**字符串相关算法问题**

**1问题： 将一个整数转换成字符串，要求不能使用系统调用。**

// 将整数转换成字符串

void hitoa(int num, char a[])

{

    int n;

    int ti = num;

    int i = 0, j;

    while (ti)  {

        a[i] = ti%10 + '0';      //取最后一个数，并转换成ASCII编码值保存到字符数组中

        i++;                //向后移动一位，保存下一个字符

        ti /= 10;             //取得已经去掉几位的整数

    }

    a[i] = '\0';                //这里一定要记住最后的'\0'

    i -= 1;                   //这里i也要注意是到最后一个非'\0'字符进行反转

    for (j = 0; j <= i; ) {                 //把得到的字符串反转一下

        n = a[i];

        a[i] = a[j];

        a[j] = n;

        i--;

        j++;

    }

    //printf("%s\n", a);

}

**2问题：反转字符串**

**开始时想到的反转算法非常简单，就是利用折半法，把前后对应位置的字符互换。但该算法没有考虑速度和空间的优化。**

//将给的字符串反转

void reverse1(char \*str)

{

    char \*p, \*p2;

    char c;

    p = str;                     //指向字符串的首部

    p2 = str + strlen(str) - 1;           //这里一定要注意，是指向最后一个非’\0‘字符

    while (p <= p2) {             //交换过程

        c = \*p;

        \*p = \*p2;

        \*p2 = c;

        p ++;

        p2 --;

    }

}

reverse1算法还可以在速度上进行优化：

//字符串反转算法2

void strrev3(char \*a)   
{  
    assert(NULL != a);    
         
    char \*h = a;  
    char \*t = a + strlen(a) - 1;

    while (h < t) {  
        \*h = \*h + \*t;     
        \*t = \*h - \*t;   
        \*h = \*h - \*t;   
        t--;  
        h++;      
    }     
}

//字符串反转算法3

void strrev2(char \*a)   
{  
    assert(NULL != a);    
         
    char \*h = a;  
    char \*t = a + strlen(a) - 1;

    while (h < t) {  
        \*h ^= \*t;     
        \*t ^= \*h;   
        \*h ^= \*t;   
        t--;  
        h++;      
    }     
}

**3问题：字符串拷贝**

// 把src中的字符串复制到dst的空间中

void tcpy(char \*dst, const char \*src)

{

    assert(NULL != dst && NULL != src);  //这里使用断言可以增加程序的可靠性，非常必要

    while('\0' !=\*src)                                         //不要偷懒哦，

        \*dst++ = \*src++;

    \*dst='\0';                                                       //前面这样写了后，这里必须这么处理

}

**4问题：字符子串删除**

         如a = “students”; b=”st”;

从a中删除b中存在的任一一个字符，得到的结果是:uden。

// 从a中删除r中存在的任一字符

void tremove(char a[], char r[])

{

    register char \*p;

    char \*p2;

int ex;

char\*pdst=a;                                               //防止改变a的指针值

    for (p = a; '\0' != \*p; p++) {

        ex = 0;

        for (p2 = r; '\0' != \*p2; p2 ++) {  //在r中查找a中的每一个字符是否存在，若r

            if (\*p2 == \*p) {    //中不存在，就要保留该字符，如r中存在就要删除该字符

                ex = 1;

                break;

            }

        }

        if (0 == ex)                      //若r中不存在该字符，保留该字符，主意这里不需要重新

            \*pdst++ = \*p;                    //不需要开启一个新的缓冲区

    }

    \*pdst = '\0';

}

// 删除子串的第2版本

void del\_sub(char str[], char sub[])

{

    char \*p;

    char \*ps = str;

    for (p = str; '\0' != \*p; p++) {   //遍历母字符串的每一个字符

        if (NULL == strchr(sub, \*p)) {  //若在sub中不存在，则保存起来，否则删除之

            \*ps++ = \*p;

        }

    }

    \*ps = '\0';

}

但从算法效率来看，该算法的时间复杂度为O(n\*m)，那么有没有更好的解决方案呢？

可以借鉴散列的思想来优化，由于只有128个ASCII字符编码，可以从这里入手来解决这个问题。

//删除子串的第3个版本

//O(n+m)

void del\_sub\_v3(char \*str, char \*sub)

