



编程学习网站



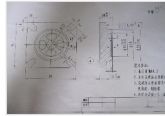
1t固态硬盘价格



app开发报价单



达内的



编程学习入门



app外包



win7旗舰版下载



c++培训

[首页](#) > [程序开发](#) > [软件开发](#) > [C++](#) > 正文

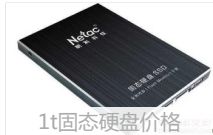
LeetCode:Substring with Concatenation of All Words (summarize)

2014-06-26

0 条评论

收藏

 我要投稿



[编程学习入门](#) [酒店式公寓月租](#) [c++培训](#)
[c++入门教程](#) [lol代练价格表](#) [哈尔滨到成都](#)

[计算机编程入门](#) [ios培训班](#) [app外包](#)
[ios工程师月薪](#) [黑马程序员培训](#)

You are given a string, S, and a list of words, L, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

For example, given:

S: "barfoothefoobarman"

L: ["foo", "bar"]

You should return the indices: [0,9].

(order does not matter).

算法1

暴力解法，从字符串s的每个位置都判断一次（如果从当前位置开始的子串长度小于L中所有单词长度，不用判断），从当前位置开始的子串的前段部分能不能由集合L里面的单词拼接而成。

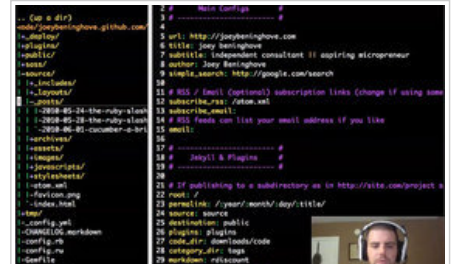
从某一个位置 i 判断时，依次判断单词s[i, i+2], s[i+3, i+5], s[i+6, i+8]...是否在集合中，如果单词在集合中，就从集合中删除该单词。

我们用一个hash map来保存单词，这样可以在O(1)时间内判断单词是否在集合中

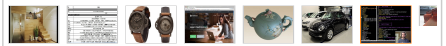
算法的时间复杂度是O (n*(1*k)) n是字符串的长度，1是单词的个数，k是单词的长度

递归代码如下：

```
class Solution {
private:
    int wordLen;
```



编程



[文章](#) [读书](#)

- Win2000下关闭无用端口
- 禁止非法用户登录综合设置 [win9x篇]
- 关上可恶的后门——消除NetBIOS隐患
- 网络入侵检测系统
- 潜伏在Windows默认设置中的陷阱
- 调制解调器的不安全
- 构建Windows 2000服务器的安全防护林
- SQL Server 2000的安全配置



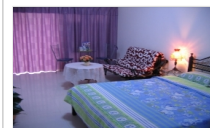
邮箱订阅 红黑联盟 精彩内容

立即订阅

点击排行

- 利用栈实现十进制数制转换成其他进制
- 图的m着色问题(回溯)
- 创建和使用静态库Lib
- 比较两个字符串str1和str2的大小
- 走迷宫
- Dev-C++创建使用类(Class)工程举例
- BMP图像旋转——C++实现
- C++统计代码注释行数 & 有效代码行数

今日头条



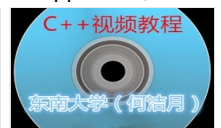
酒店式公寓月租



app开发报价单



达内的



c++入门教程

新闻排行榜

天

```

public:
    vector<int> findSubstring(string S, vector<string> &L) {
        unordered_map<string, int>wordTimes;
        for(int i = 0; i < L.size(); i++)
            if(wordTimes.count(L[i]) == 0)
                wordTimes.insert(make_pair(L[i], 1));
            else wordTimes[L[i]]++;
        wordLen = L[0].size();

        vector<int> res;
        for(int i = 0; i <= (int)(S.size()-L.size()*wordLen); i++)
            if(helper(S, i, wordTimes, L.size()))
                res.push_back(i);
        return res;
    }

//判断子串s[index...]的前段是否能由L中的单词组合而成
bool helper(string &s, const int index,
            unordered_map<string, int>&wordTimes, const int wordNum)
{
    if(wordNum == 0)return true;
    string firstWord = s.substr(index, wordLen);
    unordered_map<string, int>::iterator ite = wordTimes.find(firstWord);
    if(ite != wordTimes.end() && ite->second > 0)
    {
        (ite->second)--;
        bool res = helper(s, index+wordLen, wordTimes, wordNum-1);
        (ite->second)++;//恢复hash map的状态
        return res;
    }
    else return false;
}

};

```

非递归代码如下:

```

class Solution {
private:
    int wordLen;

public:
    vector<int> findSubstring(string S, vector<string> &L) {
        unordered_map<string, int>wordTimes;
        for(int i = 0; i < L.size(); i++)
            if(wordTimes.count(L[i]) == 0)
                wordTimes.insert(make_pair(L[i], 1));
            else wordTimes[L[i]]++;
    }

