審判長講評

審判長　石畑 清

今年の国内予選は，西日本の大雨の影響をもろに受けた。大学の休校や鉄道の運休が相次ぎ，大学で集まることが困難だという報告を多くのチームから受けた。このため，当日の朝になって，急遽今年限りの特別ルールを設けて，自宅その他での参加も認めることにした。かなりドタバタしたが，大きな混乱もなく国内予選の競技を進めることができたのは何よりだった。

433 チームが参加登録した。この数は過去最多である。去年の 391 チームより約 1 割増えている。大雨の影響で，事前に棄権を表明したチームが 12 あったが，この数を引いても過去最多であることは変わらない。

チームが解いた問題の数に関する統計データを示す。表 1 は，正解した問題数ごとのチーム数である。「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 は，問題ごとの提出チーム数，正解チーム数，最初の正解時間である。

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	1	ABCDEFGH: 1
7	1	ABCDEFG: 1
6	4	ABCDEF: 4
5	16	ABCDE: 14，ABCDF: 1，ABCDG: 1
4	41	ABCD: 41
3	55	ABC: 48，ABD: 1，ACD: 6
2	96	AB: 27，AC: 69
1	174	A: 173，B: 1
0	45	

表 2：問題ごとの結果（全 433 チーム）

問題	提出	正解	最初の正解（秒）
A	400	387	177
B	201	140	940
C	216	186	714
D	73	70	2182
E	23	20	4044
F	8	7	5682
G	4	3	6381
H	3	1	10129

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。

審判団にとっての誤算は，問題 B が予想より難しかったことである。問題 C より易しいはずと思っていたのだが，正解チーム数が問題 C を大きく下回った。問題 B のプログラムを作ったのに正解に

到達できなかったチームが 61 もあり，この数は過去の国内予選のどの問題よりも多い。問題の複雑さが多くのチームのレベルを越えていたのだろうと思われる。B の位置にはもっと易しい問題を置かなければならないという意味で，審判団にとっての反省材料である。

1 チームが全問正解した。終了 10 分ほど前のことで，この結果はほとんど理想的だったと言える。ICPC では，全問正解を避けることを目標の一つにしているが，正解ゼロの問題がないようにするという，もう一つの目標との同時達成は困難である。終了直前の全問正解はむしろ好ましいことと，日本の審判団は考えている。問題 E 以降の各問題の正解数の分布もほぼ理想的で，問題 B を除けば難易度調整に成功したと言ってよいだろう。

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：所得格差

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。過去数年の問題 A よりも易しくして，解けないチームはないだろうと思えるレベルにした。387 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。

平均値以下のものの個数を答えることが求められている。最初のループで全部の合計を求めてから，平均値を計算する。その後のもう一つのループで，それぞれの値が平均値以下かどうかを判定して，平均値以下のものの個数を数えればよい。計算量は，明らかに $O(n)$ である。

問題 A のプログラムを作ったのに，正解に到らなかったチームが 13 ある。その内容を分析してみたところ，そのうちの 11 チームは，二つのデータに対して続けて正解するというルールを理解していないか，または操作ミスをしているかのいずれかだった。つまり，問題が解けなかったわけではないのだ。操作ミスは，プログラムの出力を送らずに，入力データを送ったり，プログラムのソースを送ったり，コンパイル結果のバイナリーを送ったり，といろいろなパターンのものがあった。

本当にプログラムが間違っていたのは 2 チームだけだった。そのうちの 1 チームは，和の計算に `float` 型を使っていた。所得額の合計が最大で 10^9 つまり 2^{30} 程度になることがあり，仮数部が 23 ビットしかない `float` では精度が不足する。もう一つのチームは，$n = 2$ の時に何も答を出さないプログラムになっていた。問題文中にある $2 \leq n \leq 10000$ のような制約条件をチェックして，制約違反なら計算をしないようにしているチームが多いが，これは制約の意味を誤解している。問題文中の制約は，無条件で仮定してよいもので，自分でチェックする必要はない。今回のように，制約チェックを間違えて，結果として不正解になる場合もあるので，チェックしようとは考えないようにしてほしい。

