Algorithms

Howard Halim

January 4, 2021

Introduction

Problems will often ask you to show that something exists, and one way to solve them is by specifying an algorithm that finds or constructs it. This algorithmic approach proves existence by giving an explicit example, as opposed to non-constructive approaches like a proof by contradiction.

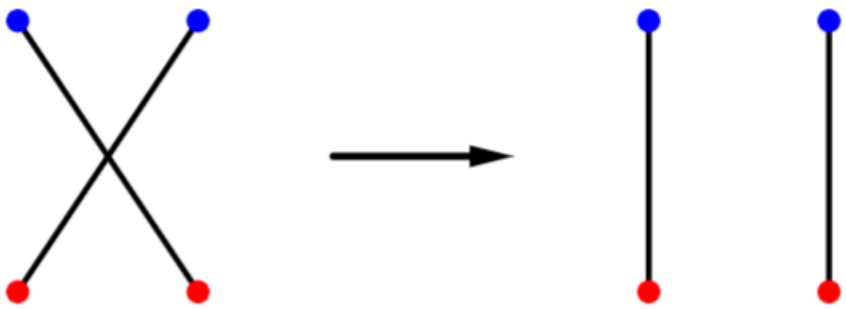
Existence problems aren't the only cases where algorithms are useful: some problems ask you to show that something is always possible, and these can be solved by providing an algorithm to do it. The first example below is a problem of this form.

Examples

Example 1 (Putnam 1979 A4)

There are n red points and n blue points in the plane, with no 3 points collinear. Prove that we can draw n segments joining a red point to a blue point such that no pair of segments intersect.

Key Observation — Whenever two segments intersect, we can always uncross them like this:



> **Useful Tip —** If you have a move that gets you closer to a desired state, try repeating it as many times as possible.

In this case, you might heuristically expect the number of intersections to decrease when uncrossing segments.[1] This brings us closer to the desired state of 0 intersections. Motivated by the tip above, we get the following algorithm:

> **Algorithm —** Start with an arbitrary matching between the red and blue points. If there is an intersection, use the uncrossing step above. Repeat until there are no more intersections!

In order for this algorithm to be a rigorous solution to the problem, we need to prove two things about it:

1. The algorithm eventually terminates after a finite number of steps.
2. It terminates with a valid answer (in this case, a matching with no intersections).

The second part is easy to prove—the algorithm can only terminate when the current matching has no intersections, so if it *does* terminate, we know that it has to be a valid matching.

> **Useful Tip —** To prove that an algorithm terminates, try finding a monovariant.

A monovariant is a number that always decreases at each step of the algorithm, and is bounded from below. For this algorithm, a natural guess is that the number of intersections is a monovariant.

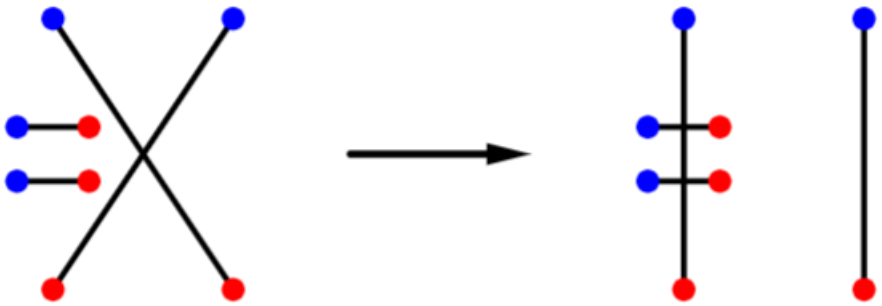
Let's say the initial matching has 100 intersections, and this number decreases after every step. As you apply the algorithm, the number of intersections at each step might look like

$$100,\ 95,\ 93,\ 88,\ \ldots,\ 2,\ 1,\ 0$$

Since the number of intersections decreases every time, and is always nonnegative, the algorithm cannot go on forever, and must terminate in a finite number of steps (at most 100 steps, in this case).

Unfortunately, we can't solve the problem this way because the number of intersections is *not* a monovariant. In some cases, uncrossing two segments will actually *increase* the total number of intersections.

[1] Generally, replacing one segment with another randomly chosen segment has a 50:50 chance of adding/removing new intersections. However, the uncrossing step *guarantees* that one intersection will be removed, so the number of intersections should decrease on average.



Instead, we'll use a different monovariant: the total length of the n segments. Try to convince yourself that this total length decreases after every step (hint: triangle inequality).

