【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

```
conection
connection,today
```

输出

```
connection
```

示例二

输入

```
bdni,wooood
bind,wrong,wood
```

输出

```
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

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

显然， `set()` 在此处去到去重的作用。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(RlogR+AlogA)` ，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

```
riddles = input().split(",")
answers = input().split(",")

res = list()

# 遍历每一个谜面单词riddle
```

```python
for riddle in riddles:
    # 先假设riddle找不到谜底，设置一个flag为False
    flag = False
    # 遍历所有谜底answer
    for answer in answers:
        # 如果匹配，则将answer加入res
        # 且标记flag为True，表示找到了谜底
        # 同时退出循环
        if check(riddle, answer):
            res.append(answer)
            flag = True
            break
    # 如果上述内层循环结束后，仍然没有找到谜底，则
    # 往res中加入"not found"
    if not flag:
        res.append("not found")

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(RlogR+AlogA))` 。

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(RlogR+AlogA))` 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` 。

```
answers_dic = dict()
for answer in answers:
    a = "".join(sorted(set(answer)))
    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

```
for riddle in riddles:
    r = "".join(sorted(set(riddle)))
    if r in answers_dic:
        res.append(answers_dic[r])
    else:
        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(nRlogR+mAlogA)` 。

其中 `O(mAlogA)` 是构建谜底库 `answers_dic` 的开销， `O(nRlogR)` 是上述遍历过程的开销。

代码

解法一：哈希表预处理谜底

Python

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

riddles = input().split(",")
answers = input().split(",")

res = list()

# 构建谜底库answers_dic，其中
# key为每一个谜底answer去重后排序的结果a
# value为每一个谜底answer自身
answers_dic = dict()
for answer in answers:
    a = "".join(sorted(set(answer)))
    answers_dic[a] = answer


# 遍历每一个谜面riddle
for riddle in riddles:
    # 得到riddle去重后排序的结果r
    r = "".join(sorted(set(riddle)))
    # 查看r是否位于谜底库中
    # 若在则将对应的谜底answers_dic[r]加入res中
    # 否则储存字符串"not found"
    if r in answers_dic:
        res.append(answers_dic[r])
    else:
        res.append("not found")


print(",".join(res))
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // 读取谜面和谜底
```

```java
        Scanner scanner = new Scanner(System.in);
        String[] riddles = scanner.nextLine().split(",");
        String[] answers = scanner.nextLine().split(",");

        List<String> res = new ArrayList<>();
        Map<String, String> answersMap = new HashMap<>();

        // 构建谜底库answersMap，其中
        // key为每一个谜底answer去重后排序的结果a
        // value为每一个谜底answer自身
        for (String answer : answers) {
            String a = sortAndDeduplicate(answer);
            answersMap.put(a, answer);
        }

        // 遍历每一个谜面riddle
        for (String riddle : riddles) {
            // 得到riddle去重后排序的结果r
            String r = sortAndDeduplicate(riddle);
            // 查看r是否位于谜底库中
            if (answersMap.containsKey(r)) {
                res.add(answersMap.get(r));
            } else {
                res.add("not found");
            }
        }

        // 输出结果
        System.out.println(String.join(",", res));
        scanner.close();
    }

    // Helper方法：将字符串去重并排序
    public static String sortAndDeduplicate(String s) {
        Set<Character> charSet = new TreeSet<>(); // 使用TreeSet自动去重并排序
        for (char c : s.toCharArray()) {
            charSet.add(c);
        }
        StringBuilder sb = new StringBuilder();
        for (char c : charSet) {
            sb.append(c);
        }
        return sb.toString();
    }
}
```

C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <set>
#include <sstream>
#include <algorithm>

using namespace std;

// Helper函数：将字符串去重并排序
string sortAndDeduplicate(const string& s) {
    set<char> charSet(s.begin(), s.end());  // 自动去重并排序
    string result;
    for (char c : charSet) {
        result += c;
    }
    return result;
}

int main() {
    // 读取谜面和谜底
    string riddlesStr, answersStr;
    getline(cin, riddlesStr);
    getline(cin, answersStr);

    // 使用stringstream进行字符串切分
    stringstream riddlesStream(riddlesStr), answersStream(answersStr);
    vector<string> riddles, answers, res;
    string riddle, answer;

    // 处理谜面和谜底
    while (getline(riddlesStream, riddle, ',')) {
        riddles.push_back(riddle);
    }
    while (getline(answersStream, answer, ',')) {
        answers.push_back(answer);
    }

    // 构建谜底库answersMap，其中
    // key为每一个谜底answer去重后排序的结果a
    // value为每一个谜底answer自身
    unordered_map<string, string> answersMap;
    for (const string& ans : answers) {
        string a = sortAndDeduplicate(ans);
```

```
        answersMap[a] = ans;
    }

    // 遍历每一个谜面riddle
    for (const string& rid : riddles) {
        string r = sortAndDeduplicate(rid);
        // 查看r是否位于谜底库中
        if (answersMap.find(r) != answersMap.end()) {
            res.push_back(answersMap[r]);
        } else {
            res.push_back("not found");
        }
    }

    // 输出结果
    for (size_t i = 0; i < res.size(); ++i) {
        if (i > 0) cout << ",";
        cout << res[i];
    }
    cout << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(nRlogR+mAlogA)`。

空间复杂度：`O(1)`。

解法二：暴力匹配解（会超时）

Python

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

