5.10 節末問題 5.10 的解答

問題 5.10.1

使用分配律（**→5.10.2項**）可以如以下所示輕鬆地計算而解決。

問題 (1)

$$
\begin{aligned}
&37 \times 39 + 37 \times 61 \\
&= 37 \times (39 + 61) \\
&= 37 \times 100 \\
&= \mathbf{3700}
\end{aligned}
$$

問題 (2)

$$
\begin{aligned}
&2021 \times 333 + 2021 \times 333 + 2021 \times 334 \\
&= 2021 \times (333 + 333 + 334) \\
&= 2021 \times 1000 \\
&= \mathbf{2021000}
\end{aligned}
$$

問題 5.10.2

$1 + 2 + \cdots + N \times 1 + 2 + \cdots + N =$

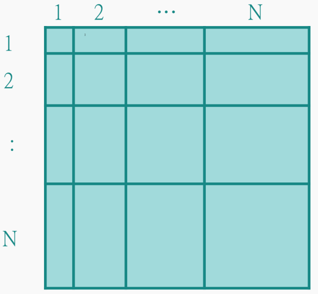
本問題是例題2（**→5.10.2項**）的一般化。使用分配律，可知以下有關求和符號式子的特徵：×

- 當 $i = 1$ 時的總和：$(1 \times 1) + (1 \times 2) + \cdots + (1 \times N) = 1 \times (1 + 2 + \cdots + N)$
- 當 $i = 2$ 時的總和：$(2 \times 1) + (2 \times 2) + \cdots + (2 \times N) = 2 \times (1 + 2 + \cdots + N)$
- ：
- 當 $i = N$ 時的總和：$N \times 1 + N \times 2 + \cdots + N \times N = N \times (1 + 2 + \cdots + N)$

所求的答案是藍色顯示的值的總和，因此，根據分配律，結果如下。

$$\sum_{i=1}^{N} \sum_{j=1}^{N} ij = (1 + 2 + \cdots + N) \times (1 + 2 + \cdots + N) = \frac{N(N+1)}{2} \times \frac{N(N+1)}{2}$$

對此沒有概念的話，可以思考右圖中的正方形面積。



此正方形的縱長是$N(N+1)/2$，橫長也是 $N(N+1)/2$。

因此，提出如以下輸出答案的程式即為正解。此外，這個問題的限制是$N \leq 10^9$之大，因此 $N \quad N+1\ /2 \times N(N+1)/2$ 的值可能超過10^{30}。注意若不進行在計算過程中取餘數（**→4.6.1項**）等處理，即使使用 `long long` 型態的 64 位元整數，也有可能會溢出。

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

const long long mod = 1000000007;
long long N;

int main() {
    // 輸入
    cin >> N;

    // 求出答案
    long long val = N * (N + 1) / 2;
    val %= mod;
    cout << val * val % mod << endl;
    return 0;
}
```

※ Python 等原始碼請參閱 chap5-10.md 。

問題 5.10.3

對此思考如下的立方體，可知本問題的答案是：

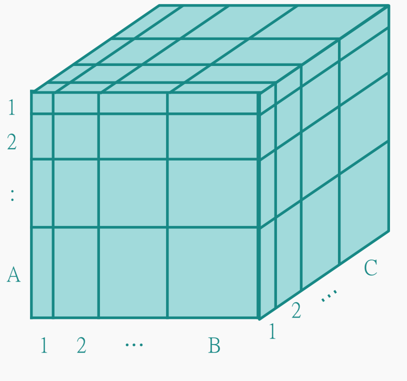
$$\sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{N} ijk = (1+\cdots+A)(1+\cdots+B)(1+\cdots+C)$$

因此，根據和的公式（**→2.5.10項**）以下式子會成立。

$$1+2+\cdots+A = \frac{A(A+1)}{2}$$

$$1+2+\cdots+B = \frac{B(B+1)}{2}$$

$$1+2+\cdots+C = \frac{C(C+1)}{2}$$



所以答案為 $\frac{A(A+1)}{2} \times \frac{B(B+1)}{2} \times \frac{C(C+1)}{2}$ 。

因此，提出如下將答案輸出的程式即為正解。另外，本問題的限制是 $A, B, C \leq 10^9$，因為很大，為了防止溢出，將變數設定如下：

- $D = A(A+1)/2$
- $E = B(B+1)/2$
- $F = C(C+1)/2$

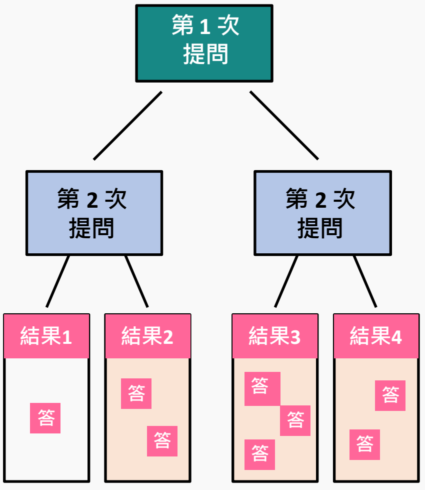
並進行在計算過程中取餘數等處理。

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

const long long mod = 998244353;
long long A, B, C;

int main() {
    // 輸入
    cin >> A >> B >> C;

    // 計算
    long long D = A * (A + 1) / 2; D %=
    mod; long long E = B * (B + 1) / 2; E %=
    mod; long long F = C * (C + 1) / 2; F %=
    mod;

    // 輸出答案
    // 即使在此計算 (D * E * F) % mod ，可能在途中處理到 10^27
    // 因此，注意即使是 long long 型態也會發生溢出！
    cout << (D * E % mod) * F % mod << endl;
    return 0;
}
```