問題 B：折り紙

2 次元平面の上で折り紙をする設定だが，幾何の問題ではない。水平または垂直の線に沿って折るだけだし，折る位置の座標もすべて整数なので，最初の長方形に含まれる大きさ 1×1 の単位正方形それぞれの動きを追跡するだけで解ける。最初の長方形の大きさが 32×32 以下で，折る回数が 20 回以下なので，最大でも $32 \times 32 \times 20 = 20480$ 回の操作をすれば十分である。

何回か折った後の紙の状態を表現するデータが必要だが，具体的にどんなデータで表現するかは選択の余地がある。素直なのは，途中にできる図形（必ず長方形）を構成する単位正方形それぞれの位置で紙が何層になっているかを記録する方法だろう。ほかに，最初の長方形に含まれる単位正方形それぞれの現在位置の座標を記録する方法もある。いずれの方法でも，折返しをした時のデータの更新操作にはある程度の注意が必要である。たとえば，折り線の位置が中央より左か右かで場合分けする必要があるかも知れない。

簡単な操作だけで解ける問題なので，多くのチームが解くだろうと審判団は考えていたが，結果は予想を裏切った。2 次元のデータを扱い，しかも縦横両方向の反転が生じる複雑さが，多くのチームにとっての障壁になったものと思われる。

問題 C：超高層ビル「みなとハルカス」

最下層の階数を a，連続するフロアの数を n とすると，$b = a + (a+1) + \cdots + (a+n-1) = n(n+2a-1)/2$，つまり $2b = n(n+2a-1)$ が成り立つ。したがって，n は $2b$ の約数でなければならない。しかも，$n + 2a - 1 > n$ なので，$n < \sqrt{2b}$ である。1 から $\sqrt{2b}$ までの範囲の n のそれぞれについて，$2b$ を割り切るかどうかを調べ，割り切る場合は $a = (2b/n - n + 1)/2$ が 1 以上の整数になるかどうかを調べればよい。条件を満たす最大の n が答である。

繰返しの回数は，最大でも $\sqrt{2b}$ 回で済む。$b < 10^9$ なので，$\sqrt{2b}$ は 4 万を少し超える程度であり，計算の量にはまったく問題がない。平方根までで打ち切ることを思いつかずに，1 から b までの範囲のループにした場合，問題を解くことはできるが，計算時間がかかりすぎるはずである。

問題 D：全チームによるプレーオフ

ごく単純に枝刈りありの再帰的全数探索を行えば解ける問題である。探索空間の大きさを簡単に見積もってみよう。9 チームあったとする。最初のチームは 8 試合のうち 4 試合ちょうどに勝たなければならない。場合の数は ${}_8\mathrm{C}_4 = 70$ になる。次のチームは，最初のチームとの対戦結果がすでに決まっているので，7 試合に 3 勝（または 4 勝）で，場合の数は ${}_7\mathrm{C}_3 = 35$ になる。以下，各チームの場合の数が最も多くなるように対戦結果が決まっていたとしても，場合の数はそれぞれ ${}_6\mathrm{C}_3 = 20$，${}_5\mathrm{C}_2 = 10$，${}_4\mathrm{C}_2 = 6$，${}_3\mathrm{C}_1 = 3$，${}_2\mathrm{C}_1 = 2$，${}_1\mathrm{C}_0 = 1$ を超えることはない。全部掛け算しても $70 \times 35 \times 20 \times 10 \times 6 \times 3 \times 2 \times 1 = 17640000$ にしかならない。実際には，1 試合以上の結果がすでに決まっているし，2 番目以降のチームの対戦結果の勝ちと負けの数が等しくなっているとは限らないので，場合の数はこれよりも少なくなる。単純な解法で問題ないと見当をつけることが可能なはずである。

プログラミングは，再帰的な関数を使えば簡単である。残っている試合の一つを選んで，勝ちの場合と負けの場合それぞれについて再帰的に探索を続ければよい。いずれかのチームの勝ち数か負け数が試合数の半分を超えた場合は，全チームプレーオフになる可能性がなくなるので，再帰呼出しをせずに探索を打ち切る。これが枝刈りに当たり，プログラムの性能を確保するためのポイントになる。

問題 E：浮動小数点数

浮動小数点数の仮想的な表現を設定して，その上での演算操作のシミュレーションを行う問題である。この問題の浮動小数点数表現は，現実のコンピュータのものと似ているが，1 以上の数しか表現しないこと，計算結果を正確に表現できない場合の丸めが常に切捨てであること，などいくつかの点で現実のものと違っている。

