# 基础数据结构--KMP目录

[一、KMP原理:](#_Toc15024_WPSOffice_Level1) [1](#_Toc15024_WPSOffice_Level1)

[二、 KMP整体模板 （下标从0开始的模板）](#_Toc21151_WPSOffice_Level1) [4](#_Toc21151_WPSOffice_Level1)

[三、 KMP题目](#_Toc17813_WPSOffice_Level1) [6](#_Toc17813_WPSOffice_Level1)

[四、 最小表示法](#_Toc7907_WPSOffice_Level1) [7](#_Toc7907_WPSOffice_Level1)

### 一、KMP原理:

T串主串 P串模式串

i

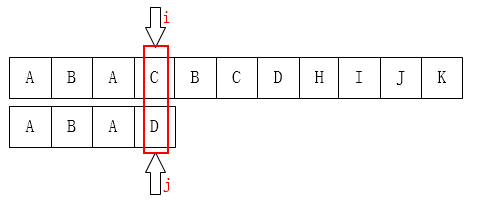
T串：A B C A B C D H I J K

P串： A B C E

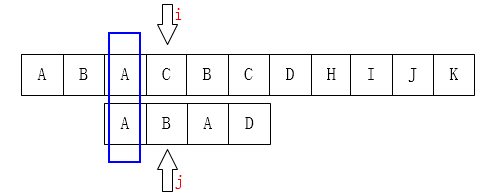
j

**保持i指针不回溯，通过修改j指针，让模式串尽量地移动到有效的位置**

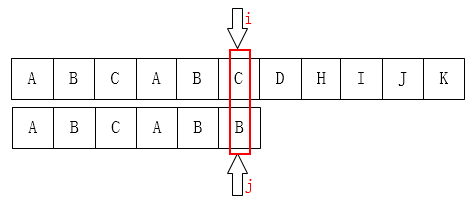
接下来我们自己来发现j的移动规律：



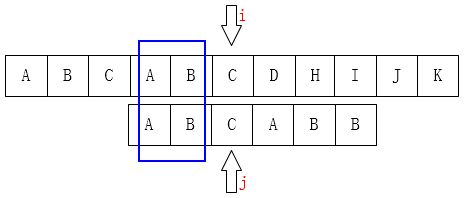
如图：C和D不匹配了，我们要把j移动到哪？显然是第1位。为什么？因为前面有一个A相同啊：



如下图也是一样的情况：



可以把j指针移动到第2位，因为前面有两个字母是一样的：



当匹配失败时，j要移动到下一个位置k。（模式串）存在着这样的性质：**最前面的k个字符和j之前的最后k个字符是一样的。**

**P[0~k-1] == P[j-k~j-1]**

**比如: a b c d a b c**    
**i == : 0 1 2 3 4 5 6  7**   
**next:-1 0 0 0 0 1 2**  3

**void get\_next() //初始化next数组 lenp为p数组的长度**

**{**

**int i,j;**

**Next[0] = j = -1;**

**i = 0;**

**while(i<lenp) //最后一位的判断其实是多余的**

**{**

**while(j!=-1&&p[j]!=p[i])**

**j = Next[j];**

**Next[++i] = ++j;**

**}**

**}**

**-----------------------------------------------------------------------------------------------------------------------**

