第三部分

3.1.回溯算法

17. 电话号码的字母组合

```
# 17. 电话号码的字母组合
from typing import List

class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        alpha_dict = {"2":"abc", "3":"def", "4":"ghi", "5":"jkl",
"6":"mno","7":"pqrs","8":"tuv","9":"wxyz"}
        result_list = []
        if len(digits) == 0:
            return []
        for alpha in alpha_dict.get(digits[0]):
            self.letterHelp(alpha_dict, alpha, result_list, digits[1:])
        return result_list

    def letterHelp(self, alpha_dict: dict, cur_alpha: str, result_list, digits:
str):
        if len(digits) == 0:
            result_list.append(cur_alpha)
            return None
        for alpha in alpha_dict.get(digits[0]):
            self.letterHelp(alpha_dict, cur_alpha + alpha, result_list,
digits[1:])
```

22. 括号生成

```
# 22. 括号生成
from typing import List

class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        cur_list = [['(', 1, 0]]
        cur_list = self.tran(n, 1, cur_list)
        result = [n[0] for n in cur_list]
        return result

    def tran(self, n, i , cur_list):
        if i == 2 * n:
            return cur_list
        tmp = []
        for ele in cur_list:
            out, left, right = ele[0], ele[1], ele[2]
            if left == n:
```

```
                out += ')'
                right += 1
                tmp.append([out, left, right])
            elif left == right:
                out += "("
                left += 1
                tmp.append([out, left, right])
            elif left > right:
                tmp.append([out + "(", left + 1, right])
                tmp.append([out + ")", left, right+1])
        cur_list = tmp
        return self.tran(n , i+1, cur_list)
```

37. 解数独

```
from typing import List

# 37. 解数独
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        self.recur_solve(board)
        return board

    def recur_solve(self, board):
        status = False
        for i in range(9):
            for j in range(9):
                if board[i][j] == '.':
                    row_idx_list = self.get_row_list(i, j)
                    col_idx_list = self.get_col_list(i, j)
                    row_col_list = self.get_row_col_list(i, j)
                    remain_n_list = self.get_remain_n(row_idx_list + col_idx_list
+ row_col_list, board)
                    # 有符合的数字
                    status = False
                    for n in remain_n_list:
                        board[i][j] = n
                        next_status = self.recur_solve(board)
                        if next_status is False:
                            continue
                        else:
                            status = True
                            break
                    # 没有符合的数字
                    if status is False:
                        board[i][j] = "."
                        return status
        return True
```

```
    def get_remain_n(self, idx_list, board):
        n_list = [str(i) for i in range(1, 10)]
        for row_col in idx_list:
            i, j = row_col[0], row_col[1]
            if board[i][j]  != "." and board[i][j] in n_list:
                n_list.remove(board[i][j])
        return n_list

    def get_row_list(self, i, j):
        tmp = []
        for n in range(9):
            if n != j:
                tmp.append([i, n])
        return tmp

    def get_col_list(self, i, j):
        tmp = []
        for n in range(9):
            if n != i:
                tmp.append([n, j])
        return tmp

    def get_row_col_list(self, i, j):
        row_list, col_list = None, None
        if i < 3:
            row_list = [0, 3]
        elif i >= 3 and i < 6:
            row_list = [3, 6]
        elif i >= 6:
            row_list = [6, 9]

        if j < 3:
            col_list = [0, 3]
        elif j >= 3 and j < 6:
            col_list = [3, 6]
        elif j >= 6:
            col_list = [6, 9]

        row_col_list = []
        for row in range(row_list[0], row_list[1]):
            for col in range(col_list[0], col_list[1]):
                if i == row and j == col:
                    continue
                row_col_list.append([row, col])
        return row_col_list
```

39. 组合总和

```
# 39. 组合总和
from typing import List
```

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
        result_list = []
        pre_list = []
        self.getSum(candidates, result_list, target, pre_list)
        return result_list


    def getSum(self, candidates, result_list, target, pre_list):
        if target == 0:
            result_list.append(pre_list):
            return

        for n in candidates:
            if target - n >= 0:
                tmp_list = pre_list[::]
                tmp_list.append(n)
                self.getSum(cand_dict, result_list, target-n, pre_list)
        return
```

46. 全排列

```
# 46. 全排列
from typing import List

class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        result, pre_list = [], []
        self.permute_help(nums, result, pre_list)
        return result

    def permute_help(self, nums, result, pre_list):
        if len(nums) == 0:
            result.append(pre_list)
            return
        for n in nums:
            tmp = pre_list[::]
            tmp.append(n)
            tmp_nums = nums[::]
            tmp_nums.remove(n)
            self.permute_help(tmp_nums, result, tmp)
        return
```

77. 组合

```
#   77. 组合
from typing import List
```

```python
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        result, pre_list = [], []
        self.combine_help(1, n+1, k, result, pre_list)
        return result

    def combine_help(self, i,j, k, result, pre_list):
        if k == 0:
            result.append(pre_list)
            return
        for n in range(i, j):
            tmp_list = pre_list[::]
            tmp_list.append(n)
            self.combine_help(n+1, j, k-1 ,  result, tmp_list)
        return
```

78. 子集

```python
# 78. 子集
from typing import List

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        result = []
        self.sub_list(nums, result, [])
        return result

    def sub_list(self, nums, result, pre_list):
        if len(nums) == 0:
            result.append(pre_list[::])
            return
        result.append(pre_list[::])
        for i, n in enumerate(nums):
            tmp_list = pre_list[::]
            tmp_list.append(n)
            tmp_nums = nums[i+1:]
            self.sub_list(tmp_nums, result, tmp_list)
```

51. N 皇后

```python
#  51. N 皇后
from typing import List

class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        self.res = []
        board = [['.' for _ in range(n)] for _ in range(n)]
        self.backtrack(board, 0)
```

```
        return self.res

    def backtrack(self, board, row):
        if row == len(board):
            tmp_list = ["".join(ns) for ns in board]
            self.res.append(tmp_list)
            return
        n = len(board[row])
        for col in range(n):
            if not self.isValid(board, row, col):
                continue
            board[row][col] = 'Q'
            self.backtrack(board, row + 1)
            board[row][col] = '.'

