【模拟】-分糖果

题目描述与示例

题目描述

小明从糖果盒中随意抓一把糖果，每次小明会取出一半的糖果分给同学们。

当糖果不能平均分配时，小明可以选择从糖果盒中（假设盒中糖果足够）取出一个糖果或放回一个糖果。

小明最少需要多少次（取出、放回和平均分配均记一次），能将手中糖果分至只剩一颗。

输入描述

抓取的糖果数（`<10000000000`）

输出描述

最少分至一颗糖果的次数

示例一

输入

```
15
```

输出

```
1 5
```

说明

`15+1=16`

`16/2=8`

`8/2=4`

`4/2=2`

`2/2=1`

示例二

输入

```
1 6
```

输出

```
1 3
```

说明

`6/2=3`

`3-1=2`

`2/2=1`

解题思路

本题是一道比较有意思的题目。从题意描述到解题思路，个人都比较喜欢。

本题的数据量看似给得非常大（`N < 10000000000`），但是仔细思考可以发现，每一次我们都会**近似地让糖果数量减半**，因此糖果数减少到 `1` 的速度是对数级别的。

我们可以直接使用一个 `O(logN)` 的模拟算法来解决这个问题。

考虑某一个特定的糖果数量 `num`，且到达 `num` 的操作次数为 `time`。当

- `num` 是偶数的时候，我们获得数量 `num // 2` 的糖果，需要 `time + 1` 次操作。这里的 `1` 次操作，是减半操作。
- `num` 是奇数的时候，我们获得数量 `(num+1) // 2` 或 `(num-1) // 2` 的糖果，需要 `time + 2` 次操作。这里的 `2` 次操作，有 `1` 次是减半操作，有 `1` 次是 `+1` 或 `-1` 的操作。

显然，对于任意的 `num`，如果已知对应的 `time`，我们就可以计算出**其近似减半后的结果**。

而近似减半后的结果得到之后，我们又可以做类似的重复操作，直到最终糖果数量减少到 `1`。

这种重复的过程，容易想到可以使用递归来完成。考虑递归三要素：

1. 递归入口：传入初始值的当前数量 `num = n`，以及初始操作次数 `time = 0`
2. 递归子问题：`num` 的减半过程，也就是上述的讨论。
3. 递归终止条件：当 `num` 减少到 `1` 的时候，可以更新答案。

> 本题的递归过程和动态规划非常类似，可以多加以比较。

代码

Python

```python
# 题目：【模拟】2024E-分糖果
# 分值：100
# 作者：许老师-闭着眼睛学数理化
```

```python
# 算法：模拟，递归
# 代码看不懂的地方，请直接在群上提问


from math import inf

# 输入初始数量n
n = int(input())
# 初始化答案变量ans为无穷大
ans = inf

# 构建递归辅助函数，num为当前糖果数，time为得到num所花费的操作次数
def help(num, time):
    # 设置ans为全局变量
    global ans
    # 如果num为1，则直接更新ans
    if num == 1:
        ans = min(ans, time)
        return
    # 否则，进行递归
    else:
        # 如果num是偶数，则获得 num // 2的操作次数为time+1
        if num % 2 == 0:
            help(num//2, time+1)
        # 如果num是奇数，则获得 (num+1)//2 和 (num-1)//2的操作次数为time+2
        else:
            help((num+1)//2, time+2)
            help((num-1)//2, time+2)

# 递归入口，传入num = n以及time = 0
help(n, 0)

# 输出答案
print(ans)
```

Java

```java
import java.util.Scanner;

public class Main {

    // 初始化答案变量ans为无穷大
    private static long ans = Long.MAX_VALUE;

    // 构建递归辅助函数，num为当前糖果数，time为得到num所花费的操作次数
```

```java
    public static void help(long num, long time) {
        // 如果num为1，则直接更新ans
        if (num == 1) {
            ans = Math.min(ans, time);
            return;
        }
        // 否则，进行递归
        else {
            // 如果num是偶数，则获得 num // 2的操作次数为time+1
            if (num % 2 == 0) {
                help(num / 2, time + 1);
            }
            // 如果num是奇数，则获得 (num+1)//2 和 (num-1)//2的操作次数为time+2
            else {
                help((num + 1) / 2, time + 2);
                help((num - 1) / 2, time + 2);
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入初始数量n
        long n = scanner.nextLong();

        // 递归入口，传入num = n以及time = 0
        help(n, 0);

        // 输出答案
        System.out.println(ans);

        scanner.close();
    }
}

```

C++

```cpp
#include <iostream>
#include <climits>

using namespace std;

// 初始化答案变量ans为无穷大
```

```cpp
long long ans = LLONG_MAX;

// 构建递归辅助函数，num为当前糖果数，time为得到num所花费的操作次数
void help(long long num, long long time) {
    // 如果num为1，则直接更新ans
    if (num == 1) {
        ans = min(ans, time);
        return;
    }
    // 否则，进行递归
    else {
        // 如果num是偶数，则获得 num // 2的操作次数为time+1
        if (num % 2 == 0) {
            help(num / 2, time + 1);
        }
        // 如果num是奇数，则获得 (num+1)//2 和 (num-1)//2的操作次数为time+2
        else {
            help((num + 1) / 2, time + 2);
            help((num - 1) / 2, time + 2);
        }
    }
}

int main() {
    // 输入初始数量n
    long long n;
    cin >> n;

    // 递归入口，传入num = n以及time = 0
    help(n, 0);

    // 输出答案
    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(logN)`。

空间复杂度：`O(1)`。