5.10 節末問題 5.10 の解答

問題 5.10.1

分配法則（→**5.10.2項**）を使うと、以下のように楽な計算で解くことができます。

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

この問題は、例題 2（→**5.10.2項**）の一般化です。

分配法則を使うと、シグマ記号の式について以下のことが分かります。

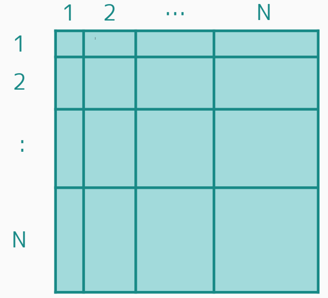
- $i = 1$ のときの総和：$(1 \times 1) + (1 \times 2) + \cdots + (1 \times N) = 1 \times (1 + 2 + \cdots + N)$
- $i = 2$ のときの総和：$(2 \times 1) + (2 \times 2) + \cdots + (2 \times N) = 2 \times (1 + 2 + \cdots + N)$
- ：
- $i = N$ のときの総和：$(N \times 1) + (N \times 2) + \cdots + (N \times N) = N \times (1 + 2 + \cdots + N)$

求めるべき答えは青色で示した値の総和であるため、分配法則より

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

となります。イメージが湧かない人は、右図の正方形の面積を考えてみると良いと思います。

縦の長さが $N(N+1)/2$、横の長さが $N(N+1)/2$ となっています。



したがって、以下のように答えを出力するプログラムを提出すると、正解が得られます。なお、この問題は制約が $N \leq 10^9$ と大きく、$N(N+1)/2 \times N(N+1)/2$ の値が 10^{30} を超える可能性があります。計算途中で余りを取る（→**4.6.1項**）などの工夫を行わなければ、`long long` 型などの 64 ビット整数でもオーバーフローを起こす可能性があるので注意してください。

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

const long long mod = 1000000007;
long long N;

int main() {
    // 入力
    cin >> N;

    // 答えを求める
    long long val = N * (N + 1) / 2;
    val %= mod;
    cout << val * val % mod << endl;
    return 0;
}
```

※ Python などのソースコードは chap5-10.md をご覧ください。

問題 5.10.3

以下のような立方体を考えると、この問題の答えが

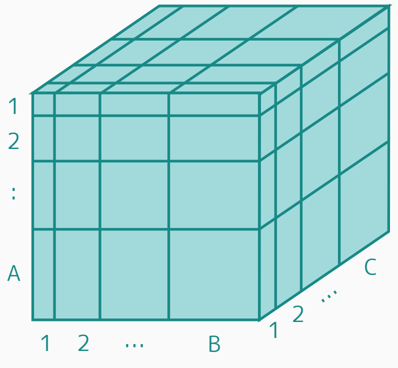
$$\sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{C} ijk = (1+\cdots+A)(1+\cdots+B)(1+\cdots+C)$$

であることが分かります。そこで和の公式（→**2.5.10項**）より、次式が成り立ちます。

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

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

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



よって、答えは $\frac{A(A+1)}{2} \times \frac{B(B+1)}{2} \times \frac{C(C+1)}{2}$ です。

したがって、以下のように答えを出力するプログラムを提出すると、正解となります。なお、本問題は制約が $A, B, C \leq 10^9$ と大きいので、オーバーフローを防ぐために

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

と変数をおいたうえで、計算途中で余りをとるなどの工夫をしています。

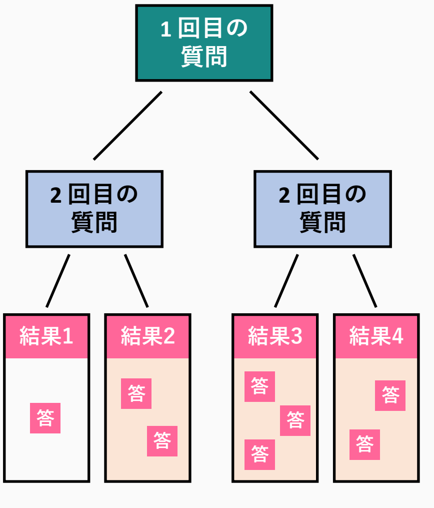
```
#include <iostream>
using namespace std;

