Grid Unique Paths

Brute force :

Exploring all the paths that can be found to reach to the destination. We make use of recursion to find all the paths if a path is foun to the destination then we return 1 and this gets added for all the recursive calls. 2 base cases needs to be taken care of one where the pointers go out of bounds while moving to right and down i.e i≥m && j≥n then we return 0 because no more path to be explored remains and the other base case is when pointers reach the destination where we return 1.

Now while moving right we make recursive call with (i,j+1) and while moving down the recursive call is made to (i+1,j) summation of these two recursive calls will be the ans.

```
class Solution {
public:
    int countUniqPaths(int i,int j,int m,int n)
    {
        if(i>=m || j>=n)
            return 0;
        if(i==m-1 && j==n-1)
            return 1;

        return countUniqPaths(i+1,j,m,n)+countUniqPaths(i,j+1,m,n);
    }
    int uniquePaths(int m, int n) {
        return countUniqPaths(0,0,m,n);
    }
};
```

- Time Complexity :Exponential
- Space Complexity : Stack space exponential

Better Approach : Use DP to optimise.

Using memoization or bottom up approach optimise to get quadratic complexity. In memoization we store the value of already completed recursive calls so that if a call is made in future we can simply return the value stored. So minor changes needs to be made

1. A vector of changing parameter dimension needs to be made (here m+1 x n+1) and initialised to -1.
2. If the recursive call is made then a value would have been set at that set so return the value i.e

if(dp[i][j]≠-1) return dp[i][j]

3. Store recursive calls value dp[i][j]=countUniqPaths(i+1,j,m,n,dp)+countUniqPaths(i,j+1,m,n,dp)

```
class Solution {
public:
    int countUniqPaths(int i,int j,int m,int n,vector<vector<int>> &dp)
    {
        if(dp[i][j]!=-1)
            return dp[i][j];
        if(i>=m || j>=n)
            return 0;
        if(i==m-1 && j==n-1)
            return 1;

        return dp[i][j]=countUniqPaths(i+1,j,m,n,dp)+countUniqPaths(i,j+1,m,n,dp);
    }
    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m+1,vector<int>(n+1,-1));
        return countUniqPaths(0,0,m,n,dp);
    }
};
```

- Time Complexity : O(n*m)
- Space Complexity : O(n*m) dp matrix

Optimal Approach : Combinatorics Solution

We know that at max we can move m-1 times in right direction and n-1times downwards which gives us a total of m-1 + n-1 = m+n-2 out of these if we can choose the position of either right or down then we can know the ans by nCr when n=m+n-1 and r=m-1 or n-1 that is choosing positions when to move rightwards(say). To find nCr we now 10C2 is 9*10/2*1 i.e we need to write the number r number of times starting from 1 in the

denominator and for numerator it is n-r+i where n-r gives us the starting position of number and + i gives the position till when we have to move in numerator ex : n-r in 10C2 is 8 +1 = 9 so the r loop runs for denominator from 1 to m-1 that is 1 to 2 so in denominator correspondingly we get 9*10.

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        int N = m+n-2;
        int R = m-1;
        double res=1;
        for(int i=1;i<=R;i++)
        {
            res=res*(N-R+i)/i;
        }
        return (int)res;
    }
};
```

- Time Complexity : O(m)
- Space Complexity : O(1)