電気電子情報工学実験 II (b)
実践的・競技プログラミング 本番コンテスト（Contest 2-2）
解説

廣田悠輔
y-hirota@u-fukui.ac.jp

目次

1 問題 A：やさしい逆 FizzBuzz 問題

解説

問題文の説明のとおりに Fizz，Buzz，FizzBuzz と書かれた行を対応する数値に置き換えて出力するだけでよい．これを実現するプログラムの作成するには，文字列処理（空白を含む行全体の読み込み，文字列比較）についての理解が必要になる．

この問題では入力行に空白が含まれる可能性がある．scanf 関数は改行の他に空白やタブを区切り文字として扱うため，行に空白が含まれる場合には scanf("%s", str); としても行全体を読み込むことはできない．空白が含まれる可能性がある行をすべて読み込むには fgets を使うと良い[*1]．fgets 関数を使用する際の注意点の一つに入力ストリームの改行文字の扱いがある．scanf は書式に示される読み込みを行った後，最後に現れる区切り文字（改行文字等）を入力ストリームに残す．これに対して fgets は最初の改行文字まで読み込み，入力ストリームからはその改行文字が取り除かれる．このため，標準入力一行目の整数値 N を scanf で読み込み，以降の入力を fgets で読み込もうとすると，scanf の残した改行文字が入力ストリーム先頭に残った状態で fgets が実行されることになり，最初の fgets の結果が空行となる．この問題を解決するには

- 回答例のように最初の整数の読み込みを fgets と sscanf[*2]の組合せにより行う，
- 最初の scanf の直後に，空行を処理するための fgets を挿入する

[*1] gets 関数を使っても可能であるが，gets はセキュリティ上の問題があるため使うべきでない．なお，C11 以降の C 言語規格では gets は廃止されている．

[*2] scanf の誤りではない．

などの手段がある．

文字列の比較は単に strcmp 関数を利用すれば良い．fgets で読み込んだ文字列は末尾に改行文字を含むため，strcmp("Fizz\n", str) のように比較対象文字列にも改行文字を含めなければならない点に注意せよ．

コメント

ロジック部分の難度が低い問題であるにもかかわらず，文字列処理に手間取り正答できない受講生が多く見られた．そのような受講生は文字列処理を復習しておくこと．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_LEN (256 + 1)
6
7  int main(void)
8  {
9    int i, n;
10   char str[MAX_LEN];
11
12   fgets(str, MAX_LEN, stdin);
13   sscanf(str, "%d", &n);
14
15   for (i = 0; i < n; ++i) {
16     fgets(str, MAX_LEN, stdin);
17
18     if (strcmp("Fizz\n", str) == 0) {
19       printf("3\n");
20     } else if (strcmp("Buzz\n", str) == 0) {
21       printf("5\n");
22     } else if (strcmp("FizzBuzz\n", str) == 0) {
23       printf("15\n");
24     } else {
25       printf("%s", str);
26     }
27   }
28
29   return EXIT_SUCCESS;
30 }
```

2　問題 B：異物混入対応

解説

M 個の玩具セットすべてについて，異物が混入された可能性のある重量であるか否かを判定すれば良い．異物が混入された可能性のある玩具セットの重量の集合

$$\{W_i + W_j + W_C + W_B \mid 1 \leq i < j \leq N\}$$

の元の数は最大 $N(N-1)$ である．したがって，M 個の玩具セットすべてについて集合の元であるか否かをすべての i, j について比較することで判定した場合，比較の回数は最大で約 MN^2 となる．一回の比較は $O(1)$ の計算量で可能であり，また制約として $N \leq 100$，$M \leq 100$ が与えられていることから，上記の判定は制限時間に余裕を持って実行可能である．

解答例では 29–33 行目で予め異物が混入した場合の玩具セットの重量一覧を計算しているが，事前の重量計算をせずに 38 行目で比較をする際に都度計算しても良い．いずれの場合も計算量の次数は変わらず，実行時間制限に十分に間に合うはずである．

コメント

異物が混入された可能性のある玩具の個数を数えるとき，一つの玩具セットに対して重複してカウントアップしている誤答が複数みられた．38 行目の分岐の条件は複数の `i`，`j` に対して真となりえるため，この if 文内で毎回カウントアップすると誤答となる．回答例では `s_k_is_candidate` というフラグで異物が混入された可能性のある玩具か否かを管理し，それを用いて 43–45 行目で一度だけカウントアップが行われるようになっている．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum {
5    MAX_N = 100,
6    MAX_M = 100
7  };
8
9  int main(void)
10 {
11   int n, m;
12   int w_c, w_b;
13   int w[MAX_N];
14   int s[MAX_M];
15   int feasible_weight[MAX_N][MAX_N];
16   int i, j, k;
```

```
17    int n_candidate = 0;
18
19    scanf("%d␣%d", &n, &m);
20    scanf("%d", &w_c);
21    scanf("%d", &w_b);
22    for (i = 0; i < n; ++i) {
23      scanf("%d", &w[i]);
24    }
25    for (k = 0; k < m; ++k) {
26      scanf("%d", &s[k]);
27    }
28
29    for (i = 0; i < n; ++i) {
30      for (j = i + 1; j < n; ++j) {
31        feasible_weight[i][j] = w[i] + w[j] + w_c + w_b;
32      }
33    }
34    for (k = 0; k < m; ++k) {
35      int s_k_is_candidate = 0;
36      for (i = 0; i < n; ++i) {
37        for (j = i + 1; j < n; ++j) {
38          if (s[k] == feasible_weight[i][j]) {
39            s_k_is_candidate = 1;
40          }
41        }
42      }
43      if (s_k_is_candidate) {
44        n_candidate++;
45      }
46    }
47    printf("%d\n", n_candidate);
48
49    return EXIT_SUCCESS;
50  }
```

