**=> Possible Approaches for Graph Problem :**

1. DFS.(Single & multi src using bitmasks).
2. BFS (Single source & Multi Source).
3. Dijkstra & Bellman Ford for shortest route.
4. Minimum spanning tree (Prim & Kruskal)
5. Topological Sort (implement using a DFS variation)
6. Strongly Connected Components by Kosaraju & Tarjan’s algo
7. Travelling nodes from 0 to n-1 for m-coloring problem & not necessarily in bfs or dfs manner.

**=> Theory about BFS & DFS:**

* **FLOW :**

-> DFS first traverses a whole path & when it reaches a leaf ,then it backtracks to 2nd last node,if that has another children , then again visit another path from there,and so on….

-> BFS starts from the root & visits all nodes strictly level by level. Then it visits all node at 2nd level ,then all nodes at 3rd level and so on. **So when any node is reached for the first time, we can say with surety that we reached there through the shortest path possible through the root.**

* **CONCEPT OF VISITED IN BFS & DFS in Undirected Graphs :**

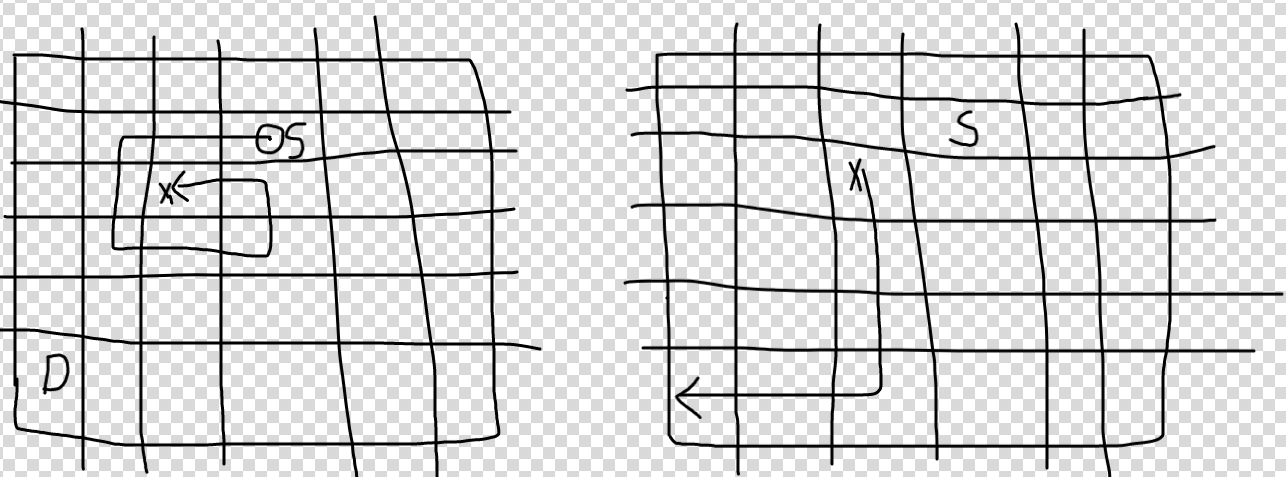
1. **In BFS,when a node is marked as VISITED, this means we reached there through the SHORTEST POSSIBLE PATH.**
2. **In DFS , when we mark a NODE as VISITED,then**

1. **It doesn’t mean we have reached that particular node through the shortest possible path from the root**,so even if we check by reaching the destination from that node through all possible paths,it’s not guaranteed to be the shortest path,bcoz that node itself was not guaranteed to have been reached through the shortest path,now after assuming we traversed all paths from there , its dfs will complete, but we didn’t actually got the shortest path,so when we visit that node again through some shorter path ,we need to call dfs on that again and the process repeats which give TLE.

3. **When it’s DFS of a node ends** ,

**-> All it’s descendants would have been visited.**

**-> But not necessarily ALL PATHS from that node**,means if some adjacent node is visited in some other path , we will see it as visited and not go there,so actually we missed all paths from that node.This can again be solved by samePath[] vector,but as discussed above it gives TLE,as the node itself is not guaranteed to have been reached through the shortest possible path, so need to do many dfs calls from the same node which gives TLE.

