November 9, 2022 ▪ Data Structure / Graph

Alien Dictionary – Topological Sort: G-26

Problem Statement: Given a sorted dictionary of an alien language having N words and k starting alphabets of a standard dictionary. Find the order of characters in the alien language.

Note: Many orders may be possible for a particular test case, thus you may return any valid order.

Examples:

```
Example 1:
Input: N = 5, K = 4
dict =
{"baa","abcd","abca","cab","cad"}
Output: b d a c
Explanation:
We will analyze every
consecutive pair to find out the
```

Search

Search

Recent Posts

2023 – Striver's SDE Sheet Challenge

Top Array Interview Questions – Structured Path with Video Solutions

Longest Subarray with sum K | [Postives and Negatives]

```
order of the characters.
The pair “baa” and “abcd”
suggests ‘b’ appears before ‘a’
in the alien dictionary.
The pair “abcd” and “abca”
suggests ‘d’ appears before ‘a’
in the alien dictionary.
The pair “abca” and “cab”
suggests ‘a’ appears before ‘c’
in the alien dictionary.
The pair “cab” and “cad”
suggests ‘b’ appears before ‘d’
in the alien dictionary.
So, [‘b’, ‘d’, ‘a’, ‘c’] is a
valid ordering.

Example 2:
Input: N = 3, K = 3
dict = {"caa","aaa","aab"}
Output: c a b
Explanation: Similarly, if we
analyze the consecutive pair
for this example, we will figure
out [‘c’, ‘a’, ‘b’] is
a valid ordering.
```

***Disclaimer**: Don’t jump directly to the solution, try it out yourself first.*

Problem Link.

Solution:

Let’s consider the first example where **N** = 5; **K** = 4 and **dict** = {“baa”, “abcd”, “abca”, “cab”, “cad”}. So, here we need to find out the correct ordering of the first 4 letters of the alphabet(i.e. ‘a’, ‘b’, ‘c’, ‘d’). If we consider the first 2 words and try to figure out why “baa”

Count Subarray sum Equals K

Binary Tree Representation in Java

Accolite Digital Amazon Arcesium arrays Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing infosys inorder Interview Experience Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCS TCS CODEVITA TCS Ninja

appears before “abcd”, we can clearly observe that they are differentiated by the first letter i.e. ‘b’ and ‘a’. That is why, we can conclude that in the alien dictionary, ***‘b’ appears before ‘a’(i.e. ‘b’ is smaller than ‘a’).***
We can correspond this differentiating factor to a ***directed graph*** like the following:

Let’s understand *why we need not check* ***“baa” and “abca” (the 1st and the 3rd word) next:***
Until now, we have figured out why “baa” appears before “abcd”. So, by convention, if “abcd” is appearing before “abca” and “baa” is appearing before “abcd”, ***“baa” will obviously appear before “abca”***. That is why we will check the pair of “abcd” and “abca” next rather than checking “baa” with any other words and this flow will be continued for the rest of the words.

Note: *Points to remember that we need not check every pair of words rather we will just check the consecutive pair of words in the dictionary.* Comparing each pair of consecutive words in the dictionary, we can construct a directed graph like the following:

Now, we have successfully reduced the problem to a known ***directed graph problem***. If we look at the problem from the graph point of view, we just need to find out the ***linear ordering of the nodes of the directed graph***. And we can do this easily using the topological sort algorithm which we have previously learned.

To further simplify the problem, we will denote the alphabet with numbers like: 'a' with 0, 'b' with 1, 'c' with 2, and so on. For example, if the letter is 'z', we will denote it using 25. Finally, the directed graph will look like the following illustration:

Note: *The intuition is to check every consecutive pair of words and find out the differentiating factor. With these factors, we will form a directed graph, and the whole problem balls down to a topological sort of problem.*

Edge Case: The problem arises when the value of K becomes 5 and there is no word in the dictionary containing the letter **'e'**. In this case, we will add a separate node with the value 'e' in the graph and it will be considered a component of the directed graph like the following, and the same algorithm will work fine for multiple components.

