5.2 節末問題 5.2 の解答

問題 5.2.1

フィボナッチ数の第 12 項までと、それらを 4 で割った余りは以下の通りです。

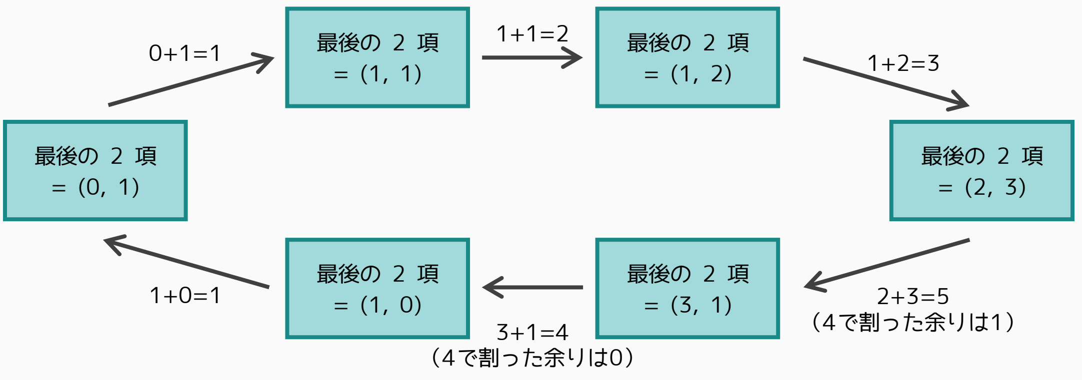
$1 \to 1 \to 2 \to 3 \to 1 \to 0 \to \cdots$ が周期的に繰り返されていますね。

N	1	2	3	4	5	6	7	8	9	10	11	12
第 N 項	1	1	2	3	5	8	13	21	34	55	89	144
4 で割った余り	1	1	2	3	1	0	1	1	2	3	1	0

この周期性は N が大きくなっても成り立つのでしょうか。実は

- フィボナッチ数において、項の値は直前の二項のみから決まること
- (第 1 項, 第 2 項) と (第 7 項, 第 8 項) が一致していること

から証明できます。イメージ図は以下の通りです。



したがって、フィボナッチ数の第 N 項を 4 で割った余りは、第 $(N \bmod 6)$ 項を 4 で割った余りと一致します（N が 6 の倍数は第 6 項と一致）。

$10000 \bmod 6 = 4$ なので、フィボナッチ数の第 10000 項は、第 4 項を 4 で割った余りである 3 となります。

（解説は次ページへ続きます）

問題 5.2.2

まず、石が 1 個の状態から石を取ることはできないため、$N=1$ は負けの状態（後手必勝）です。一方、石が 2 個のときは先手が 1 個の石を取ると後手が手を打てなくなるので、$N=2$ は勝ちの状態です。

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
状態	負	勝													

次に $N=3$ の場合を考えましょう。一般に、ゲームは負けの状態に遷移可能な場合のみ勝つことができます（→**5.2.2項**）。しかし、「石を 1 個取り、残り 2 個（勝ちの状態）に減らす」という操作しかできないため、$N=3$ は負けの状態です。

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
状態	負	勝	負												

次に、石が $4, 5, 6$ 個の状態からは一手で石の数を 3 個（負けの状態）に減らすことができるため、それらは勝ちの状態です。一方、石が 7 個の状態からは、

- 石を 1 個取り、石の数を 6 個に減らす
- 石を 2 個取り、石の数を 5 個に減らす
- 石を 3 個取り、石の数を 4 個に減らす

という 3 つの操作方法がありますが、すべて勝ちの状態に遷移します。よって、$N=7$ は負けの状態です。

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
状態	負	勝	負	勝	勝	勝	負								

同じように考察を進めていくと、$N=8, 9, 10, 11, 12, 13, 14$ は勝ちの状態、$N=15$ は負けの状態であることが分かります。

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
状態	負	勝	負	勝	勝	勝	負	勝	勝	勝	勝	勝	勝	勝	負

ここまでの時点で $1, 3, 7, 15$ 個が負けの状態であるため、勘の良い人は「$2^k - 1$ で表される数だけが負けの状態ではないか」という周期性が頭に浮かぶと思います。（浮かばなかった人は、$N = 16$ 以降も調べてみてください）

