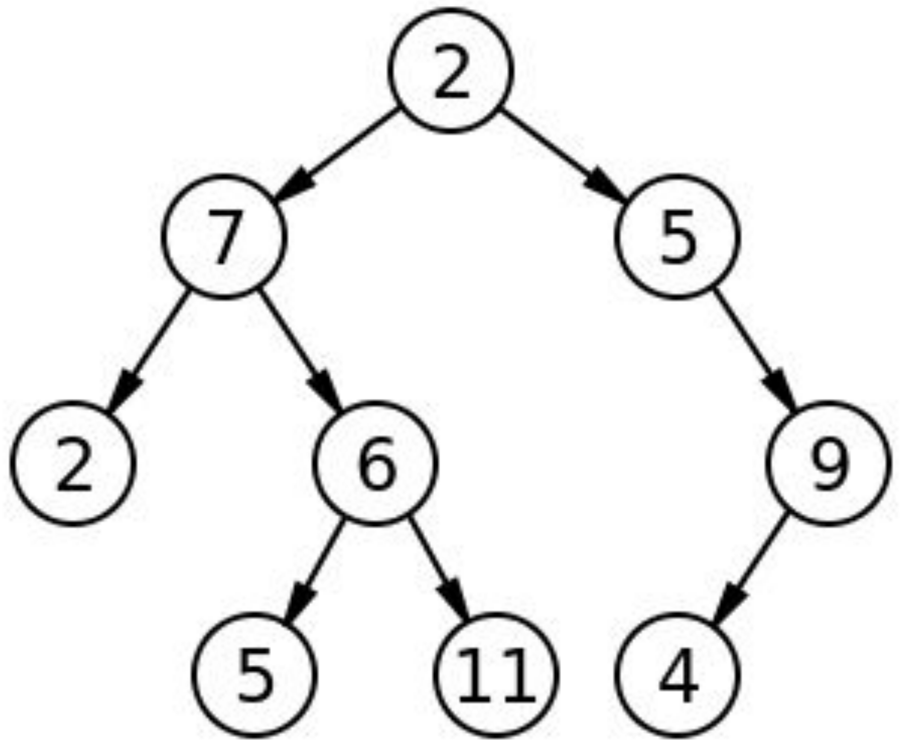
Problem Domain
Write a function called breadth first which takes in an arguments of a tree. This function should return a list of all values in the tree, in the order they were encountered.

NOTE: Traverse the input tree using a Breadth-first approach

Input

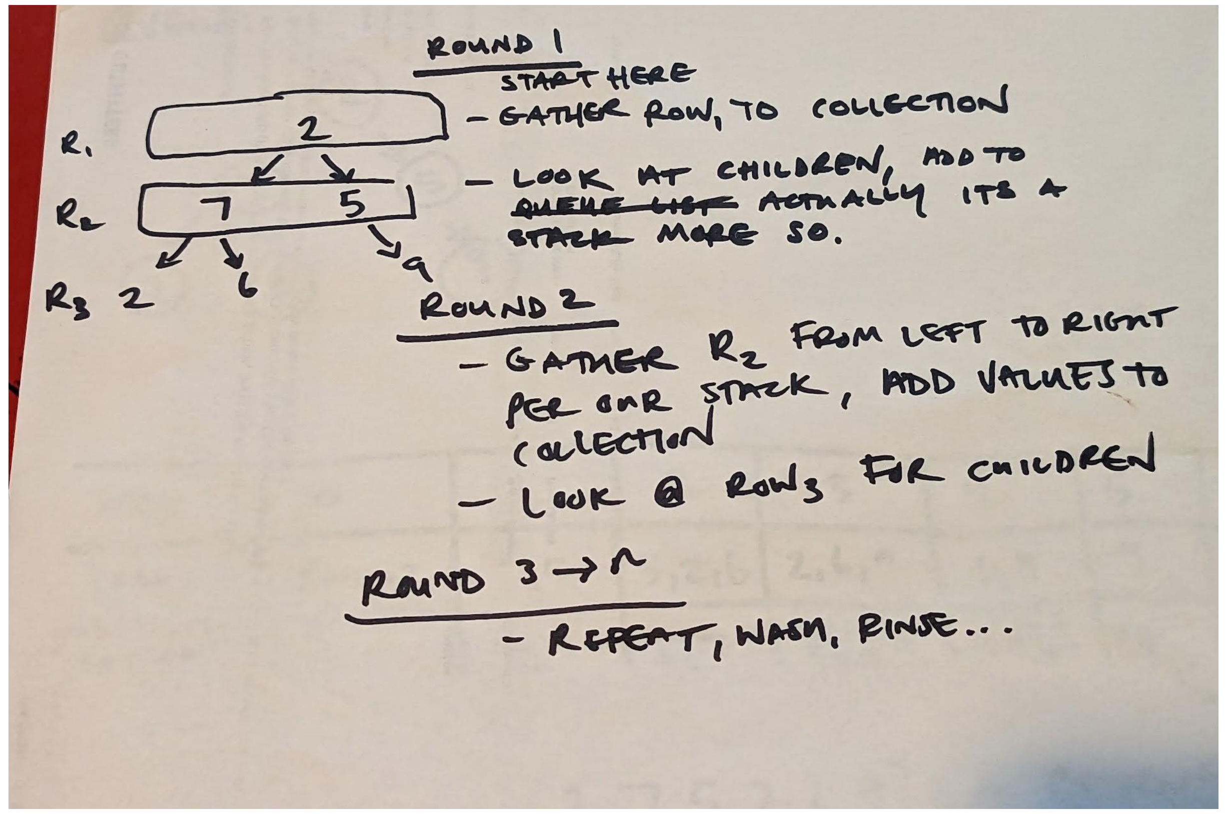


Output

```
[2,7,5,2,6,9,5,11,4]
```

BigO
- Time = O(n) where n is the number of nodes in our graph
- Space = O(n) where n is the number of nodes in our graph

Visualization


R1
2
R2
7
5
R3
2
6
9

ROUND 1
START HERE
- GATHER ROW, TO COLLECTION
- LOOK AT CHILDREN, ADD TO ~~QUEUE LIST~~ ACTUALLY ITS A ~~STACK~~ MORE SO.

