题目描述

一天一只顽猴想去从山脚爬到山顶，途中经过一个有个N个台阶的阶梯，但是这猴子有一个习惯：

每一次只能跳1步或跳3步，试问猴子通过这个阶梯有多少种不同的跳跃方式？

输入描述

输入只有一个整数N（0<N<=50）此阶梯有多少个台阶。

输出描述

输出有多少种跳跃方式（解决方案数）。

用例

输入	50
输出	122106097
说明	无

输入	3
输出	2
说明	无

题目解析

这题是一道经典的分治算法题、以及动态规划基础题。

这题既可以使用分治递归来自顶向下地求解，也可以使用动态规划递推地自底向上求解。

但是动态规划的效率更高。

这题和fibnaci数列很像，递归公式如下：

- jump(1) = 1
- jump(2) = 1
- jump(3) = 2
- jump(n) = jump(n-1) + jump(n-3)，n>3

状态转移方程如下：

- dp[0] = 0
- dp[1] = 1
- dp[2] = 1
- dp[3] = 2
- dp[i] = dp[i-1] + dp[i-3]，i>3

上面递归公式和状态转移方程的规律发现如下

台阶数	跳跃方式	几种跳跃方式
1	1	1
2	11	1
3	111 3	2
4	1111 13 31	3
5	11111 113 131 311	4
6	111111 1113 1131 1311 3111 33	6
7	1111111 11113 11131 11311 13111 31111 133 313 331	9

Java算法源码

分治递归求解（带缓存优化）

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  static int[] cache = new int[51];

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();

    Arrays.fill(cache, -1);
    cache[0] = 0;
    cache[1] = 1;
    cache[2] = 1;
    cache[3] = 2;

    System.out.println(recursive(n));
  }

  public static int recursive(int n) {
    if (cache[n] != -1) return cache[n];
    cache[n] = recursive(n - 1) + recursive(n - 3);
    return cache[n];
  }
}
```

动态规划算法求解

```java
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    System.out.println(getResult(n));
  }

  public static int getResult(int n) {
    int[] dp = new int[n + 1];

    if (n >= 1) dp[1] = 1;
    if (n >= 2) dp[2] = 1;
    if (n >= 3) dp[3] = 2;

    for (int i = 4; i <= n; i++) {
      dp[i] = dp[i - 1] + dp[i - 3];
    }

    return dp[n];
  }
}
```

JS算法源码

分治递归求解（带缓存优化）

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const n = parseInt(line);
  console.log(recursive(n));
});

const cache = [0, 1, 1, 2];
function recursive(n) {
  if (cache[n]) return cache[n]; // n > 0
  cache[n] = recursive(n - 1) + recursive(n - 3);
  return cache[n];
}
```

动态规划算法求解

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const n = parseInt(line);
  console.log(getResult(n));
});

function getResult(n) {
  const dp = new Array(n + 1).fill(0);
  if (n >= 1) dp[1] = 1;
  if (n >= 2) dp[2] = 1;
  if (n >= 3) dp[3] = 2;

  for (let i = 4; i <= n; i++) {
    dp[i] = dp[i - 1] + dp[i - 3];
  }

  return dp[n];
}
```

Python算法源码

分治递归求解（带缓存优化）

```python
# 输入获取
n = int(input())

cache = [-1] * 51
cache[0] = 0
cache[1] = 1
cache[2] = 1
cache[3] = 2


# 算法入口
def recursive(n):
    if cache[n] != -1:
        return cache[n]

    cache[n] = recursive(n - 1) + recursive(n - 3)
    return cache[n]


# 调用算法
print(recursive(n))
```

动态规划算法求解

```python
# 输入获取
n = int(input())


# 算法入口
def getResult():
    dp = [0]*(n+1)

    if n >= 1:
        dp[1] = 1
    if n >= 2:
        dp[2] = 1
    if n >= 3:
        dp[3] = 2

    for i in range(4, n+1):
        dp[i] = dp[i-1] + dp[i-3]

    return dp[n]


# 调用算法
print(getResult())
```