審判長講評

審判長　鵜川 始陽

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	3	5	4	11	9	8	4	9	4	0	0
チーム数累計	1	4	9	13	24	33	41	45	54	58	58	58

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
正答チーム数	58	50	1	43	34	57	29	8	8	10	42
誤答チーム数	0	8	4	8	5	1	3	10	6	3	10
提出数	59	97	14	100	82	73	49	37	42	28	115

また、審判団が想定していた難易度の順に並べた問題番号と正答数を表 3 に示す。

表 3: 想定難易度順の問題番号と正答数

問題	A	B	F	D	G	K	I	E	H	J	C
正答数	58	50	57	43	29	42	8	34	8	10	1

今年は久しぶりに全ての問題に正答したチームがあった。1 位のチームは、コンテスト開始後 4 時間経過し、スコアボードがフリーズされた直後に問題 C に正解し、1 時間を残して全ての問題に正解した。このチームは全 11 問中 10 問目の問題は 2 時間 34 分で正解しており、スコアボードがフリーズされるのを待って最後の問題の解答を送信したのかもしれない。スコアボードフリーズの時点でも、10 問正解はこのチームだけであり、1 位のチームの強さが際立った。ただ、そのチームを除いても、続く 3 チームが最終的には 10 問正解しており、以降、多くの問題に正解したチームが比較的多く続き、表 1 に示すように理想的な正解数の分布となった。問題セット全体の難易度は適切だったと言える。

各問題の難易度分布についても、問題 C を除くと理想的だった。問題 C を除くと、正解数が少なかった I, H, J の 3 問の正解チーム数がそれぞれ 8, 8, 10 チームとほぼ等しく、同程度の難易度だったと分かる。この 3 問はジャンルの異なる問題であり、上位チームの勝敗を分ける難しい問題で、チームが得意とするジャンルが出題されたかどうかが順位に与える影響を小さくできた。また、正解チーム数の絶対値も例年よりやや多いが適切だった。

一方、問題 C は 1 位のチームしか正解しない非常に難しい問題だった。最も難しいレベルの問題が 1 問だけという状況は、チームが得意とするジャンルが出題されるかどうかが順位に影響を与える可能性という観点では、あまり良くない。しかし、今回のコンテストで仮に問題 C の代わりに問題 I, H, J 程度の難易度の問題を出題していたとすれば、1 位のチームはかなりの時間を余らせてしまっただろう。問題 C があった現状でも、スコアボードフリーズを待っていたとすれば、実際はど

れだけの時間を余らせたのかは分からないのだが。かといって、問題 C 程度の問題を複数問用意しても、解かれない問題が多くなってしまう。難しいところである。

前回コンテスト (2022 年 12 月開催の ICPC2022) からの比較的大きな変更としては、前回導入したインタラクティブ問題で、デバッグに使えるインタラクタを提供したことが挙げられる。インタラクティブ問題は選手が提出したプログラムが、審判が用意したプログラムであるインタラクタと通信して、何回かデータを送受信することで正解を見つけるような問題である。このデバッグでは、実際に実行してみて、どのようなデータが送受信されているのかを確認するのが効果的だが、そのためにはインタラクタが必要になる。前回コンテストでは、インタラクタも必要に応じて選手が作ることになっていたが、今回はインタラクタの実行ファイルをあらかじめ競技用のコンピュータにインストールしておく形で提供した。ソースコードではなく実行ファイルを提供したのは、インタラクタのソースコードが C++だったので、C++に不得手なチームが不利になることを避けるためである。今回のコンテストでは、選手は C++の他に C, Java, Python, Kotlin を使うことができ、このうち 1 言語のみに精通していることも考えられる。

以下では各問題について、解法を中心に簡単に解説する。なお、問題解説には問題の原案を提案した審判と、主担当として作問に中心的にかかわった審判の名前を記すが、それ以外に問題のチェックや難易度調整、問題文の校正、想定誤答プログラムの作成などで多くの審判がかかわっており、どの問題も審判全員で作りあげた問題である。

