题目：

Consider the string `s` to be the infinite wraparound string of "abcdefghijklmnopqrstuvwxyz", so `s` will look like this: "...zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcd....".

Now we have another string `p`. Your job is to find out how many unique non-empty substrings of `p` are present in `s`. In particular, your input is the string `p` and you need to output the number of different non-empty substrings of `p` in the string `s`.

Note: `p` consists of only lowercase English letters and the size of p might be over 10000.

Example 1:

```
Input: "a"

Output: 1


Explanation: Only the substring "a" of string "a" is in the string s.
```

Example 2:

```
Input: "cac"

Output: 2

Explanation: There are two substrings "a", "c" of string "cac" in the string
 s.
```

Example 3:

```
Input: "zab"

Output: 6

Explanation: There are six substrings "z", "a", "b", "za", "ab", "zab" of st
ring "zab" in the string s.
```

思路：

这道题说有一个无限长的封装字符串，然后又给了我们另一个字符串 p，问我们 p 有多少非空子字符串在封装字符串中。我们通过观察题目中的例子可以发现，由于封装字符串是 26 个字符按顺序无限循环组成的，那么满足题意的 p 的子字符串要么是单一的字符，要么是按字母顺序的子字符串。这道题遍历 p 的所有子字符串会 TLE，因为如果 p 很大的话，子字符串很多，会有大量的满足题意的重复子字符串，必须要用到 trick，而所谓技巧就是一般来说你想不到的方法。我们看 abcd 这个字符串，以 d 结尾的子字符串有 abcd, bcd, cd, d，那么我们可以发现 bcd 或者 cd 这些以 d 结尾的字符串的子字符串都包含在 abcd 中 ,那么我们知道以某个字符结束的最大字符串包含其他以该字符结束的字符串的所有子字符串，说起来很拗口，但是理解了我上面举的例子就行。那么题目就可以转换为分别求出以每个字符(a-z)为结束字符的最长连续字符串就行了，我们用一个数组 cnt 记录下来 ,最后在求出数组 cnt 的所有数字之和就是我们要的结果.

1.时间：O（N^2）;空间：O（N）　->暴力

```
class Solution {
public:
    int findSubstringInWraproundString(string p) {
        if (p.size() < 2) return p.size();
        int count = 0;
        std::unordered_set<std::string> hashTable;
        for (int i = 0; i < p.size(); ++i){
            for (int k = i; k < p.size(); ++k){
                std::string substr;
                if (k == i){
                    substr = p.substr(i, 1);

                } else{
                    if ((p[k - 1] - 'a' + 1) % 26 != (p[k] - 'a')) break;
                    substr = p.substr(i, k - i + 1);
                }
                if (hashTable.find(substr) == hashTable.end()){
                    count++;
                    hashTable.insert(substr);
                }
            }
```

```
        }
        return count;
    }
};
```

2.时间：O (N) ; 空间：O (1)

```
class Solution {
public:
    int findSubstringInWraproundString(string p) {
        if (p.size() < 2) return p.size();
        int len = 0;
        std::vector<int> cnt(26, 0);
        for (int i = 0; i < p.size(); ++i){
            if (i > 0 && ((p[i - 1] - 'a') + 1) % 26 == (p[i] - 'a')){
                len++;
            } else{
                len = 1;
            }
            int &count = cnt[p[i] - 'a'];
            count = std::max(count, len);
        }
        return std::accumulate(cnt.begin(), cnt.end(), 0);
    }
```

```
};
```

3.时间：O（N）；空间：O（1） ->优化掉初始条件判断

```
class Solution {
public:
    int findSubstringInWraproundString(string p) {
        if (p.size() < 2) return p.size();
        std::vector<int> cnt(26, 0);
        cnt[p[0] - 'a']++;
        int len = 1;
        for (int i = 1; i < p.size(); ++i){
            if (((p[i - 1] - 'a') + 1) % 26 == (p[i] - 'a')){
                len++;
            } else{
                len = 1;
            }
            int &count = cnt[p[i] - 'a'];
            count = std::max(count, len);
        }
        return std::accumulate(cnt.begin(), cnt.end(), 0);
    }
};
```