    def isValid(self, board, row, col):
        n = len(board)
        # 列检查
        for i in range(n):
            if board[i][col] == 'Q':
                return False
        # 右上方
        i, j = row -1 , col + 1
        while i >= 0 and j < n:
            if board[i][j] == 'Q':
                return False
            i -= 1
            j += 1
        # 左上方
        i , j = row-1, col-1
        while i >= 0 and j >= 0:
            if board[i][j] == 'Q':
                return False
            i -= 1
            j -= 1
        return True
```

104.二叉树最大深度

```
# Definition for a binary tree node.
from typing import Optional

# 104.二叉树最大深度
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        return self.getDepth(root)
```

```
    def getDepth(self, root):
        if root is None:
            return 0
        return max(self.getDepth(root.left), self.getDepth(root.right)) + 1
```

494. 目标和

```
# 494. 目标和
from typing import List

class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        result = 0
        result = self.trac(nums, 0, result, 0, len(nums) , target)
        return result

    def trac(self, nums, base, result, i, n, target):
        if i == n:
            if base == target:
                result += 1
            return result
        result = self.trac(nums, base + nums[i], result, i+1, n, target)
        result = self.trac(nums, base - nums[i], result, i+1, n, target)
        return result

    def calc(self, n_list):
        base = 0
        for i in range(int(len(n_list)/2)):
            symbol, val = n_list[i*2], n_list[i*2 + 1]
            if symbol == '+':
                base += int(val)
            else:
                base -= int(val)
        return base

class Solution1:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        if len(nums) == 0:
            return 0
        self.memo = dict()
        return self.dp(nums, 0, target)

    def dp(self, nums, i, rest):
        if len(nums) == i:
            if rest == 0:
                return 1
            return 0
        key = str(i) + "," + str(rest)
        # 自底向上的过程中这个剪枝很关键
        if self.memo.get(key) is not None:
```

```
            return self.memo.get(key)
        result = self.dp(nums, i + 1, rest - nums[i]) + self.dp(nums, i+1, rest +
nums[i])
        self.memo[key] = result
        return result
```

698. 划分为k个相等的子集

```
#698. 划分为k个相等的子集
# leetcode 暴力搜索会超时
from typing import List

class Solution:
    def canPartitionKSubsets(self, nums: List[int], k: int) -> bool:
        if k > len(nums):
            return False
        sum_val = sum(nums)
        if sum_val % k != 0:
            return False
        used_list = [False for _ in nums]
        target = sum_val/k
        return self.backtrack(k, 0, nums, 0, used_list, target)

    def backtrack(self, k, bucket, nums, start, used_list, target):
        status = False
        if k == 0:
            status = True
            return status
        if bucket == target:
            return self.backtrack(k -1, 0, nums, 0, used_list, target)
        for i in range(start, len(nums)):
            if used_list[i]:
                continue
            if nums[i] + bucket > target:
                continue

            used_list[i] = True
            bucket += nums[i]
            if self.backtrack(k, bucket, nums, i+1, used_list, target):
                status = True
                return status
            used_list[i] = False
            bucket -= nums[i]
        return False
```

3.2.DFS算法

130. 被围绕的区域

```python
# 130. 被围绕的区域
from typing import List

class Solution:
    def solve(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        if board is None or len(board) == 0:
            return
        row, col = len(board), len(board[0])
        for i in range(row):
            self.dfs(board, i, 0)
            self.dfs(board, i, col - 1)
        for j in range(col):
            self.dfs(board, 0, j)
            self.dfs(board, row-1, j)
        for i in range(row):
            for j in range(col):
                if board[i][j] == 'O':
                    board[i][j] = 'X'
                if board[i][j] == '-':
                    board[i][j] = 'O'

    def dfs(self, board, i, j):
        if (i< 0 or j < 0 or i >= len(board) or j >= len(board[0]) or board[i][j]
!= "O"):
            return
        board[i][j] = '-'
        self.dfs(board, i-1, j)
        self.dfs(board, i+1, j)
        self.dfs(board, i, j-1)
        self.dfs(board, i, j+1)
        return
```

200. 岛屿数量

```python
# 200. 岛屿数量
from typing import List

class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        row, col = len(grid), len(grid[0])
        counter = 0
        for i in range(row):
            for j in range(col):
                if grid[i][j] == '1':
                    counter += 1
                    self.dfs(grid, i, j, row, col)
                    print(i, j, grid)
```

```
        return counter

    def dfs(self, grid, i, j, row, col):
        if grid[i][j] == '1':
            grid[i][j] = '0'
        else:
            return
        if i + 1 < row:
            self.dfs(grid, i+1, j, row, col)
        if i -1 >= 0:
            self.dfs(grid, i-1, j, row, col)
        if j + 1 < col:
            self.dfs(grid, i, j+1, row, col)
        if j - 1 >= 0:
            self.dfs(grid, i, j-1, row, col)
        return
```

694.不同的岛屿数量

```
# 694.不同的岛屿数量
class Solution:
    def numDistinctIslands(self, grid):
        islands_dict = dict()
        row, col = len(grid), len(grid[0])
        for i in range(row):
            for j in range(col):
                if grid[i][j] == "1":
                    res = []
                    self.dfs(grid, i, j, res, 666, row, col)
                    islands_dict[",".join(res)] = 1
        return len(islands_dict)

    def dfs(self, grid, i, j, res, dir, row, col):
        if grid[i][j] == '1':
            grid[i][j] = '0'
            res.append(str(dir))
        else:
            return
        if i + 1 < row:
            self.dfs(grid, i+1, j, res, 1, row, col)
        if i - 1 >= 0:
            self.dfs(grid, i-1, j, res, 2, row, col)
        if j + 1 < col:
            self.dfs(grid, i, j+1, res, 3, row, col)
        if j - 1 >= 0:
            self.dfs(grid, i, j-1, res, 4, row, col)
        return
```

695. 岛屿的最大面积

```python
# 695. 岛屿的最大面积
from typing import List


class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        row, col = len(grid), len(grid[0])
        area_list = []
        for i in range(row):
            for j in range(col):
                if grid[i][j] == 1:
                    area = self.dfs(grid, i, j, row, col, 0)
                    area_list.append(area)
        return max(area_list)

    def dfs(self, grid, i, j, row, col, area):
        if grid[i][j] == 1:
            grid[i][j] = 0
            area += 1
        else:
            return area
        if i + 1 < row:
            area = self.dfs(grid, i+1, j, row, col, area)
        if i - 1 >= 0:
            area = self.dfs(grid, i-1, j, row, col, area)
        if j + 1 < col:
            area = self.dfs(grid, i, j+1, row, col, area)
        if j - 1 >= 0:
            area = self.dfs(grid, i, j-1, row, col, area)
        return area
```

1020.飞地的数量