Unlike the number of intersections, the total length is an arbitrary real number. Even though it decreases at every step and is bounded below, that doesn't mean it will reach a minimum value. It might look like

$$2.1,\ 2.01,\ 2.001,\ 2.0001,\ \ldots$$

Fortunately, there are only a finite number of matchings between the n blue and n red points ($n!$ matchings, to be precise). This means the total length can only take on a finite number of values, and cannot decrease forever. Therefore, our algorithm will eventually terminate, and once it does, we will get our desired matching with no intersections.

> **Write-Up Tool —** When your algorithm has a monovariant, it can usually be simplified with the extremal principle.

Since we have a decreasing monovariant—the total length of the n segments—we pick the matching with the minimal total length (there are finitely many matchings, so one of them must have minimal total length). If it has an intersection, we could uncross it to get a matching with even smaller total length, contradicting the minimality assumption. Therefore, this matching must have no intersections.

This extremal proof is essentially the same as the algorithmic one, except much shorter: it skips directly to the minimal matching instead of starting at an arbitrary matching and gradually decreasing the total length. You can take advantage of this to simplify your write-ups.

Example 2

You are given 50 points in the plane, with no 4 collinear. Prove that you can choose 10 of these points such that no 3 chosen points are collinear.

In this problem, the goal is to find a set of 10 points satisfying a certain condition (no 3 points collinear). The way we construct such a set is by adding points one at a time, until there are no more valid points to add.

> **Algorithm —** Let $S = \varnothing$ be our (initially empty) set of points. Whenever a new point can be added to S without breaking the no-3-collinear rule, we add it to S. Repeat this until there are no more points that can be added.

It's clear that this algorithm will always terminate—there are only 50 points to add to S. The main difficulty is proving that it ends with the state we're looking for. Let k be the number of points in S at the end of the algorithm, then our goal is to prove that $k \geq 10$ for any given configuration of 50 points.

> **Useful Tip —** Whenever you have an algorithm, look at the ending state![2]

Our algorithm only ends when no more points can be added to S without creating a triple of collinear points. This tells us a lot about the ending state!

Any point not in S must lie on a line through 2 points in S (otherwise we could add the point to S without causing a contradiction). On the other hand, any line through 2 points in S contains at most 1 point not in S (otherwise, 4 of the original points would be collinear). This gives us the following inequality:

$$\#(\text{lines through 2 points in } S) \geq \#(\text{points not in } S)$$

Now, the rest is straightforward. We can compute both the left and right sides in terms of k, then solve the inequality to get a bound for k.

Each of the $\binom{k}{2}$ pairs of points in S determines a unique line, since no 3 points in S are collinear. Also, k of the 50 total points are in S, so the remaining $50 - k$ points are not in S. Therefore,

[2] This applies not just to algorithms, but to any multi-step process such as a game where players take turns making moves. A good example of this is USAMO 1999 P5.

$$\begin{aligned}\binom{k}{2} &\geq 50-k \\ \frac{k(k-1)}{2} &\geq 50-k \\ \frac{k(k+1)}{2} &\geq 50 \\ k &\geq 10\end{aligned}$$

Now we'll use the same method to make an IMO P6 look easy!

Example 3 (IMO 2014 P6)

A set of lines in the plane is in *general position* if no two are parallel and no three pass through the same point. A set of lines in general position cuts the plane into regions, some of which have finite area; we call these its *finite regions*. Prove that for all sufficiently large n, in any set of n lines in general position it is possible to color at least $\sqrt{n}$ lines blue in such a way that none of its finite regions has a completely blue boundary.

Much like the previous example, we want to contruct a set S of blue lines satisfying a certain property. Initially, S will be empty, and all lines will be a different color (red, for example). Then we convert red lines to blue as long as they don't create a completely blue region.

Algorithm — Start by coloring all n lines red. Then, if a red line can be colored blue without forming a finite region with a completely blue boundary, we color it blue. Repeat this until no additional lines can be colored blue.

Next, we analyze the ending state. This algorithm can only end when each red line cannot be colored blue, which means that the red line must be part of an *almost-blue* region (a region with only one red edge).

