Algorithm 1 Path between nodes in a full BT

```
1: procedure MAIN()
2:     N ← Input the graph
3:     levelorder[ ] ← level order traversal array of size N
4:     Input the Levelorder for the tree
5:     Root = insertLevelOrder(levelOrder, root, 0, n)
6:     printPath(root, x, y) // path between x and y to be found out
7: end procedure

8: function INSERTLEVELORDER(arr, root, i, n)
9:     if i < N then
10:        temp = NewNode(arr[i])
11:        root = temp
12:        root− >left = insertLevelOrder(arr, root− >left, 2 * i + 1, n)
13:        root− >right = insertLevelOrder(arr, root− >right, 2 * i + 2, n)
14:    end if
15:    return root
16: end function

17: function FIND PATH(root, path, k)
18:    if root == NULL then
19:        return False
20:    end if
21:    path.pushback(root− >key)
22:    if root− >key == k then
23:        return True
24:    end if
25:    if (root− >left and findPath(root− >left, path, k)) or (root− >right
   and findPath(root− >right, path, k)) then
26:        return True
27:    end if
28:    path.pop_back()
29:    returnfalse
30: end function

31: function PRINTPATH(root, n1, n2)
32:    path1[] // Initialize
33:    path2[] // Initialize
34:    if (!findPath(root, path1, n1) or !findPath(root, path2, n2)) then
35:        Path Does Not Exist
36:    end if
37:    for i ← 0 to (i < path1.size() and i < path2.size()) do
38:        if path1[i] != path2[i] then
39:            break
40:        end if
41:    end for
42:    for j ← path1.size() − 1 to j >= i−1  do
43:        print path1[j]
44:    end for
45:    for j ← i to j < path2.size() do
46:        print path2[j]
47:    end for
48: end function
```