搜索

二叉树的序列化与反序列化BFS（297）

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

示例:

序列化为 "[1,2,3,null,null,4,5]"
提示: 这与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

说明: 不要使用类的成员 / 全局 / 静态变量来存储状态，你的序列化和反序列化算法应该是无状态的。

In [1]:

```python
class Codec:
    def serialize(self, root):
        if root == None:
            return '*!'
        pre_str = ''
        pre_str +=str(root.val)+'!'
        pre_str +=self.serialize(root.left)
        pre_str +=self.serialize(root.right)
        return pre_str
    def deserialize(self, data):
        values = data.split('!')
        return self.deserialize_pre_str(values)

    def deserialize_pre_str(self,values):
        value = values.pop(0)
        if value == '*':
            return None
        root = TreeNode(value)
        root.left = self.deserialize_pre_str(values)
        root.right = self.deserialize_pre_str(values)
        return root
```

岛屿的最大面积 DFS（695）

给定一个包含了一些 0 和 1的非空二维数组 grid , 一个 岛屿 是由四个方向 (水平或垂直) 的 1 (代表土地) 构成的组合。你可以假设二维矩阵的四个边缘都被水包围着。

找到给定的二维数组中最大的岛屿面积。(如果没有岛屿，则返回面积为0。)

示例 1:

[[0,0,1,0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],
[0,1,0,0,1,1,0,0,1,0,1,0,0],
[0,1,0,0,1,1,0,0,1,1,1,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,0,0,0,0,0,0,1,1,0,0,0,0]]

对于上面这个给定矩阵应返回 6。注意答案不应该是11，因为岛屿只能包含水平或垂直的四个方向的'1'。

示例 2:

[[0,0,0,0,0,0,0,0]]
对于上面这个给定的矩阵, 返回 0。
注意: 给定的矩阵grid 的长度和宽度都不超过 50。

In [2]:

```python
class Solution(object):
    def maxAreaOfIsland(self, grid):
        max_area = 0
        if len(grid) == 0 or len(grid[0]) == 0: return max_area
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 0: continue
                tmp_area = 0
                children = [
                    (i, j),
                ]
                while children:
                    r, c = children.pop()
                    if grid[r][c] == 0: continue
                    grid[r][c] = 0
                    tmp_area += 1
                    if r - 1 >= 0 and grid[r - 1][c]:
                        children.append((r - 1, c))
                    if r + 1 < len(grid) and grid[r + 1][c]:
                        children.append((r + 1, c))
                    if c - 1 >= 0 and grid[r][c - 1]:
                        children.append((r, c - 1))
                    if c + 1 < len(grid[0]) and grid[r][c + 1]:
                        children.append((r, c + 1))
                max_area = max(max_area, tmp_area)
        return max_area
```

In [3]:

```python
s = Solution()
grid1 = [[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
         [0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0],
         [0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]]
result1 = s.maxAreaOfIsland(grid1)
print(result1)

grid2 = [[0,0,0,0,0,0,0,0]]
result2 = s.maxAreaOfIsland(grid2)
print(result2)
```

```
6
0
```

电话号码的字母组合Backtracking（17）

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。
给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。
示例:
输入："23"
输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
说明:
尽管上面的答案是按字典序排列的，但是你可以任意选择答案输出的顺序。

In [4]:

```python
class Solution:
    def letterCombinations(self, digits):
        KEY = {
            '2': ['a', 'b', 'c'],
            '3': ['d', 'e', 'f'],
            '4': ['g', 'h', 'i'],
            '5': ['j', 'k', 'l'],
            '6': ['m', 'n', 'o'],
            '7': ['p', 'q', 'r', 's'],
            '8': ['t', 'u', 'v'],
            '9': ['w', 'x', 'y', 'z']
        }
        if digits == '':
            return []
        ans = ['']
        for num in digits:
            ans = [pre + suf for pre in ans for suf in KEY[num]]
        return ans
```

In [5]:

```python
s = Solution()
result = s.letterCombinations("23")
print(result)
```

```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
```

动态规划

爬楼梯 - 斐波那契数列（70）

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。
每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？
注意：给定 n 是一个正整数。
示例 1：
输入： 2
输出： 2
解释： 有两种方法可以爬到楼顶。

1. 1 阶 + 1 阶
2. 2 阶

示例 2：
输入： 3
输出： 3
解释： 有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶
2. 1 阶 + 2 阶
3. 2 阶 + 1 阶

