審判長講評

審判長 石畑 清

最初に統計データを示すことにしよう。表 1 に，正解した問題数ごとのチーム数を示す。「内訳」は，正解した問題の組み合わせごとのチーム数をまとめたものである。表 2 に，問題ごとの提出チーム数，正解チーム数，最初の正解時間を示す。

表 1：正解した問題数

正解数	チーム数	正解した問題の内訳
8	3	ABCDEFGH: 3
7	4	ABCDEFG: 4
6	7	ABCDEF: 3，ABCDEG: 1，ABCDFG: 3
5	11	ABCDE: 3，ABCDF: 6，ABCDG: 2
4	31	ABCD: 28，ABCF: 2，ABCG: 1
3	94	ABC: 93，ABD: 1
2	71	AB: 62，AC: 9
1	137	A: 136，B: 1
0	33	

表 2：問題ごとの結果（全 391 チーム）

問題	提出	正解	最初の正解（秒）
A	378	357	225
B	251	213	812
C	167	158	829
D	68	54	1991
E	19	14	3115
F	22	21	3124
G	21	14	4688
H	4	3	7986

8 問用意すること，問題 A を極力易しくしてどのチームも 1 問は解けるようにすること，問題 D を解けるか否かが予選通過の可否を分ける鍵になるようにすること，などの基本的な方針は従来どおりである。これらについては，ほぼうまくいったと評価している。問題 A〜D の正解数は想定どおりだった。問題 E の正解数が F や G 以下だったのは意外で，実装が面倒に見える点が敬遠されたのではないかと考えている。

今年は，全問正解が 3 チームあった。しかも，最初の全問正解は，45 分以上残してのものだった。これは審判団にとって想定外のことである。上の正解数の分布から見ると，問題 F や G が少し易しすぎたということは言えるだろう。

以下では各問題について，解法を中心に簡単に解説する。

問題 A：太郎君の買物

この問題は，プログラミングの初歩を学んでいれば誰でも解けるレベルをねらって作題した。357チームが解いた結果は，その意図どおりだったと言ってよいだろう。

二つの品物の価格の和が最大のものを求めればよい。ただし，与えられた上限価格以下のものでなければならないという条件がついている。ごく単純な二重ループで，$n \times (n-1)/2$ とおりの和を片端から求めるのが最も簡単な解法である。計算量が $O(n^2)$ になるが，n が 1000 以下と規定されているので，遅いことを心配する必要はない。

別の解法としては，全体をソートしてから，二つの添字を配列の左右の端から内側に向かって並行して動かしていく方法がある。左右の和が m を超えていれば，右の添字を一つ左に動かす。逆に m 以下なら，左の添字を一つ右に動かす。この動作を左右の添字が一致するまで続ければよい。添字を動かす部分の計算量は $O(n)$ なので，ソートの計算量 $O(n \log n)$ が全体の計算量になる。もちろん，こちらの方が効率的だが，簡単な解法で十分に解ける問題をわざわざ難しく解く必要もあるまい。

問題 B：ほとんど同じプログラム

日本の国内予選では，2 回の解答提出に添付されたプログラムが完全に一致していなければ，正解と認められない。ところが，このルールの理解が不十分で，ファイル名を表す文字列一つだけを変更して提出するチームが毎年ある。今年も，いくつかあった。審判団にとって悩みの種であるこの事実をモチーフにした問題を作ってみた。

与えられた二つの文字列を二重引用符で区切って，対応する部分同士を比較していけばよい。不一致となる部分が一つだけで，それが奇数番目の二重引用符とその次の二重引用符の間であれば `CLOSE` と出力する。二つ以上の違いがあったり，文字列以外の場所での違いなら `DIFFERENT`，完全に一致していれば `IDENTICAL` と出力する。

プログラムで行うべきことは，このように明確であり，計算時間を気にする必要もないので，非常に簡単な問題だと言える。しかし，うっかり間違いをしやすい問題であることも確かなようだ。たとえば，比較しなければいけない部分を一つ抜かしてしまうといった間違いが多い。慎重なプログラミングが特に求められる問題である。

