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。正解数ごとのチーム数を表 1 に示し，各問題の解答状況を表 2 に示す。

表 1：正解数ごとのチーム数

正解数	11	10	9	8	7	6	5	4	3	2	1	0
チーム数	1	0	2	1	7	3	4	3	9	11	1	0
チーム数累計	1	1	3	4	11	14	18	21	30	41	42	42

表 2：問題ごとの結果（全 42 チーム）

問題	チーム数		誤答 submit 数				
	正解	不正解	合計	WA	TL	RE	CE
A	42	0	19	12	4	1	2
B	41	1	35	33	2	-	-
C	28	12	118	80	25	12	1
D	21	4	40	37	2	1	-
E	16	1	16	2	10	2	2
F	15	5	36	30	3	3	-
G	14	1	15	11	1	3	-
H	2	0	0	-	-	-	-
I	2	4	11	8	3	-	-
J	3	2	8	6	-	2	-
K	2	3	6	6	-	-	-

不正解チーム数は，submit したが正解に到らなかったチームの数である。
WA : wrong-answer，TL : timelimit，RE : run-error，CE : compiler-error

ほとんどのチームが解いた A と B，14〜28 チームが解いた C〜G，2〜3 チームしか解けなかった H〜K と，難易度が三つにはっきり分かれる結果になった。正解数で見ても，7 問と 2 問にそれぞれ山がある。C〜G の難易度にもう少し差をつけて，正解数がばらつくようにした方がよいのだろうが，審判団にとっても難易度の精密な調整は難しい。全体として見れば，ほぼ適切な難易度設定だったと考えている。

今年の特徴は，比較的易しい問題の誤答が多かったことで，問題 C や D に顕著である。これらの問題をもう少し易しくすべきだったとする意見もあるだろう。しかし，データを分析すると，かなりの回数の誤答を経て正解したチームが相当数あったことが分かる。下位チームには厳しかったかも知れないが，挑戦に値する問題だったと考えれば，必ずしも悪くはなかったと考えている。

今年も昨年に引き続いて 11 問を出題した。参加チームの実力が上がってくる一方，易しい問題も残す必要があるため，10 問では支えきれなくなった感がある。それでも今年は全問正解チームが出てしまったが，2 位チームとの大きな差を考えれば，やむを得ないことと考えている。

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：Decimal Sequences

いろいろな解き方が考えられるが，0，1，...の順に数の表現が与えられた数字列の中に含まれているかどうか調べていって，含まれていない最初の数を答とするのが簡明だろう。表現が含まれているかどうかは，文字列のマッチングと同様の方法で判定できる。この解法なら，数字列の長さを n として $O(n^2)$ の計算量で解ける。

a 番目から b 番目までの数字で表される数をすべて求める $O(n^3)$ と思われるプログラムが見られたが，これは無駄である。数の表現の開始位置が 1000 とおりしかないのだから，4 桁の数は最大 997 個しかなく，答が 1997 を越えることはない（本当の最大値はおそらくもっと小さい）。

かなり苦しんだチームも見られたが，最終的には全チームが正解した。

問題 B：Squeeze the Cylinders

幾何の問題であるが，幾何特有の難しさはなく，簡単な数値計算をするだけの易しい問題である。二つの円がぶつからないようにするために中心の x 座標をどれだけ離さなければいけないかを求めることがポイントになるが，ピタゴラスの公式を利用すれば簡単である。

中心の位置を決める際に隣の円だけを見ればよいわけではない点が罠と言えば罠になっている。円の半径が大小さまざまなので，隣の小さな円にぶつかるより早く，いくつか先の大きな円にぶつかることがある。つまり，ある円の中心位置を決める際は，その円より左にあるすべての円との位置関係を確認する必要がある。計算量は，円の数を n として $O(n^2)$ である。同じ理由で，全体の幅を求める際に 1 番と n 番の円だけを見ればよいわけではない点にも注意が必要である。

例年の問題 B に比べて，かなり誤答が多かった。全チーム正解を想定していたが，41 チーム正解にとどまった。

問題 C：Sibling Rivalry

グラフの問題であるが，難しいグラフアルゴリズムの知識は必要としない。きちんと考えれば解法の見つかる問題であり，下位チームでも解けると考えていたが，正解数は 28 と想定を下回った。誤答の submit も，過去に例を見ないほど多かった。min-max 法の考え方に基づいてゲームの最適戦略を求めることが必要であり，そのあたりの考え方に慣れていないチームが多かったのかも知れない。審判団にとっては，やや誤算であった。なお，1 回の submit で正解したチーム数は 10 で，誤答を繰り返してから正解したチームの方が多かった。

