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，各問題の正答チーム数，誤答チーム数，提出数を表 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	2	0	2	1	5	6	4	16	11	9	4	0
チーム数累計	2	2	4	5	10	16	20	36	47	56	60	60

表 2：問題ごとの正答チーム数，誤答チーム数，提出数

問題	A	B	C	D	E	F	G	H	I	J	K
正答数	60	48	53	19	9	3	36	2	4	5	19
失敗数	0	12	4	6	6	3	6	5	2	3	9
提出数	89	208	90	42	39	31	64	19	6	17	59

失敗数：解答を提出したが正解にまで到らなかったチームの数
提出数：正誤の如何にかかわらず，提出されたプログラムの総数

ほとんどのチームが C++をメインの言語として使用している。C++を使っていないのは 3 チームだけで，そのうち 2 チームが Java，1 チームが Python を使っている。C++と他の言語を併用しているチームも七つあった。

審判団が想定していた各問題の難しさは次のとおりである。解法設計とコーディングの二つの側面それぞれについて，最も易しいと想定した問題から順に並べてある。

解法設計	A	B	C	G	D	K	E	F	J	H	I
コーディング	C	A	B	G	K	D	E	H	I	J	F

審判団は，かなり難しい問題セットだと考えていた。たとえば，全問正解するチームはないだろうと予想していた。ところが，この予想は大きくはずれて，トップの 2 チームが全問正解した。しかも，この 2 チームは 2 時間近く残して 10 問を正解していて，最後の 1 問（問題 F）で何とか競技を成立させていたような状況だった。

この結果を審判団として反省しなければならないが，具体的にどうすればよいかは難しい。ここ数年，トップグループとその下のグループの間に大きな差が生ずることが多い。今回も，3 位以下のチームは最大 9 問しか解けていない。これらのチームにとっては十分に難しい問題セットだったと言えるだろう。実力差が大きいので，適切な難易度調整はかなり難しい作業である。

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：Digits Are Not Just Characters

問題仕様に従って素直に二つの文字列を比較すればよい。文字列を数字が連続する部分とそれ以外の部分に分ける。数字が連続する部分は，その数字列が表す整数値を計算して，その値を比較対

象にする。それ以外の部分は，1 文字ずつ文字コードを比較すればよい。全体として，辞書式の比較となるようにする必要があるが，その実現法は難しくないだろう。

全チームが解ける一番簡単な問題を意図していた。問題仕様が理解できれば，解けないはずのない問題である。実際，全 60 チームがこの問題に正解した。

問題 B：Arithmetic Progressions

等差数列を具体的に構成していけば解ける簡単な問題だが，制限時間内に解くためには何らかの工夫が必要になる。愚直な方法だと計算量が $O(n^3)$ になる恐れがあり，これでは遅すぎる。

想定解は次のとおり。最初に，入力された値をソートする。その上で，v_i と v_j のすべてのペアに対して，$v_j - v_i = v_k - v_j$ となる v_k を事前に求めておく。等差数列の上で v_i，v_j の次になる値である。この作業は，v_j から左右両方向に，マージと同様の要領で添字を並行して動かしていけば高速に実現できる。具体的には，v_j との値の差が小さい方の左右いずれかの添字を v_j から遠ざかる方向に一つ動かす。これを繰り返していけば，v_j との差が同じである v_i と v_k の組をすべて求めることができる。後は，任意の 2 数から始まる等差数列を伸ばしていくだけだが，$1, 2, 3, \ldots$ を調べた後で，$2, 3, \ldots$ を調べるような無駄を避ける必要がある。等差数列上で隣り合った二つの値のペアに印を付けておいて，次に同じペアが別の等差数列に現れたら，その等差数列はそこで打切りにすればよい。全体の計算量は $O(n^2)$ になる。

もっと簡単な方法で解くことも可能である。最初に値すべてを表に入れておいて，$2v_j - v_i$ を探すようにすれば，v_k は簡単に見つかる。表の具体的な構成法としては，C++の `unordered_set`，Java の`HashSet` を使うのが常識的だが，長さ 10^9 の論理値配列を使えばもっと高速になる。この方法なら，上に述べた同じ値のペアを 2 回以上調べる無駄は気にしなくてもよいようだ。ある数を初項とする等差差数列上の値をスキャンする回数の総和は，調和級数の n 倍で抑えられるので，全体の計算量は $O(n^2 \log n)$ を超えない。これでも間に合う。

