# Substitution Algebra

A new field of math and an exploration of problem solving

By Kenneth Ge and Michael Roehrl

*Could not have been done without the help of*
*Alisa Harrison, our awesome BC Calculus*
*Instructor at Edgemont HS*

## Foreword

It's amazing how you can take a few simple rules and turn them into an entire field of study. Think about counting. The most basic rule is that every time you find what you're looking for, you add one to the total count of the number of things you've found. For example, let's say you're looking for red balls. Every time you find a red ball, you'll want to add one to the total number of red balls you've seen. But from this simple idea, we get an entire field dedicated to probability and statistics, with so many beautiful properties and uses in the real world.

Substitution algebra is just like that. With just one rule and a few basic ideas, you can discover so many interesting conclusions and properties. One of the issues with school math is the sheer amount of content knowledge you have to commit to memory: different rules and properties, different theorems, different numbers, different functions. And in this scramble to memorize new tricks and rules and techniques, we lose some of the intuitive and de novo problem-solving skills we need when we're faced with a problem we haven't seen before, a problem where we just don't have clever algebraic manipulations to rely on.

Therefore, one of the key focuses of only having one rule is to force you to develop skills and techniques that can help you solve *any* problem, not just one related to substitution algebra. One of the most important skills is to be able to make observations about a problem, usually aspects that you find neat or odd. Once you have an observation, you then have to be able to determine whether it's useful, and if it is, how to use it. By keeping the rules simple, you're forced to make observations specific to each problem since you have no other well-studied techniques to fall back on. As a consequence, it's also easier to follow along with the material since there isn't really any content you need to know, so you don't have to worry about falling behind.

That's not to say substitution algebra can't be difficult or that there aren't general ideas to look out for -- in fact, being able to notice ideas like invariants (things that won't ever change in the problem) and conservation (when the amount of something--say, the number of times A appears in a sequence) is incredibly important and often overlooked in the problem-solving process. And in this fresh, new environment, it's easier to develop these skills without confusing yourself with other knowledge you may already know. It's a fresh, sterile playing field, a clean slate for anything you might want to know.

We hope that, whether you're a student who wants to be better at solving problems or a teacher who wants something fun to do at the end of the year, you'll be able to loosen up and enjoy this book!

## Introduction

The following 30 problems are mostly based on sequences of letters and a simple rule: that certain letters or sequences of letters can be replaced with other letters or sequences of letters. Note that we'll define ∅ as nothing. ∅ can be inserted into and removed from any place in a sequence of letters. The rules are represented either with = or with arrows. Notice the direction of the arrows. Unidirectional arrows mean a substitution or replacement is valid in that direction only. Once you've substituted, you can't go back the other way. Bidirectional arrows and the equal sign work both ways, so you can substitute and un-substitute as many times as you'd like. For clarity, both the equals sign and bidirectional arrows are the exact same thing, so you can interchange them at any point if you'd like. Other than that, it's all just about trying, exploring and discovering ideas and methods to get to the desired result.

Let's turn this theory into a little bit of practice first:
Suppose we have the sequence AAA.
The rules given are A → B and B ⇔ C.
This means each A can be turned into a B (but not the other way around--notice the type of arrow) because of our first rule, so AAA will become BBB. The second rule means that every B can become a C and vice versa. So really, AAA can become CCC, CCB, CBC, BCC, CBB, BCB, BBC, or BBB.

If a problem asked us to count the number of substitutions we made (like problem 16 will) to get from AAA to BBB, this would count as 3 substitutions since we applied our first rule three times. This can be easily seen in a step-by-step notation: AAA → BAA → BBA → BBB. There are 3 arrows and thus 3 steps.

Aside from that, note that, unless the problem specifies, you are never forced to make a substitution. If you have the option to do one of many substitutions, you can choose which one to use, or you could choose not to do any. For example, if the problem asked us to get from AAA to BAA, we could just apply A → B to our first A and just stop there. Even though we can apply B ⇔ C to get CAA, we're not forced to by the substitution.

But before you start, I'd like you to look at problem 26, and work on it for a minute. See how far you get. If you're stuck, that's ok. Do the problems in order, come back to problem 26, and you'll be surprised by how much easier it will be the second time around!

One final note is that although the solutions to the problems offer only one way to get the desired result, there are actually many ways to arrive at the solution to a problem. After you've solved a problem, make sure to read the solution! We might use it to discuss an important problem-solving technique. Also, if you get stuck, we've indicated what techniques might be helpful for some of the problems. Have fun!

## Problems

Problem 1 - Chaining
Given L=A and the set of substitutions A=B B=C C=D, show that A=D

Problem 2 - One, Three, Five
Given A=AAA, show that A=5A
Notationally, 5A represents AAAAA. Note that the 5 only applies to the letter that comes directly after. So, 5AB would be AAAAAB. If you wanted to express ABABABABAB, you could write 5(AB), but you're not going to encounter this notation in this textbook.

Problem 3 - Clarifying the Basics
Given a sequence of As, followed by a sequence of Bs, followed by a sequence of As, etc., find a way to decimate each sequence using the transformation AB → ∅ and BA → ∅. In substitution algebra, decimating is when you reduce a sequence down to nothing. Do this for each of the following sequences:

a. AAAABBBB
b. ABABABAB

Problem 4 - Notice the Invariant

A.

Starting sequence S = AALAB
Transformations:
A=CC, CCC=A
ALCCB=D
Ending sequence T = D

Notationally, we're often going to use S to represent the starting sequence and T to mean the ending sequence.

B. If we instead started with the sequence AALABA, determine whether or not you can still obtain the ending sequence D

Problem 5 - Induce the Nausea
Explain why you can get any number (counting number) of As under the substitution A=AA and the starting letter A.

Problem 6 - Math is an Active Field (technique: avoid fixations and open your mind)
S = DABABCCD
Transformations:
ABC → ∅
AB → ∅
CCD → ∅
Final sequence = DDD

Hint: If you want to change the number of Ds, find a substitution where two strings, with different numbers of Ds, are equal.

If you get stuck, you might want to consult the solution, since the intended way to solve this is still actively being debated. Although it seems intuitive to the three of us, depending on how you define certain operations, you might not think it makes sense.

