# 题目

泰波那契序列 Tn 定义如下：

T0 = 0，T1 = 1，T2 = 1，且在n >= 0 的条件下Tn+3 = Tn + Tn+1 + Tn+2

给你整数 n，请返回第n个泰波那契数 Tn的值。

示例 1：

输入：n = 4

输出：4

解释：

T\_3 = 0 + 1 + 1 = 2

T\_4 = 1 + 1 + 2 = 4

示例 2：

输入：n = 25

输出：1389537

提示：

0 <= n <= 37

答案保证是一个32位整数，即 answer <= 2^31- 1。

# 分析

## 方法一：动态规划

泰波那契数的边界条件是：T(0)=0，T(1)=1，T(2)=1。

当n>2n>2时，每一项的和都等于前三项的和，因此有如下递推关系：

T(n)=T(n-1)+T(n-2)+T(n-3)

由于泰波那契数存在递推关系，因此可以使用动态规划求解。动态规划的状态转移方程即为上述递推关系，边界条件为T(0)、T(1)和T(2)。

根据状态转移方程和边界条件，可以得到时间复杂度和空间复杂度都是O(n)的实现。由于T(n)只和前三项有关，因此可以使用「滚动数组思想」将空间复杂度优化成O(1)。

如下的代码中给出的就是这种实现。

**代码：**

动态规划的推导过程如下。

定义状态：T[i]表示泰波那契数列的第i个数字；

状态转移方程：T(N) = T(N - 1) + T(N - 2) + T(N - 3), 其中N >= 3.

初始条件：T(0) = 0, T(1) = 1, T(2) = 1.

class Solution {

public:

int tribonacci(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

if (n == 2) return 1;

vector<int> dp(n + 1);

dp[0] = 0;

dp[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];

}

};

时间复杂度：O(n)

空间复杂度：O(n)

**或（下面的写法不需要额外内存，但是稍微难理解一些）：**

class Solution {

public:

int tribonacci(int n) {

if (n == 0) {

return 0;

}

if (n <= 2) {

return 1;

}

int p = 0, q = 0, r = 1, s = 1;

for (int i = 3; i <= n; ++i) {

p = q;

q = r;

r = s;

s = p + q + r;

}

return s;

}

};

**复杂度分析：**

时间复杂度：O(n)。

空间复杂度：O(1)。