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	1	0	1	2	2	4	12	9	11	4	3	1
チーム数累計	1	1	2	4	6	10	22	31	42	46	49	50

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	49	39	42	2	10	12	19	2	35	1	3
失敗数	0	8	3	2	11	14	5	3	2	0	2
提出数	51	95	82	7	61	81	56	6	51	1	8

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

昨年までは易しい問題から順に A，B，C，... と並べていたが，今年はこの方式をやめた。世界大会などと同じく，難易度と無関係な並べ方にした。ただし，最初の A，B，C だけは一番易しい 3 問にした。

プログラミング言語として Python を使えるようにしたことも，今年の新機軸である。実際に Python を使って問題を解いたチームがあったが，その数は少なかった。ほかに，Java を使用するチームが極端に少なくなったことが目についた。

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

実際の結果は，ほぼこの予想どおりだったが，いくつか審判団の予想を裏切る結果になった問題があった。一番驚かされたのは問題 J で，1 チームしか解けず，結果的に最難問になってしまったが，本当はそれほど難しい問題ではない。また，問題 B の正答数が予想ほど伸びず，審判団をがっかりさせた。

トップのチームは，1 時間以上残して全問正解した。この結果だけ見ると，問題が易しすぎたという評価になるだろう。しかし，2 位以下のチームの結果を見れば，これとは違う評価が可能だと思う。もう少し難しくして，2 位以下のチームの正解数が減ってしまったら，かえって面白くない結果になったと思われる。実力差が大きいので，審判団としてこれ以上のことはできなかったと考えている。

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：Secret of Chocolate Poles

高さがちょうど h になるようなポールの作り方の数を P_h とする。これが計算できれば，$h = 1$〜l の範囲の P_h の和を求めるだけで，問題の答が得られる。h が大きい場合の P_h に関する漸化式は，一番上の黒の円盤が薄い場合と厚い場合に分けて考えると，$P_h = P_{h-2} + P_{h-(k+1)}$ であることが分かる。難しいのは h が小さい場合だが，$P_0 = 0$，$P_1 = 1$，$P_h = P_{h-2}$ $(2 \leq h \leq k-1)$，$P_k = P_{k-2} + 1$ とすればよい。この漸化式をそのまま計算する再帰的な関数を書くと，Fibonacci 数の再帰的計算と同じようなことになって計算時間がかかるが，h の小さい方から順に求める動的計画法を使うか，再帰的関数にメモ法を適用すれば問題なく解ける。

ほかに別解として，厚い黒の円盤の枚数を a，薄い黒の円盤の枚数を b として，$(k+1)a + 2b \leq l + 1$ であるような a と b の組合せを列挙し，${}_{a+b}\mathrm{C}_a$ の総和を求める方法もある。$k+1$ と 2 は，それぞれ隣の白の円盤も勘定に入れた時の厚さである。

全チームが解ける一番簡単な問題を意図していた。実際，全 50 チーム中 49 チームがこの問題に正解した。

問題 B：Parallel Lines

素朴な全探索が想定解法である。いろいろな工夫が可能なように見えるが，面倒になるだけで，あまり得にはならない。素直な方法で十分だと判断する能力も，プログラミングには重要である。

ごく単純に，m 個の点を 2 個ずつペアにするやり方を片端から生成して，平行な線対の数をそれぞれ数えればよい。ペアにするやり方の全生成は，単純な再帰的探索で可能である。再帰のそれぞれのレベルで，残っている点のうちの一つを固定して，それとペアにする相手の点を片端から試していけばよい。生成される組合せの数は，最大の $m = 16$ の場合でも $15 \times 13 \times 11 \times 9 \times 7 \times 5 \times 3 \times 1 = 2027025$，つまり約 2 百万にしかならない。計算時間を気にする必要のない数である。2 本の線分が平行であることは，それぞれの両端の点の座標の差を (x_1, y_1)，(x_2, y_2) とすると，$x_1 y_2 - x_2 y_1 = 0$ で判定できる。

基本的な再帰的全探索の問題であり，問題 A に次いで易しいと想定していたが，解けたチームの数 39 は審判団の想定を下回った。標準的な演習問題とは少し違う再帰的探索にとまどったのだろうか。誤答の中では，16! とおりの順列をすべて生成することによってペアにするやり方の全生成を目指すプログラムが目についた。これでは明らかに計算時間が不足する。

