01.02 字符串包含

题目描述

给定两个分别由字母组成的字符串A和字符串B，字符串B的长度比字符串A短。请问，如何最快地判断字符串B中所有字母是否都在字符串A里？

为了简单起见，我们规定输入的字符串只包含大写英文字母，请实现函数bool StringContains(string &A, string &B)

比如，如果是下面两个字符串：

String 1：ABCD

String 2：BAD

答案是true，即String2里的字母在String1里也都有，或者说String2是String1的真子集。

如果是下面两个字符串：

String 1：ABCD

String 2：BCE

答案是false，因为字符串String2里的E字母不在字符串String1里。

同时，如果string1：ABCD，string 2：AA，同样返回true。

分析与解法

题目描述虽长，但题意很明了，就是给定一长一短的两个字符串A，B，假设A长B短，要求判断B是否包含在字符串A中。

初看似乎简单，但实现起来并不轻松，且如果面试官步步紧逼，一个一个否决你能想到的方法，要你给出更好、最好的方案时，恐怕就要伤不少脑筋了。

解法一

判断string2中的字符是否在string1中?最直观也是最简单的思路是，针对string2中每一个字符，逐个与string1中每个字符比较，看它是否在String1中。

代码可如下编写：

```
bool StringContain(string &a,string &b)
{
    for (int i = 0; i < b.length(); ++i) {
        int j;
        for (j = 0; (j < a.length()) && (a[j] != b[i]); ++j)
            ;
        if (j >= a.length())
        {
```

```
                return false;
            }
        }
        return true;
    }
```

假设n是字符串String1的长度，m是字符串String2的长度，那么此算法，需要O(n*m)次操作。显然，时间开销太大，应该找到一种更好的办法。

解法二

如果允许排序的话，我们可以考虑下排序。比如可先对这两个字符串的字母进行排序，然后再同时对两个字串依次轮询。两个字串的排序需要(常规情况)O(m log m) + O(n log n)次操作，之后的线性扫描需要O(m+n)次操作。

关于排序方法，可采用最常用的快速排序，参考代码如下：

```
//注意A B中可能包含重复字符，所以注意A下标不要轻易移动。这种方法改变了字符串。如不想改变请自己复制
bool StringContain(string &a,string &b)
{
    sort(a.begin(),a.end());
    sort(b.begin(),b.end());
    for (int pa = 0, pb = 0; pb < b.length();)
    {
        while ((pa < a.length()) && (a[pa] < b[pb]))
        {
            ++pa;
        }
        if ((pa >= a.length()) || (a[pa] > b[pb]))
        {
            return false;
        }
        //a[pa] == b[pb]
        ++pb;
    }
    return true;
}
```

解法三

有没有比快速排序更好的方法呢？

我们换一种角度思考本问题：

假设有一个仅由字母组成字串，让每个字母与一个素数对应，从2开始，往后类推，A对应2，B对应3，C对应5，……。遍历第一个字串，把每个字母对应素数相乘。最终会得到一个整数。

利用上面字母和素数的对应关系，对应第二个字符串中的字母，然后轮询，用每个字母对应的素数除前面得到的整数。如果结果有余数，说明结果为false。如果整个过程中没有余数，则说明第二个字符串是第一个的子集

了（判断是不是真子集，可以比较两个字符串对应的素数乘积，若相等则不是真子集）。

思路总结如下：

1. 按照从小到大的顺序，用26个素数分别与字符'A'到'Z'一一对应。
2. 遍历长字符串，求得每个字符对应素数的乘积。
3. 遍历短字符串，判断乘积能否被短字符串中的字符对应的素数整除。
4. 输出结果。

如前所述，算法的时间复杂度为O(m+n)的最好的情况为O(n)（遍历短的字符串的第一个数，与长字符串素数的乘积相除，即出现余数，便可退出程序，返回false），n为长字串的长度，空间复杂度为O(1)。

```
//此方法只有理论意义，因为整数乘积很大，有溢出风险
bool StringContain(string &a,string &b)
{
    const int p[26] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
59,61, 67, 71, 73, 79, 83, 89, 97, 101};
    int f = 1;
    for (int i = 0; i < a.length(); ++i)
    {
        int x = p[a[i] - 'A'];
        if (f % x)
        {
            f *= x;
        }
    }
    for (int i = 0; i < b.length(); ++i)
    {
        int x = p[b[i] - 'A'];
        if (f % x)
        {
            return false;
        }
    }
    return true;
}
```

此种素数相乘的方法看似完美，但缺点是素数相乘的结果容易导致整数溢出。

解法四

如果面试官继续追问，还有没有更好的办法呢？计数排序？除了计数排序呢？

事实上，可以先把长字符串a中的所有字符都放入一个Hashtable里，然后轮询短字符串b，看短字符串b的每个字符是否都在Hashtable里，如果都存在，说明长字符串a包含短字符串b，否则，说明不包含。

再进一步，我们可以对字符串A，用位运算（26bit整数表示)计算出一个"签名"，再用B中的字符到A里面进行查找。

```
// “最好的方法”，时间复杂度O(n + m)，空间复杂度O(1)
bool StringContain(string &a,string &b)
{
    int hash = 0;
    for (int i = 0; i < a.length(); ++i)
    {
        hash |= (1 << (a[i] - 'A'));
    }
    for (int i = 0; i < b.length(); ++i)
    {
        if ((hash & (1 << (b[i] - 'A'))) == 0)
        {
            return false;
        }
    }
    return true;
}
```

这个方法的实质是用一个整数代替了hashtable，空间复杂度为O(1)，时间复杂度还是O(n + m)。

举一反三

1、变位词

- 如果两个字符串的字符一样，但是顺序不一样，被认为是兄弟字符串，比如bad和adb即为兄弟字符串，现提供一个字符串，如何在字典中迅速找到它的兄弟字符串，请描述数据结构和查询过程。