```python

# 检查谜面riddle和谜底answer是否匹配的函数
def check(riddle, answer):
    r = "".join(sorted(set(riddle)))
    a = "".join(sorted(set(answer)))
    return r == a


riddles = input().split(",")
answers = input().split(",")

res = list()

# 遍历每一个谜面单词riddle
for riddle in riddles:
    # 先假设riddle找不到谜底，设置一个flag为False
    flag = False
    # 遍历所有谜底answer
    for answer in answers:
        # 如果匹配，则将answer加入res
        # 且标记flag为True，表示找到了谜底
        # 同时退出循环
        if check(riddle, answer):
            res.append(answer)
            flag = True
            break
    # 如果上述内层循环结束后，仍然没有找到谜底，则
    # 往res中加入"not found"
    if not flag:
        res.append("not found")

print(",".join(res))
```

Java

```java
import java.util.*;

public class Main {
    // 检查谜面riddle和谜底answer是否匹配的函数
    public static boolean check(String riddle, String answer) {
        // 去重后排序，然后比较
        String r = sortAndDeduplicate(riddle);
        String a = sortAndDeduplicate(answer);
        return r.equals(a);
```

```java
    }

    // Helper函数：将字符串去重并排序
    public static String sortAndDeduplicate(String s) {
        Set<Character> charSet = new TreeSet<>(); // 使用TreeSet自动去重并排序
        for (char c : s.toCharArray()) {
            charSet.add(c);
        }
        StringBuilder sb = new StringBuilder();
        for (char c : charSet) {
            sb.append(c);
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        // 读取谜面和谜底
        Scanner scanner = new Scanner(System.in);
        String[] riddles = scanner.nextLine().split(",");
        String[] answers = scanner.nextLine().split(",");

        List<String> res = new ArrayList<>();

        // 遍历每一个谜面单词riddle
        for (String riddle : riddles) {
            boolean flag = false; // 先假设找不到谜底
            // 遍历所有谜底answer
            for (String answer : answers) {
                // 如果匹配，则将answer加入res，且退出循环
                if (check(riddle, answer)) {
                    res.add(answer);
                    flag = true;
                    break;
                }
            }
            // 如果没有找到匹配的谜底，加入"not found"
            if (!flag) {
                res.add("not found");
            }
        }

        // 输出结果
        System.out.println(String.join(",", res));
        scanner.close();
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <sstream>
#include <algorithm>

using namespace std;

// Helper函数：将字符串去重并排序
string sortAndDeduplicate(const string& s) {
    set<char> charSet(s.begin(), s.end());  // 自动去重并排序
    string result;
    for (char c : charSet) {
        result += c;
    }
    return result;
}

// 检查谜面riddle和谜底answer是否匹配的函数
bool check(const string& riddle, const string& answer) {
    // 去重后排序，然后比较
    string r = sortAndDeduplicate(riddle);
    string a = sortAndDeduplicate(answer);
    return r == a;
}

int main() {
    // 读取谜面和谜底
    string riddlesStr, answersStr;
    getline(cin, riddlesStr);
    getline(cin, answersStr);

    // 使用stringstream进行字符串切分
    stringstream riddlesStream(riddlesStr), answersStream(answersStr);
    vector<string> riddles, answers, res;
    string riddle, answer;

    // 处理谜面和谜底
    while (getline(riddlesStream, riddle, ',')) {
        riddles.push_back(riddle);
    }
    while (getline(answersStream, answer, ',')) {
        answers.push_back(answer);
```

```
    }

    // 遍历每一个谜面单词riddle
    for (const string& rid : riddles) {
        bool flag = false; // 先假设找不到谜底
        // 遍历所有谜底answer
        for (const string& ans : answers) {
            // 如果匹配，则将answer加入res，且退出循环
            if (check(rid, ans)) {
                res.push_back(ans);
                flag = true;
                break;
            }
        }
        // 如果没有找到匹配的谜底，加入"not found"
        if (!flag) {
            res.push_back("not found");
        }
    }

    // 输出结果
    for (size_t i = 0; i < res.size(); ++i) {
        if (i > 0) cout << ",";
        cout << res[i];
    }
    cout << endl;

    return 0;
}
```

时空复杂度

时间复杂度：`O(nm(RlogR+AlogA))`。

空间复杂度：`O(1)`。