問題 C：Medical Checkup

簡単な例についてタイムチャートをいくつか書いてみれば，解法はほぼ明らかになるだろう。検査に要する時間の長い人の後に短い人が並んでいる場合，後の人は前の人の検査が終わるまで常に待たされる。つまり，一つの検査を始めてから次の検査を始めるまでの時間間隔は，前の長い人の所要時間に等しい。この事実が分かればプログラムまで後一歩である。

より一般化すると，それぞれの人の検査間の時間間隔は，自分以前の人（自分を含む）の所要時間の最大値に等しい。先頭から順に所要時間の最大値を求めながら，各人の時間間隔と最初の検査を始める時刻を決めていけばよい。時刻 t の時点で何番目の検査を受けているかは，簡単な割り算で求められる。

3 番目に易しい問題と想定していた。問題 B が予想より難しかったので，結果的には 2 番目に易しい問題になったが，42 チーム正解という結果はほぼ想定どおりである。計算の途中に 32 ビット整数では表現できない数が現れる可能性があり，64 ビット整数の使用が必須である。この点が罠になっていると言える。

問題 D：Making Perimeter of the Convex Hull Shortest

幾何の難問である。個々の操作は難しくないのだが，境界ケースに漏れなく対処することがかなり難しい。解いたチーム数は 2 にとどまったが，これはほぼ想定どおりである。

何はともあれ，凸包を求めるアルゴリズムが必要である。$O(n \log n)$ の計算量で凸包を求めるアルゴリズムはいくつも存在するが，この問題に適したアルゴリズムとそうでないアルゴリズムがある。この問題では，点を削除した時に凸包を部分的に求め直す必要がある。これが簡単にできるアルゴリズムが望ましい。Andrew のアルゴリズムを採用するのが最善だというのが審判団の結論である。Graham スキャンなどでも，同様のことは可能なはずだが，角度の基準となる点を削除した時にソートのやり直しが必要になる。Andrew のアルゴリズムなら，そのようなことを考える必要がなく，シンプルな解法が可能である。

Andrew のアルゴリズムは，与えられた点を x 座標の順にソートする。同点の場合は y 座標もソートに使う。左右両端の点の間のスキャンを 2 回行って，それぞれ凸包の上半分と下半分を求める。スタックに点を順に入れながら，凸包に含まれないと分かった点を削除していけばよい。スキャンの際に行う操作は，点 A から B の方向を向いた時に，次の点 C が右に見えるか左に見えるかの判定だけである。どの点を除いた場合でも，ソートの結果は変わらないから，前の計算の結果を生かすことができ，この問題に適している。

初めに，全部の点に対する凸包を求める。この多角形を H と呼ぶことにしよう。H に含まれていない点を削除しても凸包の長さは変わらないので，問題で要求されている 2 点の選び方は，次の三つのケースだけ考えればよい（それぞれサンプルの 1〜3 に対応している）。

(1) H の上で隣り合わない 2 点
(2) H の上で隣り合う 2 点
(3) H の上の 1 点と，それを削除した時にできる新しい凸包上の 1 点

H の上の 1 点 B を削除した時の新しい凸包は，その隣の 2 点を A と C として，最初にソートした点列の上で A から C までの区間だけ Andrew スキャンをやり直せば求められる。この操作を H の上のすべての点に対して 1 回ずつ行う必要があるが，全部で $O(n)$ の計算量にしかならない。たとえば，A と B の間の点は，A を削除した時と B を削除した時の 2 回しかスキャンの対象にならないので，こう結論づけられる。

(1) のケースは，二つの点での凸包の変化が独立なので，それぞれの短縮値の和を求めればよい。ただし，二つの点が隣り合っている場合を排除しなければならない。短縮値最大の二つを記録するだけだと，隣り合う点を選んでしまう可能性が残る。短縮値の上位 4 位までを記録しておくのが解決策である。上位 4 位までの中には，隣り合わない 2 点が必ず含まれる。

