Problem Statement: The Doomed Dice Challenge

You are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to 6. You can only roll both the dice together & your turn is guided by the obtained sum.

Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9 You may represent Dice as an Array or Array-like structure.Die A = [1, 2, 3, 4, 5, 6] where the indices represent the 6 faces of the die & the value on each face.

Part-A :

1. How many total combinations are possible? Show the math along with the code!

Ans: 36 (by multiplying the size of the both dice we get 6*6 = 36)

2. Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code! Hint: A 6 x 6 Ans:

All possible outcomes:

[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
[(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6)]
[(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6)]
[(4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6)]
[(5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6)]
[(6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)]

All possible sum that can be obtained:

[2, 3, 4, 5, 6, 7]
[3, 4, 5, 6, 7, 8]
[4, 5, 6, 7, 8, 9]
[5, 6, 7, 8, 9, 10]
[6, 7, 8, 9, 10, 11]
[7, 8, 9, 10, 11, 12]

3. Matrix. 3. Calculate the Probability of all Possible Sums occurring among the number of combinations from (2). Example: P(Sum = 2) = 1/X as there is only one combination possible to obtain Sum = 2. Die A = Die B = 1.

Ans:{"2": 0.0278, "3": 0.0556, "4": 0.0833, "5": 0.1111, "6": 0.1389, "7": 0.1667, "8": 0.1389, "9": 0.1111, "10": 0.0833, "11": 0.0556, "12": 0.0278}

dice(): This function simulates rolling two dice, calculates their sum, and prints the individual values and their sum.

comb(die): [Combination] This function calculates and prints the total number of combinations when rolling two dice.

dis_comb(die): [Distribution Combination] This function creates a 2D array (dis_arr) representing all possible combinations of rolling two dice. The combinations are printed, and the array is returned.

dis_poss_comb(dieA, dieB): [Distribution possible Combination] This function creates a 2D array (comb_dis) representing the sum distribution of all possible combinations of rolling two dice. The sum distribution is printed.

prob(die): [Probability] This function calculates and prints the probability of each possible sum when rolling two dice. It iterates through all possible sums (from 2 to 12) and counts the occurrences in the simulated dice rolls.

Output for Part-A

```
(3,4)
7
Total combinations : 36
All possible combinations
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
[(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6)]
[(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6)]
[(4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6)]
[(5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6)]
[(6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)]
sum distribution
[2, 3, 4, 5, 6, 7]
[3, 4, 5, 6, 7, 8]
[4, 5, 6, 7, 8, 9]
[5, 6, 7, 8, 9, 10]
[6, 7, 8, 9, 10, 11]
[7, 8, 9, 10, 11, 12]
sum probability
 p(sum=2)=0.02777777777777776=0.028
 p(sum=3)=0.02777777777777776=0.056
 p(sum=4)=0.02777777777777776=0.083
 p(sum=5)=0.02777777777777776=0.111
 p(sum=6)=0.02777777777777776=0.139
 p(sum=7)=0.02777777777777776=0.167
 p(sum=8)=0.02777777777777776=0.139
 p(sum=9)=0.02777777777777776=0.111
 p(sum=10)=0.02777777777777776=0.083
 p(sum=11)=0.02777777777777776=0.056
 p(sum=12)=0.02777777777777776=0.028
```

Part-B:

Now comes the real challenge. You were happily spending a lazy afternoon playing your board game with your dice when suddenly the mischievous Norse God Loki (You love Thor too much & Loki didn't like that much) appeared. Loki dooms your dice for his fun removing all the "Spots" off the dice. No problem! You have the tools to re attach the "Spots" back on the Dice. However, Loki has doomed your dice with the following conditions:

- Die A cannot have more than 4 Spots on a face.
- Die A may have multiple faces with the same number of spots.
- Die B can have as many spots on a face as necessary i.e. even more than 6.

But in order to play your game, the probability of obtaining the Sums must remain the same! So if you could only roll P(Sum = 2) = 1/X, the new dice must have the spots reattached such that those probabilities are not changed.

Input:

- Die_A = [1, 2, 3, 4, 5, 6] & Die B = Die_A = [1, 2, 3, 4, 5, 6]

Output:

- A Transform Function undoom_dice that takes (Die_A, Die_B) as input & outputs New_Die_A = [?, ?, ?, ?, ?, ?],New_Die_B = [?, ?, ?, ?, ?, ?] where, No New_Die A[x] > 4

Ans:

New Dice are
(1, 2, 2, 3, 3, 4)
(1, 3, 4, 5, 6, 8)

Logic:

The problem's logic lies in maintaining the total of 42 spots after the Loki's restriction.The first approach that comes to the mind is to brute force it with all the combinations of the dice A with 6 faces and spots of [1,2,3,4] and dice B with 6 faces and spots of 1 to 11 .After brute-forcing the output had multiple redundant values with different positions.But still gave the answer arrays in the time complexity of n to the power of 6 by 6 nested for loops. After brute-forcing the recursion and memoization i.e. Dynamic programming concept is used to store the combinations and the redundant values have been removed. This reduced the space and time complexity.

After finding all the possibilities of the dice A and B now they are passed to a function iteratively to find the sum probabilities and compare them with the original probability.

In order to reduce the complexity further the condition of sum of all numbers in the both dice equals to 42 is used.And after some analysis it is clear that there needs to be [1,4] and [1,8] in dice A and B
Because 2 and 12 can be made only with these numbers under the Loki's restriction.so the free spots in both dice is reduced to 4 and the values needs to be given as input is reduced in die B is reduced from 12 to 6.This reduced the time complexity drastically.

printOrderly(array): This function takes an array as input and prints its elements in a new line.

sumOfArray(array): This function calculates and returns the sum of the elements in the given array.

sum_Distribution(Die_A, Die_B): This function creates a 2D array representing the sum distribution of all possible combinations of Die_A and Die_B values.

single_Probability(Die_A, Die_B, sum): This function calculates the probability of a specific sum occurring when two dice are rolled.

all_Probability(Die_A, Die_B): This function calculates the probabilities of sums ranging from 2 to 12 for the given dice combinations and returns them as a dictionary.

posibilities_Calc(curr, free_space, input_values, posibilities, fixed_values, repetition=True): This is a recursive function that calculates all possible combinations of values for a die, considering fixed values and free spaces. The results are stored in a set to eliminate duplicates.

transform(die_a, die_b): This is the main function that transforms the given dice based on Loki's rules. It calculates the original probabilities, defines fixed and input values for each die, and then finds unique possibilities for both dice. Finally, it checks combinations of Die_A and Die_B to identify a new set of dice that satisfies Loki's rules and returns them.

Output for Part-B

```
New Dice are
(1, 2, 2, 3, 3, 4)
(1, 3, 4, 5, 6, 8)
```