問題 A: Yokohama Phenomena

原案: 稲葉 一浩　　主担当: 鵜川 始陽

アルファベット 1 文字が書かれたマスが縦横に並んだ盤面が与えられ、その中に隠れた特定の文字列を数える問題である。この問題では、探す文字列を開催地にちなんで「YOKOHAMA」にしている。つまり、Y のマスから始めて、その縦横に隣接する O のマス、さらにその縦横に隣接する K のマスと順にたどり、最後に A のマスに到達できるような経路がいくつあるか数える問題である。この経路を Yokohama trace と呼んでいる。Yokohama trace には同じマスが複数回含まれてもよい。YOKOHAMA には O と A が 2 回ずつ現れるので、これらには同じマスを使ってもよい。

想定解法は動的計画法だが、動的計画法を使わず、どのように探索をしても十分に間に合うように盤面を小さくしている。例えば、それぞれのマスを開始点として深さ優先探索で 8 文字の文字列を列挙し、それが YOKOHAMA かどうかを判定するという素朴なアルゴリズムでも正解できる。

動的計画法を使う場合は、それぞれのマスについて、そのマスで終わる Yokohama trace の i 文字の接頭辞がいくつあるかを $i = 1$ から順に計算する。$i > 1$ の場合、YOKOHAMA の i 文字目（例えば $i = 3$ では K）の書かれたマス c で終わる i 文字の接頭辞の数は、c に隣接する全てのマスの、そのマスで終わる $i - 1$ 文字の接頭辞の数の合計になる。YOKOHAMA の i 文字目が書かれていないマスについては、0 である。$i = 8$ まで求めれば、それぞれのマスについて、そのマスで終わる完全な Yokohama trace の数を求めたことになるので、これを合計すればよい。

問題 B: Rank Promotion

原案: 稲葉 一浩　　主担当: 稲葉 一浩

Y と N からなる長さ n の文字列を先頭から順に見ていき、長さ c 以上の連続した部分列で Y の割合が有理数 p/q 以上であればそこで「昇段」するとしたとき、段位が最終的にいくつまであがるか求めよ、という問題である。それぞれの $1 \leq i \leq n$ に対して i 文字目で昇段するかどうか検査するの

だが、i 文字目から文字列の先頭に向かって昇段の条件を満たすか順に判定するという $O(n^2)$ 時間のアルゴリズムは、入力の制約が $n \leq 5 \times 10^5$ なので間に合わない。

入力の制約で c が n に比べてかなり小さく設定されていることを活用するのが、想定された最も簡単な解法になる。長さ $2c$ 以上の列で Y の割合が p/q 以上ならば、その列を半分に分けた前半と後半のどちらかは Y の割合が十分大きいし、長さも c 以上である。この性質を考慮すると、昇段するか否かの検査は長さ $2c$ の部分列まで調べれば事足りるため、全通り調べても $O(nc)$ の計算量で答えを求めることができる。

実際には c の大小によらない $O(n)$ 時間の解法も存在し、多くのチームがこちらで正答していた。この問題は、Y を 1、N を 0 とみなすと、平均値が p/q 以上の部分列を探す問題と見ることができる。さらに、各要素からあらかじめ p/q を引いておくと、総和が 0 以上の部分列を探す問題と言い換えることができる。先頭からの累積和を持っておけば、総和が最大になる区間を見つけるのはたやすい。

問題 C: Ferris Wheel

原案: 隈部 壮　　主担当: 隈部 壮

円周上に等間隔に配置された $2n$ 個の点を条件を満たすように k 色で塗り分ける方法を数え上げる問題である。この問題の条件は、塗り分けた後に各点がちょうど 1 本の線分の端点になるように同じ色の点同士を結ぶ n 本の線分を引くとき、それらが交差しない引き方が存在することである。この問題ではさらに難しいことが要求されており、円を回転させて一致する塗り分け方は同一視して一つと数えなければならない。つまり、塗り分け方の回転による商を数える必要がある。

この問題は非常に難しい問題として出題され、1 チームにしか正解されることはなかった。ここでは、回転させて一致する塗り分け方も区別して数え上げるところまでを説明する。回転による商を求める方法は、閉会式で使った問題解説スライドをを参照されたい。

