4.7 節末問題 4.7 の解答

問題 4.7.1

まず、行列の掛け算は以下のようになります（行列も整数と同様、掛け算を優先して計算します）。分からない人は、4.7.3 項に戻って確認しましょう。

$$\begin{bmatrix}1 & 0 & 1\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 1\\ 1 & 1 & 1\\ 1 & 0 & 1\end{bmatrix}=\begin{bmatrix}2 & 0 & 2\\ 1 & 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}1\\ 2\end{bmatrix}\begin{bmatrix}1 & 1 & 1\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\ 2 & 2 & 2\end{bmatrix}$$

求める答えは、青く示されている 2 つの 2×3 行列の和なので、以下の通りです。

$$\begin{bmatrix}2 & 0 & 2\\ 1 & 1 & 1\end{bmatrix}+\begin{bmatrix}1 & 1 & 1\\ 2 & 2 & 2\end{bmatrix}=\begin{bmatrix}3 & 1 & 3\\ 3 & 3 & 3\end{bmatrix}$$

問題 4.7.2

まず、$a_3=2a_2+a_1$、$a_2=a_2$ であることから、以下の式が成り立ちます。

$$\begin{bmatrix}a_3\\ a_2\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}a_2\\ a_1\end{bmatrix}$$

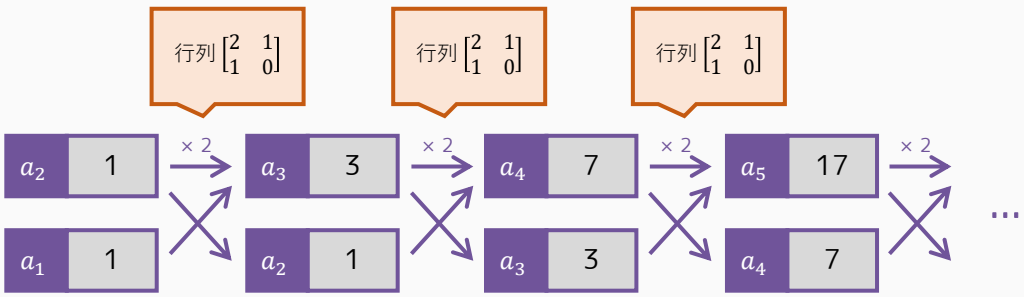
同様に、$a_4=2a_3+a_2$、$a_3=a_3$ であることから、以下の式が成り立ちます。

$$\begin{bmatrix}a_4\\ a_3\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}a_3\\ a_2\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}a_3\\ a_2\end{bmatrix}$$

同じような計算を a_5 以降についても繰り返すと、以下のようになります。このことから、A を $2,1,1,0$ からなる行列とするとき、a_N の値が A^{N-1} の $(2,1)$ 成分と $(2,2)$ 成分を足した値であることが分かります。

$$\begin{bmatrix}a_{N+1}\\ a_N\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 0\end{bmatrix}^N\begin{bmatrix}a_2\\ a_1\end{bmatrix}=A^{N-1}\begin{bmatrix}1\\ 1\end{bmatrix}$$