12 チームがこの問題を解けずに終わった。いずれも，この問題に挑戦して，何度もプログラムを提出していたのだが，最後まで制限時間の壁を破れなかった。

問題 C：Emergency Evacuation

バスからの脱出のプロセスを裏返してみると，解法が見えてくる。乗客全員がバスの外にいる状態から，1 ステップに 1 人ずつバスに乗り込んでいく状況を想定する。全員が自分の所定の席に着くまでの時間が分かれば，それが問題で要求している脱出の所要時間に等しい。乗り込む際に乗客間の待ち合わせは生じないので，元の問題より易しくなっている。

k 番の客の出口から席までの距離を d_k とする。この客が時刻 t_k（$0 \le t_k \le p-1$）に乗り込むとすると，$\max(d_k + t_k)$ が全員が着席するまでの所要時間になる。後は，客を乗り込ませる順序を決めるだけだが，この式の値を小さくするには，席までの距離の大きい客ほど先に乗らせるのがよいことは明らかである。p 人の客の席までの距離をソートすれば解けたも同然である。

別解として，脱出に要する時間を 1 人ずつ決めていく解法も成立する。各人単独の脱出所要時間をソートして，この値の小さい客から順に調べる。自分の所要時間が前の人の時間以下なら，前の人の時間に 1 を加えた時間を自分の時間とする。同じ時間の人が複数いれば，一方は待たされるはずだからである。待たされた人がまた別の人の邪魔になる状況もあり得るが，このことの影響もこの操作で正しく表現できている。この操作を続けていって，最後の人の脱出時間を答とすればよい。

解法さえ分かれば，コーディングには何の難しさもない問題である。審判団の中では，解法設計が難しいという意見と，非常に易しい問題であるという意見の両方があった。結果は，60 チーム中 53 チーム正解で，C の位置に置く問題として適当な難易度だったと考えている。

問題 D：Shortest Common Non-Subsequence

最初に，0 と 1 から成る長さ k の文字列の non-subsequence とはどんなものか考えてみよう。文字列の最後に don't care 文字（0 とも 1 ともマッチする）を追加した上で，non-subsequence を認識するオートマトンを考える。このオートマトンの状態は，文字列中の文字と文字の間の位置である。状態には 0〜$k+1$ の番号を与える。状態 0 は先頭の文字の直前，1〜$k+1$ は各文字の直後の位置を表す。

文字列中の各位置から，それより後で最初に現れる文字 0 の直後の位置に 0 で遷移する辺を張る。文字 1 についても同様である。don't care 文字はどちらともマッチするので，現在位置より後に該当の文字がない場合は，位置 $k+1$ に遷移するようにする。このオートマトンの上で，位置 0 から出発して位置 $k+1$ に到達する状態遷移を起こす文字列が non-subsequence である。don't care 文字を通過していることが，元の文字列に含まれないことの証しと考えればよい。

本題である A と B の共通 non-subsequence にも，同じ考え方が適用できる。A 中の位置と B 中の位置のペアを頂点とするグラフを考える。各頂点から文字 0 と文字 1 に対応する辺を張る。0 の辺の行き先は，A，B それぞれの次の 0 の位置のペアである。1 の辺の行き先も同様に決める。A と B の長さをそれぞれ m，n とすると，このグラフの上で，(0, 0) から出発して $(m+1, n+1)$ に到達する最短の経路を求めればよい。この問題は，幅優先探索によって簡単に解ける。計算量は $O(mn)$ であり，十分に小さい。0 を 1 より先に調べることにすれば，辞書順最小という要求も自然に満たされる。

同じ問題を動的計画法を適用して解くことも可能で，こちらの方が少し速くなる。

問題 E：Eulerian Flight Tour

一筆書きができるように，グラフに辺を追加していく問題である。ただし，多重辺を追加することはできない。一筆書きに関する有名な定理に従えば，すべての頂点における次数が偶数で，かつ連結なグラフを構成できればよいことが分かる。

