# 题目

给定一个整数 n，计算所有小于等于 n 的非负整数中数字 1 出现的个数。

示例 1：

输入：n = 13

输出：6

示例 2：

输入：n = 0

输出：0

提示：

0 <= n <= 109

# 分析

要解决计算所有小于等于n的非负整数中数字1出现的个数的问题，我们可以通过按位分析的方法，逐位计算每个数位上1出现的次数，再累加所有数位的结果。这种方法无需遍历每个数字，能高效处理大数值（如n达10^9）。

解题思路

1、数位分解：将数字n按个位、十位、百位等不同数位分解，分别计算每个数位上1出现的次数。

2、按位计算规则：

- 对于当前数位d（设为第k位，从0开始计数，单位为10^k），将数字分为三部分：

high：当前数位左边的数字（高位部分）。

curr：当前数位上的数字。

low：当前数位右边的数字（低位部分）。

该数位的权重为base = 10^k（如个位base=1，十位base=10）。

- 根据curr的值分情况计算该数位上1出现的次数：

若curr < 1：次数为high \* base。

若curr == 1：次数为high \* base + low + 1。

若curr > 1：次数为(high + 1) \* base。

3、累加结果：遍历所有数位，累加每个数位上1出现的次数，得到最终结果。

代码：

class Solution {

public:

int countDigitOne(int n) {

if (n < 1) return 0; // 非负整数中，n<1时没有1

long long count = 0;

long long base = 1; // 当前数位的权重（10^k）

long long high = n / 10; // 高位部分

long long curr = n % 10; // 当前数位

long long low = 0; // 低位部分

while (high != 0 || curr != 0) { // 直到所有数位处理完毕

if (curr < 1) {

count += high \* base;

} else if (curr == 1) {

count += high \* base + low + 1;

} else {

count += (high + 1) \* base;

}

// 更新各部分，处理更高一位

low += curr \* base; // 低位累加当前数位的值

curr = high % 10; // 当前数位移至高位的最后一位

high /= 10; // 高位缩短一位

base \*= 10; // 权重升级（如从个位1→十位10）

}

return count;

}

};

代码解释：

- 数位分解：通过high = n / 10、curr = n % 10、low = 0初始化高位、当前位和低位，再通过循环逐步处理更高数位（每次base乘10，high缩短一位，`low`累加当前位的值）。

- 分情况计算：根据当前数位curr的值，按规则计算该数位上1出现的次数，确保覆盖所有可能的情况（当前位小于1、等于1、大于1）。

- 终止条件：当high和curr均为0时，所有数位已处理完毕，累加的count即为结果。

复杂度分析

- 时间复杂度：O(log n)，其中n是输入整数。循环次数等于n的位数，而n的位数为log10(n) + 1，因此时间复杂度与log n成正比。

- 空间复杂度：O(1)，仅使用常数个变量存储中间结果，无需额外空间。

这种方法通过数学分析避免了遍历每个数字，高效解决了大数值下的计算问题。