※ Python 等原始碼請參閱 chap5-10.md 。

問題 5.10.4

首先，太郎想到的數字有 8 種可能性，但對於兩次提問的回答組合只有「Yes→Yes」、「Yes→No」、「No→Yes」、「No→No」這四種。



由於 $8 > 4$，因此如右圖所示，「不確定結果為1種的情況」一定會存在。因此，無法透過兩次問題來確定答案。

問題 5.10.5

若自然地進行實作，會如下所示。

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

int main() {
        // 輸入
        double a, b, c;
        cin >> a >> b >> c;

        // 計算左邊和右邊
        double v1 = log2(a);
        double v2 = b * log2(c);

        // 輸出
        if (v1 < v2) cout << "Yes" << endl;
        else cout << "No" << endl;
        return 0;
}
```

但是，若提出這個程式，100 個案例中有 15 個案例是不正確的。其原因是誤差（**→5.10.1項**）。

例如，在 $(a, b, c) = (10^{18} - 1, 18, 10)$ 的情況下，真正的答案是 Yes，但卻錯誤的輸出了 No。實際上如下。

$$\log_2 a = 59.794705707972522602 \ldots$$
$$b \log_2 c = 59.794705707972522616 \ldots$$

左邊和右邊的相對誤差約為 10^{-19}左右。因為過於接近，超出了電腦的極限，被判定成「是相同的數字」。

改善方法①

左邊那麼，該如何防止由誤差導致的不正確呢？一種方法是全部用整數來處理。根據對數的性質（**→2.3.10項**）以下成立。

因為 $b \log_2 c = \log_2(c^b)$

當 $\log_2 a < \boxed{b \log_2 c}$ 時，$\log_2 a < \boxed{\log_2(c^b)}$

即使只取 log 的中心
大小關係也不會改變

因此，$a < c^b$

若 $a < c^b$ 則輸出Yes，若非如此則輸出No。

將此方法進行實作如下：

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

int main() {
    // 輸入
    long long a, b, c;
    cin >> a >> b >> c;

    // 計算右邊（c 的 b 次方）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        v *= c;
    }

    // 輸出
    if (a < v) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

但是，這會成為非正解。原因是溢出（**→5.10.1項**）。此程式會直接計算 c^b 的值，但由於限制是 $a, b, c \leq 10^{18}$，在最壞情況下需要計算10^{18} 的 10^{18} 次方。不用說C++了，連Python也無法進行計算。

改善方法②

接著，如何防止溢出呢？典型的方法是「在計算過程中取餘數」等，但這個問題不是餘數計算，因此這種方法不適用。

因此，由於在計算乘方的過程中，右邊的**值**超過 a的當下就可以確定為 Yes，故將迴圈中止的對策是有效的。自然地實作如下：

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

int main() {
    // 輸入
```

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

int main() {
    // 輸入
    long long a, b, c;
    cin >> a >> b >> c;

