编程之美解题报告

姚大海 (acprimer.yao@gmail.com)

https://github.com/acprimer/Beauty-of-Programming

最后更新 2014-12-20

版权声明

内容简介

编程之美解题报告

数学符号

符号	意义
$lg(x)$	以 2 为底的对数 $log_2(x)$
$log_b(x)$	以 b 为底的对数 $log_b(x)$
$\lfloor x \rfloor$	下取整函数
$\lceil x \rceil$	上取整函数

GitHub 地址

本书是开源的，GitHub 地址：https://github.com/acprimer/Beauty-of-Programming

目录

第 1 章
游戏之乐

暂无

第 2 章
数学之魅

2.1 求二进制数中 1 的个数

问题

对于一个字节（8bit）的无符号整型变量，求其二进制表示中“1”的个数，要求算法的执行效率尽可能高。

解法一

利用模运算求取 x 的每个二进制位上是否为 1。

```
// 时间复杂度 O(lg(x)), 空间复杂度 O(1)
int count_1(int x) {
    int ans = 0;
    while (x != 0) {
        ans += (x % 2);
        x /= 2;
    }
    return ans;
}
```

解法二

利用位运算求取 x 的每个二进制位上是否为 1。位运算比模运算效率要高。

```
// 时间复杂度 O(lg(x)), 空间复杂度 O(1)。
int count_2(int x) {
    int ans = 0;
    while (x != 0) {
        ans += (x & 0x01);
        x >>= 1;
    }
    return ans;
}
```

解法三

利用按位与 x&(x-1) 每次消除一个 1，直到 x 为 0。

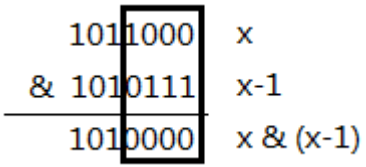

图 2-1 x & (x-1)

```
// 时间复杂度 O(M)，空间复杂度 O(1)，其中 M 为二进制中 1 的个数。
int count_3(int x) {
    int ans = 0;
    while (x != 0) {
        x &= (x - 1);
        ans++;
    }
    return ans;
}
```

扩展问题 1

如果变量是 32 位的 DWORD，你会使用上述的哪一种算法，或者改进哪一个算法？

解法

DWORD 是一个 32 位无符号整数，因此该问题与原问题唯一的区别就是数据的范围不同。上述的三种解法都适用，而且都不要做任何改进。另外对于原书中的解法四使用分支操作和解法五查表法是不适用的。时间复杂度和空间复杂度和原解法相同。

扩展问题 2

给定两个正整数（二进制形式表示）A 和 B，问把 A 变为 B 需要改变多少位（bit）？也就是说，整数 A 和 B 的二进制表示中有多少位是不同的？

解法

拿到这个问题，我们首先想到的就是按照从低到高依次检查 A 和 B 的每一位是否相同，然后记录个数，时间复杂度是 $O(log_2(Max(A,B)))$。接着，我们会想能不能把复杂度降到只和位不同的个数相关。自然的就会想到通过一种位运算把不同的 1 取出来，这种位

运算就是按位异或 ^。计算 C=A^B，再计算 C 中 1 的个数就行了，可以采用原题中的解法三。

```
// 时间复杂度 O(M)，空间复杂度 O(1)，其中 M 为 A 和 B 二进制中不同的个数。
int exercise_2(int A, int B) {
    return count(A ^ B);        // 调用上面的 count 函数，计算 A^B 中 1 的个数。
}
```

扩展问题 3

给定整数 N，判断它是否为 2 的幂，比如 $0, 2, 4, 8, \cdots$ 都是 2 的幂。这是原书中 2.2 的扩展问题，我感觉放在这里比较合适。

解法

```
boolean isPowerOf2(int N) {
    if (N >= 0 && ((N & (N - 1)) == 0)) return true;
    return false;
}
```

2.2 不要被阶乘吓倒

问题

1. 给定一个整数 N，那么 N 的阶乘 $N!$ 末尾有多少个 0 呢？例如，$N = 10$，$N! = 3628800$，$N!$ 的末尾有两个 0。

2. 求 $N!$ 的二进制表示中最低位 1 的位置。

问题 1 解法

对 $N!$ 进行质因数分解得到 $N! = \prod {p_i}^{k_i}$，由于 10=2×5，所以 $N!$ 末尾 0 的个数只和 5 的指数有关。计算 $N!$ 中有多少个 5，可以利用公式 $\lfloor \frac{N}{5} \rfloor + \lfloor \frac{N}{5^2} \rfloor + \lfloor \frac{N}{5^3} \rfloor + \cdots$.

```
// 时间复杂度 O(log5(N))，空间复杂度 O(1)
int count_1(int N) {
    int ans = 0;
        while (N != 0) {
        N /= 5;
        ans += N;
    }
    return ans;
}
```