(2) のケースも，方法はほとんど同じである。点 B と C を削除した時，その 2 点をはさむ点 A から D までの Andrew スキャンをやり直せばよい。この場合の計算量も，全部で $O(n)$ で済む。(3) も，話がやや複雑になるだけで，基本的には同じである。計算量は，やはり全部で $O(n)$ になる。

プログラム全体としては，(1)〜(3) のすべてのケースを計算して，短縮値最大の結果を答とすればよい。プログラムの計算量は，最初のソートで決まり，$O(n \log n)$ である。

詳しくは述べなかったが，左右の端の点を削除した時の Andrew スキャンには，ほぼ間違いなく特別の配慮が必要になる。このような特別なケースまで正しく扱うには，かなりのコーディング力，デバッグ力が必要である。

問題 E：Black or White

動的計画法（以下 DP と呼ぶ）を使って解くのが想定解法である。ただし，DP の漸化式を構成するための考察が容易ではなく，一山も二山も越えなければならない。

$i+1$ 番以降の煉瓦には手を触れずに，1 番から i 番までの煉瓦を目的の色にするために必要な最少の手数を f_i とする。t_i が s_i に等しければ，$f_i = f_{i-1}$ である。t_i が s_i と違う場合は，この煉瓦を必ず塗らなければならない。この時，この煉瓦を含めて何個を一度に塗るかに選択の余地があり，1 個から k 個までのすべての可能性を考えなければならない。j 個を一度に塗った場合，$i-j+1$ 番から $i-1$ 番までがいったん t_i と同じ色になるので，この範囲に含まれる t_i と逆の色の区間の数だけ塗り戻す必要がある。

t_i が白の場合を考える（黒の場合も同じ考え方で別に計算する）。$1 \le r \le p$ を満たす煉瓦 r 番の中で，t_r が黒，t_{r+1} が白（または $r+1 > n$）であるものの個数を b_p とする。黒の区間の右端に当たる煉瓦の個数である。$t_i \neq s_i$ の場合の f_i は，$j = 1, 2, \ldots, k$ における $f_{i-j} + (b_i - b_{i-j}) + 1$ の最小値である。$b_i - b_{i-j}$ が $i-j+1$ 番から $i-1$ 番までにある黒の区間の数になる。これで DP による解法が構成できたが，このままでは $O(nk)$ の計算量が必要で遅すぎる。もう一段の工夫が必要である。

f_i の式は $(b_i + 1) + (f_{i-j} - b_{i-j})$ と変形できるので，$f_{i-j} - b_{i-j}$ の最小値を求めればよいことが分かる。つまり，数列 $f_j - b_j$ の幅 k の範囲の最小値を順次求める計算になる。最初の k 番目までの最小値は普通に求めるが，$k+1$ 番を入れたら 1 番は除外する，$k+2$ 番を入れたら 2 番は除外する，... という操作をしながら，その時点の範囲の中の最小値を順次求めていけばよい。この計算は，セグメント木やヒープを使う標準的な方法で，1 回当たり $O(\log n)$ の計算量で可能である。この方法を使えば，プログラム全体の計算量は $O(n \log n)$ になる。ここまでできれば正解である。

固定幅 k の最小値を求めるためのより効率的な方法として，スライド最小値と呼ばれるものがある。現在の範囲の中で将来最小値になり得る値の位置だけを記録したデータ構造を用意する方法である。スライド最小値を用いれば，1 回当たりの計算量は $O(1)$ になり，プログラム全体の計算量は $O(n)$ になる。

中程度の難しさの問題と想定していた。コーディングよりも，DP による解法の設計に力点を置いた問題である。いったん解法が設計できれば，コーディングの量は少ないのだが，そこに到るまでがかなり厄介である。10 チームが正解した。

問題 F：Pizza Delivery

有向グラフで道路網をモデル化する。辺のうちの 1 本の向きを反転した時に，出発地から目的地までの最短路の長さが長くなるか（到達不能になるケースを含む，SAD），短くなるか（HAPPY），変わらないか（SOSO）を答えればよい。

