電気電子情報工学実験 II (b)
実践的・競技プログラミング 練習コンテスト（Contest 1-3）
解説

廣田悠輔
y-hirota@u-fukui.ac.jp

目次

1 問題 A：合同実験

解説

問題文のとおりに 2 人組を作り，その 2 人組の数を足し合わせれば回答可能である．しかしながら，**もっと簡単に回答することもできる**．出力すべき値は作られる 2 人組の総数であるので，学生数の内訳（M および N）に関係なく 2 人組の数は学生総数 $M + N$ を 2 で割った値を切り上げた整数となる．

競技プログラミングではこの問題のように回答に必要のない情報が問題文で説明されることがある．回答の際はよく考え，本当に必要な情報を使ってプログラムを作成するように努められたい．現実のプログラム設計においては設計のために必要最低限の情報だけが与えられるというようなことはなく，むしろ不要な情報が数多くある中から本質的に必要な部分を取り出して用いることが多いのである．

回答例

```
1 #include <stdio.h>
2
3 int main(void) {
4   int m, n;
5   scanf("%d", &m);
6   scanf("%d", &n);
7
```

```
8     printf("%d\n", (m + n + 1) / 2);
9
10    return 0;
11  }
```

2　問題 B：カードシャッフル

解説

問題文で説明されたシャッフルの手順をプログラムでシミュレートすれば回答できる．単一の配列を使って並び替えをシミュレートするのはプログラムが複雑になるので，カードの並びを記憶する主たる配列に加えて，作業用の配列を 1 つ余分に用いて並び替えを行うとより容易である．後掲の回答例では `cards` および `workspace` という 2 つの配列を用いている．カードの並び替えのシミュレーションは，例えば，次の手順で実現できる．

- 問題文中の図の赤囲みより上にある $p-1$ 枚の青囲みのカードの数値を，作業用配列の移動後の位置に対応する要素にコピーする（回答例 26–28 行目）．
- 問題文中の図の赤囲みのカードの数値を，作業用配列の移動後の位置に対応する要素にコピーする（回答例 29–31 行目）．
- 作業用配列の最初の $p+q-1$ 要素をカードの並びを記憶する主たる配列にコピーする（回答例 32–24 行目）

  ここまでの操作で 1 回分の並び替えが完了する．移動前に赤囲みより下にあったカードは位置が変わらないため，一切コピーが必要でないことに注意．
- 以上の操作を $i = 1, 2, \ldots, M$ と繰り返す．

全体の計算量は以下のように見積もられる．1 回の並び替えで移動するカード枚は最大で N 枚である．1 枚のカードにつきコピーは 2 回発生する．カードの並び替えは M 回行われる．したがってコピーの回数は最大 $2NM$ 回である．本問題は $1 \leq N \leq 100$，$1 \leq M \leq 100$ であるので最悪ケースでもコピー回数は 2×10^4 程度となり，制限時間に余裕をもって間に合うと予想できる．

インデクス（配列の番号）の計算がやや煩雑であるので，プログラミングの際はミスに十分注意されたい．

ノート

この問題は 2004 年度の ACM ICPC 国内インターネット予選の問題 A[1] を参考に作成された（ほぼ同じである）．

回答例

```
1   #include <stdio.h>
2
3   enum {
4     N_MAX = 100
5   };
```

```
6
7  int main(void)
8  {
9    int cards[N_MAX];
10   int workspace[N_MAX];
11   int n, m;
12   int i, j;
13
14   scanf("%d␣%d", &n, &m);
15
16   /* Initial state */
17   for (j = 0; j < n; ++j) {
18     cards[j] = j + 1;
19   }
20
21   /* Shuffling */
22   for (i = 0; i < m; ++i) {
23     int p, q;
24     scanf("%d␣%d", &p, &q);
25
26     for (j = 0; j < p - 1; ++j) {
27       workspace[q + j] = cards[j];
28     }
29     for (j = p - 1; j < p + q - 1; ++j) {
30       workspace[j - (p - 1)] = cards[j];
31     }
32     for (j = 0; j < p + q - 1; ++j) {
33       cards[j] = workspace[j];
34     }
35   }
36
37   printf("%d\n", cards[0]);
38
39   return 0;
40 }
```