問題 C：池のある庭園

池の場所と大きさの候補を片端から数え上げて，問題文で規定されている池の条件を満たしているかどうかを調べればよい。条件を満たす池のうち，容量最大のものが答である。

池の位置は，x の範囲と y の範囲で決まるから，x の最小と最大，y の最小と最大の合計四つをそれぞれ動かす 4 重のループになる。さらに，この内側で，池を構成するそれぞれのセルの高さを調べるのに，x 方向と y 方向それぞれに動くループが必要なので，全体では 6 重のループということになる。d と w の大きい方を n と呼ぶことにすると，計算量は $O(n^6)$ になる。しかし，n の最大は 10 なので，n^6 でも 100 万にしかならず，計算時間の問題はない。

問題 D：弁当作り

この問題のポイントは，n と m の積が 500 以下という制約にある。二つの正整数の積が 500 以下なので，どちらか一方は 22 以下であることが言える。パラメーター k が 22 以下なら，2^k が 400 万程度なので，k 個のものの全部の組み合わせを試すことが可能だと判断できる。

想定解法は，$n \leq m$ の場合と，$m < n$ の場合に分けて，それぞれ違うプログラムで解くというものである。n が小さい場合は，レシピの全組み合わせを調べればよい。n 種類のレシピの組み合わせを再帰的に生成して，それぞれの組み合わせで材料が余るかどうかを調べるだけで解ける。

m が小さい場合は，典型的な動的計画法（DP）による解法になる。m 種類の材料の状態を m ビットのビット列で表す。材料がピッタリ（偶数個）使えるときは該当のビットを 0 とし，半端が生じるときは 1 とする。こうして得られる m ビットのビット列を 2 進整数と見なして，配列の添字にする。長さ 2^m の配列を用意して，それぞれの材料の状態ごとに，レシピの種類の最大数を入れる。レシピを一つずつ調べながら，そのレシピを追加したときに，配列の中にある材料の状態がどう変わるかを調べ，新しい状態のレシピの種類数を更新していく。

以上の想定解法に特に難しい点はないと思われるが，不必要に複雑な解き方だとする評価もあり得る。m が小さい場合と同様の動的計画法を m の大小にかかわらずに適用してしまえば解けるのだ。$n < m$ で m が大きい場合，動的計画法で使う配列の要素数は 2^m と多くなるが，実はそのうち 2^n 個の要素にしか値が入らない。レシピの組み合わせが 2^n しかないのだから，材料の状態も 2^n とおりを超えることはない。つまり，どんな場合でも $2^{\min(n,m)}$ 個の要素しかプログラムで操作することはない。配列でなく，連想配列のようなデータ構造（Map など）を用い，値の入っていない要素には触れないようにプログラミングすれば，1 種類のプログラムで想定解法と同じ計算量とすることができる。配列の添字，Map ならキーとなるのは，最大 500 ビットのビット列である。これを簡単に操作できないとプログラミングが面倒になるが，C 以外の言語ならビット集合または多倍長整数で対応できる。

問題 E：論理式圧縮機

与えられた例を見ると，`(1*a)` を `a` に簡約化し，`(-a*a)` を `0` に簡約化するようなルールをいくつも用意し，それらのルールを順次適用するようなプログラムを書きたくなるかも知れない。しかし，この方針の先に待っているのは泥沼でしかないと思われる。適用可能なルールがいくつあるか分かったものではなく，どこまで行っても完全という確信を得ることはできないだろう。

現実的な解法は，長さ 16 以下の論理式を片端から生成し，それぞれの論理関数を表す最短の式の長さを事前に求めておく方法である。生成する論理式の数が莫大なものになることが心配だが，実は大した数にはならない。排他的論理和や論理積の式には必ず括弧が付くので，文字数の消費が大きい。ちょっと考えると，16 文字以下の式の中に括弧付きの式は最大でも 3 個しか入れられないことが分かる。これなら大した数にはならないとの見通しを得ることが可能なはずである。

