

Week-10 Programming Assignment

Problem 1

[Solution](#)

[Public Test Cases](#)

[Private Test Cases](#)

Problem 2

[Solution](#)

[Public Test case](#)

[Private Test Case](#)

Problem 1

Write a Python function **BMCount(t, p)** that accepts two arguments a text **t** and a pattern **p** and implements a string-matching algorithm based on Boyer-Moore skipping heuristic discussed in lectures, and returns intermediate steps data as listed below.

- Record the indexes of the characters in text **t** that will be matched with the last character of **p** in a list say **skipL**. **skipL** is the list of integers sorted in ascending order, where each integer is an index of a character in text **t**.
- Also count the number of character comparisons performed between **t** and **p** in a variable say **count**

and finally return **skipL** and **count** from the function in the same order.

Sample Input

```
1 straw plus berry is strawberry    # t
2 strawberry                        # p
```

Sample Output

```
1 0 9 18 19 20
2 14
```

Solution

```
1 def BMCount(t,p):
2     last = {} # Preprocess
3     for i in range(len(p)):
4         last[p[i]] = i
5     poslist,i, count = [], 0, 0 # Loop
6
7     while i <= (len(t)-len(p)):
8         matched,j = True,len(p)-1
9         poslist.append(i)
10        while j >= 0 and matched:
11            count += 1
12            if t[i+j] != p[j]:
```

```

13         matched = False
14         j = j - 1
15         if matched:
16             i = i + 1
17         else:
18             j = j + 1
19             if t[i+j] in last.keys():
20                 i = i + max(j-last[t[i+j]],1)
21             else:
22                 i = i + j + 1
23         return poslist, count

```

Suffix(Visible)

```

1 t = input()
2 p = input()
3 list, c = BMCount(t, p)
4 print(*list)
5 print(c)

```

Public Test Cases

Input 1

```

1 straw plus berry is strawberry
2 strawberry

```

Output

```

1 0 9 18 19 20
2 14

```

Input 2

```

1 lcetcxedt dfashoxdwkevf biztvrwh xqhjtxntfplur lpkrbpvgehojnkagvqla
2 ojnka

```

Output

```

1 0 5 10 15 16 21 26 31 34 36 41 46 51 55 56
2 19

```

Private Test Cases

Input 1

```

1 straw plus berry is strawberry
2 strawberry

```

Output

```
1 | 0 9 18 19 20
2 | 14
```

Input 2

```
1 | lcetcxedt dfashoxdwkevfbiztvrwh xqhjtxntfplurlpkrbpvgehojnkagvqla
2 | ojnka
```

Output

```
1 | 0 5 10 15 16 21 26 31 34 36 41 46 51 55 56
2 | 19
```

Input 3

```
1 | wrbbphtchnrpnhwqfbppqwweijdgosxztpekucgmgyDlkBfiqbjpgkfzklltmfpgptiuywcpkdzwd
   | bmylptppeerash
2 | yDlkBf
```

Output

```
1 | 0 6 12 18 24 30 32 38 42 43 49 50 56 61 67 69 75 78 84
2 | 26
```

Input 4

```
1 | wrbbphtchnrpnhwqfbppqwweijdgosxztpekucgmgyDlkBfiqbjpgkfzklltmfpgptiuywcpkdzwd
   | bmylptppeerash
2 | yDlkBf
```

Output

```
1 | 0 6 12 18 24 30 32 38 44 50 56 61 67 69 75 78 84
2 | 18
```

Input 5

```
1 | ncUQLceWAepfsxctbipkcskgsb nwqmbgpqbrvrhdcetncUQLceWAghvp nm1gaez
   | frtmpncUQLceWAdmvsrfnpagxvrj ggm bz1kezzyznqaxqhxyrcrkj
   | rrnhsrikkpnrucqbncniwncUQLceWAsdmjtfstwxswonhloqrojymfiz
2 | ncUQLceWA
```

Output

```
1 | 0 1 3 12 15 24 33 42 45 46 55 64 72 73 82 91 100 109 118 126 134 143 144 145
   | 154 163
2 | 58
```

Problem 2

You are given a string $s = s_1s_2 \dots s_n$, where n is the length of string s , and s_i is its i^{th} character. The prefix of the string s of length l ($1 \leq l < n$) is string $s_1s_2 \dots s_l$.

Write a function `PrefixMatch(s)` that accepts a string `s` and returns a list of all unique substrings of `s[1:]` that matches with any prefix of `s`.

Hint:- Use concept of KMP algorithm to write an efficient solution.

Sample Input 1

```
1 | ababa #s
```

Output

```
1 | ['a', 'ab', 'aba'] #substrings that matched with prefix of s
```

Explanation

All possible unique substring of `s[1:]` or `baba` are `b`, `a`, `ba`, `bab`, `baba`, `ab`, and `aba` but only `a`, `ab`, `aba` matched with prefix of `s`

Sample Input 2

```
1 | abbabbba
```

Output

```
1 | ['a', 'ab', 'abb']
```

Solution

```
1 | # Solution Code
2 | def kmp_fail(p):
3 |     m = len(p)
4 |     fail = [0 for i in range(m)]
5 |     j,k = 1,0
6 |     while j < m:
7 |         if p[j] == p[k]:
8 |             fail[j] = k+1
9 |             j,k = j+1,k+1
10 |        elif k > 0:
11 |            k = fail[k-1]
12 |        else:
13 |            j = j+1
14 |    return(fail)
15 |
16 | def PrefixMatch(s):
17 |     res=[]
18 |     if len(s)==0:
```

```
19         return res
20     m = kmp_fail(s)
21     for i in m:
22         if i != 0:
23             res.append(s[:i])
24     ss = list(set(res))
25     return ss
```

Suffix(Visible)

```
1 s = input()
2 print(sorted(PrefixMatch(s)))
```

Public Test case

Input 1

```
1 ababa
```

Output 1

```
1 ['a', 'ab', 'aba']
```

Input 2

```
1 abababbbbabababbbbaaaabbb
```

Output 2

```
1 ['a', 'ab', 'aba', 'abab', 'ababa', 'ababab', 'abababb', 'abababbb']
```

Input 3

```
1 abcdef
```

Output 3

```
1 []
```

Private Test Case

Input 1

```
1 abcdefghabc
```

Output 1

```
1 ['a', 'ab', 'abc']
```

Input 2

```
1 | aaaaaaaaaa
```

Output 2

```
1 ['a', 'aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaaaa', 'aaaaaaaa']
```

Input 3

```
1 abababababababababa
```

Output 3

```
1 ['a', 'ab', 'aba', 'abab', 'ababa', 'ababab', 'abababa', 'abababab',
  'ababababa', 'ababababab', 'abababababa', 'abababababab', 'ababababababa',
  'ababababababab', 'abababababababa', 'abababababababab', 'ababababababababa',
  'ababababababababab', 'abababababababababa', 'abababababababababab',
  'ababababababababababa']
```

Input 4

```
1 | abcdefghijklmnopqrstuvwxyz
```

Output 4

```
1 | ['a']
```

Input 5

```
1 | abcdeedcba
```

Output 5

```
1 | ['a']
```