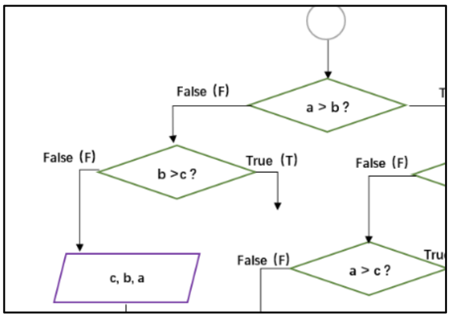
1. Flowchart

[10 points] Write a function Print_values with arguments a, b, and c to reflect the following flowchart. Here the purple parallelogram operator on a list [x, y, z] is to compute and print x+y-10z. Try your output with some random a, b, and c values. Report your output when a = 5, b = 15, c = 10.

Result:

当 a = 5, b = 15, c = 10，不执行运算，输出：none，运行结果如图：




```
test 1 a= 12 b= 8 c= 7
-50
test 2 a= 4 b= 6 c= 9
-25
test 3 a= 20 b= 14 c= 17
-103
a=5, b=15, c=10 reslut:
None
```

2. Continuous celing function

[10 points] Given a list with N positive integers. For every element x of the list, find the value of continuous ceiling function defined as F(x) = F(ceil(x/3)) + 2x, where F(1) = 1.

Result:

首先看到 ceil 考虑使用 math 模块，观察函数为递归，定义边界情况后实现即可（最简单思路，若提高效率可考虑使用哈希表等记忆函数值），运行结果如图：

```
1 1
2 5
3 7
4 13
5 15
6 17
7 21
8 23
9 25
10 33
```

3. Dice rolling

3.1 [15 points] Given 10 dice each with 6 faces, numbered from 1 to 6. Write a function Find_number_of_ways to find the number of ways to get sum x, defined as the sum of values on each face when all the dice are thrown.

Result：

来 自 于 Leecode 1155 题 (https://leetcode.com/problems/number-of-dice-rolls-with-target-sum/description/)，是原题的简版，即设定了 n=10、k=6，最合适算法无疑是 dynamic programming


题库

题目描述 | 题解 | 提交记录

1155. 掷骰子等于目标和的方法数

中等 相关标签 相关企业 提示 Aa

这里有 n 个一样的骰子，每个骰子上都有 k 个面，分别标号为 1 到 k 。

给定三个整数 n、k 和 target，请返回投掷骰子的所有可能得到的结果（共有 k^n 种方式），使得骰子面朝上的数字总和等于 target。

由于答案可能很大，你需要对 $10^9 + 7$ 取模。

示例 1:

输入：n = 1, k = 6, target = 3
输出：1
解释：你掷了一个有 6 个面的骰子。
得到总和为 3 的结果的方式只有一种。

示例 2:

输入：n = 2, k = 6, target = 7
输出：6
解释：你掷了两个骰子，每个骰子有 6 个面。
有 6 种方式得到总和为 7 的结果：1+6, 2+5, 3+4, 4+3, 5+2, 6+1。

这里尝试更容易理解的递归，即从后往前推当最后一个骰子的值为 1-6 时，前 9 个骰子应该为多少，递归解决剩余问题。基准情况：如果没有骰子(n=0)且目标为 0，说明成功找到一种方法，返回 1,如果没有骰子但目标不为 0，或者目标变成负数，说明这种方法不可行，返回 0。（递归思路：1.https://www.deepseek.com/zh）

```
定义递归函数：ways(n, target) 表示用 n 个骰子掷出总和 target 的方法数

基准情况：
1. 如果 n == 0 且 target == 0：返回 1（成功）
2. 如果 n == 0 且 target != 0：返回 0（失败）
3. 如果 target < 0：返回 0（不可能）

递归关系：
最后一个骰子可能掷出 1 到 6：
ways(n, target) = ways(n-1, target-1)  // 最后一个骰子为1
                + ways(n-1, target-2)  // 最后一个骰子为2
                + ways(n-1, target-3)  // 最后一个骰子为3
                + ways(n-1, target-4)  // 最后一个骰子为4
                + ways(n-1, target-5)  // 最后一个骰子为5
                + ways(n-1, target-6)  // 最后一个骰子为6
```

传统递归会出现重复计算的问题，大量情况被重复计算浪费大量时间，考虑使用记忆化 ，所有值只计算一次。（记忆存储思路：B 站 up 主 flyingchow123 ）

Python 中的 functools.lru_cache 装饰器，functools.lru_cache 是 Python 标准库中 functools 模块的一部分。lru_cache 可以用来为一个函数添加一个缓存系统。它可以帮助我们优化递归函数，避免重复计算已经计算过的值。（@lru_cache 装饰器理解：https://zhuanlan.zhihu.com/p/640954732），运行结果如图：

```
总和10的方法数: 1
总和15的方法数: 2002
总和30的方法数: 2930455
总和45的方法数: 831204
总和60的方法数: 1
```

3.2 [5 points] Count the number of ways for any x from 10 to 60, assign the number of ways to a list called Number_of_ways, so which x yields the maximum of Number_of_ways?