初めに，各頂点から a 歩，b 歩，c 歩で到達できる頂点の集合を求める必要がある。a 歩の場合について言えば，この計算はグラフの隣接行列の a 乗を求めることに相当する。一般に，x 乗は $O(\log x)$ 回の掛け算で求めることが可能で，この方法を使えば，$O(n^3 \log a)$ の計算量になる。しかし，パラメーターを小さく設定してあるので，遷移を a 回繰り返す $O(n^3 a)$ の方法でも十分である。

これが求まったら，各頂点からゴールまでの所要ステップ数を繰り返し近似の要領で求めていけばよい。各頂点にゴールまでのステップ数を持たせる。初めは，ゴールが 0，それ以外の頂点が無限大である。各頂点から a 歩で行ける頂点のステップ数の最小値を求める。b 歩，c 歩についても同様に求め，三つの値の最大値に 1 を加えたものをその頂点の値とする。この操作を収束するまで繰り返していけばよい。最小値と最大値の両方が現れる点に注意。

問題 D：Wall Clocks

n 人の人の視界が壁のどこからどこまでであるかをまず求める。この計算は簡単である。こうして求まった n 個の視界を右端の位置をキーとしてソートしておく。次に，時計をどこに置けば各人の要求を満たしつつ最少個数にできるかを求めるわけだが，この部分の計算は greedy に行えばよい。k 番の人の視界の右端に時計を置くと決める。その後は，すでに置いてある時計が視界にかからない最初の人の右端に時計を置く操作を一周するまで繰り返して，時計の個数を求める。k を 1 から n まで順次変えながら時計の個数を求め，最少の値を答とすればよい。計算量は $O(n^2)$ である。

誰か一人の視界の右端に時計を置いてよいということが分かれば，この解法の正しさは理解できるだろう。視界の右端にない置き方が最適だったとしても，その時計を誰かの視界の右端までずらして，同じ個数にすることが可能なはずである。

壁がグルッと一周しているため，1 次元座標の取り方が難しく，座標の右端のすぐ次に左端があると考えて計算しなければならない。このあたりの細かな問題点をいかに簡明に整理して，混乱を避けるかがポイントとなる問題である。

問題 E：Bringing Order to Disorder

n 桁の数字列は 10^n 個ある。これを一つ一つ調べていたのでは計算時間がかかりすぎる。同じ数字の組み合わせで作れる列を一つにまとめて扱うことがポイントである。たとえば，1123，1231，3211 などは，1 が 2 個，2 が 1 個，3 が 1 個の同じ組み合わせから作られる列である。組み合わせの個数は，$n = 14$ の場合でも ${}_{10}\mathrm{H}_{14} = {}_{23}\mathrm{C}_{14} = 817190$ にしかならない。

組み合わせを片端から生成していく。この組み合わせと与えられた数字列の大小が最初の二つの規則で決まる場合，数字列の生成は不要である。組み合わせが与えられた数字列より大きければ何もしないし，小さければその組み合わせから作られる列の個数を求めるだけでよい。この計算は，階乗の割り算をするだけである。

最初の二つの規則では大小を決められない組み合わせに対してだけ具体的な数字列を構成することになるが，この場合もすべての列を調べたりしてはいけない。列の先頭の桁から順に数字を決めていく。ある桁の数字で大小が決まるなら，それ以降の桁は調べずに，列の個数の計算だけをする。その桁で大小が決まらない場合に限って，次の桁を調べる。この方法なら，ある組み合わせを調べるのに必要な計算量は桁数に比例する程度で済む。

問題 F：Deadlock Detection

時刻に関する二分法が基本戦略である。答が含まれている可能性のある時刻の範囲を順次狭めていく。最初は，時刻 0〜t の範囲である。範囲の中央の時刻が deadlock unavoidable になっているかどうか調べ，unavoidable になっていればそれより前の時刻だけ，なっていなければそれより後の時刻だけを残す要領である。