条件を満たす塗り分け方には、同じ色に塗る隣接する 2 点 A, B が存在し、その 2 点を取り除いてもやはり条件を満たすという性質がある。なぜなら、点 X, Y があり、A–X, B–Y に線分が引けるとき、この 2 本の線分の代わりに A–B, X–Y に線分を引いても他の線分と交差することはないからである。

円周を適当な点で切り開いて、点列を作る。上述の性質を利用すると、条件を満たす塗り分けかどうかは点のスタックを使って、点列を左から順に 1 回走査することで判定できる。最初はスタックは空である。列の左から点を一つずつ取って、その点と同じ色がスタックトップにあればスタックトップを取り除く。そうでなければ、その点をスタックに積む。最終的にスタックが空なら、条件を満たす。そうでないなら、満たさない。

スタックに積む操作を開き括弧 '(' で、取り除く操作を ')' で表すと見通しがよくなる。条件を満たすとき、この括弧列は全体でバランスがとれている。この括弧列を、バランスがとれた部分括弧列に分解し、その連接と考える。それ以上分解できなくなったとき、各部分括弧列を「原子括弧列」と呼ぼう。ある特定の括弧列が i 個の原子括弧列に分解されるとき、その括弧列に対応する塗り分け方は $k^i(k-1)^{n-i}$ 通りある。原子括弧列の先頭は k 色自由に選べ、それ以外の開き括弧は先頭の色を除いた中から選べるからである。これを用いて、i 個の原子括弧列に分解される括弧列の数を x_i とすると、求める塗り分け方は

$$\sum_{i=1}^{n} x_i k^i (k-1)^{n-i}$$

と表せる。

では x_i を求めよう。これはカタラン数の要領で二項係数で書ける。平面座標の $(0,0)$ から右か上に 1 ずつ進みながら $x \geq y$ である領域だけを通って (s,t) に至る経路の集合を $\mathrm{C}(s,t)$ とする。このとき、$x_i = |\mathrm{C}(n-1, n-1-i)|$ となることが証明できる。$y = x$ 上の点を i 回通る $\mathrm{C}(a,b)$ の経路は、$y = x$ 上の点を $i-1$ 回通る $\mathrm{C}(a,b-1)$ の経路に対応させることができるからである。以上より、回転させて一致する塗り分けを区別する場合は $O(n)$ 時間で計算できる。

問題 D: Nested Repetition Compression

原案: 江本 健斗　　主担当: 近山 隆

繰返しに着目した単純な文字列圧縮手法における最適な圧縮を求める問題である。同じ部分文字列の繰返しについて、繰返し回数の後に繰り返す対象の部分文字列を左右括弧で囲んで示すもので、たとえば `ababab` は `3(ab)` と圧縮することができる。繰返し回数は 1 桁の数字だけで示すので、最大 9 回なのだが、入れ子にすることによって、たとえば `aaaaaaaaaaaa` は `3(4(a))` などとすることができる。

この圧縮手法では部分列の最適圧縮は他の部分の圧縮とは独立であり、その部分列のみによって決まる。最適圧縮の方法は以下の 3 種のいずれかである。

A 元のまま、つまり圧縮しない

B 最適圧縮した部分列二つを連接したもの

C 最適圧縮した部分列の繰返し

所与の文字列全体の最適圧縮法を求めるためには、短い部分列の最適圧縮から始めて、次第に長い部分列の最適圧縮法を決めていけばよい。長さ 4 以内の部分列については繰返しによる圧縮は効果がないので、そのままとする (方法 A) のが最適である。ある長さ以下の部分列の最適圧縮法を既知としたとき、より長い部分列について以下の方法による圧縮を比較し、短い方を採用する。

- 方法 B による圧縮。二つの空でない部分列への分割すべてについて、分割した部分列二つの最適圧縮長の和が最小であるものを採用する。
- 方法 C による圧縮。方法 C による圧縮には、対象部分列全体の長さ l が繰返し回数 r の整数倍である必要があり、候補となる r は l の 9 以下の約数に限られる。そのような r について実際に繰り返されているかどうかを判定する。繰返しになっているものの中で、繰返し部分の最適圧縮長が最小であるものを採用する。対象部分列全体の最適圧縮長は、繰返し部分の最適圧縮長に繰返し数指定と左右の括弧に要する 3 を加えたものになる。

