# 题目

给定一个非负整数 num。对于 0 ≤ i ≤ num 范围中的每个数字 i ，计算其二进制数中的 1 的数目并将它们作为数组返回。

**示例 1:**

输入: 2

输出: [0,1,1]

**示例 2:**

输入: 5

输出: [0,1,1,2,1,2]

**进阶:**

给出时间复杂度为O(n\*sizeof(integer))的解答非常容易。但你可以在线性时间O(n)内用一趟扫描做到吗？

要求算法的空间复杂度为O(n)。

你能进一步完善解法吗？要求在C++或任何其他语言中不使用任何内置函数（如 C++ 中的 \_\_builtin\_popcount）来执行此操作。

# 分析

## 方法一：位运算

class Solution {

public:

vector<int> countBits(int num) {

vector<int> ret(num+1);

for(int i=1;i<=num;i++)

{

if(i & 1)

{

//如果是奇数,则相邻位必然为1,否则i就是偶数了

ret[i] = ret[i-1] + 1;

}

else

{

//如果是偶数,则相邻位不一定为1,需要右值计算

ret[i] = ret[i>>1];

}

}

return ret;

}

};

## 方法二：动态规划

思路：

要在线性时间复杂度O(n)内解决这个问题，并且不使用任何内置函数来计算一个整数的二进制表示中1的个数，我们可以使用动态规划的方法。

思路是这样的：对于任何正整数num，如果我们知道num - 1中1的个数，那么我们可以通过观察num和num - 1的二进制表示来得到num中1的个数。具体地，如果num是偶数，那么num的二进制表示最后一位是0，而num - 1的二进制表示的最后一位是1，并且除了最后一位之外，num和num - 1的二进制表示是相同的。因此，num中1的个数就是num - 1中1的个数加1。如果num是奇数，那么num的二进制表示最后一位是1，而num - 1的二进制表示的最后一位是0，并且除了最后一位之外，num的二进制表示比num - 1的二进制表示多出一个1（即最高位的1）。因此，num中1的个数就是num / 2中1的个数加1。

基于这个思路，我们可以从0开始遍历到n，并使用一个数组dp来保存每个数中1的个数。对于每个数i，我们根据i是奇数还是偶数来更新dp[i]。

代码：

class Solution {

public:

vector<int> countBits(int n) {

vector<int> dp(n + 1, 0); // 初始化dp数组，长度为n+1，所有元素初始值为0

for (int i = 1; i <= n; i++) {

if (i % 2 == 0) {

dp[i] = dp[i / 2]; // 如果i是偶数，则dp[i]等于dp[i/2]

} else {

dp[i] = dp[i / 2] + 1; // 如果i是奇数，则dp[i]等于dp[i/2]加1

}

}

return dp;

}

};