Problem 7 - Raze the Tower (technique: visualization)
Given: a sequence of letters S made of alternating sequences of As and Bs, e.g.
5A3B2A6B2A2BABA
Transformations: A <-> AA and B <-> BB

Apply exactly one of these transformations as many times as is necessary on one contiguous sequence of As or one sequence of Bs and figure out the minimum number of hills/valleys you can get. Don't count hills and valleys at the ends.

A hill is defined as a set of exactly three contiguous sequences such that the middle sequence has strictly more letters than either of the two neighboring sequences. A valley is a set of exactly three contiguous sequences such that the middle sequence has strictly fewer letters than either of the two neighboring sequences.

For example, 2A3B1A is a hill because 3B has more letters than 2A and 3B has more letters than 1A.

Another example: in the sequence 5A4B3A4B4A, there would be exactly one valley (the 3B has fewer letters than its neighbors), and in the sequence 3A4B4A3B, there would be 0 hills or valleys because we only count valleys when there are three contiguous sequences.

Problem 8 - Shifting (technique: visualization)
Starting sequence: 25ABBA25A
Substitution: BBA <-> ABB
If you have to apply the substitution exactly 10 times, how many possible ending sequences can you get?

For example, if you were to start with the sequence BBAAA and apply the transformation twice, you could get AABBA

With visualization, the techniques are really open-ended. E.g. for problem 7, the technique would be towers, but for another, we could do holes, and for yet another, we could do colors or we could do circles

Problem 9 - Shifty Bees (technique: conservation/visualization)

Given the transformations BA -> AB and B3A -> 3AB, figure out the total number of ways to get from the starting sequence S = B8A to T = 8AB.

Problem 10 - Buffalo Buffalo (technique: simplify the problem)
Have you ever seen the sentence: “Buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo”? Even though it might look like gibberish, it actually has meaning, since the word “buffalo” can be a noun, verb, or adjective. “Buffalo” can mean the animal, “to bully,” or the place (as an adjective). But instead of “buffalo,” “buffalo,” and “buffalo,” let’s instead call these forms “horses,” “kill,” and “pink/blue/orange.” When we translate it like this, you get the sentence “pink horses (that) blue horses kill kill orange horses.” This means “a group of pink horses that other blue horses kill also end up killing the orange horses.”

Inspired by this sentence, try to solve the following problem:

> Starting string: S = 5NL
> Substitutions:
> N <-> V, NTNV -> G, NTGV -> G, NVN -> C, GVG -> C, GVN -> C, NVG -> C, NV -> C, GV -> C, CL -> ∅, T <-> ∅

Figure out the number of ways to completely decimate the starting string using the substitutions listed above. In other words, figure out how many ways there are to interpret this sentence.

Fun fact: we can up with this problem as an example of how substitution algebra can provide a framework for language analysis. This is what each of the letters stands for:

- N stands for “noun”
- V stands for “verb”
- G stands for “group,” which is a more complicated way of expressing a noun (NTNV = horses that horses kill)
- C stands for “clause”
- T stands for “that”
- L marks the end of the sentence

Problem 11 - Why is this substitution algebra? (technique: ad-hoc)
Assuming every noun is plural, you have 15 possible ways to interpret a sentence. If you’re unsure whether the 5 nouns in that sentence are singular or plural, how many ways are there to interpret that sentence? This situation could happen if you have a noun that is spelled the same if it's singular or plural, e.g." buffalo," and a verb that doesn't conjugate, e.g. "buffalo." The sentence might look like "the buffalo ate the fish."

Problem 12 - Constructions (type: constructive)
Given an arbitrary starting sequence S, come up with three or fewer substitutions that allows you to get another arbitrary ending sequence T in at least five steps minimum. The three substitutions you come up with can be unidirectional or bidirectional.

Problem 13 - Sorting and Shuffling (technique: work backwards)
Part A
Given the sequence S = 5H7V and the transformation HV <-> VH, is it possible to achieve every permutation of S? Provide an informal proof/explanation of your answer.

Part B
This time, given any permutation of any number of hills and valleys, and given the substitutions HHV <-> VHH and HVV <-> VVH, is it still possible to get any other permutation?

Problem 14 - Eccentric Hills (technique: get as far as you can, then figure out why you can't go further)
Substitution of a hill with its nth nearest valley (for example the second closest valley)

As an example take the bold **H.**
**H**VVHHVHVVHV

It's 1st closest valley is:
**H**VVHHVHVVHV
It's 2nd closest valley is:
**H**VVHHVHVVHV
It's 2nd closest valley is:
**H**VVHHVHVVHV

Given a starting sequence of hills and valleys, can you make every possible sequence using the numbers of hills and valleys given:

For example swapping with the second closest valley:
Given HVVV, can you get all variations of 1 hill and 3 valleys? No, HVVV can only become VVHV. You cannot get VHVV or VVVH.

But given HHVV, you can create HVVH by moving the second hill and VHVH by moving the first hill. From HVVH you can get VVHH by moving the hill at the beginning. From VVHH you can get HVHV by moving the hill at the end. Since we could create 6 different arrangements there are only 4!/2!/2! = 6 ways to arrange 2 valleys and 2 hills (HHVV, HVHV, HVVH, VHHV, VHVH, VVHH)

Now for the problem:
Swapping a hill with its second closest valley, can you get all combinations given HHHVV?

Problem 15 - The Vanishing K (technique: what's easy?, simplify, avoid fixations)
S = ABKCD, T = E
Substitutions:
ABCD <-> AD, AD <-> E, AB <-> ∅, KC <-> ∅, KK <-> ∅, CD <-> ∅, CK <-> AB, KCK <-> ∅

Problem 16 - Duo Decimation (technique: can you prove that it's optimal?)
Start with 16A, end with ∅
Substitutions:
∅ = A = AA

What is the fewest number of substitutions you need to get from S to T? What is your reasoning? Make sure to count all substitutions you use. Remember that you are allowed to use intermediate substitutions that you end up getting along the way. Also, if you use the same rule on multiple letters, that counts as different steps.