```

- 1 Win10小马 原版镜像激活工具 永
- 2 KMS通用激活工具v2016.05.
- 3 Microsoft Toolkit (
- 4 C++ QT库开发
- 5 win7激活工具 win7旗舰版激活
- 6 Android仿美团切换城市
- 7 基于Android的计步器(Pedo
- 8 Oracle使用——PLSQL的中文
- 9 win7永久激活码免费分享
- 10 关于SpringMVC的文件上传



```

        wordLen = L[0].size();

        vector<int> res;
        for(int i = 0; i <= (int)(S.size()-L.size()*wordLen); i++)
            if(helper(S, i, wordTimes, L.size()))
                res.push_back(i);

        return res;
    }

    //判断子串s[index...]的前段是否能由L中的单词组合而成
    bool helper(const string &s, int index,
        unordered_map<string, int>wordTimes, int wordNum)
    {
        for(int i = index; wordNum != 0 && i <= (int)s.size()-wordLen; i+=wordLen)
        {
            string word = s.substr(i, wordLen);
            unordered_map<string, int>::iterator ite = wordTimes.find(word);
            if(ite != wordTimes.end() && ite->second > 0)
                {ite->second--; wordNum--;}
            else return false;
        }
        if(wordNum == 0)return true;
        else return false;
    }
};

```

OJ递归的时间小于非递归时间，因为非递归的helper函数中，hash map参数是传值的方式，每次调用都要拷贝一次hash map，递归代码中一直只存在一个hash map对象

算法2

回想前面的题目：LeetCode:Longest Substring Without Repeating Characters 和 LeetCode:Minimum Window Substring，都用了一种滑动窗口的方法。这一题也可以利用相同的思想。

比如s = “a1b2c3a1d4” L={ “a1”，“b2”，“c3”，“d4” }

窗口最开始为空，

a1在L中，加入窗口 **【a1】** b2c3a1d4

本文地址

b2在L中，加入窗口 **【a1b2】** c3a1d4

c3在L中，加入窗口 **【a1b2c3】** a1d4

a1在L中了，但是前面a1已经算了一次，此时只需要把窗口向右移动一个单词a1 **【b2c3a1】** d4

d4在L中，加入窗口a1 **【b2c3a1d4】** 找到了一个匹配

如果把s改为“a1b2c3kka1d4”，那么在第四步中会碰到单词kk，kk不在L中，此时窗口起始位置移动到

kk后面alb2c3kk 【ald4

```
class Solution {
public:
    vector<int> findSubstring(string S, vector<string> &L) {
        unordered_map<string, int> wordTimes; // L中单词出现的次数
        for(int i = 0; i < L.size(); i++)
            if(wordTimes.count(L[i]) == 0)
                wordTimes.insert(make_pair(L[i], 1));
            else wordTimes[L[i]]++;
        int wordLen = L[0].size();

        vector<int> res;
        for(int i = 0; i < wordLen; i++)
            { // 为了不遗漏从s的每一个位置开始的子串，第一层循环为单词的长度
                unordered_map<string, int> wordTimes2; // 当前窗口中单词出现的次数
                int winStart = i, cnt = 0; // winStart为窗口起始位置, cnt为当前窗口中的单词
数目
                for(int winEnd = i; winEnd <= (int)S.size() - wordLen; winEnd += wordLen)
                    { // 窗口为[winStart, winEnd)
                        string word = S.substr(winEnd, wordLen);
                        if(wordTimes.find(word) != wordTimes.end())
                        {
                            if(wordTimes2.find(word) == wordTimes2.end())
                                wordTimes2[word] = 1;
                            else wordTimes2[word]++;

                            if(wordTimes2[word] <= wordTimes[word])
                                cnt++;
                            else
                                { // 当前的单词在L中，但是它已经在窗口中出现了相应的次数，不应
该加入窗口
                                // 此时，应该把窗口起始位置想左移动到，该单词第一次出现的位
置的下一个单词位置

                                for(int k = winStart; ; k += wordLen)
                                    {
                                        string tmpstr = S.substr(k, wordLen);
                                        wordTimes2[tmpstr]--;
                                        if(tmpstr == word)
                                            {
                                                winStart = k + wordLen;
                                                break;
                                            }
                                    }
                                cnt--;
                            }
                        }

                        if(cnt == L.size())
                            res.push_back(winStart);
                    }
            }
    }
};
```

```
else
{
    //发现不在L中的单词
    winStart = winEnd + wordLen;
    wordTimes2.clear();
    cnt = 0;
}

}

}

return res;
}

};
```

算法时间复杂度为O (n*k) n是字符串的长度，k是单词的长度



酒店式公寓月租



app开发报价单



硬盘数据恢复



灵璧石价格



C++培训

点击复制链接 与好友分享!

回本站首页

上一篇: [十进制转二进制-快速算法](#)
下一篇: [探讨C++中的Map映射机制](#)

相关文章

- [Leetcode: Longest Substring With](#)
- [Leetcode: Decode Ways](#)
- [Leetcode: ZigZag Conversion](#)
- [Leetcode: 3Sum](#)
- [Leetcode: Implement strStr\(\)](#)
- [Leetcode: Candy](#)
- [Leetcode: Clone Graph](#)
- [Leetcode: Median of Two Sorted](#)
- [Leetcode: Substring with Concat](#)
- [Leetcode: Reverse Nodes in k-Group](#)

你的生肖决定了你这辈子的命运

属鼠人的命运

属牛人的命运

属虎人的命运

属兔人的命运

属龙人的命运

属蛇人的命运

属马人的命运

属羊人的命运

属猴人的命运

属鸡人的命运

属狗人的命运

属猪人的命运

图文推荐



达内的



程序员



app开发报价单



酒店式公寓月租



C++设计模式系列之三



【C++】泛型编程基础




全面回顾认识C++(十)



C++工厂模式详解——

我有话说(0条评论)



来说两句吧...

搜狐登录

微博登录

QQ登录

手机登录

还没有评论，快来抢沙发吧！