実は、この周期性は N が大きくなっても成り立ちます（証明略）。したがって、$N = 2^k - 1$ となるかどうかを $1 \leq k \leq 60$ の範囲で全探索する以下のようなプログラムを書くと、正解が得られます。

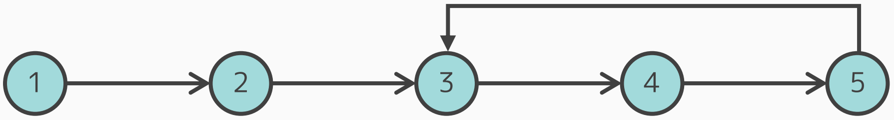
なお、本問題の制約は $N \leq 10^{18}$ であり、$2^{60} > 10^{18}$ であるため、$k \leq 60$ までの探索で十分です）

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

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

	// N = 2^k-1 の形で表されるかどうかを調べる
	bool flag = false;
	for (int k = 1; k <= 60; k++) {
		if (N == (1LL << k) - 1LL) flag = true;
	}

	// 出力
	if (flag == true) cout << "Second" << endl;
	else cout << "First" << endl;
	return 0;
}
```

※ Python などのソースコードは chap5-2.md をご覧ください。

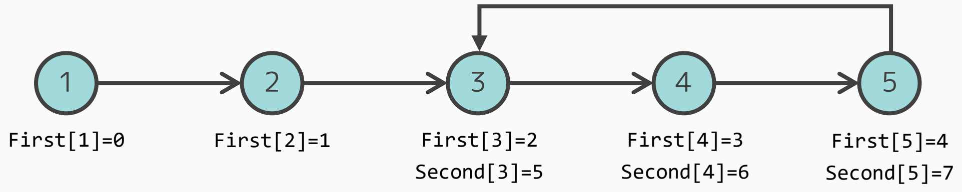
問題 5.2.3

この問題は少し複雑なので、まずは $N = 5,\ A = (2, 3, 4, 5, 3)$ の場合を考えましょう。テレポータの転送は下図のようになり、町 1 から出発した場合、$1 \to 2 \to 3 \to 4 \to 5 \to 3 \to 4 \to 5 \to 3 \to 4 \to 5 \to \cdots$ と周期的に移動します。



この周期性は一般のケースでも成り立ちます。N 回以内の移動で、既に訪問した町に戻ってきて、その後は周期的な移動を行うことが証明できます。

ここで、最初に町 u に訪れたときに $First[u]$ 回テレポーターを使っており、二度目に訪れたときに $Second[u]$ 回テレポーターを使っているとしましょう。（下図は具体例となります）



$L = Second[u] - First[u]$（周期の長さ）とすると、町 u にはテレポーターを $First[u], First[u] + L, First[u] + 2L, \ldots$ 回使ったときに訪問します。上の例では、

- 町 3：テレポーターを $2, 5, 8, 11, 14, 17, \ldots$ 回使ったときに訪問
- 町 4：テレポーターを $3, 6, 9, 12, 15, 18, \ldots$ 回使ったときに訪問
- 町 5：テレポーターを $4, 7, 10, 13, 16, 19, \ldots$ 回使ったときに訪問

となります。

したがって、$(K - First[u]) \bmod L = 0$ のとき、K 回の移動で町 u に到着します。このような u を調べる以下のようなプログラムを提出すると、正解が得られます。

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

long long N, K;
long long A[200009];
long long First[200009], Second[200009];

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

    // 配列の初期化
    for (int i = 1; i <= N; i++) First[i] = -1;
    for (int i = 1; i <= N; i++) Second[i] = -1;

    // 答えを求める
    // cur は現在いる町の番号
    long long cnt = 0, cur = 1;
    while (true) {
        // First, Second の更新
```

```
        if (First[cur] == -1) First[cur] = cnt;
        else if (Second[cur] == -1) Second[cur] = cnt;

        // K 回の移動後に町 cur にいるかどうかの判定
        if (cnt == K) {
            cout << cur << endl;
            return 0;
        }
        else if (Second[cur] != -1 && (K-First[cur]) % (Second[cur]-First[cur]) == 0) {
            cout << cur << endl;
            return 0;
        }

        // 位置の更新
        cur = A[cur];
        cnt += 1;
    }
    return 0;
}
```

※ Python などのソースコードは chap5-2.md をご覧ください。