HDOJ 1588 – Gauss Fibonacci

——by A Code Rabbit

Description

g(x) = k * x + b。
f(x) 为 Fibonacci 数列。
求 f(g(x))，从 x = 1 到 n 的数字之和，并对 m 取模。

Types

Maths :: Matrix

Analysis

我们知道 f(x)中，两个元素之间的关系是

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}$$

因为 g(x) – b，为一个等比数列，所以，他们之间也有一个类似 Fibonacci 的关系，并且可以用矩阵来表示

$$\left|\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}\right|^{k}$$

而为了求和，我们可以添加一项 s(x)，用来表示前 x 项的和，最后得到矩阵
(s(x-1),f(g(x)),f(g(x)-1)) = (s(x – 2), f(g(x – 1), f(g(x – 1) – 1)))

$$\begin{pmatrix}1 & 0 & 0\\ \begin{matrix}1\\ 0\end{matrix} & \multicolumn{2}{c}{\left|\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}\right|^{k}}\end{pmatrix}$$

剩下的工作就是矩阵乘法和快速幂了。

Solution

```
// HDOJ 1588
// Gauss Fibonacci
// by A Code Rabbit

```

```
#include <cstdio>

int k, b, n, m;

struct Matrix {
    long long element[3][3];
};

const Matrix mat_unit = {
    1, 0, 0,
    0, 1, 0,
    0, 0, 1,
};

const Matrix mat_zero = {
    0, 0, 0,
    0, 0, 0,
    0, 0, 0,
};

Matrix mat_one;
Matrix mat_ans;

void INIT1();
void INIT2();

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

int Fibonacci(int x);

int main() {
    while (scanf("%d%d%d%d", &k, &b, &n, &m) != EOF) {
        // Initialize in the first time.
        INIT1();
        // Quick power for competing the matrix of the relation of two nest pair of
numbers in g(x).
        mat_ans = QuickPower(mat_ans, mat_one, k, 2);
        // Initialize in the second time.
        INIT2();
        // Quick power for competing the sum of g(x).
```

```
        mat_ans = QuickPower(mat_ans, mat_one, n - 1, 3);
        // Outputs.
        int original_solution[] = {
            Fibonacci(b),
            Fibonacci(k + b),
            Fibonacci(k + b - 1),
        };
        long long sum = 0;
        for (int i = 0; i < 3; ++i)
            sum += original_solution[i] * mat_ans.element[i][0];
        printf("%lld\n", sum % m);
    }

    return 0;
}

void INIT1() {
    mat_one.element[0][0] = 1;
    mat_one.element[0][1] = 1;
    mat_one.element[1][0] = 1;
    mat_one.element[1][1] = 0;
    mat_ans = mat_unit;
}

void INIT2() {
    mat_one = mat_zero;
    mat_one.element[0][0] = 1;
    mat_one.element[1][0] = 1;
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j) {
            mat_one.element[i + 1][j + 1] = mat_ans.element[i][j];
        }
    }
    mat_ans = mat_unit;
}

Matrix Multiply(Matrix mat_a, Matrix mat_b, int order) {
    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] %= m;
        }
    }
    return mat_result;
}

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

int Fibonacci(int x) {
    if (!x) {
        return 0;
    }
    Matrix mat_ans;
    Matrix mat_one;
    // Initialize mat_ans.
    mat_ans.element[0][0] = 1;
    mat_ans.element[0][1] = 0;
    mat_ans.element[1][0] = 0;
    mat_ans.element[1][1] = 1;
    // Initialize mat_one.
    mat_one.element[0][0] = 1;
    mat_one.element[0][1] = 1;
    mat_one.element[1][0] = 1;
    mat_one.element[1][1] = 0;
    // Quick power.
    mat_ans = QuickPower(mat_ans, mat_one, x, 2);
    // Compete and return the result.
    return mat_ans.element[1][0];
}
```