# Number of paths with exactly k coins

Given a matrix where every cell has some number of coins. Count number of ways to reach bottom right from top left with exactly k coins. We can move to (i+1, j) and (i, j+1) from a cell (i, j).

Example:

```
Input:  k = 12
        mat[][] = { {1, 2, 3},
                    {4, 6, 5},
                    {3, 2, 1}
                  };
Output:  2
There are two paths with 12 coins
1 -> 2 -> 6 -> 2 -> 1
1 -> 2 -> 3 -> 5 -> 1
```

**We strongly recommend that you click here and practice it, before moving on to the solution.**

The above problem can be recursively defined as below:

```
pathCount(m, n, k):   Number of paths to reach mat[m][n] from mat[0][0]
                      with exactly k coins

If (m == 0 and n == 0)
   return 1 if mat[0][0] == k else return 0
Else:
    pathCount(m, n, k) = pathCount(m-1, n, k - mat[m][n]) +
                         pathCount(m, n-1, k - mat[m][n])
```

Below is C++ implementation of above recursive algorithm.

```
// A Naive Recursive C++ program to count paths with exactly
// 'k' coins
#include <bits/stdc++.h>
#define R 3
#define C 3
using namespace std;

// Recursive function to count paths with sum k from
// (0, 0) to (m, n)
int pathCountRec(int mat[][C], int m, int n, int k)
{
    // Base cases
    if (m < 0 || n < 0) return 0;
    if (m==0 && n==0) return (k == mat[m][n]);

    // (m, n) can be reached either through (m-1, n) or
    // through (m, n-1)
    return pathCountRec(mat, m-1, n, k-mat[m][n]) +
           pathCountRec(mat, m, n-1, k-mat[m][n]);
}

// A wrapper over pathCountRec()
int pathCount(int mat[][C], int k)
{
    return pathCountRec(mat, R-1, C-1, k);
}

// Driver program
int main()
{
    int k = 12;
    int mat[R][C] = { {1, 2, 3},
                      {4, 6, 5},
                      {3, 2, 1}
                  };
    cout << pathCount(mat, k);
    return 0;
}
```

Output:

```
2
```

The time complexity of above solution recursive is exponential. We can solve this problem in Pseudo Polynomial Time (time complexity is dependent on numeric value of input) using Dynamic Programming. The idea is to use a 3 dimensional table dp[m][n][k] where m is row number, n is column number and k is number of coins. Below is Dynamic Programming based C++ implementation.

```
// A Dynamic Programming based C++ program to count paths with
// exactly 'k' coins
#include <bits/stdc++.h>
#define R 3
#define C 3
#define MAX_K 1000
using namespace std;

int dp[R][C][MAX_K];

int pathCountDPRecDP(int mat[][C], int m, int n, int k)
{
    // Base cases
    if (m < 0 || n < 0) return 0;
    if (m==0 && n==0) return (k == mat[m][n]);

    // If this subproblem is already solved
    if (dp[m][n][k] != -1) return dp[m][n][k];

    // (m, n) can be reached either through (m-1, n) or
    // through (m, n-1)
    dp[m][n][k] = pathCountDPRecDP(mat, m-1, n, k-mat[m][n]) +
                  pathCountDPRecDP(mat, m, n-1, k-mat[m][n]);

    return dp[m][n][k];
}

// This function mainly initializes dp[][][] and calls
// pathCountDPRecDP()
int pathCountDP(int mat[][C], int k)
{
    memset(dp, -1, sizeof dp);
    return pathCountDPRecDP(mat, R-1, C-1, k);
}

// Driver Program to test above functions
int main()
{
    int k = 12;
    int mat[R][C] = { {1, 2, 3},
                      {4, 6, 5},
                      {3, 2, 1}
                  };
    cout << pathCountDP(mat, k);
    return 0;
}
```

Output:

```
2
```

Time complexity of this solution is O(m*n*k).