22. 括号生成

难度 中等 1943 收藏 分享 切换为英文 接收动态

数字 `n` 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 **有效的** 括

示例 1：

```
输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]
```

示例 2：

```
输入：n = 1
输出：["()"]
```

```
classSolution:
    defgenerateParenthesis(self, n: int) -> List[str]:
        res = []
        cur_str = ''

        defdfs(cur_str, left, right):
            if left == 0and right == 0:
                res.append(cur_str)
                return

            if right < left:
                return

            if left > 0:
                dfs(cur_str + '(', left - 1, right)

            if right > 0:
                dfs(cur_str + ")", left, right - 1)

        dfs(cur_str, n, n)
        return res
```


89. 格雷编码

难度 中等 319 收藏 分享 切换为英文 接收动态 反

```python
class Solution:
    def grayCode(self, n: int) -> List[int]:
        if n == 0:
            return [0]
        res = []
        def dfs(cur, x):
            if len(cur) == n:
                res.append(int(cur, 2))
                return

            if x == 0:
                dfs(cur + '0', 0)
                dfs(cur + '1', 1)
            else:
                dfs(cur + '1', 0)
                dfs(cur + '0', 1)

        dfs('', 0)
        return res
```

17. 电话号码的字母组合

难度 中等 1426 收藏 分享 切换为英文 接收动态 反

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。答案可以按 **任意顺序**

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 1：

```
输入：digits = "23"
输出：["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

示例 2：

```
输入：digits = ""
输出：[]
```



```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []
        d = ["  ", "*", "abc", "def", "ghi", "jkl", "mno", "pqrs", "t
        res = []
        def dfs(tmp, index):
            if index == len(digits):
                res.append(tmp)
                return

            c = digits[index]
            letters = d[ord(c) - 48]

            for letter in letters:
                dfs(tmp + letter, index + 1)

        dfs('', 0)
        return res
```

51. N 皇后

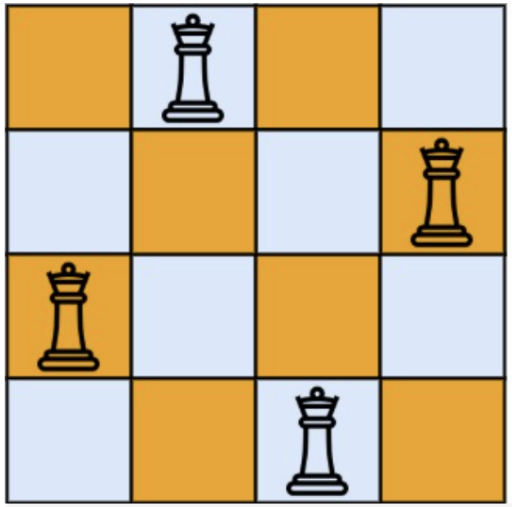
难度 困难 959 收藏 分享 切换为英文 接收动态 反

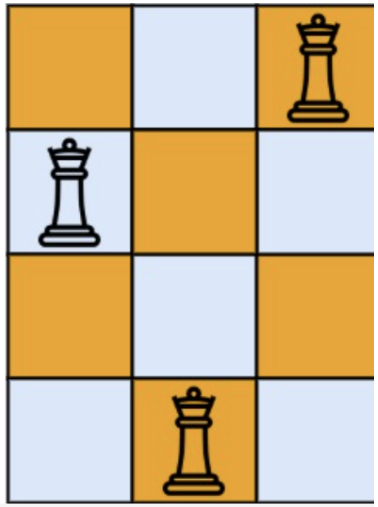
n 皇后问题 研究的是如何将 `n` 个皇后放置在 `n×n` 的棋盘上，并且使皇后彼此之间不能相互

给你一个整数 `n` ，返回所有不同的 **n 皇后问题** 的解决方案。

每一种解法包含一个不同的 **n 皇后问题** 的棋子放置方案，该方案中 `'Q'` 和 `'.'` 分别代表
位。

示例 1：





输入： n = 4
输出： [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]
解释： 如上图所示，4 皇后问题存在两个不同的解法。


```python
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        def DFS(queens, xy_dif, xy_sum):
            p = len(queens)
            if p == n:
                result.append(queens)
                return

            for q in range(n):
                if q not in queens and p - q not in xy_dif and p + q
                    DFS(queens + [q], xy_dif + [p - q], xy_sum + [p

