Problem Set 5

This problem set – the last one purely on discrete mathematics – is designed as a cumulative review of the topics we’ve covered so far and a proving ground to try out your newfound skills with mathematical induction. The problems here span all sorts of topics – higher dimensions, tiling problems, games, and star-drawing – and we hope that it serves as a fitting coda to our whirlwind tour of discrete math!

We recommend that you read Handout #28, “Guide to Induction,” before starting this problem set. It contains a lot of useful advice about how to approach problems inductively, how to structure inductive proofs, and how to not fall into common inductive traps. Additionally, before submitting, be sure to read over Handout #29, the “Induction Proofwriting Checklist,” for a list of specific things to watch for in your solutions before submitting.

As a note on this problem set – normally, you're welcome to use any proof technique you'd like to prove results in this course. On this problem set, we've specifically requested on some problems that you prove a result inductively. For those problems, you should prove those results using induction or complete induction, even if there is another way to prove the result. (If you'd like to use induction in conjunction with other techniques like proof by contradiction or proof by contrapositive, that's perfectly fine.)

As always, please feel free to drop by office hours, visit Piazza, or send us emails if you have any questions. We'd be happy to help out.

Good luck, and have fun!

Due Friday, November 2nd at 2:30PM.
Problem One: Recurrence Relations

A recurrence relation is a recursive definition of the terms in a sequence. Typically, a recurrence relation specifies the value of the first few terms in a sequence, then defines the remaining entries from the previous terms. For example, the Fibonacci sequence can be defined by the following recurrence relation:

\[
\begin{align*}
F_0 &= 0 \\
F_1 &= 1 \\
F_{n+2} &= F_n + F_{n+1}
\end{align*}
\]

The first terms of this sequence are \(F_0 = 0, F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8\), etc.

Some recurrence relations define well-known sequences. For example, consider the following recurrence relation:

\[
\begin{align*}
a_0 &= 1 \\
a_{n+1} &= 2a_n
\end{align*}
\]

The first few terms of this sequence are \(1, 2, 4, 8, 16, 32, \ldots\), which happen to be powers of two. It turns out that this isn’t a coincidence – this recurrence relation perfectly describes the powers of two.

i. Prove by induction that for any \(n \in \mathbb{N}\), we have \(a_n = 2^n\).

In case you’re wondering what you’re asked to prove here, you can think of this recurrence relation as a mathematical way of writing out this recursive function:

```c
int a(int n) {
    if (n == 0) return 1;
    return 2 * a(n - 1);
}
```

For any natural number \(n\), you can compute \(a(n)\) by just running this code, and after doing some computation it will return the value of \(a_n\). What we’re asking you to do is the mathematical equivalent of showing that the value returned by \(a(n)\) is always \(2^n\). While it might help to think about things in terms of this analogy, your proof should not reference this code and should just use the definitions given in the problem statement.

Minor changes to the recursive step in a recurrence relation can lead to enormous changes in what numbers are generated. Consider the following two recurrence relations, which are similar to the \(a_n\) sequence defined above but with slight changes to the recursive step:

\[
\begin{align*}
b_0 &= 1 \\
b_{n+1} &= 2b_n - 1 \\
c_0 &= 1 \\
c_{n+1} &= 2c_n + 1
\end{align*}
\]

ii. Find non-recursive definitions for \(b_n\) and \(c_n\), then prove by induction that your definitions are correct.

This one is hard to do just by eyeballing the recurrences. Try expanding out the first few terms of these sequences and see what you find.

Finding non-recursive definitions for recurrences (often called “solving” the recurrence) is useful in the design and analysis of algorithms. Commonly, when trying to analyze the runtime of a recursive algorithm, you will arrive at a recurrence relation describing the runtime on an input of size \(n\) in terms of the runtime on inputs of smaller sizes. Solving the recurrence then lets you precisely determine the runtime. To learn more, take CS161, Math 108, or consider reading through the excellent textbook *Concrete Mathematics* by Graham, Knuth, and Patashnik.
Problem Two: Induction and Recursion

There's a close connection between mathematical induction and recursion, and many of the proofs by induction that we did in class can be thought of as claims about how specific recursive functions work.

