**进制转换**

#include<stdio.h>

#include<string.h>

const int MAXN = 1000;

int t[MAXN], A[MAXN];

char OldData[MAXN], NewData[MAXN]; // 转换前、后的数据

int olds, news; // 转换前、后的进制

void trans()

{

int i, len, k;

len = strlen(OldData);

for(i=len; i>=0; --i)

t[len-1-i] = OldData[i] - (OldData[i]<58 ? 48 : OldData[i]<97 ? 55 : 61);

for(k=0; len;)

{

for(i=len; i>=1; --i)

{

t[i-1] += t[i]%news\*olds;

t[i] /= news;

}

A[k++] = t[0] % news;

t[0] /= news;

while(len>0 && !t[len-1]) --len;

}

NewData[k] = NULL;

for(i=0; i<k; ++i)

NewData[k-1-i] = A[i] + (A[i]<10 ? 48 : A[i]<36 ? 55 : 61);

}

int main()

{

scanf("%d %d",&olds,&news);

scanf("%s",OldData);

trans();

printf("%s\n",NewData);

return 0;

}

**最大公共子序列：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

最大公共子序列：

最大公共子序列长度：int Lcs\_length( char \*str1, char \*str2, int \*\*c, int \*\*b)

输出最大公共子序列：void Print\_Lcs( char \*str, int \*\*b, int i, int j)

输出最大公共子序列长度及最大公共子序列：void Find\_Lcs( char \*str1, char \*str2)

时间复杂度：构建矩阵花费了O(MN)的时间，回溯时花费了O（M+N)的时间，两者相加最终花费了O(MN)的时间。

空间复杂度：构建矩阵花费了O(MN)的空间，标记函数也花费了O(MN)的空间，两者相加最终花费了O(MN)的空间。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define EQUAL 1 //EQUAL表示c[i][j]是由c[i-1][j-1]+1来的=====此时两个序列有相同的字符

#define UP 2 //UP表示c[i][j]是由c[i-1][j]来的============此时两个序列没有相同的字符

#define LEFT 3 //LEFT表示c[i][j]是由[ci][j-1]来的==========此时两个序列没有相同的字符

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数：int Lcs\_length( char \*str1, char \*str2, int \*\*c, int \*\*b)

输入： str1: 待比较字符串1

str2: 待比较字符串2

\*\*c: 储存最大公共子序列长度

\*\*b: 储存最大公共子序列检索路径

返回值：str1和str2最大公共子序列

时间复杂度：O(mn)

空间复杂度：O(mn)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Lcs\_length( char \*str1, char \*str2, int \*\*c, int \*\*b)

{

int len1 = strlen(str1),

len2 = strlen(str2);

int i,j;

for( i = 1; i <= len1; i++)

c[i][0] = 0;

for ( j = 0; j <= len2; j++)

c[0][j] = 0;

for( i = 1; i <= len1; i++)

for( j = 1; j <= len2; j++)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

使用i-1和j-1

算法导论书上写的是比较str1[i]和str[j]，但是算法导论书上的两个序列下标是由1开始的

这里使用i-1以及j-1是由于数组的下标从0开始

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if( str1[i-1] == str2[j-1] )

{

c[i][j] = c[i-1][j-1] + 1;

b[i][j] = EQUAL;

}

else if (c[i-1][j] >= c[i][j-1])

{

c[i][j] = c[i-1][j];

b[i][j] = UP;

}

else

{

c[i][j] = c[i][j-1];

b[i][j] = LEFT;

}

}

return c[len1][len2];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数：void Print\_Lcs( char \*str, int \*\*b, int i, int j

str: 待比较字符串1

\*\*b: 储存最大公共子序列检索路径

i: 待比较字符串1的长度

j: 待比较字符串2的长度

返回值：无

打印值：输出字符串1和字符串2的最长公共子序列

时间复杂度：O(m+n)

空间复杂度：O(m+n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Print\_Lcs( char \*str, int \*\*b, int i, int j)

{

if( i == 0 || j == 0)

return;

if( b[i][j] == EQUAL)

{

Print\_Lcs(str, b, i - 1, j - 1);

printf("%c ", str[i-1]);

}

else if ( b[i][j] == UP )

Print\_Lcs(str, b, i - 1, j);

else

Print\_Lcs(str, b, i , j - 1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数：void Find\_Lcs( char \*str1, char \*str2)

str1: 待比较字符串1

str2: 待比较字符串2

返回值：无

打印值：输出最大公共子序列长度以及最大公共子序列

时间复杂度：O(mn)

空间复杂度：O(mn)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Find\_Lcs( char \*str1, char \*str2)

{

int i,j,length;

int len1 = strlen(str1),

len2 = strlen(str2);

