""""

compute\_lps 函数用于计算模式字符串的LPS表。LPS表是一个数组，

其中的每个元素表示模式字符串中当前位置之前的子串的最长前缀后缀的长度。

该函数使用了两个指针 length 和 i，从模式字符串的第二个字符开始遍历。

"""

def compute\_lps(pattern):

"""

计算pattern字符串的最长前缀后缀（Longest Proper Prefix which is also Suffix）表

:param pattern: 模式字符串

:return: lps表

"""

m = len(pattern)

lps = [0] \* m # 初始化lps数组

length = 0 # 当前最长前后缀长度

for i in range(1, m): # 注意i从1开始，lps[0]永远是0

while length > 0 and pattern[i] != pattern[length]:

length = lps[length - 1] # 回退到上一个有效前后缀长度

if pattern[i] == pattern[length]:

length += 1

lps[i] = length

return lps

#lps的作用是在**ab**sdhjk**ab**中找到和后面的“ab”（后缀）相同的最长的前缀，相当于一个回溯

#pattern是目标串

def kmp\_search(text, pattern):

n = len(text)

m = len(pattern)

if m == 0:

return 0

lps = compute\_lps(pattern)

matches = []

# 在 text 中查找 pattern

j = 0 # 模式串指针

for i in range(n): # 主串指针

while j > 0 and text[i] != pattern[j]:

j = lps[j - 1] # 模式串回退

if text[i] == pattern[j]:

j += 1

if j == m:

matches.append(i - j + 1) # 匹配成功

j = lps[j - 1] # 查找下一个匹配

return matches

example：

text = "ABABABABCABABABABCABABABABC"

pattern = "ABABCABAB"

index = kmp\_search(text, pattern)

print("pos matched：", index)

# pos matched： [4, 13]

匹配步骤

i = 0, j = 0：

text[0] = "A"，pattern[0] = "A"。

匹配成功，j += 1，j = 1。

i = 1, j = 1：

匹配成功，j += 1，j = 2。

i = 2, j = 2：

匹配成功，j += 1，j = 3。

i = 3, j = 3：

匹配成功，j += 1，j = 4。

i = 4, j = 4：

text[4] = "A"，pattern[4] = "C"。

**匹配失败，回退 j = lps[j - 1] = lps[3] = 2。（此时把i的index为结尾的字符串看作是pattern之前那个开头“AB”的对应者，也就是在这一步保存有最大程度的匹配度，并期待在未来的对text的扫描中可以获得以后的部分）**

i = 4, j = 2：

text[4] = "A"，pattern[2] = "A"。

匹配成功，j += 1，j = 3。

……

text[12] = "B"，pattern[8] = "B"。

匹配成功，j += 1，j = 9。

模式串完全匹配，记录匹配位置 i - j + 1 = 12 - 9 + 1 = 4。

回退 j = lps[j - 1] = lps[8] = 4，继续查找下一个匹配。**（仍然是最大程度的利用）**

i = 13, j = 4：

ps，理解的核心是lps[i]表示到i为止（包含字符i）的字符串最大程度保留原有前缀后的index**+1（即lps[i]处的pattern[lps[i]]是下一个检测对象，不一定符合）**



