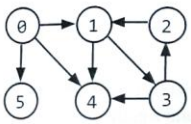
```
class Node{
   public String name;
   public List<Node> children;
   public Node(String name) {
      this.name = name;
      this.children = new ArrayList<>();
   }
}

class Graph{
   public List<Node> nodes;
   public Graph() {
      this.nodes = new ArrayList<>();
   }

   public void addNode(Node node) {
      nodes.add(node);
   }

   public void addEdge(Node from, Node to) {
      from.children.add(to);
   }
}
```

Graph



Depth-First Search

```
1   Node 0
2    Node 1
3     Node 3
4      Node 2
5      Node 4
6   Node 5
```

Breadth-First Search

```
1   Node 0
2   Node 1
3   Node 4
4   Node 5
5   Node 3
6   Node 2
```

Breadth-first search and depth-first search tend to be used in different scenarios. DFS is often preferred if we want to visit every node in the graph. Both will work just fine, but depth-first search is a bit simpler.

```
1   void search(Node root) {
2      if (root == null) return;
3      visit(root);
4      root.visited = true;
5      for each (Node n in root.adjacent) {
6         if (n.visited == false) {
7            search(n);
8         }
9      }
10  }
```

```
1   void search(Node root) {
2      Queue queue = new Queue();
3      root.marked = true;
4      queue.enqueue(root); // Add to the end of queue
5
6      while (!queue.isEmpty()) {
7         Node r = queue.dequeue(); // Remove from the front of the queue
8         visit(r);
9         foreach (Node n in r.adjacent) {
10           if (n.marked == false) {
11              n.marked = true;
12              queue.enqueue(n);
13           }
14        }
15     }
16  }
```