How to Avoid Disqualification in 75 Easy Steps (avoid)

by Tobias Lenz

Throughout, we denote the positions of the chairmen by $1 \le a \le b \le 1\,000$, and $N = 1\,000$ is the number of positions.

Subtask 1. $R = 10$, $H = 1$, and both chairmen are located at the same position.

This subtask is easiest to solve if we invert the problem: instead of determining for each robot which positions it scouts, we will determine for each position p the set of robots $R_p \subseteq \{1, \dots, 10\}$ scouting it.

If both chairmen are located at position a, a robot r will detect at least one chairman if and only if $r \in R_a$. Hence, the results of all 10 robots are equivalent to knowing the set R_a. To determine a from this, we therefore need to ensure that the sets R_p are all different.

Fortunately, $2^{10} \ge 1\,000$, and so we can assign a unique subset $R_p \subseteq \{1, \dots, 10\}$ of the robots to each position p. A simple way to do this is to use the binary representation of p—then, the results of the robots spell out the position of the chairmen in binary, solving this subtask.

Subtask 2. $R = H = 20$

This subtask can be solved with two binary searches. With the first binary search, we want to determine the smallest position a of one of the chairmen. For this, we will always keep an interval $[\ell, r]$ that contains a, starting with $\ell = 1$ and $r = 1\,000$. Then, in each step, we pick $m = \lfloor (\ell + r)/2 \rfloor$, send a robot to the positions $\{\ell, \ell + 1, \dots, m\}$, and wait for its result. If this robot detects a chairman, we know that the interval $[\ell, m]$ must contain a, and so we set $r = m$. Otherwise, a must be contained in $[m + 1, r]$, and so we assign $\ell = m + 1$. Once we have $\ell = r$, which happens after at most $\lceil \log_2 1\,000 \rceil = 10$ steps (and 10 robots), we have determined a as required.

With the second binary search, we can simply determine the largest position b of one of the chairmen analogously, again with 10 robots. Therefore, we can determine the positions of both chairmen with 20 robots in total.

Note that we could also execute the above two binary searches simultaneously which would reduce the total time required from 20 hours to 10 hours. It is easy to prove by counting that at least 19 robots are necessary, but the Scientific Committee does not know whether there exists a solution that uses at most 19 robots.

Subtask 3. $R = 30$, $H = 2$

To find the positions of the two chairmen in 2 hours, recall our approach to Subtask 1. To all positions p, we had assigned a distinct set $R_p \subseteq \{1, \dots, 10\}$ of robots, namely the binary representation of p. We will denote the i-th bit of p by p_i, and so $i \in R_p$ if and only if $p_i = 1$.

Now, in this subtask, we will send 10 robots as described by the sets R_p, and we send an additional 10 robots for the complements R_p^c, all during the first hour. Based on the results of these robots, we will know for all i that either $a_i = b_i$, in which case we will also the value of a_i and b_i, or $a_i \ne b_i$.

Now, in the second hour, we need to reconstruct the remaining information. If i and j are bits such that both $a_i \ne b_i$ and $a_j \ne b_j$, we need to determine whether $a_i = a_j$, and equivalently $b_i = b_j$, or whether $a_i \ne a_j$, and equivalently $b_i \ne b_j$.

To do this, let i be a bit such that $a_i \ne b_i$. Now, for every j that is different between a and b, we will simultaneously send a robot to all those positions p where $p_i = p_j$. If this robot detects a chairman,

we know that $a_i = a_j$ and also $b_i = b_j$, and otherwise we know that $a_i \neq a_j$ and also $b_i \neq b_j$. This lets us reconstruct a and b based on the i-th bit. This strategy uses at most 29 robots in 2 hours.

Subtask 4. $R = 75, H = 1$

This is the only output-only subtask of this CEOI.*

Constructive solutions. We begin by describing some constructive solutions.

- Firstly, we can adapt the solution to Subtask 3. Instead of using the second phase of that strategy, we can, for each pair of bits i and j, already send a robot in the first phase to all positions p with $p_i = p_j$. This uses a total of $20 + \binom{10}{2} =$ **65** robots. Then, we already know the results for all robots that we would have sent out during the second phase, and so we can immediately reconstruct a and b after just one hour.
- To get a solution with fewer robots, we can use a base other than 2. We will describe a solution with base 3, where we again denote the i-th digit of a position p by p_i. Then, for every digit i and every $d \in \{0, 1, 2\}$, we send a robot to all positions p with $p_i = d$. Since there are $\lceil \log_3 1\,000 \rceil = 7$ digits, this uses 21 robots, and it will tell us for every digit i whether $a_i = b_i$, in which case we will also know the values of a_i and b_i, or whether $a_i \neq b_i$. In this second case we know which two digits a_i and b_i are, so we know two digits $d_i^1, d_i^2 \in \{0, 1, 2\}$ such that $\{a_i, b_i\} = \{d_i^1, d_i^2\}$, but we do not know whether $a_i = d_i^1$ or $a_i = d_i^2$, and similarly for b_i.