Since each red line is part of at least one almost-blue region, and each almost-blue region contains only one red line, we get the inequality:

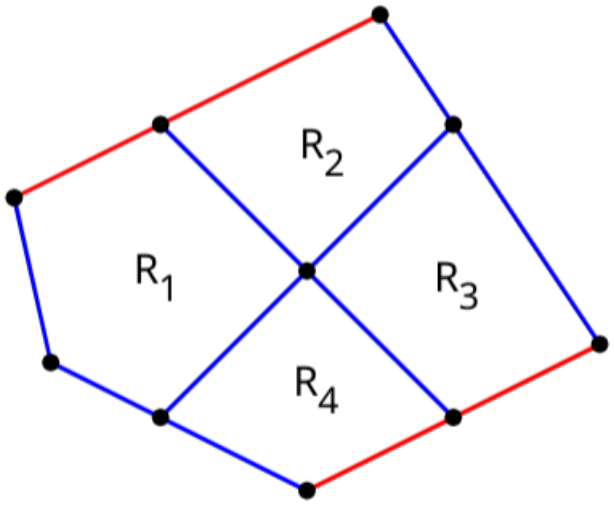
$$\#(\text{almost-blue regions}) \geq \#(\text{red lines})$$

Let k be the number of blue lines at the end of the algorithm. Our goal is to prove that $k \geq \sqrt{n}$, ideally with the above inequality.

The right-hand side of the inequality is $n - k$. However, the left-hand side is not as simple! There isn't an easy way to express the number of almost-blue regions in terms of n and k, so we'll replace the almost-blue regions with something that's easier to count: intersections of blue lines.

For each red line, we choose one of its almost-blue regions and pick the vertex that's clockwise of the red edge (on the chosen region's boundary). This assigns each red line to an intersection of blue lines. The last step is to bound the number of red lines that can be assigned to the same intersection.

Each intersection is part of 4 regions, so there are at most 4 red lines assigned to it. But we can get an even better bound! Since a red line assigned to an intersection will always extend to the next region, there can be at most 2 red lines assigned to any intersection (see diagram and explanation below).



R_1 is an almost-blue region whose red line is assigned to the middle intersection. It's red line is also part of R_2, which means that R_2 cannot connect a new red line to the same intersecction. Similarly, the red line in R_3 will always extend to R_4.

This means that each intersection of blue lines has at most 2 red lines, but each red line is assigned to 1 intersection of blue lines, so we get the inequality:

$$2 \cdot \#(\text{intersections of blue lines}) \geq \#(\text{red lines})$$

Since there are k blue lines in general position, there are exactly $\binom{k}{2}$ intersections of blue lines, so

$$2 \cdot \binom{k}{2} \geq n - k$$
$$k^2 - k \geq n - k$$
$$k \geq \sqrt{n}$$

The first 3 examples have all used greedy algorithms, so here's a different example.

Example 4 (Slovenia TST 2018 P1)

Let n be a positive integer. We have n^2 ornaments in n different colors, not necessarily n of each color. Prove that we can hang the ornaments on n Christmas trees with n ornaments per tree, such that each tree has at most 2 different ornament colors.

In this problem, the goal is to assign ornaments to trees, while maintaining a maximum of two colors per tree. Instead of figuring out how to assign ornaments to all n trees in one go, we can break it down into an n-step algorithm, where the i^{th} step assigns ornaments to the i^{th} tree.

The first step is easy—we only need to choose n out of the n^2 ornaments such that the chosen ornaments have at most 2 distinct colors, which can always be done by picking all ornaments from the most common color (a color appears n times on average, so the most common color appears at least n times).

However, the last step is the trickiest. There will only be n remaining ornaments for this step, so we are forced to use all of them. If there are 3 or more distinct colors left, the algorithm will fail. We need to avoid this scenario by carefully choosing which ornaments to use in the earlier steps!

To avoid having too many colors remaining, we will try to use up as many colors as possible. Once all ornaments of a given color have been used, that color is no longer relevant for future steps. It turns out we can always use up a new color at every step, by following this process:

Algorithm — For each step, choose ornaments from the least common color. If ornaments of this color run out and more ornaments are needed, choose the remaining ornaments from the most common color.

Why does this work? Since we are using n ornaments per tree, when there are c trees remaining, there will be cn ornaments remaining. Assuming that previous steps decreased the number of colors by 1 each time (starting from n colors), there will also be c colors remaining, so there are an average of n ornaments per color.

Therefore, the least common color appears at most n times, which guarantees that the least common color gets completely used up. Also, the most common color appears at least n times, which is enough to ensure that the tree has n ornaments.