4 変数の論理関数は，4 変数の真偽の 16 とおりの組み合わせそれぞれについて，真偽のいずれかの値をとるものである。したがって，論理関数の種類は $2^{16} = 65536$ である。16 とおりのそれぞれの場合を特定のビットに対応させれば，4 変数論理関数は 16 ビットの整数値一つで一意に表すことができる。この整数値に対するビット単位の論理演算を使えば，16 とおりの場合すべてについての計算が一度に片付く。たとえば，16 進で `a` を `FF00`，`b` を `F0F0`，`c` を `CCCC`，`d` を `AAAA` と表すことが考えられる。この場合，`(a*b)` は `F000` に，`-d` は `5555` になる。

論理式の生成は，目標の長さを短い方から順に決めて，再帰的なプログラムで文法を忠実に追っていけばよい。BNF 中の `<E>` には，すでに求まっている論理式を片端から当てはめていく。生成した論理式がどの論理関数になるのかを調べ，それがその論理関数を表現する最も短い式であれば記録する。実際には，論理式そのものの記録は不要で，長さだけを記録すれば十分である。

それぞれの論理関数を表す論理式の最短の長さが求まっていれば，後は簡単である。与えられた論理式を構文解析して，それがどの論理関数を表すかを求め，すでに求まっている最短の長さをプリントすればよい。

問題Ｆ：リボンたたみ

この問題は，数学的な考察ができるかどうかが勝負である。解法の数学的な定式化さえできれば，プログラミングは非常に簡単なものになる。

定式化の方法はさまざまなものが可能だが，そのうちの一つを紹介する。まず，折る前のリボンの左端からの番号を左位置，折った後の上からの層の番号を上位置と呼ぶことにする。左位置，上位置とも，問題文では 1 から数えることにしているが，2 進数としての定式化に具合が悪いので，以下ではすべて 0 から数えることにする。

折る前のリボンの上側を表，下側を裏と呼ぶことにする。すると，何回か折りたたんだ後の各層は，上から順に裏，表，裏，表，... と交互になる。このことは簡単に確かめられる。1 回折った状態は，上が裏，下が表である。もう 1 回これを折ると，この全体を上下反転して，裏表も反対にしたものがこの上に乗ることになるから，やはり裏，表の 2 層が元の 2 層の上に乗る。何回か折った後の状態も同様である。

最後の 1 回の折りを考える。折る直前に上位置が偶数（0 ベースで）の層は，裏なので，左位置の小さい側が右，大きい側が左にある。この二つの左位置は，最下位ビットが違うだけなので，最下位ビットが 0 の数，つまり偶数が右，奇数が左にあることになる。これを左から右に折ると，左位置奇数の部分が上半分の上位置奇数の層に来る。また，左位置偶数の部分は，下半分の上位置偶数の層に留まる。つまり，上位置の偶奇と左位置の偶奇が一致すれば最後の折りは L，逆に，一致しなければ R であることが分かる。

最後の 1 回の折りが分かったら，その折りをする前の状態に戻す。左位置は，最後の回の折り目を全部消して数え直せばよいので，半分にすればよい。上位置は，最後の折りだけ戻した状態を考えれば簡単に計算できる。上半分なら元の上位置を p として $2^{n-1}-p-1$ が新しい上位置である。下半分なら $p-2^{n-1}$ となる。この手順を繰り返していけば，全部の回の折りの方向が求まる。

問題Ｇ：迷宮を一周

部屋の接続関係をグラフとして表現すると，4 点を通る単純閉路があるかどうかを判定する問題となる。この問題は，一般には非常に難しいのだが，グラフが平面的で，かつ 4 点がすべて外周上にあるという制約があるので，解くことができる。