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

int main() {
    // 入力
    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;

    // 答えを出力
    // ここで (D * E * F) % mod にしても、途中で 10^27 を扱う可能性がある
    // そのため、long long 型でもオーバーフローすることに注意！
    cout << (D * E % mod) * F % mod << endl;
    return 0;
}
```

※ Python などのソースコードは chap5-10.md をご覧ください。

問題 5.10.4

まず、太郎君の思い浮かべている数として 8 通りが考えられますが、2 回の質問に対する回答の組合せは「Yes→Yes」「Yes→No」「No→Yes」「No→No」の 4 通りしかありません。

$8 > 4$ ですから、右図のように「結果が 1 通りに定まっていないもの」が必ず存在します。そのため、2 回で確実に当てることは不可能です。



問題 5.10.5

自然に実装すると、以下のようになります。

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

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

        // 左辺と右辺の計算
        double v1 = log2(a);
        double v2 = b * log2(c);

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

しかし、このプログラムを提出すると 100 ケース中 15 ケースで不正解となってしまいます。その理由は誤差（→**5.10.1項**）です。

たとえば $(a,b,c) = (10^{18}-1, 18, 10)$ のケースでは、本当の答えは Yes なのに間違って No と出力しています。実際、

$$\begin{aligned} \log_2 a &= 59.794705707972522602\ldots \\ b\log_2 c &= 59.794705707972522616\ldots \end{aligned}$$

であり、左辺と右辺の相対誤差は 10^{-19} 程度です。あまりにも近すぎるので、コンピュータの限界を超えてしまい、「同じ数である」と判定してしまうのです。

改善方法①

それでは、誤差による不正解を防ぐにはどうすれば良いのでしょうか。一つの方法は全部整数で扱うことです。対数の性質（→**2.3.10項**）より、

$b\log_2 c = \log_2(c^b)$ であるから

$$\log_2 a < \boxed{b\log_2 c} \text{ のとき、} \log_2 a < \boxed{\log_2(c^b)}$$

よって、$a < c^b$

log の中身だけを取っても
大小関係は変わらない

であるため、$a < c^b$ であれば Yes、そうでなければ No と出力すれば良いです。

これを実装すると、以下のようになります。

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

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

    // 右辺の計算（c の b 乗）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        v *= c;
    }

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

しかし、これは不正解となってしまいます。その理由はオーバーフロー（→**5.10.1 項**）です。このプログラムは c^b の値をそのまま計算しますが、制約が $a, b, c \leq 10^{18}$ と大きく、最悪の場合 10^{18} の 10^{18} 乗を計算することになります。C++ はもちろん、Python でも計算することができません。

改善方法②

次に、オーバーフローを防ぐにはどうすれば良いのでしょうか。典型的な方法として「計算の途中で余りを取る」などが考えられますが、本問題は余りの計算ではないため、この方法は通用しません。

そこで、累乗を計算している途中で右辺の値が a を上回ったらこの時点で Yes 確定なので、ループ処理を打ち切るという対策が有効です。自然に実装すると、以下のようになります。

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

int main() {
    // 入力
```

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

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

    // 右辺の計算（c の b 乗）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        if (a / c < v) {
            // この条件分岐は a < (v * c) を言い換えただけ
            // 条件の言い換えをした理由は、v, c が 10^{18} 程度になる可能性があるため
            // a < v * c にすると最悪の場合 v * c = 10^{36} になりオーバーフローするから
            // 注：long long 型の限界は 2^{63}-1（約 10^{19}）
            cout << "Yes" << endl;
            return 0;
        }
        v *= c;
    }