**E.g :** imagine dfs for below scenario:

-> In DFS , we maintain a visited array to mark the nodes already visited in a current path, so that the execution doesn’t fall in an INFINITE LOOP.

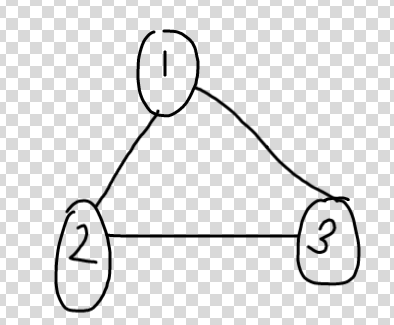
-> But just see the node marked by CROSS in 1st figure , when execution reaches there , all nodes adjacent to it are already visited. So in current scenario it feels that no path is possible from it to the destination, but if you see in the 2nd figure , there is a path.

1. When DFS of a node ends,

(IMP) In Undirected graph , If the node you reach has already been passed through once,then there has to be 2 possibilities :

1. Unique Cycle Detected.
2. That path has already been traversed(means that cycle already detected).

E.g :



1->2->3->1 , 3 detected a cycle as 1 is on the same path , cycle detected, dfs(3) completes,dfs(2) complete, now 1 tries to go to 3,its found visited, doesn’t mean its unique cycle,as this cycle has already been visited by travelling in 1 direction.

-> So a samePath[] array will have to be maintained,and when dfs of any node ‘u’ completes,mark samePath[u] = false.So when 1 try to go to 3 , samePath[3] would be false as its dfs would already have been completed.

1. **DFS WORKS WHEN WE ONLY CARE,WHETHER AFTER MARKING THE NODE AS VISITED,all it’s ADJACENT VERTICES ARE SURELY VISITED OR NOT.**

**E.G :** [**https://leetcode.com/problems/flood-fill/**](https://leetcode.com/problems/flood-fill/)

-> In flood fill, we are given a specific color,lets say color 1, now we only need to ensure that if a cell of this color 1,then all it’s adjacent cells also have to be filled with color 1.So here , we don’t even care about the paths,here when a node is VISITED, it’s execution ends only after all it’s adjacent vertices are visited,and thats what we want.

->So for this problem BFS & DFS both takes O(V+E) time.

**-> But for some problems , where there is only 1 path to destination, DFS will reach early most of the times,as it goes till depth first, while bfs in worst case will traverse all nodes before reaching there.**

**=> DIRECTED GRAPHS :**

1. **Back Edge** : Edge from a Node to another node which is visited earlier(ANCESTOR) in the same strongly connected component.
2. **Cross Edge** : Edge from a node to a node which is visited earlier but in different Strongly Connected Component.

=> **DIRECTED ACYCLIC GRAPHS :**

-> In Directed Acyclic Graph , there are **no Back** Edges.

**-> Difference between Tree and Directed Acyclic Graph.**



1. **Tree is an Undirected graph without cycle + with only n-1 edges , but DAG can have any number of edges.**(means only those edges which are required to keep the graph connected).

**(imp)So there is AT MOST 1 path between any 2 nodes in the tree, but in DAG there can be many paths(e.g 2 paths from 2 to 7 in the tree).**

1. **Tree becomes directed depending on the way we traverse it, but DAG is already directed.**

-> E.g : we traversed it from 1 bcoz of which there is no path between 7 and 8, if we would have started traversing it from 7 or 8 which are the endpoints of the longest path in the tree ,then there would have been 1 path between every node in the tree.

-> Just bcoz DAG can have more than n-1 edges,if you remove the directions,it will definitely have a Cycle.

-> So tree can’t always be a DAG ,as it’s initially an undirected graph, now as shown in the fig.

2-3-5-7-6-4 is cycle for undirected graph on right,and so you DFS it in any manner a cycle will always be detected ,but it can be a DAG, if we give it directions as shown in the fig.