  Thus, for every pair of digits i and j we also need to send robots to make sure that if $d_i^1 \neq d_i^2$ and $d_j^1 \neq d_j^2$, we know whether d_i^1 and d_j^1 are both digits of a or both digits of b, or whether this is not the case. It turns out that it is again sufficient for this to send a robot to all positions p with $p_i = p_j$. Indeed, one of the digits d_i^1 or d_i^2 must have the same value as one of the digits d_j^1 or d_j^2, and so the result of this robot tells us whether these two digits are both digits of a or both digits of b. This uses an additional $\binom{7}{2} = 21$ robots, for a total of **42** robots.

  There also exists a solution with base 4 that uses **40** robots.
- Finally, it is possible to come up with a scheme that takes a solution for N positions and R robots and produces a scheme for N^2 positions and $3R$ robots. For this, let R_p be the sets of robots assigned to each position p in the scheme for N positions. We then represent the N^2 positions as pairs (p, q) with $1 \leq p, q \leq N$, and for (p, q) we send R robots each according to R_p, R_q, and $R_{p+q \pmod N}$. It can be checked that this allows you to uniquely determine the positions of the chairmen among all N^2 positions. Starting with a solution with $N = R = 6$ and applying this twice leads to a solution with **54** robots.

A general approach. Assuming that we have already determined where each robot is sent to, we can check efficiently whether this is a valid scheme that always determines the positions of the chairmen with certainty. To do so, consider again the sets R_p of robots assigned to each position p. A robot r will detect a chairman if and only if $r \in R_a$ or $r \in R_b$, or equivalently $r \in R_a \cup R_b$. This means that the results of the robots are equivalent to knowing $R_a \cup R_b$, and so we can determine a and b uniquely if and only if $R_a \cup R_b$ is different from all other such unions. This can be checked by representing each R_p in binary as an **__int128_t** so that $R_p \cup R_q$ is simply the bitwise or of the two numbers, running in time $O(N^2)$ when using a hash map to check for collisions.†

Reconstructing a and b from the answers then simply works in exactly the same way. We can iterate through all p and q and check whether the $R_p \cup R_q$ matches the results of all robots.

* Wait, what? Keep on reading to understand why...

† You might have noticed that no heuristic managed to get more points than intended in the last subtask...

All of our full solutions are based on generating the sets R_p *locally* on our own computer, potentially taking minutes or even hours, until we have a valid scheme. Such a scheme can then be encoded as a list of numbers in the submission, and we use the above strategy to determine a and b from the results of the robots.

Generating robots. Let us first describe some strategies that try to generate the plan robot by robot.

- Fix $0 < p < 1$ and send any robot ϱ to any position x with probability p, independently of any other choices. We claim that for sufficiently many robots this will yield a valid scouting plan with positive probability. To prove this, let us say that an assignment for a single robot *distinguishes* two sets $L := \{x, y\}$ and $L' := \{x', y'\}$ of possible chairman positions if the robot returns 1 for *precisely* one of these two sets. Note that this happens with a positive probability q (maximized for $p \approx \frac{1}{3}$). Moreover, for any other robot ϱ', the events that ϱ distinguishes the two given L, L' or that ϱ' distinguishes them are independent; in particular, for r robots the probability that we do not distinguish them is $(1 - q)^r$. Summing over all L, L' this yields an upper bound of $N^4(1 - q)^r$ for the probability that we do *not* find a correct scouting plan. For $r \to \infty$ robots, this failure probability converges to 0. In practice, this approach suffices to find solutions with **50** robots in reasonable time.
- The previous strategy tends to generate plans that almost work, i.e. there are few pairs (L, L') which we cannot separate. This allows us to get better results by 'supersampling': we first use the above strategy for $N > 1\,000$ positions ($N \approx 2\,000$ seems to be the sweet spot); as long as there are $\{x, y\}$ and $\{x', y'\}$ which we can't distinguish, we randomly throw away one of x, y, x', y'. This yields solutions with around **35** to **40** robots, depending on how much computing time you are willing to invest or how clever you are in selecting the positions to discard.
- Instead of all the fancy randomness, we can try to be more systematic in assigning positions to the robots, somewhat akin to binary search.