    // ループが打ち切られない場合
    cout << "No" << endl;
    return 0;
}
```

しかし、このプログラムは 100 ケース中 2 ケースで実行時間制限超過（TLE）となります。その原因は $c = 1$ のケースです。

たとえば、$(a, b, c) = (2, 10^{18}, 1)$ のケースを考えましょう。1 は何乗しても 1 なので、「現在の右辺の値 v が a を超えたら打ち切る」という処理は効きません。そのため、$b = 10^{18}$ 回のループを行ってしまいます。

なお、$2^{60} > 10^{18}$ であるため、$c \geq 2$ のケースでは必ず 60 回以内のループで処理が終わるといえます。

改善方法③

最後に $c = 1$ のケースで場合分けをしましょう。この問題の制約では $a \geq 1$ なので、$c^b = 1$ より、答えは必ず No となります。したがって、次ページのようなプログラムを書くと、ようやく正解が得られます。

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

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

    // c = 1 のときの場合分け
    if (c == 1) {
        cout << "No" << endl;
        return 0;
    }

    // 右辺の計算（c の b 乗）
    long long v = 1;
    for (long long i = 1; i <= b; i++) {
        if (a / c < v) {
            // この条件分岐は a < (v * c) を言い換えただけ
            // 条件の言い換えをした理由は、v, c が 10^18 程度になる可能性があるため
            // a < v * c にすると最悪の場合 v * c = 10^36 になりオーバーフローするから
            cout << "Yes" << endl;
            return 0;
        }
        v *= c;
    }

    // ループが打ち切られない場合
    cout << "No" << endl;
    return 0;
}
```

※ Python などのソースコードは chap5-10.md をご覧ください。

問題 5.10.6

まず、$m = 1, 2, \dots, N$ についてそれぞれ調べる方法が考えられますが、制約が $N \leq 10^{11}$ と大きいため、実行時間制限超過（TLE）となってしまいます。

そこで、m としてあり得るパターンの数より $f(m)$ としてあり得るパターンの数の方が圧倒的に小さいため、以下のアルゴリズムが効率的です。

- $f(m)$ としてあり得る候補を全列挙する。
- $f(m)$ が決まれば $m = f(m) + B$ と決まるので、それぞれの候補について m の各桁の積が $f(m)$ と一致するかどうかをチェックする。

それでは $f(m)$ の候補はどうやって全列挙すれば良いのでしょうか。実は、1123 や 12233599 のような単調増加な数 m について $f(m)$ を計算するだけで良いです。なぜなら、数の順番を並べ替えても一般性を失わないからです。たとえば、

- $m = 1123$ 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 $f(m) = 3 \times 1 \times 1 \times 2 = 6$

とすべて同じになります。なお、単調増加な 11 桁以内の数はおよそ 30 万個しか存在せず、全列挙は十分現実的です。

したがって、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、単調増加な数 m は再帰関数 `func(digit, m)` で全列挙しており、`digit` は現在の桁数を表したものです。再帰関数が分からない人は、3.6 節に戻って確認しましょう。

また、関数 `product(m)` は整数 m の各桁の積を返すものです。数を 10 で割り続けていくことで、各桁の値を計算しています。 2 進数に変換するアルゴリズム（→**2.1.9項**）と似ています。

※注意：この C++ プログラムは、本書では扱っていない `set` 型を利用しています。知らない方は、インターネットなどで調べてみてください。（GitHub に掲載されている Python・JAVA のソースコードでも `set` 型が用いられています）

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

// f(m) としてあり得る候補
// set 型についてはインターネットで調べてみてください！
set<long long> fm_cand;

// m の各桁の積を返す関数
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;
```

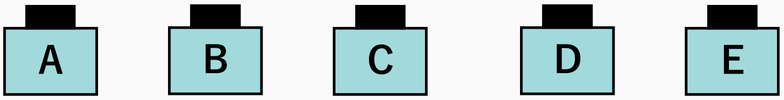
```
    }
}

void func(int digit, long long m) {
    // m の桁数は 11 桁以下
    // 注：余った桁を 1 で埋めれば、全部 11 桁と仮定しても良い
    int min_value = (m % 10);
    for (int i = min_value; i <= 9; i++) {
        // 10 * m + i は m の後ろに数字 i を付けたもの
        func(digit + 1, 10 * m + i);
    }
}

int main() {
    // f(m) の候補を列挙
    func(0, 0);

