BREADTH FIRST SEARCH

CONCEPT

- Go breadth first, then go to the second level of depth
- Template

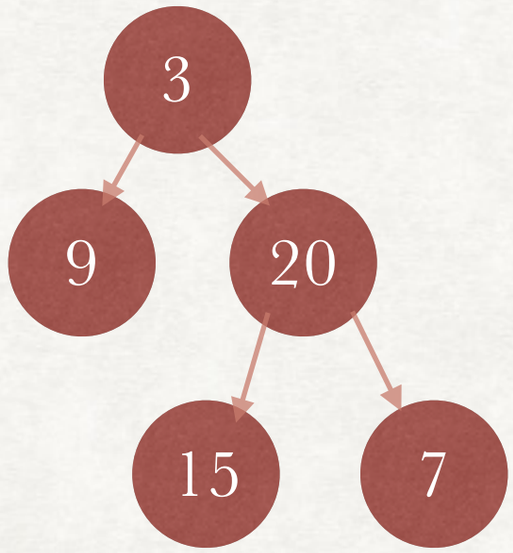
```
while len(queue) > 0:
    node = queue.unshift(0)
    if node.left: queue.shift( node.left)
    if node.right: queue.shift(node.right)
```

NO. 102

BINARY TREE LEVEL ORDER TRAVERSAL

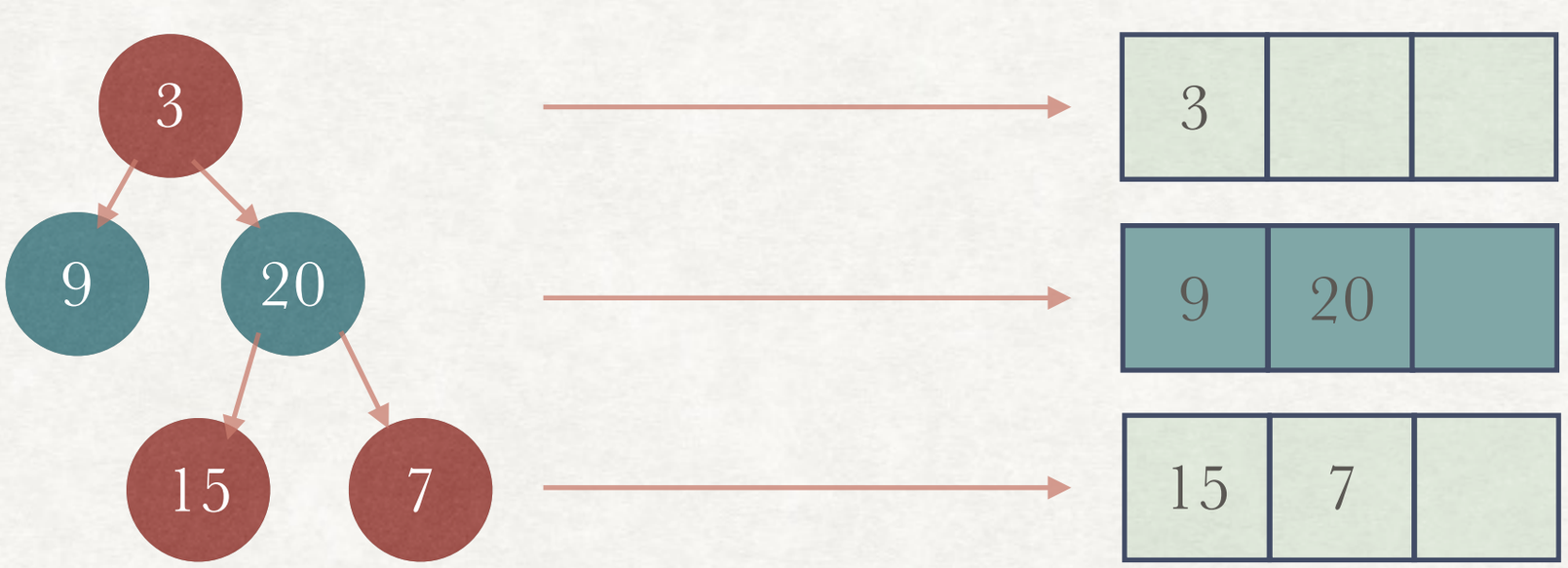
PROBLEM DESCRIPTION

- Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).
- For example: Given binary tree [3,9,20,null,null,15,7]
- Return its level order traversal as: [[3], [9,20], [15,7]]



