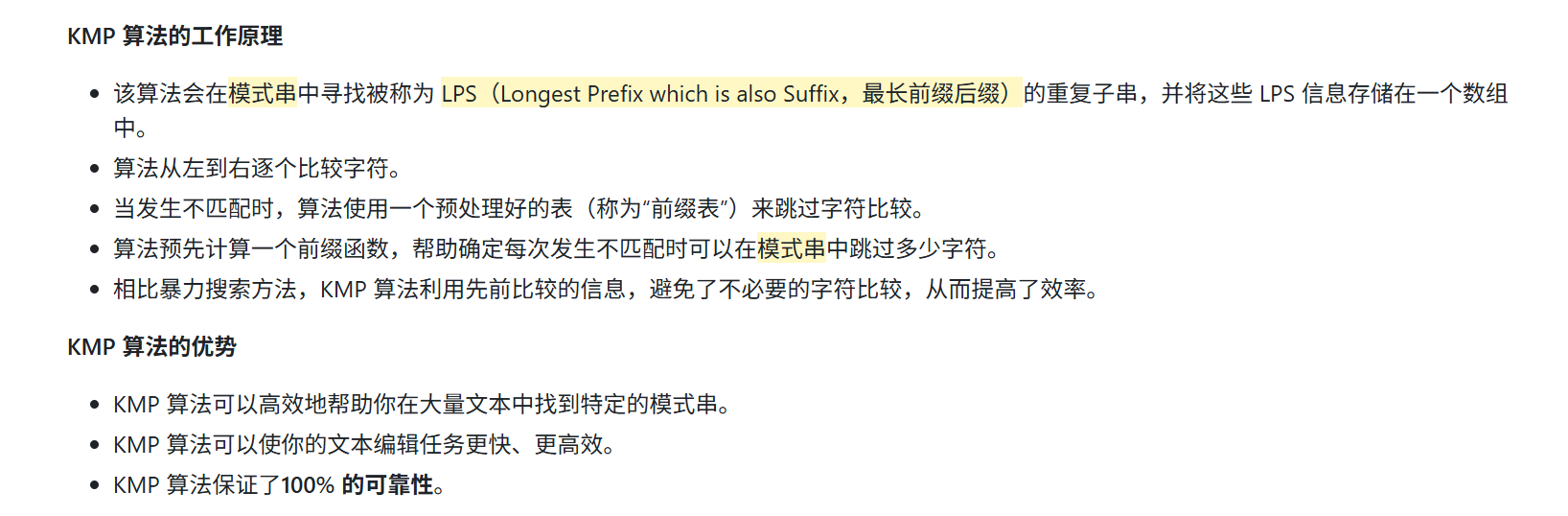
**Kmp:**



""""

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**

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

text = "ABABABABCABABABABCABABABABC"

pattern = "ABABCABAB"

index = kmp\_search(text, pattern)

print("pos matched：", index)

# pos matched： [4, 13]

'''

这是一个字符串匹配问题，通常使用KMP算法（Knuth-Morris-Pratt算法）来解决。

使用了 Knuth-Morris-Pratt 算法来寻找字符串的所有前缀，并检查它们是否由重复的子串组成，

如果是的话，就打印出前缀的长度和最大重复次数。

'''

# 得到字符串s的前缀值列表

**def kmp\_next(s):**

**# kmp算法计算最长相等前后缀**

**next = [0] \* len(s)**

**j = 0**

**for i in range(1, len(s)):**

**while s[i] != s[j] and j > 0:**

**j = next[j - 1]**

**if s[i] == s[j]:**

**j += 1**

**next[i] = j**

**return next**

def main():

case = 0

while True:

n = int(input().strip())

if n == 0:

break

s = input().strip()

case += 1

print("Test case #{}".format(case))

next = kmp\_next(s)

**for i in range(2, len(s) + 1):**

**k = i - next[i - 1] # 可能的重复子串的长度**

**if (i % k == 0) and i // k > 1:**

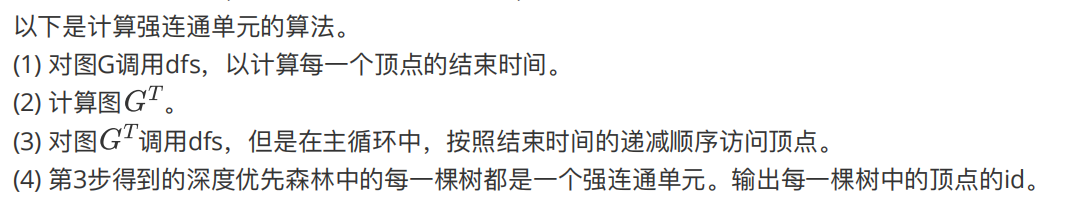
**print(i, i // k)**

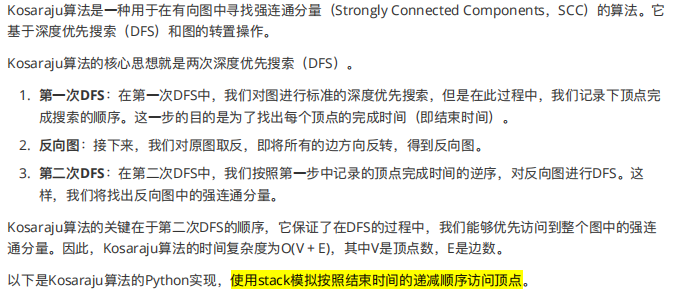
print()

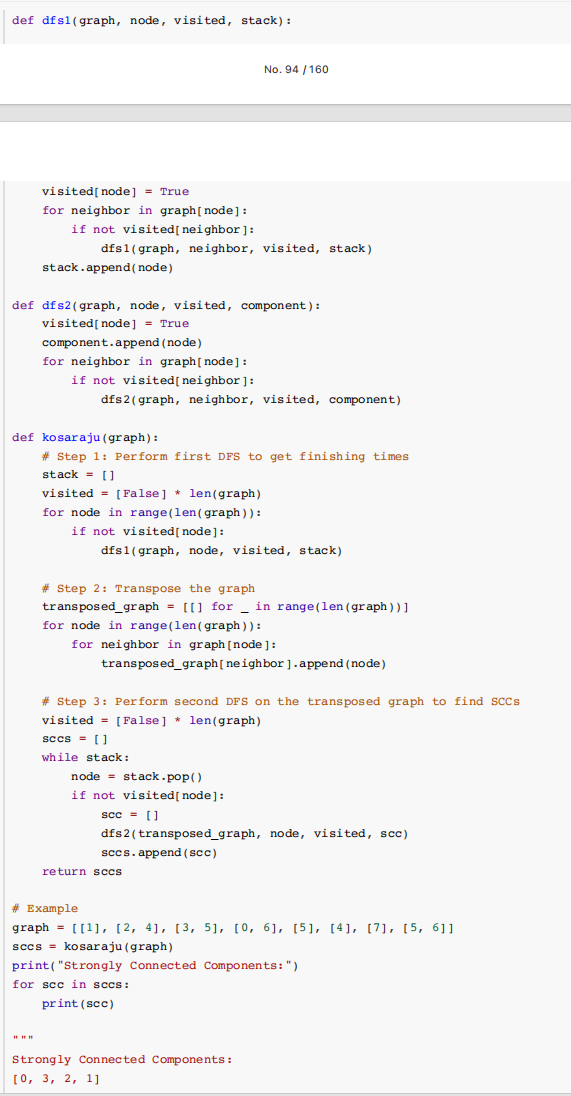
if \_\_name\_\_ == "\_\_main\_\_":

main()

**强联通单元：**







**算法2 ：**