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

    // m - f(m) == B になるかどうかチェック
    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;
        }
    }

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

※ Python などのソースコードは chap5-10.md をご覧ください。

問題 5.10.7 (1)

この問題は様々な解法が考えられるので、そのうち一つを紹介します。

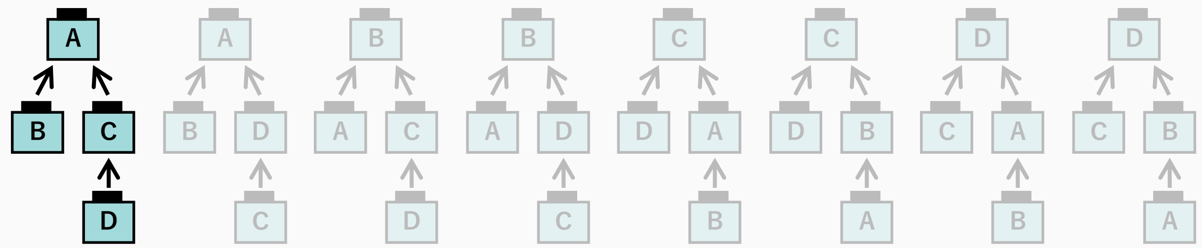
なお、ここでは説明の都合上、それぞれのおもりに A, B, C, D, E というラベルを付けることにします。