In [6]:

```python
class Solution:
    def climbStairs(self, n):
        if n == 1 or n == 2:
            return n
        else:
            a, b, count = 1, 2, 3
            while count <= n:
                a, b = b, a+b
                count += 1
        return b
```

In [7]:

```python
s = Solution()
result1 = s.climbStairs(2)
print(result1)
result2 = s.climbStairs(3)
print(result2)
```

```
2
3
```

最小路径和-矩阵路径（64）

给定一个包含非负整数的 m x n 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。
说明：每次只能向下或者向右移动一步。
示例:
输入:
[[1,3,1],
[1,5,1],
[4,2,1]]
输出: 7
解释: 因为路径 1→3→1→1→1 的总和最小。

In [8]:

```python
class Solution:
    def minPathSum(self, grid):
        row = len(grid)
        column = len(grid[0])
        dp = [[0 for _ in range(column)] for p in range(row)]
        for i in range(column):
            dp[0][i] = dp[0][i-1] + grid[0][i]
        for i in range(1, row, 1):
            dp[i][0] = dp[i-1][0] + grid[i][0]
            for j in range(1, column, 1):
                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
        return dp[row-1][column-1]
```

In [9]:

```python
s = Solution()
grid = [[1, 3, 1], [1, 5, 1], [4, 2, 1]]
result = s.minPathSum(grid)
print(result)
```

```
7
```

区域和检索 - 数组不可变-数组区间（303）

给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i，j 两点。
示例:
给定 nums = [-2, 0, 3, -5, 2, -1]，求和函数为 sumRange()
sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
说明:
你可以假设数组不可变。
会多次调用 sumRange 方法。

In [10]:

```python
class NumArray(object):
    def __init__(self, nums):
        self.nums = [0] + nums
        for i in range(1, len(self.nums)):
            self.nums[i] = self.nums[i-1] + self.nums[i]

    def sumRange(self, i, j):
        return self.nums[j+1] - self.nums[i]
```

In [11]:

```python
nums = [-2, 0, 3, -5, 2, -1]
n = NumArray(nums)
result1 = n.sumRange(0, 2)
print(result1)
result2 = n.sumRange(2, 5)
print(result2)
result3 = n.sumRange(0, 5)
print(result3)
```

```
1
-1
-3
```

整数拆分-分割整数（343）

给定一个正整数 n，将其拆分为至少两个正整数的和，并使这些整数的乘积最大化。 返回你可以获得的最大乘积。
示例 1:
输入: 2
输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1。
示例 2:
输入: 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36。
说明: 你可以假设 n 不小于 2 且不大于 58。

In [12]:

```python
class Solution(object):
    def integerBreak(self, n):
        if n<=3:
            return n-1
        dp = [x+1 for x in range(n)]
        for i in range(3,n):
            dp[i] = max(dp[i-2]*2,dp[i-3]*3)
        return max(dp)
```

In [13]:

```python
s = Solution()
result1 = s.integerBreak(2)
print(result1)
result2 = s.integerBreak(10)
print(result2)
```

```
1
36
```

最长递增子序列（300）

给定一个无序的整数数组，找到其中最长上升子序列的长度。
示例:
输入: [10,9,2,5,3,7,101,18]
输出: 4
解释: 最长的上升子序列是 [2,3,7,101]，它的长度是 4。
说明:
可能会有多种最长上升子序列的组合，你只需要输出对应的长度即可。
你算法的时间复杂度应该为 O(n2) 。
进阶: 你能将算法的时间复杂度降低到 O(n log n) 吗?

In [14]:

```python
class Solution(object):

    # 动态规划的思路：将 dp 数组定义为：以 nums[i] 结尾的最长上升子序列的长度
    # 那么题目要求的，就是这个 dp 数组中的最大者
    # 以数组  [10, 9, 2, 5, 3, 7, 101, 18] 为例：
    # dp 的值：  1  1  1  2  2  3  4    4

    def lengthOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        size = len(nums)
        if size <= 1:
            return size
        dp = [1] * size
        for i in range(1, size):
            for j in range(i):
                if nums[i] > nums[j]:
                    # + 1 的位置不要加错了
                    dp[i] = max(dp[i], dp[j] + 1)
        # 最后要全部走一遍，看最大值
        return max(dp)
```

In [15]:

```python
nums =  [10,9,2,5,3,7,101,18]
s = Solution()
result = s.lengthOfLIS(nums)
print(result)
```

```
4
```

分割等和子集--0-1 背包（416）

给定一个只包含正整数的非空数组。是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。
注意:
每个数组中的元素不会超过 100
数组的大小不会超过 200
示例 1:
输入: [1, 5, 11, 5]
输出: true
解释: 数组可以分割成 [1, 5, 5] 和 [11].
示例 2:
输入: [1, 2, 3, 5]
输出: false
解释: 数组不能分割成两个元素和相等的子集.