        result = []
        DFS([], [], [])
        return [['.' * i + 'Q' + '.' * (n - i - 1) for i in sol] for
```

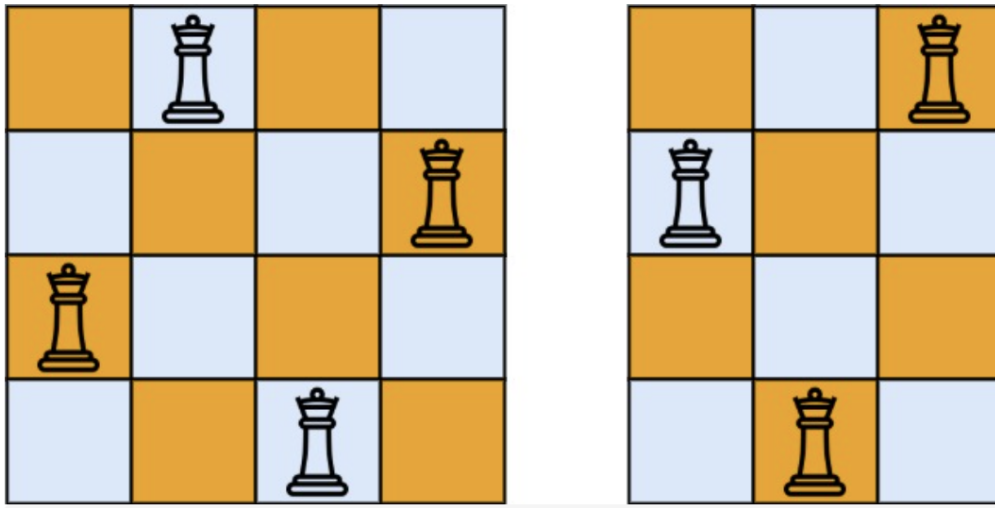
52. N皇后 II

难度 困难 283 收藏 分享 切换为英文 接收动态 反

n 皇后问题 研究的是如何将 `n` 个皇后放置在 `n×n` 的棋盘上，并且使皇后彼此之间不能相互

给你一个整数 `n` ，返回 **n 皇后问题** 不同的解决方案的数量。

示例 1：



输入： n = 4
输出： 2
解释： 如上图所示，4 皇后问题存在两个不同的解法。


```python
class Solution:
    def totalNQueens(self, n: int) -> int:
        if n < 1:
            return []

        self.res = 0
        self.dfs(n, 0, 0, 0, 0)
        return self.res

    def dfs(self, n, row, col, lfs, rfs):
        if row == n:
            self.res += 1
            return

        bits = (~(col | lfs | rfs)) & ((1 << n) - 1)

        while bits:
            p = bits & (-bits)
            bits = bits & (bits - 1)
            self.dfs(n, row + 1, col | p, (lfs | p) << 1, (rfs | p)
```

131. 分割回文串

难度 中等 798 收藏 分享 切换为英文 接收动态 反

给你一个字符串 `s`，请你将 `s` 分割成一些子串，使每个子串都是 **回文串** 。返回 `s` 所有可
案。

回文串 是正着读和反着读都一样的字符串。

示例 1：

```
输入：s = "aab"
输出：[["a","a","b"],["aa","b"]]
```

示例 2：

```
输入：s = "a"
输出：[["a"]]
```



```python
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        res = []
        path = []
        n = len(s)
        if n == 0:
            return res
        self.dfs(s, n, 0, path, res)
        return res

    def dfs(self, s, n, start, path, res):
        if start == n:
            res.append(path[:])
            return
        for i in range(start, n):
            if not self.check_pali(s, start, i):
                continue
            path.append(s[start:i + 1])
            self.dfs(s, n, i + 1, path, res)
            path.pop()

    def check_pali(self, s, left, right):
        while left < right:
            if s[left] != s[right]:
                return False
            left += 1
            right -= 1
        return True
```

93. 复原 IP 地址

难度 中等 641 收藏 分享 切换为英文 接收动态 反

给定一个只包含数字的字符串，用以表示一个 IP 地址，返回所有可能从 `s` 获得的 **有效 IP 地**
以按任何顺序返回答案。

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 `0`），整
`'.'` 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 **有效** IP 地址，但是 "0.011.255.245"、"192.168
"192.168@1.1" 是 **无效** IP 地址。

示例 1：

```
输入：s = "25525511135"
输出：["255.255.11.135","255.255.111.35"]
```

示例 2：

```
输入：s = "0000"
输出：["0.0.0.0"]
```

示例 3：

```
输入：s = "1111"
输出：["1.1.1.1"]
```



```python
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        n = len(s)
        if n < 4 or n > 12:
            return []
        path = []
        res = []
        self.dfs(s, n, 0, 0, path, res)
        return res

    def dfs(self, s, n, begin, split, path, res):
        if begin == n:
            if split == 4:
                res.append('.'.join(path[:]))
            return

        if n - begin < 4 - split and n - begin > 3 * (4 - split):
            return

        for i in range(3):
            if begin + i >= n:
                break

            ip = self.ip_judge(s, begin, begin + i)
            if ip != -1:
                path.append(str(ip))
                self.dfs(s, n, begin + i + 1, split + 1, path, res)
                path.pop()

