<https://www.lintcode.com/problem/892>

**Description**

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where words are **sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

1. You may assume all letters are in lowercase.
2. **The dictionary is invalid, if string a is prefix of string b and b is appear before a.**
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return the smallest in normal lexicographical order.
5. The letters in **one** string are of the same rank by default and are sorted in Human dictionary order.

**Example**

**Example 1:**

Input：["wrt","wrf","er","ett","rftt"]

Output："wertf"

Explanation：

from "wrt"and"wrf" ,we can get 't'<'f'

from "wrt"and"er" ,we can get 'w'<'e'

from "er"and"ett" ,we can get 'r'<'t'

from "ett"and"rftt" ,we can get 'e'<'r'

So return "wertf"

**Example 2:**

Input：["z","x"]

Output："zx"

Explanation：

from "z" and "x"，we can get 'z' < 'x'

So return "zx"

**Example 3:**

Input：["z","x","z"]

Output：""

The order is invalid, so return ""

**Attempt 1: 2022-11-21**

**Solution 1:  Detect Cycle in a Directed Graph using BFS [Topological Sort ] (120min)**

public class Solution {

/\*\*

\* @param words: a list of words

\* @return: a string which is correct order

\*/

public String alienOrder(String[] words) {

Map<Character, Set<Character>> graph = new HashMap<Character, Set<Character>>();

int[] indegree = new int[26];

boolean validDictionary = buildGraph(words, graph, indegree);

String order = topologicalSort(graph, indegree);

if(validDictionary) {

return order.length() == graph.size() ? order : "";

} else {

return "";

}

}

private boolean buildGraph(String[] words, Map<Character, Set<Character>> graph, int[] indegree) {

for(String word : words) {

for(Character c : word.toCharArray()) {

graph.put(c, new HashSet<Character>());

}

}

// Compare the first different character between two words

// e.g a -> b, a has an out edge point to b

for(int i = 1; i < words.length; i++) {

String first = words[i - 1];

String second = words[i];

int length = Math.min(first.length(), second.length());

// Corner case:

// The dictionary is invalid, if string a is prefix of string b and b is appear before a.

// 'first' is string b, 'second' is string a

int j;

for(j = 0; j < length; j++) {

char parent = first.charAt(j);

char child = second.charAt(j);

if(parent != child) {

if(!graph.get(parent).contains(child)) {

graph.get(parent).add(child);

indegree[child - 'a']++;

}

break;

}

}

// When 'j' equal to string a's length, means string a is prefix of string b

// return 'false' as invalid dictionary

if(j == second.length()) {

return false;

}

}

return true;

}

private String topologicalSort(Map<Character, Set<Character>> graph, int[] indegree) {

PriorityQueue<Character> queue = new PriorityQueue<Character>();

for(Character c : graph.keySet()) {

if(indegree[c - 'a'] == 0) {

queue.offer(c);

}

}

StringBuilder sb = new StringBuilder();

while(!queue.isEmpty()) {

Character c = queue.poll();

sb.append(c);

if(graph.get(c) == null || graph.get(c).size() == 0) {

continue;

}

for(char neighbor : graph.get(c)) {

indegree[neighbor - 'a']--;

if(indegree[neighbor - 'a'] == 0) {

queue.offer(neighbor);

}

}

}

return sb.toString();

}

}

**Corner cases:**

Input：["xyz","xy"]

Output：""

Explaination:

The dictionary is invalid, if string a is prefix of string b and b is appear before a.

In above input, "xyz" is string b, "xy" is string a, "xyz" (string b) appears before "xy" (string a), hence dictionary invalid

Input：["xy","xyz"]

Output："xyz"

Explaination:

Opposite to above example, if we switch the position of "xy" and "xyz", let's say "xy" is string a, "xyz" is string b, even "xy" appears before "xyz", but not satisfy string b appears before a, hence dictionary is valid

**Solution 2:  Detect Cycle in a Directed Graph using DFS [Topological Sort ] (120min)**

**Not understanding !!!**

**To pass below test case on LintCode, we have to reverse edge connection from "parent -> child" to "child -> parent" and reverse final string as result**

**Input Data: ["zy","zx"]**

**Output Data: "zyx"**

**Expected: "yxz"**

public class Solution {

/\*\*

\* @param words: a list of words

\* @return: a string which is correct order

\*/

public String alienOrder(String[] words) {

Map<Character, Set<Character>> graph = new HashMap<Character, Set<Character>>();

boolean validDictionary = buildGraph(words, graph);

