第 5 章

数学的問題

5.1 問題 B26：Output Prime Numbers (難易度：★2相当)

この問題は、本の 158 ページで説明した「エラトステネスのふるい」を使って解くことができます。

本問題の制約は $N = 1000000$ ですが、エラトステネスのふるいの計算量は $O(N \times \log\log N)$ ですので、$\log\log 1000000 \fallingdotseq 3$ より余裕を持って実行時間制限に間に合います。

解答例（C++）

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

int N;
bool Deleted[1000009]; // 整数 x が消されている場合に限り Deleted[x]=true

int main() {
    // 入力
    cin >> N;

    // エラトステネスのふるい（i は √N 以下の最大の整数までループする）
    for (int i = 2; i <= N; i++) Deleted[i] = false;
    for (int i = 2; i * i <= N; i++) {
        if (Deleted[i] == true) continue;
        for (int j = i * 2; j <= N; j += i) Deleted[j] = true;
    }

    // 答えを出力
    for (int i = 2; i <= N; i++) {
        if (Deleted[i] == false) cout << i << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.2 問題 B27：Calculate LCM (難易度：★2相当)

まず、整数 a, b の最大公約数と最小公倍数に関して、以下の性質が必ず成り立ちます。

$$a \times b = (\text{最大公約数}) \times (\text{最小公倍数})$$

たとえば 25 と 30 の最大公約数は 5、最小公倍数は 150 であり、たしかに等式「25×30=5×150」が成り立っています※1。したがって、最小公倍数の値は、以下のアルゴリズムで高速に計算できます。

1. ユークリッドの互除法を使って、a, b の最大公約数を計算する
2. (最小公倍数) = $a \times b \div$ (最大公約数) を出力する

解答例（C++）

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

int GCD(int A, int B) {
    while (A >= 1 && B >= 1) {
        if (A >= B) {
                A = (A % B); // A の値を変更する場合
        }
        else {
                B = (B % A); // B の値を変更する場合
        }
    }
    if (A != 0) return A;
    return B;
}

int main() {
    long long A, B;
    cin >> A >> B;
    cout << A * B / GCD(A, B) << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

※1　これが成り立つ理由の証明は少し難しいですが、詳しくは https://manabitimes.jp/math/1032 をご覧ください

5.3 問題 B28：Fibonacci Easy (難易度：★2相当)

自然に実装すると、以下のようになります。しかし、制約の最大値である $N = 10^7$ の場合、第 N 項 a_N の値は 200 万桁を超え、残念ながらオーバーフローを起こしてしまいます。

```
a[1] = 1;
a[2] = 1;
for (int i = 3; i <= N; i++) a[i] = (a[i - 1] + a[i - 2]) % 1000000007;
cout << a[N] % 1000000007 << endl;
```

そこで、a_i の値を 1 回計算するごとに余りをとると、オーバーフローを防ぐことができます。解答例は以下の通りです。$2 \times 1000000007 < 2^{31}$ より、int 型などの 32 ビット整数でも十分であることに注意してください。

解答例（C++）

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

const int mod = 1000000007;
int N, a[10000009];

int main() {
    // 入力
    cin >> N;

    // フィボナッチ数列の計算
    a[1] = 1;
    a[2] = 1;
    for (int i = 3; i <= N; i++) {
        a[i] = (a[i - 1] + a[i - 2]) % mod;
    }

    // 出力
    cout << a[N] << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.4 問題 B29：Power Hard（難易度：★3相当）

例題 A29 の解答例（本の 172 ページ）では、30 回のループを行いました。しかし今回は制約が $b \leq 10^{18}$ であるため、ループ回数を 60 回まで増やす必要があります（$10^{18} < 2^{60}$ より、60 回あれば十分です）。

その他の注意点として、9 行目の変数 wari は long long 型などの 64 ビット整数にする必要があること、などが挙げられます。解答例は以下の通りです。