想定解法は，二つのフェーズに分けて解くものである。最初に，連結性は無視して，とにかくすべての頂点の次数を偶数にする。その後で，次数を偶数に保ったまま連結になるように，さらに辺を加えていく。

最初のフェーズは，線形連立方程式で解決するのが簡単である。元のグラフに含まれない辺のそれぞれを追加するか否かを表す変数を考える。各変数の値は 1（辺を加える場合）または 0 である。これらの変数に関する連立方程式をたてる。それぞれの頂点に関して，辺を加えた結果の次数が偶数になることを表す方程式を作り，これを並べればよい。連立方程式に関する計算は，すべて mod 2 で行う。連立方程式は，掃き出し法などの普通の解法を使って $O(n^4)$ の計算量で解ける。実は，連立方程式を使わずに，計算量をもっと小さくできる解法もあるのだが，この程度の解法で十分である。

第 2 のフェーズは，最初のフェーズで作った偶数次数のグラフの形に関する場合分けになる。こちらの方が難しい。グラフが連結なら，それで終わりである。連結でなく，三つ以上の連結成分に分かれている場合は，各連結成分から一つずつ頂点を選んで，それらをリング状に結べばよい。連結成分が二つで，両者とも二つ以上の頂点から成るのなら，二つずつ頂点を選んで，2×2 本の辺すべてを加える。一方が 1 頂点（孤立点）で，もう一方が完全グラフでなければ，辺で結ばれていない 2 頂点と孤立点を三角形になるように結ぶ。完全グラフであっても，その中に最初のフェーズで加え

た辺が含まれている場合は，その辺をはずして，両端を孤立点とつなぐ。結局，元の問題が解けないのは，最初のフェーズの連立方程式に解がない場合と，元のグラフが孤立点と完全グラフに分かれている場合の二つだけである。

中程度の難しさの問題と想定していた。9 チームが解いた結果は，ほぼ想定どおりである。

問題 F：Fair Chocolate-Cutting

最初に，多角形の各頂点を通って多角形の面積を二等分する直線が多角形を横切るもう一つの点を求める。この点を対向点と呼ぶことにする。対向点の位置を求めるには，まず最初の頂点から頂点を一つずつ加えていった部分多角形の面積が全体の半分を超える所まで進める。その後で，三角形の面積と底辺の長さに関する比例計算をすればよい。

多角形の周は，n 個の頂点と n 個の対向点で $2n$ 個の区間に分けられる。しかも，頂点から始まる区間とその対向点から始まる区間がペアになっていて，面積二等分の線の両端は，ペアの区間の中を連動して動く。面積二等分線の長さが最大になるのは，区間ペアの端に限られる。それぞれの端で面積二等分線の長さを計算すれば，最大値は簡単に求まる。最小値の方は，二次関数の最小値として代数的に求めることもできるし，三分法などの反復近似計算で求めることもできる。長さが最小になるのは，二つの区間の属する 2 辺がなす角の二等分線と垂直な線になる場合である。

幾何の問題としてはそれほど難しいものではないと考えていた。ところが，実際の結果はこの想定とは大きく違っていた。トップ 2 チームがこの問題以外の 10 問をすべて解いた後で，この問題を解けずに 2 時間近く奮闘していた。誤差に起因するエラーが解消できずに苦労していたらしい。ある頂点の対向点が別の頂点に等しい場合など，誤差の影響の現れやすい落とし穴がいくつかあるようだ。

問題 G：What Goes Up Must Come Down

初めに，どんな計算をすれば答が求まるかを考える必要がある。その上で，その作業を高速に行うためのアルゴリズム（またはデータ構造）の工夫が必要である。いずれも，それほど難しくはない。

値の小さいカードから順に目標の場所に移動することを想定する。すると，左に動かす場合は自分の左にある自分より大きいカードの枚数だけ移動する必要があることが分かる。右に動かす場合も同様である。つまり，それぞれのカードから見て，自分の左側と右側のそれぞれについて，自分より大きいカードが何枚あるかを数えて，枚数の少ない方に動かせば，移動ステップ数が最小になる。この値を全カードについて合計すれば答が得られる。