    // 計算右邊（c 的 b 次方）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        if (a / c < v) {
            // 輸此條件分支只是將 a < (v * c) 換句話說
            // 行條件換算的理由是 v, c 可能會達到 10^18 左右
            // 若進行 a < v * c，最差的情況下 v * c = 10^36 而產生溢出
            // 注：long long 型態的極限為2^{63}-1（約 10^{19}）
            cout << "Yes" << endl;
            return 0;
        }
        v *= c;
    }

    // 迴圈無法中止的情況
    cout << "No" << endl;
    return 0;
}
```

但是，這個程式在 100 個測試案例中有 2 個超過了執行時間限制（TLE）。原因是 $c = 1$ 的案例。

若例如 $(a, b, c) = (2, 10^{18}, 1)$ 的情況。1 的任何次方都是 1，故「目前右邊的值 v 超過 a 就中止」的處理沒有作用。因此，仍然進行了 $b = 10^{18}$ 次迴圈。

此外，由於 $2^{60} > 10^{18}$，因此當 $c \geq 2$ 時，一定可以在 60 次迴圈內完成處理。

改善方法③

最後，對於 $c = 1$ 的情況區分處理。由於此問題的限制是 $a \geq 1$，因此當 $c^b = 1$ 時，答案一定是 No。因此，撰寫如下程式後，總算可獲得正解。

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

int main() {
    // 輸入
    long long a, b, c;
    cin >> a >> b >> c;

    // 區分 c = 1 時的狀況
    if (c == 1) {
        cout << "No" << endl;
        return 0;
    }

    // 計算右邊（c 的 b 次方）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        if (a / c < v) {
            // 此條件分支只是將 a < (v * c) 換句話說
            // 進行條件換算的理由是 v, c 可能會達到 10^18 左右
            // 若進行 a < v * c，最差的情況下 v * c = 10^36 而產生溢出
            cout << "Yes" << endl;
            return 0;
        }
        v *= c;
    }

    // 迴圈無法中止的情況
    cout << "No" << endl;
    return 0;
}
```

※ Python 等原始碼請參閱 chap5-10.md 。

問題 5.10.6

若首先，可以考慮分別對$m = 1, 2, \ldots, N$ 進行調查的方法，但由於限制是 $N \leq 10^{11}$ 之大，執行時間限制會超過（TLE）。

也就是說，m 可能的模式數遠大於 $f(m)$ 可能的模式數，故以下演算法更有效率。

- 列舉所有可能的 $f(m)$ 候選。
- 若確定 $f(m)$ 則也確定$m = f(m) + B$，因此對每個候選檢查 m 的每位數字的乘積是否與 $f(m)$ 一致。

那麼該如何列舉 $f(m)$ 的候選呢？其實，只需對單調增加數字 m，如 1123 或 12233599的 $f(m)$ 即可。因為即使數字順序調換也不失普遍性，例如以下結果全部相同。

- $m = 1123$ のとき $f(m) = 1 \times 1 \times 2 \times 3 = 6$
- $m = 2131$ のとき $f(m) = 2 \times 1 \times 3 \times 1 = 6$
- $m = 3112$ のとき $f(m) = 3 \times 1 \times 1 \times 2 = 6$

此外，單調增長的 11 位以內的數字約有 30 萬個，全部列舉是現實可行的。

因此，撰寫如下程式可以得到正解。又，單調增長數字 m 用遞迴函數 `func(digit, m)` 進行全列舉，`digit` 代表目前的位數。若不瞭解遞迴函數，可以回到第3.6節確認。

此外，函數 `product(m)` 會回傳整數 m 每位數字的乘積。藉由不斷將數字除以10來計算每位數字。與轉換成二進制的演算法（**→2.1.9項**）類似。

※注意：此 C++ 程式使用了本書未討論的 `set` 型態。不瞭解的人可以上網調查。（GitHub 上刊載的 Python、JAVA 的原始碼也使用了 `set` 型態）

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

// 作為 f(m) 可能有的候選
// 關於 set 型態，請在網路上查詢！
set<long long> fm_cand;

