6.4 最終確認問題 16-20 の解答

問題 16

あり得るすべてのパターンをしらみつぶしに調べる方法を「全探索」（→**2.4.5 項**）といいます。

この問題では、$1 \leq a < b < c \leq N, a+b+c = X$ となる整数 (a, b, c) の組み合わせを完全に無駄なく列挙することは決して簡単ではありません。しかし、$1 \leq a < b < c \leq N$ となる (a, b, c) を列挙することは簡単であり、そのすべてを試して、その中で $a+b+c = X$ となる個数を数える、といった方法が最もシンプルです。

この解法を C++ で実装すると、以下のようになります。(a, b, c) のループ処理は、**コード 3.3.1** と似ています。

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

int main() {
    // 入力
    int N, X;
    cin >> N >> X;

    // すべての (a, b, c) の組み合わせを試す
    int answer = 0;
    for (int a = 1; a <= N; a++) {
        for (int b = a + 1; b <= N; b++) {
            for (int c = b + 1; c <= N; c++) {
                if (a + b + c == X) {
                    answer += 1;
                }
            }
        }
    }

    // 答えを出力
    cout << answer << endl;
    return 0;
}
```

※ Python などのソースコードは chap6-16_20.md をご覧ください。

問題 17

長方形の面積は（縦の長さ）×（横の長さ）と計算されます。これがどちらも整数であるから、面積が N となるためには、縦の長さ・横の長さが「N の約数」になる必要があります。

36 の約数 … 1, 2, 3, 4, 6, 9, 12, 18, 36

このため、**3.1.5 項**の方法で約数列挙すれば、可能な長方形を全て調べ上げることができます。この中で周の長さが最小になるものを求める事になります。

一方で、よりシンプルな解法もあります。(縦の長さ) ≦ (横の長さ) としても一般性を失わない (→**5.10.4 項**) ことを利用すると、(縦の長さ) ≦ √N になります。つまり、縦の長さ x を 1 以上 $\sqrt{N}$ 以下で全探索して、

- 横の長さ $N \div x$ が整数になるもの
- その中で周の長さ $2x + 2(N \div x)$ が最小になるもの

を求める事になります。計算量は $O(\sqrt{N})$ で、C++ での実装は以下のようになります。

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

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

    // 縦の長さを 1 から √N まで全探索
    long long answer = (1LL << 60);
    for (long long x = 1; x * x <= N; x++) {
        if (N % x == 0) {
            answer = min(answer, 2 * x + 2 * (N / x));
        }
    }

    // 答えを出力
    cout << answer << endl;
    return 0;
}
```

※ Python などのソースコードは chap6-16_20.md をご覧ください。

問題 18

整数 A, B の最大公約数を $\mathrm{GCD}(A, B)$、最小公倍数を $\mathrm{LCM}(A, B)$ とします。すると、$A \times B = \mathrm{GCD}(A, B) \times \mathrm{LCM}(A, B)$ という関係（→**2.5.2 項**）が成り立ちます。したがって、$\mathrm{GCD}(A, B)$ がユークリッドの互除法（→**3.2 節**）で計算できれば、最小公倍数は

$$\mathrm{LCM}(A, B) = A \times B \div \mathrm{GCD}(A, B)$$

と求められます。計算量は $O(\log(A + B))$ です。

しかし、このまま求めると C++ などの言語ではオーバーフローします。なぜなら、long long 型でも 19 桁程度までの数しか扱えず、$A \times B$ を計算する時点でこれを超える可能性があるからです。この問題は、以下の 2 つの方法で対処できます。

1. 計算順序を「$A \times B \div \mathrm{GCD}(A, B)$」から「$A \div \mathrm{GCD}(A, B) \times B$」に変える
2. 最小公倍数が 10^{18} を超えるかどうかを、うまい方法で判定する

2. に関しては、$A \times B \div \mathrm{GCD}(A, B) > 10^{18}$ つまり $A \div \mathrm{GCD}(A, B) > 10^{18} \div B$ であるかを判定すればよいです。左辺は必ず整数なので、右辺は整数に切り捨ててもうまくいきます。したがって、以下のようなプログラムを書けば正解できます。

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

long long GCD(long long A, long long B) {
    if (B == 0) return A;
    return GCD(B, A % B);
}

int main() {
    // 入力
    long long A, B;
    cin >> A >> B;