  For this, let us first consider the case of a single robot. Ideally, the two possible answers `0` and `1` should occur for around the same number of possible chairman positions. Doing the math, we see that for this we should send the robot to a fraction of

  $$\alpha := 1 - \frac{\sqrt{2}}{2} = 0.29289\ldots$$

  of the positions (note that this is *not* the optimal probability p from the random approach!).

  Similarly, for two robots, the outcomes `00`, `01`, `10`, and `11` should occur for around the same number of possible chairman positions. To achieve this, we can send the second robot to a fraction of α of the first robot's positions *and* to a fraction of α of the positions *not* visited by the first robot.

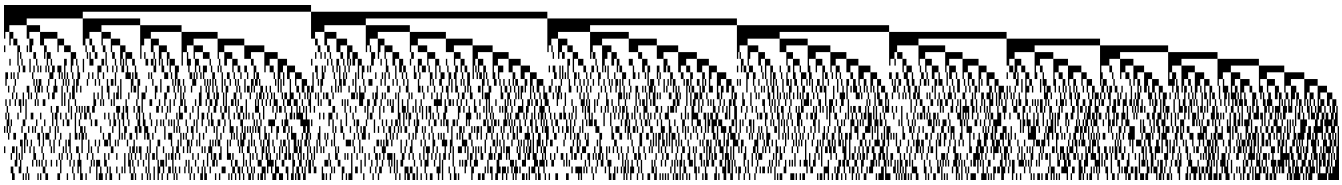
  If we iterate this idea to determine where to send k robots, we run into the issue that $\alpha^k \to 0$ rapidly; in particular, rounding errors take over around $k \approx 7$ and we do not separate the 'blocks' of possible positions for fixed return values evenly any more. We therefore switch to a greedy approach that tries several random plans for each individual robot, and then takes the one that minimizes the maximum size of any block. Possibly combining this with some local optimization, we obtain solutions with around **30** robots.

Generating positions. Another way to generate a solution is to fix the number of robots and to generate positions one-by-one. We start without any positions. Then, we can iteratively add new positions to our solution, always choosing a new set R_p for a new position p in such a way that there are no collisions with any or the previous positions.

- If we simply pick each set R_p randomly (with the above probabilities), this manages to generate a solution with **30** robots.
- If we try fewer robots than this, the number of valid choices for the next set R_p becomes very small. This means that every iteration takes very long and we do not reach the 1 000 positions that we need.

  Instead, at some appropriate point during the computation we can simply start to keep all possible compatible sets that would create no collision with the positions that we have generated to far. This makes it much faster to select the next set R_p. Moreover, we can connect this with other heuristics, such as selecting R_p in a way that maximises the number of remaining compatible sets. This can produce a solution with **27** robots.
- Right now, the Scientific Committee only knows a single way to obtain a solution with fewer robots. For this, we again select the sets R_p iteratively. However, we perform this in a very specific way. Namely, we only add sets with exactly 8 elements to our solution, and we also only add them if their symmetric difference with every set chosen so far contains at least 6 elements. Moreover, we try to add these sets in increasing order if we look at them in their binary representation. This produces a solution with 992 positions, which is not sufficient yet. If we then iterate through all remaining sets that we haven't tried so far (e.g. because they do not have the correct number of elements), and we try to add them greedily to our solution, it is possible to improve this solution to 998 positions. Unfortunately, this is still not enough. At this point we are stuck...

  ...is what we would have said if we had not tried to throw some simulated annealing on this solution to generate additional positions. After a short time, this manages to add two more positions to our solution. This is then a solution with **26** robots and 1 000 positions, shown in the picture below, where a black box (2 pixels wide and 10 pixels high) in row ϱ and column p means that robot ϱ is sent to position p:



  It turns out that our way of generating the initial solution with 992 positions is quite fragile. Whenever we tried to change any part of this approach (say by going through the sets R_p in a random order when trying to add them), the solution got much worse.

For 25 robots, the best we were able to do is a solution with slightly below 800 positions. The Scientific Committee also has no idea what the minimum number of robots is that a solution requires—we have not even been able to prove that more than 19 robots are necessary.