HDOJ 1575 – Tr A

——by A Code Rabbit

Description

求矩阵高次幂求余后主对角线上的和。

Types

Maths :: Matrix

Analysis

矩阵乘法和快速幂，其实快速幂是废话，矩阵乘法一般都会结合快速幂。

矩阵乘法要注意，初始化一个矩阵的时候，要按需求化为单位矩阵或者零矩阵。

一般单位矩阵用于乘法，而零矩阵用于加法。

写矩阵乘法的时候要细心，除了初始化还有蛮多点要注意的，好在代码比较简单，debug 比较容易。

而快速幂，推荐用循环 + 位运算的方式，代码量少，速度快。

对于是面向对象来写，还是面向过程来写，我两种都写了，还是面向过程比较适合 ACM，面向对象过于冗长，而且他的好处也不是体现得很明显。

Solution

```
1. 面向对象的写法
// HDOJ 1575
// Tr A
// by A Code Rabbit

#include <cstdio>
#include <cstring>

const int LIMITS = 12;
const int DIVISOR = 9973;

class Matrix{
public:
    Matrix();
    Matrix(int);
    void Read();
    Matrix operator*(Matrix);
    Matrix operator=(Matrix);
    Matrix operator%(int);
    int GetTrace();
```

```
private:
    int element[LIMITS][LIMITS];
    int order;
};

int t;

int n, k;


int main() {
    scanf("%d", &t);
    while (t--) {
        // Inputs.
        scanf("%d%d", &n, &k);
        Matrix matrix_one(n);
        matrix_one.Read();
        // Quick Power.
        Matrix matrix_ans(n);
        while (k) {
            if (k & 1) {
                matrix_ans = (matrix_ans * matrix_one) % DIVISOR;
            }
            k >>= 1;
            matrix_one = (matrix_one * matrix_one) % DIVISOR;
        }
        // Outputs.
        printf("%d\n", matrix_ans.GetTrace());
    }

    return 0;
}

Matrix::Matrix() {
}

Matrix::Matrix(int order) {
    this->order = order;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            this->element[i][j] = i == j ? 1 : 0;
```

```
        }
    }
}

void Matrix::Read() {
    for (int i = 0; i < this->order; ++i) {
        for (int j = 0; j < this->order; ++j) {
            scanf("%d", &this->element[i][j]);
        }
    }
}

Matrix Matrix::operator*(Matrix one) {
    Matrix result(n);
    for (int i = 0; i < this->order; ++i) {
        for (int j = 0; j < this->order; ++j) {
            int sum = 0;
            for (int k = 0; k < this->order; ++k) {
                sum += this->element[i][k] * one.element[k][j];
            }
            result.element[i][j] = sum;
        }
    }
    return result;
}

Matrix Matrix::operator=(Matrix one) {
    for (int i = 0; i < this->order; ++i) {
        for (int j = 0; j < this->order; ++j) {
            this->element[i][j] = one.element[i][j];
        }
    }
    return *this;
}

Matrix Matrix::operator%(int num) {
    for (int i = 0; i < this->order; ++i) {
        for (int j = 0; j < this->order; ++j) {
            this->element[i][j] %= num;
        }
    }
```

```
    return *this;
}

int Matrix::GetTrace() {
    int sum = 0;
    for (int i = 0; i < this->order; ++i) {
        sum = (sum + this->element[i][i]) % DIVISOR;
    }
    return sum;
}
```

2. 面向过程的写法

```
// HDOJ 1575
// Tr A
// by A Code Rabbit

#include <cstdio>
#include <cstring>

const int LIMITS = 12;
const int DIVISOR = 9973;

struct Matrix {
    int element[LIMITS][LIMITS];
};

int t;

Matrix matrix_one;
int n, k;

Matrix matrix_ans;

void Multiply(Matrix& a, Matrix b);
int GetTrace(Matrix a);

int main() {
    scanf("%d", &t);
    while (t--) {
        // Inputs.
        scanf("%d%d", &n, &k);
```

```
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                scanf("%d", &matrix_one.element[i][j]);
            }
        }
        // Unitization.
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                matrix_ans.element[i][j] = i == j ? 1 : 0;
            }
        }
        // Quick Power.
        while (k) {
            if (k & 1) {
                Multiply(matrix_ans, matrix_one);
            }
            k >>= 1;
            Multiply(matrix_one, matrix_one);
        }
        // Outputs.
        printf("%d\n", GetTrace(matrix_ans));
    }

    return 0;
}

void Multiply(Matrix& a, Matrix b) {
    Matrix c;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            c.element[i][j] = 0;
            for (int k = 0; k < n; ++k) {
                c.element[i][j] += a.element[i][k] * b.element[k][j];
            }
            c.element[i][j] %= DIVISOR;
        }
    }
    a = c;
}

int GetTrace(Matrix a) {
```

```
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += a.element[i][i];
    }
    return sum % DIVISOR;
}
```