3　問題 C：4 × 4 迷路

解説

一定移動回数以内に可能な経路を全て調べることが，この迷路の最短経路を求める基本的な方法となる．最短経路が存在すれば，そのときの移動回数は 15 ($\simeq 4 \times 4 - 1$) 回以下となる（最短経路ならば同じマスを 2 回通る必要がないのでマス数以上の移動回数にはならないので）．初回移動では最大四方向，二回目以降の移動では最大三方向に移動できる（直前に居たマスへの移動は明らかに最短経路とならないので除外しているため）．したがって，最大で 4×3^{14} ($\simeq 1.9 \times 10^7$) 通りの移動経路を調べれば良いことになる．実際には，4×4 マスの盤外への移動や一度移動したことのあるマスへの移動，移動不能マス # への移動を禁じるため，調べるべき移動経路数は 1.9×10^7 よりずっと少ない．

移動経路の探索をプログラムとして実現するには，図 1 のように移動経路をツリーとして表現し，そのノード（節）を順に探索すれば良い．探索順は幅優先の方が探索回数を削減できるが，深さ優先探索を行った場合でも制限時間には十分に間に合うのでどちらを用いても良い．後掲の回答例では深さ優先探索を再帰関数を使って実装している．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  enum {
6    BUFLEN = 0x100,
7    MAZE_X = 4,
8    MAZE_Y = 4,
9  };
10
11 /**
12  * x: Current x position.
13  * y: Current y position.
14  * from: Previous position. 0: left, 1: upper, 2: right, 3: lower.
15  */
16 int solve(char const maze[][MAZE_Y], int const x, int const y, int
       const from,
17           int const current_step, int const step_limit)
18 {
19   int i;
20   int next_x, next_y;
21   int min_step = INT_MAX;
22   int this_step;
```

```
23
24  if (current_step > step_limit) {
25    return INT_MAX;
26  }
27
28  if (maze[x][y] == 'G') {
29    return current_step;
30  }
31
32  for (i = 0; i < 4; ++i) {
33    if (i == from) {
34      continue; // Do not go back to the previous position.
35    }
36    switch (i) {
37    case 0:
38      next_x = x - 1;
39      next_y = y;
40      break;
41    case 1:
42      next_x = x;
43      next_y = y - 1;
44      break;
45    case 2:
46      next_x = x + 1;
47      next_y = y;
48      break;
49    case 3:
50      next_x = x;
51      next_y = y + 1;
52      break;
53    }
54    if (next_x < 0 || next_x >= MAZE_X || next_y < 0 || next_y >=
          MAZE_Y || maze[next_x][next_y] == '#') {
55      continue; // Infeasible move.
56    }
57    this_step = solve(maze, next_x, next_y, (i + 2) % 4, current_step +
          1, step_limit);
58    if (this_step < min_step) {
59      min_step = this_step;
60    }
```

