2020 年度 電気電子情報工学実験 II (b)
実践的・競技プログラミング 宿題解説

廣田悠輔
y-hirota@u-fukui.ac.jp

2020 年 7 月 13 日

目次

1 問題 X：フクイ国の両替

解説

所持金 N フクをすべてを硬貨の枚数が最小になるように両替したときの，1 フク硬貨，10 フク硬貨，40 フク硬貨，50 フク硬貨，200 フク硬貨の枚数をそれぞれ $a^{(1)}$，$a^{(10)}$，$a^{(40)}$，$a^{(50)}$，$a^{(200)}$ とおく．このときの 1 フク硬貨の枚数が $a^{(1)} \geq 10$ であると仮定する．すると，別の硬貨の組み合わせ「1 フク硬貨 $a^{(1)} - 10$ 枚，10 フク硬貨 $a^{(10)} + 1$ 枚，40 フク硬貨，50 フク硬貨，200 フク硬貨がそれぞれ $a^{(40)}$ 枚，$a^{(50)}$ 枚，$a^{(200)}$ 枚」は，その合計金額が N フクかつ硬貨合計枚数が元の両替の合計枚数より少なくなる．このことは，元の両替の硬貨合計枚数が最小であることに矛盾する．よって，$a^{(1)} \leq 9$ である．同様の議論により，10 フク硬貨，40 フク硬貨，50 フク硬貨の枚数について $a^{(10)} \leq 3$，$a^{(40)} \leq 4$，$a^{(50)} \leq 3$ が成り立つ．一方，硬貨の合計金額が N フクを超えないようにするため $200a^{(200)} \leq N$．したがって，硬貨の合計枚数が最小になる両替を行ったときの各硬貨の枚数は以下の条件を満たす．

$$\begin{aligned} 0 \leq a^{(1)} \leq 9, \quad 0 \leq a^{(10)} \leq 3, \quad 0 \leq a^{(40)} \leq 4, \\ 0 \leq a^{(50)} \leq 3, \quad 0 \leq a^{(200)} \leq \lfloor N/200 \rfloor. \end{aligned} \tag{1}$$

ただし，$\lfloor a \rfloor$ は a 以下の最大の整数（a の床関数値）である．本問題では N は最大で 10^7 であるので，このような範囲の $a^{(1)}$，$a^{(10)}$，$a^{(40)}$，$a^{(50)}$，$a^{(200)}$ の組み合わせの数は最大 $4N = 4 \times 10^7$ である．したがって，(1) の範囲を全探索する五重ループにより，それぞれの組み合わせについて合計金額が N フクとなるか，N フクとなる場合に硬貨の合計枚数が何枚になるか調べるプログラムを書けば問題に答えられる．

以下に示す追加の考察を行うことで，探索すべき範囲をさらに削減することができる．

- 1 フク硬貨以外をどのように組み合わせても 1 フクから 9 フクの金額を作ることはできない．一方，$0 \leq a^{(1)} \leq 9$ である．したがって，硬貨の枚数が最小となる組み合わせにおける 1 フク硬貨の枚数

は $a^{(1)} = N \bmod 10$ と他の硬貨とは独立して決定できる．ただし，$a \bmod b$ は a を b で割った余りである．

- 範囲 (1) のもとで 10 フク硬貨，40 フク硬貨，50 フク硬貨のみを組み合わせて作ることができる 200 フク以上の金額は，200 フク，210 フク，220 フク，...，340 フクの 15 通りである．これらの金額は，200 フク硬貨を 1 枚含む組み合わせによって，より少ない硬貨の合計枚数で作ることができる．したがって，硬貨の枚数が最小となる組み合わせにおける 200 フク硬貨の枚数は $a^{(200)} = \lfloor N/200 \rfloor$ と他の硬貨とは独立して決定できる．

したがって，1 フク硬貨，200 フク硬貨の枚数は上記の方法で独立して決定し，10 フク硬貨，40 フク硬貨，50 フク硬貨の枚数は (1) の範囲を調べる三重ループによって探索することにより，O(1) の計算量で問題を解くことができる．

