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 第 1 回コンテスト解説

廣田悠輔
y-hirota@u-fukui.ac.jp

2020 年 6 月 4 日

目次

1 問題 A：こんにちはこんにちは!!

コメント

scanf による標準入力からの変数の読み込み，for による繰り返し，printf による標準出力ができれば難なく正答できるはずである．この問題を解くのに苦労した場合，標準入出力の意味の理解や，C 言語の復習に時間をとること．

解答例

```
1 #include <stdio.h>
2
3 int main(void) {
4   int n;
5   int i;
6   scanf("%d", &n);
7
8   for (i = 0; i < n; ++i) {
9     printf("Hello.\n");
```

```
  }

  return 0;
}
```

2　問題 B：最小公倍数と最大公約数

解説

最初に最大公約数 B を求め，その結果を使って最小公倍数 A を求めるのが簡単である（問題名や出力の順序と逆である点に注意）．最大公約数 B の計算は Euclid の互除法により求められる．最小公倍数 A は

$$A = \frac{NM}{B}$$

で求められるので，その通りの計算を行えば良い．

解答例

```
#include <stdio.h>

int gcd(int n, int m) {
  // Assume 1 <= n <= m.
  while (m % n != 0) {
    int p = m % n;
    m = n;
    n = p;
  }
  return n;
}

int main(void) {
  int n, m;
  int a, b; // (n, m)'s LCM, GCD.
  scanf("%d␣%d", &n, &m);

  b = gcd(n, m);
  a = n / b * m;
  printf("%d␣%d\n", a, b);

  return 0;
}
```

3　問題 C：簡単な制約付き整数方程式

解説

変数は A, B, C, D の 4 個しかなく，それらの取りうる値はいずれも $-M, -M+1, ..., M$ に限定される．したがって，A, B, C, D 取りうる値の組み合わせの数は

$$(2M+1)^4$$

である．例えば，$M = 1$ のとき，取りうる値の組み合わせは

- $A = -1, B = -1, C = -1, D = -1$
- $A = -1, B = -1, C = -1, D = 0$
- $A = -1, B = -1, C = -1, D = 1$
- $A = -1, B = -1, C = 0, D = -1$
- $A = -1, B = -1, C = 0, D = 0$
- $A = -1, B = -1, C = 0, D = 1$
- $A = -1, B = 0, C = -1, D = -1$

  ⋮

- $A = 1, B = 1, C = 0, D = 1$
- $A = 1, B = 1, C = 1, D = -1$
- $A = 1, B = 1, C = 1, D = 0$
- $A = 1, B = 1, C = 1, D = 1$

の 81 通りとなる．制約

$$1 \leq M \leq 10$$

が与えられているので，A, B, C, D 取りうる値の組み合わせの数は最大でも 21^4 $(= 194481)$ でしかない．したがって，すべての組み合わせについて方程式の変数に値を代入して等式が成り立つかどうかを調べ，成り立つケースを数え上げれば良い．

すべての組み合わせを網羅する簡単な方法には

- 四重ループにより各変数がとりうる値を網羅する方法，
- 再帰による深さ優先探索

がある．いずれを使っても良い．ただし，前者のような他重ループによる網羅は本問題のように変数が高々数個に固定されている場合には容易に使用できるが，例えば数十個の変数を持つ場合や，変数の数が入力依存である場合には使用困難であることに注意せよ．そのような場合には，深さ優先探索を使用することになる．

解答例（4 重ループ）

```
1 #include <stdio.h>
2
```

```
3  int n_solutions(int n, int m) {
4    int a, b, c, d;
5    int count = 0;
6    for (a = -m; a <= m; ++a) {
7      for (b = -m; b <= m; ++b) {
8        for (c = -m; c <= m; ++c) {
9          for (d = -m; d <= m; ++d) {
10           if (a * b + c * d * 2 + 1 == n) {
11             count++;
12           }
13         }
14       }
15     }
16   }
17   return count;
18 }
19
20 int main(void) {
21   int n, m;
22   scanf("%d", &n);
23   scanf("%d", &m);
24
25   printf("%d\n", n_solutions(n, m));
26
27   return 0;
28 }
```