```
61    }
62    return min_step;
63  }
64
65  int main(void)
66  {
67    char buf[BUFLEN];
68    char maze[MAZE_X][MAZE_Y];
69    int i, j;
70    int min_step;
71    int const step_limit = MAZE_X * MAZE_Y - 1;
72    int start_pos_x = -1;
73    int start_pos_y = -1;
74
75    for (i = 0; i < MAZE_X; ++i) {
76      fgets(buf, BUFLEN, stdin);
77      for (j = 0; j < MAZE_Y; ++j) {
78        maze[i][j] = buf[j];
79        if (buf[j] == 'S') {
80          start_pos_x = i;
81          start_pos_y = j;
82        }
83      }
84    }
85
86    min_step = solve(maze, start_pos_x, start_pos_y, -1, 0, step_limit);
87    if (min_step <= step_limit) {
88      printf("%d\n", min_step);
89    } else {
90      printf("-1\n"); // No route to goal.
91    }
92
93    return EXIT_SUCCESS;
94  }
```

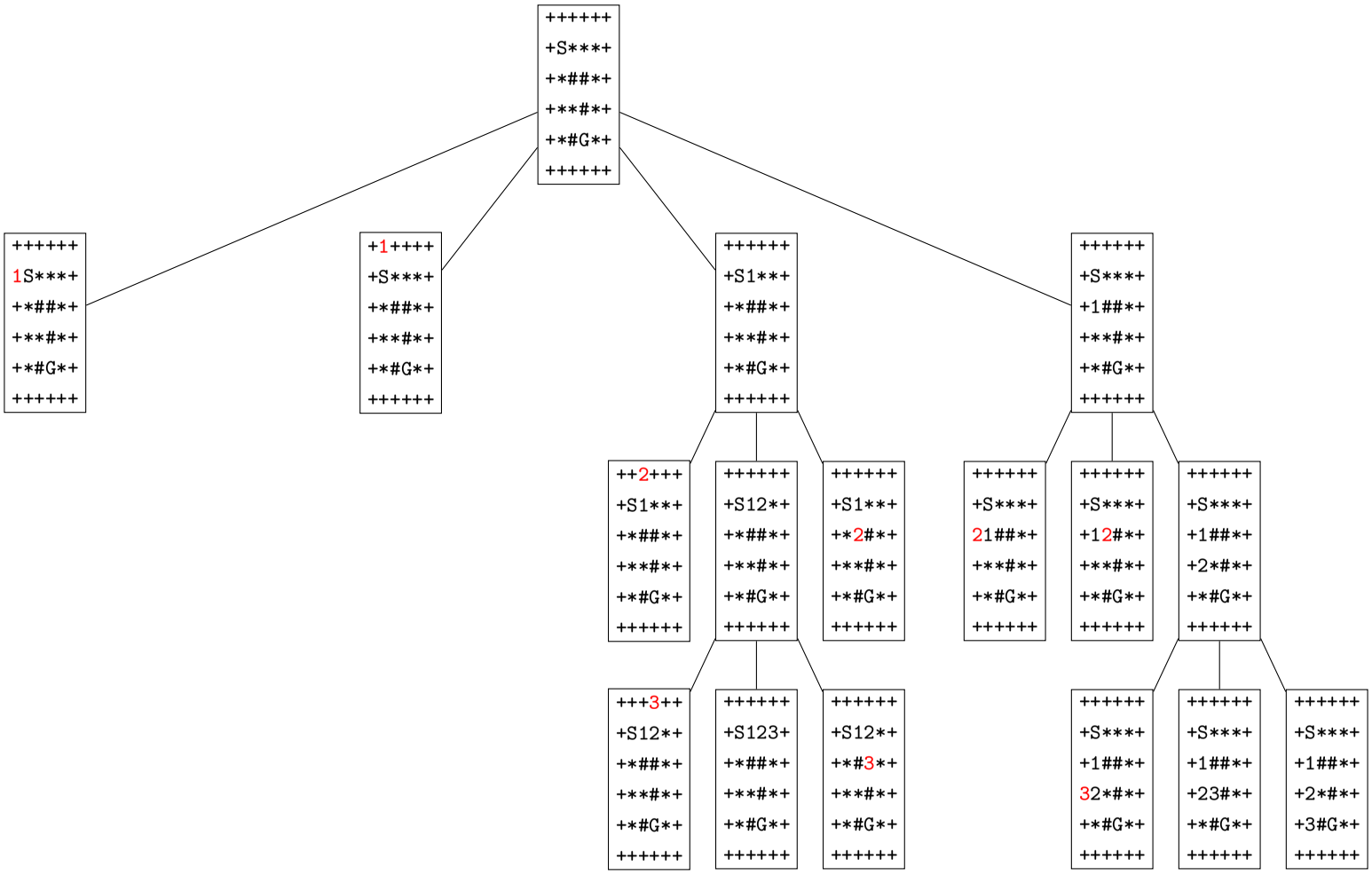

図 1　ツリーによる 3 回目までの移動経路の表現（問題文の説明で使用された迷路）．記号 + は盤外マスを意味する．赤文字は移動が禁止されたマスへの移動を表す．

4　問題 D：Speech is Silver, Silence is Golden?

解説

予算 b ドルの範囲内で金塊と銀塊を一個ずつ購入して，その購入金額を最大化する問題は次のように定式化できる．

次の値を求めよ．

$$\max_{1\le i\le m,\ 1\le j\le n} f(i,j)$$

$$\text{where}\quad f(i,j)=\begin{cases} G_iP_{\mathrm{G}}+S_jP_{\mathrm{S}} & (G_iP_{\mathrm{G}}+S_jP_{\mathrm{S}}\le b)\\ -1 & (\text{otherwise})\end{cases}$$

このような問題を解く素朴な方法は，$1\le i\le m$ および $1\le j\le n$ を満たす i,j の全組み合わせ mn 個を二重ループにより調べることである．しかしながら，本問題では最大で $m=n=10^6$ が与えられるため，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を超過すると予想される．

上述の素朴な方法より効率的な解法が存在する．説明を簡単にするために 2 つの仮定を置く．

- m 個の金塊および n 個の銀塊のそれぞれの価格が既に計算されている．金塊の価格を $P_{\mathrm{G},i}$ $(i=1,2,\ldots,m)$，銀塊の価格を $P_{\mathrm{S},i}$ $(i=1,2,\ldots,n)$ とおく．
- 金塊の価格を $P_{\mathrm{G},i}$ $(i=1,2,\ldots,m)$ および銀塊の価格を $P_{\mathrm{S},i}$ $(i=1,2,\ldots,n)$ がそれぞれ昇順に並んでいる（$P_{\mathrm{G},1}\le P_{\mathrm{G},2}\le\cdots\le P_{\mathrm{G},m}$ および $P_{\mathrm{S},1}\le P_{\mathrm{S},2}\le\cdots\le P_{\mathrm{S},n}$ が成り立つ）．