正答例

```c
#include <stdio.h>

int main(void) {
  int n, rest;
  int sum;
  int min_10_40_50;
  int i, j, k;

  scanf("%d", &n);

  sum = n / 200; // 200-Fuku coin
  rest = n % 200;

  min_10_40_50 = 9999; // Huge value
  for (i = 0; i <= 3; i++) { // Num. 10-Fuku
    for (j = 0; j <= 4; j++) { // Num. 40-Fuku
      for (k = 0; k <= 3; k++) { // Num. 50-Fuku
        if (i * 10 + j * 40 + k * 50 == (rest / 10) * 10) {
          if (i + j + k < min_10_40_50) {
            min_10_40_50 = i + j + k;
          }
        }
      }
    }
  }
  sum += min_10_40_50;

```

```
28    rest %= 10;
29    sum += rest; // 1-Fuku coin
30
31    printf("%d\n", sum);
32
33    return 0;
34  }
```

2　問題 Y：カードシャッフル

解説

目的の出力を得る最も簡単な方法は，Algorithm 1 に示すように，カード交換の様子をプログラム上でシミュレーションしてしまうことである．しかしながら，このような方法の計算量は $\mathrm{O}(ML)$（出力部分を考慮したより正確な評価は $\mathrm{O}(ML+N)$）であり，本問題の M, L の最大値はそれぞれ 10^5, 10^6 であるため，制限時間内に出力を得ることは極めて困難である．

続いて，ML 回のカード交換を回避する方法を考える．毎回のカード交換を互換と考え，その積を取れば L 回の交換に相当する置換 σ が計算できる．したがって，最初に与えられたカードの数値の並びに対して置換 σ に対応するように並び替える処理を M 回繰り返せば目的の出力が得られる．アルゴリズムを Algorithm 2 に示す．例えば，$N=4$，$L=3$，$(A_{1,1},A_{1,2})=(2,3)$，$(A_{2,1},A_{2,2})=(1,3)$，$(A_{3,1},A_{3,2})=(2,4)$ である場合，L 回の交換に相当する置換 σ は，

$$\sigma=\sigma_{2,4}\sigma_{1,3}\sigma_{2,3}=\begin{pmatrix}1&2&3&4\\2&4&1&3\end{pmatrix}$$

となる．ただし，$\sigma_{p,q}$ は p と q を交換する互換

$$\sigma_{p,q}=\begin{pmatrix}1&2&\cdots&p-1&p&p+1&\cdots&q-1&q&q+1&\cdots&N\\1&2&\cdots&p-1&q&p+1&\cdots&q-1&p&q+1&\cdots&N\end{pmatrix}$$

である．さらに，$M=3$，最初のカードの並び順を $(C_1,C_2,C_3,C_4)=(11,22,33,44)$ としたとき，カードは $(11,22,33,44)\to(22,44,11,33)\to(44,33,22,11)\to(33,11,44,22)$ と並び替えられる．このアルゴリズムの計算量は，置換 σ を求めるのに $\mathrm{O}(L)$，M 回の置換の適用に $\mathrm{O}(MN)$ の計算量が必要であり，全体の計算量は $\mathrm{O}(MN+L)$ となる．本問題の M, L, N の最大値はそれぞれ 10^5, 10^6, 10^4 であるので，制限時間内に確実に出力を得るのは難しい（良いデータ構造を選択して実装すればかろうじて間に合うかもしれない）．

より少ない計算量で目的の出力を得る方法に，ダブリングを応用するものがある．M は重複のない 2 のべき乗である K 個の自然数 $t_1,t_2,\ldots,t_K$ により

$$M=\sum_{i=1}^{K}t_i$$

と表される（$K\le\log M$）．このとき，M 回の置換の積は

$$\sigma^M=\sigma^{t_K}\sigma^{t_{K-1}}\cdots\sigma^{t_1}\tag{2}$$

と表現することができる．例えば，$M = 13$ の場合，$M = 8 + 4 + 1$ であるので，

$$\sigma^M = \sigma^8 \sigma^4 \sigma$$

と表現できる．置換 σ の 2 乗，4 乗，8 乗，... は，

$$\sigma^{2i} = \sigma^i \sigma^i \quad (i = 1, 2, \ldots)$$

を順に計算することで求められる．したがって，置換の 2 乗，4 乗，8 乗，... をあらかじめ計算しておき，最初に与えられたカードの数値の並びに対して (2) の右辺に現れるものを順に適用すれば，目的の出力が得られる（Algorithm 3）．このアルゴリズムは，σ を求めるのに $\mathrm{O}(L)$（2–4 行目），σ^2, σ^4, σ^8 ... を求めるのに $\mathrm{O}(N \log M)$（5–7 行目），σ, σ^2, σ^4 ... から必要なものを適用するのに $\mathrm{O}(N \log M)$（8–10 行目）の計算量が必要である．したがって，全体の計算量は $\mathrm{O}(N \log M + L)$ となり，制限時間内に余裕をもって問題を解くことができる．

