686 Repeated String Match

2018年4月26日 15:48

Given two strings A and B, find the minimum number of times A has to be repeated such that B is a substring of it. If no such solution, return -1.

For example, with A = "abcd" and B = "cdabcdab".

Return 3, because by repeating A three times（"abcdabcdabcd"), B is a substring of it; and B is not a substring of A repeated two times ("abcdabcd").

Note:

The length of A and B will be between 1 and 10000.

来自 <https://leetcode.com/problems/repeated-string-match/description/>

给定两个字符串 A 和 B, 寻找重复叠加字符串A的最小次数，使得字符串B成为叠加后的字符串A的子串，如果不存在则返回 -1。

举个例子，A = "abcd"，B = "cdabcdab"。

答案为 3， 因为 A 重复叠加三遍后为 “abcdabcdabcd”，此时 B 是其子串；A 重复叠加两遍后为"abcdabcd"，B 并不是其子串。

注意:

A 与 B 字符串的长度在1和10000区间范围内。

Solution for Python3:

```python
class Solution:
    def repeatedStringMatch(self, A, B):
        """
        :type A: str
        :type B: str
        :rtype: int
        """
        # len(A)+len(B)<=q*len(A)
        # q>=1 + len(B)/len(A)
        # 考虑到AB长度相等时，q最少只需要1，而上述算出结果是2
        # 所以，将分子len(B)先减去1再对分子len(A)取整
        # q >= 1 + (len(B) -1)//len(A)
        q = (len(B) - 1) // len(A) + 1
        for i in range(2):
           if B in A * (q + i):
               return q + i
        return -1
```

Solution for C++:

```cpp
class Solution1 {
public:
    int repeatedStringMatch(string A, string B) {
        int q = (B.length() - 1) / A.length() + 1;
        string tmp = A;
        for (int i = 1; i < q; i++) {
            tmp += A;
        }
        if (tmp.find(B) != string::npos)
            return q;
        tmp += A;
        if (tmp.find(B) != string::npos)
            return q + 1;
        return -1;
    }
};

// KMP O(m + n)
// https://leetcode.com/problems/repeated-string-match/discuss/112570/C++-KMP-algo-o(m-+n)-detailed
//复习KMP算法
class Solution2 {
public:
    int repeatedStringMatch(string A, string B) {
        vector<int> kmp(B.size() + 1);
        for (int i = 1, j = 0; i < B.size();) {
            if (B[j] == B[i]) {
                kmp[i++] = ++j;
            } else {
                if (j == 0)
                    i++;
                else
                    j = kmp[j - 1];
            }
        }
        for (auto i = 0, j = 0; i < A.size(); i++, j = kmp[j-1]) {
            while (j < B.size() && A[(i+j) % A.size()] == B[j]) {
                printf("match i%d j%d\n", i, j);
                ++j;
            }
            if (j == B.size())
                return ceil((float)(i + j) / A.size());
            else
                printf("unmatch i%d j%d\n", i, j);
        }
        return -1;
    }
};
```