{

    char \*p;

    int i, j;

    int asc[128] = {0};             //假设都是ACSII字符

    for (p = sub; '\0' != \*p; p++) {    //先给要删除的字符设置位1

        asc[\*p + '\0'] = 1;

    }

    for (i = 0, j = 0; i < strlen(str); i++) {  //遍历母串，把不删的字符复制给母串，

        if (!asc[str[i]-'\0']) {   //不需要临时空间。

            str[j] = str[i];    //注意遍历时，是strlen而不用减1了，细心点。

            j++;

        }

    }

    str[j] = '\0';      //这里要记住

}

**5问题：在一个字符串str中查找一个子串sub，不使用任何的系统调用。**

// 在str中找到sub的字符串

char \*hstrstr(char \*str, char \*sub)

{

    assert(NULL != str);

    char \*p, \*p2, \*p1;

    for (p = str; '\0' != \*p; p++) {            //遍历str

        for (p2 = sub, p1 = p; ; p2++, p1++) {  //遍历sub子串，并从str的现在位置开始匹配

            if ('\0' == \*p2)                //已经匹配到子串的末尾，说明已经匹配完全

                return p;

            if (\*p2 != \*p1)    //本次匹配完成，让str向前移动一位开始匹配，

                break;                //这里还可以优化关键在于游标的选择

        }

    }

    return NULL;

}

**6问题：实现一个连接两个字符的函数。**

char \*hstrcat(char \*dst, char \*s2)

{

    assert(NULL != dst);

    char \*p;

    p = dst + strlen(dst);      //p此时指向dst最后一个字符'\0'

    while ('\0' != \*s2) {

        \*p++ = \*s2++;           //把s2的字符串接到dst的后面

    }

    \*p = '\0';                  //千万不要忘记

    return dst;

}

**7问题：编写一个高效的函数，找到字符串中首个非重复字符。例如：total 中的 首个非重复字符是 o，teeter是 r。讨论算法的效率。**

解答：此问题是字符串操作的经典问题，考察的主要是查找的效率和实现的思路。可以使用一般的思路，但最坏的时间复杂度是O(n^2)，显然不是好的解法。

其高效的解法有几种：

（1）借鉴了perl的可以用字符作为数组的下标的思想。也可以用一个数据结构来表示：

struct hnode {

         int times;  //次数

         char c;      //单个字符

};

（2）使用树来解。

//查找一个字符串中的首个非重复字符

int find\_norepeat\_v1(char \*str)

{

    int asc[256] = {0};         //acsii 编码共256个字符值

    char \*p = str;

    int i;

    while ('\0' != \*p) {

        asc['\0' + \*p] += 1;;   //遍历字符串，并统计每个字符的重复次数

        p++;

    }

    for (i = 0; i < 256; i++) { //查找重复次数是1的ascii编码值,也就找到了该字符

        printf("%c times=[%d]\n", i-'\0', asc[i]);

        if (asc[i] == 1) {

            putchar(i-'\0');

            return 1;

        }

    }

    return 0;

}

从上面可以看出，此解法的好处是，时复杂度为O(2n)，但对于太长的字符串来说不是特别的好，此时需要使用其他的数据结构来解决。

**8问题：查找两个字符串的公共子串。**

/\*  
 \* 找出一个两个字符串中的最大公共子串  
 \*/  
  
#include "all.h"  
  
/\*  
  s1 : 横向  
  s2 : 纵向  
  \*/  
  
char \*find\_lcs(char \*s1, char \*s2)  
{  
    int xlen = strlen(s1);    
    int ylen = strlen(s2);    
    int i, j, len = 0, end = 0;  
    char \*p;  
    int start;  
  
    char \*c = (char \*)malloc(ylen);  
    if (!c)  
        return NULL;  
    memset(c, 0, ylen);  
  
