Module 2 Creativity Exercises

2.5.11)

We can find the middle node on a DLL (or just SLL) by using two pointers, one a slow pointer that jumps one node on each iteration and one a fast pointer that will jump at twice the speed (aka two nodes) at each iteration. Once the fast pointers next node is null or fast is pointing to null (since we are using sentinel nodes) we can return the slow pointer which should be at the middle node of the list.

```
#start code
// will assume that the header sentinel node is assigned to a var called root
// also will assume that the node objects have an attribute called next that links to the next node in the list

slow = fast = root

while fast != null or fast.next != null:
    slow ← slow.next
    fast ← fast.next.next

return slow
```

2.5.20)

We will use a BFS approach and within the queue we will store both the node object and its level. For each iteration we will check for any children and add them to the queue along with the current node level incremented by 1. This way we can keep track of everything in the queue and then just mimic the adds to our answer

```
#start code
// assuming our node objects consist of a children attribute which is an array of the child node objects
// also will assume that we have at least a root node var assigned already

curr_level ← 0
queue ← [(root, curr_level)] // array of tuples
ans ← [(root, curr_level)] // will mimic our queue but will retain values

while queue:
    curr_node ← queue.dequeue()
    node_obj ← curr_node[0]
    curr_node_level ← curr_node[1]

    if node_obj.children:
        for child_node in node_obj.children:
```

```
                    ans.append((child_node, curr_node_level + 1))
                    queue.enqueue((child_node, curr_node_level + 1))
return ans
```