解答例（再帰）

```
1  #include <stdio.h>
2
3  int n_solutions_recursive(int n, int m, int *x, int depth) {
4    int counter = 0;
5
6    if (depth == 4) {
7      if (x[0] * x[1] + x[2] * x[3] * 2 + 1 == n) {
8        return 1;
9      } else {
10       return 0;
11     }
```

```
12    }
13
14    for (x[depth] = -m; x[depth] <= m; ++x[depth]) {
15      counter += n_solutions_recursive(n, m, x, depth + 1);
16    }
17    return counter;
18  }
19
20  int main(void) {
21    int n, m;
22    int x[4];
23    scanf("%d", &n);
24    scanf("%d", &m);
25
26    printf("%d\n", n_solutions_recursive(n, m, x, 0));
27
28    return 0;
29  }
```

4　問題 D：それにつけても金の欲しさよ

解説[*1]

与えられた問題文を素直に解釈すると次のように読める．

> n 個の袋から m 個を選ぶ組み合わせの中で総和が最大になるものを探せ．

ところが，この解釈そのままにすべての組み合わせを調べるアルゴリズムに落とし込もうとすると，候補となる組み合わせの数が膨大になってしまう．そこで，問題文を以下のように読み替える．

> n 個の袋から金額の大きい m 個を選び，その総和を求めよ．

ここまで辿り着ければ，以下の解法に自然に落とし込める．

1. 入力を読み込む．
2. 袋を金額の大きい順に並び替える（降順ソート）．
3. 金額の大きい袋から n 個について総和を計算する．

ただし，ソートについては実行時間の制限から，バブルソートなどの $\mathrm{O}(n^2)$ のソートではなく，マージソートなどの $\mathrm{O}(n \log n)$ のソートを使用する必要がある．

[*1] ここでは最小限の説明のみを行う．詳細については別資料「競技プログラミングの問題を解くための考え方」を参考にされたい．

解答例

```c
#include <stdio.h>
#include <stdlib.h>

int descending_int(const void *a, const void *b) {
  return *((int *)b) - *((int *)a);
}

int main(void) {
  int n, m;
  int *a;
  int i;
  int sum;

  scanf("%d", &n);
  scanf("%d", &m);
  a = malloc(sizeof(int) * n);
  for (i = 0; i < n; ++i) {
    scanf("%d", &a[i]);
  }

  qsort(a, n, sizeof(int), descending_int);
  sum = 0;
  for (i = 0; i < m; ++i) {
    sum += a[i];
  }
  printf("%d\n", sum);

  free(a);
  return 0;
}
```

5 問題 E：一夜漬け

解説

参考書を読む区間の開始ページを S，終了ページを E とおくと，問題は以下のように読み替えられる．

次の値を求めよ．

$$\min_{1 \le S \le E \le N} E - S + 1$$
$$\text{where} \quad \sum_{i=S}^{E} A_i \ \ge \ P.$$

このような問題を解くシンプルな方法は，$1 \le S \le E \le N$ を満たす S, E の全組み合わせ約 $N^2/2$ 個を調べることである．しかしながら，本問題では最大で $N = 10^5$ が与えられる可能性があるため，個々の組み合わせをどのように高速に調べたとしても制限時間超過となってしまう．

上述のシンプルな方法よりも効率的な探索方法にしゃくとり法がある．しゃくとり法は，Algorithm 1 のように，$S = 1, 2, \ldots, N$ について，それぞれの S で条件を満たし $E - S + 1$ が最小になる E_S を探索し，最後に $E_S - S + 1$ $(S = 1, 2, \ldots, N)$ の中で最小となるを求めるというのが基本的な考え方となる．このような大枠は前述のシンプルな方法に類似しているが，しゃくとり法では，Algorithm 1 の 2 行目の E_S の探索の際，過去の探索結果の情報を使って探索範囲を絞り込むことで探索回数を大きく削減し，効率的な探索を実現している．