Result:

```
3.2 计算所有可能总和的方式数:
方式数列表长度: 51
产生最大方式数的x值: 35
最大方式数: 4395456
```

4. Dynamic programming

4.1 [5 points] Write a function Random_integer to fill an array of N elements by randomly selecting integers from 0 to 10.

Result:

使用 random 模块的 random.randint 生成随机整数，使用 append 添加到列表中。

```python
import random

# 4.1
def Random_integer(N):
    result = []
    for i in range(N):
        num = random.randint(0, 10)
        result.append(num)
    return result

arr = Random_integer(5)
print(arr)
```

```
[4, 5, 9, 5, 10]
```

4.2 [15 points] Write a function Sum_averages to compute the sum of the average of all subsets of the array. For example, given an array of [1, 2, 3], you Sum_averages function should compute the sum of: average of [1], average of [2], average of [3], average of [1, 2], average of [1, 3], average of [2, 3], and average of [1, 2, 3].

Result:

动态规划思路描述：（来源：1.https://chat.deepseek.com/ 2.）

(1)状态定义：

dp_sum[k]：存储所有长度为 k 的子集的和的总和

dp_count[k]：存储所有长度为 k 的子集的个数

(2)初始化：

dp_count[0] = 1 表示空集，这是动态规划的起点

(3)核心动态规划过程：

遍历数组中的每个数字，对于每个数字，从后往前更新状态（避免重复计算）

dp_sum[k] = dp_sum[k] + dp_sum[k-1] + num * dp_count[k-1]：

dp_sum[k]：保持已有的长度为 k 的子集和

dp_sum[k-1]：之前长度为 k-1 的子集和

num * dp_count[k-1]：当前数字添加到所有 k-1 长度子集中产生的新和

dp_count[k] = dp_count[k] + dp_count[k-1]：更新子集数量

(4)计算最终结果：

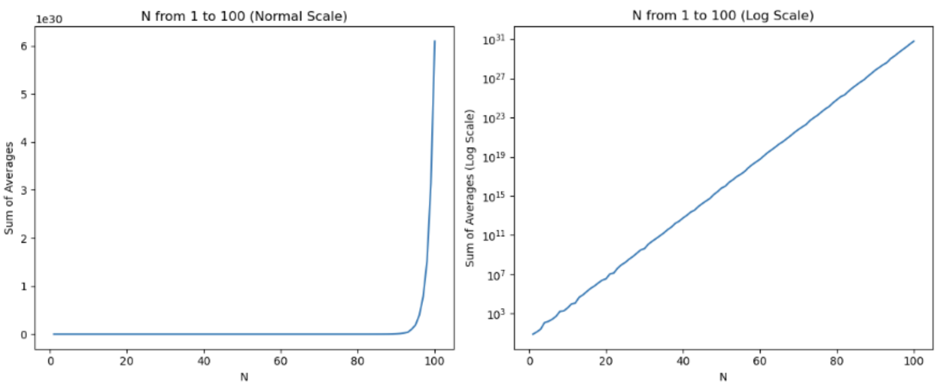
遍历所有可能的子集长度（1 到 n）对于每个长度 k，将 dp_sum[k] / k 加到总和中这样就得到了所有子集平均值的总和。

```
数组 [1, 2, 3] 的结果: 14.0
数组 [1, 2, 3, 4, 5] 的结果: 93.0
```

4.3 [5 points] Call Sum_averages with N increasing from 1 to 100, assign the output to a list called Total_sum_averages. Plot Total_sum_averages, describe what do you see.

Result:

从图中可以观察到，随着数组长度 N 的增大，所有子集平均值之和（Sum of Averages）呈现快速增长的指数趋势(左图为正常图，右图对 y 取了对数以便观察趋势)，子集数量的指数级增长主导了整个求和过程，即使单个子集的平均值很小，但数量庞大的子集使得总和呈现爆炸式增长。



5. Path counting

5.1 [5 points] Create a matrix with N rows and M columns, fill the right-bottom corner and top-left corner cells with 1, and randomly fill the rest of matrix with integer 0 or 1.

