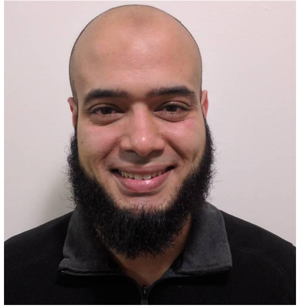
Data Structures

Level Order Traversal 2

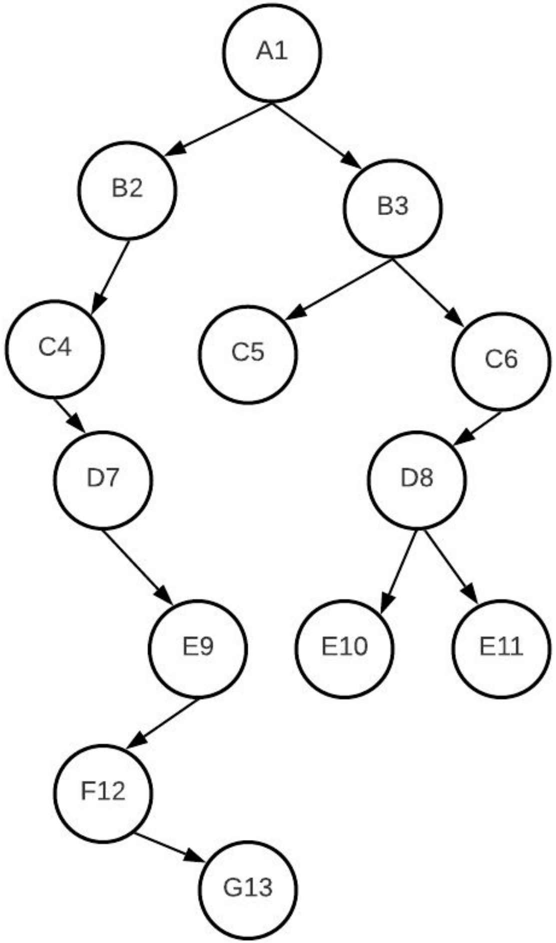
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Let’s check the queue

- A1 : remove A1, add B2, B3
- B2, B3 : remove B2, add C4
- B3, C4 : remove B3, add C5, C6
- **C4, C5, C6** : remove C4, add D7
- **C5, C6, D7** : remove C5, add nothing
- C6, D7 : remove C6, add D8
- D7, D8 : remove D7, add E9
- D8, E9 : remove D8, add E10, E11
- E9, E10, E11 : remove E9, add F12
- E10, E11, F12
- F12
- G13



Queue Implementation

- We can use our Queue implementations for efficiency
    - But better we use more built-in staff
- What about list?
    - lst.pop(0) is O(n) NOT O(1)
    - Remember, list internally is an array.
    - Removing the first element results in shifting left the whole array
- queue = collections.deque()
    - A better option is to use the built-in deque, which is **constant time pops** at both ends
    - queue.popleft(): is like list.pop(0)
    - queue.pop(): pop from the right side

Implementation v1

- Just simulate the process using the code
- Although we're printing level by level, we don't know the exact level of each node!
- 2 ways
    - Add the level into the queue
    - Or smartly, process level by level

