HDOJ 2842 - Chinese Rings

——by A Code Rabbit

Description

中国环……

中国环是一串环，你要解开第 n 个环的话，必须先将前 n – 2 个环脱下来，第 n – 1 个环仍在串上，然后才可以去脱下或者串上第 n 个环。

第 1 个环可以随便操作。

输入环的数量 n。

输出最少几步可以把所有环都脱下来。

Types

Maths :: Matrix

Analysis

这种题一看就欠打表。

欠打表的题一般都是递推。

所以这题是递推……

我们设 f(x)为把前 x 个的环脱下来的最少步骤，f(n)就是我们要求的解。

然后我们来思考，如何推出 f(x)。

要把前 x 个环脱下来，就要先把前 x – 2 个环脱下来，然后脱下第 x 个环，

这一步要花费 f(x – 2) + 1 步。

然后我们要考虑如何把第 x + 1 个环脱下来。

由于一个环可以脱下来的时候，也可以套上去，这个过程可逆。

所以，我们可以花费 f(x – 2)步，把前 x – 2 个环套上去。

这时候我们发现，我们的任务转化为如何把前 x – 1 个环脱下来，即 f(x – 1)。

这时候，便得到我们的递推公式：

- f(x) = f(x – 2) + 2 * f(x – 1) + 1

由于 n 很大，我们应该用矩阵乘法 + 快速幂求解。

Solution

```
// HDOJ 2842
// Chinese Rings
// by A Code Rabbit

#include <cstdio>
```

```
const int ORDER = 3;
const int DIVISOR = 200907;

struct Matrix {
    long long element[ORDER][ORDER];
};

int n;

Matrix mat_unit;
Matrix mat_one;
Matrix mat_ans;

int original_solution[] = {
    2, 1, 1,
};

void INIT();

Matrix QuickPower(Matrix mat_result, Matrix mat_one, int index);
Matrix Multiply(Matrix mat_a, Matrix mat_b);

int main() {
    while (scanf("%d", &n), n) {
        INIT();
        if (n <= 2) {
            printf("%d\n", original_solution[2 - n]);
            continue;
        } else {
            mat_ans = QuickPower(mat_unit, mat_one, n - 2);
            int sum = 0;
            for (int i = 0; i < ORDER; ++i) {
                sum += original_solution[i] * mat_ans.element[i][0];
            }
            printf("%d\n", sum % DIVISOR);
        }
    }

    return 0;
}
```

```
void INIT() {
    for (int i = 0; i < ORDER; ++i) {
        for (int j = 0; j < ORDER; ++j) {
            mat_unit.element[i][j] = i == j ? 1 : 0;
        }
    }
    for (int i = 0; i < ORDER; ++i) {
        for (int j = 0; j < ORDER; ++j) {
            mat_one.element[i][j] = 0;
        }
    }
    mat_one.element[0][0] = 1;
    mat_one.element[0][1] = 1;
    mat_one.element[1][0] = 2;
    mat_one.element[2][0] = 1;
    mat_one.element[2][2] = 1;
}

Matrix QuickPower(Matrix mat_result, Matrix mat_one, int index) {
    while (index) {
        if (index & 1) {
            mat_result = Multiply(mat_result, mat_one);
        }
        mat_one = Multiply(mat_one, mat_one);
        index >>= 1;
    }
    return mat_result;
}

Matrix Multiply(Matrix mat_a, Matrix mat_b) {
    Matrix mat_result;
    for (int i = 0; i < ORDER; ++i) {
        for (int j = 0; j < ORDER; ++j) {
            mat_result.element[i][j] = 0;
            for (int k = 0; k < ORDER; ++k) {
                mat_result.element[i][j] += mat_a.element[i][k] * mat_b.element[k][j];
            }
            mat_result.element[i][j] %= DIVISOR;
        }
    }
    return mat_result;
```

```
}
```