Result:

使用 matrix = [[0 for _ in range(M)] for _ in range(N)]的写法来代替传统写法：

[0 for _ in range(M)]：创建一个长度为 M 的列表，每个元素都是 0

for _ in range(M)：循环 M 次，_是约定表示不关心循环变量值的变量名

0：每次循环都放入数字 0

for _ in range(N)：外层循环 N 次

每次循环都创建一个新的长度为 M 的零列表

测试结果如下：

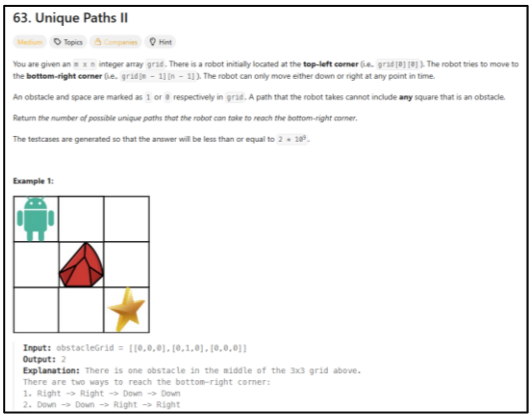
```
# 测试
matrix = create_matrix(3, 4)
for row in matrix:
    print(row)
```

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

5.2 [25 points] Consider a cell marked with 0 as a blockage or dead-end, and a cell marked with 1 is good to go. Write a function Count_path to count total number of paths to reach the right-bottom corner cell from the top-left corner cell.

Notice: for a given cell, you are **only allowed** to move either rightward or downward.

来自于 Leecode 63 题(https://leetcode.com/problems/unique-paths-ii/)，改动了有无障碍物的状态（0 代表障碍物），也是经典的动态规划问题。


63. Unique Paths II

Medium | Topics | Companies | Hint

You are given an `m x n` integer array `grid`. There is a robot initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

An obstacle and space are marked as `1` or `0` respectively in `grid`. A path that the robot takes cannot include **any** square that is an obstacle.

Return *the number of possible unique paths that the robot can take to reach the bottom-right corner.*

The testcases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:

```
Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
Output: 2
Explanation: There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right
```

（算法思路：1.B 站 up 代码随想录 2. https://chat.deepseek.com/）

（1）状态定义

创建 dp[i][j] 表示从起点 (0,0) 到达 (i,j) 的路径数量

（2）初始化

起点 (0,0)：如果可通过，路径数为 1；如果是障碍物，路径数为 0

（3）状态转移

对于每个单元格 (i,j)：

如果是障碍物：dp[i][j] = 0（无法到达）

如果不是障碍物：路径数 = 从上面来的路径数 + 从左边来的路径数

dp[i][j] = dp[i-1][j] + dp[i][j-1]

（4）边界处理

第一行 (i=0)：只能从左边来（不能从上面来）

第一列 (j=0)：只能从上面来（不能从左边来）

时间复杂度：O(N×M)，遍历整个网格一次 空间复杂度：O(N×M)，存储 DP 表格

测试结果如下：

```
测试矩阵:
[1, 1, 0, 0, 1, 1]
[1, 0, 1, 0, 0, 0]
[1, 1, 1, 1, 0, 0]
[1, 1, 1, 0, 1, 1]
[0, 1, 1, 1, 1, 1]
[1, 0, 1, 1, 0, 1]
路径数量: 5
```

```
测试矩阵:
[1, 1, 0]
[1, 1, 1]
[1, 1, 1]
路径数量: 5
```

5.3 [5 points] Let N = 10, M = 8, run Count_path for 1000 times, each time the matrix (except the right-bottom corner and top-left corner cells, which remain being 1) is re-filled with integer 0 or 1 randomly, report the mean of total number of paths from the 1000 runs.

Result:

```
[57]: #5. Path counting
#5.3
def run_simulation():
    N = 10
    M = 8
    total_paths = 0
    runs = 1000

    for i in range(runs):
        matrix = create_matrix(N, M)
        paths = Count_path(matrix)
        total_paths += paths

    mean_paths = total_paths / runs
    return mean_paths

mean_result = run_simulation()
print(f"1000次运行的平均路径数: {mean_result}")

1000次运行的平均路径数: 0.338
```

```
[58]: #5. Path counting
#5.3
def run_simulation():
    N = 10
    M = 8
    total_paths = 0
    runs = 1000

    for i in range(runs):
        matrix = create_matrix(N, M)
        paths = Count_path(matrix)
        total_paths += paths

    mean_paths = total_paths / runs
    return mean_paths

mean_result = run_simulation()
print(f"1000次运行的平均路径数: {mean_result}")

1000次运行的平均路径数: 0.426
```