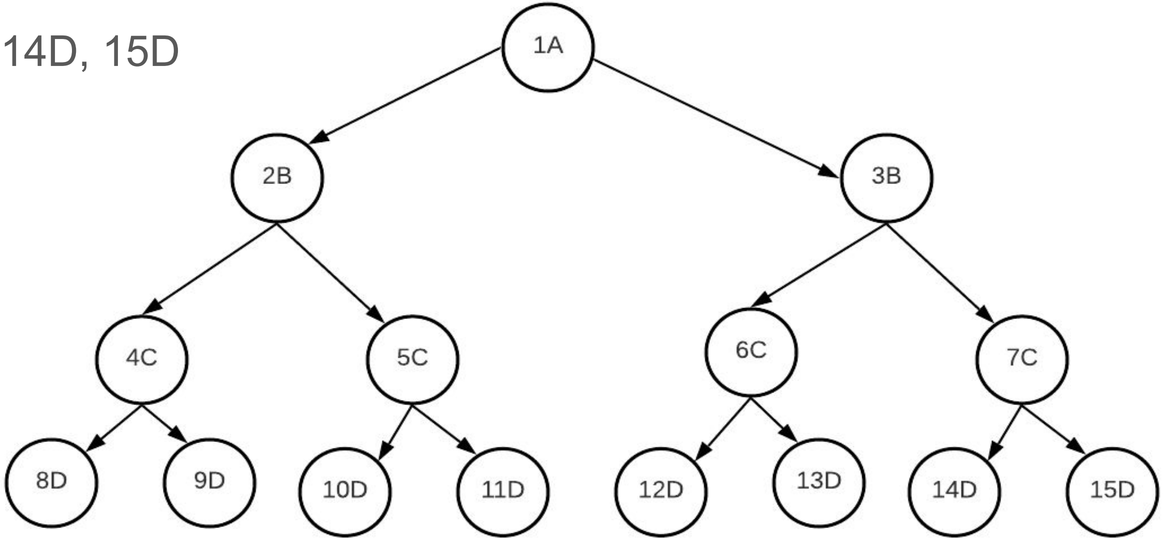
```python
def level_order_traversal1(self):
    import collections
    nodes_queue = collections.deque()
    nodes_queue.append(self.root)

    while nodes_queue:
        cur = nodes_queue.popleft()

        print(cur.val, end=' ')

        if cur.left:
            nodes_queue.append(cur.left)
        if cur.right:
            nodes_queue.append(cur.right)
    print("")
```

Print level by level, knowing level

- Let's assume that the queue right now ONLY contains nodes from level 5
  - Assume there are 4 nodes.
  - Let's call the number of nodes sz
- While the queue is not empty, process the nodes 'sz' number of times
  - Now the sz (4) nodes are removed!
  - Only their children are added

Process based on current size

- 1A   sz = 1, Process 1 step
- 2B, 3B   sz = 2, Process 2 steps
- 4C, 5C, 6C, 7C   sz = 4
- 8D, 9D, 10D, 11D, 12D, 13D, 14D, 15D



Implementation v2

- We can now trivially work out which level we are at
- In each step:
    - We process all current parents
    - Add all their children
    - Hence, the queue will always contain nodes from one level
- Both methods are O(n) time
    - We iterate on each node: ~n
    - We move through each edge: ~n
        - A tree has n-1 edges

```python
def level_order_traversal2(self):
    import collections
    nodes_queue = collections.deque()
    nodes_queue.append(self.root)
    level = 0

    while nodes_queue:
        print(f'\nLevel {level}: ', end='')
        sz = len(nodes_queue)
        for step in range(sz):
            cur = nodes_queue.popleft()

            print(cur.val, end=' ')

            if cur.left:
                nodes_queue.append(cur.left)
            if cur.right:
                nodes_queue.append(cur.right)
        level += 1
```

```
Level 0: 1
Level 1: 2 3
Level 2: 4 5 6 7
Level 3: 8 9 10 11 12 13 14 15
```

Time Complexity

- Fact: A tree of n nodes has always n-1 edges (think about it)
- Time complexity
  - In both recursive and level traversals: we iterate on each node ⇒ ~n steps
  - From each node, we pass on its children. **Total** edges ~n
    - Don't just say/assume it will be a constant maximum of 2! Think about the total here
  - So, it's O(n) time in total

Memory Complexity

- In recursion, we have a **stack** of depth h. So O(h)
- But for level order? We have a queue of items
- We know the queue will never have more than n nodes, so O(n)
  - Actually, we will only have a subset of them: the max level per tree
- So, in a perfect tree, we have a max of 2^h nodes in the last level, which is O(2^h)
  - However, if the tree is degenerate, this means we have n nodes, but O(1) complexity
- Overall, this should encourage the following choices:
  - The best case: O(1) for degenerate tree
  - The worst case: for a perfect tree, we have O(2^h).. As h = ~log n. It's again O(n)
    - Math Tip: 2 ^ logn = n
  - **Overall: a better representation is O(n) memory complexity**

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."