方法 B, C による圧縮方法は、分割位置や繰返し回数の違いにより一般に複数通り存在する。枝刈りは簡単ではないが、すべて試しても計算量は文字列長 n に対して $O(n^3)$ にしかならず、この問題の制約である $n \leq 200$ には十分である。方法 C による圧縮が可能な場合でも、より優れている方法 B による圧縮が存在することがある。たとえば `a` が 55 個並ぶ列は `6(9(a))` と `a` の連接 `6(9(a))a` で 8 文字に圧縮できるが、最外側を繰返しにするのでは `5(9(a)aa)` などの 9 文字にしかできない。

問題 E: Chaya

原案: 隈部 壮　　主担当: 城下 慎也

1 から n までの整数をシャッフルした列の中で、所与の制約を全て満たす列の数を求める問題である。制約は列中の三つの整数が現れる順序を定める。例えば、制約 $(1, 2, 4)$ を満たす列では、列の先頭から末尾に向かって 1, 2, 4 がこの順に現れるか、その逆順である 4, 2, 1 の順に現れる。1,2,4 の間に他の整数があってもよい。したがって、$1, 2, 3, 4, \ldots$ はこの制約を満たす。

想定解法は、整数の集合に対して、その整数を全て使って作れる制約に違反しない列の数を、小さい集合から順に求める動的計画法である。ある整数集合の要素を全て使って作られる列の末尾には、その集合の要素のどれかが現れる。したがって、ある整数集合の要素を全て使って作られる列は、その集合から 1 要素取り除いた集合で作った列の末尾に、取り除いた要素を追加した列である。ただし、制約 (a, b, c) があるとき、次のような集合 S から作られる列は全て制約に違反する。

(A) $b \in S$ かつ $a, c \notin S$

(B) $b \notin S$ かつ $a, c \in S$

したがって、このような集合 S については制約に違反しない列の数を 0 とし、それ以外の集合 S については、S から 1 要素取り除いた集合それぞれの制約に違反しない列の数の合計を S の制約に違反しない列の数とすることで正解にたどりつける。しかし、条件 (A), (B) に違反するかどうかを個々の集合ごとに計算すると、制約の数を m としたとき全体の計算量は $O(m2^n)$ になり、m が大きいこの問題では間に合わない。

高速化のために、いずれかの制約に違反する集合かどうかを前処理で求めておく。このとき、制約の 2 番目の整数 b が等しい制約をまとめて、その制約のいずれかに違反する集合を一度に求める。これは、真偽値を値とする $O(n \times 2^n)$ 時間の動的計画法で求められる。この動的計画法は 1 ビットしか使わず、b によらず同じ方法で計算できる。そのため、異なる b に異なるビットを与えて並列に計算できる。$n \leq 24$ であるこの問題では、$O(n2^n)$ 時間で全ての b に対する前処理を終えることができる。この結果を使うことで $O(1)$ 時間で制約に違反する集合かどうか判定できるため、全体の計算量は $O(n2^n)$ に抑えられる。

前処理の具体的なアルゴリズムを説明する。ここでは条件 (A) に違反する集合を求める方法だけ説明する。所与の制約のうち、2 番目の整数が b である i 番目の制約を (a_i, b, c_i) とする。いずれかの i について、a_i と c_i を除く全ての整数の集合は直ちに条件 (A) に違反する。集合 S について、このような集合であれば真、そうでなければ偽と定義した関数を $f(S)$ とする。b を含む集合 S について、いくつかの要素を追加することで f を真にすることができれば、つまり、$f(S \cup T)$ が真となる T が存在するなら、S は条件 (A) に違反するといえる。これを動的計画法で求める。

$k-1$ 以下の整数を適切に加えることで f を真にする集合が全て既知とする。これを全て集めた集合族を G_{k-1} とする。このとき、以下のいずれかの条件を満たす集合 S は k 以下の整数を適切に加えることで f を真にする。

