Problem Description: Counting Independent Sets in a Tree

In this algorithm project, our primary objective is to develop an efficient algorithm that works in linear time to count independent sets within a tree structure. Independent sets, in the context of graph theory, refer to subsets of vertices in which no two vertices are adjacent. In our case, we focus on trees, which are a special type of graph with no cycles.

Solution Description:

We utilized the depth fisrt search method. Traverse the nodes from the leaves to the root(bottom-up). For each node, we consider three states:

1. Count the number of ISs without picking the node. denote as U, the recursive procedure to do so is combinationsWithoutCurrent,

2. Count the number of ISs with the node, denote as C, the procedure is combinationWithCurrent .

3. the all IS count rooting from the node, denoted as S , which is intuitively calculated as $S = C + U$, the procedure is getAllSetsCount.

We will calculate above three states for each node recursively, until we reach the root, the number of all IS count of the root, S_{root}, is exactly the output expected. Pseudo code in python is as below:

```
1  def combinationsWithoutCurrent ( current ):
2      num = 1
3      for child in current :
4          num *= child . cachedWithoutValue + child .
               cachedWithValue
5      return num
6
7
8  def combinationsWithCurrent ( current ):
9      num = 1
10     for child in current :
11         num *= child . cachedWithoutValue
12     return num
13
14
15 def getAllSetsCount ( arrayOfNodes ):
16     for node in arrayOfNodes :
17         with_current = combinationsWithCurrent ( node )
18         without_current = combinationsWithoutCurrent (
               node )
19         node . cachedWithValue = with_current
20         node . cachedWithoutValue = without_current
21     # return root node calculated value [ -1] is the
           last element which is root in this case
22     return arrayOfNodes [ -1]. cachedWithValue +
           arrayOfNodes [ -1]. cachedWithoutValue
```

The tree and its node definitions are as below:

```
1  class TreeNode :
2      def __init__ ( self , data ):
3          self . data = data
4          self . cachedWithValue = 0
5          self . cachedWithoutValue = 0
6          self . children = []
7
8  class Tree :
9      def __init__ ( self , root_data ):
10         self . root = TreeNode ( root_data )
```

Tree is nothing but a root tree node where we can end and all IS count of the tree is exactly S_{root} of the root.

Proof of correctness and time complexity analysis

We traverse the tree node exactly once, with cache, for topdown, we can avoid redundant search. hence, $O(N)$

Proof, here we only consider the bottom up recursion, using the induction method:

- Base case: leaf node N, who has no children, according to the algorithm $U = 1, C = 1, S = 2$, which is correct, IS set count is: $\emptyset + N = 2$

- Inductive Hypothesis: The parent of the node N, denoted P. To calculate IS Count roots from P, let's divide into two cases:

  1. IS without P
  2. IS with P.

  In case of 1, the result is combinations of IS of each child. $\prod_S(C)$. We denote $S(C)$ as the stable set of the node C, the below follows this convention.

  In case of 2, the result will be the combinations of IS of each child without the child node inside, $\prod_U(C)$. However when we were calculating the $S(C)$, recall, $S = C + U$, we got and memorized the result of U, so this step does not cost any computational complexity.

  Above two cases are exactly what described in the pseudo code. The stable set of P is sum of two cases

- Inductive Step: We level up until reach the root. The stable set of the root is the result expected.

Input and output

Input: a text file, each line represents the tree in a string format, for example: 1 [2 [4, 5], 3 [6, 7]], representing a tree as below:
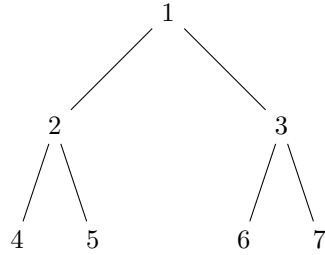
The expected output is 41, a number represents the count of the independent sets in the tree.

Input file will be input.txt under the current directory while output file will be output.txt.

Each line of the input is a tree, the corresponding line in the output file will be IS count.