参考までに対角化の応用による方法を紹介する．この方法は実装が難しく実用は見込めないが，$\mathrm{O}(N^3 + L)$ と M に依らない計算量で目的の出力が得られる興味深いものである．L 回のカード交換に相当する置換

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & N \\ s_1 & s_2 & \cdots & s_N \end{pmatrix}$$

に対して，

$$(P)_{i,j} = \begin{cases} 1 & (j = s_i) \\ 0 & (\text{otherwise}) \end{cases}$$

である行列 P を考える（置換行列）．行列 P とベクトル $\boldsymbol{x} = [x_1, x_2, \ldots, x_N]^\top$ の積は，

$$P\boldsymbol{x} = \begin{bmatrix} x_{s_1} \\ x_{s_2} \\ \vdots \\ x_{s_N} \end{bmatrix}$$

と，$\boldsymbol{x}$ の要素を置換 σ に対応して並び替えたものになる．したがって，最終的なカードの並び替え結果は，最初に与えられたカードの数値を順に並べたベクトルを

$$\boldsymbol{c} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix}$$

としたとき，

$$P^M \boldsymbol{c}$$

の要素を順に並べたものになる．一方，置換行列 P は正規行列である（$PP^{\mathrm{H}} = P^{\mathrm{H}}P$ である）ので，ある対角行列 Λ と正則行列 B を使って

$$P = B\Lambda B^{-1}$$

と分解できる．したがって，

$$\begin{aligned} P^M \boldsymbol{c} &= (B\Lambda B^{-1})^M \boldsymbol{c} \\ &= B\Lambda^M B^{-1} \boldsymbol{c} \end{aligned}$$

Algorithm 1 カード交換の様子のシミュレーションによる方法

1: **for** $i = 1, 2, \ldots, M$ **do**
2: **for** $j = 1, 2, \ldots, L$ **do**
3: Swap the $A_{j,1}$-th card for the $A_{j,2}$-th card.
4: **end for**
5: **end for**
6: Output the card numbers in the current order.

Algorithm 2 L 回のカード交換に相当する置換を計算する方法

1: $\sigma \leftarrow \begin{pmatrix} 1 & 2 & \cdots & N \\ 1 & 2 & \cdots & N \end{pmatrix}$ (an identity permutation)
2: **for** $j = 1, 2, \ldots, L$ **do**
3: $\sigma \leftarrow \sigma_{A_{j,1},A_{j,2}}\sigma$ where $\sigma_{p,q}$ is a transposition that swaps p and q.
4: **end for**
5: **for** $i = 1, 2, \ldots, M$ **do**
6: Apply the permutation σ to the number sequence.
7: **end for**
8: Output the card numbers in the current order.

となる．Λ^M は Λ の対角要素をそれぞれ M 乗することで求まる．以上の事実から，Algorithm 4 に示す方法で最終的なカードの並びが求まることが分かる．アルゴリズムの 2–4 行目の計算量は $\mathrm{O}(L)$ であり，5–9 行目の計算量は $\mathrm{O}(N^3)$ である[*1] ので，全体の計算量は $\mathrm{O}(N^3 + L)$ と M に依らないものになる．しかしながら，本手順は複素数を扱う必要があるため，プログラム作成の際は慎重に数値誤差を扱う必要がある．特にステップ 2 に示した分解（非対称固有値問題の全固有値と全固有ベクトルの計算）は数値誤差の扱いが難しい．また，よほど N が小さく M が大きいケースでなければ，ダブリングによる $\mathrm{O}(N \log M + L)$ の方法と比べて計算量の面でも優位性がない．このため，対角化による方法の実用性は低い．