- $S \in G_{k-1}$
- $(S \cup \{k\}) \in G_{k-1}$

そのような S が G_k の要素となる。なお、G_0 は $f(S)$ が真となる S を集めた集合とする。G_n が条件 (A) に違反する集合を集めた集合である。

問題 F: Color Inversion on a Huge Chessboard

原案: 森田 晃平　　主担当: 森田 晃平

白黒に塗られたボードで、色の白黒を指示に従って反転させた結果を計算する問題である。チェスボードのような市松模様を初期状態とし、指定された行や列のマスの色を全て反転させる「反転操作」を何回か行う。反転操作を 1 回行うごとに、同じ色の「連結成分」がいくつ存在するかを答える。ここで「連結成分」とは、縦か横に隣接した同じ色のマスを連結するマスと定義し、連結するマス同士をできる限り集めたマスの集合を言う。初期状態では全てのマスが隣接するマスと異なる色なので、$N \times N$ のマスがあるボードでは連結成分の数は N^2 である。また、全てのマスが同じ色になったとすれば、連結成分の数は 1 である。

素朴にシミュレーションしても正解は得られるが、1 回の反転操作で N マス反転させる必要があり、さらに、連結成分を数えるたびに $O(N^2)$ 時間かかる。Q 回の反転操作と連結成分を数える計算量は $O(QN^2)$ であり、$Q \leq 5 \times 10^5$, $N \leq 5 \times 10^5$ であるこの問題では、とても間に合わない。

そこで、縦横の色の間隔が 1 マスとは限らず、2 マス以上も許す一般化した市松模様を考える。一般化市松模様とは、何本かの縦線と横線で区切られた各長方形の領域が、辺を挟んで隣接する領域とは異なる色で塗られた模様とする。この問題では、どのように反転操作を行っても、その結果は一般化市松模様になる。これに気づけば、行と列を独立に 1 行目や 1 列目の連結成分の個数を求める一次元の問題として考え、それを掛け合わせることで全体の連結成分の数が求められると分かる。これによって、反転操作は定数時間、連結成分を数える計算は $O(N)$ 時間になり、全体の計算量が $O(QN)$ になる。これでもまだ制限時間内には計算が終わらない。

反転操作ごとに連結成分を数えるのではなく、反転操作によって起こる連結成分の増減を計算すれば、反転操作ごとに定数時間で連結成分の数を求められる。具体的には、反転したマスの両側それぞれのマスについて、反転前に同色であれば反転によって連結成分は 1 増え、そうでなければ 1 減る。ここまですれば、1 行目と 1 列目の状態を保持する配列を初期化する時間を含めても、計算量は $O(Q + N)$ まで下げられ、時間内に正解を求められる。

問題 G: Fortune Telling

原案: 楠本 充　　主担当: 楠本 充

一列に並んだ n 枚のカードがある。ここに、左から 6 枚のカード（カードが 6 枚未満のときは全てのカード）の内からランダムに 1 枚の「起点カード」を選び、そこから 5 枚おきに 1 枚ずつカードを取り除くという操作を繰り返し行うと、必ずカードが 1 枚だけ残った状態になる。この問題では、n 枚のカードそれぞれについて、この 1 枚になる確率を求める。この 1 枚を使って運命を予言する占い師というストーリで出題したので、この 1 枚を「運命のカード」と呼ぶことにしよう。

まず、素朴な動的計画法の解を考える。残りのカードの枚数が n' $(n' > 1)$ のときの各カードが運命のカードとなる確率は、残りのカードの枚数が $n' - 1$ 枚以下のときの各カードが運命のカードとなる確率から計算できる。1 回の操作での取り除き方は起点カードの選び方の 6 通りしかないので、その全ての場合について、操作を行った後に残るカードの枚数と、各カードが左から何番目になるか、あるいは取り除かれるかを考えればよい。しかし、この方法は $O(n^2)$ の計算量になり、$n \leq 3 \times 10^5$ であるこの問題では許容されない。

カードの枚数の変化をもう少し精確に考えることで、計算量を抑えることができる。もし n が 6 の倍数であれば、1 回の操作を行った後に残る枚数は $5n/6$ 枚である。n が 6 の倍数でない場合は、起点カードの選び方により $\lfloor 5n/6 \rfloor$ 枚か $\lfloor 5n/6 \rfloor + 1$ 枚になる。いずれにせよ、残りのカードが n 枚

