

問題 5.2.1

以下為費波那契數到第 12 項為止的數及其除以 4 的餘數。。

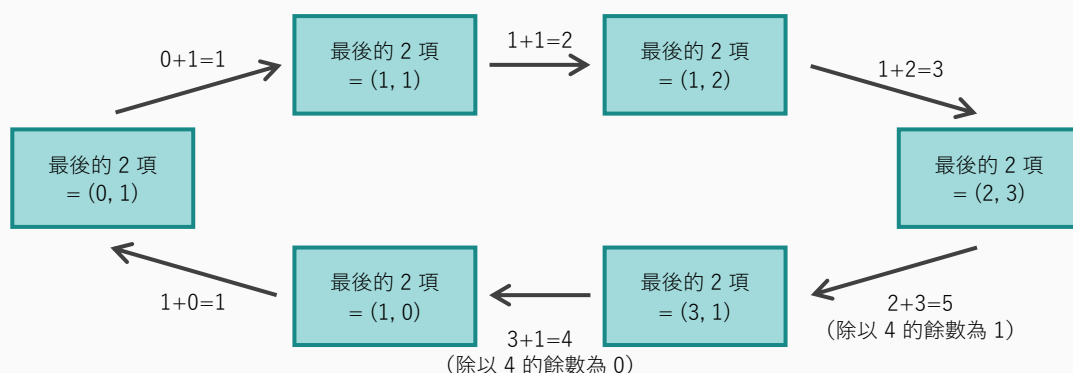
$1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow \dots$ 會週期性地重複。

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
除以 4 的餘數	1	1	2	3	1	0	1	1	2	3	1	0

此週期性在 N 增大時是否依然成立呢？實際上可以從以下兩點來證明：

- 在費波那契數中，每一項的值只由前兩項的值決定
- (第 1 項, 第 2 項) 與 (第 7 項, 第 8 項) 一致

示意圖如下。



因此，費波那契數的第 N 項除以 4 的餘數，與第 $(N \bmod 6)$ 項除以 4 的餘數相同（當 N 是 6 的倍數時是與第 6 項相同）。

$10000 \bmod 6 = 4$ ，所以費波那契數的第 10000 項除以 4 的餘數是第 4 項除以 4 的餘數，即 3。

（解說在下一頁繼續）


問題 5.2.2

首先，石頭為 1 個的狀態下，無法再拿石頭，因此 $N = 1$ 是敗北狀態（後手必勝）。
另一方面，當有 2 個石頭時，若先手取 1 個石頭，後手無法再行動，因此 $N = 2$ 是獲勝狀態。

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

接下來考慮 $N = 3$ 的情況。一般而言，遊戲只有在能轉換到敗北狀態時才能獲勝（→ 5.2.2項）。但由於「取 1 個石頭，減少到剩下 2 個（獲勝狀態）」是唯一能做的操作，因此 $N = 3$ 是敗北狀態。

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



接下來，當石頭有 4, 5, 6 個的狀態時，可以一步減少到 3 個石頭（敗北狀態），因此這些是獲勝狀態。此外，當石頭有 7 個的狀態，有以下三種操作：

- 取 1 個石頭，剩下 6 個（獲勝狀態）
- 取 2 個石頭，剩下 5 個（獲勝狀態）
- 取 3 個石頭，剩下 4 個（獲勝狀態）


但所有操作都轉換到獲勝狀態。因此 $N = 7$ 是敗北狀態。

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



進行同樣的考察的話，可以知道 $N = 8, 9, 10, 11, 12, 13, 14$ 是獲勝狀態，而 $N = 15$ 是敗北狀態。

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



至此，我們發現 1, 3, 7, 15 是敗北狀態，敏銳的人可能會想到「是否只有以 $2^k - 1$ 表示的數字是敗北狀態呢？」的週期性。（如果沒想到， $N = 16$ 之後的情況也請調查看看）。

事實上，這種週期性在 N 增大時依然成立（證明略）。因此，撰寫以下程式，在 $1 \leq k \leq 60$ 的範圍內進行全搜尋以判斷是否為 $N = 2^k - 1$ ，可以得到正解。

此外，本問題的限制 $N \leq 10^{18}$ ，因為 $2^{60} > 10^{18}$ ，所以僅需搜尋到 $k \leq 60$ 為止即可。

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

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

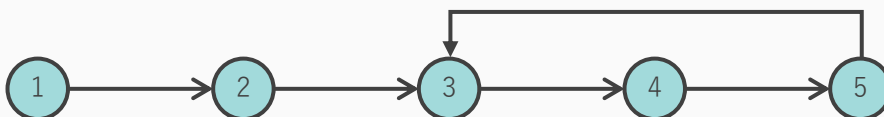
    // 檢查是否以  $N = 2^k - 1$  的形式表示
    bool flag = false;
    for (int k = 1; k <= 60; k++) {
        if (N == (1LL << k) - 1LL) flag = true;
    }

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

※ Python等原始碼請參閱chap5-2.md。

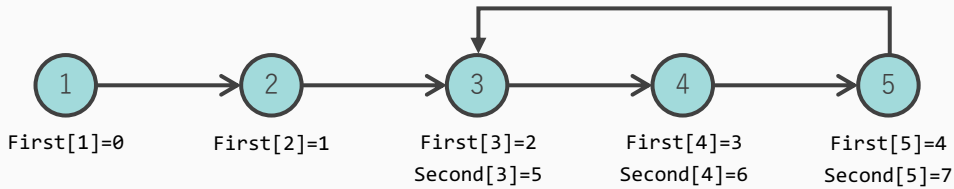
問題 5.2.3

這個問題稍微有點複雜，因此首先思考 $N = 5$, $A = (2, 3, 4, 5, 3)$ 的情況。發信機的轉發如下圖所示，從城鎮 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots$ 週期性地移動。



此週期性在一般情況下也成立。可以證明在 N 次移動以內，會回到已訪問過的城市，之後將以週期性方式進行移動。

在此，假設第一次訪問城鎮 u 時，使用了 $First[u]$ 次發信機，第二次訪問時，使用了 $Second[u]$ 次發信機吧。（下圖為具體例）



設 $L = Second[u] - First[u]$ （週期的長度），則在使用了第 $First[u], First[u] + L, First[u] + 2L, \dots$ 次發信機時訪問城鎮 u 。上述例子中，

- 城鎮 3：在使用第 2, 5, 8, 11, 14, 17, ... 次發信機時訪問
- 城鎮 4：在使用第 3, 6, 9, 12, 15, 18, ... 次發信機時訪問
- 城鎮 5：在使用第 4, 7, 10, 13, 16, 19, ... 次發信機時訪問

因此，當 $(K - First[u]) \bmod L = 0$ 時，在第 K 次移動到達城市 u 。提出可以檢查這種 u 的程式來得到正解。。

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

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

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

    // 陣列的初始化
    for (int i = 1; i <= N; i++) First[i] = -1;
    for (int i = 1; i <= N; i++) Second[i] = -1;

    // 求出答案
    // cur 為現在所在城鎮的編號
    long long cnt = 0, cur = 1;
    while (true) {
        // First, Second 的更新
```

```

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

    // 判斷在 K 次移動後是否位在城鎮 cur
    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;
    }

    // 位置更新
    cur = A[cur];
    cnt += 1;
}
return 0;
}

```

※ Python等原始碼請參閱 chap5-2.md。