解答例（C++）

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

// a の b 乗を m で割った余りを返す関数
// 変数 a は a^1 → a^2 → a^4 → a^8 → a^16 → ··· と変化
long long Power(long long a, long long b, long long m) {
    long long p = a, Answer = 1;
    for (int i = 0; i < 60; i++) {
        long long wari = (1LL << i);
        if ((b / wari) % 2 == 1) {
                Answer = (Answer * p) % m; // 「a の 2^i 乗」が掛けられるとき
        }
        p = (p * p) % m;
    }
    return Answer;
}

int main() {
    long long a, b;
    cin >> a >> b;
    cout << Power(a, b, 1000000007) << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.5 問題 B30：Combination 2 (難易度：★4相当)

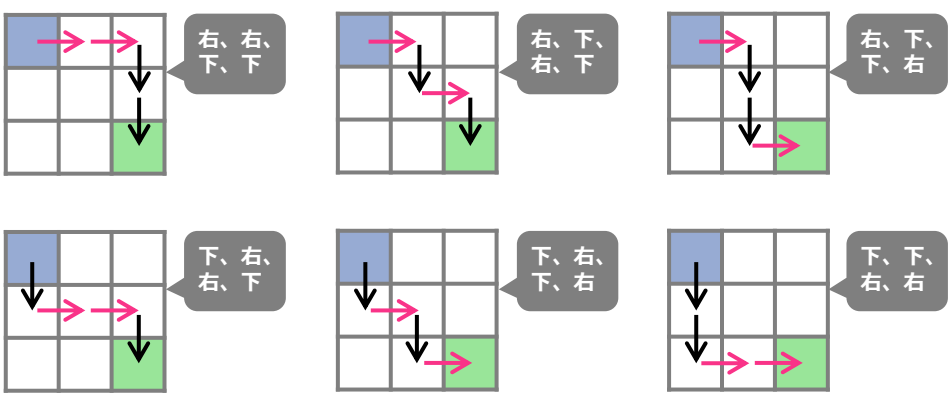
マス $(1,1)$ からマス (H,W) まで行くためには、全部で $H+W-2$ 回の移動を行う必要があり、その中の $W-1$ 回が右方向である必要があります。

逆に、右方向の移動回数が $W-1$ 回であれば、必ずマス (H,W) でゴールします。したがって、求める移動方法の数は、$H+W-2$ 個の中から $W-1$ 個を選ぶ方法の数 ${}_{H+W-2}\mathrm{C}_{W-1}$ 通りになります。

具体例

たとえば $H=3, W=3$ の場合を考えましょう。左上のマス $(1,1)$ から右下のマス $(3,3)$ まで行くためには、全部で 4 回の移動を行う必要があり、その中の 2 回が右方向の移動である必要があります。