    def ip_judge(self, s, left, right):
        size = right - left + 1
        if size > 1 and s[left] == '0':
            return -1
        num = int(s[left: right + 1])
        if num > 255:
            return -1
        return num
```

980. 不同路径 III

难度 困难 148 收藏 分享 切换为英文 接收动态 反馈

在二维网格 `grid` 上，有 4 种类型的方格：

- `1` 表示起始方格。且只有一个起始方格。
- `2` 表示结束方格，且只有一个结束方格。
- `0` 表示我们可以走过的空方格。
- `-1` 表示我们无法跨越的障碍。

返回在四个方向（上、下、左、右）上行走时，从起始方格到结束方格的不同路径的数目。

每一个无障碍方格都要通过一次，但是一条路径中不能重复通过同一个方格。

示例 1：

```
输入：[[1,0,0,0],[0,0,0,0],[0,0,2,-1]]
输出：2
解释：我们有以下两条路径：
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,
2. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,
```



```python
class Solution:
    def uniquePathsIII(self, grid: List[List[int]]) -> int:
        self.res = 0
        m, n, empty = len(grid), len(grid[0]), 1
        for i in range(m):
            for j in range(n):
                if grid[i][j] == 1:
                    x, y = i, j
                if grid[i][j] == 0:
                    empty += 1

        def dfs(x, y, empty):
            if not (0 <= x < m and 0 <= y < n and grid[x][y] >= 0):
                return
            if grid[x][y] == 2:
                self.res += empty == 0
                return
            grid[x][y] = -2
            dfs(x + 1, y, empty - 1)
            dfs(x - 1, y, empty - 1)
            dfs(x, y + 1, empty - 1)
            dfs(x, y - 1, empty - 1)
            grid[x][y] = 0

        dfs(x, y, empty)
        return self.res
```

77. 组合

难度 中等 648 收藏 分享 切换为英文 接收动态 反

给定两个整数 `n` 和 `k`，返回范围 `[1, n]` 中所有可能的 `k` 个数的组合。

你可以按 **任何顺序** 返回答案。

示例 1：

```
输入：n = 4, k = 2
输出：
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

示例 2：

```
输入：n = 1, k = 1
输出：[[1]]
```



```python
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        if n <= 0 or k <= 0 or k > n:
            return []
        res = []
        self.__dfs(1, k, n, [], res)
        return res

    def __dfs(self, start, k, n, pre, res):
        if len(pre) == k:
            res.append(pre[:])
            return

        for i in range(start, n - (k - len(pre)) + 2):
            pre.append(i)
            self.__dfs(i+1, k, n, pre, res)
            pre.pop()
```

46. 全排列

难度 中等 1485 收藏 分享 切换为英文 接收动态

给定一个不含重复数字的数组 `nums` ，返回其 **所有可能的全排列** 。你可以 **按任意顺序** 返回

示例 1：

```
输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

示例 2：

```
输入：nums = [0,1]
输出：[[0,1],[1,0]]
```

示例 3：

```
输入：nums = [1]
输出：[[1]]
```



```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        def dfs(size, nums, depth, path, res, record):
            if depth == size:
                res.append(path[:])
                return
            for i in range(size):
                if nums[i] not in record:
                    path.append(nums[i])
                    record.add(nums[i])
                    dfs(size, nums, depth + 1, path, res, record)
                    path.pop()
                    record.remove(nums[i])
        size = len(nums)
        depth = 0
        path = []
        res = []
        record = set()
        dfs(size, nums, depth, path, res, record)
        return res
```

47. 全排列 II

难度 中等 767 收藏 分享 切换为英文 接收动态 反

给定一个可包含重复数字的序列 `nums` ，**按任意顺序** 返回所有不重复的全排列。

示例 1：

```
输入：nums = [1,1,2]
输出：
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

示例 2：

```
输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```



```python
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        def dfs(nums, size, depth, path, used, res):
            if depth == size:
                res.append(path[:])
                return

            for i in range(size):
                if not used[i]:

                    if i > 0 and nums[i] == nums[i-1] and not used[i-
                        continue

                    used[i] = True
                    path.append(nums[i])
                    dfs(nums, size, depth+1, path, used, res)
                    used[i] = False
                    path.pop()

        size = len(nums)
        if size == 0:
            return []
        nums.sort()
        used = [False] *size
        res = []
        dfs(nums, size, 0, [], used, res)
        return res
```

50. Pow(x, n)

难度 中等 704 收藏 分享 切换为英文 接收动态 反

实现 pow(*x*, *n*)，即计算 x 的 n 次幂函数（即，x^n）。

示例 1：