以下の図は、行列と数列 $a = (a_1, a_2, a_3, \ldots)$ の関係を表したものです。



したがって、以下のような実装をすると、正しい答えを出すことができます。なお、コード 4.7.1 と異なる部分を濃い青色で示しています。（逆にいえば、このプログラムはコード 4.7.1 とほぼ同一です。）

```
#include <iostream>
using namespace std;

struct Matrix {
    long long p[2][2] = { {0, 0}, {0, 0} };
};

Matrix Multiplication(Matrix A, Matrix B) { // 2×2 行列 A, B の積を返す関数
    Matrix C;
    for (int i = 0; i < 2; i++) {
        for (int k = 0; k < 2; k++) {
            for (int j = 0; j < 2; j++) {
                C.p[i][j] += A.p[i][k] * B.p[k][j];
                C.p[i][j] %= 1000000007;
            }
        }
    }
    return C;
}

Matrix Power(Matrix A, long long n) { // A の n 乗を返す関数
    Matrix P = A, Q;
    bool flag = false;
    for (int i = 0; i < 60; i++) {
        if ((n & (1LL << i)) != 0LL) {
            if (flag == false) { Q = P; flag = true; }
            else { Q = Multiplication(Q, P); }
        }
        P = Multiplication(P, P);
    }
    return Q;
}

int main() {
    // 入力 → 累乗の計算（N が 2 以上でなければ正しく動作しないので注意
```

```
    long long N;
    cin >> N;

    // 行列 A の作成
    Matrix A;
    A.p[0][0] = 2; A.p[0][1] = 1; A.p[1][0] = 1;

    // B=A^{N-1} の計算
    Matrix B = Power(A, N - 1);

    // 答えの出力
    cout << (B.p[1][0] + B.p[1][1]) % 1000000007 << endl;
    return 0;
}
```

※ Python などのソースコードは chap4-7.md をご覧ください。

問題 4.7.3

まず、$a_4 = a_3 + a_2 + a_1, a_3 = a_3, a_2 = a_2$ より、以下の式が成り立ちます。

$$\begin{bmatrix} a_4 \\ a_3 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix}$$

同様に、$a_5 = a_4 + a_3 + a_2, a_4 = a_4, a_3 = a_3$ より、以下の式が成り立ちます。

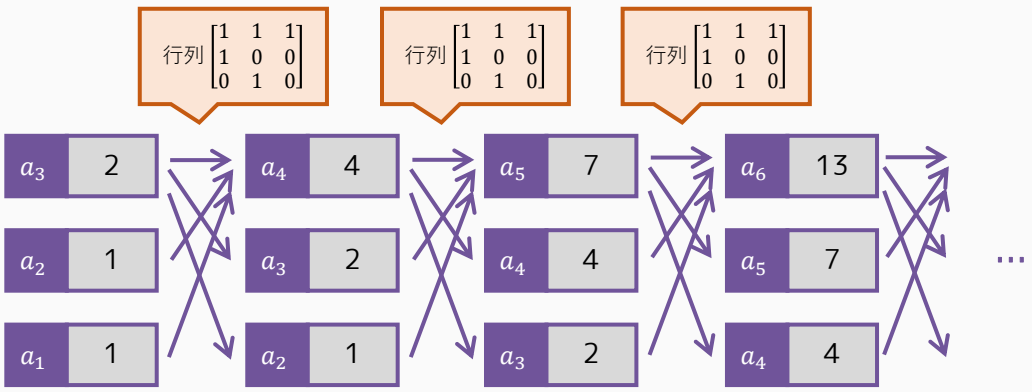
$$\begin{bmatrix} a_5 \\ a_4 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_4 \\ a_3 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^2 \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix}$$

同じような計算を a_6 以降も繰り返すと、以下のようになります。ただし、$1, 1, 1, 1, 0, 0, 0, 1, 0$ からなる 3×3 行列を A とします。

$$\begin{bmatrix} a_{N+2} \\ a_{N+1} \\ a_N \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^{N-1} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = A^{N-1} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

このことから、a_N の値が $[A^{N-1}$ の $(3,1)$ 成分 $\times 2 + (3,2)$ 成分 $+ (3,3)$成分$]$ で表されることが分かります。