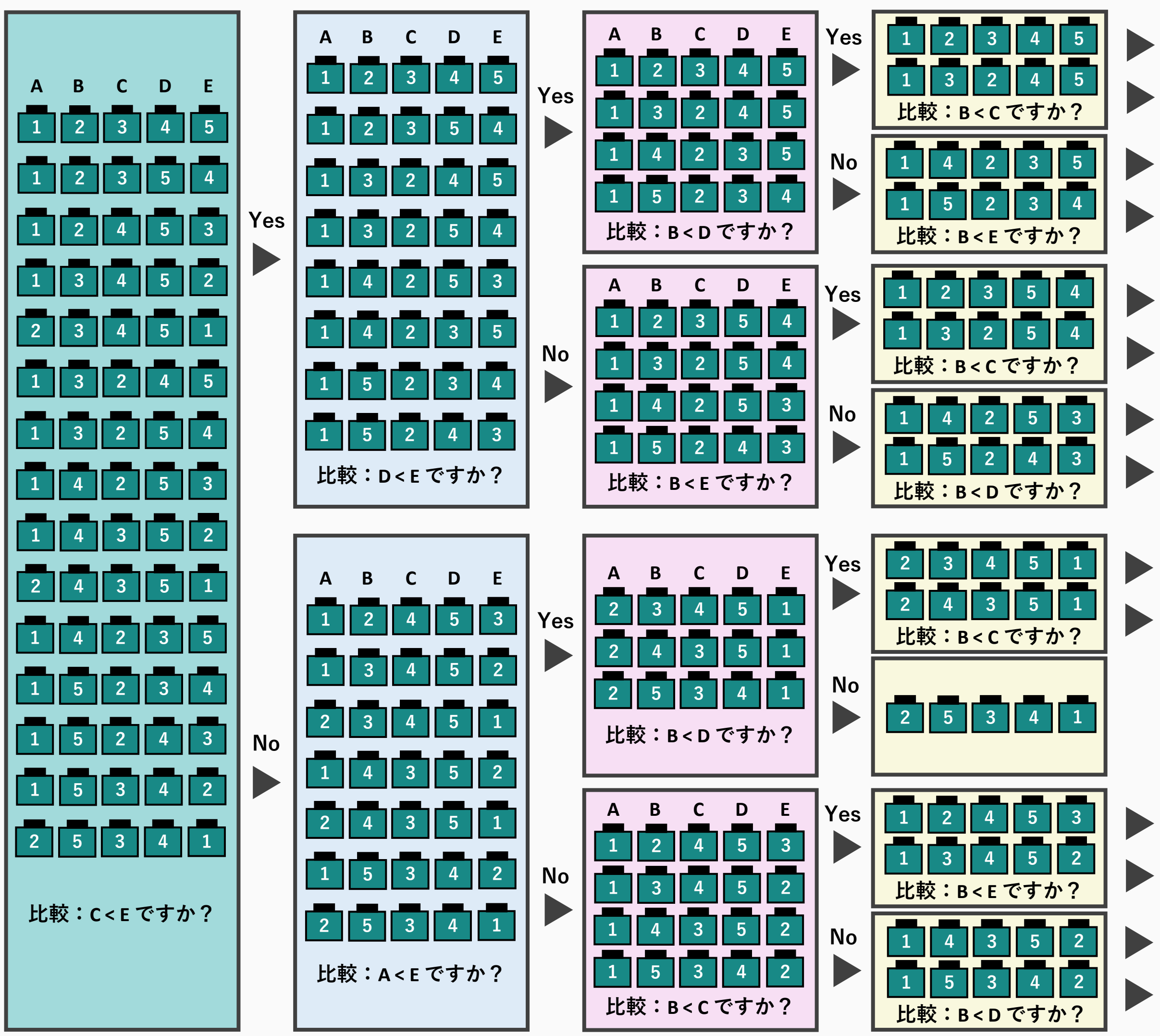
まず、最初の 3 回は以下のような比較を行います。

1. おもり A とおもり B を比較する。
2. おもり C とおもり D を比較する。
3. 1. の軽い方と 2. で軽い方を比較する。

3 回の質問の結果は以下の 8 通りがあり得ますが、対称性より「おもり A が最も軽く、おもり C よりおもり D の方が重い」という一番左のパターンであることを仮定しても、一般性を失いません。



さて、一番左のパターンとなるおもりの重さの組合せは 15 通りありますが、以下のような比較により、必ず 4 回で当てることが可能です。（数字は重さ [kg]）



問題 5.10.7 (2)

おもりの重さの組合せは $5! = 120$ 通り存在する一方、6 回の比較の結果（左側・右側のどちらが重いか）の組合せは $2^6 = 64$ 通りです。後者の方が小さいため、6 回で当てることはできません。

問題 5.10.7 (3)

まず、以下のようにして最小回数が 45 回以上であることが証明できます。

おもりの重さの組合せを P 通りとすると、L 回で比較を行うためには $2^L \geq P$ すなわち $L \geq \log_2 P$ を満たす必要があります。
そこでおもりが 16 個のとき $\log_2 P = \log_2 16! = 44.2501 \ldots$ となるため、少なくとも 45 回の比較が必要です。

それでは、何回が最小なのでしょうか。まず、マージソート（→**3.6節**）をこの問題に適用すると、以下のような操作を行うことになります。

- 「1 個のおもりの列 2 つを Merge する」× 8 回
- 「2 個のおもりの列 2 つを Merge する」× 4 回
- 「4 個のおもりの列 2 つを Merge する」× 2 回
- 「8 個のおもりの列 2 つを Merge する」× 1 回

l 個のおもりの列に対する Merge 操作では $2l - 1$ 回の比較を行うため（→**3.6.10 項**）、合計比較回数は $(1 \times 8) + (3 \times 4) + (7 \times 2) + (15 \times 1) = 49$ 回となります。

また、2021 年 12 月現在、46 回以内で確実に当てられる方法が考案されています。詳しく知りたい方は、以下の論文をお読みください。

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

しかしながら、最小回数が 45 回であるか、はたまた 46 回であるのかは誰も知りません。興味を持たれた方は、この未解決問題にぜひ挑戦してみましょう。