# 题目

实现strStr()函数。

给定一个haystack字符串和一个needle字符串，在haystack字符串中找出needle字符串出现的第一个位置 (从0开始)。如果不存在，则返回-1。

**示例 1:**

**输入:** haystack = "hello", needle = "ll"

**输出:** 2

**示例 2:**

**输入:** haystack = "aaaaa", needle = "bba"

**输出:** -1

**提示：**

1 <= haystack.length, needle.length <= 10^4

haystack 和 needle 仅由小写英文字符组成

**说明:**

当needle是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当needle是空字符串时我们应当返回0。这与C语言的strstr()以及Java的indexOf()定义相符。

# 分析

求子串问题：双指针+KMP。

## 方法一：双指针

思路：

可以使用双指针来实现这个算法。具体步骤如下：

1、初始化两个指针 i 和 j 分别指向 haystack 和 needle 的开头。

2、遍历 haystack 字符串，比较 haystack[i] 和 needle[j] 是否相等：

如果相等，继续比较下一个字符，即 i++ 和 j++。

如果不相等，重置 i 和 j，使得 i 回到上一次比较的位置的下一个字符，j 回到 needle 的开头。

3、如果 j 指针走到了 needle 的末尾，说明找到了匹配的子串，返回 i - j，即匹配子串在 haystack 中的起始位置。

4、如果遍历完 haystack 字符串仍未找到匹配的子串，则返回 -1。

在算法中，当当前字符不匹配时，需要将指针i回溯到上一次比较的位置的下一个字符。这里的i - j + 1实际上是在找到一个不匹配的位置后，将i回溯到上一次比较的位置的下一个字符的索引。

具体解释如下：

- i是在遍历haystack字符串时的指针，表示当前遍历到的位置。

- j是在比较haystack[i]和needle[j]时的指针，表示在needle字符串中的位置。

当发现不匹配时，需要回溯i的位置，以重新开始匹配。因为i是在遍历haystack时的指针，所以回溯到上一次比较的位置的下一个字符，即i - j + 1。

例如，假设当前i指向haystack中的位置10，而j指向needle中的位置3，并且在位置10发现了不匹配。那么i - j + 1就是10 - 3 + 1 = 8，表示要回溯到haystack中的位置8开始重新匹配。

代码：

class Solution {

public:

int strStr(string haystack, string needle) {

int i = 0, j = 0;

// 双指针

while (i < haystack.size() && j < needle.size()) {

if (haystack[i] == needle[j]) {

i++;

j++;

} else {

// 返回haystack的下一个位置:i-j+1

i = i - j + 1;

j = 0;

}

}

// 如果j为needle的大小则代表完全匹配了,返回haystack中的位置:i-j

if (j == needle.size()) {

return i - j;

}

return -1;

}

};

或：

class Solution {

public:

int strStr(string haystack, string needle) {

int i = 0;

int j = 0;

while(haystack[i]!='\0'&&needle[j]!='\0')

{

if(needle[j]==haystack[i])//判断是否相等

{

j++;

i++;

}

else//不相等退回开始的位置，i+1，j=0;

{

i = i - j + 1;

j = 0;

}

}

if(j == needle.length())//j为步长

return i-j;

return -1;

}

};

**复杂度分析：**

时间复杂度：最坏时间复杂度为O((N−L)L)，最优时间复杂度为O(N)。

空间复杂度：O(1)。

或：

思路：

你可以使用双指针的方法来解决这个问题。遍历 haystack 的每个字符，当找到与 needle 的第一个字符相匹配的字符时，记录当前位置，然后用另一个指针从 needle 的第二个字符开始继续匹配，直到完全匹配或者不匹配为止。

代码：

class Solution {

public:

int strStr(string haystack, string needle) {

int m = haystack.size(), n = needle.size();

if (n == 0) return 0; // 处理 needle 为空字符串的情况

for (int i = 0; i <= m - n; ++i) {

int j = 0;

while (j < n && haystack[i + j] == needle[j]) {

++j;

}

if (j == n) return i; // 完全匹配返回下标 i

}

return -1; // 未找到匹配项，返回 -1

}

};

这段代码中，`i` 是 haystack 中当前字符的下标，`j` 是 needle 中当前字符的下标。我们在 haystack 中找到与 needle 第一个字符匹配的位置后，使用 `j` 来继续比较 needle 的剩余字符是否与 haystack 中的相匹配。如果完全匹配，则返回当前位置 `i`；如果未找到匹配项，则返回 -1。

## 方法二：KMP

class Solution {

public:

int strStr(string haystack, string needle) {

if(!needle.size()) return 0;

if(!haystack.size()) return -1;

//先构造pattern

int j = -1, i = 0;//j在后面，i在前面

vector<int> b(needle.size() + 1);

b[i] = j;

while(i < needle.size())

{

while(j >= 0 && needle[i] != needle[j]) j = b[j];

i++, j++;

b[i] = j;

}

j = 0, i = 0; //j这回是text的， i是pattern的

while(j < haystack.size())

{

while(i >= 0 && needle[i] != haystack[j]) i = b[i];

i++, j++;

if(i == needle.size())

{

return j - needle.size();

}

}

return -1;

}

};