以下の図は、行列と数列 $a = (a_1, a_2, a_3, \ldots)$ の関係を表したものです。



したがって、以下のような実装をすると、正しい答えを出すことができます。なお、行列の大きさが 3×3 になっていることに注意してください。（コード 4.7.1 と異なる部分を濃い青色で示しています。）

```
#include <iostream>
using namespace std;

struct Matrix {
    long long p[3][3] = { {0, 0, 0}, {0, 0, 0}, {0, 0, 0} };
};

Matrix Multiplication(Matrix A, Matrix B) { // 3×3 行列 A, B の積を返す関数
    Matrix C;
    for (int i = 0; i < 3; i++) {
        for (int k = 0; k < 3; k++) {
            for (int j = 0; j < 3; j++) {
                C.p[i][j] += A.p[i][k] * B.p[k][j];
                C.p[i][j] %= 1000000007;
            }
        }
    }
    return C;
}

Matrix Power(Matrix A, long long n) { // A の n 乗を返す関数
    Matrix P = A, Q;
    bool flag = false;
    for (int i = 0; i < 60; i++) {
        if ((n & (1LL << i)) != 0LL) {
            if (flag == false) { Q = P; flag = true; }
            else { Q = Multiplication(Q, P); }
        }
        P = Multiplication(P, P);
    }
    return Q;
}
```

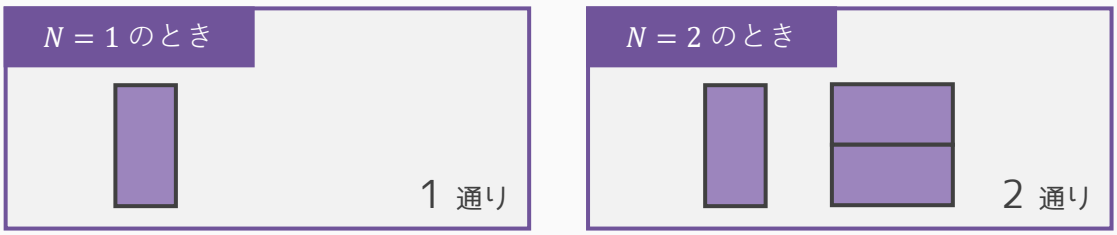
```cpp
int main() {
    // 入力 → 累乗の計算（N が 2 以上でなければ正しく動作しないので注意）
    long long N;
    cin >> N;

    // 行列 A の作成
    Matrix A;
    A.p[0][0] = 1; A.p[0][1] = 1; A.p[0][2] = 1; A.p[1][0] = 1; A.p[2][1] = 1;

    // B=A^{N-1} の計算
    Matrix B = Power(A, N - 1);

    // 答えの出力
    cout << (2LL * B.p[2][0] + B.p[2][1] + B.p[2][2]) % 1000000007 << endl;
    return 0;
}
```

※ Python などのソースコードは chap4-7.md をご覧ください。

問題 4.7.4 (1)

$2 \times k$ の長方形を左から順番に敷き詰めていくとき、最後に置くピースは以下の 2 つのうちいずれかになります。（$k \geq 2$ の場合）

したがって、$2 \times k$ の長方形を敷き詰める方法の数を a_k とすると、$a_k = a_{k-1} + a_{k-2}$ という漸化式が成り立ちます。また、$N = 1$ のときの答えは $a_1 = 1$、$N = 2$ のときの答えは $a_2 = 1$ です。