Note that in order to prove the correctness of a single step, we assumed that all previous steps of the algorithm worked correctly. The n^{th} step only works because the first $n-1$ steps removed a different color each time, leaving only 1 color for the last step. Proofs like this are naturally suited for induction.

> **Write-Up Tool —** You can use strong induction to prove that your algorithm is correct. When proving that the k^{th} step works, you can assume that the first $k-1$ steps work as intended.

Induction can also work in the reverse order, starting with a proof that the last step of the algorithm is correct, and building up from there. For this algorithm, the reverse proof would start at 1 tree and induct up to n trees. Here's what the induction looks like:

> **Reverse Induction Proof —** We prove the following claim by induction on c:
>
> Suppose there are cn ornaments in c different colors. We can hang the ornaments on c trees with n ornaments per tree, such that each tree has at most 2 different ornament colors.
>
> The base case $c = 1$ is trivial. When $c > 1$, we can reduce this to the $c-1$ case by picking ornaments from the least and most common colors, as described earlier.

Example 5 (USAMO 1994 P2)

The sides of a 99-gon are initially colored so that consecutive sides are red, blue, red, blue, …, red, blue, yellow. We make a sequence of modifications in the coloring, changing the color of one side at a time to one of the three given colors (red, blue, yellow), under the constraint that no two adjacent sides may be the same color. By making a sequence of such modifications, is it possible to arrive at the coloring in which consecutive sides are red, blue, red, blue, red, blue, …, red, yellow, blue?

Often, you'll encounter problems which define an algorithm for you (or define a move that you can repeatedly use), and ask you to analyze it. For problems like this, finding invariants can be helpful.

> **Useful Tip —** To prove that an algorithm will never reach a given end state, find an invariant which has different values at the start and end states.

To find an invariant, focus on a single step of the algorithm and see what stays the same. In this case, a single step involves changing the color of one side.

Note that when one side changes color (from red to yellow, for example), then the neighboring sides must both be the third color (blue). They cannot be red or yellow, otherwise they would conflict with the first edge before or after the change in color.

Also, since no two edges are the same color, if we label the colors with values of 0, 1, or 2 (mod 3), and read the labels clockwise around the 99-gon, then the labels will either increase or decrease by 1 (mod 3) each time.

Whenever a side changes color, there will always be one increase and one decrease between the side and its neighbors, before and after the change. For example, if $1, 0, 1$ changes to $1, 2, 1$, then the initial $1 \to 0$ decrease and $0 \to 1$ increase becomes a $1 \to 2$ increase and $2 \to 1$ decrease. This means that the total number of increases (and decreases) is invariant!

But the initial red, blue, red, blue, ..., red, blue, yellow state has 51 increases and 48 decreases, while the red, blue, red, blue, ..., red, yellow, blue state has 48 increases and 51 decreases. Therefore, it's impossible to go from one state to the other.

Example 6 (Tournament of Towns 2016, IOI 2014 P3)

There are 64 towns in a country and some pairs of towns are connected by roads but we do not know these pairs. We may choose any pair of towns and find out whether they are connected or not. Our aim is to determine whether it is possible to travel from any town to any other by a sequence of roads. Prove that there is no algorithm which enables us to do so in less than 2016 questions.

Sometimes, instead of proving that a task is always impossible, you'll need to prove that it's impossible to do within a certain number of steps (equivalently, a minimum number of steps is required to perform the task). A simple invariant proof will not be enough, so you'll need a careful argument to prove limits on what the algorithm can do. For this problem, we restrict the information that the algorithm gets.

Interpret the towns and roads as a graph, and note that there are $\binom{64}{2} = 2016$ potential edges, so the problem statement is asking us to show that no algorithm can determine the connectivity of an arbitrary graph without asking every possible pair of vertices.

Assume for the sake of contradiction that there is an algorithm which does this. Since the algorithm work for all of the 2^{2016} possible graphs, we can adversarially choose the underlying graph, and modify it whenever a question is asked (as long as we stay consistent with previous answers). This lets us reply to the algorithm's questions with any yes/no answer of our choice.

To reduce the number of graphs to consider, we will answer in a very specific way: We answer all questions with a No, except for when answering No would immediately tell the algorithm that the graph is disconnected, in which case we answer with a Yes.

Much like the previous examples, we analyze the ending state of this algorithm. We know that terminates in under 2016 steps, its final conclusion must be that the graph is connected, because of how we answered.

Let E be the set of edges which we answered Yes to. These edges must form a subgraph connecting all 64 vertices, since these are the only edges that the algorithm has confirmed—from the algorithm's point of view, it's possible that all edges besides those in E do not exist, and it was still certain that the graph was connected.

