LC 783: Minimum Distance Between BST Nodes

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def minDiffInBST(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        res = [float('inf')]
        def min_max(node):
            l, r = node.val, node.val
            if node.left:
                mi, ma = min_max(node.left)
                res[0] = min(res[0], node.val - ma)
                l = mi
            if node.right:
                mi, ma = min_max(node.right)
                res[0] = min(res[0], mi - node.val)
                r = ma
            return l, r
        _, _ = min_max(root)
        return res[0]
```

Success Details ›

Runtime: 20 ms, faster than 58.37% of Python online submissions for Minimum Distance Between BST Nodes.

Memory Usage: 12.1 MB, less than 14.29% of Python online submissions for Minimum Distance Between BST Nodes.

Reverse a Doubly linked list using recursion

```python
class Node:
    def __init__(self, data):
        self.val = data
        self.next = None
        self.prev = None

def reverse(node):
    if not node:
        return None
    tmp = node.next
    node.next = node.prev
    node.prev = tmp
    return node if not tmp else reverse(tmp)

def print_out(node):
    while node:
        print(node.val)
        node = node.next

a, b, c, d = Node(1), Node(2), Node(3), Node(4)
a.next = b
b.prev, b.next = a, c
c.prev, c.next = b, d
d.prev = c
print('Double linked list:')
print_out(a)
reverse = reverse(a)
print('Revered double linked list:')
print_out(reverse)
```

Output:

```
Double linked list:
1
2
3
4
Revered double linked list:
4
3
2
1
```

LC 687: Longest Univalue Path

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def longestUnivaluePath(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        res = [0]
        def sub_max_len(node):
            if not node:
                return 0
            l, r = sub_max_len(node.left), sub_max_len(node.right)
            l = l + 1 if node.left and node.val == node.left.val else 0
            r = r + 1 if node.right and node.val == node.right.val else 0
            res[0] = max(res[0], l + r)
            return max(l, r)

        _ = sub_max_len(root)
        return res[0]
```

Success Details ›

Runtime: 436 ms, faster than 67.23% of Python online submissions for Longest Univalue Path.

Memory Usage: 18.9 MB, less than 27.27% of Python online submissions for Longest Univalue Path.

LC 698. Partition to K Equal Sum Subsets

```python
class Solution(object):
    def canPartitionKSubsets(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """
        if not nums or not k:
            return False
        total = sum(nums)
        if total%k != 0:
            return False
        target = total/k
        visited = [False]*len(nums)
        nums.sort(reverse = True)
        return self.dfs(0, nums, visited, k, 0, target)

    def dfs(self, start, nums, visited, k, temp_sum, target):
        if k == 1:
            return True
        if temp_sum == target:
            return self.dfs(0, nums, visited, k - 1, 0, target)
        for i in range(start, len(nums)):
            if  visited[i] or temp_sum + nums[i] > target:
                continue
            visited[i] = True
            if self.dfs(i + 1, nums, visited, k, temp_sum + nums[i], target):
                return True
            visited[i] = False
        return False
```

Success Details ›

Runtime: 20 ms, faster than 98.01% of Python online submissions for Partition to K Equal Sum Subsets.

Memory Usage: 11.7 MB, less than 80.00% of Python online submissions for Partition to K Equal Sum Subsets.

LC 794: Valid Tic-Tac-Toe State

```python
class Solution(object):
    def validTicTacToe(self, board):
        """
        :type board: List[str]
        :rtype: bool
        """
        n_o, n_x = 0, 0
        for i in range(len(board)):
            for j in range(len(board[i])):
                if board[i][j] == 'X':
                    n_x += 1
                elif board[i][j] == 'O':
                    n_o += 1

        if n_o > n_x:
            return False
        elif n_o == n_x:
            if self.check_win(board, 'X'):
                return False
            else:
                return True
        elif n_x == n_o + 1:
            if self.check_win(board, 'O'):
                return False
            else:
                return True
        else:
            return False

    def check_win(self, board, cha):
        if   board[0][0] == board[0][1] == board[0][2] == cha:
            return True
        elif board[1][0] == board[1][1] == board[1][2] == cha:
            return True
        elif board[2][0] == board[2][1] == board[2][2] == cha:
            return True
        elif board[0][0] == board[1][0] == board[2][0] == cha:
            return True
        elif board[0][1] == board[1][1] == board[2][1] == cha:
            return True
        elif board[0][2] == board[1][2] == board[2][2] == cha:
            return True
        elif board[0][0] == board[1][1] == board[2][2] == cha:
            return True
        elif board[2][0] == board[1][1] == board[0][2] == cha:
            return True
        else:
            return False
```

Success Details ›

Runtime: 12 ms, faster than 94.02% of Python online submissions for Valid Tic-Tac-Toe State.

Memory Usage: 11.7 MB, less than 50.00% of Python online submissions for Valid Tic-Tac-Toe State.

LC 726: Number of Atoms

```python
import collections
class Solution(object):
    def countOfAtoms(self, formula):
        """
        :type formula: str
        :rtype: str
        """
        def sub_arr(idx, times):
            while idx >= 0 and formula[idx] != '(':
                n, power = 0, 0
                while formula[idx].isdigit():
                    n += int(formula[idx])*10**power
                    idx -= 1
                    power += 1
                if not n:
                    n = 1
                if formula[idx] == ')':
                    idx = sub_arr(idx - 1, n * times)
                    continue
                name = ''
                while formula[idx].islower():
                    name += formula[idx]
                    idx -= 1
                name += formula[idx] # This is a capital letter, start of atom name
                idx -= 1
                cnt_map[name[::-1]] += n * times
            return idx - 1
        cnt_map = collections.defaultdict(int)
        sub_arr(len(formula) - 1, 1)
        res = ''
        for name, cnt in sorted(cnt_map.items()):
            res += name + (str(cnt) if cnt > 1 else '')
        return res
```

Success Details ›

Runtime: 16 ms, faster than 85.45% of Python online submissions for Number of Atoms.

Memory Usage: 11.9 MB, less than 100.00% of Python online submissions for Number of Atoms.