【华为OD机考 统一考试机试C卷】数字游戏（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

小扇和小船今天又玩起来了数字游戏，小船给小扇一个正整数 n（1 ≤ n ≤ 1e9），小扇需要找到一个比 n 大的数字 m，使得 m 和 n 对应的二进制中 1 的个数要相同，如：

- 4对应二进制100
- 8对应二进制1000
- 其中1的个数都为1个

现在求 m 的最小值。

输入描述

输入一个正整数 n（1 ≤ n ≤ 1e9）

输出描述

输出一个正整数 m

用例

输入	2
输出	4
说明	2的二进制10，4的二进制位100，1的个数相同，且4是满足条件的最小数

输入	7
输出	11
说明	7的二进制111，11的二进制位1011，1的个数相同，且11是满足条件的最小数

解题思路

1. **理解题目要求**：给定一个正整数 `n`，找到一个比 `n` 大的最小正整数 `m`，使得 `m` 和 `n` 的二进制表示中 `1` 的个数相同。
2. **观察二进制规律**：在二进制数中，找到一个比当前数大的数，通常需要将一个较低位的 `0` 变成 `1`。同时，为了确保这个新的数尽可能小，我们希望这个 `0` 尽可能靠右，而且这个 `0` 右边的 `1` 尽可能少。
3. **找到关键位点**：从低位到高位，找到第一个 `01` 模式的位置（即一个 `1` 后面紧跟着一个 `0`），这个 `01` 将被翻转成 `10`。这样做会增加数值，同时保持 `1` 的总数不变。
4. **翻转位**：将找到的 `01` 模式翻转成 `10`。这可以通过将 `0` 的位置设为 `1`（即 `n |= (1 << p)`），然后将该位右边的所有位清零（即 `n &= ~((1 << p) - 1)`）来实现。
5. **调整右侧位**：由于我们已经将一个 `0` 变成了 `1`，为了保持 `1` 的总数不变，我们需要在右侧插入 `c1 - 1` 个 `1`，这里 `c1` 是 `01` 模式左侧 `1` 的数量。这可以通过 `n |= (1 << (c1 - 1)) - 1` 来实现。
6. **输入为4 4**：对于输入 `n = 4`，其二进制表示为 `100`。按照上述步骤，我们需要找到第一个 `01` 模式，但是在 `100` 中不存在 `01` 模式。因此，我们需要在最低位添加一个 `1`，并且将最左边的 `1` 向右移动一位，得到 `1000`，即 `8`。这里 `8` 是比 `4` 大的下一个数，且二进制中 `1` 的数量相同。

C++

```cpp
#include <iostream>
using namespace std;
// 寻找下一个具有相同数量的1的数字
int findNextNumberWithSameNumberOfOnes(int n) {
    int c0 = 0, c1 = 0;
    int temp = n;

```

```
    // 统计 "01" 模式中 0 的个数
    while ((temp & 1) == 0 && temp != 0) {
        c0++;
        temp >>= 1;
    }

    // 统计 "01" 模式中 1 的个数
    while ((temp & 1) == 1) {
        c1++;
        temp >>= 1;
    }

    // 如果 n 是形如 "111...11000...0" 的数，则没有 "01" 模式
    if (c0 + c1 == 31 || c0 + c1 == 0) {
        return -1;
    }

    // p 是我们要翻转的 "01" 模式的位置
    int p = c0 + c1;

    // 翻转 "01" 为 "10"
    n |= (1 << p);        // 将 p 位置设为 1
    n &= ~((1 << p) - 1); // 清除 p 位右边的所有位
    n |= (1 << (c1 - 1)) - 1; // 在 p 位右边插入 (c1-1) 个 1

    return n;
}

int main() {
    int n;
    cin >> n;
    int m = findNextNumberWithSameNumberOfOnes(n);
    cout << m << endl;
    return 0;
}
```

Java

```
import java.util.Scanner;