HAPPY になる辺は簡単に判定できる。$A \to B$ の辺を反転した時に最短路が短くなるのは，出発地から B までの距離，A から目的地までの距離，$A-B$ の長さの三つを加えたものが元の最短路の長さより短い場合である。出発地から各点までの距離は Dijkstra のアルゴリズムで求まるし，各点から目的地までの距離は辺の向きを逆にしたグラフの上で目的地から Dijkstra のアルゴリズムを適用すれば求まる。全地点についてこの二つの値を求めておけば，後の計算は簡単である。Dijkstra のアルゴリズムの計算量は $O(m \log n)$ である。

HAPPY でない辺のうち，最短路の一部ではない辺を反転しても，最短路の長さが変わることはない。この種の辺は SOSO だと判定できる。

残るケース，つまり最短路の一部である辺の場合，SOSO なのか SAD なのかは，別のアルゴリズムを適用しないと判別できない。辺のうちで，最短路の一部となり得るものだけを残したグラフを構成する。最短の経路が複数ある場合は，それらのどれか一つにでも含まれている辺をすべて残すという意味である。ある辺を削除した時にこのグラフが非連結になるようなら，その辺は SAD であり，そうでなければ SOSO と判定できる。削除した場合にグラフが非連結になるような辺を橋（bridge）と呼ぶ。橋の検出は，深さ優先探索をしながら経由した頂点に番号を付けていく技法によって，$O(n+m)$

の計算量で可能である。この技法は，強連結成分，関節点など，グラフの問題の多くに適用可能なので，勉強しておいて損はない。

中程度の難しさの問題と想定していた。グラフの典型的な応用問題である。最短路アルゴリズムが基本になるが，ほかのグラフアルゴリズムに関する知識も要求される。コーディングの量が多く，それなりの難しさではあるが，問題の定式化には変化の余地がなく，比較的素直な問題だと言ってよかろう。ちょうど問題 E と反対の性格の問題と言える。12 チームが正解した。

問題 G：Rendezvous on a Tetrahedron

正四面体の上での動きを忠実にシミュレートしようとすると，立体図形を扱うことになって容易でない。展開図の上での動きに焼き直して考えることが正解までの第一歩である。辺にぶつかった時に角度を変えずに動くことになっているので，展開図の上での直線運動だけを考えればよい。

平面上に無限個の正三角形を隙間なく敷き詰めたものを正四面体の展開図と見なす。与えられた角度と移動距離でこの展開図の上を動いた時に，どの面の上で止まるかを判断すればよい。展開図からはみ出すようなことを考える必要はなく，非常にシンプルなプログラムになる。展開図の上には，一つの面，たとえば面 ABC に対応する正三角形が規則的に並んでいる。この規則を解読して，それをプログラムにすれば，止まった位置がどの面に属するかを判定することができる。

易しい幾何の問題である。正解の解法に特に難しい点はないが，単純な直交座標とは違う世界を扱うので，ケアレスミスを犯しやすい。実際，正解数 19 に対して，提出数 56 になっていて，誤答がかなり多かったことが分かる。

問題 H：Homework

最大値と最小値の二つを求めることになっているが，最大値の方は簡単である。科目が決まった時の宿題の選び方が問題文に与えられている。その日に解答できる宿題のうち，締切が一番迫っているものを選ぶことになっている（貪欲戦略）。この選び方は，処理する宿題の数を最大化するための最善の戦略なので，二つの科目の選び方にもこの戦略を適用すればよい。締切が一番迫っている方の科目を常に選ぶことにすれば，処理できる宿題の数が最大になる。

最小値の方はそれほど簡単ではない。一見すると，簡単な探索で解けるように見えるかも知れないが，実は奥の深い問題である。想定解法は次のとおりである。数学の宿題全体を A，情報の宿題全体を B，日全体の集合を D とする。$(A \cup B) \cup D$ を頂点集合とし，宿題とそれを処理できる日それぞれとの間に辺を張った二部グラフを考える。数学を選んだ日の集合を X，情報を選んだ日の集合を Y としよう。貪欲戦略が最適であるという考察から，処理できる宿題の数は，二つの二部グラフ (A, X) と (B, Y) の最大マッチングの大きさの和に等しいことが分かる。したがって「日の集合 D を X と Y に分割したときの最大マッチングの和」の最小値を求めればよい。