```python
# 1020.飞地的数量
from typing import List


class Solution:
    def numEnclaves(self, grid: List[List[int]]) -> int:
        row, col = len(grid), len(grid[0])
        for j in range(col):
            if grid[0][j] == 1:
                self.dfs(grid, 0, j, row, col)
            if grid[row-1][j] == 1:
                self.dfs(grid, row-1, j, row, col)

        for i in range(row):
            if grid[i][0] == 1:
                self.dfs(grid, i, 0, row, col)
            if grid[i][col-1] == 1:
```

```
                self.dfs(grid, i, col-1, row, col)

        counter = 0
        for i in range(row):
            for j in range(col):
                if grid[i][j] == 1:
                    counter += 1
        return counter

    def dfs(self, grid, i, j, row, col):
        if grid[i][j] == 1:
            grid[i][j] = 0
        else:
            return
        if i + 1 < row:
            self.dfs(grid, i + 1, j, row, col)
        if i - 1 >= 0:
            self.dfs(grid, i-1, j, row, col)
        if j + 1 < col:
            self.dfs(grid, i, j+1, row, col)
        if j - 1 >= 0:
            self.dfs(grid, i, j-1, row, col)
        return
```

1254. 统计封闭岛屿的数目

```
# 1254. 统计封闭岛屿的数目
from typing import List

class Solution:
    def closedIsland(self, grid: List[List[int]]) -> int:
        row, col = len(grid), len(grid[0])
        for j in range(col):
            if grid[0][j] == 0:
                self.dfs(grid, 0, j, row, col)
            if grid[row-1][j] == 0:
                self.dfs(grid, row-1, j, row, col)

        for i in range(row):
            if grid[i][0] ==  0:
                self.dfs(grid, i, 0, row, col)
            if grid[i][col-1] == 0:
                self.dfs(grid, i, col-1, row, col)

        counter = 0
        for i in range(1, row-1):
            for j in range(1, col-1):
                if grid[i][j] == 0:
                    self.dfs(grid, i, j, row, col)
                    counter += 1
        return counter
```

```
    def dfs(self, grid, i, j, row, col):
        if grid[i][j] == 0:
            grid[i][j] = 1
        else:
            return
        if i + 1 < row:
            self.dfs(grid, i+1, j, row, col)
        if i - 1 >= 0:
            self.dfs(grid, i-1, j, row, col)
        if j + 1 < col:
            self.dfs(grid, i, j+1, row, col)
        if j - 1 >= 0:
            self.dfs(grid, i, j-1, row, col)
        return
```

1905. 统计子岛屿

```
# 1905. 统计子岛屿
from typing import List

class Solution:
    def countSubIslands(self, grid1: List[List[int]], grid2: List[List[int]]) ->
int:
        row, col = len(grid1), len(grid1[0])
        counter = 0
        for i in range(row):
            for j in range(col):
                if grid2[i][j] == 1:
                    status = self.findIsland(grid1, grid2,i, j, row, col, True)
                    print("mark1")
                    if status:
                        counter += 1
        return counter

    def findIsland(self, grid1, grid2, i, j, row, col, status):
        if grid2[i][j] == 1:
            grid2[i][j] = 0
            if grid1[i][j] == 0:
                # print("grid1:", i, j)
                status = status and False
        else:
            return status

        if i + 1 < row:
            status = self.findIsland(grid1, grid2, i+1, j, row, col, status)
        if i -1 >= 0:
            status = self.findIsland(grid1, grid2, i-1, j, row, col, status)
        if j + 1 < col:
            status = self.findIsland(grid1, grid2, i, j+1, row, col, status)
        if j - 1 >= 0:
```

```
            status = self.findIsland(grid1, grid2, i, j-1, row, col, status)
        return status
```

3.3.BFS算法

102. 二叉树的层序遍历

```
# Definition for a binary tree node.
from typing import List

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# 102. 二叉树的层序遍历
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if root is None:
            return []
        node_list = [root]
        result = []
        self.recur_line(node_list, result)
        return result

    def recur_line(self, node_list, result):
        if len(node_list) == 0:
            return
        tmp_list = []
        lines = []
        for node in node_list:
            if node.left is not None:
                tmp_list.append(node.left)
            if node.right is not None:
                tmp_list.append(node.right)
            lines.append(node.val)
        result.append(lines)
        self.recur_line(tmp_list, result)
```

103. 二叉树的锯齿形层序遍历

```
# Definition for a binary tree node.
# 103. 二叉树的锯齿形层序遍历
from typing import List


class TreeNode:
    def __init__(self, val=0, left=None, right=None):
```

```
        self.val = val
        self.left = left
        self.right = right
class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if root is None:
            return []

        node_list = [root]
        result = []
        self.recur_node(node_list, result, 1)
        return result

    def recur_node(self, node_list, result, depth):
        if len(node_list) == 0:
            return
        sub_nodes = []
        lines = []
        for node in node_list:
            if node.left is not None:
                sub_nodes.append(node.left)
            if node.right is not None:
                sub_nodes.append(node.right)
            lines.append(node.val)
        if depth % 2 == 0:
            lines = lines[::-1]
        result.append(lines)
        self.recur_node(sub_nodes, result, depth+1)
```

107. 二叉树的层序遍历 II

```
# Definition for a binary tree node.
from typing import List

# 107. 二叉树的层序遍历 II
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
        if root is None:
            return []
        node_list = [root]
        result = []
        self.recur_lines(node_list, result)
        return result

    def recur_lines(self, node_list, res):
```

```
        if len(node_list) == 0:
            return
        new_nodes = []
        tmp = []
        for node in node_list:
            if node.left is not None:
                new_nodes.append(node.left)
            if node.right is not None:
                new_nodes.append(node.right)
            tmp.append(node.val)
        self.recur_lines(new_nodes, res)
        res.append(tmp)
        return
```

111. 二叉树的最小深度