// 回傳 m 的各個位數乘積的函式
long long product(long long m) {
    if (m == 0) {
        return 0;
    }
    else {
        long long ans = 1;
        while (m >= 1) {
            ans *= (m % 10);
            m /= 10;
        }
        return ans;
```

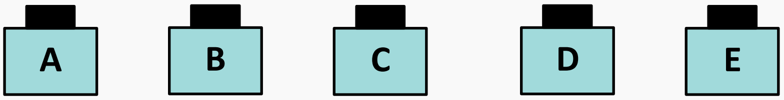
```
    }
}

void func(int digit, long long m) {
    // m 的位數為 11 位以下
    // 注：如果用 1 填充剩餘的位數，可以假設全部都是 11 位。
    int min_value = (m % 10);
    for (int i = min_value; i <= 9; i++) {
        // 10 * m + i 是在 m 之後附帶數字 i 的值
        func(digit + 1, 10 * m + i);
    }
}

int main() {
    // 列舉 f(m) 的候選
    func(0, 0);

    // 輸入
    long long N, B;
    cin >> N >> B;

    // 檢查是否為 m - f(m) == B
    long long Answer = 0;
    for (long long fm : fm_cand) {
        long long m = fm + B;
        long long prod_m = product(m);
        if (m - prod_m == B && m <= N)
        {
            Answer += 1;
        }
    }