问题 2 解法一

和问题 1 基本相同，只是现在二进制表示中末尾 0 的个数是和 $N!$ 中 2 的指数有关。利用公式 $\lfloor \frac{N}{2} \rfloor + \lfloor \frac{N}{2^2} \rfloor + \lfloor \frac{N}{2^3} \rfloor + \cdots$.

```
// 时间复杂度 O(lg(N)), 空间复杂度 O(1)
int count_base_2(int N) {
    int ans = 0;
    while (N != 0) {
        N >>= 1;
        ans += N;
    }
    return ans;
}
```

问题 2 解法二

$N!$ 中质因数 2 的个数等于 $N - N$ 的二进制表示中 1 的个数。时间复杂度 O(M)，M 是 N 二进制表示中 1 的个数。这个解法技巧性太强了，不看解答完全想不到，而且扩展性差。

代码略。

扩展问题

求整数 N 的 B 进制表示中末尾 0 的个数。题目来源 UVA 10061.

解法

对 $N!$ 进行质因数分解得到 $N! = \prod {p_i}^{k_i}$，对 B 进行质因数分解得到 $N! = \prod {p_i}^{t_i}$，我们用 $f(N, B)$ 来表示 N 的 B 进制表示中末尾 0 的个数，可以得出下面的等式: $f(N, B) = min\{\frac{k_i}{t_i}\}$.

```
// 时间复杂度 O(lgB(N)), 空间复杂度 O(1)
int countZeros(int n, int b) {
    int ans = Integer.MAX_VALUE;
    for (int i = 2; i <= b; i++) {
        if (b % i != 0) continue;
        int cnt = 0;
        while (b % i == 0) {
            b /= i;
            cnt++;
        }
        int tmp = 0, tn = n;
        while (tn != 0) {
            tn /= i;
            tmp += tn;
```

```
            }
            ans = Math.min(ans, tmp / cnt);
        }
        return ans;
    }
```

这里我们可以考虑一个问题，可不可以直接求出最大的 $p_i{}^{t_i}$，然后计算 $N!$ 中有多少个 p_i。提示：N=4，B=40.

2.3 寻找发帖“水王”

问题

给定一个整型数组，每个数组元素表示一个 ID，数组长度为 N，数组中有一个 ID 出现的次数超过了数组长度的一半，求出这个 ID。

解法 1

最简单的解法就是暴力枚举每一个数组元素，判断该元素是否超过一半。时间复杂度是 $O(N^2)$。

然后我们考虑对数组进行排序（很多问题一旦排序之后就会有思路了），这时候数组的中位数一定就是我们要求的 ID。时间复杂度是 $O(Nlg(N))$。

最后，我们考虑这样一个场景，让不同的 ID 进行 PK，最后剩下的一定就是超过一半的 ID。可以用一个栈来模拟数组中的元素 PK 的过程。

```
// 时间复杂度 O(N)，空间复杂度 O(N)
int findMostWithStack(int[] ids) {
    Stack<Integer> stack = new Stack<Integer>();
    for(int i=0;i<ids.length;i++) {
        if(stack.isEmpty()) {
            stack.push(ids[i]);
                } else {
        if(stack.peek()==ids[i]) stack.push(ids[i]);
        else stack.pop();
    }
}
return stack.peek();
}
```

解法 2

下面我们就来考虑一下是否可以降低空间复杂度。其实上面的做法中我们对于栈中的元素并不关心，我们只关心当前栈顶元素和栈的大小，所以我们就可以不用栈来处理了，用两个变量记录栈的大小和栈顶元素即可。

```
// 时间复杂度 O(N), 空间复杂度 O(1)
public int findMost(int[] ids) {
    int candidate = -1, count = 0;
    for (int i = 0; i < ids.length; i++) {
        if (count == 0) {
            candidate = ids[i];
            count = 1;
        } else {
            if (ids[i] == candidate) count++;
            else count--;
        }
    }
    return candidate;
}
```

解法 3

利用哈希表。

扩展问题 1

原问题中保证一定存在一个元素超过一半，那么如果去掉这个条件呢？求出这个元素的下标（数组下标从 0 开始），如果不存在则返回 -1.

解法

如果不存在超过一半的元素，上面的两个解法其实并不能正确返回。比如 [1,2,3]，返回的就是 3. 其实这里要做的就是对这个得到的结果进行判断出现次数是否超过一半，时间和空间复杂度都不改变。

代码略。

扩展问题 2

有 3 个元素出现的次数超过了数组长度的 $\frac{1}{4}$，找出这 3 个元素。

解法

先排序再查找是可以找出的，时间复杂度是 $O(Nlg(N))$，这里就不给出具体代码了。