Business Administration Karachi
Leadership and Ideas for Tomorrow

Subject:
Quiz #: Date :
Class/Section:
Teacher's Name:

(Q5) DISCLAIMER: This solution has been found after consulting a series of sources including geeksforgeeks, afteracademy.com, tutorialspoint, techiedelight.com, chatGPT, pepcodeacamedy (YouTube channel) and medium.com. However, I do under-stand the solution and the workings and I have not plagiarized but there will be many similarities between codes found at these sources and my approach just because it is the nature of the problem. In short, please don't think I am a cheater. Thank you.

Input: m, n, S
$m \downarrow$ Faces
$n \downarrow$ dice
$S \downarrow$ target sum.

Cases:

$S == 0$; if target sum is 0 then just by not throwing anymore dice is how we get to the sum and that is the only way so n must be 0 too. ~~So~~

$n == 0$; if no dice are thrown, we can not reach our sum S so there is no way (return 0) that this happens.

$s < 0$; if sum being calculated is negative value, there is no way to reach it because all dice can hold only positive values from 1 to m so return 0.

Scanned with CamScanner

$s = s - i$;
$n = n - 1$;

Suppose $s > 0$ and $n != 0$ then we can check what would be each of the results (i.e. could we reach a sum) if this n^{th} dice would get each of its 1 to m faces. If any of these faces result in it getting or not getting the sum, add it to the number of ways of reaching the sum at this dice. At each throw, s would be reduced by the value i between 1 & m, and no. of dice be reduced by 1.

To log these results, we will need a matrix of $(1+n) \times (S+1)$ size. D [n] [S] corresponds to total ways of reaching the sum S from dice n's outcomes.

Algorithm: (Using top-down approach!)

// matrix of size $(1+n) \times (S+1)$ created to log results.

```
D = int [n+1] [S+1];
for i = 0 to n
{   for j = 0 to S
    {  M[i][j] = -1 ; }
```

Now we call actual algorithm: DICE (n, m, M[][], S);

Subject:
Quiz #: Date :
Class/Section:
Teacher's Name:

```
DICE ( n , m , M[][] , S )
{
    if (S==0 && n==0) then return 1;
    if (n==0) then return 0;
    if (S<0) then return 0;
    if W[n][S] ≠ -1 then return W[n][S]; // W[n][S] already been computed so return.
  else
    W[n][S] = 0
    for i = 1 to m
    {
        W[n][S] = W[n][S] + DICE( n-1 , m , M[][] , S-i );
    }                         // One die gone and then the value of ~~gone guy~~ subtracted from original S to give new S.
    return W[n][S];
}
```

Time complexities: $O(m \times n \times S)$

↳ Traverse each element of $(n+1) \times (S+1)$ matrix in $O(n \times S)$ time. In each entry, there traverse a 1 to m for-loop. So it becomes $O(m \times n \times S)$.

Space complexity: $O(n \times S)$

↳ $(n+1) \times (S+1)$ size matrix used to log results.

WORKING: (Thought process shown as proof of concept →)

