**PART-B**

def product(\*args, repeat=1):

    pools = [tuple(pool) for pool in args] \* repeat

    result = [[]]

    for pool in pools:

        result = [x + [y] for x in result for y in pool]

    return result

def get\_dice\_sums(dice):

    sums = [sum(combination) for combination in product(\*dice)]

    return sums

def generate\_all\_configurations(max\_value, length):

    return list(product(range(1, max\_value + 1), repeat=length))

def undoom\_dice(Die\_A, Die\_B):

    # Calculate original probabilities

    original\_sums = get\_dice\_sums([Die\_A, Die\_B])

    unique\_sums = set(original\_sums)

    original\_probabilities = {sum\_val: original\_sums.count(sum\_val) / len(original\_sums) for sum\_val in unique\_sums}

    # Generate all possible configurations for Die\_A with no more than 4 spots

    possible\_configs\_A = generate\_all\_configurations(4, len(Die\_A))

    # Filter configurations based on the original probabilities

    valid\_configs\_A = [

        config for config in possible\_configs\_A

        if all(x <= 4 for x in config) and get\_dice\_sums([config, Die\_B]) == original\_sums

    ]

    if not valid\_configs\_A:

        return Die\_A, Die\_B

    new\_die\_A = valid\_configs\_A[0]

    return new\_die\_A, Die\_B

die\_A=[2,3]

die\_B=[4,5]

undoom\_dice(die\_A,die\_B)

**OUTPUT:**

([2, 3], [4, 5])



