連載

国内予選を突破せよ

湯淺 太一（京都大学大学院情報学研究科）
yuasa@kuis.kyoto-u.ac.jp

■はじめに

ACM/ICPC の日本国内予選が去る 7 月 2 日に開催された．世界大会やアジア地区予選と異なり，日本国内予選はインターネットを使って行われる．インターネットを使って問題をダウンロードし，解答を送付し，判定結果を受け取る．この点を除けば，世界大会やアジア地区予選と基本的に同じルールで競技を行う．今年度は 44 校 199 チームが参加し，うち 26 チームが国内予選を通過し，12 月に愛媛大学で開催されるアジア地区予選への出場権を獲得した．

Web で公開されている国内予選結果[☆1]によると，出題された 6 問中，3 問以上を解いたのは 30 チームであり，うち 16 チームが国内予選を通過している．このほかに，2 問しか解けなかったチームのうち 10 チームが通過している．国内予選は，同一の大学からの予選通過チーム数を制限しているので，3 問解けたからといって，必ずしも予選を通過できるわけではない．しかし，同一大学内の競争がなければ，3 問解けばまず予選を通過できると考えてよさそうである．そこで今回は，「国内予選を突破せよ」という題目で，今年度の国内予選で出題された 6 問のうち，比較的易しい問題を 3 つ取り上げることにする．来年度の参加を検討しているグループが，本稿を見て，国内予選突破のコツをつかんでいただけることを期待している．

■問題A Hanafuda Shuffle

n 枚 ($1 \le n \le 50$) のカードの山がある．問題の題名から，花札のカードだと考えてよいが，花札でなくてもかまわない．図 -1 に示すように，山の一番上から p 枚目 ($p > 0$) のカードから連続した c 枚 ($c > 0, p + c \le n + 1$) のカードを抜き取り，それをそのまま山の上に置く．この「カット操作」を r 回 ($1 \le r \le 50$) 繰り返す．各回のカット操作は，p と c のペアで与えられる．i 番目 ($1 \le i \le r$) のカット操作を，以下では (p_i, c_i) と表すことにする．カット操作をシミュレートして，最初のカードの山で，下から何番目にあったカードが最終的に山の一番上にくるかを答えるプログラムを書け，という問題である．

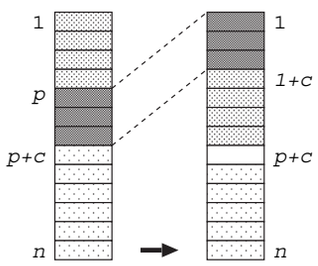


図-1　カット操作

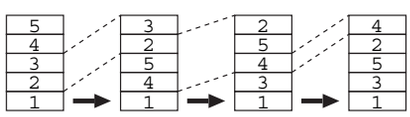


図-2　カット操作の例

たとえば，n=5，r=3，カット操作が順に (3, 2), (2, 3), (3, 1) のとき，カードの山は図 -2 のように変化し，最初に下から 4 番目にあったカード（図中の番号 4 のカード）が山の一番上にくるので，プログラムは 4 を出力する．

プログラムへの入力は，$n, r, p_1, c_1, \ldots, p_r, c_r$ の順に与えられ，それぞれの間は，空白や改行などの文字で区切られている．これが 1 セットであり，何セットかの後に n=r=0 となっていたら，入力の終わりである．C 言語であれば，main 関数を次のように定義すればよい．

☆1 http://www.ehime-u.ac.jp/ICPC/jp/

```
int main(void) {
    for (;;) {
        int n, r;
        scanf("%d%d", &n, &r);
        if (n == 0) break;
        printf("%d¥n", shuffle(n, r));
    }
}
```

ここで関数 shuffle は，カット操作を読み込みながら，それらをシミュレートし，最終的にカードの山の一番上にあるカードの番号を返す．

カードの山は当然配列で表現するが，C 言語では配列の添字が 0 から始まる．図 -1 を見ると，添字が 1 から始まるほうが都合が良さそうだ．そこで配列

```
int deck[n+1];
```

を用意し，deck[1] を山の一番上，deck[n] を一番下とする（deck[0] はまったく使わない）．そして次のように初期化しておく．

```
for (i = 1; i <= n; i++)
    deck[i] = n - i + 1;
```

つまり，deck[1] には n，deck[2] には $n-1$, ..., deck[n] には 1 を入れておく．こうしておけば，shuffle は，最終的に deck[1] に格納されている値を返せばよい．shuffle は，毎回 p と c の値を読み込んで，カット操作（のシミュレーション）を r 回繰り返す．この繰り返しは，

```
while (r-- > 0) {
    int p, c;
    scanf("%d%d", &p, &c);
    《ここにカット操作》
}
```

