KMP算法（Leetcode第28题）

Computer Science / 26 / 0 / 发布于 9个月前

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

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。

```
int *getNext(const string &s) {
    int *next = new int[s.length()];
    next[0] = -1;
    if (s.length() == 1) return next;//如果子串只有一位，直接输出next数组
    next[1] = 0;//第二位固定是0，就是要回溯到首位
    int i = 1, j = 0;
    while (i < s.length() - 1) {//每次要确定的其实是i+1处不匹配，要移动到何处
        while (j > 0 && s[i] != s[j]) {
            j = next[j];//前缀和后缀匹配与子串和主串匹配很相似
        }
        if (s[i] == s[j]) {//如果这一位相等，前缀后移一位
            ++j;
        }
        ++i;//确定i+1处不匹配的情况
        next[i] = j;//如果是因为相等跳出递归，移动到j+1位，如果是因为递归到了0位跳出递归，移动到0位
    }
    return next;
```

```
}

int strStr(string haystack, string needle) {
    if (needle.empty()) return 0;
    int j = 0;
    int *next = getNext(needle);
    for (int i = 0; i < haystack.length(); ++i) {//遍历主串
        while (j > 0 && haystack[i] != needle[j]) {
            j = next[j];//递归调用next数组，直到回溯到0处或回溯到相等的位置
        }
        if (haystack[i] == needle[j]) {//如果这一位相等，子串对应位置后移
            ++j;
        }
        if (j == needle.length()) {//因为前面进行了后移操作，所以这里比较的是length而不是length-1
            return i - j + 1;
        }
    }
    return -1;
}
```

以下文字转载自：www.cnblogs.com/tangzhengyue/p/431...

作者：唐小喵

KMP 的 next 数组求法是很不容易搞清楚的一部分，也是最重要的一部分。我这篇文章就以我自己的感悟来慢慢推导一下吧！保证你看完过后是知其然，也知其所以然。

如果你还不知道 KMP 是什么，请先阅读上面的链接，先搞懂 KMP 是要干什么。

下面我们就来说说 KMP 的 next 数组求法。

KMP 的 next 数组简单来说，假设有两个字符串，一个是待匹配的字符串 strText, 一个是要查找的关键字 strKey。现在我们要在 strText 中去查找是否包含 strKey，用 i 来表示 strText 遍历到了哪个字符，用 j 来表示 strKey 匹配到了哪个字符。

如果是暴力的查找方法，当 strText [i] 和 strKey [j] 匹配失败的时候，i 和 j 都要回退，然后从 i-j 的下一个字符开始重新匹配。

而 KMP 就是保证 i 永远不回退，只回退 j 来使得匹配效率有所提升。它用的方法就是利用 strKey 在失配的 j 为之前的成功匹配的子串的特征来寻找 j 应该回退的位置。而这个子串的特征就是前后缀的相同程度。

所以 next 数组其实就是查找 strKey 中每一位前面的子串的前后缀有多少位匹配，从而决定 j 失配时应该回退到哪个位置。

我知道上面那段废话很难懂，下面我们看一个彩图：

这个图画的就是 strKey 这个要查找的关键字字符串。假设我们有一个空的 next 数组，我们的工作就是要在这个 next 数组中填值。

下面我们用数学归纳法来解决这个填值的问题。

这里我们借鉴数学归纳法的三个步骤（或者说是动态规划？）：

1. 初始状态
2. 假设第 j 位以及第 j 位之前的我们都填完了
3. 推论第 j+1 位该怎么填

初始状态我们稍后再说，我们这里直接假设第 j 位以及第 j 位之前的我们都填完了。也就是说，从上图来看，我们有如下已知条件：

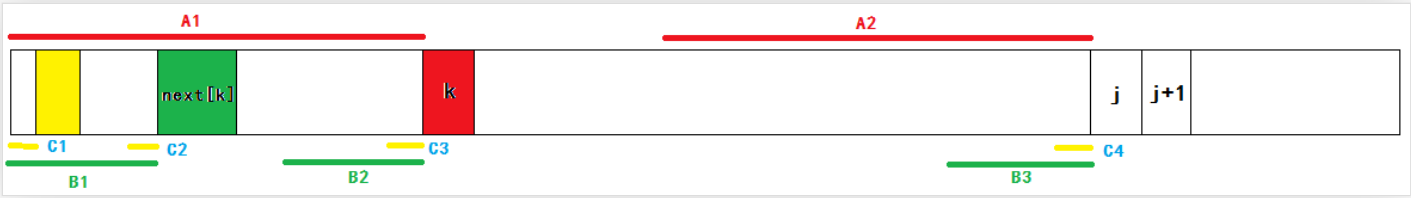
next[j] == k;

next [k] == 绿色色块所在的索引；

next [绿色色块所在的索引] == 黄色色块所在的索引；

这里要做一个说明：图上的色块大小是一样的（没骗我？好吧，请忽略色块大小，色块只是代表数组中的一位）。

我们来看下面一个图，可以得到更多的信息：



1. 由”next [j] == k;” 这个条件，我们可以得到 A1 子串 == A2 子串（根据 next 数组的定义，前后缀那个）。
2. 由”next [k] == 绿色色块所在的索引；” 这个条件，我们可以得到 B1 子串 == B2 子串。
3. 由”next [绿色色块所在的索引] == 黄色色块所在的索引；” 这个条件，我们可以得到 C1 子串 == C2 子串。
4. 由 1 和 2 (A1 == A2，B1 == B2) 可以得到 B1 == B2 == B3。
5. 由 2 和 3 (B1 == B2， C1 == C2) 可以得到 C1 == C2 == C3。
6. B2 == B3 可以得到 C3 == C4 == C1 == C2

上面这个就是很简单的几何数学，仔细看看都能看懂的。我这里用相同颜色的线段表示完全相同的子数组，方便观察。

接下来，我们开始用上面得到的条件来推导如果第 j+1 位失配时，我们应该填写 next [j+1] 为多少？

next [j+1] 即是找 strKey 从 0 到 j 这个子串的最大前后缀：

#：(#: 在这里是个标记，后面会用) 我们已知 A1 == A2，那么 A1 和 A2 分别往后增加一个字符后是否还相等呢？我们得分情况讨论：

1. 如果 str [k] == str [j]，很明显，我们的 next [j+1] 就直接等于 k+1。

用代码来写就是 next [++j] = ++k;

2. 如果 str [k] != str [j]，那么我们只能从已知的，除了 A1，A2 之外，最长的 B1，B3 这个前后缀来做文章了。

那么 B1 和 B3 分别往后增加一个字符后是否还相等呢？

由于 next [k] == 绿色色块所在的索引，我们先让 k = next [k]，把 k 挪到绿色色块的位置，这样我们就可以递归调用”#：”标记处的逻辑了。

由于 j+1 位之前的 next 数组我们都是假设已经求出来了的，因此，上面这个递归总会结束，从而得到 next [j+1] 的值。

我们唯一欠缺的就是初始条件了：

next[0] = -1, k = -1, j = 0

另外有个特殊情况是 k 为 - 1 时，不能继续递归了，此时 next [j+1] 应该等于 0，即把 j 回退到首位。

即 next [j+1] = 0; 也可以写成 next [++j] = ++k;