このとき，問題の解はアルゴリズム 1 に示す手順で解が求められる．アルゴリズム 1 では，選択する金塊を先に固定して，その後に予算内で購入可能な最高額の銀塊を調べている（アルゴリズム 1 3–15 行目）．ここで重要なポイントは，予算内で購入可能な最高額の銀塊を調べるときに毎回 m 個の銀塊をすべて調べるのではなく，明らかに予算内に収まらない銀塊や，明らかに最高額とならない銀塊を回避しているころにある．例えば，i' 番目の金塊を選択したときに j' 番目の銀塊が予算内で購入可能な最高額のものであったとする（すなわち $j'+1$ 番目以降の銀塊は予算上限を超過している）．すると，$i'+1$ 番目の金塊を選択したときも，$j'+1$ 番目以降の銀塊は予算内で購入不可能である（$i'+1$ 番目の金塊は i' 番目の金塊より高額であるため）．したがって，j 番目までの銀塊だけを調べれば良い．j 番目の銀塊から降順に銀塊を調べていき，j'' 番目の銀塊ではじめて $i'+1$ 番目の金塊との合計金額が予算内に収まった（$P_{\mathrm{S},i'+1}+P_{\mathrm{S},j''}\le b$）とする．すると，この j'' 番目の銀塊が，$i'+1$ 番目の金塊を選択したときに予算内で購入可能な最高額の銀塊となる．なぜならば，銀塊の価格は昇順ソートされていて $P_{\mathrm{S},j}\le P_{\mathrm{S},j''}$ $(j<j'')$ が成り立つためである．したがって，j' 番目の銀塊から降順に調べて $i'+1$ 番目の金塊との合計額が予算内に収まるものが一つ見つかれば，$i'+1$ 番目の金塊に対する銀塊の探索は打ち切って良い（アルゴリズム 1 13 行目）．このように探索することで解は $\mathrm{O}(\max(m,n))$ で求まる[*1]．

実際には，金塊の価格 $P_{\mathrm{G},i}$ $(i=1,2,\ldots,m)$ および銀塊の価格 $P_{\mathrm{S},i}$ $(i=1,2,\ldots,n)$ は計算されておらず昇順にもなっていない．このため，事前にこれらの価格を計算したうえで昇順にソートする必要がある．金塊および銀塊の価格計算の計算量は $\mathrm{O}(m+n)$ である．マージソートなどの効率的なソートアルゴリズムを使えば

[*1] アルゴリズム 1 が二重ループなので $\mathrm{O}(mn)$ であるように感じられるかもしれないが誤解である．変数 j は最大でも n 回しか変化しないので評価回数は $\mathrm{O}(\max(m,n))$．

Algorithm 1 効率的アルゴリズム（金塊価格および銀塊価格が昇順ソートされていると仮定）

$P_{\max} \leftarrow -1$
$j \leftarrow n$
for $i = 1, 2, \ldots, m$ **do**
  while $j \geq 1$ **do**
    $P \leftarrow P_{\mathrm{G},i} + P_{\mathrm{S},j}$
    if $P > b$ **then**
      $j \leftarrow j - 1$
      continue (over budget).
    end if
    if $P > P_{\max}$ **then**
      $P_{\max} \leftarrow P$ (update max. value).
    end if
    break.
  end while
end for
if $P_{\max} \geq 0$ **then**