Note: *If the value of K is greater than the number of unique characters appearing in the dictionary, then the extra characters will be considered the different components of the directed graph formed.*

The follow-up question for the interview:

- **When is the ordering not possible?**
  There are two such cases when ordering is not possible:

    - **If every character matches and the largest word appears before the shortest word:** For example, if “abcd” appears before “abc”, we can say the ordering is not possible.
    - **If there exists a cyclic dependency between the characters:** For example, in the dictionary: **dict: {“abc”, “bat”, “ade”}** there exists a cyclic dependency between ‘a’ and ‘b’ because the dictionary states **‘a’ < ‘b’ < ‘a’**, which is not possible.

Approach:

We will apply the BFS(Breadth First Search) traversal technique. Breadth First Search or BFS is a traversal technique where we visit the nodes level-wise, i.e., it visits the same level nodes simultaneously, and then moves to the next level.

Initial Configuration:

Adjacency List: Initially it will be empty and we will create this adjacency list comparing the consecutive pair of words.

Indegree Array: Initially all elements are set to 0. Then, We will count the incoming edges

for a node and store it in this array. For example, if the indegree of node 3 is 2, indegree[3] = 2.

Queue: As we will use BFS, a queue is required. Initially, the node with indegree 0 will be pushed into the queue.

Answer array(topo): Initially empty and is used to store the linear ordering.

The algorithm steps are as follows:

1. First, we need to create the adjacency list for the graph. The steps are the following:
    1. We will run a loop from the starting index to the **second last index** because we will check the i^{th} element and the $(i+1)^{th}$ element.
    2. Inside the loop, we will pick two words (the word at the current index(**s1**) and the word at the next index(**s2**)). For comparing them, we will again run a loop up to the length of the smallest string.
    3. Inside that second loop, if at any index we find inequality (**s1[i] != s2[i]**), we will add them to the adjacency list (**s1[i] —> s2[i]**) in terms of numbers(subtracting 'a' from them), and then we will immediately come out of the loop.

    4. This is only because we want the first differentiating character. Finally, we will get the adjacency list.
    5. In short, we need to find the differentiating character for adjacent strings and create the graph.
2. Once the graph is created, simply perform a topo sort, whose steps are given below.
3. Then, we will calculate the indegree of each node and store it in the indegree array. We can iterate through the given adj list, and simply for every node u->v, we can increase the indegree of v by 1 in the indegree array.
4. Initially, there will be always at least a single node whose indegree is 0. So, we will push the node(s) with indegree 0 into the queue.
5. Then, we will pop a node from the queue including the node in our answer array, and for all its adjacent nodes, we will decrease the indegree of that node by one. For example, if node u that has been popped out from the queue has an edge towards node v(u->v), we will decrease indegree[v] by 1.
6. After that, if for any node the indegree becomes 0, we will push that node again into the queue.
7. We will repeat steps 3 and 4 until the queue is completely empty. Finally,

completing the BFS we will get the linear ordering of the nodes in the answer array.

8. For the final answer, we will iterate on the answer array and add each element in terms of character(adding 'a' to each of them) to the final string. Then we will return the string as our final answer.

Note: *If you wish to see the dry run of the above approach, you can watch the video attached to this article.*

Code:

