Whiteboard Problems 6-24-20 W4D3

For both of these problems, assume there is a Node class. The node class will take in a value as part of its initialization. You will monkeypatching the following methods:

# 1. Write a method `bfs` that does a breadth-first search starting at a root node. It takes in a target, and a proc as an argument.

```
Def bfs(target, &prc)
    Prc ||= { |x, y| x ⇔ y }
    Queue = [self.root_node]
    Until queue.empty?
        Front = queue.pop
        If prc.call(front.value) == 0
            Return front
        End
        Queue.unshift(self.root_node.children)


    End
    nil
end
```

# 2. Write a method `dfs` that does a depth-first search starting at a root node. It takes in a target, and a proc as an argument.

```
Class Node
    Def dfs(target, &prc)
        Return self if self.value == target
        self.children.each do |node|
            Result = node.dfs(target, &prc)
            Return result if result
        End
        nil
    end
```

Whiteboard Problems 6-24-20 W4D3

```
end

# Example usage:

# n1 = Node.new(1) # making a node with a value of 1

# n1.bfs(1) #=> n1

# n1.dfs { |node| node.value == 1 } #=> n1 (found a node with value == 1)

class Node

 # -- Assume nodes have a value, and a attr_reader on value

 # -- Also, assume there are working parent/child-related methods for Node

def bfs(&prc)

   raise "Must give a proc or target" if prc.nil?

   queue = [self]

   until queue.empty?

     visited = queue.shift

     return visited if prc.call(visited)

     queue += visited.children

   end

   nil

 end

 def dfs(, &prc)

   raise "Must give a proc or target" if prc.nil?

   return self if prc.call(self)

   self.children.each do |node|

     result = node.dfs(target, &prc)

     return result if result

   end

   nil

 end

end
```