In [ ]:

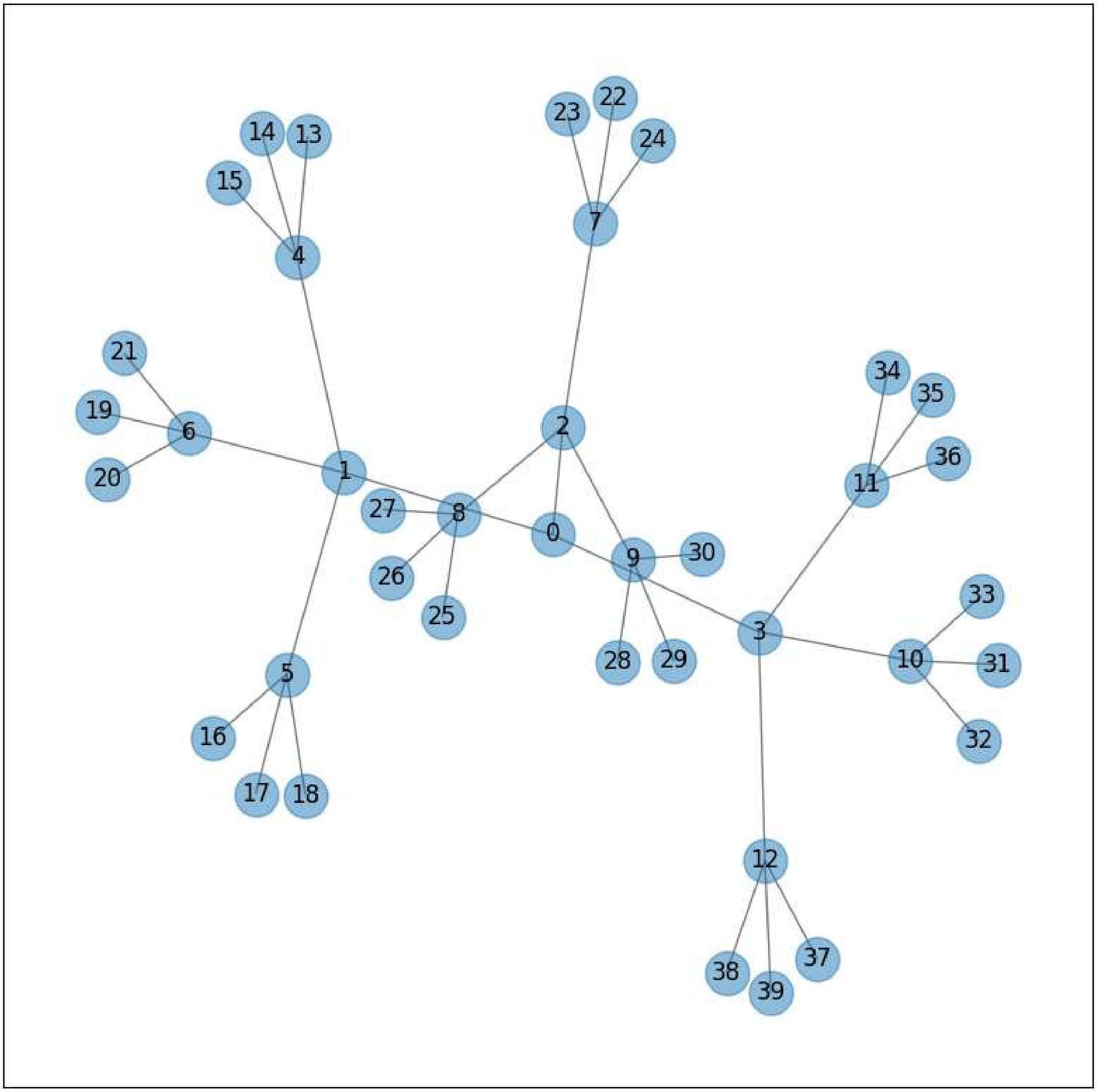
```
import networkx as nx
from matplotlib import pyplot as plt
plt.rcParams["figure.figsize"] = (10,10)
```

In [ ]:

```
G = nx.balanced_tree (3,3)
```

In [ ]:

```
def draw_graph(G):
    pos = nx.spring_layout(G)
    nx.draw_networkx_nodes(G, pos, node_size=500, alpha=0.5)
    nx.draw_networkx_labels(G, pos)
    nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)
draw_graph(G)
```



In [ ]:

```
G.nodes
```

Out[ ]:

```
NodeView((0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39))
```

In [ ]:

```
G.edges
```

Out[ ]:

```
EdgeView([(0, 1), (0, 2), (0, 3), (1, 4), (1, 5), (1, 6), (2, 7), (2, 8), (2, 9), (3,
10), (3, 11), (3, 12), (4, 13), (4, 14), (4, 15), (5, 16), (5, 17), (5, 18), (6, 19),
(6, 20), (6, 21), (7, 22), (7, 23), (7, 24), (8, 25), (8, 26), (8, 27), (9, 28), (9,
29), (9, 30), (10, 31), (10, 32), (10, 33), (11, 34), (11, 35), (11, 36), (12, 37),
(12, 38), (12, 39)])
```

In [ ]:

```
def bfs(graph, starting_node):
    visited = []
    queue = [starting_node]

    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            for edge in graph.edges:
                if edge[0] == node:
                    queue.append(edge[1])
                elif edge[1] == node:
                    queue.append(edge[0])
    return visited
```

In [ ]:

```
bfs(G, 1)
```

Out[ ]:

```
[1,
 0,
 4,
 5,
 6,
 2,
 3,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 7,
 8,
 9,
 10,
 11,
 12,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
 36,
 37,
 38,
 39]
```

In [ ]:

```python
def find_shortest_path(graph, starting_node, goal):
    visited = []
    queue = [[starting_node]]

    while queue:
        path = queue.pop(0)
        node = path[-1]
        if node not in visited:
            neighbours = []
            for edge in graph.edges:
                if edge[0] == node:
                    neighbours.append(edge[1])
                elif edge[1] == node:
                    neighbours.append(edge[0])
            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)

                if neighbour == goal:
                    return new_path

            visited.append(node)

    return []
```

In [ ]:

```
find_shortest_path(G, 1, 15)
```

Out[ ]:

```
[1, 4, 15]
```