

【哈希表】-猜字谜

视频直播讲解: [📺 2024/09/07 真题讲解 \(2024E卷\)](#)

题目描述与示例

题目描述

小王设计了一个简单的猜字谜游戏，游戏的谜面是一个错误的单词，比如 `nesw`，玩家需要猜出谜底库中正确的单词。猜中的要求如下：

对于某个谜面和谜底单词，满足下面任一条件都表示猜中：

1. 变换顺序以后一样的，比如通过变换 `w` 和 `e` 的顺序，`"nwes"` 跟 `"news"` 是可以完全对应的；
2. 字母去重以后是一样的，比如 `"woood"` 和 `"wood"` 是一样的，它们去重后都是 `"wod"`

请你写一个程序帮忙在谜底库中找到正确的谜底。谜面是多个单词，都需要找到对应的谜底，如果找不到的话，返回 `"not found"`

输入描述

- 谜面单词列表，以 `","` 分隔
- 谜底库单词列表，以 `","` 分隔

输出描述

- 匹配到的正确单词列表，以 `","` 分隔
- 如果找不到，返回 `"not found"`

备注

- 单词的数量 `N` 的范围: `0 < N < 1000`
- 词汇表的数量 `M` 的范围: `0 < M < 1000`
- 单词的长度 `P` 的范围: `0 < P < 20`
- 输入的字符只有小写英文字母，没有其他字符

示例一

输入

```
1 conection
2 connection,today
```

输出

```
1 connection
```

示例二

输入

```
1 bdni,woood
2 bind,wrong,wood
```

输出

```
1 bind,wood
```

解题思路

谜面和谜底如何匹配

首先考虑谜面单词变量 `riddle` 和谜底单词变量 `answer` 如何进行匹配。

谜面和谜底的匹配必须遵循两个条件

1. 两个单词均自身去重
2. 两个单词去重后的所有字母出现次数均相等

第二点也可以换另一种表述，即两个单词**去重后再进行排序后的结果完全相等**。

因此我们可以构造出这样一个函数 `check()`，来检查某个 `riddle` 和某个 `answer` 是否一致

```
1 def check(riddle, answer):
2     r = "".join(sorted(set(riddle)))
3     a = "".join(sorted(set(answer)))
4     return r == a
```

显然，`set()` 在此处起到去重的作用。我们对去重后的结果进行排序得到列表，再合并为字符串得到 `r` 和 `a`。

通过比较 `r` 和 `a` 是否完全一致，来判断 `answer` 是否为 `riddle` 的谜底。

这个函数的时间复杂度为 $O(R\log R + A\log A)$ ，其中 `R` 和 `A` 分别为 `riddle` 和 `answer` 的长度。

暴力匹配所有谜底

输入的第一行是若干的谜面，输入的第二行是若干谜底。

显然，对于任意一个谜面 `riddle`，我们希望它能够跟**每一个谜底** `answer` 进行比较，以找到匹配的那个谜底。

这种直观的穷举的写法如下

```
1 riddles = input().split(",")
2 answers = input().split(",")
3
4 res = list()
5
6 # 遍历每一个谜面单词riddle
```

```

7 for riddle in riddles:
8     # 先假设riddle找不到谜底，设置一个flag为False
9     flag = False
10    # 遍历所有谜底answer
11    for answer in answers:
12        # 如果匹配，则将answer加入res
13        # 且标记flag为True，表示找到了谜底
14        # 同时退出循环
15        if check(riddle, answer):
16            res.append(answer)
17            flag = True
18            break
19    # 如果上述内层循环结束后，仍然没有找到谜底，则
20    # 往res中加入"not found"
21    if not flag:
22        res.append("not found")
23
24 print(",".join(res))

```

这种写法使用了双重 `for` 循环的写法，如果谜面和谜底数组的长度分别为 `n` 和 `m`，在每一次循环中我们都要调用 `check()` 函数，那么总时间复杂度就为 $O(nm(R\log R + A\log A))$ 。