```
# Definition for a binary tree node.
# 111. 二叉树的最小深度
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def minDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        return self.getDepth(root)

    def getDepth(self, root):
        if root.left is None and root.right is None:
            return 1
        elif root.left is None and root.right is not None:
            return self.getDepth(root.right) + 1
        elif root.left is not None and root.right is None:
            return self.getDepth(root.left) + 1
        elif root.left is not None and root.right is not None:
            return min(self.getDepth(root.left), self.getDepth(root.right)) + 1
```

752. 打开转盘锁

```
# 752. 打开转盘锁
from typing import List

class Solution:
    def openLock(self, deadends: List[str], target: str) -> int:
        dead_dict = {d:1 for d in deadends}
        visited_dict = dict()
```

```
        q = []
        step = 0
        q.append(['0', '0', '0', '0'])
        visited_dict["0000"] = 1
        while len(q) > 0:
            sz = len(q)
            for i in range(sz):
                cur = q.pop(0)
                cur_str = ''.join(cur)
                if dead_dict.get(cur_str) is not None:
                    continue
                if cur_str == target:
                    return step

                for j in range(4):
                    up = self.plusOne(cur[::], j)
                    up_str = ''.join(up)
                    if visited_dict.get(up_str) is None:
                        q.append(up)
                        visited_dict[up_str] = 1

                    down = self.minusOne(cur[::], j)
                    down_str = ''.join(down)
                    if visited_dict.get(down_str) is None:
                        q.append(down)
                        visited_dict[down_str] = 1
            step += 1
        return -1

    def plusOne(self, s, j):
        if s[j] == '9':
            s[j] = '0'
        else:
            s[j] = str(int(s[j]) + 1)
        return s

    def minusOne(self, s, j):
        if s[j] == '0':
            s[j] = '9'
        else:
            s[j] = str(int(s[j]) -1)
        return s
```

773. 滑动谜题

```
# 773. 滑动谜题
from typing import List

class Solution:
    def slidingPuzzle(self, board: List[List[int]]) -> int:
        row, col = len(board), len(board[0])
```

```
        start_row, start_col = None, None
        for i in range(row):
            for j in range(col):
                if board[i][j] == 0:
                    start_row, start_col = i, j

        q_list = [[[start_row, start_col, board]]]
        board_dict = dict()

        counter = 0
        while len(q_list) > 0:
            cur_list =  q_list.pop(0)
            tmp_list = []
            for cur in cur_list:
                cur_x, cur_y, cur_board = cur[0], cur[1], cur[2]
                if self.checkBoard(cur_board):
                    return counter
                if cur_x + 1 < row:
                    tmp_board = [b[::] for b in cur_board]
                    tmp_board[cur_x][cur_y], tmp_board[cur_x+1][cur_y] =
tmp_board[cur_x+1][cur_y], tmp_board[cur_x][cur_y]
                    tmp_board_str = self.castStr(tmp_board)
                    if board_dict.get(tmp_board_str) is None:
                        tmp_list.append([cur_x+1, cur_y, tmp_board])
                        board_dict[tmp_board_str] = 1
                if cur_x - 1 >= 0:
                    tmp_board = [b[::] for b in cur_board]
                    tmp_board[cur_x-1][cur_y], tmp_board[cur_x][cur_y] =
tmp_board[cur_x][cur_y], tmp_board[cur_x-1][cur_y]
                    tmp_board_str = self.castStr(tmp_board)
                    if board_dict.get(tmp_board_str) is None:
                        tmp_list.append([cur_x-1, cur_y, tmp_board])
                        board_dict[tmp_board_str] = 1
                if cur_y + 1 < col:
                    tmp_board = [b[::] for b in cur_board]
                    tmp_board[cur_x][cur_y], tmp_board[cur_x][cur_y+1] =
tmp_board[cur_x][cur_y+1], tmp_board[cur_x][cur_y]
                    tmp_board_str = self.castStr(tmp_board)
                    if board_dict.get(tmp_board_str) is None:
                        tmp_list.append([cur_x, cur_y+1, tmp_board])
                        board_dict[tmp_board_str] = 1
                if cur_y - 1 >= 0:
                    tmp_board = [b[::] for b in cur_board]
                    tmp_board[cur_x][cur_y], tmp_board[cur_x][cur_y-1] =
tmp_board[cur_x][cur_y-1], tmp_board[cur_x][cur_y]
                    tmp_board_str = self.castStr(tmp_board)
                    if board_dict.get(tmp_board_str) is None:
                        tmp_list.append([cur_x, cur_y-1, tmp_board])
                        board_dict[tmp_board_str] = 1
            if len(tmp_list) > 0:
                q_list.append(tmp_list)
                counter += 1
        return -1
```

```
    def checkBoard(self, board):
        status = True
        for n1, n2 in zip(board[0], [1, 2, 3]):
            if n1 != n2:
                status = False
        for n1, n2 in zip(board[1],[4, 5, 0]):
            if n1 != n2:
                status = False
        return status

    def castStr(self, board):
        n_list = []
        for n in board:
            n_list.append("".join([str(i) for i in n]))
        return "".join(n_list)
```

3.4.一维DP

45. 跳跃游戏 II

```
# 45. 跳跃游戏 II
from typing import List

class Solution:
    def jump(self, nums: List[int]) -> int:
        n = len(nums)
        end, farthest = 0, 0
        jump_count = 0
        for i in range(n-1):
            farthest = max(nums[i]+i, farthest)
            if end == i:
                jump_count += 1
                end = farthest
        return jump_count
```

55. 跳跃游戏

```
# 55. 跳跃游戏
from typing import List

class Solution:
    def canJump(self, nums: List[int]) -> bool:
        farthest = 0
        end = 0
        for i in range(len(nums)-1):
            farthest = max(nums[i] + i, farthest)
            # 碰上0值就直接返回为False
            if farthest <= i:
                return False
```

```
            if end == i:
                end = farthest
        return farthest >= len(nums) - 1
```

53. 最大子数组和

```
# 53. 最大子数组和
import sys
from typing import List
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 0:
            return 0
        dp = [0 for _ in range(n)]
        dp[0] = nums[0]
        for i in range(1, n):
            dp[i] = max(nums[i], nums[i] + dp[i-1])
        return max(dp)
```

70. 爬楼梯

