# DYNAMIC PROGRAMMING

**Q1: There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1, 2, 3...m stairs at a time where m is a user-given integer. Count the number of ways the person can reach the top.**

```java
public class StairClimbing {
    public static int countWays(int n, int m) {
        int[] dp = new int[n + 1];
        dp[0] = 1;

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m && j <= i; j++) {
                dp[i] += dp[i - j];
            }
        }

        return dp[n];
    }

    public static void main(String[] args) {
        int n = 5;
        int m = 3;
        int ways = countWays(n, m);
        System.out.println("Number of ways to reach the top: " + ways);
    }
}
```

**Q2: The Tribonacci sequence Tn is defined as follows: TO = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0. Given n, return the value of nth tribonacci number.**

```java
public class Tribonacci {
    public static int tribonacci(int n) {
        if (n == 0) return 0;
```

```
        if (n == 1 || n == 2) return 1;

        int[] dp = new int[n + 1];
        dp[0] = 0;
        dp[1] = 1;
        dp[2] = 1;

        for (int i = 3; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
        }

        return dp[n];
    }

    public static void main(String[] args) {
        int n1 = 4;
        int result1 = tribonacci(n1);
        System.out.println("Tribonacci(" + n1 + ") = " + result1);

        int n2 = 25;
        int result2 = tribonacci(n2);
        System.out.println("Tribonacci(" + n2 + ") = " + result2);
    }
}
```

**Q3: You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.**

```java
public class HouseRobber {
    public static int rob(int[] nums) {
        int n = nums.length;
        if (n == 0) return 0;
        if (n == 1) return nums[0];
```

```
        int[] dp = new int[n];
        dp[0] = nums[0];
        dp[1] = Math.max(nums[0], nums[1]);

        for (int i = 2; i < n; i++) {
            dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i]);
        }

        return dp[n - 1];
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 1};
        int maxAmount = rob(nums);
        System.out.println("Maximum amount of money that can be robbed: " +
maxAmount);
    }
}
```

**Q4: There is a robot on an m x n grid. The robot is initially located at the top-left corner. The robot can only move either down or right at any point in time. Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.**

```java
public class UniquePaths {
    public static int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (i == 0 || j == 0) {
                    dp[i][j] = 1;
                } else {
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
```

```
            }
        }
    }

    return dp[m - 1][n - 1];
  }

  public static void main(String[] args) {
    int m = 3;
    int n = 7;
    int paths = uniquePaths(m, n);
    System.out.println("Number of unique paths: " + paths);
  }
}
```

**Q5: Given a triangle array, return the minimum path sum from top to bottom.**

```java
import java.util.List;

public class MinimumPathSum {
  public static int minimumTotal(List<List<Integer>> triangle) {
    int n = triangle.size();
    int[][] dp = new int[n][n];

    for (int i = 0; i < n; i++) {
      dp[n - 1][i] = triangle.get(n - 1).get(i);
    }

    for (int i = n - 2; i >= 0; i--) {
      for (int j = 0; j <= i; j++) {
        dp[i][j] = triangle.get(i).get(j) + Math.min(dp[i + 1][j], dp[i + 1][j + 1]);
      }
    }
```

```
        return dp[0][0];
    }

    public static void main(String[] args) {
        List<List<Integer>> triangle = List.of(
                List.of(2),
                List.of(3, 4),
                List.of(6, 5, 7),
                List.of(4, 1, 8, 3)
        );

        int minPathSum = minimumTotal(triangle);
        System.out.println("Minimum path sum: " + minPathSum);
    }
}
```