Problem 17 - Little Boxes (technique: invariants, visualization, think in reverse)
S = mL, where m is a natural number (integer >= 2)

Substitutions:
∅ = A, ∅ = B, ∅ = C

Count the number of ways you can use each of A, B, and C at least once. Between every two Ls, there should be at most one of each kind of letter. The order of the letters between two Ls doesn't matter. Express your answer in terms of m.

Problem 18 - It's time to BE-lieve (technique: break the problem into different steps)
Substitutions:
A = BCB, BBB = D
C = EEE, EE = C, CC = B

Prove that A = D

Problem 19 - Logic Galore (technique: experimentation)
Given NN = ∅ and NF = T, show that NT = F.

Fun fact: the N stands for "not," T for "true," and F for "false." So, if something is "not not false," then it's false, and if it's "not false," then it's true. Your goal is to show that this logic works in substitution algebra by proving the final piece: that "not true" is "false."

Problem 20 - Brackets and Parentheses (technique: translating the problem)
Given a list of left and right parentheses: )(())(((
Figure out the minimum number of parentheses you need to reverse (e.g. turning a "(" into a ")" or a ")" into a "(") for the parentheses to be balanced. A balanced set of parentheses occurs when every left parenthesis can be matched up with a right parenthesis such that all the parentheses between them are also balanced.

Problem 21 - P = NP (technique: break the problem into smaller problems)
Using the following substitutions, show that P = NP

Substitutions:

P = TFT
NN = ∅, NT = F
F = I, 4I = ∅, I = 8I
FT = TF = ∅
I = J, 2I = K
JK = P

Fun fact: we chose to make one of the substitutions involve “JK” because this joke is not a serious proof of the infamous unsolved problem P = NP. Also, the =P looks like a smiley face with a tongue.

Problem 22 - The Seven Cs

Imagine each C is one of the seven seas,
I is an island, T is a treasure chest, S is a pirate ship, and P is a pirate.
Given: C C I C C I C C C
Rules: SC ⇔ CS
CPC ⇒ CC
S ⇔ PPP
PI ⇒ TP
PT ⇔ TP

Where should an S be placed in the given sequence so that 2 Ts can be made in the fewest number of substitutions/moves? Write out all the substitutions to show how you would get the two Ts.

Problem 23 - Controversy?!

SD ⇔ DS
SBD ⇔ BBD
DB ⇔ ∅

Look at rule two: SBD ⇔ BBD. Hm. Only the beginning changes. The end stays the same. Can you prove that S ⇔ B given the other rules? Can you argue that S ⇔ B without the other rules?

Problem 24 - A tricky one

A = BB
B = AAA
C = AA
B = CC

Can you show B = CA? How about A = BC? How about C = BA?

Problem 25 - Can R disappear?

TRT = ∅
TT = ∅

Is R = ∅? Prove your answer.

Problem 26 - Screw Morality (technique: try your best)
Givens:

- A right is three lefts (R = LLL)
- A left is three rights (L = RRR)
- A right is two wrongs (R = WW)
- A wrong is two rights (W = RR)

Prove that a right is a wrong (R = W)

## Outside knowledge challenge questions

Problem 27 - Some Basic Combinatorics
Given the sequence of letters ABCDE and the set of substitutions:

A=B B=C

Count the number of total combinations of letters that you can get

Problem 28 - Recurrence, Counting, and Dynamic Programming (type: constructive)
Given a sequence of n As, write a recurrence to solve for the number of ways to swap the As with Bs such that there are no more than 3 consecutive As

E.g. if n = 3 and k = 2, you have:

AAA

And you can swap the As with Bs in the following manner:

BBB
BAB
BBA
ABB
AAB
BAA
ABA

Problem 29 - Graph Theory
Argue that the following set of substitutions is related to K3, which is the complete graph of order 3:

A=B B=C A=C

Problem 30 - One Linear Scan (technique: look at the function, what do we not care about?)
Given a sequence of letters S and a sequence of letters T, come up with an efficient way to remove all occurrences of T in S in one linear scan of the sequence from front to back. One linear scan means you should start at the first letter and scan forward through the sequence, going back and rescanning as little as possible. For example, if:

S=DABABCCD and T=ABC

Then removing all occurrences would leave you with

DD

## Solutions

Problem 1
Using all of the rules in the order they're given, A = B = C = D, so we have shown that A and D are equivalent.

Problem 2
This problem demonstrates an incredibly important concept -- that you can apply a substitution more than once! Not only that, but you can also apply a substitution to itself. Let's take a look at the problem:

So we start with A, and we have to get 5A using the substitution A = 3A. First, let's apply A = 3A to our A. Now, we have A = 3A. Using this, we can then apply A = 3A again to 3A to get A = 5A. The final sequence of substitutions looks like this:

A = AAA = AAAAA

Problem 3
This problem nicely demonstrates the idea that sequences of letters can change, allowing substitutions to be made that were previously not possible. It also shows how the nothing character, ∅, can be used.

For part a., only one substitution can be made initially, **AB** → ∅. AAA**AB**BBB becomes AAA∅BBB. ∅ is allowed to be removed (see introduction), so we actually end up with AAABBB. The sequence has changed, and it now allows us to make another substitution! AA**AB**BB → AA∅BB ⇔ AABB. AABB → A∅B ⇔ AB. AB → ∅. We have successfully removed all the characters.

For part b., there are multiple ways to remove all the letters, but the easiest is just to replace all the ABs with ∅. So, ABABABAB → ∅ABABAB → ∅∅ABAB → ∅∅∅AB → ∅∅∅∅ ⇔ ∅.

Problem 4
Part A:

The first and most important thing to notice is that the L separates the sequence into two independent parts. We know this because none of the substitutions moves or removes or duplicates the L in the middle. Since the position of the L never changes, we'll call it an invariant. In the future, we may use L to represent invariants.

Our final sequence of substitutions is as follows:

AALAB = CCCCLAB = ACLAB = CCCLAB = ALAB = ALCCB = D

Apply A = CC first, then apply CCC = A second in order to reduce the number of Cs and thus the number of As.

Part B:

No, because there is no way to obtain CCB from ABA on the right half of the string since there is no way to remove the last A. We know we can't remove the last A because there are no substitutions that lead from A to nothing. Furthermore, through the same logic we used above for the L, the B also acts as an invariant, or a wall, since there is no way to move things from one side of the B to the other.

