1008. Construct Binary Search Tree from Preorder Traversal

```
from bisect import *
class Solution:
    def bstFromPreorder(self, preorder: List[int]) -> TreeNode:
        if not preorder:
            return None
        node = TreeNode(preorder[0])
        length = len(preorder)
        idx = bisect_left(preorder, preorder[0], 1, length)
        node.left = self.bstFromPreorder(preorder[1:idx])
        node.right = self.bstFromPreorder(preorder[idx:])
        return node
```

```
from bisect import *
class Solution:
    def bstFromPreorder(self, preorder: List[int]) -> TreeNode:
        def helper(i, j):
            if i > j:
                return None
            node = TreeNode(preorder[i])
            idx = bisect_left(preorder, preorder[i], i + 1, j + 1)
            node.left = helper(i + 1, idx - 1)
            node.right = helper(idx, j)
            return node
        return helper(0, len(preorder) - 1)
```

O(n) solution

```
class Solution:
    def bstFromPreorder(self, preorder: List[int]) -> TreeNode:
        self.i = 0
        def helper(bound):
            if self.i == len(preorder) or preorder[self.i] > bound:
                return None
            node = TreeNode(preorder[self.i])
            self.i += 1
            node.left = helper(node.val)
            node.right = helper(bound)
            return node
        return helper(float('inf'))
```

536. Construct Binary Tree from String

TC is O(n)

```
class Solution:
```

```
    def str2tree(self, s: str) -> TreeNode:
        if not s:
            return None
        num = ""
        stack = []
        for c in s:
            if c not in '()':
                num += c
            else:
                if c == '(':
                    if num:
                        stack.append(TreeNode(int(num)))
                else:
                    node = TreeNode(int(num)) if num else stack.pop()

                    if stack[-1].left:
                        stack[-1].right = node
                    else:
                        stack[-1].left = node
                num = ''
        return stack[-1] if not num else TreeNode(int(num))
```

133. Clone Graph

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        memo = {}
        cur = set([node])
        memo[node] = Node(node.val, [])
        while cur:
            next_ite = set()
            for n in cur:
                for c in n.neighbors:
                    if c not in memo:
                        memo[c] = Node(c.val, [])
                        next_ite.add(c)
            cur = next_ite
        def helper(node):
            for c in node.neighbors:
                memo[node].neighbors.append(memo[c])
                if not memo[c].neighbors:
                    helper(c)
        helper(node)
        return memo[node]
```

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        memo = {}
        memo[node] = Node(node.val, [])
        def helper(node):
            for c in node.neighbors:
                if c not in memo:
                    memo[c] = Node(c.val, [])
                memo[node].neighbors.append(memo[c])
                if not memo[c].neighbors:
                    helper(c)
        helper(node)
        return memo[node]
```