Problem 1948: Delete Duplicate Folders in System

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Due to a bug, there are many duplicate folders in a file system. You are given a 2D array

paths

, where

paths[i]

is an array representing an absolute path to the

i

th

folder in the file system.

For example,

["one", "two", "three"]

represents the path

"/one/two/three"

.

/a

Two folders (not necessarily on the same level) are

identical

if they contain the

same non-empty

set of identical subfolders and underlying subfolder structure. The folders

do not

need to be at the root level to be identical. If two or more folders are

identical

, then

mark

the folders as well as all their subfolders.

For example, folders

"/a"

and

"/b"

in the file structure below are identical. They (as well as their subfolders) should

all

be marked:

/a

/a/x

/a/x/y

/a/z

/b

/b/x

/b/x/y

/b/z

However, if the file structure also included the path

"/b/w"

, then the folders

"/a"

and

"/b"

would not be identical. Note that

"/a/x"

and

"/b/x"

would still be considered identical even with the added folder.

Once all the identical folders and their subfolders have been marked, the file system will

delete

all of them. The file system only runs the deletion once, so any folders that become identical after the initial deletion are not deleted.

Return

the 2D array

ans

containing the paths of the

remaining

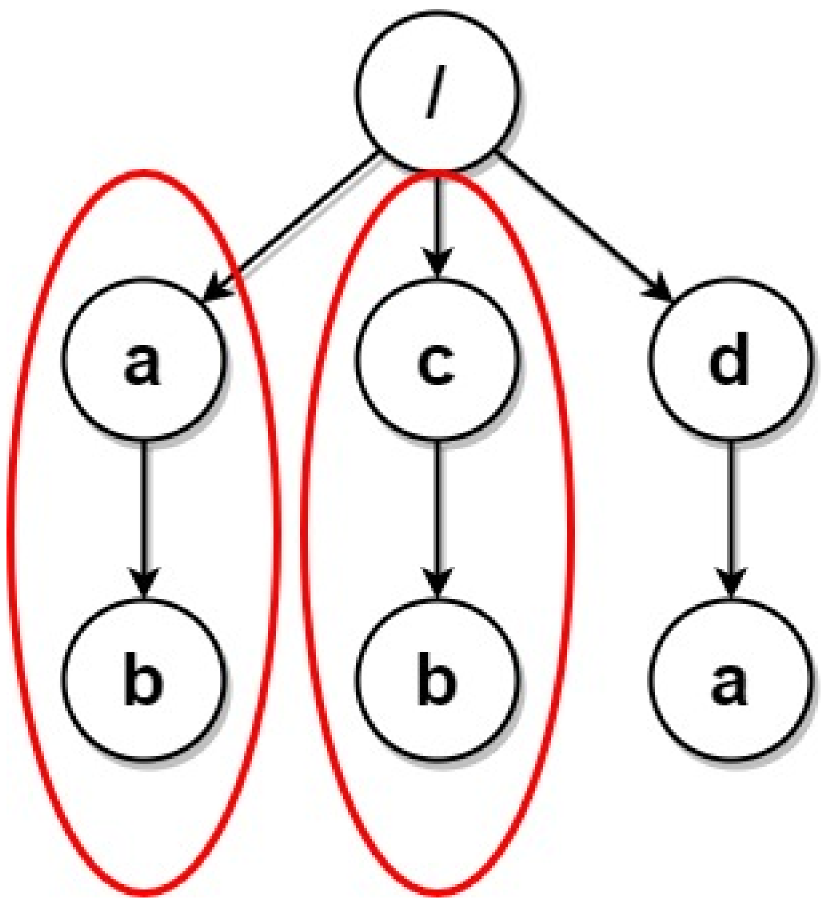
folders after deleting all the marked folders. The paths may be returned in

any

order

.

Example 1:



Input:

paths = [["a"],["c"],["d"],["a","b"],["c","b"],["d","a"]]