One of the first proofs by induction that we did was to prove that, given a collection of $3^n$ coins containing a single counterfeit coin that's heavier than the rest, it is always possible to discover which coin is fake using exactly $n$ weighings on a balance. The key idea behind the proof was, essentially, a recursive algorithm that can be used to actually go and find which of the coins is counterfeit!

i. Implement a recursive function

```cpp
Coin counterfeitIn(std::vector<Coin> coins, Balance balance);
```

that takes as input a set of exactly $3^n$ coins for some natural number $n$, one of which is counterfeit and weighs more than the rest, and returns which one that is. You're provided a balance you can use to weigh groups of coins and can make at most $n$ weighings on that balance. Check the header CounterfeitCoins.h for a description of the relevant types here.

Our provided starter files provide an interface you can use to test out your function on a number of different inputs and will show you which coins actually get weighed against one another.

Your code from part (i) shows that the inductive argument we made in class can be converted into a recursive algorithm that actually finds the coin!

Now, here's a fun little variant on the counterfeit coin problem. Imagine that you're given a collection of coins. You're told that there might be a counterfeit in it, but then again, there might not be. If there is a counterfeit coin, it's guaranteed to be heavier than the rest. Your job is to determine whether there even is a counterfeit coin at all and, if so, to return which one it is.

ii. Implement a recursive function

```cpp
Coin maybeCounterfeitIn(std::vector<Coin> coins, Balance balance);
```

that takes as input a set of exactly $3^n - 1$ coins for some natural number $n$, which might contain a counterfeit that weighs more than the rest. The function should either return the counterfeit coin if one exists, or return the special constant None if none of the coins are counterfeit. You're provided a balance you can use to weigh groups of coins and can make at most $n$ weighings on that balance.

Again, test locally, and test thoroughly – it's easy to miss cases!

iii. Using the recursive intuition that you developed in the course of solving part (ii) of this problem, prove that given any collection of exactly $3^n - 1$ coins, of which at most one is a counterfeit that weighs more than the rest, it is always possible to identify which coin that is using at most $n$ weighings on a balance (or to report that all coins are genuine). Your proof should have a similar structure to the one about counterfeit coins from lecture. While you should not explicitly reference the code you wrote in part (ii) of this problem, you may want to use the same recursive insight from that problem to guide the structure of your proof.

We hope that this exercise gives you a better sense for the interplay between theory (proof by induction) and practice (recursive problem-solving). If you're interested in this sort of thing, we strongly recommend checking out CS161, where you'll alternate between designing clever algorithms and using induction to prove that they work correctly.
Problem Three: Tiling with Triominoes

A right triomino is an L-shaped tile that looks like this:

Suppose you're given a $2^n \times 2^n$ grid of squares and want to tile it with right triominoes by covering the grid with triominoes such that all triominoes are completely on the grid and no triominoes overlap. Here's an attempt to cover an $8 \times 8$ grid with triominoes, which doesn't manage to cover all squares:

Amazingly, it turns out that it is always possible to tile any $2^n \times 2^n$ grid that's missing exactly one square with right triominoes. It doesn't matter what $n$ is or which square is removed; there is always a solution to the problem. For example, here are all the ways to tile a $4 \times 4$ grid that has a square missing:

This question explores why this is the case.

i. Prove, by induction, that $4^n - 1$ is a multiple of three for any $n \in \mathbb{N}$. (An integer $n$ is a multiple of three if there is an integer $k$ such that $n = 3k$.)

Any $2^n \times 2^n$ grid missing a square has a number of squares has exactly $4^n - 1$ squares, and so its number of squares is a multiple of three. Although you can show that a figure can't be tiled with triominoes by showing that its number of squares isn't a multiple of three, you can't show that a figure can be tiled with triominoes purely by showing that its number of squares is a multiple of three. The arrangement matters.

ii. Draw a figure made of squares where the number of squares is a multiple of three, yet the figure cannot be tiled with right triominoes. Briefly justify your answer; no formal proof is necessary.

iii. Prove by induction that for any natural number $n$, any $2^n \times 2^n$ grid with any one square removed can be tiled by right triominoes.