3　問題 C：ゲームイベントのクリア可能性

解説

練習コンテスト（Contest 1-3）の問題 C と同様，Q 分木を全探索することで回答可能である．ゲームイベントの残り日数は D 日であり，毎日 Q 個のクエストから 1 つを選んで実施できるので，ゲームイベント開催期間中のクエストの選択方法は Q^D 通り存在する．クエストの選択方法ごとに何日目にクリアできるかを調べ，クリア可能な選択方法が存在する場合はその最小日数が出力すべき値であり，クリア不可能な場合は `-1` が出力すべき値となる．

ある選択方法で何日目にクリア可能であるかは，クエストで得られるアイテム A, B, C の数の和を求め M_1, M_2, M_3 と比較することで判定できる．したがって，選択方法ごとの計算量は $\mathrm{O}(D)$ である．制約として $1 \leq D \leq 8$，$1 \leq Q \leq 6$ が与えられることから，選択方法の個数は最大でも $Q^D \leq 6^8 < 1.7 \times 10^6$ であり，全探索可能である．回答例では，クエストの選択方法の全探索を Q 分木の深さ優先探索により実現している．

全探索の実装方法にはまったくエレガントでない別解も考えられる．ゲームイベントの日数は $1 \leq D \leq 8$ のいずれかであることから，一重，二重，三重，...，八重ループの全探索関数をすべて作成し，入力 D の値に応じて全探索を行う関数を選択するといった方法である．このような方法はコード記述量が多く，また将来的な D の最大値の変更にも対応が容易ではないため，再帰を用いた深さ優先探索を使う方がはるかに良い．

回答例（深さ優先探索）

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  #define Q_MAX (6)
6
7  int get_n_days_required(int n_days_left, int n_item_a, int n_item_b,
       int n_item_c,
8                          int q, int n[][3], int m[])
9  {
10   int i;
11   int shortest;
12
13   if (n_item_a >= m[0] && n_item_b >= m[1] && n_item_c >= m[2]) {
14     return 0; /* Event cleared */
15   } else if (n_days_left == 0) {
16     return INT_MAX; /* Event time over */
17   }
18
19   shortest = INT_MAX;
20   for (i = 0; i < q; ++i) {
```