    for (i = 0; i <xlen; i++)="" {  
        for (j = ylen - 1; j >= 0; j--) { //注意y必须从最大开始，否则会覆盖以前的结果  
            if (s1[i] == s2[j]) {       //找到一个相等的字符  
                if (i == 0 || j == 0)     
                    c[j] = 1;  
                else  
                    c[j] = c[j-1] + 1;  
            } else {  
                c[j] = 0;  
            }  
  
            if (c[j] > len) {   //记录最长字符长度  
                len = c[j];    //c[j] 中保存的是公共子串的长度  
                end = j;   //记录最长字符串的最后一个字符，用它来计算起始位置  
            }  
        }  
    }     
      
    start=end-len+1;    //计算起始位置     
    p =(char\*)malloc(len+1); //数组p纪录最长公共子串  
    for(i=start; i<=end;i++)  
        p[i-start] = s2[i];  
  
    p[len]='\0';  
    return p;  
}  
  
  
int main(void)  
{  
    char s1[] = "21232523311324";  
    char s2[] = "312123223445";  
  
    printf("%s\n", find\_lcs(s1, s2));     
  
    return 0;  
}

取s1中的一个字符与s2中的所有字符比较，并置c数组的标志位.

数组c的变化情况为：  
 0 0 1 0 1 0 1 1 0 0 0 0  
 0 1 0 2 0 0 0 0 0 0 0 0  
 0 0 2 0 3 0 1 1 0 0 0 0  
 1 0 0 0 0 4 0 0 2 0 0 0  
 0 0 1 0 1 0 5 1 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 1  
 0 0 1 0 1 0 1 1 0 0 0 0  
 1 0 0 0 0 2 0 0 2 0 0 0  
 1 0 0 0 0 1 0 0 1 0 0 0  
 0 2 0 1 0 0 0 0 0 0 0 0  
 0 1 0 1 0 0 0 0 0 0 0 0  
 1 0 0 0 0 1 0 0 1 0 0 0  
 0 0 1 0 1 0 2 1 0 0 0 0  
 0 0 0 0 0 0 0 0 0 1 1 0

9问题：求一个字符串s**的最大连续递增数字子串。**

比如：s="abcdefg1234efg345";   的结果是 r="1234"。

int inc\_str\_con(char \*str, char \*result)  
{  
    int len = strlen(str);      
    int n = 1, start = 0, end = 0;  
    int i, maxl = 0;  
  
    for (i = 0; i < len; i++) {  
        if (str[i] >= '0' && str[i] <= '9' &&    
            str[i+1] >= '0' && str[i+1] <= '9' &&    
            str[i] == str[i+1]-1) {  
            n++;  
        } else {  
            if (n > maxl) {  
                maxl = n;  
                end = i;  
                n = 1;  
            }     
        }     
    }     
    //printf("maxl=[%d], end=[%d]\n", maxl, end);  
    if (maxl == 0)  
        goto end;  
    start = end - maxl + 1;  
    strncpy(result, str+start, maxl);  
    fprintf(stderr, "findit=[%s]\n", result);  
    return 0;  
  
end :  
    return 1;  
}

10字符串原地压缩。题目描述：“eeeeeaaaff" 压缩为 "e5a3f2"。  
/\*  
 \* 字符串压缩算法，把s字符串压缩处理后结果保存在res中  
 \* O(n)  
 \*/  
char \*tarstr(char \*s, char \*res)  
{  
    assert(s != NULL);

    char \*p = s;  
    int len = strlen(s);  
    int i, rl = 0;  
    char c='\0';  
    int j = 0;  
      
    for (i = 0; i < len; i++) {  
        if (p[i] == p[i+1]) {    //找到一个重复字符     
            if (rl == 0)         //若是第一次重复，计数从1开始  
                rl = 1;  
            rl++;  
            c = p[i];            //把重复字符复制给c  
        } else {  
            if (rl > 0) {        //若重复个数大于0  
                res[j++] = rl + '0';  //记录重复次数  
                res[j++] = c;         //重复字符  
                rl = 0;               //置0  
                c = '\0';             //把字符置空  
            } else {  
                res[j++] = p[i];      //不是重复字符，直接复制给结果字符  
            }     
        }     
    }

    res[j] = '\0';                    //结束字符串  
    printf("res=[%s]\n", res);

    return res;  
}