Before you write this proof, try seeing if you can find a nice recursive pattern you can follow that will let you fully tile any such board. You should be able to easily tile any $8 \times 8$ chessboard missing a square with right triominoes before you attempt to write up your answer. Once you can do this, formalize your idea in your answer. You may want to think about how to start with a larger board and subdivide it into some number of smaller boards.
Problem Four: It’ll All Even Out

Our very first proof by induction was the proof that for any natural number \( n \), we have that
\[
2^0 + 2^1 + 2^2 + \ldots + 2^{n-1} = 2^n - 1.
\]

This result is still true for the case where \( n = 0 \), since in that case the sum on the left-hand side of the equation is the empty sum of zero numbers, which is by definition equal to zero. It’s also true for the case where \( n = 1 \); in that case, the sum on the left-hand side of the equality just has a single term in it (\( 2^0 \)) and the right-hand side has the same value.

Below is a proof by complete induction of an incorrect statement about what happens when you sum up zero or more real numbers:

**Theorem:** The sum of any number of real numbers is even.

**Proof:** Let \( P(n) \) be the statement “the sum of any \( n \) real numbers is even.” We will prove by complete induction that \( P(n) \) holds for all \( n \in \mathbb{N} \), from which the theorem follows.

As a base case, we prove \( P(0) \), that the sum of any 0 real numbers is even. The sum of any zero numbers is the empty sum and is by definition equal to 0, which is even. Thus \( P(0) \) holds.

For our inductive step, assume for some arbitrary \( k \in \mathbb{N} \) that \( P(0), \ldots, P(k) \) are true. We will prove that \( P(k+1) \) is true, meaning that the sum of any \( k+1 \) real numbers is even. To do so, let \( x_1, x_2, \ldots, x_k, \) and \( x_{k+1} \) be arbitrary real numbers and consider the sum
\[
x_1 + x_2 + \ldots + x_k + x_{k+1}.
\]

We can group the first \( k \) terms and the last term independently to see that
\[
x_1 + x_2 + \ldots + x_k + x_{k+1} = (x_1 + x_2 + \ldots + x_k) + (x_{k+1}).
\]

Now, consider the sum \( x_1 + x_2 + \ldots + x_k \) of the first \( k \) terms. This is the sum of \( k \) real numbers, so by our inductive hypothesis that \( P(k) \) is true we know that this sum must be even. Similarly, consider the sum \( x_{k+1} \) consisting of just the single term \( x_{k+1} \). By our inductive hypothesis that \( P(1) \) is true, we know that this sum must be even.

Overall, we have shown that \( x_1 + x_2 + \ldots + x_k + x_{k+1} \) can be written as the sum of two even numbers (namely, \( x_1 + x_2 + \ldots + x_k \) and \( x_{k+1} \)), so \( x_1 + x_2 + \ldots + x_k + x_{k+1} \) is even. Thus \( P(k+1) \) is true, completing the induction. ■

Of course, this result has to be incorrect, since there are many sums of real numbers that don’t evaluate to an even number. The sum 2 + 3 + 4, for example, works out to 9, and the sum \( \pi + 1 \) doesn’t even work out to an integer!

What’s wrong with this proof? Be as specific as possible. For full credit, you should be able to identify a specific claim made in the proof that is not correct, along with an explanation as to why it’s incorrect.

Think about our “induction as a machine” analogy from lecture that explains why you can start with a base case and inductive step and end up with a proof that works for all natural numbers. See what happens if you try that out here.
Problem Five: The Circle Game

Here's a game you can play. Suppose that you have a circle with $2n$ arbitrarily-chosen points on its circumference. Of the $2n$ points, $n$ are labeled +1, and the remaining $n$ are labeled -1. One sample circle with eight points, of which four are labeled +1 and four are labeled -1, is shown below.

Here's the rules of the game. First, choose one of the $2n$ points as your starting point. Then, start moving clockwise around the circle. As you go, you'll pass through some number of +1 points and some number of -1 points. You lose the game if at any point on your journey you pass through more -1 points than +1 points. You win the game if you get all the way back around to your starting point without losing.