プログラムでは，指数部と仮数部を分けて，二つの整数として扱うものだろう。仮数部は，52 ビットあるので，当然 64 ビット整数を使う必要がある。浮動小数点数表現では省略されている小数点の上の 1，別の言い方をすると下から 53 ビット目にあるはずの 1 も，整数表現には入れておくべきである。

このように表現を決めておけば，加算操作の実現は難しくない。必要なら仮数部の桁を合わせるためのシフトをした後で，仮数部の整数同士を足し算する。53 ビットより上に繰り上がりが生じたら，指数部を 1 増やして，仮数部全体を 1 ビット右シフトすればよい。

この操作を 10^{18} 回繰り返すのでは時間がかかりすぎる。操作の回数を減らす工夫が必要である。この問題の場合，同じ数の足し算を繰り返すだけなので，指数部が変わらない間の足し算の繰返しを掛け算で置き換えることができる。指数部が変わるまでに何回の足し算ができるかを割り算で求めてから，その回数だけ足し算した結果を求めればよい。プログラム全体としては，指数部を 1 ずつ増やしながら，足し算の残りの回数が 0 になるまで，この操作を繰り返すことになる。指数部を増やした時，和 s と足す数 a の両方を 1 ビット右シフトする。$10^{18} > 2^{53}$ なので，右シフトを続けていくと，a が 0 になってしまうことがある点に注意が必要である。

問題 F：正三角形の柵

幾何の問題である。交点を求めるなどの難しい幾何の計算は不要なのだが，やはり幾何は幾何であって，微妙なコーナーケースに対する配慮が必要な問題になっているようだ。

想定解の計算量は $O(k^2 + n \log n)$ である。下辺の下に j 個の点（$0 \leq j \leq k$）を出すようにしながら，正三角形の下辺の位置を順次動かしていく。下辺の位置が決まったら，左辺を一番左まで（左辺の左に出る点がないように）動かし，右辺の右に $k - j$ 個の点が出るようにする（下辺の下からすでに出ている点は除く）。この状態から，左辺と右辺を順次右に動かしていく。右辺から一つの点を入れ，左辺から一つの点を出す操作の繰返しである。こうしていけば，除外する k 個の点の合理的な選び方をすべて尽くすことができ，計算量も $O(k^2)$ に抑えることができる。

事前準備として，n 個の点を三つの基準でソートする必要がある。y 座標によるソート，$y - \sqrt{3}x$ によるソート，$y + \sqrt{3}x$ によるソートの三つである。これらのソートによる計算量を加えると，プログラム全体の計算量は $O(k^2 + n \log n)$ になる。実は，n 個の点を完全にソートする必要はなく，最大または最小の $k + 1$ 個が分かれば十分である。部分ソートと呼ばれる方法で，計算量のうち $O(n \log n)$ の部分を少し小さくすることが可能だが，この問題の場合そこまで頑張る必要もなかろう。

右辺から一つの点を入れ，左辺から一つの点を出す操作は，微妙なコーナーケースに対する配慮が必要である。たとえば，右辺から入れようとする点がすでに左辺の外に出ていることがある。y 座標が非常に大きい場合にこのようなことが起こり得る。正三角形の中に残す点の数が少ない場合には，もっと微妙なケースもある。あらゆるケースを注意深く考えないといけない。座標が全部整数なので，左辺や右辺に二つ以上の点が乗ることはなく，この点が救いになっていると言えるだろうか。

問題 G：数式探し

1 以上の値に対する足し算と掛け算だけなので，部分文字列の範囲を広げるほど式の値は大きくなる（例外あり，後述）。このことを利用して，いわゆる尺取虫戦略を採るのが想定解法である。部分文字列の左端と右端を二つの変数で指すようにする。左端から右端までの式の値が目標値より小さければ右端を先に進め，大きければ左端を進める。この方式を使えば，目標値に等しい部分文字列すべてを $O(m)$ の計算量で検出することができる（文字列 s の長さを m とした）。