C++ Code

```
#include <bits/stdc++.h>
using namespace std;



class Solution {
    // works for multiple components
private:
    vector<int> topoSort(int V, vect
    {
        int indegree[V] = {0};
        for (int i = 0; i < V; i++)
            for (auto it : adj[i]) {
                indegree[it]++;
            }
        }

        queue<int> q;
        for (int i = 0; i < V; i++)
            if (indegree[i] == 0) {
                q.push(i);
            }
        }
        vector<int> topo;
        while (!q.empty()) {
            int node = q.front();
```

```
                q.pop();
                topo.push_back(node);
                // node is in your topo
                // so please remove it f

                for (auto it : adj[node]
                    indegree[it]--;
                    if (indegree[it] ==
                }
            }

            return topo;
        }
    public:
        string findOrder(string dict[],
            vector<int>adj[K];
            for (int i = 0; i < N - 1; i
                string s1 = dict[i];
                string s2 = dict[i + 1];
                int len = min(s1.size(),
                for (int ptr = 0; ptr <
                    if (s1[ptr] != s2[pt
                        adj[s1[ptr] - 'a
                        break;
                    }
                }
            }

            vector<int> topo = topoSort(
            string ans = "";
            for (auto it : topo) {
                ans = ans + char(it + 'a
            }
            return ans;
        }
    };

    int main() {

        int N = 5, K = 4;
        string dict[] = {"baa", "abcd",
        Solution obj;
        string ans = obj.findOrder(dict,

        for (auto ch : ans)
            cout << ch << ' ';
```

```
        cout << endl;

        return 0;
    }
```

Output: b d a c

Time Complexity: O(N*len)+O(K+E), where N is the number of words in the dictionary, 'len' is the length up to the index where the first inequality occurs, K = no. of nodes, and E = no. of edges.

Space Complexity: O(K) + O(K)+O(K)+O(K) ~ O(4K), O(K) for the indegree array, and O(K) for the queue data structure used in BFS(where K = no.of nodes), O(K) for the answer array and O(K) for the adjacency list used in the algorithm.

Java Code

```
import java.util.*;


class Solution {
    private List<Integer> topoSort(i
        int indegree[] = new int[V];
        for (int i = 0; i < V; i++)
            for (int it : adj.get(i)
                indegree[it]++;
            }
        }

        Queue<Integer> q = new Linke
        for (int i = 0; i < V; i++)
            if (indegree[i] == 0) {
                q.add(i);
            }
        }
```

```
        List<Integer> topo = new Arr
        while (!q.isEmpty()) {
            int node = q.peek();
            q.remove();
            topo.add(node);
            // node is in your topo
            // so please remove it f

            for (int it : adj.get(no
                indegree[it]--;
                if (indegree[it] ==
            }
        }

        return topo;
    }
    public String findOrder(String [
        List<List<Integer>> adj = ne
        for (int i = 0; i < K; i++)
            adj.add(new ArrayList<>(
        }


        for (int i = 0; i < N - 1; i
            String s1 = dict[i];
            String s2 = dict[i + 1];
            int len = Math.min(s1.le
            for (int ptr = 0; ptr <
                if (s1.charAt(ptr) !
                    adj.get(s1.charA
                    break;
                }
            }
        }

        List<Integer> topo = topoSor
        String ans = "";
        for (int it : topo) {
            ans = ans + (char)(it +
        }

        return ans;

    }
}
```

```
public class tUf {
    public static void main(String[]
        int N = 5, K = 4;
        String[] dict = {"baa", "abc
        Solution obj = new Solution(
        String ans = obj.findOrder(d

        for (int i = 0; i < ans.leng
            System.out.print(ans.cha
        }
        System.out.println("");
    }
}
```

Output: b d a c

Time Complexity: O(N*len)+O(K+E), where N is the number of words in the dictionary, 'len' is the length up to the index where the first inequality occurs, K = no. of nodes, and E = no. of edges.

Space Complexity: O(K) + O(K)+O(K)+O(K) ~ O(4K), O(K) for the indegree array, and O(K) for the queue data structure used in BFS(where K = no.of nodes), O(K) for the answer array and O(K) for the adjacency list used in the algorithm.

> Special thanks to **KRITIDIPTA GHOSH** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com

« Previous Post
Find Eventual Safe States – BFS – Topological Sort: G-25

Next Post »
G-30 : Word Ladder-II

Load Comments