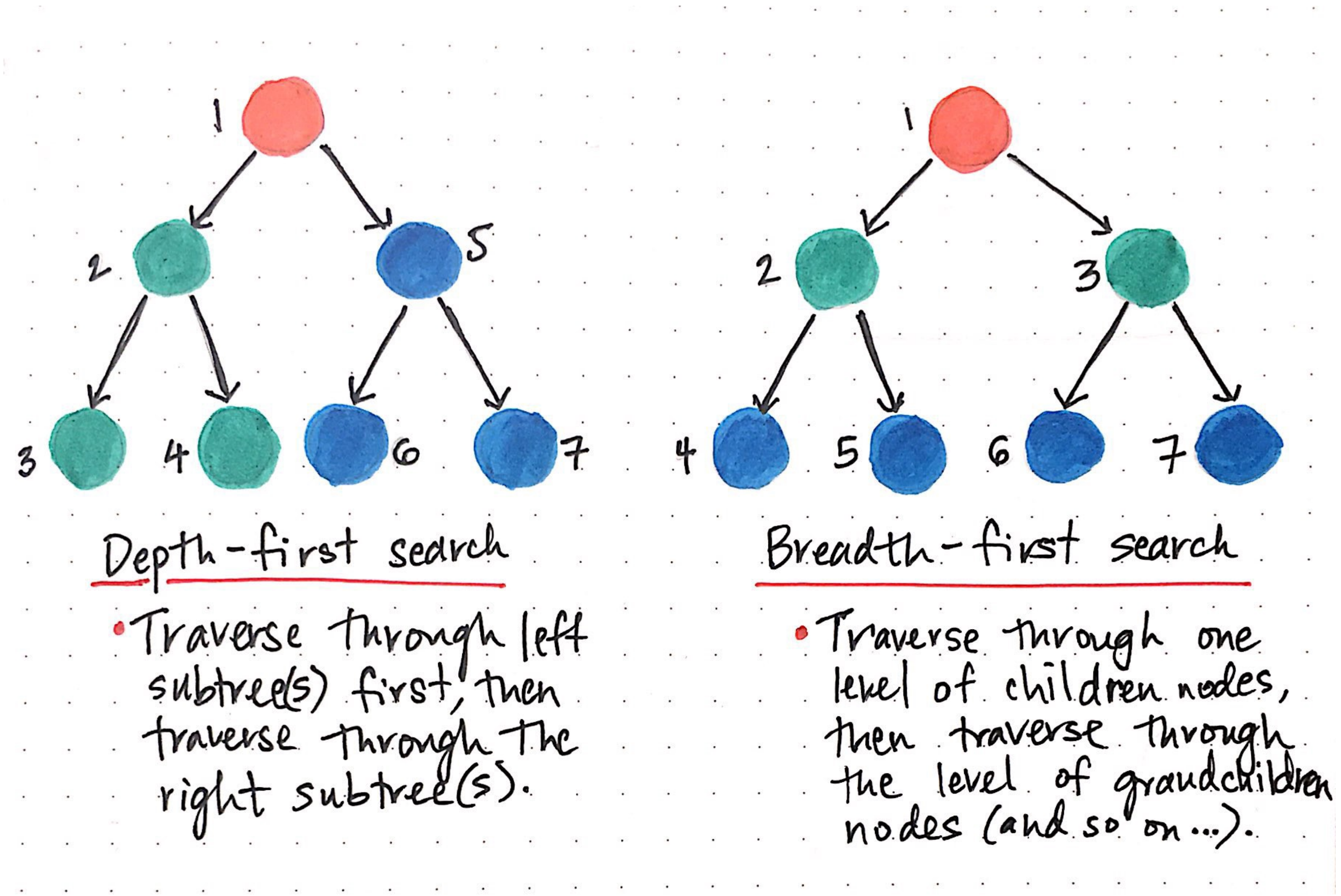

Depth-first search

- Traverse through left subtree(s) first, then traverse through the right subtree(s).

Breadth-first search

- Traverse through one level of children nodes, then traverse through the level of grandchildren nodes (and so on...).

```
#dfs
from collections import defaultdict
class Graph:

  def __init__(self): #constructor
    self.graph = defaultdict(list)
    print("instructor called")
  def addEdge(self,u,v):
    self.graph[u].append(v) 0:[1,2],1:[2],2:[0,3],3:[3]} #{
    print(self.graph)
  def DFS(self,s):
    visited = set()
    self.DFSlop(s,visited)

  def DFSlop(self,s,visited):
    visited.add(s)

    print(s)
    for edge in self.graph[s]:
      if edge not in visited: # !=
        print("edges",edge)
        print("visited",visited)
        self.DFSlop(edge,visited)
```

```
g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)
g.DFS(2)
```

```
instructor called
defaultdict(<class 'list'>, {0: [1]})
defaultdict(<class 'list'>, {0: [1, 2]})
defaultdict(<class 'list'>, {0: [1, 2], 1: [2]})
defaultdict(<class 'list'>, {0: [1, 2], 1: [2], 2: [0]})
defaultdict(<class 'list'>, {0: [1, 2], 1: [2], 2: [0, 3]})
defaultdict(<class 'list'>, {0: [1, 2], 1: [2], 2: [0, 3], 3: [3]})
2
edges 0
visited {2}
0
edges 1
visited {0, 2}
1
edges 3
visited {0, 1, 2}
3
```

Task

Q1 Design an algorithm to Implement BFS

```
# code here
```

● ✕