Problem 5

We can use a technique called induction for this one. Basically, we can start with the fact that we already have a single A. Then, to get two As, we can just apply the substitution to get AA. Then, we can keep repeating this process to get any number of As that we want. Formally, if we start with n As, we can just apply A = AA to one of the As to get n+1 As. Since we start with one A as a base case, we've now shown that you can keep doing this process as many times as you want.

Problem 6

DABABCCD → DØABCCD → DØØCCD → DØØØ → D using the substitutions AB → ∅ twice and CCD → ∅ once and remembering that ∅ can be removed (it simply represents an empty space).
Similarly, DABABCCD → DABØCD → DABCD → DØD → DD using ABC → ∅.

This means we've shown that DABABCCD can become both D and DD. Arguing that D must equal DD* will yield D = DD, which is induction from problem 5.

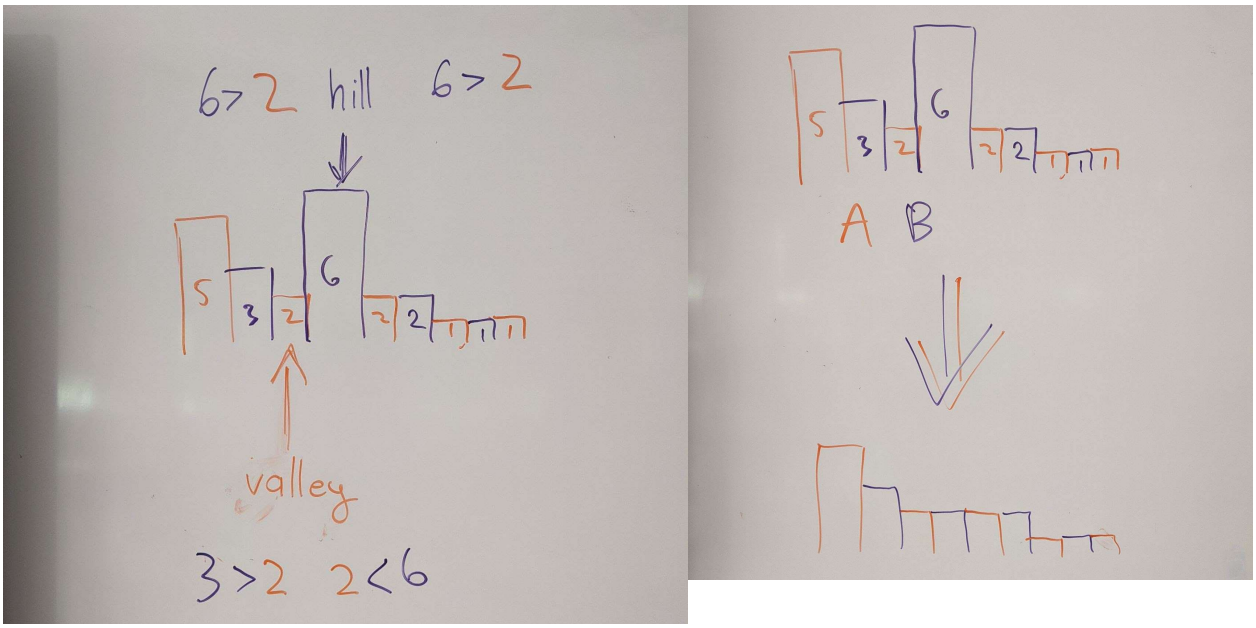
*The argument is that since DABABCCD → D and DABABCCD → DD, then D and DD are equal. Our belief is that this argument is true because "→" implies two things are the same, even if only one can be turned into the other.
E.g. let's say A → B, so A is equivalent to B. A can also be turned into B but not vice versa.

On a tangent, A being "equivalent" to B is not the same as saying "A = B" because "A = B" is the same as "A ⇔ B". "A ⇔ B" means A is BOTH equivalent to B AND the substitution works both ways.

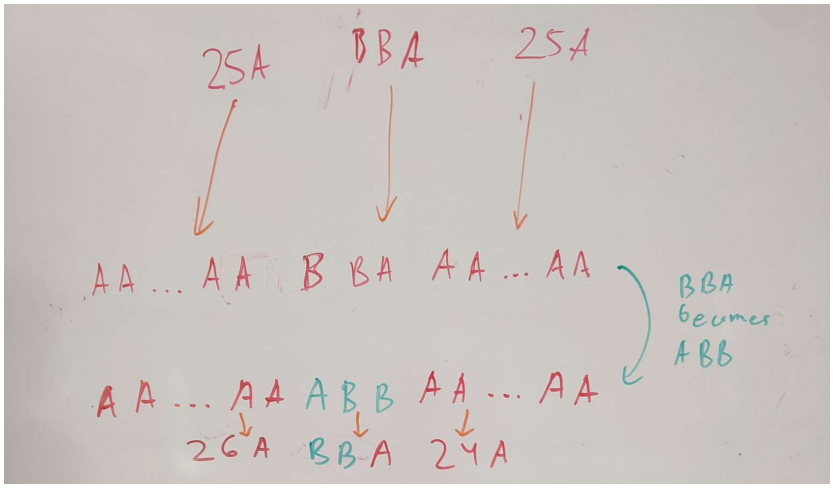
Basically, a unidirectional arrow represents an action you can do to a sequence. If you have A-> B, that means you can do the action of replacing A with B in S, but you can't do it the other way around. It's like deleting a few characters in a text file. Even though the content of the text has changed, it's still the same file. Ultimately, math is meaningless unless you define it to be a certain way. A lot of the rules (e.g. 0! = 1, 0^0 = 1, 1 is not prime or composite) we have to memorize seem arbitrary because people chose to define them that way. They seem controversial because they *were*. In the past, mathematicians spent decades debating these rules and how to define them. So, ultimately, there's nothing wrong with either argument.

Problem 7
Answer: 0

Step 1: visualize using stacks
Step 2: figure out the function of the two substitutions: that AA <-> A allows you to adjust (increase or decrease) the height of a stack. This is the reverse of the technique we used in problem 5.
Step 3: realize that there is one hill and one valley, and you can only change the height of one stack
Step 4: observe that raising a valley will only remove itself
Step 5: observe that the hill of height 6 should be lowered so its height matches one of its neighbors exactly (2)
Step 7: read the problem again to clarify some of the details, namely, that hills and valleys at the end don't count
Step 8: count the new number of hills/valleys, which is 0



Problem 8




Notice first the invariance in BB because the two Bs in our original sequence will always be present and will always be next to each other (because the rule changes both Bs at the same time and keeps them next to each other).