とすればよい．カット操作は，上から p 枚目以降の連続する c 枚（deck[p] 〜 deck[$p+c-1$]）を，いずれも $p-1$ だけ上に移動する．同時に，上から $p-1$ 枚目まで（deck[1] 〜 deck[$p-1$]）を，いずれも c だけ下に移動する．2 つのグループを移動するので，片方の移動によって，もう一方の情報が破壊されないように気をつけなければならない．ここでは，上から $p-1$ 枚目までのグループを

```
int temp[n+1];
```

に退避することにする．

```
for (i = 1; i < p; i++)
    temp[i] = deck[i];
```

そして，上から p 枚目以降の連続する c 枚を移動する．

```
for (i = p; i < p + c; i++)
    deck[i-p+1] = deck[i];
```

最後に，退避しておいた $p-1$ 枚を目的の場所に移動すればよい．

```
for (i = 1; i < p; i++)
    deck[i + c] = temp[i];
```

以上をまとめると，shuffle のコードは次のようになる．

```
int shuffle(int n, int r) {
    int i, deck[n+1], temp[n+1];
    for (i = 1; i <= n; i++)
        deck[i] = n - i + 1;
    while (r-- > 0) {
        int p, c;
        scanf("%d%d", &p, &c);
        for (i = 1; i < p; i++)
            temp[i] = deck[i];
        for (i = p; i < p + c; i++)
            deck[i-p+1] = deck[i];
        for (i = 1; i < p; i++)
            deck[i + c] = temp[i];
    }
    return deck[1];
}
```

■問題 B Red and Black

長方形の部屋の中に，赤または黒の正方形のタイルが敷き詰められている．最初に 1 人の人が部屋の黒いタイルの上に立っている．あるタイルからは，隣接する 4 つのタイルに移動することができるが，赤いタイルの上には移動できない．移動を繰り返すことによって到達できるタイルの枚数を求めるプログラムを書け，という問題である．

部屋の大きさを，タイルの枚数に換算して，横が w 枚分 ($w \leq 20$)，縦が h 枚分 ($h \leq 20$) とし，h 行 w 列の文字行列によってタイルの配置と人の初期位置を表す．'#' は赤いタイルを表し，'.' は黒いタイルを表す．ただし，最初に人がいる場所は '@' で表し，この場所のタイルは黒であるとする．たとえば，w=9, h=6 とし，タイルの配置が図 -3 のように与えられたとしよう．赤いタイルが 6 枚で，到達できない黒いタイルは 3 枚（右上角，右下角，左下角）なので，これらの枚数をタイル総数 9 × 6 = 54 から引くと，到達可能なタイルは 45 枚となる．

プログラムへの入力は，w と h の値に続けて，タイル情報として文字数 w の行が h 行続く．たとえば，図 -3 に対する入力は，

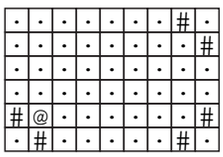


図-3　タイル

```
9 6
.......#.
........#
.........
.........
#@......#
.#.....#.
```

となる．これが 1 セットであり，何セットかの後に w=h=0 となっていたら，入力の終わりである．C 言語であれば，main 関数を次のように定義すればよい．

```
int main(void) {
    for (;;) {
        int w, h;
        scanf("%d%d¥n", &w, &h);
        if (w == 0) break;
        printf("%d¥n", tile(w, h));
    }
}
```

ここで関数 tile は，タイル情報を読み込んで到達可能なタイル数を計算する．

タイル情報を

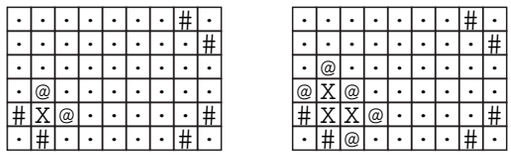
```
char tiles[20][21];
```

で表すことにする．サイズを 20 × 20 ではなく，20 × 21 としたのは，ライブラリ関数の gets を使って，次のようにタイル情報の読み込みたいからである．gets は，標準入力から 1 行読み込んで文字配列に格納する．ただし，最後の改行文字は空文字として格納するので，w=20 のときには 21 文字分のスペースが必要となる．

```
for (i = 0; i < h; i++)
    gets(tiles[i]);
```

このコードを実行すると，各要素 tiles[i] [j] (0 ≤ i < 20, 0 ≤ j < 20) には，'#'（赤いタイル），'.'（黒いタイル），'@'（人が立っている黒いタイル）のいずれかの文字が格納される．ただし，'@' は 1 カ所だけである．