具体的には，4 点の点連結度（途中の点を共有しないパスの本数）を調べれば解ける。すべての宝箱を手に入れるためには，出入り口と各宝箱との点連結度が 2 以上である必要がある。しかも，これは十分条件であることが示せる。点連結度が 2 以上かどうかの判定は，最大流や関節点列挙のアルゴリズムを用いて $O(r)$ 時間で行うことができる（部屋の数を r とした）。さらに，この問題の場合は，r が比較的小さいので，もっと遅いアルゴリズムでも十分である。単純に 4 点以外のすべての点について，その点を取り除いても（部屋に入れなくしても）4 点が連結であるかどうかを判定する $O(r^2)$ 時間の解法でも間に合う。

もう一つの解法として，できるだけ外側を通るパスを貪欲に選択していく方法もある。正解したチームの多くは，こちらの方法で解いたようだ。反時計回りの順に四つの角に行くことを考える。その際，なるべく右側の端に近い所を歩く方がよいのは明らかだろう。たとえば，南西の角から南東の角に行くときは，なるべく南の端に近い所を歩けば，次の南東から北東への移動，さらに北東から北西への移動の妨げとなりにくくなる。端に近い所を歩くための具体的なアルゴリズムとしては，迷路の攻略法として有名な右手法を使うのがよい。常に右手で壁に触れながら進めるだけ進む。目的の角に到達できれば成功，出発点の角に戻ってしまえば失敗である。右手法を使っている間は，同じ部屋を 2 回以上通ることも許容する（行き止まりから戻ることがあるから）。プログラミングは，現在の歩く方向を覚えていれば簡単で，計算量も $O(r)$ にしかならない。

問題 H：等積変形

1 回の操作で頂点を一つしか動かせない。三角形を重ねるには，三つの頂点すべてを動かす必要があるので，最低 3 回の操作が必要である。したがって，答は 3，4，`Many` の 3 種類しかない。

答が 3 になるのは，T_1 の三つの頂点を T_2 の頂点のどれかに重ねる操作を無駄なく行えた場合である。どの頂点を先にするか，ある頂点の行き先としてどの頂点をとるかで，$3! \times 3! = 36$ とおりの場合があるが，このすべてについて可能かどうかを調べ，どれか一つでも可能な手順があれば 3 と答えればよい。重ねる操作が可能なのは，動かす頂点と行き先の頂点を結ぶ直線が動かさない 2 頂点を結ぶ直線と平行な場合である。

答が 3 でないと分かったら，4 回の操作で可能かどうかを調べる。可能なら 4 と答え，可能でないなら `Many` と答えればよい。以下，4 回の操作で重ねることが可能かどうかを調べる方法について述べる。

4 回のうち 3 回は T_2 の頂点に重ねる操作になり，1 回だけ T_2 の上でない点への移動になる。この点を中継点と呼ぶことにする。T_1 の頂点から中継点に移動した直後に，中継点から T_2 の頂点に移動することはあり得ない。1 手の損にしかならないからである。このことから，合計 4 回の移動操作のうち，中継点に関する操作を行うのは，1 手目と 3 手目，1 手目と 4 手目，2 手目と 4 手目の 3 とおりだけ考えればよいことが分かる。

頂点の移動の順序が決まれば，中継点の座標が決まる。初めに T_1 の頂点から移動する際の直線と，後で T_2 の頂点に移動する際の直線の交点が中継点である。中継点の座標さえ分かれば，3 手の場合と同様の判定法で 4 手で可能かどうかが分かる。

上に述べたのは，細かく場合分けをして，それぞれの場合の操作が可能かどうかを判定する方法で，場合の数が少ないので高速であるが，場合分けを正確に行わなければならない点が欠点と言えば言える。ほかに，操作の途中に現れる三角形のグラフを作り，探索によって解く方法もあり，こちらなら場合分けは要らない。

中継点の座標は，整数とは限らない。実数を使うと誤差の問題が生じるので，分数で表すのがよい。この際，32 ビット整数ではオーバーフローする可能性があるので，注意が必要である。