Notice also that the rule BBA ⇔ ABB can be visualized in your head as moving the sequence BB to the left or right by one space.
Take for example, BBAAA. If I apply the rule once, I get ABBAA, almost as if I were able to move the BB one space forward. If I apply the rule two more times, I get AAABB, as if I moved the sequence BB over two more times.

We are given 25A BBA 25A. All the As means we have a lot of room to move our BB sequence.

The problem asks for all the possible ending sequences. Let's start with a very simple case: 25A BBA 25A. This is an answer because we can apply BBA → ABB, shifting BB over to the right, followed by ABB → BBA, shifting BB back to the left. Take, for instance, AABBAA. If we cycle, our steps would look like AABBAA → AAABBA → AABBAA. We are left with what we started but have used up two moves.

Since we have 10 moves, and this maneuver takes only 2 moves, we can perform the maneuver 5 times to get 25A BBA 25A. That's one of the possible endings. I started with this case to show the special maneuver: we can "cycle" the BB back and forth until we are out of moves.

Cycling takes 2 moves, so repeated cycling must take an even number of moves. Cycling, as we have seen, will turn your current position into the final position if you have an even number of moves left. If you don't, then you can cycle until you have 1 move left and then must move BB to the left or right.

Let's try another case to better explain cycling: let's move the BB over to the right one, and then try cycling. Well, since we used the rule once, we're at 9 moves before we start cycling and our

position is: 26A BBA 24A. After cycling 4 times (each time takes up two moves), we're at 1 move and our position is *still* 26A BBA 24A (cycling preserves the sequence). Since we have 1 move left, we can either move the BB to the left or the right. Moving it to the left would give 25A BBA 25A, but we already have that one. So let's move it to the right. We get 27A BBA 23A.

Let's try moving the BB over to the right by two. We now have 27A BBA 23A and have 8 moves left. This is an even number so we can cycle and end up with 27A BBA 23A! Notice that this is the same as when we moved BB over by one initially. This is because in that case we moved BB over by one initially, then cycled, then realized we still had one move, then moved BB over to the right again. Did you notice that we actually moved BB over twice? Once initially and once at the end. The point is: it doesn't matter whether you make all your moves and then cycle or make some moves first, then cycle, then make your final moves. What is important is to realize that the number of moves you make must be even (so that you have an even number of moves to cycle as well since you start with 10 moves).

Because we can only move BB an even number of spaces to the left or right, we can move it 0, 2, 4, 6, 8, or 10 spaces (either left or right). This gives us 5 possibilities to the left, 5 to the right, and 1 final possibility (where BB stays at the center). So, our answer is 11.

Problem 9

First, notice that in both transformations, the total number of As and Bs is conserved. From one of our previous problems, we know that when we have a single B and a lot of As, the transformation BA -> AB essentially just shifts the B over by one letter. But what about the other transformation? Well, that one just shifts it over by 3! Using this information, we could use this information alone to work backwards and solve.

One way to think about it is like this: we can have at most two 3-shifts, since we have to move 8 spots. With 2 3-shifts, we have 2 1-shifts. So, there are 4!/(2!*2!) = 3! = 6 possibilities. WIth one 3-shift, we have five 1-shifts. So, we have 6 possibilities. With zero 3-shifts, we have eight 1-shifts, which leaves us with one possibility. This gives us 13 possibilities total.

However, it might be less confusing if you turned the starting sequence B8A into a 3x3 grid because the rules become very easy to visualize: BAA becomes ABA using the first rule so it is the same as moving the B one to the right. The second rule would turn BAAA into AAAB which effectively moves B down one spot.

Shifting the B to the right one spot:

BAA ABA
AAA ⇒ AAA
AAA AAA

Shifting the B down one spot:

BAA AAA
AAA ⇒ BAA
AAA AAA

The only caveat with this method is that you can shift a B at the edge of a row, which wraps it around to the next row. Here's an example of what that looks like:

AAB  AAA
AAA ⇒ BAA
AAA  AAA

The end goal is 8AB, which is when B is in the bottom right corner of our square visualization. Regardless of what technique you use, the answer is 13.

Problem 10

This problem might look really overwhelming at first, but through a few key observations, we can actually simplify this problem down to a few basic possibilities.

Observation 1: We have to end the sentence with C since we need CL -> ∅
Observation 2: The number of Ns always goes down; in other words, we're always simplifying the sentence. So, the goal of this problem should be to try to figure out how to allocate all of our Ns.
Observation 3: The substitution NTNV -> G requires exactly three Ns.
Observation 4: We can't use the substitution NTGV -> G because it takes five Ns, leaving us with a single G. We can't turn a G into a C, so we can't use this substitution.
Observation 5: We can't use NV -> C or NVN -> C because it leaves too many Ns leftover.
Observation 6: We can't make a G with just four Ns, so we can't use GV -> C.
Observation 7: Therefore, we have to use GVN -> C or NVG -> C.

Since we have two possibilities, our answer is 2.

Problem 11

You can try to think about this in terms of substitution algebra -- the sentence is composed of nouns and not nouns, and you can swap plural nouns for singular nouns and vice versa. So, you have 15 * 2^5.

Problem 12

Example: S = A, A = AA, 10A = T

Problem 13

Part A:
Yes, because this problem boils down to "can you sort any arrangement of 0s and 1s using just adjacent swaps between distinct elements"? Since the given substitution is reversible, we know that if it's possible to sort a given arrangement, it's possible to unsort it too. We know we can swap any arrangement of 0s and 1s if they are non-distinct, and since swapping elements that are the same doesn't change the ordering, these are essentially the same thing.

Part B:

No. If you think about the function of the substitution, it basically moves a hill over by 2 spaces. You could also think of it as moving a valley over by 2 spaces if you wanted to -- just pick one to move. Thinking about positioning, you number each of these hills/valleys off, like this:

| H | V | H | V | H | H | V |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Notice that you can never move a hill from an even square to an odd square, and you can never move a hill from an odd square to an even square. So, it's not possible to get every possible permutation.

