

# Alien Dictionary (hard)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time complexity
  - Space complexity

## Problem Statement #

There is a dictionary containing words from an alien language for which we don't know the ordering of the characters. Write a method to find the correct order of characters in the alien language.

### Example 1:

Input: Words: ["ba", "bc", "ac", "cab"]

Output: bac

Explanation: Given that the words are sorted lexicographically by the rules of the alien language, 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++



## 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

This makes the current problem similar to Tasks Scheduling Order

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5066018374287360/>), 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
--	---	---	--

```

1  from collections import deque
2
3
4  def find_order(words):
5      if len(words) == 0:
6          return ""
7
8      # a. Initialize the graph
9      inDegree = {} # count of incoming edges
10     graph = {} # adjacency list graph
11     for word in words:
12         for character in word:
13             inDegree[character] = 0
14             graph[character] = []
15
16     # b. Build the graph
17     for i in range(0, len(words)-1):
18         # find ordering of characters from adjacent words
19         w1, w2 = words[i], words[i + 1]
20         for j in range(0, min(len(w1), len(w2))):
21             parent, child = w1[j], w2[j]
22             if parent != child: # if the two characters are different
23                 # put the child into it's parent's list
24                 graph[parent].append(child)
25                 inDegree[child] += 1 # increment child's inDegree
26                 break # only the first different character between the two words will help us find
27
28     # c. Find all sources i.e., all vertices with 0 in-degrees
29     sources = deque()
30     for key in inDegree:
31         if inDegree[key] == 0:
32             sources.append(key)
33
34     # d. For each source, add it to the sortedOrder and subtract one from all of its children
35     # if a child's in-degree becomes zero, add it to the sources queue
36     sortedOrder = []
37     while sources:
38         vertex = sources.popleft()
39         sortedOrder.append(vertex)
40         for child in graph[vertex]: # get the node's children to decrement their in-degrees
41             inDegree[child] -= 1
42             if inDegree[child] == 0:
43                 sources.append(child)
44
45     # if sortedOrder doesn't contain all characters, there is a cyclic dependency between characters
46     # will not be able to find the correct ordering of the characters
47     if len(sortedOrder) != len(inDegree):

```

```

47     if len(sortedOrder) != len(indegree):
48         return ""
49
50     return ''.join(sortedOrder)
51
52
53 def main():
54     print("Character order: " + find_order(["ba", "bc", "ac", "cab"]))
55     print("Character order: " + find_order(["cab", "aaa", "aab"]))
56     print("Character order: " + find_order(["ywx", "wz", "xww", "xz", "zyy", "zwz"]))
57
58
59 main()
60

```



### 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.

← Back

Next →

All Tasks Scheduling Orders (hard)

Problem Challenge 1

✓ Completed



Report an Issue



Ask a Question

([https://discuss.educative.io/tag/alien-dictionary-hard\\_\\_pattern-topological-sort-graph\\_\\_grokking-the-coding-interview-patterns-for-coding-questions](https://discuss.educative.io/tag/alien-dictionary-hard__pattern-topological-sort-graph__grokking-the-coding-interview-patterns-for-coding-questions))