Longest Word in Dictionary

Given a list of strings `words` representing an English Dictionary, find the longest word in `words` that can be built one character at a time by other words in `words`. If there is more than one possible answer, return the longest word with the smallest lexicographical order.

If there is no answer, return the empty string.

Example 1:

```
Input:
words = ["w","wo","wor","worl", "world"]
Output: "world"
Explanation:
The word "world" can be built one character at a time by "w", "wo", "wor", and "worl"
.
```

Example 2:

```
Input:
words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]
Output: "apple"
Explanation:
Both "apply" and "apple" can be built from other words in the dictionary. However, "a
pple" is lexicographically smaller than "apply".
```

Note:

- All the strings in the input will only contain lowercase letters.
- The length of `words` will be in the range `[1, 1000]`.
- The length of `words[i]` will be in the range `[1, 30]`.

Solution 1

1. Sort the words alphabetically, therefore shorter words always comes before longer words;
2. Along the sorted list, populate the words that can be built;
3. Any prefix of a word must comes before that word.

Java

```java
class Solution {
    public String longestWord(String[] words) {
        Arrays.sort(words);
        Set<String> built = new HashSet<String>();
        String res = "";
        for (String w : words) {
            if (w.length() == 1 || built.contains(w.substring(0, w.length() - 1)))
{
                res = w.length() > res.length() ? w : res;
                built.add(w);
            }
        }
        return res;
    }
}
```

C++

```cpp
class Solution {
public:
    string longestWord(vector<string>& words) {
        sort(words.begin(), words.end());
        unordered_set<string> built;
        string res;
        for (string w : words) {
            if (w.size() == 1 || built.count(w.substr(0, w.size() - 1))) {
                res = w.size() > res.size() ? w : res;
                built.insert(w);
            }
        }
        return res;
    }
};
```

written by alexander original link here

Solution 2

The idea is to first sort the words, and once sorted, add each word to the resultset, if the prefix of the word word[:-1] is there in the sortedset.

By commutative property, if the prefix is there in the resultset, then that implies all the prefixes of length 1, 2, 3 .. are also there in the resultset, due to sorted data.

Also maintaining a global res_word and updating it every time we add a word to the resultset, makes it easy to find the final result.

```
    def longestWord(self, words):
        words, resword, res = sorted(words), '', set()
        for word in words:
            if len(word) == 1 or word[:-1] in res:
                res.add(word)
                resword = word if resword == '' else word if len(word) > len(reswor
d) else resword
        return resword
```

written by johnyrufus16 original link here

Solution 3

```
class Solution {
    class TrieNode{
        public char val;
        public TrieNode[] hash;
        public boolean isWord;
        public TrieNode(){
            this.val='\u0000';
            this.hash=new TrieNode[26];
            this.isWord=false;
        }
        public TrieNode(char c){
            this.val=c;
            this.hash=new TrieNode[26];
            this.isWord=false;
        }
        public StringBuilder dfs(StringBuilder res){

            StringBuilder max=new StringBuilder();
            for(int i=0;i<26;i++){
                if(hash[i]!=null && hash[i].isWord){
                    StringBuilder temp=new StringBuilder();
                    temp.append(hash[i].val);
                    temp.append(hash[i].dfs(temp));
                    if(temp.length() > max.length())
                        max=temp;
                }
            }
            return max;
        }

    }
    public String longestWord(String[] words) {
        TrieNode root=new TrieNode();
        for(String word:words){
            TrieNode curr=root;
            for(int i=0;i<word.length();i++){
                if(curr.hash[word.charAt(i)-'a']==null){
                    TrieNode temp=new TrieNode(word.charAt(i));
                    curr.hash[word.charAt(i)-'a']=temp;
                }
                curr=curr.hash[word.charAt(i)-'a'];
                if(i==word.length()-1)
                    curr.isWord=true;
            }
        }
        StringBuilder res=new StringBuilder();
        res=root.dfs(res);

        return res.toString();
    }
}
```

written by ramya8 original link here