Problem 14

For this problem, it's important to first identify all the problems we want to find.
Specifically, we want all the possible arrangements that can be built from 3H and 2V.
These would be:
HHHVV
HHVHV
HHVVH
HVHHV
HVHVH
HVVHH
VHHHV
VHHVH
VHVHH
VVHHH

Since we are given HHHVV, we already know we have one of the ten possible arrangements. Moving the first to its second closest hill (Swapping the bold letters: **H**HHV**V**), we get VHHVH. Instead of doing this, we could also move the second hill (Swapping the bold letters: H**H**HV**V**) and get HVHVH. That's three. Or we could move the third hill of the original sequence (Swapping the bold letters: HH**H**V**V**) to get HHVVH.

The reason we first identified all the possible arrangements was to keep track of which sequences we have and which we still need to find to prove that it's possible to get every arrangement.

I'll put * next to the sequences we already have:

HHHVV *
HHVHV
HHVVH *
HVHHV
HVHVH *
HVVHH
VHHHV

VHHVH *
VHVHH
VVHHH
Starting now with HHVVH, We could move the first, second, or third hill. Moving the third hill (Moving the bold: HH**V**V**H**) would give us HHHVV, which is our original position. So instead we'll try moving the first and second hills. Moving the first hill (Moving the bold: **H**HV**V**H) gives us VHVHH. Moving the second hill (Moving the bold: H**H**V**V**H) gives us HVVHH so we have:
HHHVV *
HHVHV
HHVVH *
HVHHV
HVHVH *
HVVHH *
VHHHV
VHHVH *
VHVHH *
VVHHH

Using HVVHH and moving the third H, we get HHVHV.
Using HVVHH and moving the first H, we get VVHHH.
Using VVHHH (which we just found), we can move the third H and get HVHHV.
Now we have all but one:
HHHVV *
HHVHV *
HHVVH *
HVHHV *
HVHVH *
HVVHH *
VHHHV
VHHVH *
VHVHH *
VVHHH *

Now comes the trickiest part: can we get VHHHV.
After struggling a bit, I came to realize something. Something that is actually very, very underwhelming actually: a hill must always replace a valley. It makes a lot of sense, but from this revelation follows the idea that before I can get VHHHV, I have to either get HVHHV, HHVHV, HHHVV, VHHVH, VHVHH, or VVHHH. Why? Well, each of these cases I derived by switching of the the hills in VHHHV with one of the valleys. This is because the rule of the problem switches hills with valleys.

But!!!! The rule doesn't just switch any hills or valleys. It swaps a hill with its *second closest* valley (in terms of distance). In each of the cases (HVHHV, HHVHV, HHHVV, VHHVH…) you can see that VHHHV can *only* be created if we swap a hill with its *closest* valley, not its second

closest. It follows that it must be impossible to get to VHHHV with the rule given. Thus, in our long and difficult search for all the possible sequences, we have arrived at the fact that it's actually impossible to get all possible sequences when starting with HHHVV.

An important point to take away is that our rule can not necessarily be reversed. VHHHV can become HHHVV by swapping the third hill for the first valley, but VHHHV cannot be made from HHHVV because there is no way to swap a hill with the first valley (it's the closest valley to all the hills).

Problem 15
Don't try to remove the K.

S = ABKCD = K = KKC = KKKCC = KCC = C = KKCKK = KK = ∅ = ABCD = AD = E = T

Problem 16
Our first important observation is that we can duplicate our As. For example, by using ∅ = AA, we can do ∅ = AA∅ = AAAA using one substitution. We can do this again through ∅ = AAAA = AAAAAAAA since we already have ∅ = AAAA from our previous step. Let's call these substitutions intermediate substitutions.

Our second observation is that it takes a minimum of 1 step to clear all 16As once we have all of our intermediate substitutions, and this case only occurs if we have 16A = ∅. In all other cases, it takes at least 2 steps to clear all 16As once we've created our intermediate substitutions, and this only occurs if we have XA = ∅, where 8 <= X <= 15. This is because if we have 8A = ∅, we can apply it twice, and if we have something like 12A = ∅ and 4A = ∅, we can apply each of these once.

So, since we can duplicate As, why not take it to its conclusion? Through some trial and error, we get ∅ = 16A in three steps. With one extra step, we can clear all 16As in S. If we stop our intermediate substitutions a step early, stopping at ∅ = 8A, we need two extra steps. Since we must have at least ∅ = 8A to clear 16A in <= 2 steps, and since we duplicated As as "efficiently" as possible (i.e. we are getting as many As as we can with as few steps as possible), we know our answer of 4 is optimal.

Problem 17
First, notice that the Ls act as walls or boxes that we have to put the letters into. This will be our visualization: we'll have m - 1 boxes and 3 items to put in each box.

Then, notice that we don't care about how the letters are arranged in each box, just which letters are present.

At this point, our first instinct might be to count the number of possible ways to arrange items in each box. That way, then since there are m - 1 boxes, we can take our answer to the (m - 1)th

power to get our answer. Following this logic, we get 2^3 = 8 arrangements per box (since each of the three items can be present or not), and our answer is 8^(m - 1).

But we still have to take into account the fact that we must use each item at least once. Unfortunately, while we could use complementary counting, solving the problem this way is tricky.

Instead, let's think about the problem in reverse: instead of figuring out how many ways there are to distribute all the items to each box, let's think about how many ways there are to distribute each item to all the boxes. Let's think about item A first. Since there are m - 1 boxes, and since A can be either present or not present in each box, we have 2 ^ (m - 1) total possibilities. But, we need to subtract one to remove the single arrangement where we don't put any As in any of the boxes. So, we actually have (2 ^ (m - 1) - 1) valid arrangements.

Using this logic, we then have (2^(m - 1) - 1) ^ 3 total arrangements because we have three items.