今，ある部分集合 $Z \subseteq D$ に対して，二つの二部グラフ (A, Z) と (B, Z) がともに大きさ $|Z|$ のマッチングを持つとしよう。すると，どのような D の分割に対しても，最大マッチングの和は $|Z|$ 以上となることが簡単に分かる。この条件を満たす最大の Z は，最大流アルゴリズムで求めることができる。日の集合 D と同じものをもう一つ用意し D' とする。二つの二部グラフ (A, D) と (B, D') を作り，D と D' の同じ日同士を辺でつなぐ。ソースから A の各宿題に，B の各宿題からシンクに辺を張る。辺の流量はすべて 1 にする。こうしてできたグラフの上で最大流を求めれば，それが上の問題の答である。

さらに，最大流に対応する最小カットから分割 $D = X + Y$ を得ることができ，この分割に対して最大マッチングの和が $|Z|$ と等しいことを示すことができる（証明は簡単ではない）。したがって，

そのような $|Z|$ が求めたい最小値に等しい。最小値が最大流で求まるという一見矛盾した状況が面白い。

解法設計の面での最難問と想定していた。ただし，解法が設計できれば，コーディングは難しくない。データの個数が少ないので，最大流アルゴリズムは比較的遅いものでも十分である。正解したチーム数は 2 だった。

問題 I : Starting a Scenic Railroad Service

問題で要求されている二つの値を求めるために，どんな計算をすればよいかを最初に考える必要がある。ポリシー 2 の方は比較的簡単だろう。ある駅とその次の駅の間の乗車人数が最も多い区間の人数が答である。乗車駅番号の小さい客から順に席を割り当てていくことを想定すれば，これだけの数の席があれば十分であることが分かる。

ポリシー 1 の方はもう少し難しいが，いくつかの例を作って，必要な席の数を考えてみれば，一般のケースについての規則を推論することができるだろう。自分の乗車区間と一部でも重なりのある人の数の最も多い客を考える。この客と重なりのある人数（自分自身を含む）がポリシー 1 の答である。これだけの数の席があれば，乗車区間が重なっている人が全部違う席を選んだ場合でも，空いている席が残っているはずである。

各乗客の乗車と降車の二つのイベント，計 $2n$ 個を記録した配列を作り，これを駅の順にソートする。同じ駅の場合は，降車の方が乗車より前になるようにする。このデータがあれば，二つの値の計算は簡単である。ポリシー 1 の計算は，このデータを順に調べながら，乗車と降車の人数を数えていく。各乗客に対して，乗ろうとする時にすでに降りてしまった人数と，降りようとする時にまだ乗っていない人数を記録する。n からこの二つの人数の和を引いた値の最大が答になる。ポリシー 2 の計算はもっと簡単で，乗車している人数を増やしたり減らしたりしながら，人数の最大値を求めればよい。計算量は，最初のソートで決まり，$O(n \log n)$ である。

ほかに，駅の番号を添字とする配列を使う計算法も可能で，こちらなら，駅番号の最大値を m として計算量は $O(n + m)$ になる。

最も易しい問題の次くらいの難しさを想定していた。35 チームが解いた結果は，ほぼ想定どおりである。問題文の読解が難しい点を少し心配していたが，それは杞憂だったようだ。

問題 J : String Puzzle

部分文字列が等しいというヒントをどう活用するかがポイントであることは一目で分かると思う。このヒントが任意の部分文字列に対して与えられるようだと，とても手に負えないのだが，実際にはヒントの与え方に強い制約がある。この制約をしっかりと把握することがこの問題を解く鍵である。

説明の便のために，y_i から始まる部分文字列を根元側，h_i から始まる部分文字列を行き先側と呼ぶことにしよう。ヒントに関してポイントとなる制約は二つある。一つは，$y_1 < \cdots < y_b$ という制約である。ヒントは，y_i から始まる長さ $y_{i+1} - y_i$ の部分文字列に対して与えられている。つまり，y_i から $y_{i+1} - 1$ までの部分文字列が対象である。隣の部分文字列との重なりはない。このことから，一つの箱に適用される根元側のヒントは最大一つしかないことが分かる。もう一つのポイントは，$h_i < y_i$ という制約である。h_i が y_i より小さいので，行き先側の部分文字列は常に根元側よりも左にある。