For example, if you started at point A, the game would go like this:

Start at A: +1.
Pass through B: +2.
Pass through C: +1.
Pass through D: 0.
Pass through E: -1. (You lose.)

If you started at point G, the game would go like this:

Start at G: -1 (You lose.)

However, if you started at point F, the game would go like this:

Start at F: +1.
Pass through G: 0.
Pass through H: +1.
Pass through A: +2.
Pass through B: +3.
Pass through C: +2.
Pass through D: +1.
Pass through E: +0.
Return to F. (You win!)

Interestingly, it turns out that no matter which $n$ points are labeled +1 and which $n$ points are labeled -1, there is always at least one point you can start at to win the game.

Prove, by induction, that the above fact is true for any $n \geq 1$.

*Check the Guide to Induction and Inductive Proofwriting Checklist before starting this one.*

Problem Six: Nim

*Nim* is a family of games played by two players. The game begins with several piles of stones, each of which has zero or more stones in it, shared between the two players. Players alternate taking turns removing any nonzero number of stones from any single pile of their choice. If at the start of a player's turn all the piles are empty, then that player loses the game.

Prove, by induction, that if the game is played with just two piles of stones, each of which begins with exactly the same number of stones, then the second player can always win the game if she plays correctly.

*Play this game with a partner until you can find a winning strategy. Once you spot the pattern, see if you can find a way to formalize it using induction. Be wary of writing statements of the form “and so on” or “by repeating this;” those aren’t rigorous ways to formalize that a process will eventually do something.*
Problem Seven: Transitive Closures

Given a binary relation $R$ over a set $A$ and a natural number $n \geq 1$, we can define a new relation called the $n$th power of $R$, denoted $R^n$, over the same set $A$. This family of relations is defined as follows:

$$ xR^1y \quad \text{if} \quad xRy, $$

$$ xR^{n+1}y \quad \text{if} \quad \exists z \in A. (xRz \land zR^ny). $$

That definition might look like quite a mouthful, but it has a really nice intuition.

i. To the right is a drawing of a binary relation $S$ over the set \{a, b, c, d, e\}.

Draw a picture of $S^2$ and a picture of $S^3$.

The notation in powers of relations looks like the notation for exponentiation, and for good reason. Here’s a useful lemma about powers of relations that we’re going to ask you to prove:

**Lemma:** For any positive natural numbers $m$ and $n$, if $aR^mb$ and $bR^nc$, then $aR^{m+n}c$.

The lemma shown above concerns natural numbers, and so it’s a candidate for a proof by induction. However, unlike the other induction results you’ve seen so far, this one involves two natural numbers, not one. It’s still possible to use induction to prove this result, though.

ii. Consider the predicate $P(m)$ defined below:

$$ P(m) \text{ is “for any natural number } n > 0 \text{ and any } a, b, c \in A, \text{ if } aR^mb \text{ and } bR^nc, \text{ then } aR^{m+n}c.” $$

Suppose we prove $P(m)$ holds for all $m > 0$ using a proof by induction. Explain why this proves the above lemma.

iii. Let $R$ be some binary relation over a set $A$ (not necessarily a transitive relation). Prove by induction on $m$ that if $aR^mb$ and $bR^nc$, then $aR^{m+n}c$.

There are a lot of variables to keep track of here in the course of this proof, so be extremely careful to scope and introduce your variables properly. The inductive step of this problem, in particular, would be a great place to write out two columns, one of the things you’re assuming, and one of the things that you’re proving.

The most common class of mistake we tend to see on this problem is mixing up arbitrarily-chosen values with placeholders. One specific thing to keep an eye on: make specific claims about specific variables, and, specifically, be very careful to make sure you aren’t using placeholder variables.

Is this a problem where you’ll induct up? Or induct down?