ある時刻の状態が deadlock unavoidable かどうかは，単純かつ greedy な方法で判定できる。その時点の使用可能リソースだけで終了できるプロセスを順次終了させていき，終了できずに残るプロセスがあるかどうかを調べればよい。この計算の計算量は，単純な方法でも $O(p^2r)$ であり，工夫すれば $O(p(p+r))$ にできる。この計算量を $\log t$ 倍し，それに各時刻の状態を求めるのに必要な $O(t)$ を加えたものが全体の計算量になる。後者は，二分法で時刻の間を行き来するので厄介にも見えるが，計算量のオーダーは単純に 0〜t の各時刻の状態を順に求めるのと同じ $O(t)$ である。

問題 G：Do Geese See God?

典型的な動的計画法（以下 DP と略す）の問題である。有名な longest common subsequence の問題の裏返しに当たる shortest common supersequence（二つの文字列を部分列として含む最短の文字列，以下 SCS と略す）を求めればよく，解法も同じような DP になる。ただし，細かい工夫がいくつか必要である。DP による解法の設計はさまざまなものが可能だが，ここではそのうちの一つを紹介することにしよう。

まず，元の文字列とそれを前後逆転したものの SCS の長さを求める。これが求めたい回文（palindrome）の長さになる。与えられた文字列の長さを n として，$n \times n$ の行列を用意する。この行列を M と呼ぶことにしよう。元の文字列を S，逆転文字列を T とすると，$M_{i,j}$ には $S_{1\ldots i}$ と $T_{1\ldots j}$ の SCS の長さを入れる。行列の左上から順に n^2 個の要素値を順次決めるだけの簡単な計算である。

答の回文の長さが求まったら，その長さの回文の個数を数える。行列上の各点に，その地点を通って作れる回文が何とおりあるかを残すようにすることが目標である。この際，上で求めた行列全体を調べると，回文でないものまで勘定してしまうことになるので，左上三角行列だけを調べることにする。また，単純な方法では $S = \texttt{abaa}$ と $T = \texttt{aaba}$ の SCS の個数を数える際に，S の最初の a にマッチさせる相手として T の最初の a を選ぶ方法と 2 番目の a を選ぶ方法の両方を数えてしまう。どちらでも，結果の文字列は同じなので，一方だけ数えるようにしなければならない。行列上の動きとして斜め方向が可能なら，縦横方向の動きを数えないようにすれば，これを実現できる。

回文の個数は，簡単にオーバーフローする。64 ビットの整数型を使っても到底おさまるような数ではない。オーバーフローの影響で間違った答になることを防ぐために，10^{18} より大きい一定数 a 以上の数をすべて a に置き換えるなどの方策が必要である。

最後に，辞書式順序で k 番目の文字列が何であるかを求める。これは，行列の各点を通る回文が何とおりであるかが分かっていれば，簡単な操作である。

問題 H：Rotating Cutter Bits

幾何の問題である。幾何の問題の例に漏れず，細かな場合分けをきちんと考えなければならない面倒な問題である。ただし，交点計算など普通の幾何問題によく現れる難しい計算はほとんど不要で，整数計算だけで答を求めることができる。

物体とカッターが同じ方向に同じ角速度で回転する。これは，物体を動かさずに，物体の周りにカッターを向きを変えずに回転させることと等価である。このことに気づくかどうかが問題のポイントになる。この場合のカッターの回転半径は L である。

これに気づくことができれば，後は物体の各格子点がカッターに触れるか否かを判定すればよい。ただし，物体の格子点を一つずつ調べたのでは計算時間が不足する。x 軸または y 軸に平行な格子線ごとに，カッターに触れずに残る格子点の集合を座標値のいくつかの区間の和として把握することが必要である。

カッターに触れる座標範囲は，カッター多角形のそれぞれの辺を一つずつ調べればよい。多角形の内点に触れるなら，必ず辺に触れるはずだからである。辺は水平か垂直に限られるので，辺を半径 L で回転させると，半径 L の円を二つ水平か垂直に並べて，対応する点同士を結んだような形になる。対応点を結ぶこれらの線分に含まれる座標値なら切られるし，そうでなければ切られずに残る。