正答例（ダブリングによる方法）

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void permute_int_array(int n, int *c, int *permutation, int *iwork) {
5   int i;
6   for (i = 0; i < n; ++i) {
7     iwork[i] = c[permutation[i]];
8   }
9   for (i = 0; i < n; ++i) {
```

[*1] より正確に書くと，5 行目の固有値分解の厳密な値は有限回の計算で求めることはできないが，精度が良い近似値が $\mathrm{O}(N^3)$ の計算量で求まる．

Algorithm 3 ダブリングによる方法

$\sigma \leftarrow \begin{pmatrix} 1 & 2 & \cdots & N \\ 1 & 2 & \cdots & N \end{pmatrix}$ (an identity permutation)
for $j = 1, 2, \ldots, L$ **do**
  $\sigma \leftarrow \sigma_{A_{j,1},A_{j,2}}\sigma$ where $\sigma_{p,q}$ is a transposition that swaps p and q.
end for
for $i = 2, 4, 8, \ldots, \lfloor 2^{\log_2 M} \rfloor$ **do**
  $\sigma^i \leftarrow \sigma^{i/2}\sigma^{i/2}$
end for
for $p = 1, 2, \ldots, K$ **do**
  Apply the permutation σ^{t_p} for the current number sequence.
end for
Output the card numbers in the current order.

Algorithm 4 対角化による方法

$\sigma \leftarrow \begin{pmatrix} 1 & 2 & \cdots & N \\ 1 & 2 & \cdots & N \end{pmatrix}$ (an identity permutation)
for $j = 1, 2, \ldots, L$ **do**
  $\sigma \leftarrow \sigma_{A_{j,1},A_{j,2}}\sigma$ where $\sigma_{p,q}$ is a transposition that swaps p and q.
end for
$P \to B\Lambda B^{\top}$ (eigen decomposition)
$\boldsymbol{d} \leftarrow B^{-1}\boldsymbol{c}$
$\Lambda' \leftarrow \Lambda^M$
$\boldsymbol{d}' \leftarrow \Lambda'\boldsymbol{d}$
$\boldsymbol{d}'' \leftarrow B\boldsymbol{d}'$
Output the elements of $\boldsymbol{d}''$ in order.

```
    c[i] = iwork[i];
  }
}

void swap_int(int *i, int *j) {
  int tmp = *i;
  *i = *j;
  *j = tmp;
}

int main(void) {
  int m, l;
  int *a1, *a2;
```

```
23  int n;
24  int *c, *iwork;
25  int **permutation;
26  int p_max;
27  int i, j;
28
29  // Read input data.
30  scanf("%d␣%d", &m, &l);
31  a1 = malloc(sizeof(int) * l);
32  a2 = malloc(sizeof(int) * l);
33  for (i = 0; i < l; ++i) {
34    scanf("%d␣%d", &a1[i], &a2[i]);
35
36    // Use 0-origin index.
37    a1[i]--;
38    a2[i]--;
39  }
40  scanf("%d", &n);
41  c = malloc(sizeof(int) * n);
42  for (i = 0; i < n; ++i) {
43    scanf("%d", &c[i]);
44  }
45
46  // 1, 2, 4, ..., 2^(p_max - 1)-step permutation storage.
47  p_max = 1;
48  for (i = 2; i <= m; i *= 2) {
49    p_max++;
50  }
51  permutation = malloc(sizeof(int *) * p_max);
52  for (i = 0; i < p_max; ++i) {
53    permutation[i] = malloc(sizeof(int) * n);
54  }
55
56  // Calculate the 1-step permutation.
57  for (i = 0; i < n; ++i) {
58    permutation[0][i] = i;
59  }
60  for (i = 0; i < l; ++i) {
61    swap_int(&permutation[0][a1[i]], &permutation[0][a2[i]]);
62  }
```

```
63
64    // Calculate the 2, 4, 8, ..., 2^(p_max - 1)-step permutations.
65    for (i = 1; i < p_max; ++i) {
66      for (j = 0; j < n; ++j) {
67        permutation[i][j] = permutation[i - 1][permutation[i - 1][j]];
68      }
69    }
70
71    // Apply the 1, 2, 4, 8, ..., 2^(p_max - 1)-step permutations if
          needed.
72    iwork = malloc(sizeof(int) * n);
73    for (i = m, j = 0; i >= 1; i /= 2, j++) {
74      if (i % 2 == 1) {
75        permute_int_array(n, c, permutation[j], iwork);
76      }
77    }
78
79    // Print the results.
80    for (i = 0; i < n; ++i) {
81      printf("%d\n", c[i]);
82    }
83
84    free(iwork);
85    free(permutation);
86    free(c);
87    free(a2);
88    free(a1);
89    return 0;
90  }
```