Since the algorithm asked less than 2016 questions, there is an edge $e_1 = (a, b)$ which was never asked. Since E connects all 64 vertices, we can find a path P from vertices a to b with edges in E. Note that $P \cup \{e_1\}$ is a cycle, which is still connected after removing any edge.

Let e_2 be the last edge in P that the algorithm asked about. The fact that we answered Yes to e_2 means that answering No would force the graph to be disconnected. But this is a contradiction, since we could still have a connected graph without e_2 if we used the rest of the edges in the cycle, as well as the edges in $E \setminus P$.

Practice Problems

Come up with an algorithm

1.1 (IMO 2003 P1) Let A be a 101-element subset of the set $S = \{1, 2, \ldots, 1000000\}$. Prove that there exist numbers t_1, $t_2, \ldots, t_{100}$ in S such that the sets

$$A_j = \{x + t_j \mid x \in A\}, \qquad j = 1, 2, \ldots, 100$$

are pairwise disjoint.

1.2 (ARMO 2005 Grade 9 P6) In a $2 \times n$ array of positive real numbers, the sum of the two real numbers in each of the n columns is 1. Prove that it's possible to select one number from each column such that the sum of the selected numbers in each row is at most $\frac{n+1}{4}$.

1.3 (JMO 2015 P1) Given a sequence of real numbers, a move consists of choosing two terms and replacing each with their arithmetic mean. Show that there exists a sequence of 2015 distinct real numbers such that after one initial move is applied to the sequence – no matter what move – there is always a way to continue with a finite sequence of moves so as to obtain in the end a constant sequence.

1.4 (ARMO 2019 Grade 10 P2) Pasha and Vova play the following game, alternating turns with Pasha moving first. Initially, they have a large piece of plasticine. On his turn, Pasha cuts one of the existing pieces into three (of arbitrary sizes), and Vova merges two existing pieces into one. Pasha wins if at some point, there are 100 pieces of equal weight. Can Vova prevent Pasha's win?

1.5 Lisa and Rachel each pick n positive integers (not necessarily distinct) from the set $\{1, \ldots, n\}$. Prove that some nonempty subset of Lisa's integers and some nonempty subset of Rachel's integers have equal sum of elements.

1.6 (Codeforces #1270 G) You are given n integers $a_1, \ldots, a_n$ such that $i - n \le a_i \le i - 1$ for all $1 \le i \le n$. Prove that there is a nonempty subsequence of those integers with sum equal to 0.

1.7 (IMO 2010 P5) Each of the six boxes B_1, B_2, B_3, B_4, B_5, B_6 initially contains one coin. The following operations are allowed:

(a) Choose a non-empty box B_j, $1 \le j \le 5$, remove one coin from B_j and add two coins to B_{j+1}.

(b) Choose a non-empty box B_k, $1 \le k \le 4$, remove one coin from B_k and swap the contents (maybe empty) of the boxes B_{k+1} and B_{k+2}.

Determine if there exists a finite sequence of operations of the allowed types, such that the five boxes B_1, B_2, B_3, B_4, B_5 become empty, while box B_6 contains exactly $2010^{2010^{2010}}$ coins.

1.8 (IMO 2017 P5) An integer $N \geq 2$ is given. A collection of $N(N+1)$ soccer players, no two of whom are of the same height, stand in a row. Sir Alex wants to remove $N(N-1)$ players from this row leaving a new row of $2N$ players in which the following N conditions hold:

- no one stands between the two tallest players,
- no one stands between the third and fourth tallest players,
  ⋮
- no one stands between the two shortest players.

Show that this is always possible.

Analyze a given algorithm / prove it cannot work

2.1 (APMO 2009 P1) Consider the following operation on positive real numbers written on a blackboard: Choose a number r written on the blackboard, erase that number, and then write a pair of positive real numbers a and b satisfying the condition $2r^2 = ab$ on the board.

Assume that you start out with just one positive real number r on the blackboard, and apply this operation $k^2 - 1$ times to end up with k^2 positive real numbers, not necessarily distinct. Show that there exists a number on the board which does not exceed kr.

2.2 (EGMO 2018 P3) The n contestants of EGMO are named $C_1, C_2, \cdots C_n$. After the competition, they queue in front of the restaurant according to the following rules.