二つの制約を合わせて考えると，ある箱の文字に等しいと結論づけられる最も左にある箱の番号を確定できることが分かる。箱の番号が根元側の区間に入っていれば，それを行き先側の区間内の対応する番号に置き換える。根元側と行き先側の部分文字列に重なりがある場合も考えて，元の番号 k を $(k - y_i) \bmod (y_i - h_i) + h_i$ に置き換えればよい（mod は割り算の余りを求める演算子）。行

き先の番号が別の根元側の区間に入っていれば，同じことを繰り返す。こうやって左に行けるだけ行った箱の番号を元の箱の正規化番号と呼ぶことにしよう。x_i 番の文字が c_i であるというヒントによって文字が確定した箱それぞれの正規化番号を求めておいて，質問それぞれの正規化番号がこれらのどれかに等しければ，その文字を答とすればよい。どのヒントの正規化番号とも等しくなければ，不明とするのが答である。

中程度の難しさの問題だと考えていたが，実際の結果は審判団の予想を大きく裏切った。1 チームしか正解しなかったし，問題に挑戦したチームもこの一つだけだった。上に述べたように，制約さえきちんと把握できればそれほど難しい問題ではないので，残念な結果である。ポイントとなる制約は，問題文の中や入力の制約条件としてはっきり書いてあるのだが，さりげない書き方だったので，気づかないチームが多かったようだ。

問題 K：Counting Cycles

入力として与えられるグラフの頂点数 n と辺の本数 m に関して $n-1 \leq m \leq n+15$ という不思議な制約が与えられている。これが意味するのは，グラフが木に非常に近いということである。木にごく少数の辺が加わっただけのものになっている。木に追加された辺の数を k とすると，$0 \leq k \leq 16$ である。

木に追加された辺を一つ選び，その両端から木の根に向かって歩いていくと，どこかの頂点で左右の端からの経路がぶつかる。この時に通った辺を全部合わせれば一つの単純閉路になっていることが分かる。つまり，k 本の辺のそれぞれに対して単純閉路が定まる（これを基本サイクルと呼ぶことにしよう）。逆に k 本の辺のどれも通らなければ閉路を作れない。残る可能性は，k 本の辺のうちの複数本を通る経路が単純閉路になるかどうかだけである。つまり，k 本の辺から何本かを選んで作れる 2^k とおりの組合せそれぞれが単純閉路かどうかを判定して，単純閉路になるものの数を数えればよい。

k 本の中から複数本を選んだ時にできる経路は，それぞれの基本サイクルの和である。この場合の和は exclusive or のことだと考えればよい。ある辺が偶数回含まれていれば除外し，奇数回なら残す。こうしてできたグラフが連結であり，かつ含まれる頂点の次数がすべて 2 なら単純閉路と判断してよい。

以上の記述の中に現れるグラフの操作は，$O(m+n)$ の計算量で可能なものばかりだが，これらを 2^k 回行うと，いろいろな計算を 10^{10} 回近く行うことになって時間が足りなくなる。これを避けるために，最初にグラフを縮約しておくのがよい。与えられたグラフの中に次数 2 の頂点があれば，その相手先同士を直接接続して，その頂点は削除する。次数 1 の頂点は単に削除すればよい。削除できる頂点をすべて削除した結果として得られるグラフは，元のグラフと同じトポロジーを持っていて，当然単純閉路の数も同じである。縮約したグラフの頂点数は $O(k)$ の程度なので，計算時間の問題もなくなる。

ここまで，アルゴリズムとして特に難しいものはないし，解法の設計もそれほど難しくはないと思われるが，何せプログラムとして実現しなければならない操作の数が多い。ざっと数え上げても，木の構成（深さ優先探索または union-find），それぞれの基本サイクルの確定（lowest common ancestor という問題の例になっている），複数本の辺から作られる単純閉路の和の計算，連結性の判定（深さ優先探索または union-find），次数 2 の確認，くらいの操作が必要である。グラフの縮約も，特に難しいわけではないが，結構面倒な操作である。コーディングの頑張りが必要な問題だと言えよう。この問題を解いたチーム数は 3 にとどまった。