```

```java
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = findNextNumberWithSameNumberOfOnes(n);
        System.out.println(m);
        scanner.close();
    }

    private static int findNextNumberWithSameNumberOfOnes(int n) {
        // c0 表示在找到的 "01" 模式中 0 的个数
        // c1 表示在找到的 "01" 模式中 1 的个数
        int c0 = 0, c1 = 0;
        int temp = n;

        // 统计 "01" 模式中 0 的个数
        while (((temp & 1) == 0) && (temp != 0)) {
            c0++;
            temp >>= 1;
        }

        // 统计 "01" 模式中 1 的个数
        while ((temp & 1) == 1) {
            c1++;
            temp >>= 1;
        }

        // 如果 n 是形如 "111...11000...0" 的数，则没有 "01" 模式
        if (c0 + c1 == 31 || c0 + c1 == 0) {
            return -1;
        }

        // p 是我们要翻转的 "01" 模式的位置
        int p = c0 + c1;

        // 翻转 "01" 为 "10"
        // 第一步：将 p 位置为 1（即将 "01" 的 "0" 翻转为 "1"）
        n |= (1 << p);

        // 第二步：清除 p 位右边的所有位（即将 "01" 后面的所有位清零）
```

```
        // 创建一个掩码，它在 p 位之前都是 1，然后取反，得到 p 位及其右边都是 0 的掩码
        int mask = ~((1 << p) - 1);
        n &= mask;

        // 第三步：在 p 位右边插入 (c1-1) 个 1（即将 "01" 前面的 "1" 移动到 p 位右边）
        // 创建一个序列，其中包含 (c1-1) 个 1，然后将这个序列放在 p 位的右边
        int ones = (1 << (c1 - 1)) - 1;
        n |= ones;

        return n;
    }
}
```

javaScript

```
// 寻找下一个具有相同数量的1的数字
function find(n) {
    let c0 = 0, c1 = 0;
    let temp = n;

    // 统计 "01" 模式中 0 的个数
    while ((temp & 1) === 0 && temp !== 0) {
        c0++;
        temp >>= 1;
    }

    // 统计 "01" 模式中 1 的个数
    while ((temp & 1) === 1) {
        c1++;
        temp >>= 1;
    }

    // 如果 n 是形如 "111...11000...0" 的数，则没有 "01" 模式
    if (c0 + c1 === 31 || c0 + c1 === 0) {
        return -1;
    }

    // p 是我们要翻转的 "01" 模式的位置
    let p = c0 + c1;

```

```
    // 翻转 "01" 为 "10"
    n |= (1 << p);        // 将 p 位置设为 1
    n &= ~((1 << p) - 1); // 清除 p 位右边的所有位
    n |= (1 << (c1 - 1)) - 1; // 在 p 位右边插入 (c1-1) 个 1

    return n;
}

// 读取输入并输出结果
const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

readline.on('line', (n) => {
    const m = find(parseInt(n, 10));
    console.log(m);
    readline.close();
});
```

Python

```
def find(n):
    c0, c1 = 0, 0
    temp = n

    # 统计 "01" 模式中 0 的个数
    while (temp & 1) == 0 and temp != 0:
        c0 += 1
        temp >>= 1

    # 统计 "01" 模式中 1 的个数
    while (temp & 1) == 1:
        c1 += 1
        temp >>= 1

    # 如果 n 是形如 "111...11000...0" 的数，则没有 "01" 模式
    if c0 + c1 == 31 or c0 + c1 == 0:
        return -1

```

```python
    # p 是我们要翻转的 "01" 模式的位置
    p = c0 + c1

    # 翻转 "01" 为 "10"
    n |= (1 << p)       # 将 p 位置设为 1
    n &= ~((1 << p) - 1) # 清除 p 位右边的所有位
    n |= (1 << (c1 - 1)) - 1 # 在 p 位右边插入 (c1-1) 个 1

    return n

# 读取输入并输出结果
n = int(input())
m = find(n)
print(m)
```

C语言

```c
#include <stdio.h>

// 寻找下一个具有相同数量1的数字的函数
int findNextNumberWithSameNumberOfOnes(int n) {
    int c0 = 0, c1 = 0; // 分别用于统计0和1的个数
    int temp = n;

    // 统计 "01" 模式中0的个数
    while ((temp & 1) == 0 && temp != 0) {
        c0++;
        temp >>= 1;
    }

    // 统计 "01" 模式中1的个数
    while ((temp & 1) == 1) {
        c1++;
        temp >>= 1;
    }

    // 如果n是形如 "111...11000...0" 的数，则没有 "01" 模式
    if (c0 + c1 == 31 || c0 + c1 == 0) {
        return -1;
    }
```

```
    // p是要翻转的 "01" 模式的位置
    int p = c0 + c1;

    // 翻转 "01" 为 "10"
    n |= (1 << p);        // 将p位置设为1
    n &= ~((1 << p) - 1); // 清除p位右边的所有位
    n |= (1 << (c1 - 1)) - 1; // 在p位右边插入(c1-1)个1

    return n;
}

int main() {
    int n;
    scanf("%d", &n); // 读取输入的整数n
    int m = findNextNumberWithSameNumberOfOnes(n); // 调用函数查找m
    printf("%d\n", m); // 输出结果m
    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师