Week-10 Programming Assignment

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

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

Sample Output

```
0 9 18 19 20
13
```

Solution

```
def BMCount(t,p):
  last = {} # Preprocess
  for i in range(len(p)):
    last[p[i]] = i
  poslist,i, count = [], 0, 0 # Loop

  while i <= (len(t)-len(p)):
    matched,j = True,len(p)-1
    poslist.append(i)
    while j > 0 and matched:
      count += 1
      if t[i+j] != p[j]:
        matched = False
      j = j - 1
    if matched:
      i = i + 1
    else:
      j = j + 1
      if t[i+j] in last.keys():
        i = i + max(j-last[t[i+j]],1)
      else:
        i = i + j + 1
  return poslist, count
```

Suffix(Visible)

```
t = input()
p = input()
list, c = BMCount(t, p)
print(*list)
print(c)
```

Public Test Cases

1

Input

```
straw plus berry is strawberry
strawberry
```

Output

```
0 9 18 19 20
13
```

2

Input

```
lcetcxedt dfashoxdwkevfbiztvrwh xqhjtxntfplurlpkrbpvgehojnkagvqla
ojnka
```

Output

```
0 5 10 15 16 21 26 31 34 36 41 46 51 55 56
18
```

Private Test Cases

1

Input

```
straw plus berry is strawberry
strawberry
```

Output

```
0 9 18 19 20
13
```

2

Input

```
lcetcxedt dfashoxdwkevfbiztvrwh xqhjtxntfplurlpkrbpvgehojnkagvqla
ojnka
```

Output

```
0 5 10 15 16 21 26 31 34 36 41 46 51 55 56
18
```

3

Input

```
wrbbphtchnrpnhwqfbppqwweijdgosxztpekucgmgfYDlkBfiqbjgkfzkklltmfpgptiuywcpkdzwd
bmylptppeaash
YDlkBf
```

Output

```
0 6 12 18 24 30 32 38 42 43 49 50 56 61 67 69 75 78 84
25
```

4

Input

```
wrbbphtchnrpnhwqfbppqwweijdgosxztpekucgmgfYDlkBfiqbjgkfzkklltmfpgptiuywcpkdzwd
bmylptppeaash
YdlkBf
```

Output

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

5

Input

```
ncUQLceWAepfsxctbipkcskgbsb nwqmbgpqbrvrhdcetncUQLceWAghvp nmlgaez
frtmpncUQLceWAdmvsrfnpagxvrj ggm bzlkezzyzznqaxqhxyrcrkj
rrnhsrikkpnrcuqbcniwncUQLceWAsdmjtfstwxswonhloqrojqymfiz
ncUQLceWA
```

Output

```
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
54
```

Problem 2

You are given a string $s = s_1 s_2 \ldots s_n$, 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 $s_1 s_2 .. s_l$.

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

```
ababa #s
```

Output

```
['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

```
abbabbba
```

Output

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

Solution

```
# Solution Code
def kmp_fail(p):
    m = len(p)
    fail = [0 for i in range(m)]
    j,k = 1,0
    while j < m:
        if p[j] == p[k]:
            fail[j] = k+1
            j,k = j+1,k+1
        elif k > 0:
            k = fail[k-1]
        else:
            j = j+1
    return(fail)

def PrefixMatch(s):
    res=[]
    if len(s)==0:
```

```
        return res
    m = kmp_fail(s)
    for i in m:
        if i != 0:
            res.append(s[:i])
    ss = list(set(res))
    return ss
```

Suffix(Visible)

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

Public Test case

Input 1

```
ababa
```

Output 1

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

Input 2

```
abababbbbbababababbbaaaabbb
```

Output 2

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

Input 3

```
abcdef
```

Output 3

```
[]
```

Private Test Case

Input 1

```
abcdefghabc
```

Output 1

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

Input 2

```
aaaaaaaaa
```

Output 2

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

Input 3

```
abababababababababababa
```

Output 3

```
['a', 'ab', 'aba', 'abab', 'ababa', 'ababab', 'abababa', 'abababab',
'ababababa', 'abababababa', 'ababababababa', 'abababababab', 'ababababababa',
'abababababababa', 'ababababababababa', 'abababababababab', 'ababababababababa',
'abababababababababab', 'ababababababababababa', 'abababababababababab',
'abababababababababababa']
```

Input 4

```
abcdefghijklmnopqrstuvwxyza
```

Output 4

```
['a']
```

Input 5

```
abcdeedcba
```

Output 5

```
['a']
```