IDEA

- Nodes which located in the same level need to be contained in the same list, thus we need to use TWO queues to separate nodes from different level of depth



SOLUTION

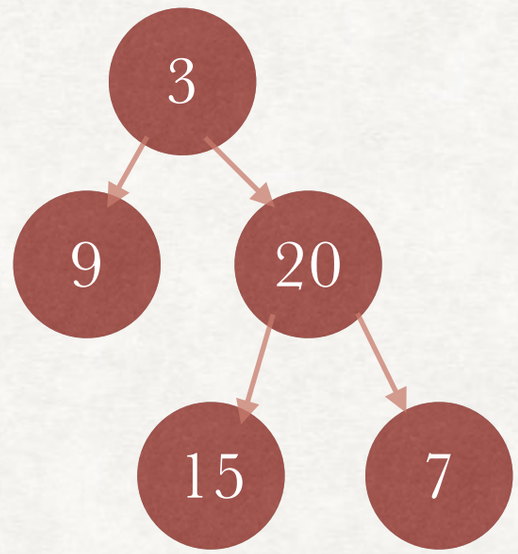
- https://github.com/Brady31027/leetcode/tree/master/102_Binary_Tree_Level_Order_Traversal

NO. 107

BINARY TREE LEVEL ORDER TRAVERSAL II

PROBLEM DESCRIPTION

- Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root). return its bottom-up level order traversal
- For example: Given binary tree [3,9,20,null,null,15,7], return [[15,7], [9,20], [3]]



IDEA

- Similar to Binary Tree Level Order Traversal except
  - local ans has to be added to the front of the ans list
    - ans = [local_ans] + ans

SOLUTION

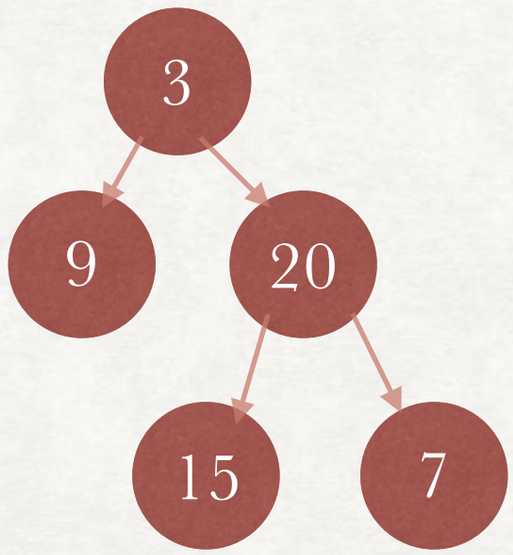
- https://github.com/Brady31027/leetcode/tree/master/107_Binary_Tree_Level_Order_Traversal_II

NO. 103

BINARY TREE ZIGZAG LEVEL ORDER TRAVERSAL

PROBLEM DESCRIPTION

- Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).
- For example: Given binary tree [3,9,20,null,null,15,7], return its zigzag level order traversal as [[3], [20,9], [15,7]]



IDEA

- Similar to Binary Tree Level Order Traversal except
  - Maintain zig-zag order using a boolean left_to_right
    - if left_to_right: local_ans = local_ans + [node.val]
    - if not left_to_right: local_ans = [node.val] + local_ans
    - left_to_right = not left_to_right

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/103_Binary_Tree_Zigzag_Level_Order_Traversal

NO. 127

WORD LADDER

PROBLEM DESCRIPTION

- Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that:

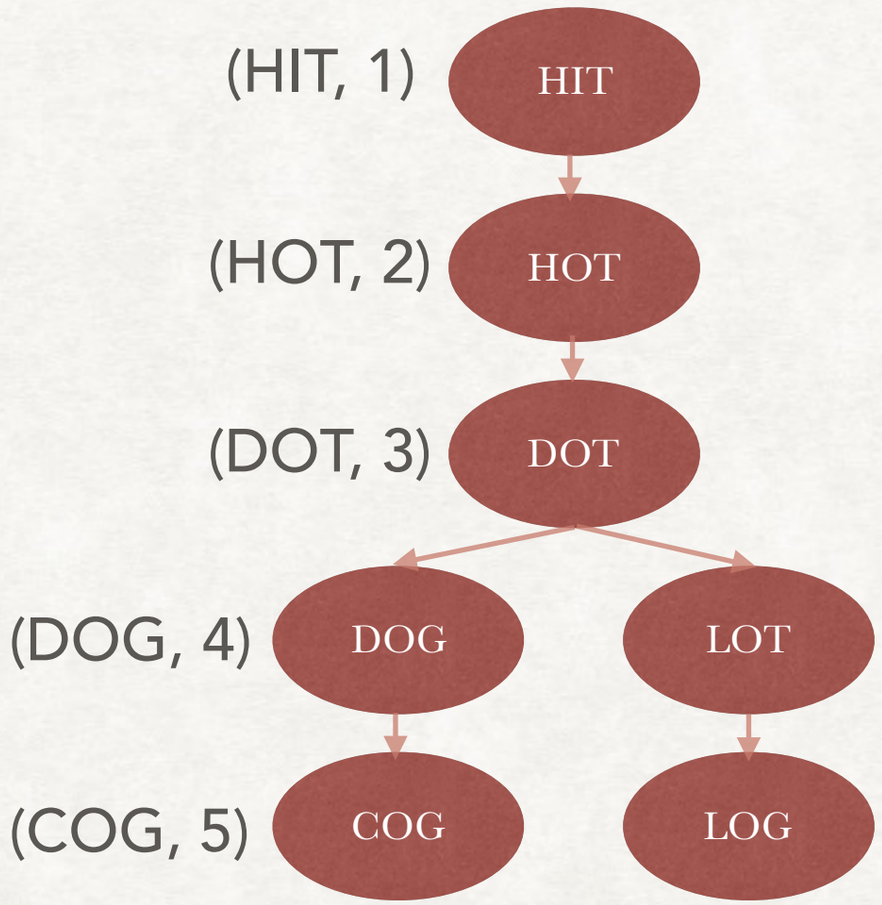
- Only one letter can be changed at a time.

- Each transformed word must exist in the word list. Note that beginWord is not a transformed word.

- For example, given: beginWord = “hit”, endWord = “cog”, wordList = [“hot”,"dot","dog","lot","log","cog"]. As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> “cog", return its length 5

IDEA

- Create a tree to maintain the words, use tuple to represent a node




```
for i in range(word):
    for char in [a-z]:
        word = item[:i] + char +item[i+1:]
        if word in wordList:
            wordList.remove(word)
            queue.append( (word, depth+1) )
```

Basic BST will be TIME OUT!!

IDEA

- Bi-directional BST, one from top, the other from bottom
  - Topology
    - reduce the degree of breath, less options, mode efficient

```
if len(top_queue) > len(bottom_queue):
    top_queue, bottom_queue = bottom_queue, top_queue
```

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/127_Word_Ladder

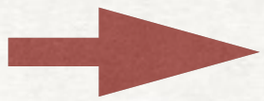
NO. 130

SURROUNDED REGIONS

PROBLEM DESCRIPTION

- Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by ‘X'. A region is captured by flipping all 'O's into 'X's in that surrounded region. — Connected component !