谜底库哈希表的构建

考虑到 `n` 和 `m` 的最大值均为 `1000`，而 `R` 和 `A` 的最大值为 `20`。

$O(nm(R\log R + A\log A))$ 的最大值可以到大约 $O(10^8)$ ，所以上述做法很可能会超时。

需要考虑时间复杂度更加优秀的解法。

很容易发现，即使对于不同的谜面 `riddle`，实际上匹配的都是同一个谜底库 `answers`。

假设有两个不同的谜面 `riddle1` 和 `riddle2`，对同一个谜底 `answer` 进行匹配操作 `check`，那么 `answer` 的去重和排序操作需要重复进行两遍。

显然这里的重复操作是冗余的。

容易想到，我们可以对谜底库 `answers` 进行预处理，构建一个用哈希表表示的谜底库 `answers_dic`。

```
1 answers_dic = dict()
2 for answer in answers:
3     a = "".join(sorted(set(answer)))
4     answers_dic[a] = answer
```

其中，谜底库 `answers_dic` 的 `key` 是每一个 `answer` 去重再排序后的结果 `a`，而 `value` 是谜底 `answer` 本身。

构建这样谜底库的好处是，对于任意一个谜面 `riddle`，我们都可以通过其去重后排序的结果 `r`，来迅速定位到谜底库中是否存在对应的谜底。

换句话说，对于特定的谜面 `riddle`，我们可以用排序后的结果 `r` 和 `a` 来作为桥梁，迅速找到谜底 `answer`。

因此，对应的代码如下

```
1 for riddle in riddles:
2     r = "".join(sorted(set(riddle)))
3     if r in answers_dic:
4         res.append(answers_dic[r])
5     else:
6         res.append("not found")
```

甚至无需构建 `check()` 函数，在代码层面上也更加简洁。

该种解法的总时间复杂度为 $O(nR\log R + mA\log A)$ 。

其中 $O(mA\log A)$ 是构建谜底库 `answers_dic` 的开销， $O(nR\log R)$ 是上述遍历过程的开销。

代码

解法一：哈希表预处理谜底

Python

```
1 # 题目：【哈希表】2024E-猜字谜
2 # 分值：100
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：哈希表
5 # 代码看不懂的地方，请直接在群上提问
6
7 riddles = input().split(",")
8 answers = input().split(",")
9
10 res = list()
11
12 # 构建谜底库answers_dic, 其中
13 # key为每一个谜底answer去重后排序的结果a
14 # value为每一个谜底answer自身
15 answers_dic = dict()
16 for answer in answers:
17     a = "".join(sorted(set(answer)))
18     answers_dic[a] = answer
19
20
21 # 遍历每一个谜面riddle
22 for riddle in riddles:
23     # 得到riddle去重后排序的结果r
24     r = "".join(sorted(set(riddle)))
25     # 查看r是否位于谜底库中
26     # 若在则将对应的谜底answers_dic[r]加入res中
27     # 否则储存字符串"not found"
28     if r in answers_dic:
29         res.append(answers_dic[r])
30     else:
31         res.append("not found")
32
33
34 print(",".join(res))
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         // 读取谜面和谜底
```

```

6      Scanner scanner = new Scanner(System.in);
7      String[] riddles = scanner.nextLine().split(",");
8      String[] answers = scanner.nextLine().split(",");
9
10     List<String> res = new ArrayList<>();
11     Map<String, String> answersMap = new HashMap<>();
12
13     // 构建谜底库answersMap, 其中
14     // key为每一个谜底answer去重后排序的结果a
15     // value为每一个谜底answer自身
16     for (String answer : answers) {
17         String a = sortAndDeduplicate(answer);
18         answersMap.put(a, answer);
19     }
20
21     // 遍历每一个谜面riddle
22     for (String riddle : riddles) {
23         // 得到riddle去重后排序的结果r
24         String r = sortAndDeduplicate(riddle);
25         // 查看r是否位于谜底库中
26         if (answersMap.containsKey(r)) {
27             res.add(answersMap.get(r));
28         } else {
29             res.add("not found");
30         }
31     }
32
33     // 输出结果
34     System.out.println(String.join(",", res));
35     scanner.close();
36 }
37
38 // Helper方法: 将字符串去重并排序
39 public static String sortAndDeduplicate(String s) {
40     Set<Character> charSet = new TreeSet<>(); // 使用TreeSet自动去重并排序
41     for (char c : s.toCharArray()) {
42         charSet.add(c);
43     }
44     StringBuilder sb = new StringBuilder();
45     for (char c : charSet) {
46         sb.append(c);
47     }
48     return sb.toString();
49 }
50 }
51

