QUESTION 1. [8 MARKS]

Read over the class declaration for BTNode, and then write the body of the function count_nodes(n).

```
class BTNode:
    """Node in a binary tree"""

    def __init__(self: 'BTNode', value: object, left: 'BTNode',
                 right: 'BTNode') -> None:
        """
        Create a new BTNode with value and (possibly)
        children left and right.
        """
        self.value, self.left, self.right = value, left, right


def count_nodes(n: BTNode) -> (int, int):
    """
    Return a tuple containing the number of interior nodes and the number
    of leaves in the tree rooted at n, or (0,0) if n is None.


    >>> count_nodes(None)
    (0, 0)
    >>> count_nodes(BTNode(5, BTNode(4, None, None), None))
    (1, 1)
    >>> count_nodes(BTNode(5, BTNode(4, None, None), BTNode(3, None, None)))
    (1, 2)
    """
    if not n:
        return (0, 0)
    else:
        left_internal, left_leaves = count_nodes(n.left)
        right_internal, right_leaves = count_nodes(n.right)
        right_count = count_nodes(n.right)
        internal, leaf = (1 if n.left or n.right else 0,
                          1 if not n.left and not n.right else 0)
        return (left_internal + right_internal + internal,
                left_leaves + right_leaves + leaf)
```

QUESTION 2. [8 MARKS]

Re-read the class declaration for BTNode, then write the body of count_even(n).

```
class BTNode:
    """Node in a binary tree"""
```

```
    def __init__(self: 'BTNode', value: object, left: 'BTNode',
                 right: 'BTNode') -> None:
        """
        Create a new BTNode with value and (possibly)
        children left and right.
        """
        self.value, self.left, self.right = value, left, right

def count_even(n: BTNode) -> int:
    """
    Return number of nodes with value that is an even int.
    You may assume all node values are integers.

    >>> count_even(None)
    0
    >>> count_even(BTNode(5, BTNode(4, None, None), BTNode(3, None, None)))
    1
    >>> count_even(BTNode(5, BTNode(4, None, None), BTNode(18, None, None)))
    2
    """
    return (count_even(n.left) + count_even(n.right) +
            (1 if n.value % 2 == 0 else 0) if n else 0)
```

QUESTION 3. [8 MARKS]

Read the class declaration for `BSTNode`. Then write the body of the function `min_path`. You may assume that the function will only be called on nodes of valid (though possibly empty) binary search trees, where all values in a left sub-tree are less than the value of the root, and all values in a right sub-tree are greater than the value of the root.

```
class BSTNode:
    """Node in a binary search tree"""

    def __init__(self: 'BSTNode', value: object, left: 'BSTNode',
                 right: 'BSTNode') -> None:
        """
        Create a new BSTNode with value and (possibly)
        children left and right."""
        self.value, self.left, self.right = value, left, right

    def __repr__(self: 'BSTNode') -> str:
        """Return a string representation of this BSTNode."""
        return ('BSTNode(' + repr(self.value) + ', ' +
                repr(self.left) + ', ' + repr(self.right) + ')')
```

```
class LLNode:
    """Node in a linked list"""

    def __init__(self: 'LLNode', value: object, nxt: 'LLNode') -> None:
        """Create a new LLnode with value, linked to nxt"""
        self.value, self.nxt = value, nxt

    def __repr__(self: 'LLNode') -> str:
        """Retur a string representation of this LLNode"""
        return 'LLNode(' + repr(self.value) + ', ' + repr(self.nxt) + ')'

def min_path(n: BSTNode) -> LLNode:
    """
    Build a linked list of the path from root to minimum element
    and return the first node in the linked list.

    >>> min_path(None)
    >>> min_path(BSTNode(4, None, None))
    LLNode(4, None)
    >>> min_path(BSTNode(4, BSTNode(3, None, None), BSTNode(5, None, None)))
    LLNode(4, LLNode(3, None))
    """
    return LLNode(n.value, min_path(n.left)) if n else None
```