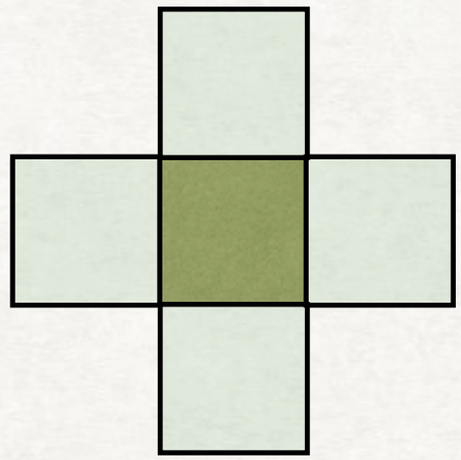
X	X	X	X
X	O	O	X
X	X	O	X
X	O	X	X



X	X	X	X
X	X	X	X
X	X	X	X
X	O	X	X

IDEA

- Take 4 directions into account
- Algorithm
  - Find entries



- For every entry, start flooding (BFS)

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/130_Surrounded_Regions

NO. 133

CLONE GRAPH

PROBLEM DESCRIPTION

- Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.
  - Deep copy !

IDEA

- BFS template

```
def cloneGraph(self, node):
    queue= [node]
    while len(queue):
        current = queue.pop()
        for neighbor in current.neighbors:
            queue.append(neighbor)
```

maintain a dict for cloned graph

if neighbor not in dict, update dict

update neighbor list for the cloned node

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/133_Clone_Graph

NO. 207

COURSE SCHEDULE

PROBLEM DESCRIPTION

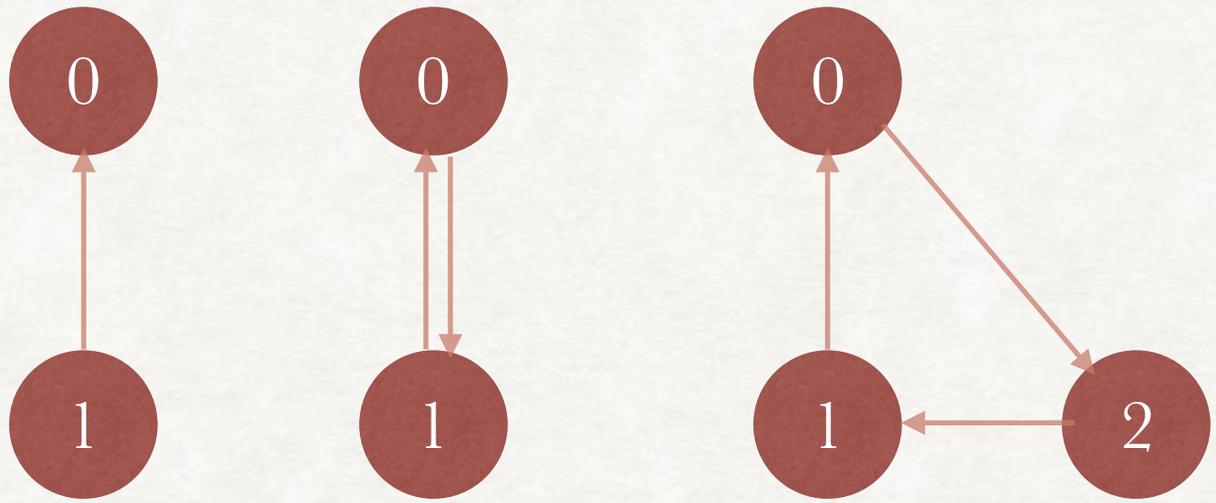
- There are a total of n courses you have to take, labeled from 0 to n - 1. Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]. Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

- For example: Given 2, [[1,0]]. There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

- Example 2, Given 2, [[1,0],[0,1]]. There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

IDEA

- Determine there is a loop inside the graph
  - E.g. 1 [[1, 0]]
  - E.g. 2 [[1,0], [0,1]]
  - E.g. 3 [[1,0], [0,2], [2,1]]



- How to determine?
  - Maintain a boolean array for every node
    - Init all nodes to TRUE
    - Set all children to FALSE
    - Remove nodes which is set to True (indep. nodes)
    - If we cannot remove any node but list is not empty, there is must a loop !

Performance BAD!