The transitive closure of a binary relation $R$ over a set $A$, denoted $R^+$, is a binary relation over $A$ defined as follows:

$$ xR^+y \quad \text{if} \quad \exists n \in \mathbb{N}. (n \geq 1 \land xR^ny). $$

iv. Let $S$ be the relation over \{a, b, c, d, e\} drawn above. Draw a picture of the relation $S^+$.

v. Let $R$ be an arbitrary binary relation over a set $A$. Using your result from part (iii) of this problem, prove that $R^+$ is transitive.

Be sure to set this problem up properly. If you did everything correctly, the proof should be rather elegant and straightforward.

Transitive closures show up all the time in discrete mathematics. You’ll see an example of them when we play around with context-free grammars next week.
Problem Eight: The Star-Drawing Saga, Part IV: The Grand Finale

A powerful technique in any mathematician’s toolbox is the concept of dual objects. The formal definition of a dual object varies by context, but the key idea is to take an object made from two different parts, swap the meanings of those parts, and end up with a new object related to the one we started with.

Every star \( \{p/s\} \) has an associated dual star \( \{s/p\} \). In other words, starting with a \( p \)-pointed star with a step size of \( s \), we create a new star with \( s \) points and a step size of \( p \). That new star might have a step size that’s much, much larger than its number of points, but that’s nothing to worry about. It’s still perfectly mathematically valid! For example, here’s the \( \{12/5\} \) star and its dual, the \( \{5/12\} \) star. You might notice that the \( \{5/12\} \) star looks a lot like the \( \{5/2\} \) star – more on that later – but it is indeed \( \{5/12\} \).

On Problem Set Two, you proved that \( \{p/s\} \) is simple if and only if there is an integer \( t \) where \( 1 \equiv p \cdot t \). Technically, you only proved this theorem in the case where \( p > 0 \), but for the purposes of this problem, we’re going to extend this theorem and assume it holds for all natural numbers \( p \), not just positive ones.

i. Using the above theorem, prove that if \( p \) and \( s \) are natural numbers, then \( \{p/s\} \) is simple if and only if its dual star is simple.

You might have noticed that the \( \{5/12\} \) star looks a lot like the \( \{5/2\} \) star. To formalize why this is, we’ll need to use the division algorithm, which, despite the CS-sounding name, isn’t actually an algorithm. It’s the following fact about natural numbers:

The Division Algorithm: For any natural numbers \( m \) and \( n \) where \( n \neq 0 \), there exist unique natural numbers \( q \) and \( r \) where \( m = nq + r \) and \( r < n \).

This is the fancy mathematical way of saying “you can divide \( m \) by \( n \) to get a quotient (\( q \)) and a remainder (\( r \)), and the remainder \( r \) is always less than \( n \).” I find it amusing that we have such a weighty name for something so simple, but hey, that’s the convention.

ii. Let \( p \) and \( s \) be natural numbers where \( s > 0 \) and let \( r \) be the remainder you get via the division algorithm when dividing \( p \) by \( s \). Explain, intuitively, why \( \{s/p\} \) and \( \{s/r\} \) are the same star.

Note that the roles of the variables \( s \) and \( p \) are reversed from their normal arrangement in this problem.

iii. Let \( p \) and \( s \) be natural numbers where \( s > 0 \) and let \( r \) be the remainder you get via the division algorithm when dividing \( p \) by \( s \). Prove that \( \{p/s\} \) is simple if and only if \( \{s/r\} \) is simple.

We’re looking for a rigorous proof here involving modular congruence, so while the intuition from part (ii) might come in handy here, you still need to use a rigorous proof.

iv. Using your result from part (iii), prove or disprove: \( \{10,820,242,327 / 71,827,319\} \) is simple.

And now the grand finale. Under what circumstances is a star simple? You know from Problem Set Two that if \( \{p/s\} \) is a simple star, then \( p \) and \( s \) are coprime. And based on what you’ve seen above, you now have enough to prove the converse of this statement.

v. Prove by induction on \( s \) that if \( p \) and \( s \) are coprime natural numbers, then \( \{p/s\} \) is a simple star.

Feel free to use the fact that the only natural number coprime with 0 is 1.

Problem Seven asked you to do a proof by induction involving multiple variables. Use the same idea here.