プログラムは，'@' の位置から始めて，その前後左右に黒いタイルがあればそちらへ移動する，ということを繰り返して，何枚の黒いタイルに出会ったかを数えればよい．ただし，一度通った黒いタイルには目印をつけておいて，重複して数えないように注意する．これを実現する最も簡単なアルゴリズムは，次のようなものだろう．つまり，'@' の位置にいる人は，隣接する 4 つのタイルを順番に見ていく．もし黒いタイルであれば，そこに自分の分身を置いて，カウントを 1 増やす．これを隣接する 4 つのタイルすべてについて行い，自分はその場で引退する．以後は，分身が同様の動作を繰り返し，これ以上分身ができなくなったら終わりである．引退している人を 'X' で表すことにすると，図 -3 の入力に対してはまず図 -4 (a) のようになり，次に図 -4 (b) となる．最終的に，到達可能なタイルはすべて 'X' となる．このアルゴリズムをそのままコードにすれば，関数 tile が次のように定義できる．



(a)　(b)

図-4　分身に託して自分は引退する

```
int tile(int w, int h) {
  int count = 1, prev, i, j;
  char tiles[20][21];
  for (i = 0; i < h; i++)
    gets(tiles[i]);
  do {
    prev = count;
    for (i = 0; i < h; i++)
      for (j = 0; j < w; j++)
        if (tiles[i][j] == '@') {
          tiles[i][j] = 'X';
          if (i>0&&tiles[i-1][j]=='.'){
            tiles[i-1][j] = '@'; count++;
          }
          if (j>0&&tiles[i][j-1]=='.'){
            tiles[i][j-1] = '@'; count++;
          }
          if (i<h-1&&tiles[i+1][j]=='.'){
            tiles[i+1][j] = '@'; count++;
          }
          if (j<w-1&&tiles[i][j+1]=='.'){
            tiles[i][j+1] = '@'; count++;
          }
        }
  } while (count > prev);
  return count;
}
```

このtileの定義では，do文による繰り返しごとに部屋全体を走査して，分身を探し出している．分身の個数はタイルの総数と比較すると少ないので，次のように再帰的に定義するほうが効率は良さそうである．

```
int w, h;
char tiles[20][21];
int aux(int i, int j) {
    int count = 1;
    tiles[i][j] = 'X';
    if (i > 0 && tiles[i-1][j] == '.')
        count += aux(i-1,j);
    if (j > 0 && tiles[i][j-1] == '.')
        count += aux(i,j-1);
    if (i < h-1 && tiles[i+1][j] == '.')
        count += aux(i+1,j);
    if (j < w-1 && tiles[i][j+1] == '.')
        count += aux(i,j+1);
    return count;
}
int tile() {
    int i, j;
    for (i = 0; i < h; i++)
        gets(tiles[i]);
    for (i = 0; i < h; i++)
      for (j = 0; j < w; j++)
        if (tiles[i][j] == '@')
          return aux(i,j);
}
int main(void) {
    for (;;) {
        scanf("%d%d¥n", &w, &h);
        if (w == 0) break;
        printf("%d¥n", tile());
    }
}
```

ただし，入力によっては再帰がかなり深くなるので，スタックオーバーフローが生じないか気になる．日本国内予選は，プログラムを審判に送付して審判が動作確認するのではなく，選手が手元でプログラムを動作させ，与えられた膨大なテスト入力に対する出力結果を審判に送付する．したがって，スタックオーバーフローするかどうかを選手自身がチェックできる．もしオーバーフローすれば仕方がない．繰り返し版に切り替えることになる．

■問題C Unit Fraction Partition

分子が1で，分母が正整数である分数を，「単位分数」と呼ぶ（と問題に書いてある）．正の有理数p/qを有限個の単位分数の和として表現したものを，p/qの「単位分数への分割」あるいは単に「分割」と呼ぶことにする．たとえば，1/2 + 1/6は2/3の単位分数への分割である．与えられた4つの正整数p, q, a, nに対して，p/qの単位分数への分割で，次の2つの条件を満たすものの個数を求めるプログラムを書け，という問題である．ただし，足し算の順序の違いは無視する．たとえば，1/6 + 1/2と1/2 + 1/6は同じ分割と見なす．

- 分割はn個以下の単位分数の和である．
- 分割に含まれる単位分数の分母の積はa以下である．

たとえば$(p, q, a, n) = (2, 3, 120, 3)$のとき，条件を満たす分割は次の4つである．

1/3 + 1/3
1/2 + 1/6
1/4 + 1/4 + 1/6
1/3 + 1/6 + 1/6

プログラムへの入力は，p, q, a, nの1セットが1行に与えられ，それぞれの値は空白で区切られている．$p \le 800$，$q \le 800$，$a \le 12000$，$n \le 7$であり，$p=q=a=n=0$で入力の終わりを表す．