```
# 70. 爬楼梯
class Solution:
    def climbStairs(self, n: int) -> int:
        if n <= 2:
            return n
        dp = [0 for _ in range(n+1)]
        dp[1] = 1
        dp[2] = 2
        for i in range(3, n+1):
            dp[i] = dp[i-1] + dp[i-2]
        return dp[n]

# 会超时
class Solution1:
    def climbStairs(self, n: int) -> int:
        return self.climbStairs(n - 1) + self.climbStairs(n-2) if n > 2 else n
```

198. 打家劫舍

```
# 198. 打家劫舍
from typing import List

class Solution:
    def rob(self, nums: List[int]) -> int:
```

```
        dp = [ 0 for _ in range(len(nums))]
        if len(nums) == 0:
            return 0
        if len(nums) == 1:
            return nums[0]
        dp[0] = nums[0]
        dp[1] = max(dp[0], nums[1])
        if len(nums) == 1:
            return dp[0]
        if len(nums) == 2:
            return dp[1]
        for i in range(2, len(nums)):
            dp[i] = max(nums[i] + dp[i-2], dp[i-1])
        return max(dp)
```

213. 打家劫舍 II

```
# 213. 打家劫舍 II
from typing import List

class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]
        val_1 = self.dp(nums, 1, len(nums)-1)
        val_2 = self.dp(nums, 0, len(nums)-2)
        return max(val_1, val_2)

    def dp(self, nums, start, end):
        if end-start == 0:
            return nums[start]
        dp = [0 for _ in range(len(nums))]
        dp[start] = nums[start]
        dp[start+1] = max(nums[start+1], dp[start])
        for i in range(start+2, end+1):
            dp[i] = max(nums[i] + dp[i-2], dp[i-1])
        return max(dp)

class Solution:
    def rob(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 1:
            return nums[0]
        memo1 = [-1 for _ in range(n)]
        memo2 = [-1 for _ in range(n)]
        return max(self.dp(nums, 0, n-2, memo1), self.dp(nums, 1, n-1, memo2))

    def dp(self, nums, start, end, memo):
        if start > end:
            return 0
        if memo[start] != -1:
```

```
            return memo[start]
        res = max(self.dp(nums, start+2, end, memo) + nums[start], self.dp(nums,
start+1, end, memo))
        memo[start] = res
        return res
```

337. 打家劫舍 III

```
# Definition for a binary tree node.
# 337. 打家劫舍 III
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def rob(self, root: TreeNode) -> int:
        self.memo = dict()
        res = self.robHelp(root)
        return res

    def robHelp(self, root):
        if root is None:
            return 0
        if self.memo.get(root) is not None:
            return self.memo.get(root)
        do_it = root.val
        if root.left is not None:
            do_it += self.robHelp(root.left.left) + self.robHelp(root.left.right)
        if root.right is not None:
            do_it += self.robHelp(root.right.left) +
self.robHelp(root.right.right)
        not_do = self.robHelp(root.left) + self.robHelp(root.right)
        res = max(do_it, not_do)
        self.memo[root] = res
        return res
```

300. 最长递增子序列

```
# 300. 最长递增子序列
from typing import List

class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        dp = [1 for _ in nums]
        for i in range(1, len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
```

```
                    dp[i] = max(dp[i], dp[j]+1)
        return max(dp)

class Solution1:
    def lengthOfLIS(self, nums):
        maxL = 0
        # 存放当前的递增序列的潜在数据
        dp = [0 for _ in nums]
        for num in nums:
            lo, hi = 0, maxL
            # 二分查找，并替换，这一步维护dp的本质是维护一个潜在的递增序列。非常trick
            while lo < hi:
                mid = lo + (hi-lo)/2
                if dp[mid] < num:
                    lo = mid + 1
                else:
                    hi = mid
            dp[lo] = num
            # 若是接在最后面，则连续递增序列张长了
            if lo == maxL:
                maxL += 1
        return maxL
```

322. 零钱兑换

```
# 322. 零钱兑换
import sys
from typing import List
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        if amount == 0:
            return 0
        dp = [sys.maxsize for _ in range(amount+1)]
        for i in range(1, len(dp)):
            for coin in coins:
                if i == coin:
                    dp[i] = 1
                else:
                    if i > coin:
                        dp[i] = min(dp[i-coin] + 1, dp[i])
        return -1 if dp[-1] == sys.maxsize else dp[-1]
```

354. 俄罗斯套娃信封问题

```
# 354. 俄罗斯套娃信封问题
from functools import cmp_to_key
from typing import List
class Solution:
    def maxEnvelopes(self, envelopes: List[List[int]]) -> int:
```

```
        def sort_item(item1, item2):
            if item1[0] == item2[0]:
                return item2[1] - item1[1]
            else:
                return item1[0] - item2[0]
        envelopes_list = sorted(envelopes, key=cmp_to_key(sort_item))
        height_list = [item[1] for item in envelopes_list]
        # O(N*2)的方式记录dp会超时，但是这个方式确实太fancy了
        return self.lengthOfLIS(height_list)

    def lengthOfLIS(self, nums):
        piles, n = 0, len(nums)
        top = [0 for _ in range(n)]
        for i in range(n):
            poker = nums[i]
            left, right = 0, piles
            while left < right:
                mid = int((left + right)/2)
                if top[mid] >= poker:
                    right = mid
                else:
                    left = mid + 1
            if left == piles:
                piles += 1
            top[left] = poker
        return piles
```

3.5.二维DP

10. 正则表达式匹配

```
# 10. 正则表达式匹配
class Solution:
    def isMatch(self, s, p):
        self.memo = dict()
        return self.dp(s, p, 0, 0)

    def dp(self, s, p, i, j):
        m , n = len(s), len(p)
        if j == n:
            return i == m
        if i == m:
            if (n -j) % 2 == 1:
                return False
            while j+1 < n:
                if p[j+1] != "*":
                    return False
                j += 2
            return True

        key = str(i) + "," + str(j)
```

```
        if self.memo.get(key) is not None:
            return self.memo.get(key)
        res = False
        if s[i] == p[j] or p[j] == '.':
            if j < n - 1 and p[j+1] == '*':
                res = self.dp(s, p, i, j+2) or self.dp(s, p, i+1, j)
            else:
                res = self.dp(s, p, i+1, j+1)
        else:
            if j < n - 1 and p[j+1] == '*':
                res = self.dp(s, p, i, j+2)
            else:
                res = False
        self.memo[key] = res
        return res
```

62. 不同路径

