Alien Dictionary (hard)

We'll cover the following

Problem Statement

There is a dictionary containing words from an alien language for which we don't know the ordering of the alphabets. Write a method to find the correct order of the alphabets in the alien language. It is given that the input is a valid dictionary and there exists an ordering among its alphabets.

Example 1:

```
Input: Words: ["ba", "bc", "ac", "cab"]
Output: bac
Explanation: Given that the words are sorted lexicographically by the rules of the alien langua
ge, so
from the given words we can conclude the following ordering among its characters:

1. From "ba" and "bc", we can conclude that 'a' comes before 'c'.
2. From "bc" and "ac", we can conclude that 'b' comes before 'a'

From the above two points, we can conclude that the correct character order is: "bac"
```

Example 2:

```
Input: Words: ["cab", "aaa", "aab"]
Output: cab
Explanation: From the given words we can conclude the following ordering among its characters:

1. From "cab" and "aaa", we can conclude that 'c' comes before 'a'.
2. From "aaa" and "aab", we can conclude that 'a' comes before 'b'

From the above two points, we can conclude that the correct character order is: "cab"
```

Example 3:

```
Input: Words: ["ywx", "wz", "xww", "xz", "zyy", "zwz"]
Output: ywxz
Explanation: From the given words we can conclude the following ordering among its characters:

1. From "ywx" and "wz", we can conclude that 'y' comes before 'w'.
2. From "wz" and "xww", we can conclude that 'w' comes before 'x'.
3. From "xww" and "xz", we can conclude that 'w' comes before 'z'
4. From "xz" and "zyy", we can conclude that 'x' comes before 'z'
5. From "zyy" and "zwz", we can conclude that 'y' comes before 'w'

From the above five points, we can conclude that the correct character order is: "ywxz"
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
import java.util.*;

class AlienDictionary {
  public static String findOrder(String[] words) {
    // TODO: Write your code here
    return "";
  }

  public static void main(String[] args) {
    String result = AlienDictionary.findOrder(new String[] { "ba", "bc", "ac", "cab" });
    System.out.println("Character order: " + result);
```

```java

    result = AlienDictionary.findOrder(new String[] { "cab", "aaa", "aab" });
    System.out.println("Character order: " + result);

    result = AlienDictionary.findOrder(new String[] { "ywx", "wz", "xww", "xz", "zyy", "zwz" });
    System.out.println("Character order: " + result);
  }
}
```

Run Save Reset

Solution

Since the given words are sorted lexicographically by the rules of the alien language, we can always compare two adjacent words to determine the ordering of the characters. Take Example-1 above: [“ba”, “bc”, “ac”, “cab”]

1. Take the first two words “ba” and “bc”. Starting from the beginning of the words, find the first character that is different in both words: it would be ‘a’ from “ba” and ‘c’ from “bc”. Because of the sorted order of words (i.e. the dictionary!), we can conclude that ‘a’ comes before ‘c’ in the alien language.
2. Similarly, from “bc” and “ac”, we can conclude that ‘b’ comes before ‘a’.

These two points tell us that we are actually asked to find the topological ordering of the characters, and that the ordering rules should be inferred from adjacent words from the alien dictionary.

This makes the current problem similar to Tasks Scheduling Order, the only difference being that we need to build the graph of the characters by comparing adjacent words first, and then perform the topological sort for the graph to determine the order of the characters.

Code

Here is what our algorithm will look like (only the highlighted lines have changed):

Java Python3 C++ JS

```java
import java.util.*;

class AlienDictionary {
  public static String findOrder(String[] words) {
    if (words == null || words.length == 0)
      return "";

    // a. Initialize the graph
    HashMap<Character, Integer> inDegree = new HashMap<>(); // count of incoming edges for every vertex
    HashMap<Character, List<Character>> graph = new HashMap<>(); // adjacency list graph
    for (String word : words) {
      for (char character : word.toCharArray()) {
        inDegree.put(character, 0);
        graph.put(character, new ArrayList<Character>());
      }
    }

    // b. Build the graph
    for (int i = 0; i < words.length - 1; i++) {
      String w1 = words[i], w2 = words[i + 1]; // find ordering of characters from adjacent words
      for (int j = 0; j < Math.min(w1.length(), w2.length()); j++) {
        char parent = w1.charAt(j), child = w2.charAt(j);
        if (parent != child) { // if the two characters are different
          graph.get(parent).add(child); // put the child into it's parent's list
          inDegree.put(child, inDegree.get(child) + 1); // increment child's inDegree
          break; // only the first different character between the two words will help us find the order
        }
      }
```

Run Save Reset

Time complexity

In step ‘d’, each task can become a source only once and each edge (a rule) will be accessed and removed once. Therefore, the time complexity of the above algorithm will be $O(V+E)$, where ‘V’ is the total number of different characters and ‘E’ is the total number of the rules in the alien language. Since, at most, each pair of words can give us one rule, therefore, we can conclude that the upper bound for the rules is $O(N)$ where ‘N’ is the number of words in the input. So, we can say that the time complexity of our algorithm is $O(V+N)$.

Space complexity

The space complexity will be $O(V+N)$, since we are storing all of the rules for each character in an adjacency list.