具体的な処理の流れについて，図 1 に示す $N = 10$, $P = 7$ の例を使って説明する．まず $S = 1$ の場合に，ページ数最小で条件を満たす区間（すなわち条件を満たす最小の E）を求める．これには単に $E = 1, 2, \ldots$ と順に E の値を増やしていき，初めて

$$\sum_{i=S(=1)}^{E} A_i \ \ge \ P$$

を満たす E を見つければ良い（一度条件を満たす E をみつければ，それより大きな値については探索不要）．図 1 の例では $S = 1$ のとき，$E = 3$ ではじめて項目数和が条件を満たす．次に $S = 2$ の場合を考えるが，ここからは一つ前の結果（$S = 2$ の場合には $S = 1$ の結果）を利用することで，無駄な探索を回避する．$S = 1$ では $E = 3$ のときに初めて条件を満たしのたであるのだから，$S = 2$ では $E < 3$ では条件を満たすことは絶対にない（各ページの項目数が非負であることに注意せよ）．したがって，$S = 2$ では $E = 3, 4, \ldots$ のみを順に調べていけば良いことになる．また，$S = 2$ から $E = 3$ の範囲の項目数和は，$S = 1$ から $E = 3$ の範囲の項目数和から A_1 を引いた値であるので，開始時の項目数和の和についても $S = 1$ での探索情報を再利用できる．以下，$S = 3, 4, \ldots$ の場合も同様であり，直前の探索結果を利用することで，探索範囲の絞り込み，項目数和の情報の再利用が可能となる．最終的には，S がある大きな値となったところで，E を最大値である N としたところで条件を満たさなくなるなるので，以後の探索は打ち切ることができる．例えば，図 1 では $S \ge 6$ から先は条件を満たさないので，探索不要である．

しゃくとり法の全体像を Algorithm 2 に示す．このアルゴリズムを見ればわかるとおり，しゃくとり法の計算量は $\mathrm{O}(N)$ となる（S, E' がそれぞれ高々 N 回しか変化していないことに注意）．

ノート

- 本問題に対するしゃくとり法の適用は，すべてのページの項目数が非負の値であることに依存している．このような条件が成り立たない問題にはしゃくとり法が適用できないことに注意せよ．
- しゃくとり法については [1, pp. 135–138] などにも解説が掲載されている．
- 解説中のアルゴリズムと，ソースコードではページ番号の開始の値が異なることに注意せよ．解説中の

Algorithm 1 しゃくとり法の概略

1: **for** $S = 1, 2, \ldots, N$ **do**
2: Find $E_S \leftarrow \min_E(E - S + 1)$ where $\sum_{i=S}^{E} A_i \leq P$.
3: **end for**
4: Output $\min_{S=1,2,\ldots,N} E_S - S + 1$.