//申请二维数组

int \*\*c = (int \*\*)malloc(sizeof(int\*) \* (len1 + 1));

int \*\*b = (int \*\*)malloc(sizeof(int\*) \* (len1 + 1));

for( i = 0; i<= len1; i++ ) ////这个等号之前没加，导致内存泄漏

{

c[i] = (int \*)malloc(sizeof(int) \* (len2 + 1));

b[i] = (int \*)malloc(sizeof(int) \* (len2 + 1));

}

//将c[len1][len2]和b[len1][len2]初始化为0

for ( i = 0; i<= len1; i++)

for( j = 0; j <= len2; j++)

{

c[i][j] = 0;

b[i][j] = 0;

}

//计算LCS的长度

length = Lcs\_length(str1, str2, c, b);

printf("The number of the Longest-Common-Subsequence is %d\n", length);

//利用数组b输出最长子序列

printf("The Longest-Common-Subsequence is: ");

Print\_Lcs(str1, b, len1, len2);

printf("\n");

//动态内存释放

for ( i = 0; i <= len1; i++)

{

free(c[i]);

free(b[i]);

}

free(c);

free(b);

}

int main()

{

char x[10] = "aabcccc";

char y[10] = "ccaabbb";

Find\_Lcs(x,y);

system("pause");

}

KMP

/\*

POJ 3461 Oulipo

统计子串出现的次数

3

BAPC

BAPC

AZA

AZAZAZA

VERDI

AVERDXIVYERDIAN

\*/

#include<stdio.h>

#include<string.h>

#include<iostream>

using namespace std;

char W[10010],T[1000010];

int wlen,tlen;

int next[10010];

void getNext()

{

int j,k;

j=0;

k=-1;

next[0]=-1;

while(j<wlen)

{

if(k==-1||W[j]==W[k])

{

next[++j]=++k;

}

else k=next[k];

}

}

int KMP\_count()

{

int ans=0;

int i,j=0;

if(wlen==1&&tlen==1)

{

if(W[0]==T[0])return 1;

else return 0;

}

getNext();

for(i=0;i<tlen;i++)

{

while(j>0&&T[i]!=W[j])

j=next[j];

if(W[j]==T[i])j++;

if(j==wlen)

{

ans++;

j=next[j];

}

}

return ans;

}

int main()

{

freopen("in.txt","r",stdin);

freopen("out.txt","w",stdout);

int tcase;

scanf("%d",&tcase);

while(tcase--)

{

scanf("%s%s",&W,&T);

wlen=strlen(W);

tlen=strlen(T);

printf("%d\n",KMP\_count());

}

return 0;

}

**二分查找函数：**

函数**lower\_bound()**在first和last中的**前闭后开**区间进行二分查找，返回大于或等于val的**第一个元素**位置。如果所有元素都小于val，则返回**last**的位置

举例如下：

一个数组number序列为：4,10,11,30,69,70,96,100.设要插入数字3,9,111.pos为要插入的位置的下标

则

pos = lower\_bound( number, number + 8, 3) - number，pos = 0.即number数组的下标为0的位置。

pos = lower\_bound( number, number + 8, 9) - number， pos = 1，即number数组的下标为1的位置（即10所在的位置）。

pos = lower\_bound( number, number + 8, 111) - number， pos = 8，即number数组的下标为8的位置（但下标上限为7，所以返回最后一个元素的下一个元素）。

所以，要记住：函数lower\_bound()在first和last中的**前闭后开**区间进行二分查找，返回大于或等于val的**第一个元素**位置。如果所有元素都小于val，则返回**last**的位置，且last的位置是越界的！！~

返回查找元素的第一个可安插位置，也就是“元素值>=查找值”的第一个元素的位置

测试代码如下：

#include <iostream>

#include <algorithm>

#include <functional>

#include <vector>

using namespace std;

int main()

{

const int VECTOR\_SIZE = 8 ;

// Define a template class vector of int

typedef vector<int > IntVector ;

//Define an iterator for template class vector of strings

typedef IntVector::iterator IntVectorIt ;

IntVector Numbers(VECTOR\_SIZE) ;

IntVectorIt start, end, it, location ;

// Initialize vector Numbers

Numbers[0] = 4 ;

Numbers[1] = 10;

Numbers[2] = 11 ;

Numbers[3] = 30 ;

Numbers[4] = 69 ;

Numbers[5] = 70 ;

Numbers[6] = 96 ;

Numbers[7] = 100;

start = Numbers.begin() ; // location of first

// element of Numbers

end = Numbers.end() ; // one past the location

// last element of Numbers

// print content of Numbers

cout << "Numbers { " ;

for(it = start; it != end; it++)

cout << \*it << " " ;

cout << " }\n" << endl ;

// return the first location at which 10 can be inserted

// in Numbers

location = lower\_bound(start, end, 1) ;

cout << "First location element 10 can be inserted in Numbers is: "

<< location - start<< endl ;

}