PERFECT SUBSET SUM

4 , 3 , 9

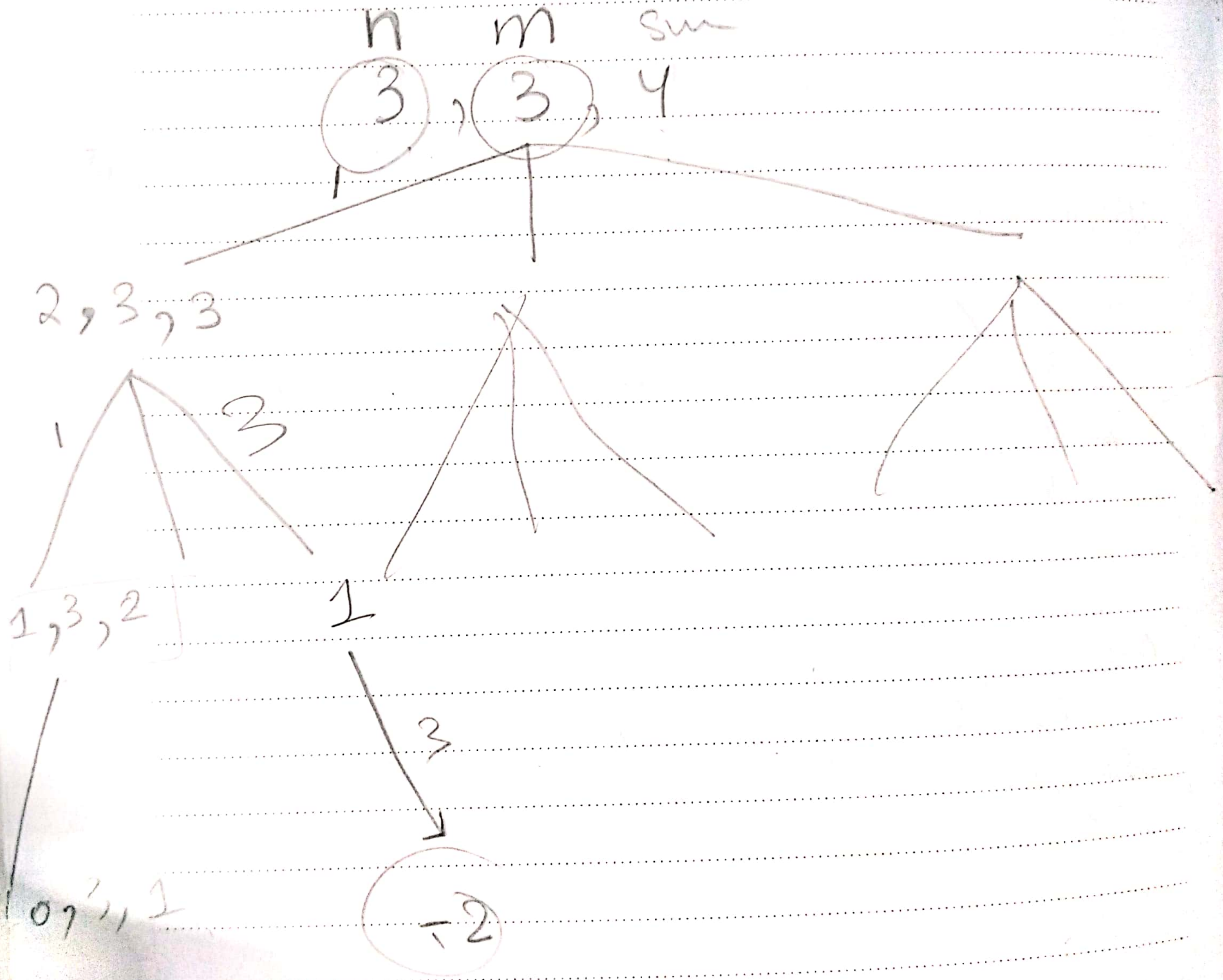
n m S

- n number of dices
- m number of faces
- s sum

When you roll a die → $d_i < s$ $s = s - d_i$, $n = n-1$

$d_i > s$ $s = S$, $n = n-1$

$s == 0$ $n = 0$ so start backtracking



4, 2

n

i

m

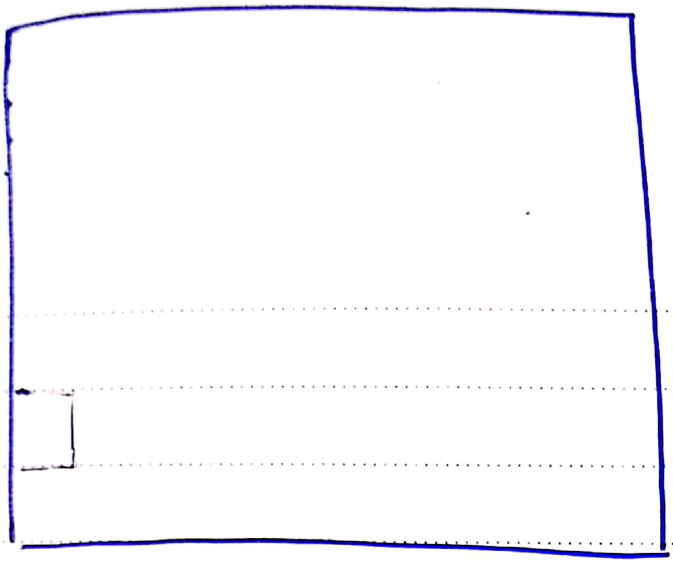
i

3, 3, 4

Q5 :

n dice

m faces



Given sum to reach.

After each throw of a dice, we are left with n - throws throws and at each throw we can get 1 to m number.

Base cases:

- ★ Sum ~~not~~ reached and no throws left → Pass
- ★ Sum not reached and no throws left → Fail
- ★ Current sum already achieved and throws remain → fail
- ★ ~~S~~ Traverse to find sum - i value and reduce throws by 1.

Algorithm:

Find Dice (n , m , S) {

~~For~~ DICEFACES SUM = [m × S] matrix

For

0 . . . . . . S

0 | $\boxed{1}$ 0 0 0 0 0 0   m x S

m | $\boxed{n,S}$

```
if (s==0 && n==0)   // target sum = 0 & no dice thrown, then you can achieve sum in 1 way.
    return 1;       // M[0][0]

if (s<0 || n==0)    // sum can never be negative and its not like we can throw no coins so both cases mean there is no way to calculate sum.
    return 0;

W = int [n+1][S+1]
for i = 0 to n.
    for j = 0 to S
        W[i][j] = -1
```

$O(nS)$.