逆に、4 回中 2 回が右方向であれば、必ずマス $(3,3)$ にたどり着くので、移動方法の数は ${}_4\mathrm{C}_2$=6 通りとなります。



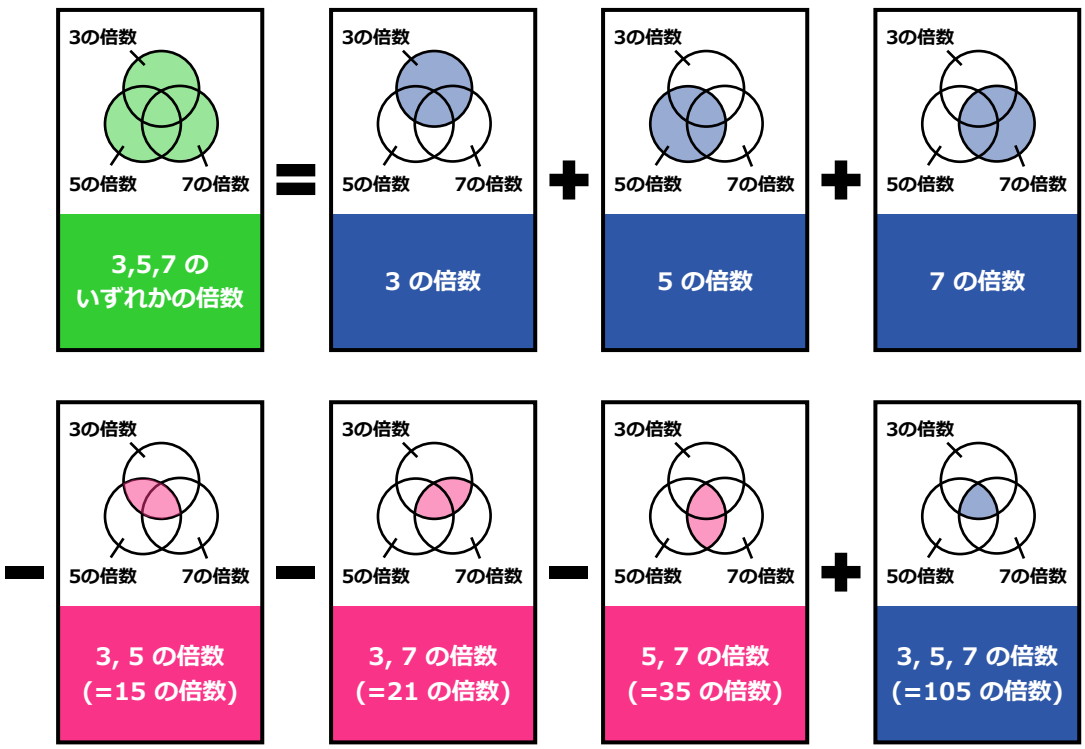
解答例（C++）

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

// a の b 乗を m で割った余りを返す関数
// 変数 a は a^1 → a^2 → a^4 → a^8 → a^16 → ··· と変化
long long Power(long long a, long long b, long long m) {
    long long p = a, Answer = 1;
    for (int i = 0; i < 30; i++) {
        int wari = (1 << i);
        if ((b / wari) % 2 == 1) {
                Answer = (Answer * p) % m; // 「a の 2^i 乗」が掛けられるとき
        }
        p = (p * p) % m;
    }
    return Answer;
}

// a ÷ b を m で割った余りを返す関数
long long Division(long long a, long long b, long long m) {
    return (a * Power(b, m - 2, m)) % m;
}

// nCr mod 1000000007 を返す関数
long long ncr(int n, int r) {
    const long long M = 1000000007;

    // 手順 1: 分子 a を求める
    long long a = 1;
    for (int i = 1; i <= n; i++) a = (a * i) % M;

    // 手順 2: 分母 b を求める
    long long b = 1;
    for (int i = 1; i <= r; i++) b = (b * i) % M;
    for (int i = 1; i <= n - r; i++) b = (b * i) % M;

    // 手順 3: 答えを求める
    return Division(a, b, M);
}

int main() {
    // 入力
    long long H, W;
    cin >> H >> W;

    // 出力
    cout << ncr(H + W - 2, W - 1) << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.6 問題 B31：Divisors Hard (難易度：★3相当)

3 つの集合の包除原理（→本の 180 ページ）より、「3, 5, 7 のいずれかの倍数であるものの個数」は以下のようにして計算することができます。



そこで、1 以上 N 以下の a の倍数の個数は $[N \div a]$ 個（$[x]$ は x の小数点以下切り捨て）であるため、本問題の答えは次式で表されます。

$$\left[\frac{N}{3}\right]+\left[\frac{N}{5}\right]+\left[\frac{N}{7}\right]-\left[\frac{N}{15}\right]-\left[\frac{N}{21}\right]-\left[\frac{N}{35}\right]+\left[\frac{N}{105}\right]$$

この値を出力する、次ページの解答例のようなプログラムを書くと、正解が得られます。計算量は $O(1)$ です。

解答例（C++）

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

int main() {
    long long N;
    cin >> N;

    long long A1 = (N /   3); // 3 で割り切れるものの個数
    long long A2 = (N /   5); // 5 で割り切れるものの個数
    long long A3 = (N /   7); // 5 で割り切れるものの個数
    long long A4 = (N /  15); // 3, 5 で割り切れるもの（= 15 の倍数）の個数
    long long A5 = (N /  21); // 3, 7 で割り切れるもの（= 21 の倍数）の個数
    long long A6 = (N /  35); // 5, 7 で割り切れるもの（= 35 の倍数）の個数
    long long A7 = (N / 105); // 3, 5, 7 で割り切れるもの（= 105 の倍数）の個数
    cout << A1 + A2 + A3 - A4 - A5 - A6 + A7 << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.7 問題 B32：Game 5 (難易度：★4相当)