尺取虫戦略は対象のデータが 1 次元的なものでないと使えないが，数式は 1 次元の存在だろうか。括弧が存在するので，一般には木で表現される立体的な構造になっているはずである。しかし，この問題の場合，部分文字列が正しい数式になっていなければならないという制約がある。対となる括弧の外で始まって中で終わったり，中で始まって外で終わったりする文字列が正しい数式になることはあり得ない。括弧対の内側をそっくり部分文字列の中に入れるか，一切入れないかのどちらか

しかない。したがって，左括弧を見つけたら，対となる右括弧までの範囲を独立に再帰的に処理して，括弧の中の式の値を表す数値データに置き換えてしまえばよい。こうすれば，それぞれのレベルの式の処理は 1 次元的なものになる。

この問題のもう一つのポイントは，`1*1*1` のようなケースの扱いである。文字列の範囲を広げるほど式の値が大きくなると述べたが，これには例外がある。1 の掛け算では値が増えないので，尺取虫戦略がうまく適用できない。特別扱いで切り抜けるしかないだろう。1 の掛け算が連続している場所は，1 の個数を数えるだけにして，尺取虫の操作では一つの単位として扱う。目標値に等しいものが見つかったら，左端の 1 の個数と右端の 1 の個数に基づいて答となる部分文字列の個数を計算すればよい。

括弧を除いた後の数式は，数値が並んでいて，間に `*` または `+` があるだけの形をしている。部分文字列の左端または右端を `+` を越えて動かす時の式の値の増減は簡単に計算できる。`*` を越えて動かす場合は，それほど簡単ではない。`*` が連続する部分の積の値を保持するデータ構造を用意しておく必要がある。

問題 H：優秀なプログラマになるには

あるスキルとその前提スキルを辺で結ぶと，スキルの木構造ができる（前提スキルの方が親になる）。想定解法は，この木の上で pre-order の深さ優先探索（以下 DFS と呼ぶ）をしながら，動的計画法（以下 DP と呼ぶ）を適用するというものである。ちょっと珍しい方法と言えよう。

各スキルについて，子供のスキル（自分を前提とするスキル）を前提レベルの大きいものから順にソートする。このリストを後続スキルリストと呼ぶことにする。さらに，DFS をした時に，自分の子孫の次に訪問するノードも後続スキルリストの最後に加える。別の言い方をすると，弟がいれば弟，それがいなければ父の弟，それもいなければ父の父の弟，... を加えることになる。たとえば，サンプルの 3 番目は，根が 1 番，1 番の子供が 2 番と 3 番，2 番の子供が 4 番と 5 番になっている。子供の順序は，ソートした結果もこのままで変わらない。この場合の後続スキルリストは，1 番が $(2, 3, 6)$，2 番が $(4, 5, 3)$，3 番が (6)，4 番が (5)，5 番が (3) となる。6 番は元の木にないノードだが，DFS の終わりを表すための仮想ノードとして加えてある。

DP の表として $(n+1) \times (k+1)$ の大きさの 2 次元配列を用意する。配列の左の添字はノードの番号とする。右の添字は支払った授業料の総額である。配列の要素には，ちょうどその金額を払って，そのノードに到達した（そのスキルの授業を受けられる状態になった）時の総合力評価値の最大値を入れる。そのスキルの授業にはまだ支払っていないことに注意。表の初期値は，1 番ノードの 0 円の状態だけ 0 とし，ほかはすべて $-\infty$ とする。

DFS をしながら，各ノードで後続スキルリストに含まれる各ノードの DP 表の値を更新していく。自分のノードのスキルレベルをいくつにすればよいかを調べて，それに重要度を掛けた値を加えることになる。自分のノードを p，後続のノードを q とすると，スキルレベルは l_q 以上 h_p 以下でなければならない。ただし，最後に後続スキルリストに加えた弟ノードについては，l_q を 0 に置き換える。前提レベルの降順にソートしてあるので，弟ノードに行く時にスキルレベルの下限を考える必要はない。p から q への DP 表の更新は，p における $k+1$ 個の要素を順次見ながら，q における $k+1$ 個の要素を順次更新していく形になる。deque（両端キュー）を使って $O(k)$ の計算量で実現できる。

答は $n+1$ 番ノード（終わりを示す仮想ノード）の DP 表に残る。全スキルの最大レベルの和が予算より小さい場合を除けば，必ず金額 k の位置である。最終的な計算量は，DFS に $O(n)$ かかり，各ノードで $O(k)$ が必要なので，$O(nk)$ になる。