HARD

Alien Dictionary

Intuition

It states that we are given n strings and we are given a sequence.

Eg.

In English language :

Input n - words	relative ordering
“Baa”	“abca”
“abcd”	“abcd”
“abca”	“baa”
“cab”	“cab”
“cad”	“cad”

This happens because
A appears before b
A appears before d
B appears before c
B appears before d

But we are given an alien dictionary word and we have to find out the alien order

We are given only first 4 element from english dictionary

A B C D

But in alien they have
B appears before a
A appears before c
D appears before a
B appears before d

Bdac is therefore the alien order
We have to return [B D A C]

This can be done using the application of standard topological sorting technique

There are first 4 core alphabets from english dictionary therefore we can assign them as numbers

Eg.

A	B	C	D
0	1	2	3

At first we will pick up first two words

BAA before ABCD
Because aliens saying B before A

1 -> 0 is an edge

We do not need to check it again

Now for next 2

ABCD before ABCA
Because A = A | B = B | C = C | D is appearing before A

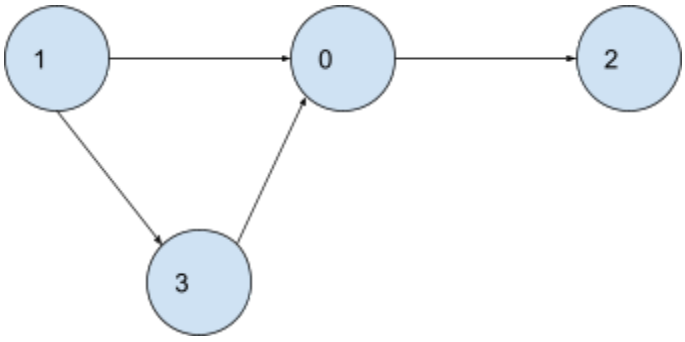
3 -> 0

ABCA before CAB
A appears before C

0 -> 2

CAB before CAD
C = C | A = A | B before D

1 -> 3



If there is not given any other alphabet then we will insert them as single non-connected nodes and they don't matter.

Creating an adjacency list :

0 - [2]
1 - [0, 3]
2 - [ ]
3 - [0]

Performing topological sort we will get :

! 3 0 2

We need to find differentiating factor between corresponding words

Ie.

S1 = arr[i]; // BAA
S2 = arr[i+1]; // ABCD

B != A : addedge(B->A)

Case when the relative ordering is not possible

The relative ordering is not possible in either of two cases :

- When the two strings have complete matching characters but the shorter string is placed after the longer string ,in this case the dictionary is faulted
- When there is cyclic dependency ie.

  B A C
  A E D
  B C E
  B before A and A before B not possible

Approach

topoSort( )

- Create indegree vector
- Initialize indegree vector with indegree of elements
- Create empty queue
- Insert all the elements with 0 indegree into the queue
- Traverse until the queue becomes empty :
  - Extract the first node from the queue
  - Pop the first node from the queue

        - Insert the first node into the result vector
        - Traverse for the adjacent elements of the node :
            - Reduce the indegree of the adjacent nodes
            - Check if indegree becomes 0 then push the node to the queue
    - Return the result vector

isPossible( )

- Create an empty adjacency list vector of vector
- Traverse for all elements in dictionary as pairs of i and i+1 :
    - Extract string 1 as dict[i]
    - Extract string 2 as dict[i+1]
    - Find minimum length of the two strings
    - Traverse for minimum length of the two strings :
        - Check if the characters are different then append the elements s2-'a' at s1-'a' in the adjacency list // do not check further break
- Sort them using topological sort and store their relative order into result vector
- Convert the result vector back to string
- Return the string

Function Code

```
vector<int> topoSort(int n, vector<int> adj[]) {
    // creating indegree vector
    vector<int> indegree(n, 0);
    // initializing indegree vector
    for (int i = 0; i < n; i++) {
        for (int j : adj[i]) {
            indegree[j] += 1;
        }
    }
    // declaring an empty queue
    queue<int> q;
    // adding elements with 0 indegree to the queue
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }
    // creating a result vector
    vector<int> result;
    // traversing until the queue becomes empty
    while (!q.empty()) {
        // extracting and popping the first element of the queue
        int node = q.front();
        q.pop();
```

```
        // inserting the first element into the result vector
        result.push_back(node);
        // traversing for adjacent elements
        for (int i : adj[node]) {
            // reducing the indegree
            indegree[i] -= 1;
            // if the indegree becomes 0 pushing element to queue
            if (indegree[i] == 0) {
                q.push(i);
            }
        }
    }
    // returning the result vector
    return result;
}

string findOrder(string dict[], int N, int K) {
    // creating an adjacency list
    vector<int> adj[K];
    // traversing through the words in pairs of i and i+1
    for(int i = 0; i < N - 1; i++) {
        // extracting strings i and i+1
        string s1 = dict[i];
        string s2 = dict[i + 1];
        // storing the size of minimum comparison for string
        int len = min(s1.size(), s2.size());
        // traversing through the minimum length
        for(int ptr = 0; ptr < len; ptr++) {
            // performing a check for strings and pushing the different
elements before other as below
            if(s1[ptr] != s2[ptr]) {
                adj[s1[ptr] - 'a'].push_back(s2[ptr] - 'a');
                break; // Once we find a differing character, we don't need
to check the rest
            }
        }
    }

    // performing topological sort
    vector<int> topo = topoSort(K, adj);
    // convert result of topological sort from integer vector to strings
    string ans = "";
    for(auto it: topo) {
```

```
        ans += char(it + 'a');
    }
    // returning result
    return ans;
}
```

Time Complexity

O(V+E)