```
输入：x = 2.00000, n = 10
输出：1024.00000
```

示例 2：

```
输入：x = 2.10000, n = 3
输出：9.26100
```

示例 3：

```
输入：x = 2.00000, n = -2
输出：0.25000
解释：2^-2 = 1/2^2 = 1/4 = 0.25
```



```python
class Solution:
    def myPow(self, x: float, n: int) -> float:
        def quickMul(N):
            if N == 0:
                return 1.0
            y = quickMul(N // 2)
            return y * y if N % 2 == 0 else y * y * x

        return quickMul(n) if n >= 0 else 1.0 / quickMul(-n)

    def myPow2(self, x: float, n: int) -> float:
        if x == 0.0:
            return 0.0
        res = 1.0

        if n < 0:
            x, n = 1.0 / x, -n

        while n:
            if n & 1:
                res *= x
            x *= x
            n >>= 1
        return res
```

78. 子集

难度 中等 1270 收藏 分享 切换为英文 接收动态 反

给你一个整数数组 `nums` ，数组中的元素 **互不相同** 。返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。你可以按 **任意顺序** 返回解集。

示例 1：

```
输入：nums = [1,2,3]
输出：[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

示例 2：

```
输入：nums = [0]
输出：[[],[0]]
```



```python
class Solution:
    def subsetsMOST(self, nums: List[int]) -> List[List[int]]:
        size = len(nums)
        if size == 0:
            return []
        res = []
        for i in range(size + 1):
            self._dfs(nums, i, 0, [], res)
        return res

    def _dfs(self, nums, depth, begin, path, res):
        if len(path) == depth:
            res.append(path[:])
            return

        for i in range(begin, len(nums)):
            path.append(nums[i])
            self._dfs(nums, depth, i + 1, path, res)
            path.pop()

    def subsets2(self, nums: List[int]) -> List[List[int]]: #
        size = len(nums)
        n = 1 << size
        res = []
        for i in range(n):
            cur = []
            for j in range(size):
                if i >> j & 1:
                    cur.append(nums[j])
            res.append(cur)
        return res
```

多抓鱼的仓库内有非常多种类的图书，这些书可以按照一定的规则被划分到不同的分类中（如社科、文学等）。
给你一个由 1 - 9 的数字组成的的二维网格，数字代表的是不同分类，相互联通的相同分类被称为做该分类的库区，请计算单个库区面积最大的分类是什么。 联通指的是水平或垂直方向为相同分类。
可以认为最大面积的库区仅有1个。
二维网格的长宽范围为 [1, 31)

示例1 输入输出示例仅供调试，后台判题数据一般不包含示例

输入 复制

```
[[1, 1, 1, 2], [2, 1, 2, 2], [3, 2, 3, 3]]
```

输出 复制

```
1
```

说明

```
1 1 1 2
2 1 2 2
3 2 3 3
库区分布如上所示
分类 1 只有一个库区，面积为 4；分类 2 有三个库区，面积分别为 1，1，3；分类 3 有两个库区，面积分别为 1，2；因此面积最大的库区面积是 4，对应的分类是 1，结果应该输出 1
```



```python
class Solution:
    def categoryOfMaxWarehouseArea(self , grid):
        if not grid or not grid[0]:
            return
        res = [0] * 10
        m, n = len(grid), len(grid[0])
        def dfs(grid, i, j, tmp):
            if 0 <= i < m and 0 <= j < n and grid[i][j] > 0 and
                grid[i][j] = -1
                return 1 + dfs(grid, i + 1, j, tmp) + dfs(grid,
            return 0


        for i in range(m):
            for j in range(n):
                b = grid[i][j]
                a = dfs(grid, i, j, b)
                res[b] = max(res[b], a)

        m = max(res)
        return res.index(m)
```

面试题：求字符串是否合法

```python
class Solution:
    def find_ans(self, s):
        n = len(s)
        if n == 0:
            return 0
        self.res = 0
        path = []
        ans = []
        begin = 0
        self.dfs(s, n, begin, path, ans)
        return (self.res, ans)

    def dfs(self, s, n, begin, path, ans):
        if begin == n:
            self.res += 1
            ans.append(path[:])
            return
        for i in range(2): # 0, 1
            if begin + i >= n:
                break
            digit = self.digit_judge(s, begin, begin +
            if digit != -1:
                path.append(str(digit))
                self.dfs(s, n, begin + i + 1, path, ans
                path.pop()
```



```python
    def digit_judge(self, s, left, right):
        size = right - left + 1
        if size > 1 and s[left] == '0':
            return -1
        elif size == 1 and s[left] == '0':
            return -1
        num = int(s[left: right + 1])
        if num > 27:
            return -1
        return num
```

爬楼梯具体步骤

```python
@lru_cache(None)
def dfs(n, cnt, s):
    if cnt == n:
        res.append(s)
        return
    elif cnt > n:
        return

    for i in range(1, 3):
        dfs(n, cnt + i, s + '{}'.format(i))
```