```
# 62. 不同路径
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[0 for _ in range(n)] for _ in range(m)]
        for j in range(n):
            dp[0][j] = 1
        for i in range(m):
            dp[i][0] = 1
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1]
        return dp[m-1][n-1]
```

64. 最小路径和

```
# 64. 最小路径和
from typing import List

class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        row, col = len(grid), len(grid[0])
        dp = [[0 for _ in range(col)] for _ in range(row)]
        for j in range(col):
            if j - 1 >= 0:
                dp[0][j] =  grid[0][j] + dp[0][j-1]
            else:
                dp[0][j] =  grid[0][j]

        for i in range(row):
            if i - 1 >= 0:
                dp[i][0] = grid[i][0] + dp[i-1][0]
```

```
            else:
                dp[i][0] = grid[i][0]

        for i in range(1, row):
            for j in range(1, col):
                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
        return dp[row-1][col-1]
```

72.编辑距离

```
# 72.编辑距离
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        m , n = len(word1), len(word2)
        # m * n 的数组保存的是，对应的word1[:i] 字符串和 对应的word2[:j]字符串的编辑距离
        dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
        for i in range(1, m+1):
            dp[i][0] = i
        for j in range(1, n+1):
            dp[0][j] = j
        for i in range(1, m+1):
            for j in range(1, n+1):
                if word1[i-1] == word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    # i-1, j 删除; i,j-1 插入； i-1,j-1 替换
                    # i-1, j 删除，是指：word1[:i]删除 对应的i的字符，所以操作+1 并取上一步的编辑距离
                    # i,j-1 插入是指，插入当前word1[i]的位置字符为对应Word2[j] 所以操作+1,插入之前的word1的字符串保持不变，
                    # 相当于插入到之前字符串的i+1的位置，因此取上一步的编辑距离，i,j-1
                    # i-1, j-1 替换操作，则+1，然后直接取上一步的编辑距离。
                    dp[i][j] = min(dp[i-1][j]+1, min(dp[i][j-1]+1, dp[i-1][j-1] + 1))
        return dp[m][n]
```

121. 买卖股票的最佳时机

```
# 121. 买卖股票的最佳时机
from typing import List

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        delta_list = []
        for p1, p2 in zip(prices[:-1], prices[1:]):
            delta_list.append(p2-p1)
        cum_val = 0
        max_val = 0
```

```
        for delta in delta_list:
            if cum_val + delta > 0:
                cum_val += delta
                max_val = max(cum_val, max_val)
            else:
                cum_val = 0
        return max_val

# DP的方式
class Solution1:
    def maxProfit(self, prices: List[int]) -> int:
        dp = [[0, 0] for _ in prices]
        # 卖出
        dp[0][0] = 0
        # 买入
        dp[0][1] = -prices[0]
        for i in range(1, len(prices)):
            # 卖出   =  max(保持 ,   前一天买入 + 当天卖出  )
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
            # 买入 =    max( 保持,  当天买入 )
            dp[i][1] = max(dp[i-1][1], -prices[i])
        return dp[-1][0]
```

122. 买卖股票的最佳时机 II

```
#122. 买卖股票的最佳时机 II
from typing import List

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        delta_list = []
        for p1, p2 in zip(prices[:-1], prices[1:]):
            delta_list.append(p2-p1)
        val_list = list(filter(lambda x: x> 0 , delta_list))
        return sum(val_list)

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        dp = [[0, 0] for _ in prices]
        dp[0][0] = 0
        dp[0][1] = -prices[0]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i])
        return dp[-1][0]
```

123. 买卖股票的最佳时机 III

```python
#123. 买卖股票的最佳时机 III
from typing import List

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        max_k = 2
        dp = [ [[0, 0] for _ in range(max_k+1)] for _ in prices]
        for k in range(max_k, 0, -1):
            dp[0][k][0] = 0
            dp[0][k][1] = -prices[0]

        for i in range(1, len(prices)):
            for k in range(max_k, 0, -1):
                dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
                dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])

        return dp[-1][max_k][0]
```

188. 买卖股票的最佳时机 IV

```python
# 188. 买卖股票的最佳时机 IV
from typing import List

class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        n = len(prices)
        if n <= 0:
            return 0

        if k > n/2:
            return self.profitNoLimit(prices)

        dp = [[[0, 0] for _ in range(k+1)] for _ in prices]
        for i in range(k, 0, -1):
            dp[0][i][0] = 0
            dp[0][i][1] = -prices[0]
        for i in range(1, len(prices)):
            for j in range(k, 0, -1):
                dp[i][j][0] = max(dp[i-1][j][0], dp[i-1][j][1] + prices[i])
                dp[i][j][1] = max(dp[i-1][j][1], dp[i-1][j-1][0] - prices[i])
        return dp[-1][k][0]

    def profitNoLimit(self, prices):
        dp = [[0, 0] for _ in prices]
        dp[0][0] = 0
        dp[0][1] = -prices[0]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i])
        return dp[-1][0]
```

309. 最佳买卖股票时机含冷冻期

```python
# 309. 最佳买卖股票时机含冷冻期
from typing import List

class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        dp = [[0, 0, 0] for _ in prices]
        dp[0][0] = 0
        dp[0][1] = -prices[0]
        dp[0][2] = 0

        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
            dp[i][1] = max(dp[i-1][1], dp[i-1][2] - prices[i])
            dp[i][2] = dp[i-1][0]
        return max(dp[-1][0], dp[-1][2])
```

714. 买卖股票的最佳时机含手续费

```python
# 714. 买卖股票的最佳时机含手续费
from typing import List

class Solution:
    def maxProfit(self, prices: List[int], fee: int) -> int:
        dp = [[0, 0] for _ in prices]
        dp[0][0] = 0
        dp[0][1] = -prices[0] - fee
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i] - fee)
        return dp[-1][0]
```

174. 地下城游戏

