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 languag
e, 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++

```python
def find_order(words):
  # TODO: Write your code here
  return ""


def main():
  print("Character order: " + find_order(["ba", "bc", "ac", "cab"]))
  print("Character order: " + find_order(["cab", "aaa", "aab"]))
  print("Character order: " + find_order(["ywx", "wz", "xww", "xz", "zyy", "zwz"]))


main()

```

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

```python
from collections import deque


def find_order(words):
  if len(words) == 0:
    return ""

  # a. Initialize the graph
  inDegree = {}  # count of incoming edges
  graph = {}  # adjacency list graph
  for word in words:
    for character in word:
      inDegree[character] = 0
      graph[character] = []

  # b. Build the graph
  for i in range(0, len(words)-1):
    # find ordering of characters from adjacent words
    w1, w2 = words[i], words[i + 1]
    for j in range(0, min(len(w1), len(w2))):
      parent, child = w1[j], w2[j]
      if parent != child:  # if the two characters are different
        # put the child into it's parent's list
        graph[parent].append(child)
        inDegree[child] += 1  # increment child's inDegree
        break  # only the first different character between the two words will help us find the order

  # c. Find all sources i.e., all vertices with 0 in-degrees
  sources = deque()
```

```
  for key in inDegree:
    if inDegree[key] == 0:
      sources.append(key)

  # d. For each source, add it to the sortedOrder and subtract one from all of its children's in-degrees
  # if a child's in-degree becomes zero, add it to the sources queue
  sortedOrder = []
  while sources:
    vertex = sources.popleft()
    sortedOrder.append(vertex)
    for child in graph[vertex]:  # get the node's children to decrement their in-degrees
      inDegree[child] -= 1
      if inDegree[child] == 0:
        sources.append(child)

  # if sortedOrder doesn't contain all characters, there is a cyclic dependency between characters, there
  # will not be able to find the correct ordering of the characters
  if len(sortedOrder) != len(inDegree):
    return ""

  return ''.join(sortedOrder)


def main():
  print("Character order: " + find_order(["ba", "bc", "ac", "cab"]))
  print("Character order: " + find_order(["cab", "aaa", "aab"]))
  print("Character order: " + find_order(["ywx", "wz", "xww", "xz", "zyy", "zwz"]))


main()

```

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.

Back

All Tasks Scheduling Orders (hard)

Next

Problem Challenge 1

Completed

Report an Issue Ask a Question