Topological sort

- Directed Graph
- It's used to test/get ordering of dependent steps
- It's also used for cycle detection in a graph
- Dependencies
- A, B, C, D: Z

```
indegrees : map<vertex, count>
q: queue to store vertices with indegree=0

compute indegree for each node in graph
push nodes with indegree  0 into q

while q is not empty():
    curr = pop from q
    remove from map

    for each neighbour of curr:
        decrease indegree  by 1
        if indegree == 0
            push to queue

        print curr node

if size(indegrees) == 0
  print("possible")
else:
  print("not possible")
```

In [ ]:

```

```

Question
[https://leetcode.com/problems/course-schedule/ (https://leetcode.com/problems/course-schedule/)](https://leetcode.com/problems/course-schedule/)

BFS

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        unordered_map<int, vector<int> > graph;
        unordered_map<int, int> indegrees;
        queue<int> q;

        for (int i =0; i < numCourses; i++) {
            graph[i] = vector<int>{};
            indegrees[i] = 0;
        }

        for (auto edge: prerequisites) {
            // build graph adj repr
            graph[ edge[1] ].push_back(edge[0]);
            indegrees[ edge[0] ] +=1;

        }

        // push node in q with indegree = 0
        for(auto p: indegrees) {
            if (p.second == 0) {
                q.push(p.first);
            }
        }

        while(q.size() > 0) {
            auto curr = q.front();
            q.pop();
            indegrees.erase(curr);

            for (auto n: graph[curr]) {
                indegrees[n] -= 1;
                if (indegrees[n] == 0) {
                    q.push(n);
                }
```

DFS: this logic is for cycle detection, not top sort

```
from collections import defaultdict

class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]])
-> bool:

        graph = defaultdict(list)

        visited = set()
        curr_path = set()

        for p in prerequisites:
            graph[p[1]].append(p[0])

        keys = list(graph.keys())
        for curr in keys:
            if self.cycleDetect(curr, graph, visited, curr_path) == Fals
e:
                return False

        return True


    def cycleDetect(self, curr, graph, visited, path):

        if curr in visited:
            return True
        visited.add(curr)
```

In [ ]:

Alien Dictionary

https://leetcode.com/problems/alien-dictionary/ (https://leetcode.com/problems/alien-dictionary/)

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. Example 1:

Input: ["wrt", "wrf", "er", "ett", "rftt"]

t < f w < e r < t e < r

w <---e w<----r t<---f e<---r r<---t

Output: "wertf"

Example 2:

Input: ["z", "x"]

Output: "zx"

Example 3:

Input: ["z", "x", "z"]

Output: ""

Explanation: The order is invalid, so return "".

Note:

- You may assume all letters are in lowercase.
- You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
- If the order is invalid, return an empty string.
- There may be multiple valid order of letters, return any one of them is fine.

In [ ]:

In [ ]:

In [21]:

```
from collections import defaultdict
import queue

def getOrder(words):

    edges = []
    for i in range(len(words) - 1):
        curr = words[i]
        next = words[i+1]

        for a,b in zip(curr, next):
            if a != b:
                edges.append([a,b])
                break

    graph = defaultdict(list)
    indegree = {}
    for word in words:
        for c in word:
            indegree[c] = 0

    for edge in edges:
        graph[edge[0]].append(edge[1])
        indegree[edge[1]] += 1

    res = []
    q = queue.Queue()
    for k,v in indegree.items():
        if v == 0:
            q.put(k)

    while not  q.empty():
        curr = q.get()
        del indegree[curr]

        for n in graph[curr]:
            indegree[n] -= 1
            if indegree[n] == 0:
                q.put(n)

        res.append(curr)

    if len(indegree) == 0:
        return ''.join(res)

    return ''


print('res=', getOrder([
"wrt",
  "wrf",
  "er",
  "ett",
  "rftt"
]))


print('res=', getOrder([
    "x",
    "z"
]))

print('res=', getOrder([
    "x",
    "z",
    "x"
]))
```

```
res= wertf
res= xz
res= 
```

In [ ]:

```

```

```
    public String AlienDictionary(String [] words){
            int n = words.length;
            Map<Character,List<Character>> graph= new HashMap<>();
            Map<Character,Integer> indegree = new HashMap<>();
            for(int i = 0 ; i < n ; i++){
                String curr = words[i];
                for(int j =0 ; j < curr.length(); j++){
                    char c = curr.charAt(j);
                    if(!graph.containsKey(c)){
                        graph.put(c,new ArrayList<>());
                    }
                    if(!indegree.containsKey(c)){
                        indegree.put(c,0);
                    }
                }
            }

            for(int i =0 ; i  < n ; i++){
                String word1= words[i];

                for(int j = i+1; j < n ; j++){
                    String word2 = words[j];
                    int w1=0 , w2= 0;
                    while(w1 < word1.length() && w2 < word2.length()){
                        if(word1.charAt(w1) == word2.charAt(w2)){
                            w1++;
                            w2++;
                        } else} else {
                            char u = word1.charAt(w1);
                            char v = word2.charAt(w2);
                            graph.computeIfAbsent(u,value->new ArrayList<>
    ()).add(v);
                            indegree.put(v,indegree.getOrDefault(v,0)+1);
                            break;
                        }
                    }
                }
            }
            StringBuilder ans = new StringBuilder();
            Queue<Character> que = new LinkedList<>();
            for(Character key : indegree.keySet()){
                if(indegree.get(key) == 0){
                    que.add(key);
                }
            }

            while(que.size()>0){

                int sz = que.size();
                for(int i =0 ; i < sz ; i++){
```

```
                Character front = que.remove();
                ans.append(front);
                if(graph.containsKey(front)){
                    for(Character adj : graph.get(front)){
                        indegree.put(adj,indegree.get(adj)-1);
                        if(indegree.get(adj) == 0){
                                que.add(adj);
                        }
                    }
                }

            }
        }
        String strAns = ans.toString();
        return strAns.length() == graph.keySet().size() ? strAns : "";

    }
```

In [ ]:

```
topSort(v, stack, visited):
    if v is visited:
        return
    mark v as visited

    for each neighbour n:
        if n is not visited:
            topSort(n, stack, visited)

    stack.push(n)


# main routine
stack:stack()
visited: set()

for each vertex v:
    topSort(v, stack, visited)

result = []
while !stack.empty():
    result.add(stack.pop())
```