**Kmp部分：**

i： 0 1 2 3 4 5 6 7 8 9

T串： a b c e a b p a b c

P串： a b c d a b c

Next[]： -1 0 0 0 0 1 2

当i=3处，j= 3 ，t[i]!=p[j],j的指针返回到next[3] 即 j = 0

i： 0 1 2 3 4 5 6 7 8 9

T串： a b c d a b p a b c

P串： a b c d a b c

Next[]： -1 0 0 0 0 1 2

若一直匹配至i=6；

j= 6，t[i]!=p[j],j返回到next[6] 移动j指针 即j=2

**int kmp1() //在t串找p串 返回下标**

**{**

**int i,j;**

**i = j = 0; //两个下标指针 i为主串的指针 j为模式串的指针**

**while(i<lent&&j<lenp)**

**{**

**while(j!=-1&&t[i]!=p[j])**

**j = Next[j];**

**i++;**

**j++;**

**}**

**if(j==lenp)**

**return i-j; //若找到返回开始下标（从0开始）**

**return -1; //找不到返回-1**

**}**

t[] aaaaaa

p[] aa

返回的ans = 5

**int kmp2() //返回匹配次数**

**{**

**int i,j;**

**i = j = 0;**

**int ans = 0;**

**while(i<lent)**

**{**

**while(j!=-1&&t[i]!=p[j])**

**j = Next[j];**

**if(j==lenp-1)**

**{**

**ans++;**

**j = Next[j];**

**}**

**i++;**

**j++;**

**}**

**return ans;**

**}**

t[] aaaaaaa

p[] aaa

返回的ans = 2

**int kmp3() //返回t串中有多少个p串**

**{**

**int i,j;**

**i = j = 0;**

**int ans = 0;**

**while(i<lent)**

**{**

**while(j!=-1&&t[i]!=p[j])**

**j = Next[j];**

**if(j==lenp-1)**

**{**

**ans++;**

**j = -1; // j指针改变为-1 然后++ 从0重新查找**

**}**

**i++;**

**j++;**

**}**

**return ans;**

**}**

**KMP算法的进一步优化 （2021数据结构考研复习指导 p117）**

void getNextVal(){ // KMP进一步优化  
 int i , j;  
 j = Next[0] = -1;  
  
 i = 0;  
 while(i < lenp){  
 while(j != -1 && p[i] != p[j])  
 j = Next[j];  
 i ++;  
 j ++;  
  
 if(p[i] != p[j]) Next[i] = j;  
 else Next[i] = Next[j];  
 }  
}

### KMP整体模板 （下标从0开始的模板）

1.在t串找p串  返回下标

2.返回匹配次数

3.返回t串中有多少个p串

#include <bits/stdc++.h>

**using** **namespace** std;

**int** Next[10000];

**char** t[10000];  //主串

**char** p[10000];  //模式串

**int** lenp,lent;

**void** get\_next()

{

**int** i,j;

    Next[0] = j = -1;

    i = 0;

**while**(i < lenp){  //最后一位的判断其实是多余的

**while**(j != -1 && p[j] != p[i])

            j = Next[j];

        Next[++i] = ++j;

    }

}

**int** kmp1(){  //在t串找p串  返回下标

**int** i , j;

    i = j = 0; //两个下标指针 i为主串的指针 j为模式串的指针

**while**(i < lent && j < lenp){

**while**(j!=-1&&t[i]!=p[j])

            j = Next[j];

        i++;

        j++;

    }

**if**(j==lenp) **return** i-j; //若找到返回开始下标（从0开始）

**return** -1;   //找不到返回-1

}

**int** kmp2(){ //返回匹配次数

**int** i , j;

    i = j = 0;

**int** ans = 0;

**while**(i < lent){

**while**(j != -1 && t[i] != p[j])

            j = Next[j];

**if**(j == lenp-1){

            ans++;

            j = Next[j];

        }

        i++;

        j++;

    }

**return** ans;

}

**int** kmp3() //返回t串中有多少个p串

{

**int** i,j;

    i = j = 0;

**int** ans = 0;

**while**(i<lent)

    {

**while**(j!=-1&&t[i]!=p[j])

            j = Next[j];

**if**(j==lenp-1)

        {

            ans++;

            j = -1;    // j指针改变为-1 然后++ 从0重新查找

        }

        i++;

        j++;

    }

**return** ans;

}

**int** main()

{

**scanf**("%s",t);

**scanf**("%s",p);

    lenp = **strlen**(p);

    lent = **strlen**(t);

    get\_next();

**int** ans = kmp3();

    cout<<ans<<endl;