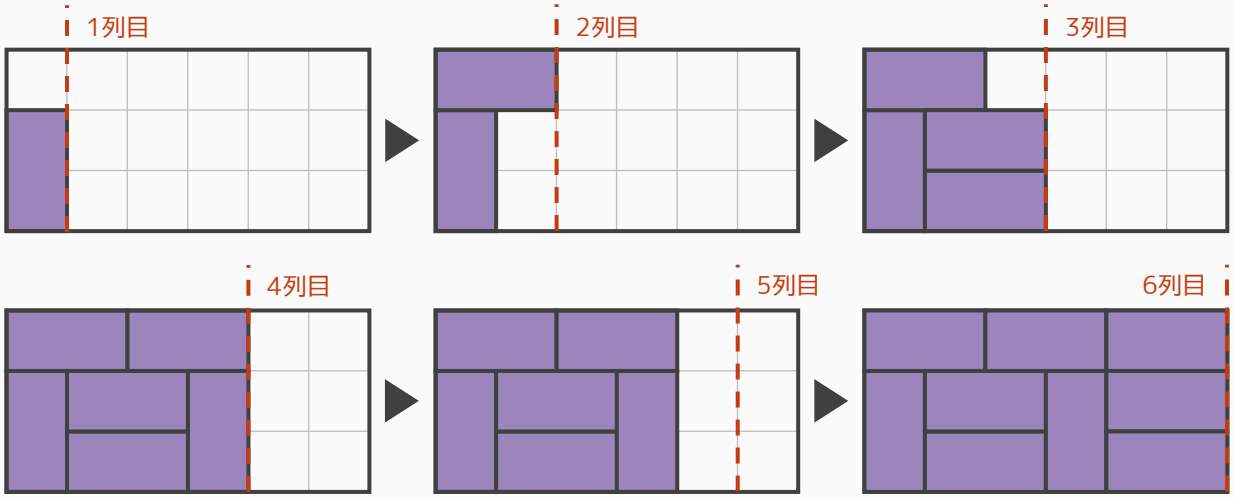


よって、$2 \times N$ のときの答えは以下のようにフィボナッチ数の第 $N + 1$ 項となるため、それを出力するプログラム（→**4.7.1項**）を作成すれば正解が得られます。

N	1	2	3	4	5	6	7	8
a_N	1	2	3	5	8	13	21	34

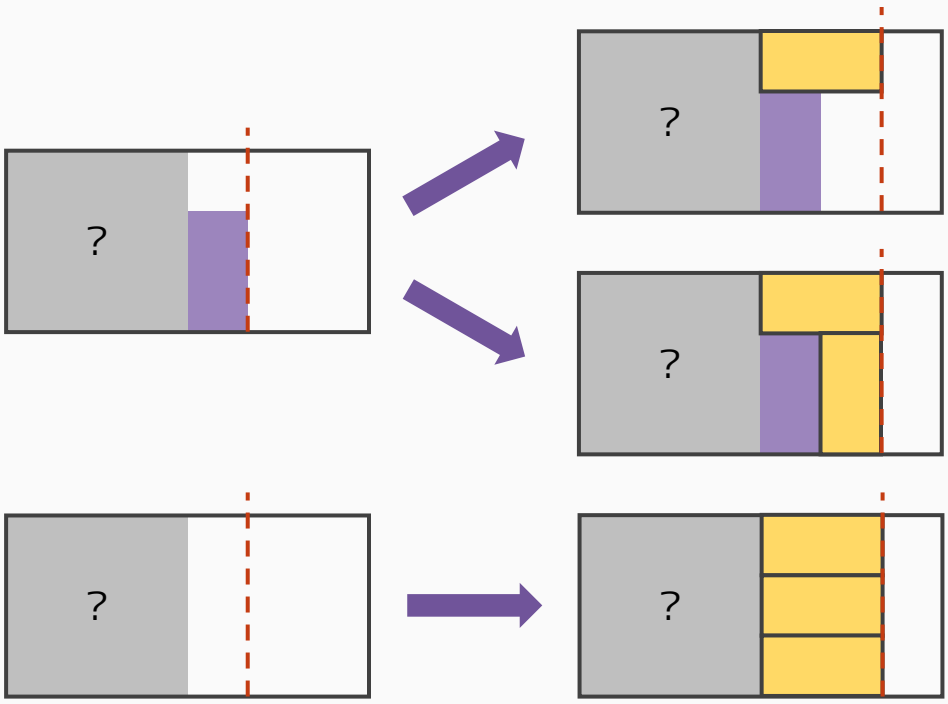
問題 4.7.4 (2), (3)

以下のように、左から一列ずつ敷き詰めていくことを考えましょう。x 列目までの敷き詰めを考えるとき、x 列目と $x+1$ 列目にまたがるピースは含めないものとします。