* **(imp)Topological Sorting(indegree concept)on DAG :**

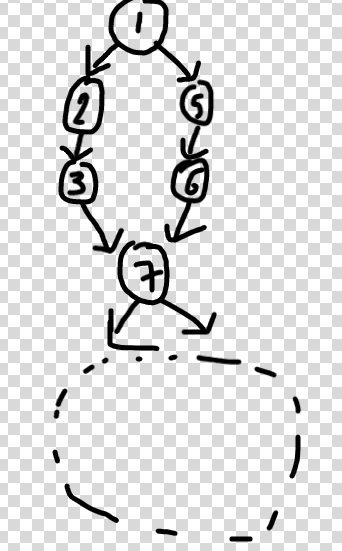
**-> A DAG will have atleast one node with indegree 0 and & atleast one node with outdegree 0.**

**PROCESS :**

1. So, you can start topological sort from a node with indegree 0 and end at node with outdegree 0.
2. After processing node with indegree 0, pass the information to it’s children and decrease the indegree of it’s children, so now there will again be atleast 1 new node with indegree 0 and this process repeats until it encounters the last node with outdegree 0.

**=> Topological Sort processes a node ‘u’ only after all it’s ancestors**(parent(node with incoming edge on ‘u’), parent’s parent(node with incoming edge on parent) and so on till node with indegree 0) **are being processed.**

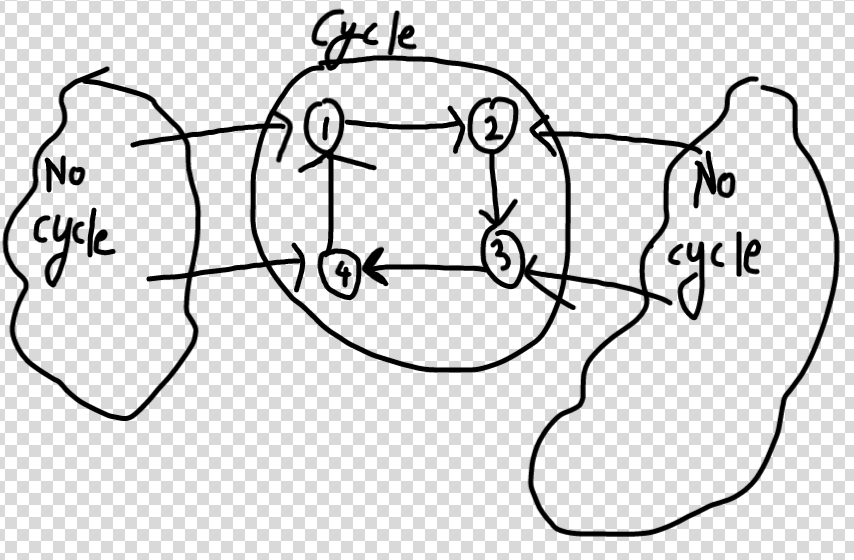
**=> (imp) This fact ensures that when it’s the turn of some node ‘u’ to get processed, ALL THE PATHS ENDING ON NODE ‘u’ would already have been processed.**



In above problem , one possible execution is 1 2 5 3 6 7 .

**-> (imp)So before we reach 7, we would have optimal answers for paths 1->2->3 and 1->5->6 which are incident on 7 , so now node 7 can be processed and this information can be passed to the children as we know that we already found optimal answer amongst all paths ending at 7.**

**=> Cycle Detection using Topological Sort :**

****

As per the topo-sort procedure, all the parts of graph where there will be no cycle will be processed without any problem initially, and also all the edges incident on the part with a cycle will be removed, **but it will be impossible to remove any edge which is a part of cycle** .

Why ? Because all nodes are dependent on each other (deadlock), to make their own indegree 0.

For e.g : 1 wants 4 to get processed, so that the edge incoming on it can be removed, but 4 can’t do it until 3 gets processed, but 3 is dependent on 2 and 2 back on 1, which creates a deadlock.

**=> So during topo sort procedure, make a boolean “visited” array, and if there is a cycle, it would be impossible for any node in the cycle to be marked as visited**(as indegree would never become 0). So if any node has visited = false after the whole procedure ends, then there definitely is a cycle.

Good Problem : <https://leetcode.com/submissions/detail/623122702/>