**return** 0;

}

1. **KMP题目**

#### T1. Period （求最小循环元和循环次数）

**结论：当 i 能够整除i - next[i]时 , s[1 ~ i - next[i]]就是s[1~i]的最小循环元**

**i - next[i] 就是s[1~i]的循环元长度**

**最大循环次数是i /(i - next[i])**

代码：下标从1开始

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 1000005;

**char** s[maxn];

**int** Next[maxn] , n ,cas;

/\*

下标全部从1开始

s :      abababaac

next[] : 0 0 1 2 3 4 5 1 0

\*/

**void** get\_Next(){

    Next[1] = 0;

**for**(**int** i = 2 , j = 0 ; i <= n ; ++ i){

**while**(j > 0 && s[i] != s[j + 1]) j = Next[j];

**if**(s[i] == s[j + 1]) j ++;

        Next[i] = j;

    }

}

/\*

当 i 能够整除i - next[i]时,s[1 ~ i - next[i]]就是s[1~i]的最小循环元

i - next[i] 就是s[1~i]的循环元长度

最大循环次数是i/(i - next[i])

\*/

**int** main(){

**while**(**scanf**("%d",&n) && n){

**scanf**("%s",s + 1);

        n = **strlen**(s + 1);

        get\_Next();

**printf**("Test case #%d\n",++cas);

**for**(**int** i = 2 ; i <= n ; ++ i){

**if**(i % (i - Next[i]) == 0 && i /(i - Next[i]) > 1)

**printf**("%d %d\n",i, i / (i - Next[i]));

        }

**printf**("\n");

    }

**return** 0;

}

1. **最小表示法**

最小表示: 一个字符串S[1~n],如果我们不断把它的最后一个赐福放到开头，最终会得到n个字符串，称这n个字符串是循环同构的。这些字符串中字典序最小的一个称为字符串S的最小表示。

/\*

BZOJ 1398 (最小表示法)

题目：

给两个长度相等字符串

要求输出：

1.两个字符串是否循环同构 （YES NO）

2.如果同构，输出当前字符串的最小表示

样例：

Sample Input :

2234342423

2423223434

Sample Output :

Yes

2234342423

方法:

1.分别求出两个字符串的最小表示

2.通过比较两个字符串的最小表示判断两个字符串是否循环同构

3.按需输出

\*/

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** maxn = 1000005;

**int** len;

**char** s1[maxn \* 2] , s2[maxn \* 2];

**int** get\_min(**char** str[]){  //获取最小表示

**for**(**int** i = 1 ; i <= len ; ++  i) str[len + i] = str[i];

**int** i = 1 , j = 2 , k;

**while**(i <= len && j <= len){

**for**(k = 0 ; k <= len && str[i + k] == str[j + k] ; ++ k);

**if**(k == len) **break**; //str只由一个字符构成，如“aaaaaaa”

**if**(str[i + k] > str[j + k]){

            i = i + k + 1;

**if**(i == j) i ++;

        }**else** {

            j = j + k + 1;

**if**(i == j) j ++;

        }

    }

**return** min(i , j); //返回最小表示的下标

}

**int** main(){

**scanf**("%s%s",s1 + 1,s2 + 1);

    len = **strlen**(s1 + 1);

    //1. 求出两个字符串最小表示

**int** pos1 = get\_min(s1);

**int** pos2 = get\_min(s2);

**bool** flag = 0;

    //2.比较两个字符串的最小表示

**for**(**int** i = pos1 , j = pos2 , cnt = 1; cnt <= len ; ++ i, ++ j , ++ cnt){

**if**(s1[i] != s2[j]){

            flag = 1;

**break**;

        }

    }

**if**(flag) **printf**("No\n");

**else**{

**printf**("Yes\n");

**for**(**int** i = pos1 , cnt = 1; cnt <= len; ++ cnt , ++ i){

**printf**("%c",s1[i]);

        }

**printf**("\n");

    }

**return** 0;

}