```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# .....................................................................
#                        B I N A R Y   T R E E
# .....................................................................

class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# .....................................................................
#    A FUNCTION TO BE COMPLETED

    # The function countSpecificNodes returns the number of nodes
    # which have both, left and right, children,
    # and which key is bigger than the sum of keys of its children.
    def countSpecificNodes( self ):
        return self.countSpecificNodesRecursive ( self.root )

    def countSpecificNodesRecursive (self, node ):
        if node == None: return 0
        leftCount  = self.countSpecificNodesRecursive( node.left )
        rightCount = self.countSpecificNodesRecursive( node.right )
        # ... suggest how to fill in the missing code here
        # ...
        # ...
        return ...

    # For example, in the following tree, where node keys are shown in square brackets,
    # the function will count nodes [100], [20], [40].
    #
    #           __________[90]___________
    #      ___[20]___           _______[50]_______
    #   [10]      [9]        __[40]____        [30]___
    #                       [12]    [14]             [15]
    #




# .....................................................................
#       M A I N    P R O G R A M
# .....................................................................

T = BinaryTree( )
T.readFromInput()
result = T.countSpecificNodes()
print( "Number of specific nodes:", result )
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# ..................................................................
#                        B I N A R Y   T R E E
# ..................................................................
class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# ..................................................................
#    A FUNCTION TO BE COMPLETED

    # The function has40Path returns True if and only if
    # there exists a path from the root to some of the leaves
        # in which all node keys are equal to 40.
        # Otherwise the function returns False.
    #( Note: A path is sequence of nodes in which each node, except for the first one,
        #        is a child of the previous node in the sequence. )
        def has40Path( self ):
        return self.has40PathRecursive ( self.root )

    def has40PathRecursive (self, node ):
        if node == None: return True
        hasLeft40Path  = self.has40PathRecursive( node.left )
        hasRight40Path = self.has40PathRecursive( node.right )
        # ... suggest how to fill in the missing code here
            # ...
            # ...
        return ...

# ..................................................................
#       M A I N    P R O G R A M

T = BinaryTree( )
T.readFromInput()
result = T.has40Path()
print( result )
'''
Example
Tree scheme
        __________ 2 40___________
   _____3 40_____              4 -10
   0 -5      1 40

(In the scheme, first comes the node label, and then the node key)
'''
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# .....................................................................
#                         B I N A R Y   T R E E
class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# .....................................................................
#    A FUNCTION TO BE COMPLETED

    # The function rightIsAlwaysLeaf returns True if and only if
    # the right child of each internal node (including the root) is a leaf.
    # Otherwise the function returns False.
    # Note that the values of node keys in this problem are not substantial
    # and can be neglected.

    def rightIsAlwaysLeaf( self ):
        return self.rightIsAlwaysLeafRecursive( self.root )

    def rightIsAlwaysLeafRecursive (self, node ):
        if node == None: return True
        leftShapeOK = self.rightIsAlwaysLeafRecursive( node.left )
        # ... suggest how to fill in the missing code here
            # ...
            # ...
        return ...

# ........................#      M A I N   P R O G R A M

T = BinaryTree( )
T.readFromInput()
result = T.rightIsAlwaysLeaf()
print( result )
'''  Example Tree scheme
        _________ 2 10___________
   _____3 20_____              4 -10
   0 -5      1 -5
(In the scheme, first comes the node label, and then the node key)
In general, the tree shape should look like follows:
                ___0___
             ___0___  0
          ___0___  0
            ...     0
        ...
   ___0___
___0___  0
0      0
'''
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# .....................................................................
#                        B I N A R Y   T R E E

class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# .....................................................................
#    A FUNCTION TO BE COMPLETED

    # The function parentHasDifferentKey returns True if and only if
    # for each node in the tree (except for the root) it holds:
        # The value of the key of the node
        # is differrent from the value of the key of the parent of the node.
    # Otherwise the function returns False.
    # In other words, the function returns False if and only if
    # there is a node (at least one) which key is the same as the key of its parent.

    def parentHasDifferentKey( self ):
        return self.parentHasDifferentKeyRecursive( self.root )

    def parentHasDifferentKeyRecursive (self, node ):
        if node == None: return True
        leftSubtreeCheck  = self.parentHasDifferentKeyRecursive( node.left )
        rightSubtreeCheck = self.parentHasDifferentKeyRecursive( node.right )
        # ... suggest how to fill in the missing code here
        # ...
        return ...

