Problem: Find the shortest path between two nodes in a graph.

Constraints

- **Is the graph directed?**
    - **Yes**
- **Is the graph weighted?**
    - **No**
- **Can we assume we already have Graph and Node classes?**
    - **Yes**
- **Are the inputs two Nodes?**
    - **Yes**
- **Is the output a list of Node keys that make up the shortest path?**
    - **Yes**
- **If there is no path, should we return None?**
    - **Yes**
- **Can we assume this is a connected graph?**
    - **Yes**
- **Can we assume the inputs are valid?**
    - **Yes**
- **Can we assume this fits memory?**
    - **Yes**

Test Cases

Input:

- **add_edge(source, destination, weight)**
- **graph.add_edge(0, 1)**
- **graph.add_edge(0, 4)**
- **graph.add_edge(0, 5)**
- **graph.add_edge(1, 3)**
- **graph.add_edge(1, 4)**
- **graph.add_edge(2, 1)**
- **graph.add_edge(3, 2)**

graph.add_edge(3, 4)

Result:

- **search_path(start=0, end=2) -> [0, 1, 3, 2]**
- **search_path(start=0, end=0) -> [0]**
- **search_path(start=4, end=5) -> None**

Algorithm

To determine the shorted path in an unweighted graph, we can use breadth-first search keeping track of the previous nodes ids for each node. Previous nodes ids can be a dictionary of key: current node id and value: previous node id.

- **If the start node is the end node, return True**
- **Add the start node to the queue and mark it as visited**
    - **Update the previous node ids, the previous node id of the start node is None**
- **While the queue is not empty**
    - **Dequeue a node and visit it**
    - **If the node is the end node, return the previous nodes**
    - **Set the previous node to the current node**
    - **Iterate through each adjacent node**
        - **If the node has not been visited, add it to the queue and mark it as visited**
            - **Update the previous node ids**
- **Return None**

Walk the previous node ids backwards to get the path.

Complexity:

- **Time: O(V + E), where V = number of vertices and E = number of edges**
- **Space: O(V + E)**