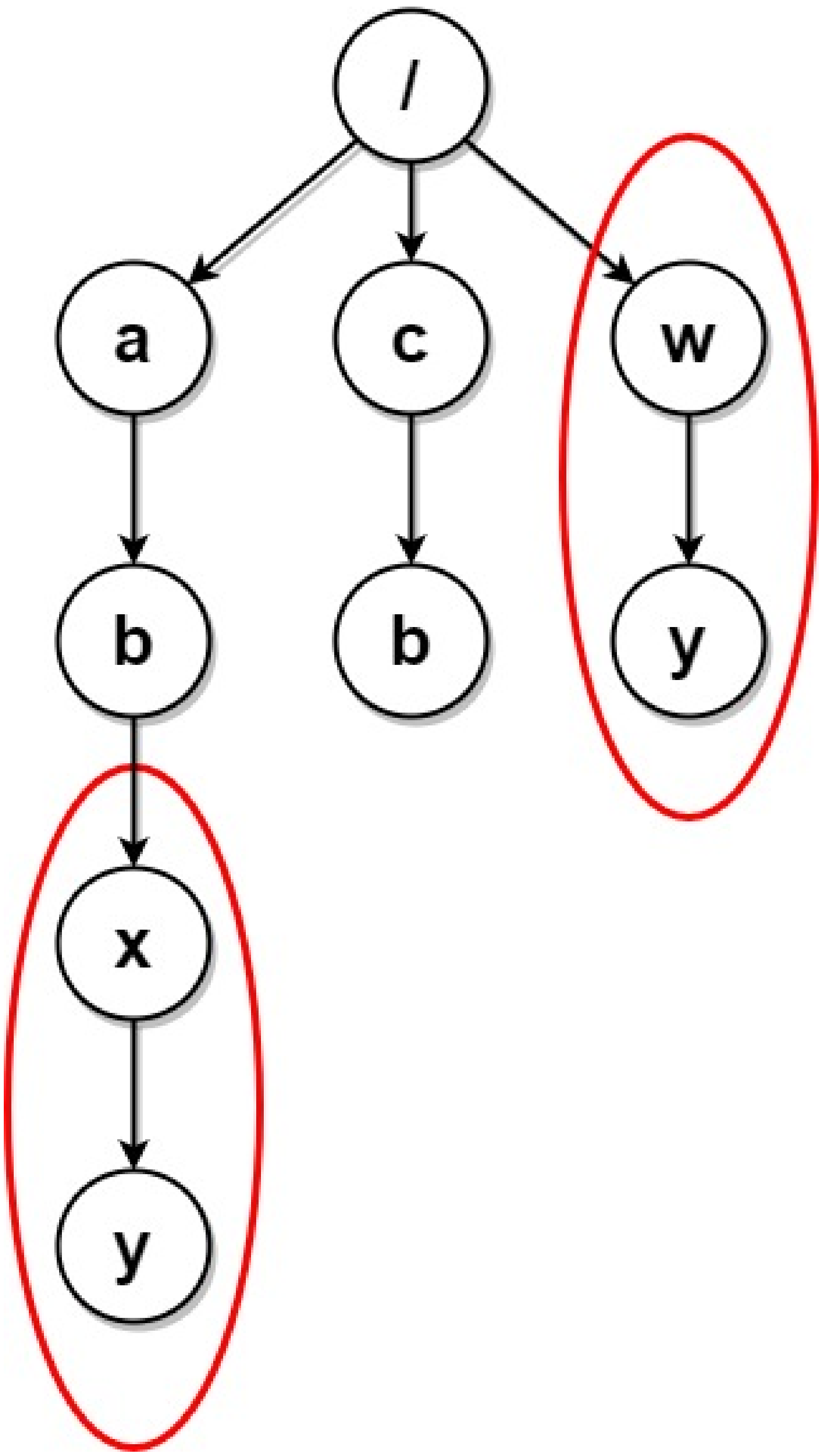
Output:

[["d"],["d","a"]]

Explanation:

The file structure is as shown. Folders "/a" and "/c" (and their subfolders) are marked for deletion because they both contain an empty folder named "b".

Example 2:



Input:

paths = [["a"],["c"],["a","b"],["c","b"],["a","b","x"],["a","b","x","y"],["w"],["w","y"]]

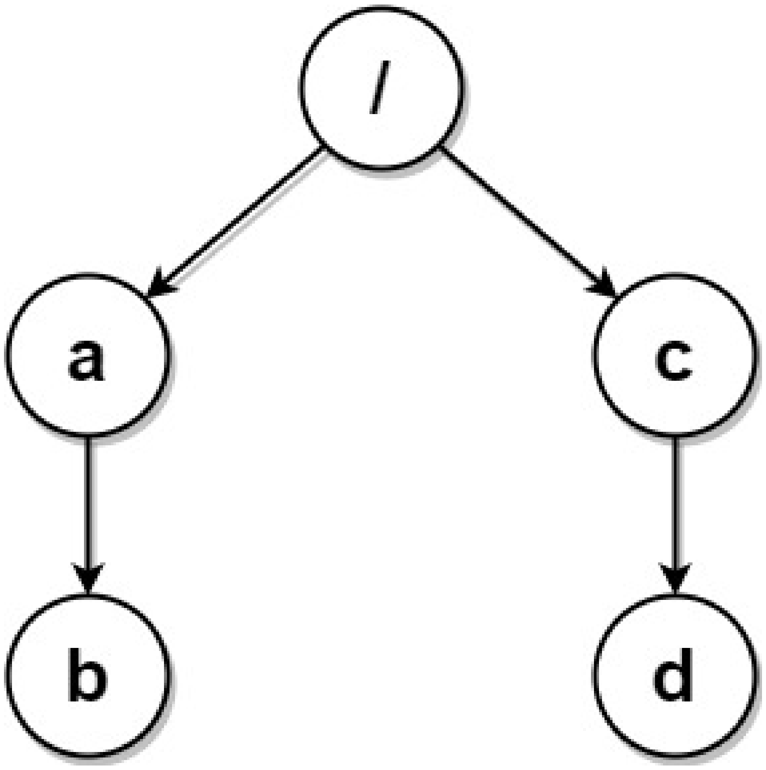
Output:

[["c"],["c","b"],["a"],["a","b"]]

Explanation:

The file structure is as shown. Folders "/a/b/x" and "/w" (and their subfolders) are marked for deletion because they both contain an empty folder named "y". Note that folders "/a" and "/c" are identical after the deletion, but they are not deleted because they were not marked beforehand.