```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 #include <set>
5 #include <sstream>
6 #include <algorithm>
7
8 using namespace std;
9
10 // Helper函数：将字符串去重并排序
11 string sortAndDeduplicate(const string& s) {
12     set<char> charSet(s.begin(), s.end()); // 自动去重并排序
13     string result;
14     for (char c : charSet) {
15         result += c;
16     }
17     return result;
18 }
19
20 int main() {
21     // 读取谜面和谜底
22     string riddlesStr, answersStr;
23     getline(cin, riddlesStr);
24     getline(cin, answersStr);
25
26     // 使用stringstream进行字符串切分
27     stringstream riddlesStream(riddlesStr), answersStream(answersStr);
28     vector<string> riddles, answers, res;
29     string riddle, answer;
30
31     // 处理谜面和谜底
32     while (getline(riddlesStream, riddle, ',')) {
33         riddles.push_back(riddle);
34     }
35     while (getline(answersStream, answer, ',')) {
36         answers.push_back(answer);
37     }
38
39     // 构建谜底库answersMap, 其中
40     // key为每一个谜底answer去重后排序的结果a
41     // value为每一个谜底answer自身
42     unordered_map<string, string> answersMap;
43     for (const string& ans : answers) {
44         string a = sortAndDeduplicate(ans);
```



```

45     answersMap[a] = ans;
46 }
47
48 // 遍历每一个谜面riddle
49 for (const string& rid : riddles) {
50     string r = sortAndDeduplicate(rid);
51     // 查看r是否位于谜底库中
52     if (answersMap.find(r) != answersMap.end()) {
53         res.push_back(answersMap[r]);
54     } else {
55         res.push_back("not found");
56     }
57 }
58
59 // 输出结果
60 for (size_t i = 0; i < res.size(); ++i) {
61     if (i > 0) cout << ",";
62     cout << res[i];
63 }
64 cout << endl;
65
66 return 0;
67 }
68

```

时空复杂度

时间复杂度： $O(nR\log R + mA\log A)$ 。

空间复杂度： $O(1)$ 。

解法二：暴力匹配解（会超时）

Python

```

1 # 题目：【哈希表】2024E-猜字谜
2 # 分值：100
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：暴力解
5 # 代码看不懂的地方，请直接在群上提问

```

```

6
7
8 # 检查谜面riddle和谜底answer是否匹配的函数
9 def check(riddle, answer):
10     r = "".join(sorted(set(riddle)))
11     a = "".join(sorted(set(answer)))
12     return r == a
13
14
15 riddles = input().split(",")
16 answers = input().split(",")
17
18 res = list()
19
20 # 遍历每一个谜面单词riddle
21 for riddle in riddles:
22     # 先假设riddle找不到谜底, 设置一个flag为False
23     flag = False
24     # 遍历所有谜底answer
25     for answer in answers:
26         # 如果匹配, 则将answer加入res
27         # 且标记flag为True, 表示找到了谜底
28         # 同时退出循环
29         if check(riddle, answer):
30             res.append(answer)
31             flag = True
32             break
33     # 如果上述内层循环结束后, 仍然没有找到谜底, 则
34     # 往res中加入"not found"
35     if not flag:
36         res.append("not found")
37
38 print(",".join(res))

```

Java