この作業を愚直に実現すると，計算量が $O(n^2)$ になって遅すぎる。値を降順にソートして，大きな値から順に，元の配列中の位置をデータ構造に入れていくことを考える。このデータ構造から，自分の位置より左のデータが何個あるかが高速に求められるようになっていれば，問題は解決する。最適なのは，binary indexed tree と呼ばれるデータ構造で，これを使えば，この操作を高速かつ簡明に実現できる。計算量は，1 回当たり $O(\log n)$，プログラム全体で $O(n \log n)$ である。binary indexed tree を知らなくても，セグメント木などの一般的なデータ構造で同じ計算量を実現できるし，平方分割でも十分に速くできる。

ソートされた値を調べる時，同じ値が複数個ある場合の処理に注意が必要である。自分に等しい値を自分より大きいと見誤ることがないようにしなければならない。同点の値は一まとめに扱うなどの注意が必要になる。また，答の最大値が 25 億近くになるので，`int` ではオーバーフローが起きる。

最初の 3 題の次に易しい問題である。36 チームが正解した。

問題 H：Four-Coloring

頂点の色を一つずつ決めていく。この際，4 色のどれを選んでも制約に違反する場合は，すでに決めた頂点の色を変えて，色が選べるようにする。次に示す手順に従えば，このような操作が可能である。

頂点を座標値に基づいてソートして，y 座標の昇順，y 座標が同じなら x 座標の昇順に頂点の色を決めていく。ある頂点を調べる時，自分に隣り合っていて，すでに色の決まっている頂点は最大 4 個である。これが 3 個以下であるか，4 個でもその中に同じ色のものがあれば，どれとも違う色を選ぶことが可能である。

4 個の頂点があって，すべての色が違っている場合は，その中の一つの色を変えることを試みる。4 個の頂点を左，左上，上，右上と呼ぶことにしよう。これらの色を順に a，b，c，d とする。左の頂点と上の頂点を色が a または c の頂点だけを通って結ぶ経路があるかどうか調べる。このような経路がなければ，左の頂点の色を c に変えることが可能である。左の頂点の属する ac 連結成分の各頂点の色をすべて反転（$a \to c$，$c \to a$）すればよい。経路がある場合は，左上の頂点と右上の頂点を色が b または d の頂点だけを通って結ぶ経路はないはずである。両方の経路があれば，グラフが平面グラフなので，途中で交差するはずで，交差位置の頂点の色に矛盾が生じるからである。この場合は，左上の頂点の色を d に変えることが可能になる。結局，4 個の頂点のうちのどれか一つの色を変えて 3 色にすることが常に可能である。

解法を設計することが難しい難問と想定していた。ただし，解法が分かれば，コーディングは難しくない。正解したのは 2 チームだけだった。

問題 I：Ranks

A^{ij} のそれぞれに対して独立にランクを求めれば解けるが，これでは遅すぎる。行列のランクは，掃き出し法を使って求めることができる。計算量は 3 乗のオーダーである。これを 2 乗オーダーの回数繰り返すのでは，計算量が 5 乗のオーダーになってしまう。3 乗のオーダーで，つまり 1 回の掃き出し法で，答が求まるようにしなければならない。

想定解法では，元の $n \times m$ 行列の右に $n \times n$ の単位行列を連結してできる $n \times (m+n)$ 行列を考える。この行列の上で m 番目の列まで掃き出し法を実行する。すると，m 列目の上から i 番目の要素を反転した場合の掃き出し法の結果の m 列目は，掃き出し後の m 列目と $(m+i)$ 列目の排他的論理和になっているはずである。したがって，$j = m$ のケースは簡単に解決できる。

$j = m$ 以外のケースにも対処するために，まず m 列目までの掃き出し法の後に後退消去を行う。その上で，j 列目を抜き取って m 列目の直後に移すことを考える。この際，j 列目の一番下の 1 を含む行に他にも 1 があれば，列を移動した結果の行列も階段状になっていて，列の移動をしてから掃き出し法を実行した結果と同一視できる。この場合は簡単である。一番下の 1 を含む行に他に 1 がない場合は，行の入れ替えを行って，さらに列の入れ替えを行うことで，階段状にすることができる。これは後退消去を行った効果である。

