

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 first search method. Traverse the nodes from the leaves to the root(bottom-up). For each node, consider a subtree, whose root is that node, to get the IS of that subtree, denoted as S , we divide it two subsets U and C :

1. IS without picking that node. denote as U (unchosen), the count number of it, it's the cardinality of the set, denoted as $|U|$, the recursive procedure to get it is `combinationsWithoutCurrent`,
2. IS with the node, denote as C (chosen), the count of it is $|C|$, the procedure is `combinationWithCurrent` .
3. the all IS, denoted as S , count number of all IS is $|S|$, the procedure is `getAllSetsCount`. Intuitively, the count number of the IS has formula as below:

$$|S| = |C| + |U|$$

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} .

Proof of correctness and time complexity analysis

We traverse the tree node exactly once, we denote the number of all nodes in the tree as N . So the time complexity of the algorithm with Big O notation is $O(N)$

Proof is as in the below, we define some notations: recall three states from the above, we denote sets U_L, C_L, S_L , as each set for the node L , we consider the bottom up recursion, using the induction method:

- Base case: Consider the case, a leaf node, denoted as L , a leaf is a node who has no children, the IS of L without picking it, is \emptyset , $|U_L| = 1$ the IS of L including it, is the node itself L , $C_L = 1$, all IS set count is: $|S_L| = |U_L| + |C_L| = 2$
- Inductive Hypothesis: Because we traverse from the bottom to the top, let's move one level up, consider the subtree starting from the parent of the node, let's denote the parent node as P . To calculate all IS Count of the subtree roots from P , let's divide into two cases:
 1. IS without P
 2. IS with P .

In case of 1, think about each child node, denoted as $D_{1...n}$ meaning N th child node of the parent node P , from 1 to N , let's consider each Independent set of the subtree roots from the child node,

$$S_D = U_D \cup C_D$$

$$S_P = \{x \in \mathbb{R} \mid x^2 = 1\}$$

The number of combinations of choosing k elements from a set of n elements is denoted as $\binom{n}{k}$. 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.

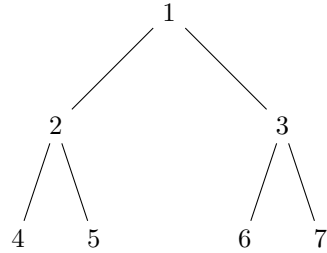
In case of 2, the result will be the combinations of IS of each child without the child node inside, $\prod_U(C)$. However when we were calculating the $S(C)$, recall, $S = C + U$, we got and memorized the result of U , so this step does not cost any computational complexity.

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.