In [16]:

```python
class Solution(object):
    def canPartition(self, nums):
        """
        动态规划，memo[i]记录的是这些数字能否组成i（True or False）
        :type nums: List[int]
        :rtype: bool
        """
        if sum(nums) % 2 == 1:
            return False

        half_sum = int(sum(nums) / 2)
        n = len(nums)
        memo = [False] * (half_sum + 1)

        # 初始化，memo[0] = True是必然的，因为我们只要取空子集，那么其sum一定为0
        memo[0] = True
        # 每遍历一个数就更新一遍memo
        # memo要从后往前遍历，因为如果i<num，前面就不需要改了
        for num in nums:
            for i in range(half_sum, num - 1, -1):
                # memo[i]表示不取这个数，memo[i-num]表示取这个数
                memo[i] = memo[i] or memo[i - num]
        return memo[half_sum]
```

In [17]:

```python
nums1 =[1, 5, 11, 5]
nums2 = [1, 2, 3, 5]
s = Solution()
result1 = s.canPartition(nums1)
print(result1)
result2 = s.canPartition(nums2)
print(result2)
```

```
True
False
```

最佳买卖股票时机含冷冻期-股票交易（309）

给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：
你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。
示例:
输入: [1,2,3,0,2]
输出: 3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]

sell[i]表示截至第i天，最后一个操作是卖时的最大收益；
buy[i]表示截至第i天，最后一个操作是买时的最大收益；
cool[i]表示截至第i天，最后一个操作是冷冻期时的最大收益；
递推公式：
sell[i] = max(buy[i-1]+prices[i], sell[i-1]) (第一项表示第i天卖出，第二项表示第i天冷冻)
buy[i] = max(cool[i-1]-prices[i], buy[i-1]) （第一项表示第i天买进，第二项表示第i天冷冻）
cool[i] = max(sell[i-1], buy[i-1], cool[i-1])

In [18]:

```python
class Solution(object):
    def maxProfit(self, prices):
        n = len(prices)
        if n == 0:
            return 0
        sell = [0 for _ in range(n)]
        buy = [0 for _ in range(n)]
        cool = [0 for _ in range(n)]
        buy[0] = -prices[0]
        for i in range(1,n):
            sell[i] = max(buy[i-1] + prices[i], sell[i-1])
            buy[i] = max(cool[i-1] - prices[i], buy[i-1])
            cool[i] = max(sell[i-1], buy[i-1],cool[i-1])
        return sell[-1]
```

In [19]:

```python
s = Solution()
prices =  [1,2,3,0,2]
result = s.maxProfit(prices)
print(result)
```

```
3
```

两个字符串的删除操作-字符串编辑（583）

给定两个单词 word1 和 word2，找到使得 word1 和 word2 相同所需的最小步数，每步可以删除任意一个字符串中的一个字符。
示例 1:
输入: "sea", "eat"
输出: 2
解释: 第一步将"sea"变为"ea"，第二步将"eat"变为"ea" 说明:
给定单词的长度不超过500。
给定单词中的字符只含有小写字母。

In [20]:

```python
class Solution(object):
    def minDistance(self, word1, word2):
        """
        :type word1: str
        :type word2: str
        :rtype: int
        """
        # 相当于找word1,word2的最长公共子序列
        dp = [[0 for _ in range(len(word1) + 1)]
              for _ in range(len(word2) + 1)]
        for i in range(1, len(word2) + 1):
            for j in range(1, len(word1) + 1):
                if word2[i - 1] == word1[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + 1
                else:
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
        return len(word1) + len(word2) - 2 * dp[-1][-1]
```

In [21]:

```python
s = Solution()
word1, word2 = "sea", "eat"
result = s.minDistance(word1, word2)
print(result)
```

```
2
```