IDEA

- How to determine? -- Method 2
  - degree list represents the number of parents for a child node
  - children list represents the child nodes for a parent node
- Build the graph accordingly

```
for c, p in prerequisites:
        degree[c] += 1
        children[p].append(c)
```

- Traverse every node
  - if degree is 0, then it's an indep. node. Remove it!
  - if degree is 0, then decrease the degrees for every child nodes

SOLUTION

- https://github.com/Brady31027/leetcode/blob/master/207_Course_Schedule/course_shedule.py

NO. 210

COURSE SCHEDULE II

PROBLEM DESCRIPTION

- There are a total of n courses you have to take, labeled from 0 to n - 1. Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]. Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses. There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

- For example: 2, [[1,0]]. There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

- Another example, 4, [[1,0],[2,0],[3,1],[3,2]]. There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

IDEA

- Similar to Course Schedule except
  - Maintain a list to record the removed courses

```
for node in removed_nodes:
        queue_ans.append(node)
        unvisited_course.remove(node)
```

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/210_Course_Schedule_II

NO. 310

MINIMUM HEIGHT TREES

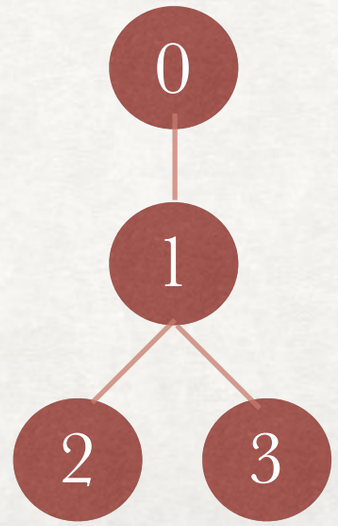
PROBLEM DESCRIPTION

- For a undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

- Format: The graph contains n nodes which are labeled from 0 to n - 1. You will be given the number n and a list of undirected edges (each edge is a pair of labels). You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

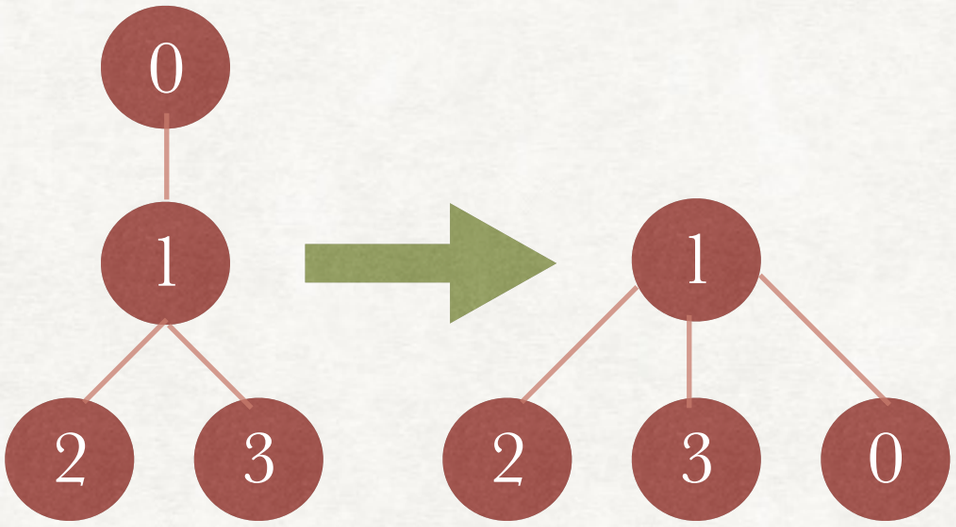
- Example 1:

- Given n = 4, edges = [[1, 0], [1, 2], [1, 3]], return [1]



IDEA

- Topology



1. calculate the degree for every node
2. remove nodes which have only 1 degree
3. remove edges which contain these nodes
4. go to step 1 until there are only 1 or 2 nodes

Node	0	1	2	3
Degree	1	3	1	1

TIME LIMIT EXCEEDED!!

IDEA

- Bottom-up BFS
  - Find leaf (degree is 1)
  - Remove leaf and the edges containing this leaf
    - If neighbor node's degree becomes to 1, add to queue
      - Next level pruning

MHT is at most 2, because the root we want must be located in the center of the tree

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/310_Minimum_Height_Trees