この問題も、問題 A32（本の 181 ページ）と同じように、石の数が 0 個のときから順番に「勝ちの状態か負けの状態か」を求めていくと、答えが分かります。アルゴリズムの具体的な流れは以下の通りです。

$i = 0,1,2,\ldots,N$ の順に、次の規則にしたがって「石が i 個のときは勝ちの状態であるか」を求める。

[規則]

- 石が $i - a_1, \ldots, i - a_K$ 個のいずれかで負けの状態のとき：勝ちの状態
- 石が $i - a_1, \ldots, i - a_K$ 個すべてで勝ちの状態のとき：負けの状態

このアルゴリズムを実装すると、以下の解答例のようになります。計算量は $O(NK)$ です。なお、プログラム上では「石が i 個」が勝ちの状態のとき `dp[i]=true`、負けの状態のとき `dp[i]=false` となっています。

解答例（C++）

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

// 配列 dp について： dp[x]=true のとき勝ちの状態、dp[x]=false のとき負けの状態
int N, K, A[109];
bool dp[100009];

int main() {
    // 入力
    cin >> N >> K;
    for (int i = 1; i <= K; i++) cin >> A[i];

    // 勝者を計算する
    for (int i = 0; i <= N; i++) {
        dp[i] = false;
        for (int j = 1; j <= K; j++) {
                if (i >= A[j] && dp[i - A[j]] == false) {
                        dp[i] = true; // 負けの状態に遷移できれば、勝ちの状態
                }
```

```
        }
    }

    // 出力
    if (dp[N] == true) cout << "First" << endl;
    else cout << "Second" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.8 問題 B33：Game 6 (難易度：★6相当)

この問題は、**「何行目か」と「何列目か」を別々に考える**ことで、山の数が $2N$ のニムに帰着させることができます。具体的には以下の通りです。

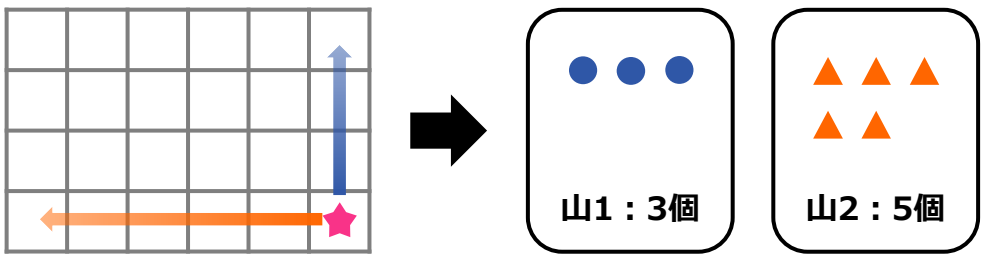
現在の i 個目のコマの位置を、上から $a_i + 1$ 行目、左から $b_i + 1$ 列目とする。このとき、一回の操作では次のことができる。

- **コマ i を左方向に動かす：** b_i を 1 以上減らす
- **コマ i を上方向に動かす：** a_i を 1 以上減らす
- ただし、マス目の範囲に収めるため、a_i, b_i は 0 以上であるべき

つまり、一回の操作は「$[a_1, a_2, \ldots, a_N, b_1, b_2, \ldots, b_N]$ の中から 1 つ選び、 負の数にならない範囲で 1 以上減らすこと」に対応する。

これは山の数が $2N$ であり、各山の石の数が $[a_1, a_2, \ldots, a_N, b_1, b_2, \ldots, b_N]$ であるニムと全く等価である。

たとえばコマが 1 個であり、そのコマが上から 4 行目・左から 6 行目に存在する場合、「石が 3, 5 個ある 2 つの山のニム」と等価です（もちろん 3 XOR 5 ≠ 0 なので、先手必勝です※2）。



したがって、本問題の答えは、$(A_1 - 1)$ XOR … XOR $(A_N - 1)$ XOR $(B_1 - 1)$ XOR … XOR $(B_N - 1) = 0$ であれば後手必勝、そうでなければ先手必勝となります。次ページに解答例を示します。