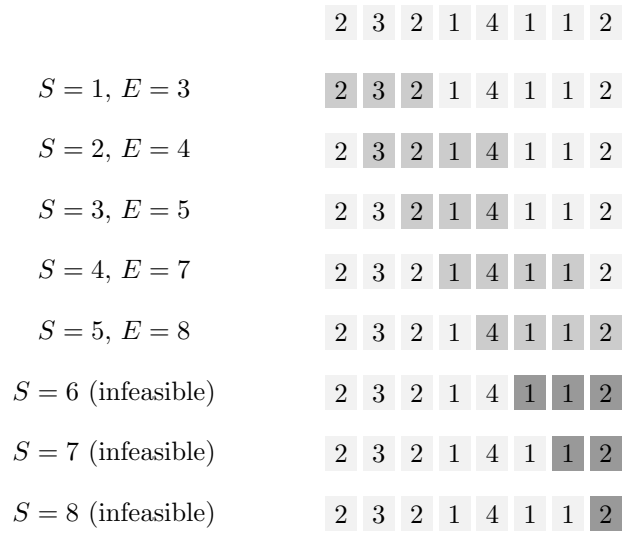

図 1　しゃくとり法の振る舞い（$N = 8, P = 7$ の例）．各行における色付きのマス目の数値はページごとの項目数を示す．

ページ番号の自然言語の習慣に慣習に従い 1 ページから開始しているが，ソースコード中では配列の扱いやすさのために 0 ページを開始番号としている．

解答例（しゃくとり法による $\mathrm{O}(n)$ の方法）

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void) {
5    int n, p;
6    int *a;
7    int i;
8    int start, end;
9    int sum, min_pages;
10
11   scanf("%d␣%d", &n, &p);
12   a = malloc(sizeof(int) * n);
13   for (i = 0; i < n; ++i) {
```

Algorithm 2 しゃくとり法

1: Set $E' \leftarrow 1$ (variable, end page number plus one).
2: Set $Q \leftarrow 0$ (variable, sum of the items in the pages $[S, E')$).
3: Set $M \leftarrow$ arbitrary value larger than N (variable, min. pages).
4: **for** $S = 1, 2, \ldots, N$ **do**
5: **while** $E' \leq N$ and $Q < P$ **do**
6: $Q \leftarrow Q + A_E$
7: $E' \leftarrow E' + 1$
8: **end while**
9: **if** $Q < P$ **then**
10: **break**
11: **end if**
12: $M \leftarrow \min(M, E' - S)$
13: $Q \leftarrow Q - A_S$
14: **end for**
15: **if** $M \leq$ N **then**
16: Output M.
17: **else**
18: Output -1 (no satisfactory solutions).
19: **end if**

```
14      scanf("%d", &a[i]);
15    }
16
17    start = 0;
18    end = 0;
19    sum = 0;
20    min_pages = n + 1;
21    while (1) {
22      while (end < n && sum < p) {
23        sum += a[end];
24        end++;
25      }
26      if (sum < p) {
27        break;
28      }
29      if (end - start < min_pages) {
30        min_pages = end - start;
31      }
```

```
32      sum -= a[start];
33      start++;
34    }
35
36    if (min_pages == n + 1) {
37      // No solution
38      printf("%d\n", -1);
39    } else {
40      printf("%d\n", min_pages);
41    }
42
43    free(a);
44    return 0;
45  }
```

正しい答えを出力するが制限時間超過になる例（累積和と $\mathrm{O}(N^2)$ 回の探索の組み合わせ）

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(void) {
5     int n, p;
6     int *psum;
7     int i;
8     int start, end, min_pages;
9
10    scanf("%d␣%d", &n, &p);
11    psum = malloc(sizeof(int) * (n + 1));
12    psum[0] = 0;
13    for (i = 0; i < n; ++i) {
14      int a;
15      scanf("%d", &a);
16      psum[i + 1] = psum[i] + a;
17    }
18
19    if (psum[n] < p) {
20      // No solution.
21      printf("-1\n");
22      return 0;
23    }
```

```
24
25    min_pages = n;
26    for (start = 0; start < n; ++start) {
27      for (end = start + 1; end <= n; ++end) {
28        if (psum[end] - psum[start] >= p && end - start < min_pages) {
29          min_pages = end - start;
30        }
31      }
32    }
33
34    printf("%d\n", min_pages);
35
36    free(psum);
37    return 0;
38  }
```

参考文献

[1] 秋葉拓哉，岩田陽一，北川宜稔，「プログラミングコンテストチャレンジブック: 問題解決のアルゴリズム活用力とコーディングテクニックを鍛える」，第 2 版，マイナビ出版，2012.