GeeksforGeeks
A computer science portal for geeks

Custom Search

Practice | GATE CS | Placements | Videos | Contribute

Login/Register

Dynamic Programming | Set 30 (Dice Throw)

Given n dice each with m faces, numbered from 1 to m, find the number of ways to get sum X. X is the summation of values on each face when all the dice are thrown.

Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.

The **Naive approach** is to find all the possible combinations of values from n dice and keep on counting the results that sum to X.

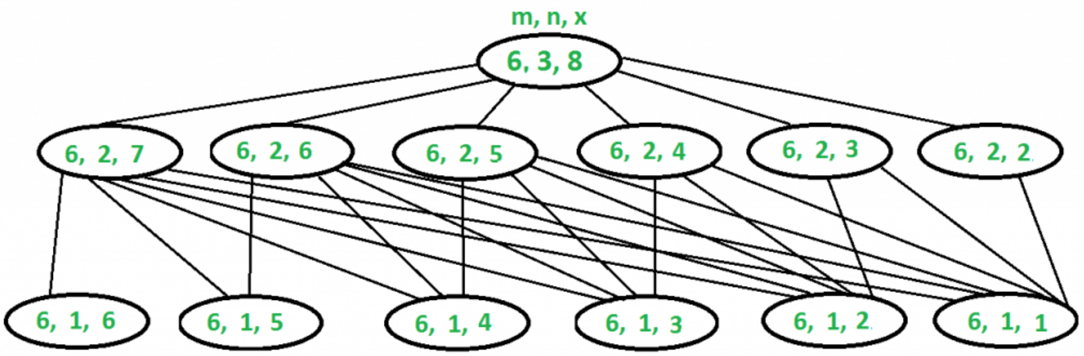
This problem can be efficiently solved using **Dynamic Programming (DP)**.

```
Let the function to find X from n dice is: Sum(m, n, X)
The function can be represented as:
Sum(m, n, X) = Finding Sum (X - 1) from (n - 1) dice plus 1 from nth dice
               + Finding Sum (X - 2) from (n - 1) dice plus 2 from nth dice
               + Finding Sum (X - 3) from (n - 1) dice plus 3 from nth dice
                  ...................................................
                  ...................................................
                  ...................................................
              + Finding Sum (X - m) from (n - 1) dice plus m from nth dice

So we can recursively write Sum(m, n, x) as following
Sum(m, n, X) = Sum(m, n - 1, X - 1) +
               Sum(m, n - 1, X - 2) +
               .................... +
               Sum(m, n - 1, X - m)
```

Why DP approach?

The above problem exhibits overlapping subproblems. See the below diagram. Also, see this recursive implementation. Let there be 3 dice, each with 6 faces and we need to find the number of ways to get sum 8:



```
Sum(6, 3, 8) = Sum(6, 2, 7) + Sum(6, 2, 6) + Sum(6, 2, 5) +
               Sum(6, 2, 4) + Sum(6, 2, 3) + Sum(6, 2, 2)

To evaluate Sum(6, 3, 8), we need to evaluate Sum(6, 2, 7) which can
recursively written as following:
Sum(6, 2, 7) = Sum(6, 1, 6) + Sum(6, 1, 5) + Sum(6, 1, 4) +
               Sum(6, 1, 3) + Sum(6, 1, 2) + Sum(6, 1, 1)

We also need to evaluate Sum(6, 2, 6) which can recursively written
as following:
Sum(6, 2, 6) = Sum(6, 1, 5) + Sum(6, 1, 4) + Sum(6, 1, 3) +
               Sum(6, 1, 2) + Sum(6, 1, 1)
..............................................
..............................................
Sum(6, 2, 2) = Sum(6, 1, 1)
```

Please take a closer look at the above recursion. The sub-problems in RED are solved first time and sub-problems in BLUE are solved again (exhibit overlapping sub-problems). Hence, storing the results of the solved sub-problems saves time.

Following is C++ implementation of Dynamic Programming approach.

```
// C++ program to find number of ways to get sum 'x' with 'n'
// dice where every dice has 'm' faces
#include <iostream>
#include <string.h>
using namespace std;

// The main function that returns number of ways to get sum 'x'
// with 'n' dice and 'm' with m faces.
int findWays(int m, int n, int x)
{
    // Create a table to store results of subproblems.  One extra
    // row and column are used for simpilicity (Number of dice
    // is directly used as row index and sum is directly used
    // as column index).  The entries in 0th row and 0th column
    // are never used.
    int table[n + 1][x + 1];
    memset(table, 0, sizeof(table)); // Initialize all entries as 0

    // Table entries for only one dice
    for (int j = 1; j <= m && j <= x; j++)
        table[1][j] = 1;

    // Fill rest of the entries in table using recursive relation
    // i: number of dice, j: sum
    for (int i = 2; i <= n; i++)
```

```
        for (int j = 1; j <= x; j++)
            for (int k = 1; k <= m && k < j; k++)
                table[i][j] += table[i-1][j-k];

    /* Uncomment these lines to see content of table
    for (int i = 0; i <= n; i++)
    {
      for (int j = 0; j <= x; j++)
        cout << table[i][j] << " ";
      cout << endl;
    } */
    return table[n][x];
}

// Driver program to test above functions
int main()
{
    cout << findWays(4, 2, 1) << endl;
    cout << findWays(2, 2, 3) << endl;
    cout << findWays(6, 3, 8) << endl;
    cout << findWays(4, 2, 5) << endl;
    cout << findWays(4, 3, 5) << endl;

    return 0;
}
```

Run on IDE

Output:

```
0
2
21
4
6
```

Time Complexity: O(m * n * x) where m is number of faces, n is number of dice and x is given sum.

We can add following two conditions at the beginning of findWays() to improve performance of program for extreme cases (x is too high or x is too low)

```
// When x is so high that sum can not go beyond x even when we
// get maximum value in every dice throw.
if (m*n <= x)
    return (m*n == x);

// When x is too low
if (n >= x)
    return (n == x);
```

Run on IDE

With above conditions added, time complexity becomes O(1) when x >= m*n or when x <= n. **Exercise:** Extend the above algorithm to find the probability to get Sum > X.

This article is compiled by Aashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner   Company Wise Coding Practice

Dynamic Programming Mathematical

Recommended Posts:

- Dynamic Programming | Set 31 (Optimal Strategy for a Game)
- Dynamic Programming | Set 22 (Box Stacking Problem)
- Top 20 Dynamic Programming Interview Questions
- Dynamic Programming | Set 36 (Maximum Product Cutting)
- Dynamic Programming | Set 32 (Word Break Problem)
- Count ways to build street under given constraints
- Maximum length subsequence with difference between adjacent elements as either 0 or 1
- Unique paths in a Grid with Obstacles
- Largest rectangular sub-matrix whose sum is 0
- Coin game winner where every player has three choices

(Login to Rate and Mark)

3.6 Average Difficulty : **3.6/5.0**
Based on **51** vote(s)

□ Add to TODO List

□ Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments   Share this post!

Contact Us! About Us! Careers! Privacy Policy