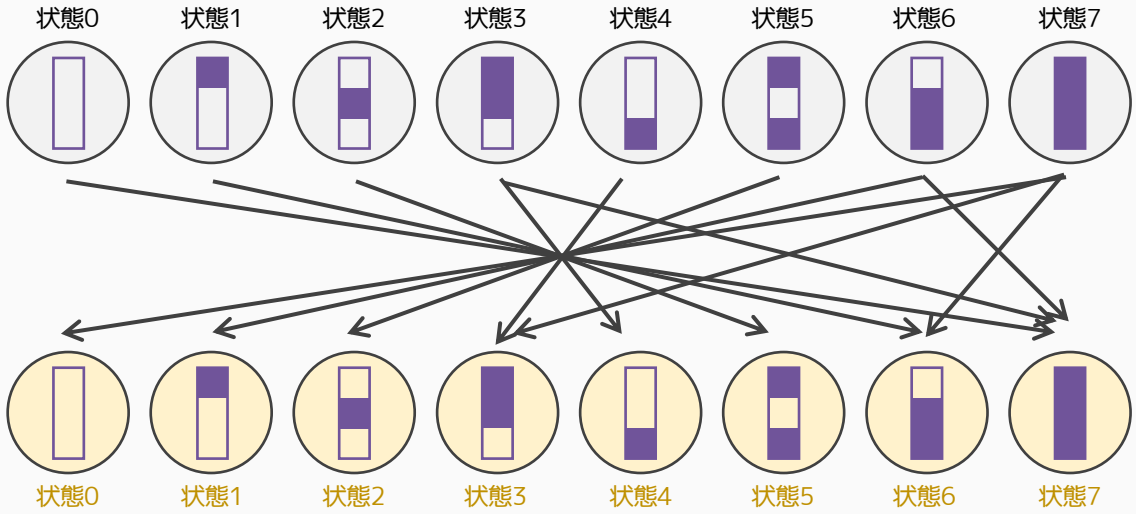


ここで、「x 列目までの敷き詰めでは必ず $x-1$ 列目までがすべて埋まっている」という重要な性質が成り立つため、この後の敷き詰め方は最後の列（x 列目）にのみ依存します。

たとえば、$K=3$ で下の 2 行にのみ既に置かれていた場合、以下の 2 通りの置き方があります。その他の場合も同様に考えることができます。



さて、「x 列目の段階で最後の列が〇〇のとき、$x+1$ 列目の段階で最後の列が△△であるようにする方法は何通りあるか？」ということを考えましょう。例えば $K=3$ の場合は下図のとおりであり、矢印 1 個につき 1 通りです。



そこで、最後の列の状態を $0, 1, \ldots, 2^K-1$ と番号を振るとき、$dp[x][y]$ を「x 列目まで敷き詰めた時点で状態 y である場合の数」とすると、以下の式が成り立ちます。

ただし、$A_{p,q}$ は状態 p から状態 q に遷移する方法の数（たとえば矢印が引かれていなければ $A_{p,q}=0$）とします。また、$L=2^K-1$ とします。

$$\begin{bmatrix} dp[x+1][0] \\ dp[x+1][1] \\ dp[x+1][2] \\ \vdots \\ dp[x+1][L] \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & \cdots & A_{0,L} \\ A_{1,0} & A_{1,1} & A_{1,2} & \cdots & A_{1,L} \\ A_{2,0} & A_{2,1} & A_{2,2} & \cdots & A_{2,L} \\ \vdots & \vdots & \vdots & & \vdots \\ A_{L,0} & A_{L,1} & A_{L,2} & \cdots & A_{L,L} \end{bmatrix} \begin{bmatrix} dp[x][0] \\ dp[x][1] \\ dp[x][2] \\ \vdots \\ dp[x][L] \end{bmatrix}$$

この式は少し複雑ですが、たとえば左辺の $(1,1)$ 成分である $dp[x+1][0]$ の値は、「x 列目まで敷き詰めて状態〇〇である場合の数 × 状態〇〇から状態 0 に遷移する方法の数」をすべての状態について足し合わせたものだと考えれば良いです。