```python
# 174. 地下城游戏
from typing import List

import sys
class Solution1:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        m , n = len(dungeon), len(dungeon[0])
        self.memo = [[-1 for _ in range(n)] for _ in range(m)]
        return self.dp(dungeon, 0, 0)

    def dp(self, dungeon, i, j):
```

```
        m, n = len(dungeon), len(dungeon[0])
        if i == m-1 and j == n-1:
            return 1 if dungeon[i][j] >= 0 else 1 - dungeon[i][j]

        if i == m or j == n:
            return sys.maxsize

        if self.memo[i][j] != -1:
            return self.memo[i][j]

        res = min(self.dp(dungeon, i, j+1), self.dp(dungeon, i+1, j)) - dungeon[i]
[j]

        self.memo[i][j] = 1 if res <= 0 else res

        return self.memo[i][j]
```

312. 戳气球

```
# 312. 戳气球
from typing import List

class Solution:
    def maxCoins(self, nums: List[int]) -> int:
        n = len(nums)
        points = [0 for _ in range(n+2)]
        points[0] = 1
        points[n+1] = 1
        for i in range(1, n+1):
            points[i] = nums[i-1]
        dp = [[0 for _ in range(n+2)] for _ in range(n+2)]
        for i in range(n, -1, -1):
            for j in range(i+1, n+2):
                for k in range(i+1, j):
                    dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] + points[i] *
points[j] * points[k])
        return dp[0][n+1]
```

416.分割等和子集

```
from typing import List

# 416.分割等和子集
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        sum_val = sum(nums)
        if sum_val % 2 != 0:
            return False
        n = len(nums)
```

```
        sum_val = int(sum_val/2)
        dp = [[False for _ in range(sum_val + 1)] for _ in range(n+1)]
        for i in range(n+1):
            dp[i][0] = True

        for i in range(1, n+1):
            for j in range(1, sum_val+1):
                if j - nums[i-1] < 0:
                    dp[i][j] = dp[i-1][j]
                else:
                    dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i-1]]
        return dp[n][sum_val]
```

494. 目标和

```
# 494. 目标和
from typing import List

class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        result = 0
        result = self.trac(nums, 0, result, 0, len(nums) , target)
        return result

    def trac(self, nums, base, result, i, n, target):
        if i == n:
            if base == target:
                result += 1
            return result
        result = self.trac(nums, base + nums[i], result, i+1, n, target)
        result = self.trac(nums, base - nums[i], result, i+1, n, target)
        return result

    def calc(self, n_list):
        base = 0
        for i in range(int(len(n_list)/2)):
            symbol, val = n_list[i*2], n_list[i*2 + 1]
            if symbol == '+':
                base += int(val)
            else:
                base -= int(val)
        return base

class Solution1:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        if len(nums) == 0:
            return 0
        self.memo = dict()
        return self.dp(nums, 0, target)

    def dp(self, nums, i, rest):
```

```
        if len(nums) == i:
            if rest == 0:
                return 1
            return 0
        key = str(i) + "," + str(rest)
        # 自底向上的过程中这个剪枝很关键
        if self.memo.get(key) is not None:
            return self.memo.get(key)
        result = self.dp(nums, i + 1, rest - nums[i]) + self.dp(nums, i+1, rest +
nums[i])
        self.memo[key] = result
        return result
```

514.自由之路

```
# 514.自由之路
import sys

class Solution:
    def findRotateSteps(self, ring: str, key: str) -> int:
        m, n = len(ring), len(key)
        self.charToIndex = dict()
        self.memo = [[0 for _ in range(n)] for _ in range(m)]
        # 索引初始化
        for i in range(m):
            if self.charToIndex.get(ring[i]) is None:
                self.charToIndex[ring[i]] = [i]
            else:
                self.charToIndex[ring[i]].append(i)
        return self.dp(ring, 0, key, 0)

    def dp(self, ring, i, key, j):
        if j == len(key):
            return 0
        if self.memo[i][j] != 0:
            return self.memo[i][j]
        n = len(ring)
        res = sys.maxsize
        # 找到最小的字母
        for k in self.charToIndex.get(key[j]):
            delta = abs(k - i)
            # 正向和逆向都可以，找到最小的
            delta = min(delta, n - delta)
            subProblem = self.dp(ring, k, key, j+1)
            res = min(res, 1 + delta +subProblem)
        self.memo[i][j] = res
        return res

if __name__ == "__main__":
    rings = "godding"
    key = "godding"
```

```
    sol = Solution()
    ret = sol.findRotateSteps(rings, key)
    print(ret)
```

518. 零钱兑换 II

```
# 518. 零钱兑换 II
from typing import List

class Solution:
    def change(self, amount: int, coins: List[int]) -> int:
        dp = [[0 for _ in range(amount+1)] for _ in range(len(coins)+ 1)]
        for i in range(len(coins) + 1):
            dp[i][0] = 1

        for i in range(1, len(coins)+1):
            for j in range(1, amount + 1):
                coin = coins[i-1]
                if j - coin < 0:
                    dp[i][j] = dp[i-1][j]
                else:
                    dp[i][j] = dp[i-1][j] + dp[i][j-coin]
        return dp[-1][-1]
```

583. 两个字符串的删除操作

```
# 583. 两个字符串的删除操作
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        self.memo = [[-1 for _ in range(len(word2))] for _ in range(len(word1))]
        res = self.dp(word1, 0, word2, 0)
        return res

    def dp(self, word1, i, word2, j):
        if i == len(word1) and j == len(word2):
            print("mark1")
            return 0
        elif i == len(word1) and j < len(word2):
            return len(word2) - j
        elif i < len(word1) and j == len(word2):
            return len(word1) - i

        if self.memo[i][j] != -1:
            return self.memo[i][j]
        if word1[i] == word2[j]:
            self.memo[i][j] = self.dp(word1, i+1, word2, j+1)
        else:
            self.memo[i][j] = min(self.dp(word1, i+1, word2, j), self.dp(word1, i,
```

```
word2, j+1)) + 1

        return self.memo[i][j]

if __name__ == "__main__":
    word1 = "sea"
    word2 = "eat"
    sol = Solution()
    ret = sol.minDistance(word1, word2)
    print(ret)
```

712. 两个字符串的最小ASCII删除和