問題 I：Routing a Marathon Race

問題 I という位置にあるが，実はこの問題はそれほど難しくない。単純な探索に枝刈りを一つ加えるだけで解ける。2 チームしか解けなかったのは意外である。

基本的には，出発点からのパスをすべて調べる。簡単な探索プログラミングであり，実現は難しくない。当然ながら，そのままだと調べるべきパスの数が多くなりすぎるので，効果的な枝刈りが必要である。この問題の場合，すでに通った地点 A の隣の地点 B に，いくつかの地点を経由した後で戻るようなパスは考えなくてよい。A から B に直行した方がそのようなパスよりもコストが小さくなるはずだからである。

この問題のミソは，この枝刈りを加えることによって調べるパスの数が十分小さくなるという事実を数学的に証明できることである。しかし，チームの立場に立って考えると，証明可能性は重要でないかも知れない。証明ができなかったとしても，枝刈りを加えれば解けるという見通しを得ることは可能だろうし，たとえ見通しがなかったとしても明快な解法を思いつかないのなら試みる価値がある。もっと多くのチームにこの解法を試してもらいたかった。

問題 J：Post Office Investigation

グラフアルゴリズムの応用問題であるが，アルゴリズム一つで片付くわけではなく，いくつものアルゴリズムを順次適用していかなければならない。難問と言えよう。使用するすべてのアルゴリズムの計算量に気を遣わなければならない。

初めに，入力のグラフを strongly connected component（以下 SCC と呼ぶ）の集まりに分解する。この操作は，深さ優先探索（以下 DFS と呼ぶ）で実現可能である。グラフを SCC の集まりと見たものは，DAG（directed acyclic graph）になっている。次に，元のグラフの頂点に対して支配木（dominator tree）と呼ぶデータ構造を作る。この木は，出発点から頂点 A に行く際に必ず頂点 B を通る時，B が A の祖先になるように構成したものである。コンパイラーのフロー解析で用いられるデータ構造で，一般のグラフの場合のアルゴリズムはいろいろ難しいのだが，DAG ならば，やはり DFS をするだけで求められる。

支配木が求まれば後は簡単である。質問に含まれる複数の頂点の共通の祖先の中で最も葉に近いものを求める。これは LCA（lowest common ancestor）と呼ばれる問題で，二つの頂点の LCA を $O(\log n)$ の計算量で求める方法が存在する。各頂点から根までの経路を二分法に似た動きでたどればよい。最後に，こうして求まった LCA から根までの経路の中にある最小コストの頂点を求める。これは，支配木を構成した時点で各頂点に対する最小コストを求めておいて，表に記憶しておけば簡単である。

問題 K：Min-Max Distance Game

今回の問題セットの中の最難問と考えていた。問題を数学的にきちんと分析しなければならない。分析さえできればプログラムは難しくない。

まず，元のゲームを勝ち負けのあるゲームに置き換えて考える。目標値 t を設定して，最後に残った二つの石の距離が t 以上なら Alice の勝ち，t 未満なら Bob の勝ちとする。このゲームが Alice 必勝か Bob 必勝かを判定できれば，t に関する二分法で元のゲームの結果を決めることができる。

n 個の石を頂点とするグラフを考える。石と石の距離が t 未満の時に限って，二つの頂点を辺で結ぶことにする。勝敗判定は，このグラフの上で定式化される。

最終着手者が Alice の場合と Bob の場合に分けて考える。Alice（石の距離を最大化する）が最終着手者の場合は，グラフの中の最大の clique（部分完全グラフ）の頂点数が $\lceil n/2 \rceil + 1$ 以上なら Bob の勝ち，そうでなければ Alice の勝ちである。Bob が勝ちのケースは簡単に分かるが，逆のケースについても証明できる点がポイントである。

Bob（石の距離を最小化する）が最終着手者の場合は，グラフの最小の頂点被覆（グラフの辺すべてについて，少なくとも一方の頂点を含む頂点集合）の頂点数が $\lfloor n/2 \rfloor - 1$ 以下なら Alice の勝ち，そうでなければ Bob の勝ちである。こちらも，Alice が勝ちのケースの証明は簡単だが，逆も証明できる。

最大 clique も，最小頂点被覆も，一般のグラフの場合は NP 困難問題である。しかし，この問題の場合のグラフは 1 次元直線上の距離に基づくものなので，そんなに難しく考える必要はない。左から順に調べていく DP で簡単に求めることができる。