    // 最小公倍数が 10^18 を超えるかどうか判定
    if (A / GCD(A, B) > 1000000000000000000 / B) {
        cout << "Large" << endl;
    }
    else {
        cout << A / GCD(A, B) * B << endl;
    }
    return 0;
}
```

※ Python などのソースコードは chap6-16_20.md をご覧ください。

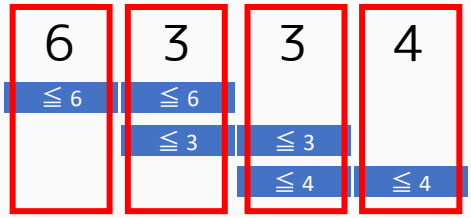
問題 19

この問題は、上界を考える（→**5.8 節**）テクニックを使うことで解けます。

一般の場合を考えるのは簡単ではないので、まずは具体例として $N = 4, B_1 = 6, B_2 = 3, B_3 = 4$ の場合を考えましょう。このとき、

- $\max(A_1, A_2) \leq 6$ … 「A_1 と A_2 は両方 6 以下」
- $\max(A_2, A_3) \leq 3$ … 「A_2 と A_3 は両方 3 以下」
- $\max(A_3, A_4) \leq 4$ … 「A_3 と A_4 は両方 4 以下」

と言い換えられます。したがって、A_1 は 6 以下、A_2 は「6 以下かつ 3 以下」だから 3 以下、A_3 は「3 以下かつ 4 以下」だから 3 以下、A_4 は 4 以下と分かります。



一般の場合も同様にして、$A_1 \leq B_1$、$A_i \leq \min(B_{i-1}, B_i)$ $(2 \leq i \leq N-1)$, $A_N \leq B_{N-1}$ という上界が得られます。実際に、この上界を当てはめた数列 A は条件を満たし、その合計を求めるプログラムを書けば正解です。以下が C++ での実装例です。

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

int N, B[109];
int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N - 1; i++) {
        cin >> B[i];
    }

    // 数列 A の要素の合計を求める → 答えの出力
    int answer = B[1] + B[N - 1];
    for (int i = 2; i <= N - 1; i++) {
        answer += min(B[i - 1], B[i]);
    }
    cout << answer << endl;
    return 0;
}
```

※ Python などのソースコードは chap6-16_20.md をご覧ください。

問題 20

この問題は、各質問に対して合計を以下のプログラムのように求めると、計算量 $O(NQ)$ となり、実行時間制限オーバー (TLE) になってしまいます。

```
int answer1 = 0, answer2 = 0;
for (int i = L; i <= R; i++) {
    if (C[i] == 1) answer1 += P[i];
    if (C[i] == 2) answer2 += P[i];
}
```

これを高速化するために、累積和（→**4.2 節**）を使いましょう。まずは、１組の合計得点だけを求めることを考えます。学籍番号 i の生徒が１組のとき $A_i = P_i$、２組のとき $A_i = 0$ とすると、１組の合計得点は $A_L + A_{L+1} + \cdots + A_R$ と計算されます。したがって、累積和を使えば各質問に $O(1)$ で答えられます。

学籍番号・組	1	2	3	4	5
得点 P_i	50	80	100	30	40
１組の得点 A_i	50	80	0	30	0
累積和	50	130	130	160	160

… １組

… ２組

２組に対しても同様のことができます。すると、全体計算量 $O(N+Q)$ でこの問題が解けます。この解法を C++ で実装すると、以下のようになります。

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

int N, C[100009], P[100009], L[100009], R[100009], S1[100009], S2[100009];
int main() {
    // 入力 → 累積和を求める
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> C[i] >> P[i];
    for (int i = 1; i <= N; i++) S1[i] = S1[i - 1] + (C[i] == 1 ? P[i] : 0);
    for (int i = 1; i <= N; i++) S2[i] = S2[i - 1] + (C[i] == 2 ? P[i] : 0);

    // 質問に答える
    cin >> Q;
    for (int i = 1; i <= Q; i++) {
        cin >> L[i] >> R[i];
        cout << S1[R[i]] - S1[L[i] - 1] << " " << S2[R[i]] - S2[L[i] - 1] << endl;
    }
    return 0;
}
```

※ Python などのソースコードは chap6-16_20.md をご覧ください。