```
# 712. 两个字符串的最小ASCII删除和
class Solution:
    def minimumDeleteSum(self, s1: str, s2: str) -> int:
        self.memo = [[0 for _ in s2] for _ in s1]
        return self.dp(s1, 0, s2, 0)

    def dp(self, s1, i, s2, j):
        if i == len(s1) and j == len(s2):
            return 0
        elif i == len(s1) and j < len(s2):
            val = 0
            while j < len(s2):
                val += ord(s2[j])
                j += 1
            return val
        elif i < len(s1) and j == len(s2):
            val = 0
            while i < len(s1):
                val += ord(s1[i])
                i += 1
            return val

        if self.memo[i][j] != 0:
            return self.memo[i][j]

        if s1[i] == s2[j]:
            self.memo[i][j] = self.dp(s1, i+1, s2, j+1)
        else:
            left = ord(s1[i]) + self.dp(s1, i+1, s2, j)
            right = ord(s2[j]) + self.dp(s1, i, s2, j+1)
            self.memo[i][j] = min(left, right)
        return self.memo[i][j]
```

1143. 最长公共子序列

```python
# 1143. 最长公共子序列
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        self.memo =[[0 for _ in text2] for _ in text1]
        return self.dp(text1, 0, text2, 0)

    def dp(self, text1, i, text2, j):
        if i == len(text1) or j == len(text2):
            return 0
        if self.memo[i][j] != 0:
            return self.memo[i][j]
        if text1[i] == text2[j]:
            self.memo[i][j] = 1 + self.dp(text1, i+1, text2, j+1)
        else:
            self.memo[i][j] = max(self.dp(text1, i+1, text2, j), self.dp(text1, i,
text2, j+1))
        return self.memo[i][j]
```

787.K站中专内最便宜的航班

```python
# 787.K站中专内最便宜的航班
from typing import List
import sys

class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst:
int, k: int) -> int:
        edges = self.getEdges(flights, n)
        self.memo = [[-2 for _ in range(n)] for i in range(k+2)]
        return self.bfs(edges, src, dst, k+1)

    def bfs(self, edges, src, dst, k):
        if src == dst:
            return 0
        if k == 0:
            return -1

        if self.memo[k][src] != -2:
            return self.memo[k][src]
        res = sys.maxsize
        for edge in edges[src]:
            sub_fee = self.bfs(edges, edge[0], dst, k-1)
            if sub_fee != -1:
                res = min(sub_fee + edge[1], res)
        res = -1 if res == sys.maxsize else res
        self.memo[k][src] = res
        return res

    def getEdges(self, flights, n):
        edges = [[] for _ in range(n)]
```

```
        for f in flights:
            start, end,  fee = f[0], f[1], f[2]
            edges[start].append([end, fee])
        return edges
```

887. 鸡蛋掉落/810

```
import sys
sys.setrecursionlimit(100000) #例如这里设置为十万
# 887. 鸡蛋掉落/810

class Solution:
    def superEggDrop(self, k: int, n: int) -> int:
        dp = [[0 for _ in range(n+1)] for _ in range(k+1)]
        m = 0
        while dp[k][m] < n:
            m += 1
            for i in range(1, k+1):
                dp[i][m] = dp[i][m-1] + dp[i-1][m-1] + 1
        return m
```

931. 下降路径最小和

```
# 931. 下降路径最小和
import sys
from typing import List
class Solution:
    def minFallingPathSum(self, matrix: List[List[int]]) -> int:
        row ,col = len(matrix), len(matrix[0])
        self.memo = [[float("inf") for _ in range(col)] for _ in range(row)]
        res_list = []
        for j in range(col):
            res = self.minPath(matrix, 0, j, row, col)
            res_list.append(res)
        return min(res_list)

    def minPath(self, matrix, i, j, row, col):
        if i == row - 1:
            self.memo[i][j] = matrix[i][j]
            return matrix[i][j]

        if self.memo[i][j] != float("inf"):
            return self.memo[i][j]
        res = float("INF")
        res = min(res, self.minPath(matrix, i+1, j, row, col))
        if j - 1 >= 0:
            res = min(res, self.minPath(matrix, i+1, j-1, row, col))
        if j + 1 < col:
            print(j)
```

```
            res = min(res, self.minPath(matrix, i+1, j+1, row, col))
        res += matrix[i][j]
        self.memo[i][j] = res
        return res
```

3.6.背包问题

416.分割等和子集

```
from typing import List

# 416.分割等和子集
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        sum_val = sum(nums)
        if sum_val % 2 != 0:
            return False
        n = len(nums)
        sum_val = int(sum_val/2)
        dp = [[False for _ in range(sum_val + 1)] for _ in range(n+1)]
        for i in range(n+1):
            dp[i][0] = True

        for i in range(1, n+1):
            for j in range(1, sum_val+1):
                if j - nums[i-1] < 0:
                    dp[i][j] = dp[i-1][j]
                else:
                    dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i-1]]
        return dp[n][sum_val]
```

494. 目标和

```
# 494. 目标和
from typing import List

class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        result = 0
        result = self.trac(nums, 0, result, 0, len(nums) , target)
        return result

    def trac(self, nums, base, result, i, n, target):
        if i == n:
            if base == target:
                result += 1
            return result
        result = self.trac(nums, base + nums[i], result, i+1, n, target)
        result = self.trac(nums, base - nums[i], result, i+1, n, target)
```

```
        return result

    def calc(self, n_list):
        base = 0
        for i in range(int(len(n_list)/2)):
            symbol, val = n_list[i*2], n_list[i*2 + 1]
            if symbol == '+':
                base += int(val)
            else:
                base -= int(val)
        return base

class Solution1:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        if len(nums) == 0:
            return 0
        self.memo = dict()
        return self.dp(nums, 0, target)

    def dp(self, nums, i, rest):
        if len(nums) == i:
            if rest == 0:
                return 1
            return 0
        key = str(i) + "," + str(rest)
        # 自底向上的过程中这个剪枝很关键
        if self.memo.get(key) is not None:
            return self.memo.get(key)
        result = self.dp(nums, i + 1, rest - nums[i]) + self.dp(nums, i+1, rest +
nums[i])
        self.memo[key] = result
        return result
```

518. 零钱兑换 II

```
# 518. 零钱兑换 II
from typing import List

class Solution:
    def change(self, amount: int, coins: List[int]) -> int:
        dp = [[0 for _ in range(amount+1)] for _ in range(len(coins)+ 1)]
        for i in range(len(coins) + 1):
            dp[i][0] = 1

        for i in range(1, len(coins)+1):
            for j in range(1, amount + 1):
                coin = coins[i-1]
                if j - coin < 0:
                    dp[i][j] = dp[i-1][j]
                else:
```

```
                    dp[i][j] = dp[i-1][j] + dp[i][j-coin]
        return dp[-1][-1]
```