TIMUS 1322 Spy Solution

write by Gestalti Lur
2012-10-03
題目鏈接：http://acm.timus.ru/problem.aspx?space=1&num=1322

題目大意

將一個長度爲 N(N<=10^5)的字符串的所有的循環字符串排列成一個字符串矩陣。將它的每一行按字典序排序之後取最後一列輸出。題目輸入數據爲這個輸出並且告訴你原始的字符串在排序之後的字符串矩陣的第幾行。要求輸出原始的字符串。

算法分析

題目中的轉換方式實際上是用於數據壓縮的 Burrows-Wheeler 变换[1]。題目要做的就是這個算法的逆過程。英文 wikipedia 上有一個樸素方法的演示[2]。Princeton 大學的一個編程作業的說明[3]和該題的樣例相同，並且快速還原原字符串的方法。

設原字符串爲 s，可以發現最後一列恰好包含 s 的所有字符。將這些字符按字典序排序後也就是排序後的字符串矩陣的第一列。也就是現在知道了第一列和最後一列，要還原 s。因爲這些字符串都是 s 的循環字符串，所以如果某一行的開始和結尾字符分別是 a,b,那麼將其向右移動一位之後的字符串應該是 ba…x。如果 b…x 這個字符串是唯一的，那麼其必然是 a…b 向右移動一位之後得到的字符串。但是如果有多個對應的情況，比如 s1 和 s2 都是形如 a…b 的字符串。若 s1<s2，那麼顯然向右移動一位之後的字符串也存在用同樣的關係，下面簡要證明之：

設 s1,s2 向右移動之後的字符串分別是 s1'和 s2'。s1'和 s2'除去 a,b 之後的子串也有和 s1,s2 相同的大小關係，因而 s1'和 s2'也具有和 s1,s2 相同的關係(因爲其前綴 ab 相同)。因而如果 s1<s2 則有 s1'<s2'。

所以如果按照順序尋找某個字符串右移後的字符串，其位置也應當符合該順序。比如題目描述中的 s11 和 s4 右移後的結果都可以轉移到 s10 和 s3,但是因爲上述關係，應當 s11 對應 s10，s4 對應 s3。

參考代碼中首先將某字符出現的位置按順序加入字符對應的鏈表中。再記錄 nxt[i]爲第 i 行的字符串右移之後得到的字符串。因爲之前的預處理過程，所以就可以用 O（1）的時間複雜度加入找到該字符串的位置。比如對於題目描述中的例子，預處理後有一個鏈表'r':10->11。按順序處理的時候 nxt[1] = 10，然後將鏈表'r'更新，到第 4 行的時候有 nxt[4] = 11,其他字符亦然。所以用 O(n)的時間複雜度就可以完成。

參考代碼

```
/*
  TIMUS 1322
  ACCEPTED
  2012-10-01
  2012-10-03
  gestapolur
*/
#include<cstdio>
```

1 http://zh.wikipedia.org/wiki/Burrows-Wheeler%E5%8F%98%E6%8D%A2
2 http://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform
3 http://www.cs.princeton.edu/courses/archive/spr03/cs226/assignments/burrows.html

```
#include<cstring>
#include<iostream>
#include<algorithm>
#define MAXN 100006
using std::sort;

int n;
char s[ MAXN ] , e[ MAXN ] , a[ MAXN ];
int next[ MAXN ] , head[ 129 ] , nxt[ MAXN ];
int st;

void solve()
{
 for( int i = 0 ; i < n ; head[ e[ i ] ] = next[ head[ e[ i ] ] ] , ++ i )
   nxt[ i ] = head[ e[ i ] ];
 for( int p = st - 1 , i = 0 ; i < n ; ++ i )
   a[ i ] = s[ p = nxt[ p ] ];
 for( int i = n - 1 ; i > -1 ; -- i ) putchar( a[ i ] );
 printf("\n");
 return ;
}

void init()
{
 scanf( "%d%s" , &st , e );
 n = strlen( e );
 strcpy( s , e );
 sort( s , s + n );
 memset( head , 255 , sizeof( head ) );
 for( int i = n ; i not_eq -1 ; -- i )
   {
    next[ i ] = head[ s[ i ] ];
    head[ s[ i ] ] = i;
   }
 return ;
}

int main()
{
 init();
 solve();
 return 0;
}
```