Example 3:



Input:

paths = [["a","b"],["c","d"],["c"],["a"]]

Output:

[["c"],["c","d"],["a"],["a","b"]]

Explanation:

All folders are unique in the file system. Note that the returned array can be in a different order as the order does not matter.

Constraints:

1 <= paths.length <= 2 * 10

4

1 <= paths[i].length <= 500

1 <= paths[i][j].length <= 10

1 <= sum(paths[i][j].length) <= 2 * 10

5

path[i][j]

consists of lowercase English letters.

No two paths lead to the same folder.

For any folder not at the root level, its parent folder will also be in the input.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> deleteDuplicateFolder(vector<vector<string>>& paths) {

}
};
```

Java:

```
class Solution {
public List<List<String>> deleteDuplicateFolder(List<List<String>> paths) {

}
}
```

Python3:

```
class Solution:
def deleteDuplicateFolder(self, paths: List[List[str]]) -> List[List[str]]:
```

Python:

```
class Solution(object):
def deleteDuplicateFolder(self, paths):
"""
:type paths: List[List[str]]
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {string[][]} paths
* @return {string[][]}
*/
var deleteDuplicateFolder = function(paths) {

};
```

TypeScript:

```
function deleteDuplicateFolder(paths: string[][]): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> DeleteDuplicateFolder(IList<IList<string>> paths)
{

}

}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** deleteDuplicateFolder(char*** paths, int pathsSize, int*
pathsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func deleteDuplicateFolder(paths [][]string) [][]string {

}
```

Kotlin:

```
class Solution {
fun deleteDuplicateFolder(paths: List<List<String>>): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func deleteDuplicateFolder(_ paths: [[String]]) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn delete_duplicate_folder(paths: Vec<Vec<String>>) -> Vec<Vec<String>> {
}
}
```

Ruby:

```
# @param {String[][]} paths
# @return {String[][]}
def delete_duplicate_folder(paths)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $paths
* @return String[][]
*/
function deleteDuplicateFolder($paths) {

}
}
```

Dart:

```
class Solution {
List<List<String>> deleteDuplicateFolder(List<List<String>> paths) {

}
}
```

Scala:

```
object Solution {
def deleteDuplicateFolder(paths: List[List[String]]): List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec delete_duplicate_folder(paths :: [[String.t]]) :: [[String.t]]
def delete_duplicate_folder(paths) do

end
end
```

Erlang:

```
-spec delete_duplicate_folder(Paths :: [[unicode:unicode_binary()]]) ->
[[unicode:unicode_binary()]].
delete_duplicate_folder(Paths) ->
.
```

Racket:

```
(define/contract (delete-duplicate-folder paths)
(-> (listof (listof string?)) (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<vector<string>> deleteDuplicateFolder(vector<vector<string>>& paths) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<List<String>> deleteDuplicateFolder(List<List<String>> paths) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete Duplicate Folders in System
Difficulty: Hard
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def deleteDuplicateFolder(self, paths: List[List[str]]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def deleteDuplicateFolder(self, paths):
"""
:type paths: List[List[str]]
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[][]} paths
* @return {string[][]}
*/
var deleteDuplicateFolder = function(paths) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function deleteDuplicateFolder(paths: string[][]): string[][] {

};
```

C# Solution:

```csharp
/*
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<IList<string>> DeleteDuplicateFolder(IList<IList<string>> paths)
{

}
}
```

C Solution:

```
/*
* Problem: Delete Duplicate Folders in System
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** deleteDuplicateFolder(char*** paths, int pathsSize, int*
pathsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Delete Duplicate Folders in System
// Difficulty: Hard
// Tags: array, string, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func deleteDuplicateFolder(paths [][]string) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun deleteDuplicateFolder(paths: List<List<String>>): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func deleteDuplicateFolder(_ paths: [[String]]) -> [[String]] {

}
}
```

Rust Solution:

```
// Problem: Delete Duplicate Folders in System
// Difficulty: Hard
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn delete_duplicate_folder(paths: Vec<Vec<String>>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {String[][]} paths
# @return {String[][]}
def delete_duplicate_folder(paths)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $paths
* @return String[][]
*/
function deleteDuplicateFolder($paths) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> deleteDuplicateFolder(List<List<String>> paths) {

}
}
```

Scala Solution:

```
object Solution {
def deleteDuplicateFolder(paths: List[List[String]]): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec delete_duplicate_folder(paths :: [[String.t]]) :: [[String.t]]
def delete_duplicate_folder(paths) do
```

```
end
end
```

Erlang Solution:

```
-spec delete_duplicate_folder(Paths :: [[unicode:unicode_binary()]]) ->
[[unicode:unicode_binary()]].
delete_duplicate_folder(Paths) ->
.
```

Racket Solution:

```
(define/contract (delete-duplicate-folder paths)
(-> (listof (listof string?)) (listof (listof string?)))
)
```