)]$ является суффиксом строки S[1..i]. Действительно, если строка S[1..i] оканчивается строкой $S[1\ldots \pi(S,i)]=S[1..k]$, а строка S[1..k] оканчивается строкой $S[1..\pi(S,k)]$, то и строка S[1..i] оканчивается строкой $S[1..\pi(S,k)]$.
3. $\forall\, j\in(k,i)$, S[1..j] не является суффиксом строки S[1..i]. В противном случае было бы неверным предположение $\pi(S,i)=k$, так как j>k.

Рассмотренные свойства позволяют получить алгоритм вычисления префикс-функции. Пусть π(S,i)=k. Необходимо вычислить π(S,i+1).

1. Если S[i+1]=S[k+1], то π(S,i+1)=k+1.
2. Иначе, если k=0, то π(S,i+1)=0.
3. Иначе положить k:=π(S,k) и перейти к шагу 1.

Ключевым моментом для понимания сути алгоритма является тот факт, что если найденный на предыдущем шаге суффикс не может быть расширен на следующую позицию, то мы пытаемся рассматривать меньшие суффиксы до тех пор, пока это возможно.

```
def prefix(s):
    v = [0]*len(s)
    for i in xrange(1,len(s)):
        k = v[i-1]
        while k > 0 and s[k] <> s[i]:
            k = v[k-1]
        if s[k] == s[i]:
            k = k + 1
        v[i] = k
    return v
```

Листинг 2. – Алгоритм вычисления префикс-функции на языке Python

Покажем, что время работы алгоритма составляет O(n), где n=|S|. Заметим, что асимптотику алгоритма определяет итоговое количество итераций цикла while. Это так, поскольку без учета цикла while каждая итерация цикла for выполняется за время, не превышающее константу. На каждой итерации цикла for k увеличивается не более чем на единицу, значит максимально возможное значение k=n–1. Поскольку внутри цикла while значение k лишь уменьшается, получается, что k не может суммарно уменьшиться больше, чем n–1 раз. Значит цикл while в итоге выполнится не более n раз, что дает итоговую оценку времени алгоритма O(n).

Рассмотрим алгоритм Кнута-Морриса-Пратта, основанный на использовании префикс-функции. Как и в примитивном алгоритме поиска подстроки, образец «перемещается» по строке слева направо с целью обнаружения совпадения. Однако ключевым отличием является то, что при помощи префикс-функции мы можем избежать заведомо бесполезных сдвигов.

Пусть S[0..m–1] – образец, T[0..n–1] – строка, в которой ведется поиск. Рассмотрим сравнение строк на позиции i, то есть образец S[0..m–1] сопоставляется с частью строки T[i..i+m–1]. Предположим, первое несовпадение произошло между символами S[j] и T[i+j], где i < j < m. Обозначим P = S[0..j–1] = T[i..i+j–1]. При сдвиге можно ожидать, что префикс S сойдется с каким-либо суффиксом строки P. Поскольку длина наиболее длинного префикса, являющегося одновременно суффиксом, есть префикс-функция от строки S для индекса j, приходим к следующему алгоритму:

1. Построить префикс-функцию образца S, обозначим ее F.
2. Положить k = 0, i = 0.
3. Сравнить символы S[k] и T[i]. Если символы равны, увеличить k на 1. Если при этом k стало равно длине образца, то вхождение образца S в строку T найдено, индекс вхождения равен i – |S| + 1. Алгоритм завершается. Если символы не равны, используем префикс-функцию для оптимизации сдвигов. Пока k > 0, присвоим k = F[k–1] и перейдем в начало шага 3.
4. Пока i < |T|, увеличиваем i на 1 и переходим в шаг 3.

Возможная реализация алгоритма Кнута-Морриса-Пратта на языке Python выглядит так:

```
def kmp(s,t):
    index = -1
    f = prefix(s)
    k = 0
    for i in xrange(len(t)):
        while k > 0 and s[k] <> t[i]:
            k = f[k-1]
        if s[k] == t[i]:
            k = k + 1
        if k == len(s):
            index = i - len(s) + 1
            break
    return index
```

Листинг 3. – Возможная реализация алгоритма Кнута-Морриса-Пратта на языке Python

Заключение. Надеюсь, кто-то посчитает эту статью полезной для себя. Алгоритм Кнута-Морриса-Пратта не единственный быстрый алгоритм поиска, но он достаточно быстрый (для задач типа олимпиадных) и при этом простой. Алгоритм Бойера-Мура близок к Кнута-Морриса-Пратта по сложности и для определенного круга задач (где образец не содержит повторяющихся фрагментов) он быстрее. Но зато для другого круга задач (где образец и строка поиска содержат повторяющиеся последовательности, как в примерах к статье) он заметно уступает алгоритму Кнута-Морриса-Пратта.

ЛИТЕРАТУРА

1. Wikipedia [Электронный ресурс] Алгоритм Кнута — Морриса — Пратта. Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Кнута_-_Морриса_-_Пратта. Дата доступа: 11.09.19.
2. Py-algorithm [Электронный ресурс] Алгоритм Кнута-Морриса-Пратта (КМП). Режим доступа: http://py-algorithm.blogspot.com/2013/04/blog-post_19.html/. Дата доступа: 15.09.19.
3. Intuit [Электронный ресурс] Алгоритмы поиска в тексте. Режим доступа: https://www.intuit.ru/studies/professional_skill_improvements/2056/courses/504/lecture/1146. Дата доступа: 18.09.19.