

讲解KMP算法

序言:

KMP解决的主要应用在**字符串匹配**上,其主要思想是: **当出现字符串不匹配时, 可以知道一部分之前已经匹配的文本内容, 可以利用这些信息避免重头再去匹配**

Next数组:(very important!!)

3、分析KMP算法里的next数组

next数组就是**前缀表(prefix table)**

4、前缀表的作用:

用来**回溯**的, 当**文本串与模板串发生不匹配**的时候, **模板串应该从哪里开始重新匹配**

5、什么是前缀表, 该如何记录?

前缀表的任务: 当某个字符匹配失败时, 前缀表会告诉你下一步匹配中, 模式串应该跳到哪个位置

6、在分析为什么要是前缀表而不是什么哈希表其他表等等, 偏偏要是前缀表

下表:

0 1 2 3 4 5

模式串:

a	a	b	a	a	f
---	---	---	---	---	---



如上图所示,文本串和模式串在下标5的位置 发生匹配失败, 由于下标5之前这部分字符串的**最长相等的前缀和后缀** 子字符串aa, 因此发生匹配失败的时候就 **将模式串的指针指向下标2即可**

7、推导前缀表

前提7.1

下表: 0 1 2 3 4 5

模式串:

a	a	b	a	a	f
---	---	---	---	---	---

前缀表:

0 1 0 1 2 0

代码随想录

前缀表中的元素含义表示: 当前位置之前的**子串(包括当前位置)**有多长**长度相同的前缀后缀**匹配过程7.2

当发生不匹配时候, 我们要看它的前一个字符前缀表的数值是多少, 然后将模式串的指针移到相应的下标位置

根据以上步骤获得前缀表即为next数组

匹配过程:

9、如何使用next数组来做一个匹配的过程

10、时间复杂度分析

其中n为文本串长度, m为模式串长度, 因为在匹配的过程中, 根据前缀表不断调整匹配的位置, 可以看出匹配的过程是 $O(n)$, 但之前还要单独生成next数组, 时间复杂度是 $O(m)$ (next数组的实现代码将在后续文章中继续讲解), 所以整个KMP算法的时间复杂度是 $O(n+m)$ 的。

例题: 在一个串中查找是否出现过另一个串, 这是KMP的看家本领

题目：28. 实现 strStr()

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。

在一个字符串中查找是否出现过一个字符串是KMP的看家本领,为了统一把haystack统称为文本串,把needle称为模式串

```
1 step1: 构造Next数组,实际就是计算模式串s,前缀表的问题
2 有两种情况s[i]与s[j]不相同,也就是遇到前后缀末尾不相同的情况,就要向前回溯
3 如果s[i]与s[j]相同,那么就同时向后移动i和j说明找到了相同的前后缀,同时还要将j(前缀的长度)赋值给next[i]
4 void getNext(int* next,const string& s){
5     int j=0;
6     next[0]=0;
7     for(int i=1;i<s.size();i++){
8         while(j>0&&s[i]!=s[j]){j=next[j-1];}
9         if(s[i]==s[j])j++;
10        next[i]=j;
11    }
12 }
13
14
15 int strStr(string haystack,string needle){//haystack为文本串,needle为模式串
16     if(needle.size()==0) return 0;
```

```

17  int next[needle.size()];
18  getNext(next, needle);
19
20  //i为指向文本串的指针,j为指向模式串的指针
21  int j=0;
22  for(int i=0;i<haystack.size();i++){//可以类比i指针为快指针
23  while(j>0&&haystack[i]!=needle[j]){
24  j=next[j-1];
25  }
26  if(haystack[i]==needle[j]) j++;
27  if(j==needle.size()){//匹配成功
28  return (i-needle.size+1);
29  //为什么要加1因为此时j+1后等于needle.size但是i并没有加一
30  //i停留在文本串中能够匹配的子串的最后一个字符位置
31  }
32  }
33  }

```

不瞒你说，重复子字符串问题，KMP很拿手

题目459.重复的子字符串

给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。

示例 1:

输入: "abab"

输出: True

解释: 可由子字符串 "ab" 重复两次构成。

示例 2:

输入: "aba"

输出: False

示例 3:

输入: "abcabcabcabc"

输出: True

解释: 可由子字符串 "abc" 重复四次构成。(或者子字符串 "abcabc" 重复两次构成。)

前缀、后缀以及最长公共前后缀的定义

前缀:不包含**最后一个字符**的所有以**第一个字符开头的连续子串**
后缀:不包含**第一个字符**的所有以**最后一个字符**结尾的**连续子串**

正式解题

假设next数组的长度为len即输入的字符串长度为len

那么整个字符串的最长前后缀的长度为next[len-1]

如果(next[len-1]!=0&&len%(len-next[len-1])==0)此时就有重复子串

解释: 为什么len%(len-next[len-1])==0就有重复子串?

假如一个字符串有n个子串substr组成, 那么**最长公共前后缀长度**为(n-1)*substr.size()

所以在有重复子串的字符串条件下, 一个子串的长度=len-最长公共前后缀长度=len-next[len-1],

所以又因为该字符串由n个子串substr组成, 此时一定满足len%(len-next[len-1])==0

```
1 class Solution {
2 public:
3     void getNext(int* next, const string& s){
4         int j=0;
5         next[j]=0;
6         for(int i=1;i<s.size();i++){
7             //当前后缀末尾不相等时
8             while(j>0&&s[j]!=s[i]) j=next[j-1];
9             if(s[j]==s[i])j++;
10            next[i]=j;
11        }
12    }
13    bool repeatedSubstringPattern(string s) {
14        if(s.size()==0) return false;
15        int next[s.size()];
16        getNext(next,s);
17        int len=s.size();
18        if(next[len-1]!=0&&(len%(len-next[len-1]))==0) return true;
19        return false;
20    }
21 };
```