したがって、$dp[N][0], dp[N][1], \ldots, dp[N][N-1]$ の値は以下のようになります。（上の式を繰り返し適用すれば良いです）

$$\begin{bmatrix} dp[N][0] \\ dp[N][1] \\ dp[N][2] \\ \vdots \\ dp[N][L] \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & \cdots & A_{0,L} \\ A_{1,0} & A_{1,1} & A_{1,2} & \cdots & A_{1,L} \\ A_{2,0} & A_{2,1} & A_{2,2} & \cdots & A_{2,L} \\ \vdots & \vdots & \vdots & & \vdots \\ A_{L,0} & A_{L,1} & A_{L,2} & \cdots & A_{L,L} \end{bmatrix}^N \begin{bmatrix} dp[0][0] \\ dp[0][1] \\ dp[0][2] \\ \vdots \\ dp[0][L] \end{bmatrix} = A^N \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

たとえば $K=3$ の場合、$dp[N][0], dp[N][1], dp[N][2], \ldots, dp[N][7]$ の値は以下の通りです。（$K=4$ の場合は長くなるため省略します）

$$\begin{bmatrix} dp[N][0] \\ dp[N][1] \\ dp[N][2] \\ dp[N][3] \\ dp[N][4] \\ dp[N][5] \\ dp[N][6] \\ dp[N][7] \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}^N \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

そこで最後の行（N 行目）はすべて埋まっていなければならないので、求める答えは $dp[N][L] = dp[N][2^K - 1]$ です。

よって、繰り返し二乗法を使って $2^K \times 2^K$ 行列の累乗を求めることで、計算量 $O((2^K)^3 \times \log N)$ でこの問題を解くことができました。以下は C++ の実装例です。

```cpp
#include <iostream>
using namespace std;

// K=2 の場合の遷移
long long Mat2[4][4] = {
    {0, 0, 0, 1},
    {0, 0, 1, 0},
    {0, 1, 0, 0},
    {1, 0, 0, 1}
};

// K=3 の場合の遷移
long long Mat3[8][8] = {
    {0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 1, 0, 0, 1},
    {0, 0, 0, 1, 0, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 1, 0, 0, 1, 0}
};

// K=4 の場合の遷移
long long Mat4[16][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
    {1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1}
};

struct Matrix {
    int size_ = 0; // 行列の大きさ（size_ × size_ の正方行列とする）
    long long p[16][16];
};

Matrix Multiplication(Matrix A, Matrix B) { // 行列 A, B の積を返す関数
    Matrix C;

    // 行列 C の初期化
    C.size_ = A.size_;
    for (int i = 0; i < A.size_; i++) {
        for (int j = 0; j < A.size_; j++) C.p[i][j] = 0;
    }

    // 行列の掛け算
    for (int i = 0; i < A.size_; i++) {
        for (int k = 0; k < A.size_; k++) {
            for (int j = 0; j < A.size_; j++) {
                C.p[i][j] += A.p[i][k] * B.p[k][j];
                C.p[i][j] %= 1000000007;
            }
        }
    }
    return C;
}

Matrix Power(Matrix A, long long n) { // A の n 乗を返す関数
    Matrix P = A, Q;
    bool flag = false;
    for (int i = 0; i < 60; i++) {
        if ((n & (1LL << i)) != 0LL) {
            if (flag == false) { Q = P; flag = true; }
            else { Q = Multiplication(Q, P); }
        }
        P = Multiplication(P, P);
    }
    return Q;
}
```

```cpp
int main() {
    // 入力
    long long K, N;
    cin >> K >> N;

    // 行列 A の作成
    Matrix A; A.size_ = (1 << K);
    for (int i = 0; i < (1 << K); i++) {
        for (int j = 0; j < (1 << K); j++) {
            if (K == 2) A.p[i][j] = Mat2[i][j];
            if (K == 3) A.p[i][j] = Mat3[i][j];
            if (K == 4) A.p[i][j] = Mat4[i][j];
        }
    }

    // B=A^N の計算
    Matrix B = Power(A, N);

    // 答えの出力
    cout << B.p[(1 << K) - 1][(1 << K) - 1] << endl;
    return 0;
}
```