より少なく $\lfloor 5n/6 \rfloor + 1$ 枚より多い状況にはならないので、その場合の各カードが運命のカードとなる確率は計算する必要がない。一般に、s 回の操作を行った後はカードの枚数は $(5/6)^s n$ 枚以上 $(5/6)^s n + (s-1)$ 枚以下になっている。この場合だけ、各カードが運命のカードとなる確率を計算すればよい。当然だが、運命のカードとなる確率の計算は残っているカードに対してのみ行う。これらの考察から必要な計算はかなり少ないことが推測できる。実際、以降で説明する導出により、必要な計算のみを重複なく行うことで計算量を $O(n)$ まで抑えられる。

以降では、改良された方法の計算量を説明する。残り枚数が 6 枚に達するまでに要する操作回数の上限を $f(n)$ とすると、確率の計算の回数の上限は

$$\sum_{s=0}^{f(n)} \sum_{i=0}^{s} \left[\left(\frac{5}{6} \right)^s n + i \right]$$

であり、整理すると

$$\sum_{s=0}^{f(n)} \left[\left(\frac{5}{6} \right)^s (s+1) n \right] + \sum_{s=0}^{f(n)} \frac{s(s+1)}{2}$$

になる。$f(n)$ はせいぜい $\dfrac{\log n}{\log(6/5)}$ 程度であるから、2 項目は $O((\log n)^3)$ 程度である。これは 1 項目に比べれば小さいから、1 項目の上界だけ求めればよい。総和を $f(n)$ までではなく、無限までとり

$$\sum_{s=0}^{\infty} \left(\frac{5}{6} \right)^s (s+1) n$$

を考えることにする。この和は $36n$ に収束する。よって、確率の計算の回数は $36n$ 回以下であり、計算量は $O(n)$ になる。ただし、ビッグ・オーには係数として $36 \cdot 6 = 216$ が隠れていることに注意。一般に起点カードを A 枚のうちから選ぶとき、計算時間は $O(A^3 n)$ とも表せる。

問題 H: Task Assignment to Two Employees

原案: 岩田 陽一　　主担当: 岩田 陽一

n 個の仕事を最適な順序で行うことで、仕事によって得られる利益の総和を最大化するスケジューリング問題である。得られる利益は仕事によって異なる「利益係数」に比例する。また、仕事をすることで成長して「能力値」が向上し、別の仕事をしたときの利益が増加する。能力値の増加量も仕事によって異なる。より具体的には、能力値の初期値が p_0 で、仕事 i をすると能力値が s_i 増加する。仕事 i をしたときの利益は、能力値 p と利益係数 v_i の積 pv_i である。ここまでであれば、素朴な貪欲法の問題だが、この問題はもっと難しい。この問題では 1 人が全ての仕事を行うのではなく、仕事を行う社員が 2 人いて、利益係数や能力値の増加量は 2 人の社員で異なる。2 人に仕事を割り振ったうえで、仕事の順序を決める必要がある。

まずは素朴な貪欲法で解ける、社員数が 1 人の場合に最適な仕事の順序を決める問題から考えてみよう。仕事 i をした時に得られる利益は、初期値の能力値 p_0 による「基礎利益」$p_0 v_i$ と、先に他の仕事をして成長したことによる利益の増分である「ボーナス」の和と整理できる。基礎利益は仕事を行う順序によらず一定である。一方、ボーナスは仕事の順序により異なる。仕事 j を仕事 i より先に行うことによりボーナスは $s_j v_i$ 増える。代わりに、仕事 i を仕事 j より先に行えば $s_i v_j$ 増える。したがって、s_k / v_k の降順で仕事を行うことで利益を最大化できる。また、仕事 i と j を適切な順序で行うことによるボーナス増加量は $\max(s_i v_j, s_j v_i)$ である。