String order = topologicalSort(graph);

if(validDictionary) {

return order.length() == graph.size() ? order : "";

} else {

return "";

}

}

private boolean buildGraph(String[] words, Map<Character, Set<Character>> graph) {

for(String word : words) {

for(Character c : word.toCharArray()) {

graph.put(c, new HashSet<Character>());

}

}

// Compare the first different character between two words

// e.g a -> b, a has an out edge point to b

for(int i = 1; i < words.length; i++) {

String first = words[i - 1];

String second = words[i];

int length = Math.min(first.length(), second.length());

// Corner case:

// The dictionary is invalid, if string a is prefix of string b and b is appear before a.

// 'first' is string b, 'second' is string a

int j;

for(j = 0; j < length; j++) {

char parent = first.charAt(j);

char child = second.charAt(j);

if(parent != child) {

//if(!graph.get(parent).contains(child)) {

// graph.get(parent).add(child);

//}

// Not understanding !!!

// To pass below test case, we have to reverse edge connection from "parent -> child"

// to "child -> parent" and reverse final string as result

// Input Data: ["zy","zx"]

// Output Data: "zyx"

// Expected: "yxz"

if(!graph.get(child).contains(parent)) {

graph.get(child).add(parent);

}

break;

}

}

// When 'j' equal to string a's length, means string a is prefix of string b

// return 'false' as invalid dictionary

if(j == second.length()) {

return false;

}

}

return true;

}

private String topologicalSort(Map<Character, Set<Character>> graph) {

List<Character> tmp = new ArrayList<Character>();

boolean[] visited = new boolean[26];

boolean[] recursionStack = new boolean[26];

for(char c : graph.keySet()) {

if(!visited[c - 'a']) {

if(hasCycle(c, graph, visited, recursionStack, tmp)) {

return "";

}

}

}

StringBuilder sb = new StringBuilder();

for(char c : tmp) {

sb.append(c);

}

//return sb.toString();

return sb.reverse().toString();

}

private boolean hasCycle(char c, Map<Character, Set<Character>> graph, boolean[] visited, boolean[] recursionStack, List<Character> tmp) {

if(recursionStack[c - 'a']) {

return true;

}

if(visited[c - 'a']) {

return false;

}

visited[c - 'a'] = true;

recursionStack[c - 'a'] = true;

for(char neighbour : graph.get(c)) {

if(hasCycle(neighbour, graph, visited, recursionStack, tmp)) {

return true;

}

}

recursionStack[c - 'a'] = false;

tmp.add(0, c);

return false;

}

}

**Refer to**

<https://www.lintcode.com/problem/892/solution/61783>

public class Solution {

/\*\*

\* @param words: a list of words

\* @return: a string which is correct order

\*/

public String alienOrder(String[] words) {

Set[] adj = new Set[26];

Boolean[] visited = new Boolean[26];

Stack<Character> st = new Stack<Character>();

StringBuilder strb = new StringBuilder();

for(int i=0; i<words.length; i++) {

for(int j=0; j<words[i].length(); j++) {

adj[words[i].charAt(j) - 'a'] = new HashSet<>();

}

}

for(int i=0; i<words.length - 1; i++) {

String word1 = words[i];

String word2 = words[i+1];

int m = word1.length();

int n = word2.length();

int minLength = Math.min(m, n);

if (m > n && word1.substring(0, minLength).equals(word2)) {

return "";

}

for(int j=0; j<minLength; j++) {

if (word1.charAt(j) != word2.charAt(j)) {

adj[word2.charAt(j) - 'a'].add(word1.charAt(j));

break;

}

}

}

for(int i=0; i<26; i++) {

char curr = (char) (i + 'a');

if(adj[i] != null && dfs(visited, curr, adj, st)) {

return "";

}

}

while(!st.isEmpty()) {

strb.append(st.pop());

}

return strb.reverse().toString();

}

private boolean dfs(Boolean[] visited, Character ch, Set[] adj, Stack<Character> st) {

if (visited[ch - 'a'] != null) return visited[ch - 'a'];

visited[ch - 'a'] = true;

for(Character nei : (Set<Character>) adj[ch - 'a']) {

if (dfs(visited, nei, adj, st)) {

return true;

}

}

visited[ch - 'a'] = false;

st.push(ch);

return false;

}

}