```

1 import java.util.*;
2
3 public class Main {
4     // 检查谜面riddle和谜底answer是否匹配的函数
5     public static boolean check(String riddle, String answer) {
6         // 去重后排序, 然后比较
7         String r = sortAndDeduplicate(riddle);
8         String a = sortAndDeduplicate(answer);
9         return r.equals(a);

```

```

10     }
11
12     // Helper函数：将字符串去重并排序
13     public static String sortAndDeduplicate(String s) {
14         Set<Character> charSet = new TreeSet<>(); // 使用TreeSet自动去重并排序
15         for (char c : s.toCharArray()) {
16             charSet.add(c);
17         }
18         StringBuilder sb = new StringBuilder();
19         for (char c : charSet) {
20             sb.append(c);
21         }
22         return sb.toString();
23     }
24
25     public static void main(String[] args) {
26         // 读取谜面和谜底
27         Scanner scanner = new Scanner(System.in);
28         String[] riddles = scanner.nextLine().split(",");
29         String[] answers = scanner.nextLine().split(",");
30
31         List<String> res = new ArrayList<>();
32
33         // 遍历每一个谜面单词riddle
34         for (String riddle : riddles) {
35             boolean flag = false; // 先假设找不到谜底
36             // 遍历所有谜底answer
37             for (String answer : answers) {
38                 // 如果匹配，则将answer加入res，且退出循环
39                 if (check(riddle, answer)) {
40                     res.add(answer);
41                     flag = true;
42                     break;
43                 }
44             }
45             // 如果没有找到匹配的谜底，加入"not found"
46             if (!flag) {
47                 res.add("not found");
48             }
49         }
50
51         // 输出结果
52         System.out.println(String.join(",", res));
53         scanner.close();
54     }
55 }
56

```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <set>
4 #include <sstream>
5 #include <algorithm>
6
7 using namespace std;
8
9 // Helper函数：将字符串去重并排序
10 string sortAndDeduplicate(const string& s) {
11     set<char> charSet(s.begin(), s.end()); // 自动去重并排序
12     string result;
13     for (char c : charSet) {
14         result += c;
15     }
16     return result;
17 }
18
19 // 检查谜面riddle和谜底answer是否匹配的函数
20 bool check(const string& riddle, const string& answer) {
21     // 去重后排序，然后比较
22     string r = sortAndDeduplicate(riddle);
23     string a = sortAndDeduplicate(answer);
24     return r == a;
25 }
26
27 int main() {
28     // 读取谜面和谜底
29     string riddlesStr, answersStr;
30     getline(cin, riddlesStr);
31     getline(cin, answersStr);
32
33     // 使用stringstream进行字符串切分
34     stringstream riddlesStream(riddlesStr), answersStream(answersStr);
35     vector<string> riddles, answers, res;
36     string riddle, answer;
37
38     // 处理谜面和谜底
39     while (getline(riddlesStream, riddle, ',')) {
40         riddles.push_back(riddle);
41     }
42     while (getline(answersStream, answer, ',')) {
43         answers.push_back(answer);
```

```

44     }
45
46     // 遍历每一个谜面单词riddle
47     for (const string& rid : riddles) {
48         bool flag = false; // 先假设找不到谜底
49         // 遍历所有谜底answer
50         for (const string& ans : answers) {
51             // 如果匹配, 则将answer加入res, 且退出循环
52             if (check(rid, ans)) {
53                 res.push_back(ans);
54                 flag = true;
55                 break;
56             }
57         }
58         // 如果没有找到匹配的谜底, 加入"not found"
59         if (!flag) {
60             res.push_back("not found");
61         }
62     }
63
64     // 输出结果
65     for (size_t i = 0; i < res.size(); ++i) {
66         if (i > 0) cout << ",";
67         cout << res[i];
68     }
69     cout << endl;
70
71     return 0;
72 }
73
74

```

时空复杂度

时间复杂度: $O(nm(R\log R + A\log A))$ 。

空间复杂度: $O(1)$ 。