

Find Duplicate File in System

Given a list of directory info including directory path, and all the files with contents in this directory, you need to find out all the groups of duplicate files in the file system in terms of their paths.

A group of duplicate files consists of at least **two** files that have exactly the same content.

A single directory info string in the **input** list has the following format:

```
"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ...  
fn.txt(fn_content)"
```

It means there are **n** files (`f1.txt` , `f2.txt` ... `fn.txt` with content `f1_content` , `f2_content` ... `fn_content` , respectively) in directory `root/d1/d2/.../dm` . Note that $n \geq 1$ and $m \geq 0$. If $m = 0$, it means the directory is just the root directory.

The **output** is a list of group of duplicate file paths. For each group, it contains all the file paths of the files that have the same content. A file path is a string that has the following format:

```
"directory_path/file_name.txt"
```

Example 1:

Input:

```
["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)", "root/c/d 4.txt(efgh)", "root 4.txt(efgh)"]
```

Output:

```
[["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]
```

Note:

1. No order is required for the final output.
2. You may assume the directory name, file name and file content only has letters and digits, and the length of file content is in the range of $[1,50]$.
3. The number of files given is in the range of $[1,20000]$.
4. You may assume no files or directories share the same name in a same directory.
5. You may assume each given directory info represents a unique directory. Directory path and file infos are separated by a single blank space.

Follow up beyond contest:

1. Imagine you are given a real file system, how will you search files? DFS or BFS ?
2. If the file content is very large (GB level), how will you modify your solution?
3. If you can only read the file by 1kb each time, how will you modify your solution?
4. What is the time complexity of your modified solution? What is the most time consuming part and memory consuming part of it? How to optimize?
5. How to make sure the duplicated files you find are not false positive?

Solution 1

Idea is simple:

Use a hashmap with names vectors to store all files contents, and then prints the duplicates

...

```
vector<vector<string>> findDuplicate(vector<string>& paths) {
    unordered_map<string, vector<string>> files;
    vector<vector<string>> result;

    for (auto path : paths) {
        stringstream ss(path);
        string root;
        string s;
        getline(ss, root, ' ');
        while (getline(ss, s, ' ')) {
            string fileName = root + '/' + s.substr(0, s.find('('));
            string fileContent = s.substr(s.find('(') + 1, s.find(')') - s.find('(') - 1);
            ;
            files[fileContent].push_back(fileName);
        }
    }

    for (auto file : files) {
        if (file.second.size() > 1)
            result.push_back(file.second);
    }

    return result;
}
```

...

Follow up questions:

1. Imagine you are given a real file system, how will you search files? DFS or BFS ?

In general, BFS will use more memory then DFS. However BFS can take advantage of the locality of files in inside directories, and therefore will probably be faster

2. If the file content is very large (GB level), how will you modify your solution?

In a real life solution we will not hash the entire file content, since it's not practical. Instead we will first map all the files according to size. Files with different sizes are guaranteed to be different. We will than hash a small part of the files with equal sizes (using MD5 for example). Only if the md5 is the same, we will compare the files byte by byte

3. If you can only read the file by 1kb each time, how will you modify your solution?

This won't change the solution. We can create the hash from the 1kb chunks, and then read the entire file if a full byte by byte comparison is required.

What is the time complexity of your modified solution? What is the most

time consuming part and memory consuming part of it? How to optimize?

Time complexity is $O(n^2 * k)$ since in worse case we might need to compare every file to all others. k is the file size

How to make sure the duplicated files you find are not false positive?

We will use several filters to compare: File size, Hash and byte by byte comparisons.

written by [elisemory](#) original link [here](#)

Solution 2

If the creation of the map can also be done using Java8 that would have been cool.

```
public static List<List<String>> findDuplicate(String[] paths) {  
    Map<String, List<String>> map = new HashMap<>();  
    for(String path : paths) {  
        String[] tokens = path.split(" ");  
        for(int i = 1; i < tokens.length; i++) {  
            String file = tokens[i].substring(0, tokens[i].indexOf('('));  
            String content = tokens[i].substring(tokens[i].indexOf('(') + 1, tokens[i].indexOf(' '));  
            map.putIfAbsent(content, new ArrayList<>());  
            map.get(content).add(tokens[0] + "/" + file);  
        }  
    }  
    return map.values().stream().filter(e -> e.size() > 1).collect(Collectors.toList());  
}
```

written by [johnyrufus16](#) original link [here](#)

Solution 3

```
public class Solution {
    public List<List<String>> findDuplicate(String[] paths) {
        List<List<String>> result = new ArrayList<List<String>>();
        int n = paths.length;
        if (n == 0) return result;

        Map<String, Set<String>> map = new HashMap<>();
        for (String path : paths) {
            String[] strs = path.split("\\s+");
            for (int i = 1; i < strs.length; i++) {
                int idx = strs[i].indexOf("(");
                String content = strs[i].substring(idx);
                String filename = strs[0] + "/" + strs[i].substring(0, idx);
                Set<String> filenames = map.getOrDefault(content, new HashSet<String>());
                filenames.add(filename);
                map.put(content, filenames);
            }
        }

        for (String key : map.keySet()) {
            if (map.get(key).size() > 1) {
                result.add(new ArrayList<String>(map.get(key)));
            }
        }

        return result;
    }
}
```

written by [shawngao](#) original link [here](#)

From [LeetCoder](#).