※2　ニムでは、各山の石の数をすべて XOR した値が 0 であれば後手必勝であったことを思い出しましょう。本の 186 ページに書かれています

解答例（C++）

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

int H, W;
int N, A[200009], B[200009];

int main() {
    // 入力
    cin >> N >> H >> W;
    for (int i = 1; i <= N; i++) cin >> A[i] >> B[i];

    // 全部 XOR した値（ニム和）を求める
    int XOR_Sum = 0;
    for (int i = 1; i <= N; i++) XOR_Sum = (XOR_Sum ^ (A[i] - 1));
    for (int i = 1; i <= N; i++) XOR_Sum = (XOR_Sum ^ (B[i] - 1));

    // 出力
    if (XOR_Sum != 0) cout << "First" << endl;
    if (XOR_Sum == 0) cout << "Second" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

5.9 問題 B33：Game 6 (難易度：★5相当)

まずは山が 1 つの場合を考えましょう。$X = 2, Y = 3$ のとき、石の数が 0 個から 39 個のときの Grundy 数は以下の通りです（本の 191 ページのように、実際に手で計算してみましょう）。

石の数	0	1	2	3	4	5	6	7	8	9
Grundy 数	0	0	1	1	2	0	0	1	1	2

石の数	10	11	12	13	14	15	16	17	18	19
Grundy 数	0	0	1	1	2	0	0	1	1	2

石の数	20	21	22	23	24	25	26	27	28	29
Grundy 数	0	0	1	1	2	0	0	1	1	2

石の数	30	31	32	33	34	35	36	37	38	39
Grundy 数	0	0	1	1	2	0	0	1	1	2

0,0,1,1,2 が周期的に繰り返されていますね。それでは、この周期性は石の数が 40 個以上でも成り立つのでしょうか。答えは Yes です。少し難しいですが、以下のようにして証明することができます。

石が $(i-3, i-2, i-1)$ 個の Grundy 数の値から、石が $(i-2, i-1, i-0)$ 個の Grundy 数を求めることを考える。このとき※3、

1. $(0, 0, 1)$ の次は $(0, 1, 1)$ である $[0, 0$ にない最小の非負整数は $1]$
2. $(0, 1, 1)$ の次は $(1, 1, 2)$ である $[0, 1$ にない最小の非負整数は $2]$
3. $(1, 1, 2)$ の次は $(1, 2, 0)$ である $[1, 1$ にない最小の非負整数は $0]$
4. $(1, 2, 0)$ の次は $(2, 0, 0)$ である $[1, 2$ にない最小の非負整数は $0]$
5. $(2, 0, 0)$ の次は $(0, 0, 1)$ である $[2, 0$ にない最小の非負整数は $1]$

5 回でまた 1. に戻るので、5 個周期で「0→0→1→1→2」と繰り返す。

※3　遷移できる「石の数が $i-3, i-2$ 個の状態」の Grundy 数を赤色で示しています。

したがって、石の数が A_1 個のとき、Grundy 数は下表のようになります。

石の数 A_1 を 5 で割った余り	0	1	2	3	4
Grundy 数	0	0	1	1	2

山が 2 個以上の場合

山が 2 つ以上の場合は、本の 192 ページに示したように、それぞれの山に対する Grundy 数を計算することで勝敗を判定することができます。

Grundy 数をすべて XOR した値 XOR_Sum が 0 であるとき後手必勝となり、そうでなければ先手必勝です。実装例を以下に示します。

解答例（C++）

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

long long N, X, Y, A[100009];

int main() {
    // 入力
    cin >> N >> X >> Y;
    for (int i = 1; i <= N; i++) cin >> A[i];

    // Grundy 数を計算
    int XOR_Sum = 0;
    for (int i = 1; i <= N; i++) {
        if (A[i] % 5 == 0 || A[i] % 5 == 1) XOR_Sum ^= 0;
        if (A[i] % 5 == 2 || A[i] % 5 == 3) XOR_Sum ^= 1;
        if (A[i] % 5 == 4) XOR_Sum ^= 2;
    }

    // 出力
    if (XOR_Sum != 0) cout << "First" << endl;
    if (XOR_Sum == 0) cout << "Second" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください