面试题 17.17 多次搜索

题目地址(面试题 17.17 多次搜索)

https://leetcode-cn.com/problems/multi-search-lcci

标签

- 前缀树

难度

- 中等

题目描述

给定一个较长字符串 big 和一个包含较短字符串的数组 smalls，设计一个方法，根据 smalls 中的每一个较短字符串，对 big 进行搜索。输出 smalls 中

示例：

输入：

```
big = "mississippi"
smalls = ["is","ppi","hi","sis","i","ssippi"]
```

输出： [[1,4],[8],[],[3],[1,4,7,10],[5]]

提示：

```
0 <= len(big) <= 1000
0 <= len(smalls[i]) <= 1000
smalls 的总字符数不会超过 100000。
你可以认为 smalls 中没有重复字符串。
所有出现的字符均为英文小写字母。
```

前置知识

- 字符串匹配
- Trie

思路

最清晰直观的方式就是直接暴力：挨个子串检索 → 暴力解法（不建议），不做过多说明。

该题这个情景我们挺常见的，我们打游戏的时候有时候生气骂人发出去的却是被和谐掉了，这就是因为我们发送的文本中包含敏感词，于是把敏感词替换成了***，而 Trie 的其中一种作用就是检测敏感词，接下来我们做个分析：

- 拿什么建树?
  - 长句：把长句所有的子串遍历一遍添加到 Trie 中，并且做个下标的记录，这样我们在遍历每个敏感词，查看这个敏感词是否存在于 Trie 中，还记得我们讲义说的 Trie 建树的空间复杂度吧，也就是树越深，复杂度很可能越高，一个长句最长的子串就是它本身，因此该种方法可能会 A 了这个题，但是并不建议使用。
  - 敏感词：把所有的敏感词都添加到 Trie 中，由于敏感词基本上长度都比较短，毕竟是个词，建成的树所消耗的空间理论上远小于用长句建树的空间的。为了方便后面找到对应敏感词所对应的下标，我们可以在 Node 中新增一个 ID 属性。

→ 用敏感词建树

- 如何 check 呢?
  - 建立好一颗由敏感词构成的 Trie。
  - 遍历长句中所有的子串，遇到符合的，直接把起始下表添加到对应敏感词的结果集中去，要注意，我们在遍历一个以某一字符为起始字符的所有子串时，在顺序遍历过程中遇到了某个子串不存在于 Trie，那么就没必要继续遍历了，因为 Trie 中并没有以该子串为 prefix 的敏感词。

→Trie 解决方案

代码

第一种方法是 Trie 的解决方法，该题说白了也是字符串匹配问题，字符串匹配也很容易想到 KMP，因此第二种方法是 KMP 方法，我仅贴出来供大家查阅，在后续 KMP 专题结束后大家可以回过头来将该题用 KMP 方法解决一遍，最后一种方法是暴力做法。

- Trie

```
class Solution {

    private Node root = new Node();

    public int[][] multiSearch(String big, String[] smalls) {

        int n = smalls.length;
        // 初始化结果集
        List<Integer>[] res = new List[n];
```

```
        for(int i = 0 ; i < n ; i++)
            res[i] = new ArrayList<>();
        // 建树
        for(int i = 0 ; i < smalls.length; i++)
            insert(smalls[i], i);

        for(int i = 0 ; i < big.length(); i++){

            Node tmp = root;

            for(int j = i ; j < big.length(); j++){
                //不存在以该串为prefix的敏感词
                if(tmp.children[big.charAt(j) - 'a'] == null)
                    break;

                tmp = tmp.children[big.charAt(j) - 'a'];

                if(tmp.isWord)
                    res[tmp.id].add(i);
            }
        }
        // 返回二维数组
        int[][] ret = new int[n][];

        for(int i = 0 ; i < n ; i++){

            ret[i] = new int[res[i].size()];

            for(int j = 0 ; j < ret[i].length; j++)
                ret[i][j] = res[i].get(j);
        }

        return ret;
    }

    private void insert(String word, int id){

        Node tmp = root;

        for(int i = 0; i < word.length(); i++){

            if(tmp.children[word.charAt(i) - 'a'] == null)
                tmp.children[word.charAt(i) - 'a'] = new Node();

            tmp = tmp.children[word.charAt(i) - 'a'];
        }

        tmp.isWord = true;
        tmp.id = id;
    }

    class Node {

        Node[] children;
```

```
        boolean isWord;
        int id;

        public Node() {

            children = new Node[26];
            isWord = false;
            id = 0;
        }
    }
}
```

- KMP

```
class Solution {

    public int[][] multiSearch(String big, String[] smalls) {

        int[][] res = new int[smalls.length][];

        List<Integer> cur = new ArrayList<>();

        for (int i = 0; i < smalls.length; i++) {

            String small = smalls[i];

            if (small.length() == 0) {

                res[i] = new int[]{};
                continue;
            }
            // kmp
            int[] next = getNext(small);

            int x = 0, y = 0;

            while (x < big.length() && y < small.length()) {

                if (big.charAt(x) == small.charAt(y)) {

                    x++;
                    y++;
                } else {

                    if (y > 0)
                        y = next[y - 1];
                    else
                        x++;
                }

                if (y == small.length()) {
```

```
                    y = next[y - 1];
                    cur.add(x - small.length());
                }
            }

            res[i] = new int[cur.size()];
            for (int j = 0; j < res[i].length; j++)
                res[i][j] = cur.get(j);

            cur.clear();
        }

        return res;
    }

    public int[] getNext(String pattern) {

        int j = 0;
        int[] next = new int[pattern.length()];

        for (int i = 1; i < pattern.length(); i++) {

            if (pattern.charAt(i) == pattern.charAt(j)) {

                next[i] = j + 1;
                j++;
            } else {

                while (j > 0 && pattern.charAt(j) != pattern.charAt(i))
                    j = next[j - 1];

                if (pattern.charAt(j) == pattern.charAt(i)) {

                    next[i] = j + 1;
                    j++;
                }
            }
        }

        return next;
    }
}
```

- Java 暴力（用库就完事了）

```
public int[][] multiSearch(String big, String[] smalls) {

    int[][] res = new int[smalls.length][];

    List<Integer> cur = new ArrayList<>();

    for (int i = 0; i < smalls.length; i++) {
```

```
        String small = smalls[i];

        if (small.length() == 0) {

            res[i] = new int[]{};
            continue;
        }

        int startIdx = 0;
        while (true) {

            int idx = big.indexOf(small, startIdx);
            if (idx == -1)
                break;

            cur.add(idx);
            startIdx = idx + 1;
        }

        res[i] = new int[cur.size()];
        for (int j = 0; j < res[i].length; j++)
            res[i][j] = cur.get(j);

        cur.clear();
    }

    return res;
}
```

复杂度分析

- 时间复杂度：$O(N * K)$，其中 K 是敏感词中最长单词长度，N 是长句的长度。

- 空间复杂度：$O(S)$, S 为所有匹配成功的位置的个数

本次 Trie 专题结束了，相信大家对 Trie 有了充分认识，希望多加练习，以后活用好这种方便的数据结构，谢谢大家！

上一页 下一页