Problem 18

First, notice that all we need to do to solve the problem is to turn the middle C into a B. While we still should look out for a more holistic approach, we can try this idea first. So, we've broken this problem down into a simpler step, mainly, proving that C = B. Then, notice that CE = EEE using EE = C. Therefore C = EEE = CE using C = EEE. We have a letter that equals itself on top of another letter. The C can become CE. CE can become CEE if you substitute the C for CE. C = CE = CEE = CEEE = … These types of relationships can be very helpful for substitution algebra because they give you an infinite amount of letter sequences to work with. It's important to notice CEE = C because C = CEE = CC using EE = C. CC = B is also a rule. So we have proven C = B, which actually solves our problem. A = BCB = BBB = D because C = B.

Problem 19

F = ∅F = NNF = NT

One way to solve this is to experiment with reason. For example, you might try inserting NN behind F to get NNF because you know that NF = T. But, you might not want to try FNN because there's nowhere to go from there.

Problem 20

3 flips

We can translate the problem as follows where B is ")" and A is "(":
Given: S = BAABBAAA
Substitutions:

AB -> ∅, A <-> B

The problem now reads: Figure out the minimum number of A = B substitutions required to clear S and result in S = ∅

The first thing to notice is that the first parenthesis has to be a "(." In other words, since we don't have BA -> ∅, we must turn the first character into an A in order to use AB -> ∅. The second key observation is that as long as two parentheses are paired, unless they mess up the pairings of the parentheses grouped inside of them, it doesn't really matter which two parentheses you pair because switching a parenthesis has the same "cost" regardless of which one you flip. So, we can cross out AABB in its entirety.

Now, we're left with AAAA and we've flipped one parenthesis. Since each A has to be paired with a B, we must use exactly two more flips in order to decimate the entire sequence. So, we have a total of 3 flips.

Problem 21

If you were paying attention to problem 19, you should immediately realize that the substitutions NN = ∅ and NT = F allow us to manipulate True, False, and Not just like you would with any normal logic. What this means is that False is Not True, True is Not False, True is Not Not True, and False is Not Not False. The reason this works is because we can use these two substitutions to show that NF = T because T = NNT = NF.

Now back to the problem: we want to show P = NP, so let's start with P and see what we can do to change it. Seeing that P = TFT and P = JK makes me think that maybe I'll need to show P = TFT = … = NJK = NP to solve the problem. P = TFT = NFFT = NF = NI, using T = NF, FT = ∅, and F = I.

At this point, we need to turn the single I into JK and we'd be done. So somehow, I = … = JK. Notice that JK = III = 3I because J = I and K = 2I. This means we have to prove I = 3I.
Let's turn the I into 8I (using I = 8I) and then turn 3 of those Is into JK (JK = 3I). Now we have:

NI = N8I = NJK5I

Since we have 5I, but none of our solutions have 5I, we need to find ways to use 4I = ∅ and possibly I = 8I to get 5I into a sequence we can turn into ∅.

At this point, there are two ways to do it: the easier or the harder way.

The easier way: well, by playing around with 5I a bit, you can see 5I = 4I followed by I. 4I is ∅, so really 5I = I. But I = 8I given in the rules, and 8I is really 4I4I. Each 4I can become ∅ so we have 5I = I = 8I = 4I4I = ∅∅ = ∅. Thus, NJK5I = NJK∅ = NJK. Using JK = P, we get NP. Since we know P = NI = NP, we've solved the problem.

The harder way: What we're left with now is a number theory problem, where we need to find a number k (there will be k Is) such that k ≡ 5 mod 7 and k ≡ 0 mod 4. Why? We need k ≡ 0 mod 4 in order to decimate all of the remaining Is, and we have k ≡ 5 mod 7 because we're originally left with 5I at the end, and the number of Is we have will stay 5 mod 7 because we can use I = 8I

to always add 7I to the end of our sequence (which doesn't change the modulus). The number k is 40, which is divisible by 4 (40 ≡ 0 mod 4) and 40 divided by 7 has a remainder of 5 (40 ≡ 5 mod 7) We can use these substitutions:

NJK5I = NJK4I8I = NJK5I7I = NJK7I5I = NJK7I4I8I = NJK7I5I7I = NJK14I5I = … = NJK35I5I = NJK40I = NP40I = NP36IØ = NP32IØ = … = NP

So, P = NP

Problem 22

The S must be to the left of both Is because the only rule to convert an I to a T is PI ⇒ TP. Starting farther away from the first I will just take more moves to get two Ts because the substitution SC ⇔ CS will have to be used. This said, the solution is

CCSICCICCC → CCPPPICCICCC
→ CCPPTPCCICCC
→ CCPTPPCCICCC
→ CCTPPPCCICCC
→ CCTSCCICCC
→ CCTCSCICCC
→ CCTCCSICCC
→ CCTCPPPICCC
→ CCTCPPTPCCC, achieving two Ts in 9 steps.

Problem 23

First question: S → SØØ → SDBDB → DSBDB → DBBDB → ØBØ.

The second question is more to provoke some thinking. Since this is a new kind of math, the boundaries of what you can and can't do is really up to you. In this case, you could argue both ways. Some might say that you have to strictly follow the rules. If I were just given an S and rule two, there's nothing I could do with it because the rule doesn't apply. Others might say that the BD after the S doesn't change and shouldn't even be considered because of its invariance. Therefore the rule should really be S ⇔ B.

Problem 24

You can. The easiest is B = CA. To solve such problems, it's useful to take your answer and work backwards. I choose to start with CA and see that C = AA, so CA = AAA. This happens also to equal B. Thus B = AAA = CA. An important technique was to write letters in terms of a single letter only (like in this problem we wrote everything in terms of A).

The next one is trickier. BC could equal CCC using B = CC, and CCC would become AAAAAA, using the only other rule involving C (i.e. C = AA). BC = CCC = AAAAAA = BB = A, the last two steps come from using B = AAA and A = BB.