- The Jury chooses the initial order of the contestants in the queue.
- Every minute, the Jury chooses an integer i with $1 \leq i \leq n$.
  - If contestant C_i has at least i other contestants in front of her, she pays one euro to the Jury and moves forward in the queue by exactly i positions.
  - If contestant C_i has fewer than i other contestants in front of her, the restaurant opens and process ends.

(a) Prove that the process cannot continue indefinitely, regardless of the Jury's choices.

(b) Determine for every n the maximum number of euros that the Jury can collect by cunningly choosing the initial order and the sequence of moves.

2.3 (ISL 1996 C6) A finite number of coins are placed on an infinite row of squares. A sequence of moves is performed as follows: at each stage a square containing more than one coin is chosen. Two coins are taken from this square; one of them is placed on the square immediately to the left while the other is placed on the square immediately to the right of the chosen square. The sequence terminates if at some point there is at most one coin on each square. Given some initial configuration, show that any legal sequence of moves will terminate after the same number of steps and with the same final configuration.

2.4 (USAMO 2019 P5, JMO 2019 P6) Two rational numbers $\frac{m}{n}$ and $\frac{n}{m}$ are written on a blackboard, where m and n are relatively prime positive integers. At any point, Evan may pick two of the numbers x and y written on the board and write either their arithmetic mean $\frac{x+y}{2}$ or their harmonic mean $\frac{2xy}{x+y}$ on the board as well. Find all pairs (m,n) such that Evan can write 1 on the board in finitely many steps.

2.5 (USA TST 2015 P3) A physicist encounters 2015 atoms called usamons. Each usamon either has one electron or zero electrons, and the physicist can't tell the difference. The physicist's only tool is a diode. The physicist may connect the diode from any usamon A to any other usamon B. (This connection is directed.) When she does so, if usamon A has an electron and usamon B does not, then the electron jumps from A to B. In any other case, nothing happens. In addition, the physicist cannot tell whether an electron jumps during any given step. The physicist's goal is to isolate two usamons that she is sure are currently in the same state. Is there any series of diode usage that makes this possible?

Find the minimum number of steps required

3.1 (Tournament of Towns 2016)

(a) There are $2n+1$ ($n>2$) batteries. We don't know which batteries are good and which are bad but we know that the number of good batteries is greater by 1 than the number of bad batteries. A lamp uses two batteries, and it works only if both of them are good. What is the least number of attempts sufficient to make the lamp work?

(b) The same problem but the total number of batteries is $2n$ ($n>2$) and the numbers of good and bad batteries are equal.

3.2 (ISL 2018 C3) Let n be a given positive integer. Sisyphus performs a sequence of turns on a board consisting of $n+1$ squares in a row, numbered 0 to n from left to right. Initially, n stones are put into square 0, and the other squares are empty. At every turn, Sisyphus chooses any nonempty square, say with k stones, takes one of these stones and moves it to the right by at most k squares (the stone should stay within the board). Sisyphus' aim is to move all n stones to square n. Prove that Sisyphus cannot reach the aim in less than

$$\left\lceil \frac{n}{1} \right\rceil + \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{n}{3} \right\rceil + \cdots + \left\lceil \frac{n}{n} \right\rceil$$

turns.

3.3 (Kurschak 2010 P1) We have n keys, each of them belonging to exactly one of n locked chests. Our goal is to decide which key opens which chest. In one try we may choose a key and a chest, and check whether the chest can be opened with the key. Find the minimal number $p(n)$ with the property that using $p(n)$ tries, we can surely discover which key belongs to which chest.

3.4 (ARMO 2019 Grade 11 P3) We are given n coins of different weights and n balances, $n > 2$. On each turn one can choose one balance, put one coin on the right pan and one on the left pan, and then delete these coins out of the balance. It's known that one balance is wrong (but it's not known ehich exactly), and it shows an arbitrary result on every turn. What is the smallest number of turns required to find the heaviest coin?

3.5 (ISL 2019 C8) Alice has a map of Wonderland, a country consisting of $n \geq 2$ towns. For every pair of towns, there is a narrow road going from one town to the other. One day, all the roads are declared to be "one way" only. Alice has no information on the direction of the roads, but the King of Hearts has offered to help her. She is allowed to ask him a number of questions. For each question in turn, Alice chooses a pair of towns and the King of Hearts tells her the direction of the road connecting those two towns.

Alice wants to know whether there is at least one town in Wonderland with at most one outgoing road. Prove that she can always find out by asking at most $4n$ questions.