  Output $P_{\max}$.
else
  Output -1 (no satisfactory solutions).
end if

金塊価格および銀塊価格のソートに必要な計算量は合計 $\mathrm{O}(m \log m + n \log n)$ となる．したがって，この問題の解を求めるのに必要な計算量は $\mathrm{O}(m \log m + n \log n)$ となり，素朴な方法と比べてはるかに効率的である．

ノート

- 本問題やその類題に対する効率的アルゴリズムが [2, 第 II 部 問題 B] に掲載されている．

回答例

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

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

int main(void) {
  int b;
  int m, n;
```

```
11  int p_g, p_s;
12  int *g_yen, *s_yen;
13  int i, j;
14
15  scanf("%d", &b);
16  scanf("%d␣%d", &m, &n);
17  scanf("%d␣%d", &p_g, &p_s);
18
19  g_yen = malloc(sizeof(int) * m);
20  s_yen = malloc(sizeof(int) * n);
21  if (g_yen == NULL || s_yen == NULL) abort();
22
23  for (i = 0; i < m; ++i) {
24    int g;
25    scanf("%d", &g);
26    g_yen[i] = g * p_g;
27  }
28  for (j = 0; j < n; ++j) {
29    int s;
30    scanf("%d", &s);
31    s_yen[j] = s * p_s;
32  }
33
34  // Ascending sort
35  qsort(g_yen, m, sizeof(int), smaller);
36  qsort(s_yen, n, sizeof(int), smaller);
37
38  {
39    int max_price = -1;
40    int jj = n - 1;
41    for (i = 0; i < m; ++i) {
42      for (; jj >= 0; --jj) {
43        int const this_price = g_yen[i] + s_yen[jj];
44        if (this_price > b) continue;
45        if (this_price > max_price) {
46          max_price = this_price;
47        }
48        break;
49      }
50    }
```

```
51      printf("%d\n", max_price);
52    }
53
54    free(s_yen);
55    free(g_yen);
56    return 0;
57  }
```

参考文献

[1] ACM International Collegiate Programming Contest Japan Domestic, Problem A “Hanafuda Shuffle”, https://icpc.iisf.or.jp/past-icpc/domestic2004/A.jp/A.html (accessed on 2021-05-24).

[2] 筧捷彦，「目指せ！プログラミング世界一 — 大学対抗プログラミングコンテスト ICPC への挑戦」，近代科学社，2009.