社員が A, B の 2 人の場合、仮に 2 人ともに全ての仕事をさせられれば、社員は大変だが 2 人合わせた利益は大きく、1 人ずつ個別に求めた利益の最大値の和になる。この実現不可能な利益を基準にして考えよう。実際には、一つの仕事を 2 人にさせることはできず、どちらかに割り振らなければならない。一方の社員に割り振ると、他方の社員からの利益は得られず、利益が基準から減ることになる。この減少分をコストと考える。仕事 i を社員 A に割り振ったことによる利益の減少分には、

- 仕事 i を B に割り振っていれば得られた基礎利益 $p_0 v_{(B,i)}$
- B に割り振った各仕事 j について、i を B に割り振っていれば得られたボーナス $\max(s_{(B,i)}v_{(B,j)}, s_{(B,j)}v_{(B,i)})$

がある。ここで、変数 s,v の添字には社員の識別子も追加している。

最小カットに帰着して、利益の減少分を最小化する。始点 a、終点 b、各仕事 i に対応する頂点 i の合計 $n+2$ 頂点のグラフを考え、a–b カットの a 側の頂点を社員 A、b 側の頂点を社員 B に割り振る仕事に対応させることにする。辺の容量は以下のように設定する。

- 仕事 i を社員 B に割り振る際の基礎利益減少分 $p_0 v_{(A,i)}$ を辺 $a \to i$ の容量に加える
- 仕事 i を社員 A に割り振る際の基礎利益減少分 $p_0 v_{(B,i)}$ を辺 $i \to b$ の容量に加える
- 頂点 i, j の少なくとも一方を社員 B に割り振ったときのボーナス減少分 $\max(s_{(A,i)}v_{(A,j)}, s_{(A,j)}v_{(A,i)})$ を辺 $a \to i$ と辺 $i \to j$ の容量に加える
- 頂点 i, j の少なくとも一方を社員 A に割り振ったときのボーナス減少分 $\max(s_{(B,i)}v_{(B,j)}, s_{(B,j)}v_{(B,i)})$ を辺 $i \to j$ と辺 $j \to b$ の容量に加える

$n+2$ 点のグラフの最小カットは Dinic 法を用いることで $O(n^4)$ 時間で計算することができる。

問題 I: Liquid Distribution

原案: 佐藤 遼太郎　　主担当: 佐藤 遼太郎

2 種類の液体 A, B が異なる比で混合された混合液がいくつかあり、これらを使って指定されたいくつかの比の混合液が全て作れるかどうかを判定する問題である。初期状態と、混合液を全て作った状態である最終状態での各混合液の量も問題で与えられる。なお、初期状態と最終状態で液体 A や B の総量は変化しない。

この問題では、初期状態や、液体同士の混合操作をした後のそれぞれの状態について、どのような混合比の液体がいくら存在しているかをうまく表現すると見通しがよくなる。ここでは、状態 s を液体 A の量から液体 B の量への関数 f_s で表現しよう。$f_s(x)$ を、状態 s で存在する混合液のうち、B の比率が小さい混合液から順に A の総量が x になるまで集めたとき、そこに含まれる B の総量と定義する。f_s は区分線形な凸関数で、一つの線分が一つの混合液に対応する。このような状態を表す関数を定義すると、状態 s から液体同士の混合操作で状態 t に遷移できることと、任意の x で $f_s(x) \le f_t(x)$ が成立することが同値であると証明できる。

したがって、本問題では初期状態 s_0 と最終状態 s_f について $f_{s_0}(x) \le f_{s_f}(x)$ が恒等的に成立するかどうかを判定すればよい。初期状態では高々 n 種類の混合液しか存在しないため $f_{s_0}(x)$ が区分線形関数であることに着目すると、特に $f_{s_0}(x)$ の傾きが不連続になる点のみ確認すればよい。ここまで分かれば、この問題の制約では素朴に $O(nm)$ で交差を判定しても間に合う。ここで m は最終状態の混合液の種類数である。より高速な判定をするには、初期状態と最終状態それぞれで x 座標でソートされた不連続点のリストを作り、その先頭で $f_{s_0}(x) \le f_{s_f}(x)$ を判定したあと x 座標が小さい方を取り除くという処理を繰り返せばよい。この方法は $O(n \log n + m \log m)$ で動作する。

問題 J: Do It Yourself?