```
21      int event_i_result = get_n_days_required(n_days_left - 1, n_item_a
            + n[i][0], n_item_b + n[i][1], n_item_c + n[i][2], q, n, m);
22      if (shortest > event_i_result) {
23        shortest = event_i_result;
24      }
25    }
26
27    if (shortest == INT_MAX) {
28      return INT_MAX;
29    }
30    return shortest + 1;
31  }
32
33  int main(void)
34  {
35    int d, q;
36    int n[Q_MAX][3], m[3];
37    int i;
38    int result;
39
40    scanf("%d", &d);
41    scanf("%d", &q);
42    for (i = 0; i < q; ++i) {
43      scanf("%d␣%d␣%d", &n[i][0], &n[i][1], &n[i][2]);
44    }
45    scanf("%d␣%d␣%d", &m[0], &m[1], &m[2]);
46
47    result = get_n_days_required(d, 0, 0, 0, q, n, m);
48    if (result > d) {
49      printf("-1\n");
50    } else {
51      printf("%d\n", result);
52    }
53
54    return EXIT_SUCCESS;
55  }
```

4　問題 D：記憶装置の初期化

解説

この問題は [1][*3]を元に作成された.

最初に入力例 1 に対する操作ついて考察する．最小の書き換え操作回数ですべてのビットをゼロにするには，問題文に記載のとおり $K = 3$ として

- 1–3 ビット目を書き換える．1101011 → 0011011
- 3–5 ビット目を書き換える．0011011 → 0000111
- 5–7 ビット目を書き換える．0000111 → 0000000

の順に書きかえれば良い．ところで,

- 5–7 ビット目を書き換える．1101011 → 1101100
- 3–5 ビット目を書き換える．1101100 → 1110000
- 1–3 ビット目を書き換える．1110000 → 0000000

と書き換えの順番を変更しても，同じ回数ですべてのビットがゼロになる．このことから分かるとおり，書き換えを行う順番は，最小操作回数には影響しない．また，最小の操作回数ですべてのビットをゼロにするとき，同じ区間（連続する K ビット）を 2 回以上書き換えない．すなわち,

- ...
- 5–7 ビット目を書き換える.
- ...
- 5–7 ビット目を書き換える.
- ...

のような無駄な操作は行わない.

これまでの考察に元に，K を適当な値に固定したときの最小操作回数を求める方法を考える．書き換えの順番は最小操作回数には影響せず，また同じ区間を 2 回以上を書き換える必要がないことから,

- 1 ビット目から始まる K ビットの書き換え操作を行うか否か,
- 2 ビット目から始まる K ビットの書き換え操作を行うか否か,
- 3 ビット目から始まる K ビットの書き換え操作を行うか否か,
- ...
- $N - K + 1$ ビット目から始まる K ビットの書き換え操作を行うか否か

だけを判断すれば良い．1 ビット目を変更可能なのは 1 ビット目から始まる書き換え操作のみであるから，入力の 1 ビット目を見れば，そのビットから始まる区間を書き換える操作を行うか否かを決定できる．次に，2 ビット目を変更可能なのは，まだ行うか否かを判断していない操作の中では 2 ビット目から始まる書き換え操作のみであるから，2 ビット目の現在の値を見れば，そのビットから始まる区間を書き換える操作を行う

[*3] 和文抄訳および解説が [2, 3.2 節] に掲載されている.

か否かを決定できる．以下 3 ビット目以降も同様に順次に決定できる．$N-K+1$ ビット目から始まる K ビットの書き換え操作を行うか否かを判断し終えた後，最後の $N-K+2$ ビット目から N ビット目がゼロにできていれば，その K で記憶装置のビットをすべてゼロに書き換えが可能であるとわかる．この方法を $K=1,2,\ldots,N$ のすべての場合について使えば，回答すべき最小操作回数とそのときの K が得られる．

K を固定したときの書き換え可能性ならびに操作回数を求める際，N ビットのビット列の書き換えを実際に行うと，各 K について $\mathrm{O}(KN)$ の計算量が必要となる．この方法を $K=1,2,\ldots,N$ について用いると，全体の計算量は $\mathrm{O}(N^3)$ となる．この問題では最大で $N=4\times10^3$ であることから，制限時間内にプログラムが実行できない可能性がある．

計算量を軽減するためビット列の書き換えを実際には行わず，j ビット目から始まる K ビットを反転させる操作を行ったか否かを f_j $(j=1,2,\ldots,N-K+1)$ という変数（回答例では配列 `f`）で記憶することを考える．このとき，i ビット目から始まる K ビットの書き換え操作を行うか否かを判定する際の i ビット目の現在のビットの状態は，

$$b_i+\sum_{j=i-k+1}^{i-1}f_j$$

が奇数であるか偶数であるかにより判定する．上式の右辺の値は

