

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 rooted trees, which are a special type of graph with no cycles. a rooted tree is a hierarchical data structure that consists of nodes connected by edges. Unlike an undirected tree, a rooted tree has a distinguished node called the "root" from which all other nodes in the tree are descended. Each node in a rooted tree has a parent-child relationship with other nodes.

1. Root: The topmost node in the tree, from which all other nodes are descended. It has no parent.
2. Node: Each element in the tree is called a node. Nodes are connected by edges.
3. Parent: A node in the tree that has one or more outgoing edges leading to its child nodes.
4. Child: A node in the tree that has a parent node.
5. Siblings: Nodes that share the same parent are called siblings.
6. Leaf: A node in the tree that has no children, i.e., it is a node with a degree of 0.
7. Subtree: A subtree is a tree formed by a node and all its descendants.

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.
5             cachedWithValue
6     return num
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(
19             node)
20         node.cachedWithValue = with_current
21         node.cachedWithoutValue = without_current
22     # return root node calculated value [-1] is the
23     # last element which is root in this case
24     return arrayOfNodes[-1].cachedWithValue +
25         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 , $U_L = \{\emptyset\}$ $|U_L| = 1$, 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 $D_{1\dots n}$, meaning N th child node of the parent node P , from 1st to N th, let's consider all Independent sets of each subtree roots from the child node,

$$S_D = U_D \cup C_D \quad (*)$$

Because there is no P in the any set of S_P , recall the definition of independent set, we have no limit in combine all the sets inside each child IS set. We consider Cartesian product of each S_{D_n} , we denote a_{D_n} as any element in S_{D_n} :

$$U_P = S_{D_1} \times S_{D_2} \times \dots \times S_{D_n} = \{(a_{D_1}, a_{D_2}, \dots, a_{D_n}) \mid a_{D_1} \in S_{D_1}, a_{D_2} \in S_{D_2}, \dots, a_{D_n} \in S_{D_n}\}$$

Here, the each pair does not need be ordered.

The cardinality of the Cartesian product $S_{D_1} \times S_{D_2} \times \dots \times S_{D_n}$ is the product of the cardinalities of the individual sets:

$$|U_P| = |S_{D_1} \times S_{D_2} \times \dots \times S_{D_n}| = |S_{D_1}| \cdot |S_{D_2}| \cdot \dots \cdot |S_{D_n}|$$

In case of 2, due to the fact the parent node is included in each set, similarly, the result will be the combinations of IS of each child without the child node inside (set of U), hence:

$$C_P = U_{D_1} \times U_{D_2} \times \dots \times U_{D_n} = \{(a_{D_1}, a_{D_2}, \dots, a_{D_n}) \mid a_{D_1} \in U_{D_1}, a_{D_2} \in U_{D_2}, \dots, a_{D_n} \in U_{D_n}\}$$

$$|C_P| = |U_{D_1} \times U_{D_2} \times \dots \times U_{D_n}| = |U_{D_1}| \cdot |U_{D_2}| \cdot \dots \cdot |U_{D_n}|$$

recall equation *, in the case one, to get S_D , we already get U_D first, due to the cache, we don't need compute it again.

hence, the case two does not add extra computational complexity.

Above two cases are exactly what described in the pseudo code.

The final step:

$$S_P = U_P \cup C_P$$

$$|S_P| = |U_P| + |C_P|$$

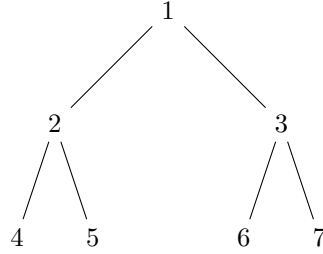
- Inductive Step: For each level up, we do the exactly as Inductive Hypothesis, until reach the root. $|S_{root}|$, is exactly the output expected.

Input and output

Input: a text file, Some lists to represent a tree: for example:

```
1 2 3
2 4 5
3 6 7
4
5
6
7
```

representing a tree as below:



The first item in each line represents the node itself, the subsequent items are its children.

We start from the first line, first node represents the root of the tree, the subsequent items are the children of it.

Then we focus on the next child of the root, in the above case is 2, we represent node 2 and its children in a new line.

Then node 3, then jump to next level, the node 4, which is the child of node 2, and etc, note we jump level by level.

When a node is a leaf, the line will be the node itself without subsequence.

The output file is a number, count of IS of the whole tree, for the above case is, 41.

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