この問題は典型的な数え上げの問題である．次の条件を満たす正整数の列$x_1, x_2, \ldots, x_k$の個数を求めればよい．

(1) $k \le n$
(2) $0 < x_1 \le x_2 \le \ldots \le x_k$
(3) $x_1 \times x_2 \times \ldots \times x_k \le a$
(4) $1/x_1 + 1/x_2 + \ldots + 1/x_k = \mathrm{p/q}$

これらの条件を満たす数列を探す過程では，次のように数列を次々と伸ばしていく．

x_1
x_1, x_2
…
$x_1, x_2, \ldots, x_k$

このそれぞれの数列$x_1, x_2, \ldots, x_i$ $(1 \le i \le k)$は，次の条件を満たしているはずである．

(1) $i \le n$
(2) $0 < x_1 \le x_2 \le \ldots \le x_i$
(3) $x_1 \times x_2 \times \ldots \times x_i \le a$
(4) $1/x_1 + 1/x_2 + \ldots + 1/x_i \le p/q$

もし最後の不等式の左辺と右辺が等しければ，$1/x_1 + 1/x_2 + \ldots + 1/x_i$はすでに$p/q$の分割になっている．そうではなくて，もし$i = n$であれば，これ以上数列を伸ばしても$p/q$の分割は得られない．その他の場合は，

(1) $y \ge x_i$
(2) $x_1 \times x_2 \times \ldots \times x_i \times y \le a$
(3) $1/x_1 + 1/x_2 + \ldots + 1/x_i + 1/y \le p/q$

となるすべての y に対して，y を追加して伸ばした数列

$x_1, x_2, \ldots, x_i, y$

について，p/q の分割が得られるかどうか調べていけばよい．

プログラムは自然に再帰的になりそうだが，どのようなパラメータを受け渡せばよいだろうか．上の処理を素直に行うとすると，

(1) 現在の数列の長さ i
(2) 数列の最後の要素 x_i
(3) 数列の全要素の積 $x_1 \times x_2 \times \ldots \times x_i$
(4) 数列の全要素の逆数の和 $1/x_1 + 1/x_2 + \ldots + 1/x_i$

の 4 つが必要になる．まずこれらをパラメータとして，再帰的な関数を擬似的に書いてみよう．

```
int aux(int i, int last, int prod,
         ratio revsum) {
  int count = 0, y;
  if (revsum == rat(p,q))
    return 1;
  if (i >= n)
    return 0;
  for (y = last; prod*y <= a; y++)
    if (revsum+rat(1,y) <= rat(p,q))
      count += aux(i+1, y, prod*y,
                   revsum+rat(1,y));
  return count;
}
```

ここで，ratio は分数値の型で，rat(p, q) は p を分子，q を分母とする分数のつもりである．また，分数値に関する加算と比較演算も適宜使っている．入力されるパラメータ p, q, a, n は大域変数になっているとして，この擬似的な関数 aux を，

```
aux(0, 1, 1, rat(0,1));
```

と呼び出せば，分割の個数が求まることになる．

C 言語のプログラムとしてきちんと動かすために，revsum の分子を c，分母を d として書き直し，p, q, a, n の宣言と main の定義を与える．

```
int p, q, a, n;

int aux(int i, int last, int prod,
         int c, int d) {
    int count = 0, y;
    if (c*q == d*p)
        return 1;
    if (i >= n)
        return 0;
    for (y = last; prod*y <= a; y++)
        if (q*(y*c+d) <= p*y*d)
            count += aux(i+1, y, prod*y,
                         y*c+d, y*d);
    return count;
}
int main(void) {
  for (;;) {
    scanf("%d%d%d%d", &p, &q, &a, &n);
    if (p == 0) break;
    printf("%d¥n", aux(0,1,1,0,1));
  }
}
```

このプログラムはちゃんと動作するのだが，p, q, a, n が大域変数というのが気に入らない．かといって，aux へのパラメータとして再帰呼び出しごとに受け渡すのも無駄である．架空の ratio 型を使った最初の aux の定義で，

```
rat(p,q)-revsumの分子をP，分母をQ，
n-iをN，
rat(a,prod)を整数にまるめた値をA
```

として書き直すと，実は，非常にすっきりしたプログラムになるのである．

```
int aux(int last, int P, int Q, int A,
         int N) {
  int count = 0, y;
  if (P == 0)
    return 1;
  if (N <= 0)
    return 0;
  for (y=last; y <= A; y++)
    if (Q <= y * P)
      count
      += aux(y, P*y-Q, Q*y, A/y, N-1);
  return count;
}
int main() {
  for (;;) {
    int p, q, a, n;
    scanf("%d%d%d%d", &p, &q, &a, &n);
    if (p == 0) break;
    printf("%d¥n", aux(1, p, q, a, n));
  }
}
```

（平成16年11月13日受付）