$$\sum_{j=i-k+1}^{i-1}f_j=\left(\sum_{j=i-k}^{i-2}f_j\right)-f_{i-k}+f_{i-1}$$

であることから，右辺式を使えば i ごとに $\mathrm{O}(1)$ で更新できる．このようにしてビット列の実際の書き換えを行わずに現在の注目しているビットの状態を判定すれば，K を固定したときの書き換え可能性ならびに書き換え回数は $\mathrm{O}(N)$ の計算量で求まる．すると全体の計算量は $\mathrm{O}(N^2)$ となり，`TIMILIMIT`（制限時間超過）を回避できる．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  #define MAX_N (4000)
6
7  int get_n_operation(int const k, int const n, int const * const bits)
8  {
9    int n_operation = 0;
10   int f[n]; /* When bits (i, ..., i + k - 1) are flipped, set f[i] = 1,
         otherwise 0. */
11   int sum_flipped = 0; /* At iteration i, sum of f[i - k + 1], ..., f[i
         - 1]. */
12   int i;
13
```

```
14    for (i = 0; i < n - k + 1; ++i) {
15      /* Update the sum. */
16      if (i >= 1) {
17        sum_flipped += f[i - 1];
18      }
19      if (i >= k) {
20        sum_flipped -= f[i - k];
21      }
22
23      /* Check if i-th bit is 0 at this step. */
24      if ((bits[i] + sum_flipped) % 2 == 0) {
25        f[i] = 0;
26      } else {
27        /* The i-th bit is 1 at this step. Flip the bits (i, ..., i + k -
              1).  */
28        f[i] = 1;
29        n_operation++;
30      }
31    }
32
33    /* Check if the last k - 1 bits are zero. */
34    for (i = n - k + 1; i < n; ++i) {
35      /* Update the sum. */
36      if (i >= 1) {
37        sum_flipped += f[i - 1];
38      }
39      if (i >= k) {
40        sum_flipped -= f[i - k];
41      }
42
43      /* Check if i-th bit is 0 at this step. */
44      if ((bits[i] + sum_flipped) % 2 != 0) {
45        /* The i-th bit is 1. For this k, the bits cannot be set to zeros
              .  */
46        return INT_MAX;
47      }
48      f[i] = 0;
49    }
50
51    return n_operation;
```

```
52 }
53
54 int main()
55 {
56   int n;
57   char s[MAX_N + 1]; /* The main string and '\0' */
58   int bits[MAX_N];
59   int i, k;
60   int min_op_m = INT_MAX;
61   int min_op_k = -1;
62
63   fgets(s, MAX_N + 1, stdin);
64   sscanf(s, "%d", &n);
65   fgets(s, MAX_N + 1, stdin);
66   for (i = 0; i < n; ++i) {
67     bits[i] = s[i] - '0'; /* char to int conversion. */
68   }
69
70   for (k = 1; k <= n; ++k) {
71     int const n_operation = get_n_operation(k, n, bits);
72     if (n_operation < min_op_m) {
73       min_op_m = n_operation;
74       min_op_k = k;
75     }
76   }
77   printf("%d␣%d\n", min_op_k, min_op_m);
78
79   return EXIT_SUCCESS;
80 }
```

参考文献

[1] PKU JudgeOnline, “Face The Right Way”, http://poj.org/problem?id=3276 (accessed on 2023-05-24).

[2] 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.