You’ve come quite a long way since you first tried drawing a simple 7-pointed star! You’ve explored modular arithmetic, coprimality, and now the division algorithm and induction! There are so many other questions you could ask about stars. Although \( \{8/2\} \) isn’t a simple star, you can draw an eight-pointed star as two copies of the \( \{4/1\} \) star. Is there some rule about how many copies of simple stars you’d need for a general \( \{p/s\} \), or what those simple stars would be? If you keep on exploring these questions – which we hope you do! – you’ll quickly run into concepts from group theory, number theory, and abstract algebra. For more on this, check out Math 120, Math 152, and CS255!
Problem Nine: A Visit to the Fourth Dimension

In zero dimensions, we have a point. In one dimension, we have a line. In two dimensions, we have a square. In three dimensions, we have a cube. And in four dimensions, we have a tesseract. I don’t know about you, but I can’t visualize objects in four-dimensional space. But using the tools that we’ve developed in this quarter, we can investigate what it might be like to hold a tesseract in your hand and spin it around.

The first question is how to even think about what a tesseract is. For this, we can turn to graph theory. There’s a family of graphs called the hypercube graphs whose nodes and edges correspond to the vertices and edges of squares, cubes, tesseracts, and even more exotic higher-dimensional objects. The hypercube graph of order $k$, denoted $Q_k$, is defined as follows:

- The nodes of $Q_k$ are the elements of $\wp([k])$. (Refer to Problem Set Three for the definition of $[k]$.)
- There is an edge between a pair of nodes $S$ and $T$ if (and only if) $|S \Delta T| = 1$.

That definition might seem like a mouthful, but it makes a lot more sense once you have a visual intuition.

i. Draw the graphs $Q_0$, $Q_1$, $Q_2$, and $Q_3$. Label each node with the set it corresponds to. Then, explain why $Q_0$ is a good approximation of a point, $Q_1$ is a good approximation of a line, $Q_2$ is a good approximation of a square, and $Q_3$ is a good approximation of a cube.

You may need to shuffle around the nodes of $Q_3$ to see why it’s a good proxy for a cube.

The tesseract is modeled by $Q_4$. There are many ways to draw $Q_4$; one of them is shown to the right. We’ve omitted the labels on the nodes for convenience. (Whoa! It’s an {8 / 3} star inside another {8 / 3} star inside a {8 / 1} star!) It’s hard to look at that picture and to imagine what it would “feel like” to hold it in your hand. But by using automorphisms, which you saw in the previous problem set, we can get a better intuition.

From your lived experience you know that if you pick up a cube, you can twist and turn it around and it has all sorts of symmetries. From a graph theory perspective, that would lead us to think that $Q_3$ should have many automorphisms, since each automorphism corresponds to a symmetry. So here’s a question: does $Q_4$ have the same sort of symmetries that you’d expect of a cube or a square?

A graph $G = (V, E)$ is called node-symmetric if, for any two nodes $u, v \in V$, there is an automorphism $\sigma$ of $G$ where $\sigma(u) = v$. Intuitively, this means that all the nodes in $G$ “look the same,” since for any pair of nodes there’s a symmetry of the graph (an automorphism) that makes the first node look like the second.

ii. Prove that for every natural number $k$, the graph $Q_k$ is node-symmetric. You can assume that symmetric difference is commutative ($S \Delta T = T \Delta S$) and associative (($S \Delta T) \Delta R = S \Delta (T \Delta R)$).

This problem is much, much easier to solve if you take the time to work through some examples. Can you find an automorphism of $Q_2$ that maps $\emptyset$ to $\{0, 1\}$? To answer that question, try connecting it back to an actual square and its symmetries, see what happens to the corners, and see if you can use that to define an automorphism. Then, find an automorphism of $Q_3$ that maps $\{1\}$ to $\{0, 2\}$ by thinking about what that means in terms of symmetries of a square. You should aim to find a general pattern here before moving on.

Once you’ve found a pattern of what these automorphisms look like, find a general formula for an automorphism $\sigma$ of $Q_k$ that maps some set $S$ to some set $T$. It should be pretty short and shouldn’t require a piecewise definition. Then, write down a list of everything you need to prove in order to show that $\sigma$ is indeed an automorphism; you did this on Problem Set Four, so perhaps that would be useful as a starting point. Then, go prove all those properties. In doing so, look back to Problem Set One or Problem Set Two. Perhaps there are some nice results from there you could use here?