原案: 山口 勇太郎　　主担当: 山口 勇太郎

n 頂点の根付き木の各頂点にタスクが一つずつ配置されている。タスクはその頂点で遂行してもよいし、根に近い頂点に移動させて遂行してもよい。各頂点 v には係数 f_v が定められており、頂点 v で x_v 個のタスクを遂行すると、$f_v x_v^2$ のコストがかかる。このとき、コストの総和を最小化する問題である。

少し考えてみると、頂点 v で i 個目のタスクを遂行するときのコストの増分は $(2i-1)f_v$ とわかる。この観察から、各頂点 v が重み $f_v, 3f_v, 5f_v, \ldots, (2n-1)f_v$ の要素を一つずつ持っているとき、どの部分木からもそのサイズ以下の要素しか選ばないという制約のもとで、木全体で n 個の要素を選び、その重みの総和を最小化する問題と言い換えられる。この問題は「暫定解を空集合から始めて、重みの小さい順に要素を見ていき、制約に違反しないならば追加する」という素朴な貪欲法で解くことができる。

貪欲法の実装と計算量について考えよう。各頂点では常に重みの昇順に要素を見るので、各頂点についてまだ見ていない要素の中で最小重みのものだけを見ればよい。優先度付きキューで管理することにすれば、次に見るべき要素は毎回 $\mathrm{O}(\log n)$ 時間で取得できる。その要素を暫定解に追加できるかどうかは、木の重軽分解 (Heavy-Light Decomposition) を利用することで、次のようにして $\mathrm{O}((\log n)^2)$ 時間で判定できる。まず、各頂点の余裕（部分木内であといくつ要素を選べるか）を管理しておく。余裕の初期値は部分木のサイズである。

- 頂点 v の要素を追加できるかどうかを判定するには、v から根に遡るパス上の余裕を調べる。それが全て正であれば追加できる。
- 頂点 v の要素を追加したときは、v から根に遡るパス上の余裕を 1 ずつ減らす。

これらは単純な区間最小値取得・区間加算クエリであり、遅延評価セグメント木を各 Heavy パス上に用意しておくことで、前述の計算時間を達成できる。

最後に重要な観察として、一度でも要素を追加できなくなった頂点については、以降見る必要は無い。したがって、全体で見るべき要素の総数は高々 $2n$ 個である。以上より、全体の計算量は $\mathrm{O}(n(\log n)^2)$ であり、適切に実装することで余裕を持って制限時間 10 秒に間に合わせることができる。

問題 K: Probing the Disk

原案: 松崎 公紀　　主担当: 松崎 公紀、楠本 充

隠された円の座標と半径を当てるインタラクティブ問題である。解答プログラムは線分を指定したクエリを送る。これに対して、インタラクタは線分のうち円の内部に入っている部分の長さを返す。クエリは 1024 回まで送ることができる。なお、円の半径は 100 以上あり、円は $(0,0)$ と $(10^5, 10^5)$ を対角とする正方形からはみ出すことはない。

この問題では、いかにして円の内部にある点を一つ見つけるかが鍵である。この点は円の中心である必要はない。円の内部の点が一つ見つかれば、さらに 4 回のクエリで中心と半径を求められる。

円の内部の点は次のようにして探す。まず Y 軸に平行な 999 本の線分 $(100i, 0)$–$(100i, 10^5)$, $1 \leq i \leq 999$ を指定したクエリを順に送る。円の半径が 100 以上なので、100 間隔で線分を引くと少なくとも 2 本は円に重なる。このうち少なくとも 1 本は、円を切り取る弦の長さが $100\sqrt{3}$ 以上である。そのような線分を 1 本選ぶ。

次に、その線分の上側の端点を下げていき、円の内部の区間が短くなり始める座標を探す。これは二分探索により 11 回で、$100\sqrt{3}$ 以上の長さの弦の中に収まるように求められる。

最後に中心の座標を求める。前述の手順で求めた円の内部の点を P とする。P を端点とする各軸に平行な十分に長い線分を指定したクエリを P の両方向に対して行うことで、合計 4 回のクエリで P を通り各軸に平行な弦の両端の座標が求まる。中心はこの垂直二等分線上にある。