結局，掃き出し法を 1 回行うだけで，すべての A^{ij} のランクを求めることができる。計算量は $O(n^2(m+n)/b)$ と表すことができる。ここで，b は 1 ワード当たりのビット数で，32 または 64 である。b ビットの論理演算を一まとめに行うことを想定している。

問題 J：Colorful Tree

質問（query）のたびに木の大きさを求める素朴な方法では，計算量が $O(nm)$ になって遅すぎる。色ごとに，その色の頂点すべてを含む最小の木の大きさ（辺の数）を記録しておいて，色を変える

コマンドの実行の際に，この値の変化を求める方法が現実的である。ある頂点の色を変えた時，元の色の木が小さくなり，新しい色の木が大きくなるが，その際の大きさの変化がそれぞれ $O(\log n)$ で計算できるようになっていればよい。

最初に，全体の木を深さ優先探索し，訪問順に頂点に番号を付ける。番号の付け方は，pre-order でも post-order でもよい。それぞれの色について，その色の頂点の番号の集合を記憶する。C++なら `set`，Java なら `TreeSet` を使う。また，その色の頂点の個数と，その色の木（その色の頂点すべてを含む最小の木）の大きさ，それに，その色の木の根を記録しておく。木の根は，その色の頂点すべての LCA（lowest common ancestor）に等しいことに注意。

ある色の木に新たに頂点を加える時は，木の根（頂点 r）と新たな頂点（頂点 p）の LCA（頂点 q）を求める。q が r と違っている場合は，新しい根 q から今までの根 r への経路と q から p への経路が重なっていないので，q と r の距離と q と p の距離の二つを足した値だけ木の大きさが増える。q が r と同じであれば，p は既存の木の途中から枝分かれすることになるので，枝分かれの分岐点を求めて，p から分岐点までの距離だけ木の大きさを増やせばよい。分岐点は次のようにして求められる。その色の頂点の中で，p の番号以上で最も小さな番号を持つ頂点（頂点 x）と，p の番号以下で最も大きな番号を持つ頂点（頂点 y）を考える。p と x の LCA と，p と y の LCA の二つのうち，p に近い方が分岐点である。ある色の木から頂点を削除する操作は，頂点を加える操作の裏返しなので，同じように考えればよい。

LCA を求める操作は 1 回当たり $O(\log n)$ であり，これを定数回だけ行えばよいので，色を変える操作 1 回当たりの計算量はやはり $O(\log n)$ で済む。ほかに，自分の次の番号を持つ頂点を探す操作が必要だが，これも平衡木（`set` または `TreeSet`）の操作として $O(\log n)$ のものが提供されている。結局，プログラム全体の計算量は $O((n+m)\log n)$ である。

グラフとデータ構造に関する知識を駆使することが要求される難問であるが，ポイントを整理して考えれば十分に解ける問題だと考えていた。5 チーム正解という結果は，想定どおりである。

問題 K：Sixth Sense

両者の手札をそれぞれ降順にソートしてあれば，何回勝てるかが $O(n)$ で分かる。自分の強い札から順に，その札で勝てる相手の一番強い札を対応させていけばよい。ここまでは簡単に分かるだろう。最初にこの操作を使って勝ち数を求めるが，ほかに本題のプログラムのサブルーチンとしてもこの操作を使う。

難しいのは辞書順最大の出し方を求めることである。それぞれの回に，なるべく強い札を出すことが求められる。ただし，ゲーム全体の勝ち数を減らさないようにしなければならない。強すぎる札を無駄に使うと勝ち数が減る。

ある回を，勝つと決めた場合と負けると決めた場合に分けて考えてみよう。勝つと決めたとして，ある値より上の札を出すと勝ち数が減り，その値以下なら勝ち数が変わらないことが分かる。このことから，二分法を使って，勝ち数が減らないギリギリの強さの札を決めればよいことが分かる。二分法の繰返しのたびに，該当の札を除いた残りの手札で，それ以降の勝ち数がいくつになるかを最初に述べたサブルーチンで計算すればよい。負けると決めた場合も同様である。勝つ場合と負ける場合のそれぞれについて，別の二分法を用意する必要がある。プログラム全体の計算量は $O(n^2 \log n)$ である。