Oh, and there’s no need to use induction here. ☺

Once you’ve finished this problem, take a step back and admire just how far you’ve come in six weeks!

Check out this video by the legendary Carl Sagan for more about what it would be like to experience higher dimensions. Take Math 120 for more on symmetries and automorphisms, or CS149 and EE108 for some very cool applications of hypercubes.
Optional Fun Problem: Egyptian Fractions (Extra Credit)

The Fibonacci sequence mentioned in Problem One is named after Leonardo Fibonacci, an eleventh-century Italian mathematician who is credited with introducing Hindu-Arabic numerals (the number system we use today) to Europe in his book *Liber Abaci*. This book also contained an early description of the Fibonacci sequence, from which the sequence takes its name. *Liber Abaci* also described a method of writing out fractions called **Egyptian fractions**, which has been employed since ancient times; the Rhind Mathematical Papyrus, composed about 3,500 years ago in Thebes, includes several tables of fractions written out this way.

An Egyptian fraction is a sum of **distinct** fractions whose numerators are all one (these fractions are called **unit fractions**). For example, here are several Egyptian fraction representations of rational numbers:

\[
\frac{2}{3} = \frac{1}{2} + \frac{1}{6} \\
\frac{7}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120} \\
\frac{2}{85} = \frac{1}{51} + \frac{1}{255}
\]

Egyptian fractions are useful for divvying up objects fairly. For example, suppose you have two cakes to distribute to fifteen people – that is, everyone should get a \(\frac{2}{15}\) fraction of those cakes. Begin by slicing each cake into tenths and giving each person one (\(\frac{1}{10}\)). Now, take the remaining tenths you haven’t distributed and cut them into thirds, giving thirtieths of the original cake. Each person then takes one of those (\(\frac{1}{30}\)). Because \(\frac{1}{10} + \frac{1}{30} = \frac{2}{15}\), everyone gets their fair share. Pretty cool, isn’t it?

One way of finding an Egyptian fraction representation of a rational number is to use a **greedy algorithm** that works by finding the largest unit fraction at any point that can be subtracted out from the rational number. For example, to compute the fraction for \(\frac{42}{137}\), we would start off by noting that \(\frac{1}{4}\) is the largest unit fraction less than \(\frac{42}{137}\). We then say that

\[
\frac{42}{137} = \frac{1}{4} + \left( \frac{42}{137} - \frac{1}{4} \right) = \frac{1}{4} + \frac{31}{548}
\]

We then repeat this process by finding the largest unit fraction less than \(\frac{31}{548}\) and subtracting it out. This number is \(\frac{1}{18}\), so we get

\[
\frac{42}{137} = \frac{1}{4} + \left( \frac{42}{137} - \frac{1}{4} \right) = \frac{1}{4} + \frac{1}{18} + \left( \frac{31}{548} - \frac{1}{18} \right) = \frac{1}{4} + \frac{1}{18} + \frac{5}{4,932}
\]

The largest unit fraction we can subtract from \(\frac{5}{4,932}\) is \(\frac{1}{987}\):

\[
\frac{42}{137} = \frac{1}{4} + \frac{1}{18} + \left( \frac{5}{4,932} - \frac{1}{987} \right) = \frac{1}{4} + \frac{1}{18} + \frac{1}{987} + \frac{1}{1,622,628}
\]

And at this point we’re done, because the leftover fraction is itself a unit fraction.

Prove that the greedy algorithm for Egyptian fractions always terminates for any rational number \(r\) in the range \(0 < r < 1\) and always produces a valid Egyptian fraction. (A **rational number** is a real number that can be written as \(\frac{r}{q}\) for some integers \(p\) and \(q\) where \(q \neq 0\).) That is, the sum of the unit fractions should be the original number, and no unit fraction should be repeated. This shows that every rational number in the range \(0 < r < 1\) has at least one Egyptian fraction representation.