# .....................................................................
#       M A I N    P R O G R A M
T = BinaryTree( )
T.readFromInput()
result = T.parentHasDifferentKey()
print( result )
'''
Example Tree scheme
        _________ 2 10___________
   _____3 20_____              4 -10
   0 -5      1 -5
(In the scheme, in each node, first comes the node label, and then the node key)
The output should be True
'''
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# ..................................................................
#                        B I N A R Y   T R E E
# ..................................................................

class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# ..................................................................
#    A FUNCTION TO BE COMPLETED

    # The function leavesNegative returns True if and only if
    # the keys in all leaves are negative
    # and the keys in all internal nodes are positive.
    # Otherwise the function returns False.
    def leavesNegative( self ):
        return self.leavesNegativeRecursive ( self.root )

    def leavesNegativeRecursive (self, node ):
        if node == None: return True
        leftConditionHolds  = self.leavesNegativeRecursive( node.left )
        rightConditionHolds = self.leavesNegativeRecursive( node.right )
        # ... suggest how to fill in the missing code here
        # ...
        # ...
        return ...

# ..................................................................
#       M A I N    P R O G R A M

T = BinaryTree( )
T.readFromInput()
result = T.leavesNegative()
print( result )
'''
Example  Tree scheme
        _________ 2 10__________
   _____3 20_____             4 -10
   0 -5      1 -5
(In the scheme, first comes the node label, and then the node key)
'''
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# .....................................................................
#                        B I N A R Y   T R E E
# .....................................................................

class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# .....................................................................
#    A FUNCTION TO BE COMPLETED

    # The function leafParentPositive returns True if and only if
    # each parent of a leaf in the tree
    # contains a positive key value.
    # Otherwise the function returns False.

    def leafParentPositive( self ):
        return self.leafParentPositiveRecursive( self.root )

    def leafParentPositiveRecursive (self, node ):
        if node == None: return True
        leftSubtreeCheck  = self.leafParentPositiveRecursive( node.left )
        rightSubtreeCheck = self.leafParentPositiveRecursive( node.right )
        # ... suggest how to fill in the missing code here
        # ...
        return ...

# .....................................................................
#       M A I N    P R O G R A M
T = BinaryTree( )
T.readFromInput()
result = T.leafParentPositive()
print( result )
'''
Example Tree scheme :
        _________ 2 10___________
   _____3 20_____              4 -10
   0 -5      1 -5
(In the scheme, in each node, first comes the node label, and then the node key)
The output should be True
'''
```

```
class Node:
    def __init__(self, label, key):
        self.left = None
        self.right = None
        self.key = key
        self.label = label

# .....................................................................
#                        B I N A R Y   T R E E
# .....................................................................

class BinaryTree:
    def __init__(self  ):
        self.root = None

    # build the tree
    def readFromInput(self):
        # ...

# .....................................................................
#    A FUNCTION TO BE COMPLETED

    # The function subtreesSameTotal returns True if and only if
    # the sum of positive keys in the left subtree of the root
    # is equal to the sum of positive keys in the right subtree of the root.
    # Otherwise the function returns False.
    def subtreesSameTotal( self ):
        left = self.recursiveTotal( self.root.left )
        right = self.recursiveTotal( self.root.left )

        # suggest how to complete the return statement
        return ...

    def recursiveTotal (self, node ):
        if node == None:
             return ....                          # choose return value
        leftTotal  = self.recursiveTotal( node.left )
        rightTotal  = self.recursiveTotal( node.right )
        # ...                   suggest how to fill in the missing code here:
        # ...

# .....................................................................
#       M A I N    P R O G R A M
T = BinaryTree( )
T.readFromInput()
result = T.subtreesSameTotal()
print( result )
'''
Example Tree scheme
        _________ 2 -10___________
   _____3 20_____              4 30
   0 -5      1 10
(In the scheme, first comes the node label, and then the node key)
'''
```