ROUND 2
- GATHER R_2 FROM LEFT TO RIGHT PER OUR STACK, ADD VALUES TO COLLECTION
- LOOK @ ROW_3 FOR CHILDREN

ROUND 3 → n
- REPEAT, WASH, RINSE...

Test Cases

happy case:
given a tree, T (see above) return [2,7,5,2,6,9,5,11,4]

no max value/no tree:
return []

if T doesn't exist

Algorithm(keeping track of the largest)
- start at head, check if it exist
- if so record value, it is our max, go left and go through all nodes/leaves comparing values
- after completing left side, go to the right side and do the same looking for the biggest.
- compare the values of the max to each side
- keep the largest, then compare these numbers and keep the max
- return max value

Code Block

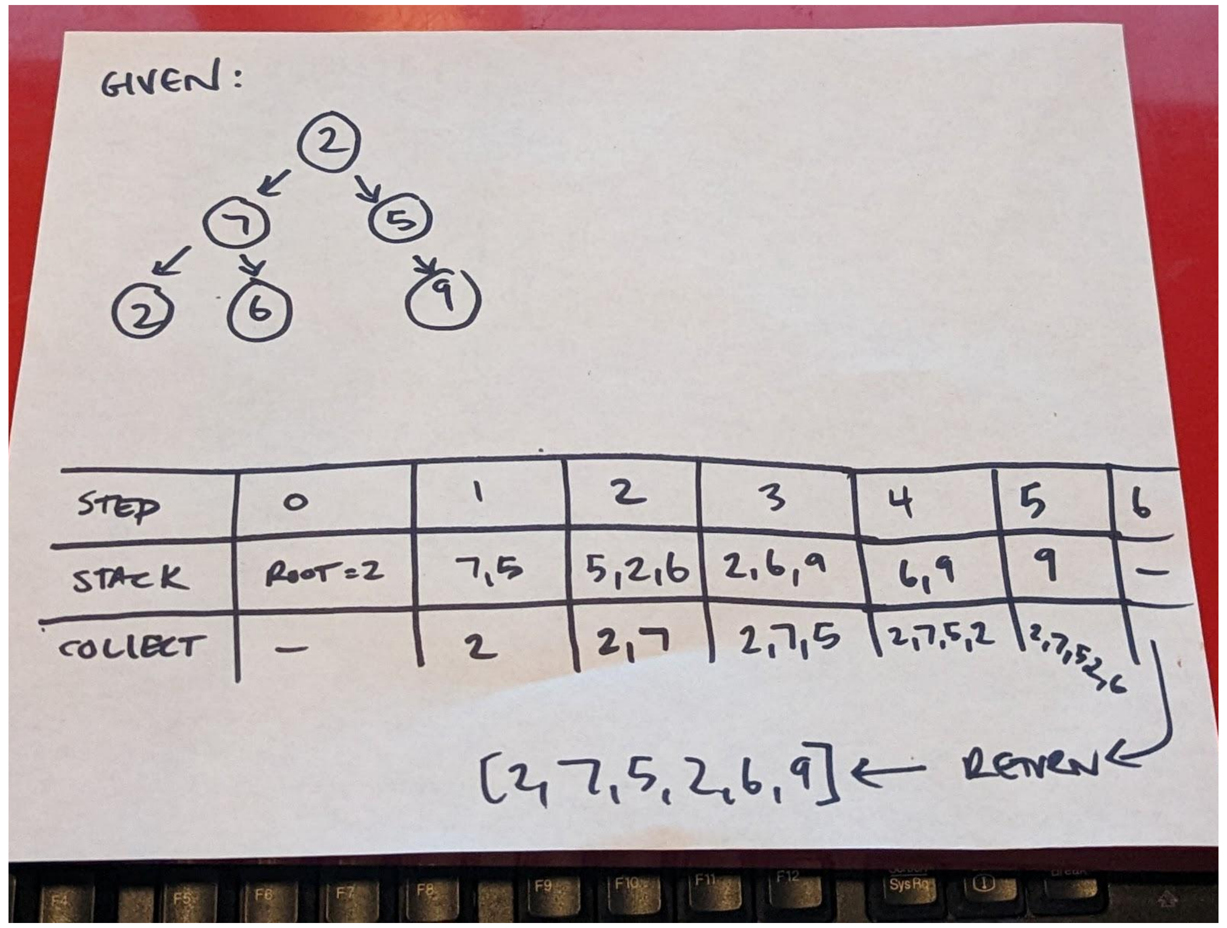
```
def breadth_first(tree):
    if not tree.root:
        return []

    stack = [tree.root]
    nodes = []

    while stack:
        node = stack.pop()
        nodes.append(node.value)
        if node.left:
            stack.insert(0, node.left)
        if node.right:
            stack.insert(0, node.right)

    return nodes
```

Step through


GIVEN:
2
7
5
2
6
9

STEP	0	1	2	3	4	5	6
STACK	ROOT=2	7,5	5,2,6	2,6,9	6,9	9	—
COLLECT	—	2	2,7	2,7,5	2,7,5,2	2,7,5,2,6	

[2,7,5,2,6,9] ← RETURN