    // 輸出
    cout << Answer << endl;
    return 0;
}
```

※ Python 等原始碼請參閱 chap5-10.md 。

問題 5.10.7 (1)

此問題可以想到多種解法，因此介紹其中一種。

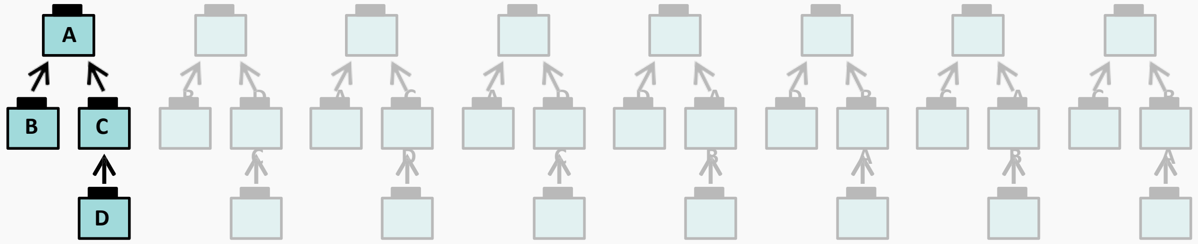
另外，為了方便說明，在每個砝碼上標記 A、B、C、D、E。



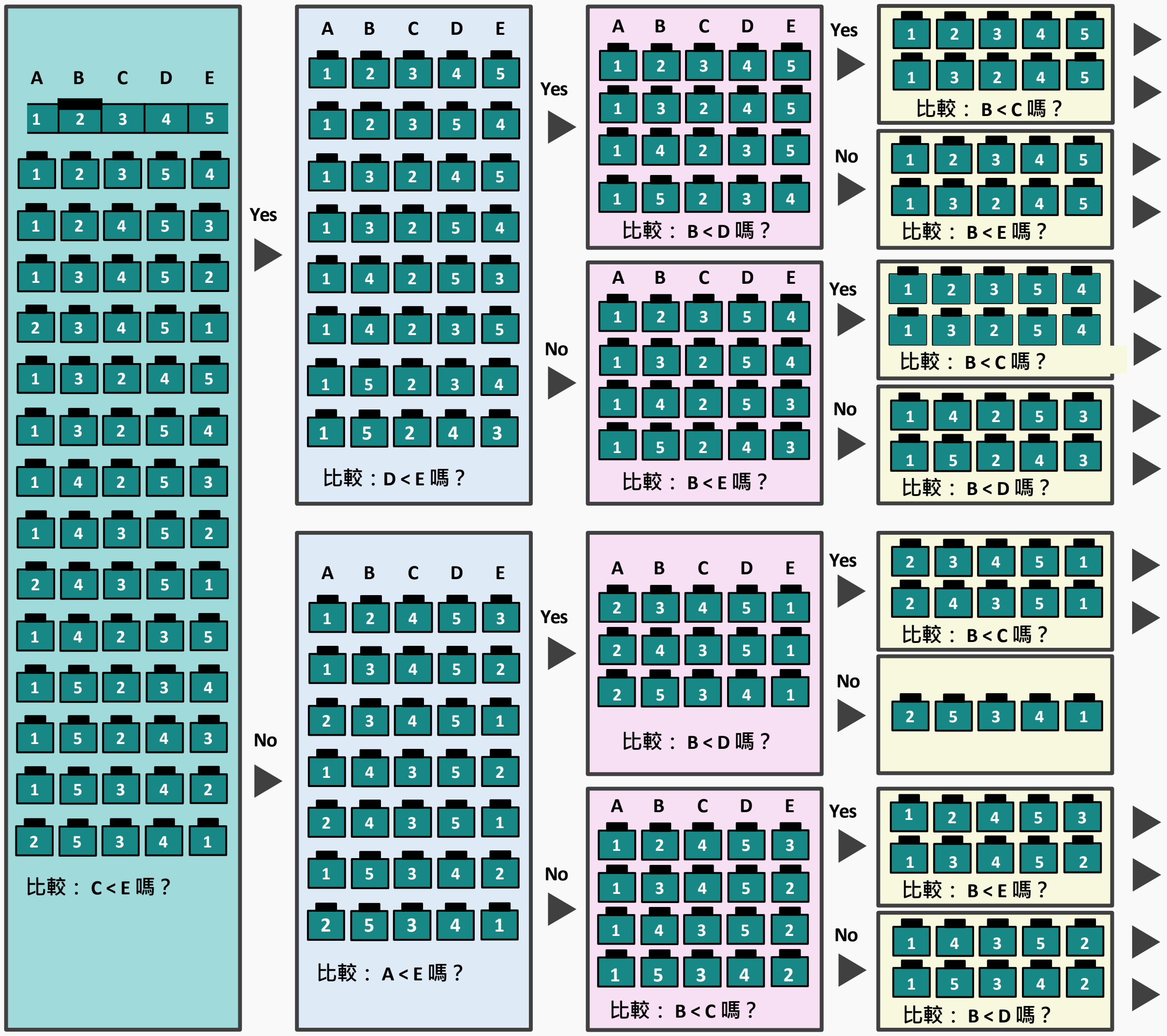
首先，前三次進行如下比較：

1. 比較砝碼 A 和砝碼 B。
2. 比較砝碼 C 和砝碼 D。
3. 比較 1 較輕的一方和 2 較輕的一方

首 3 次提問的結果有以下 8 種可能，但根據對稱性，即使假設「砝碼 A 最輕，砝碼 D 比砝碼 C 重」這種最左邊的模式，也不失普遍性。



接下來，最左邊的模式下的砝碼重量組合有 15 種，但藉由以下的比較，一定可以在 4 次確定答案。（數字為重量 [kg]）



問題 5.10.7 (2)

砝碼的重量組合有 $5! = 120$ 種， 6 次比較結果（左側、右側哪一邊比較重）的組合有 $2^6 = 64$ 種。因為後者較小，故無法用在 6 次內確定答案。

問題 5.10.7 (3)

首先，如以下說明，可證明最小次數為 45 次以上。

設砝碼的重量組合有 P 種，為了進行 L 次比較，必須滿足 $2^L \geq P$，也就是 $L \geq \log_2 P$ 。

因此，砝碼為16 個時， $\log_2 P = \log_2 16! = 44.2501 \ldots$ ，因此至少需要45次比較。

それ那麼，最小次數是多少呢？首先，將合併排序（**→3.6節**）運用到本問題上，進行以下操作：

- 「合併兩列包含 1 個砝碼的序列」 × 8 次
- 「合併兩列包含 2 個砝碼的序列」 × 4 次
- 「合併兩列包含 4 個砝碼的序列」 × 2 次
- 「合併兩列包含 8 個砝碼的序列」 × 1 次

回對包含 l 個砝碼的序列進行的合併操作需 $2l - 1$ 次比較（**→3.6.10項**），因此總計比較次數 $(1 \times 8) + (3 \times 4) + (7 \times 2) + (15 \times 1) = 49$ 次。

此外，截至2021年12月，研究出了在 46 次以內確定的方法。想知道詳細內容的讀者，請參閱以下論文。

- Peczarski, Marcin (2011). “Towards Optimal Sorting of 16 Elements”. Acta Universitatis Sapientiae. 4 (2): 215–224.

然而，最小次數到底是 45 次還是 46 次還未確定。有興趣的話請試著挑戰此未解決問題吧。