Finally, the hardest one. C = BA. BA could be many things, since three of the four rules can be applied immediately. One important thing not to forget is that we have already solved two similar

problems (B = CA and A = BC). Can we use them to our advantage? We know C = AA from our rules. Somehow, AA must become BA to prove C = BA. AA = BCA using A = BC. We are close, but there is a C in between the B and A. Using rule three we have BCA = BAAA, so we have re-written our sequence in terms of A (except the first letter). AAA = B, by rule two, so BCA = BAAA = BB. Oh no. If only we could turn one of those Bs into an A. We seem at a dead end. However, one really interesting technique in problems like this one is to use infinite sequences. Infinite sequences usually occur when a letter equals itself and some other letters. For example take D = DE. Given DE and applying the rule as many times as we want to the D, we can find DE = DEE = DEEE = DEEEE = DEEEEE = … In this problem, we don't immediately see such a rule, but maybe we can find one. Try experimenting. E.g. B = CC = AAAA = BA. Oh wow, B = BA. Therefore B = BA = BAA = BAAA = BAAAA… Importantly B = BAAA = BB. Another self-referencing rule! Therefore B = BB = BBB = BBBB = BBBBB = … Wait a minute. That means BB = B = BA! Problem solved. C = AA = BCA = BAAA = BB = B = BA.
Or, you could just do: C=AA=BBA=CCCCA=9A=3B=BA. There are many ways to arrive at the solution, and these solutions are likely not the optimal/shortest solutions nor the easiest.

Problem 25

TTRT = TØ = ØRT using rules one and two. I notice if I double RT, I get RTRT, which contains the sequence TRT, which I have in rule one. RTRT = RØ. I can also use the rule we found, T = TØ = ØRT = RT, to say RTRT = TT. Since TT = Ø by rule two, R = RØ = RTRT = TT = Ø. Like with problem 23, we have again proven something that seemed trivial: If TT is nothing, and TRT is nothing, then R, which sits right between the two Ts, should also be nothing, right? After all, if we change the problem assume R = Ø and want to find TT, we can say Ø = TRT = TØT = TT, showing that R = Ø gets us to our original givens.

Problem 26

Solution 1:
R = 3L = 9R = R8R = R4W = 3R = RW
R = RW = (RW)W = RWW = RR = W

Solution 2:
R = WW = RRRR = RL
R = RL = RRL = RRRL = … = 9R6RL = 9R3L = 3L3L = RR = W

Problem 27

Notice that all of the As, Bs, and Cs can be interchanged with each other. This means that, for each of these letters, we have three possibilities. Since we have three of them, we have 27 possibilities. Notice that the problem number is also 27...

Problem 28

One possible solution: Let f(n) be the number of ways to arrange n letters such that the nth letter is a B and we have no more than 3 consecutive As:
f(n) = f(n - 1) + f(n - 2) + f(n - 3) + f(n - 4)

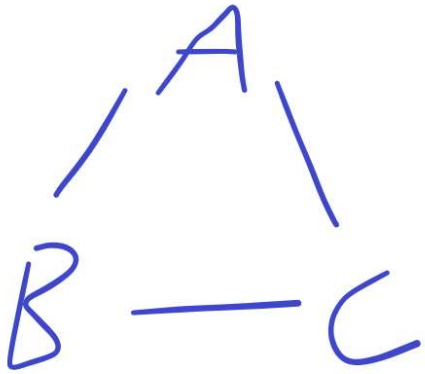
With our base cases:
f(1) = 1,  f(2) = 2, f(3) = 4, f(4) = 8

Our answer is then f(n) + f(n - 1) + f(n - 2) + f(n - 3)

To prove to yourself that it works, try using brute-force for a few small values of n. Once that gets to be too much, here's a challenge: write a computer program to compare the brute-force answer to the recursive answer for larger values of n.

Problem 29
K3 is the complete graph of order 3, meaning there are 3 vertices that are all connected to each other.

Since A can be swapped with B can be swapped with C, we could draw our substitutions like this:




Problem 30
The first thing to notice is that removing one occurrence of T in S can create another occurrence.

This means that, every time we remove an occurrence of T, we then need to check if we've created a new occurrence. If we were asked to find *any* way to remove all occurrences of T from S, we'd be done, since we can just scan the entire sequence over again every time we remove an occurrence of T.

But the problem asks us to rescan as few times as possible. Hmm… how we can avoid rescanning. Let's think about why we're rescanning in the first place: removing an occurrence of T can connect two halves of T together, creating a new occurrence of T. Let's look at the example the problem gives us:

DABABCCD

Notice that removing the blue ABC connects the two "halves" of the yellow ABC together. Notice again that as long as we know that we've seen two yellow letters, we automatically know that the next letter has to be C, which is the third

yellow letter. So, what if we kept track of ***how many*** letters we've seen, rather than ***what*** letters we've seen? That way, we don't need to rescan, since we can just keep track of how much of T we've seen at any point in time.

But first, we have to figure out how this would work. Let's take a look at this example, with colors to highlight the different ABCs:

| Sequence | A | A | B | C | B | A | B | C | C |
|---|---|---|---|---|---|---|---|---|---|
| Length of prefix | 1 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 |

What if we took this character by character, removing instances of "ABC" as we go? Let's see what this might look like.

Once we reach the end of the green ABC, we can remove that and jump back to the blue A (the character right before the green A), and continue off from there. We can repeat this process to help us combine the different blue parts together. This is what the process would look like:

Step 1: keep track of the current prefix length until we see a prefix of length 3.

| Sequence | A | A | B | C | B | A | B | C | C |
|---|---|---|---|---|---|---|---|---|---|
| Length of prefix | 1 | 1 | 2 | 3 | | | | | |
| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Step 2: remove the green ABC and realize that the blue B has a prefix length of 2 because the A that is now in the position immediately to the left has a prefix length of 1.

| Sequence | A | B | A | B | C | C |
|---|---|---|---|---|---|---|
| Length of prefix | 1 | 2 | | | | |
| Position | 1 | 2 | 3 | 4 | 5 | 6 |

Step 3: keep counting the current prefix length

| Sequence | A | B | A | B | C | C |
|---|---|---|---|---|---|---|
| Length of prefix | 1 | 2 | 1 | 2 | 3 | |
| Position | 1 | 2 | 3 | 4 | 5 | 6 |

Step 4: remove the yellow ABC and realize that the blue C has a prefix length of 3 because the B that is in the position immediately to the left has a prefix length of 2.

| Sequence | A | B | C |
|---|---|---|---|
| Length of prefix | 1 | 2 | 3 |
| Position | 1 | 2 | 3 |

Step 5: Remove the blue ABC, since you have a prefix length of 3.

Now that you know the algorithm, for an extra challenge, try to code this up!