1233. Remove Sub-Folders from the Filesystem

Given a list of folders `folder`, return *the folders after removing all* ***sub-folders*** *in those folders*. You may return the answer in **any order**.

If a `folder[i]` is located within another `folder[j]`, it is called a **sub-folder** of it. A sub-folder of `folder[j]` must start with `folder[j]`, followed by a `"/"`. For example, `"/a/b"` is a sub-folder of `"/a"`, but `"/b"` is not a sub-folder of `"/a/b/c"`.

The format of a path is one or more concatenated strings of the form: `'/'` followed by one or more lowercase English letters.

- For example, `"/leetcode"` and `"/leetcode/problems"` are valid paths while an empty string and `"/"` are not.

Example 1:

```
Input: folder = ["/a","/a/b","/c/d","/c/d/e","/c/f"]
Output: ["/a","/c/d","/c/f"]
Explanation: Folders "/a/b" is a subfolder of "/a" and "/c/d/e" is inside of folder
"/c/d" in our filesystem.
```

Example 2:

```
Input: folder = ["/a","/a/b/c","/a/b/d"]
Output: ["/a"]
Explanation: Folders "/a/b/c" and "/a/b/d" will be removed because they are
subfolders of "/a".
```

Example 3:

```
Input: folder = ["/a/b/c","/a/b/ca","/a/b/d"]
Output: ["/a/b/c","/a/b/ca","/a/b/d"]
```

Constraints:

- `1 <= folder.length <= 4 * 104`
- `2 <= folder[i].length <= 100`
- `folder[i]` contains only lowercase letters and `'/'`.
- `folder[i]` always starts with the character `'/'`.
- Each folder name is **unique**.

1233. Remove Sub-Folders from the Filesystem

```
/*
   Trie
   Time complexity: O(n.m)
   Space complexity: O(n.m)
   n: Number of folders
   m: Maximum length of a folder path
*/
class Trie{
   private:
      class TrieNode{
         public:
            std::unordered_map<std::string,TrieNode*> children;
            bool is_end_of_folder_path=false;
      };

      TrieNode* root;
   public:
      Trie(){
         root=new TrieNode();
      }

      ~Trie(){delete_trie(root);}

      // Delete the try to avoid memory leaks
      void delete_trie(TrieNode* root){
         if(!root) return;
         for(auto& [folder,node]: root->children){
            delete_trie(node);
         }
         delete root;
      }
```

```
void insert(std::string& path){
        TrieNode* cur_node=root;

        std::stringstream ss(path);
        std::string f;

        while(std::getline(ss,f,'/')){
            if(f.empty()) continue;
            TrieNode* node=cur_node->children[f];
            if(!node){
                node=new TrieNode();
                cur_node->children[f]=node;
            }
            cur_node=node;
        }

        cur_node->is_end_of_folder_path=true;
    }

    bool is_sub_folder(std::string& path){
        TrieNode* cur_node=root;

        std::stringstream ss(path);
        std::string f;

        while(std::getline(ss,f,'/')){
            if(f.empty()) continue;
            TrieNode* next_node=cur_node->children[f];

            // If the current folder is a path and is not the last folder
            if(next_node->is_end_of_folder_path && ss.tellg()!=-1) return true;

            cur_node=next_node;
        }

        return false;
    }
};
```

```
typedef std::vector<std::string> vs;

class Solution {
   public:

      vs removeSubfolders(vs& folder){
         Trie trie=Trie();

         // For each path insert the folders in the trie
         for(auto& path: folder) trie.insert(path);

         // For each pathr, check if it is a subfolder
         // It it is, don't insert it
         vs ans;
         for(auto& path: folder){
            if(!trie.is_sub_folder(path)) ans.push_back(path);
         }

         return ans;
      }
};
```