Daniel Huan Nguyen

Morse Code Decoder Analysis

I have chosen problem #1. The problem is, given a sequence of dots and dashes, return all possible outputs for that sequence. Only alphanumeric characters will be considered.

What do we know about Morse code?

Alphanumeric Morse code can have a max code length of 5 for each character. However, it is not required to be a length of 5. A character can vary in length from 1-5 dot/dashes. This brings up an interesting recurring problem. A segment of Morse code can be decoded in several ways if there are no pauses in between to signify the start and end of a character.

**Attempt #1: Recursive Brute Force Algorithm**

Given a code like above, how would one possibly figure out all the possible codes?

Let’s start off with several simple test cases. Let O represent dOt and A represent dAsh.

Case 1: O. This is a very simple test case. There is only one way to decode this.

O -> “E”

Case 2: OO. A little bit more complicated, but let’s decode using the simplest cases. The simplest way to decode this is to just take each value as individual characters.

O O -> “EE”

Then combine the codes to see if there is a way to decode the code again.

OO -> “I”

There are no other ways to decode this code. We are done.

Case 3 : OOO. Even more complicated, but let’s stick to our methodology. Start with the simplest way to decode.

O O O -> ”EEE”

We have established before that single-length codes can only be decoded one way. So start combining codes here. Combine the last 2 codes.

O OO -> “E I”

Here is the interesting part. What if we extended the first code by 1 and decrease the 2nd code’s length by 1? Like so.

OO O -> “I E”

Let’s do the same process again. Extend the 1st code by 1 and decrease the 2nd code by 1

OOO -> “S”

Case 4 : OOOO. Even longer now, but I think there is a pattern showing up. As usual, let’s start with the simplest case. But let’s place the last character of importance in **bold.**

O O O **O** -> “E E E **E**”

Increase the 2nd to last code by 1 and decrease the last code by 1.

O O **O**O -> “E E I”

Increase the 2nd to last code by 1 and decrease the last code by 1.

O **O**O O -> “E I E”

Increase the 2nd to last code by 1 and decrease the last code by 1.

O **O**OO -> “E S”

**O**O OO -> “I I”

**O**OO O -> “S E”

**O**OOO-> “H”

Proposed algorithm for attempt #1 :

Keep in mind that this section is only going to work at this time for code lengths that are 5 or less.

1. Start with simplest case where each dot or dash is decoded individually.
2. Increase the 2nd to last code-length by 1, decrease the last code-length by 1, and decode.
3. Repeat Step 2 until there is only 1 code left.

Problems with attempt #1:

Let’s go back to Case 4 in Attempt #1. Pay close attention to this section.

1. O O O **O** -> “E E E **E**”
2. O O **O**O -> “E E I”
3. O **O**O O -> “E I E”
4. O **O**OO -> “E S”
5. **O**O OO -> “I I”
6. **O**OO O -> “S E”
7. **O**OOO-> “H”

Above listed are all the outcomes of the brute force algorithm #1.

But if anybody looks long enough, they will see that there is a missing case.

OO O O -> “I E E”

This case should have been caught before outcome #5.

The problem with this algorithm is that it will not create the simplest case with the remaining code.

The algorithm should have executed as follows :

1. O **O**OO -> “E S”
2. **O**O O O -> “I E E”
3. **O**O OO -> “I I”

See at the new line 5. It reset the code after the code with the bolded O.

Let’s extend this algorithm to 5 dots and let’s see what happens.

1. O O O O **O** -> “E E E E **E**”
2. O O O **O**O -> “E E E I”
3. O O **O**O O -> “E E I E”
4. O O **O**OO -> “E E S”
5. O **O**O OO -> “E I I”
6. O **O**OO O -> “E S E”
7. O **O**OOO-> “E H”
8. **O**O OOO -> “I S”
9. **O**OO OO -> “S I”
10. **O**OOO O -> “H E”
11. **O**OOOO -> “5”

The problem becomes extremely apparent now. The program will have to reset the Morse code back to the simplest case for the Morse code segment that follows the bolded code.

In other words,

1. O **O**O OO -> “E I I”

has to be changed to

1. O **O**O O O -> “E I E E”

And

1. **O**O OOO -> “I S”

has to be changed to

8) **O**O O O O -> “I E E E”

**Attempt #2: Revised Brute Force Algorithm for Morse code of length 5 or less**

The new algorithm must satisfy 2 conditions.

1. The algorithm must keep track of which segment of Morse code should be set to the simplest case.
2. Once entire string has been decoded, the algorithm must be able to step back through the recursion and increment the code length.

Our new algorithm has to keep track of the “current” Morse code and the subsequent Morse code. So let’s devise a recursive algorithm that sends the subsequent Morse code back into the same function until it cannot do so any longer.

Pseudocode #1

// code in this case is the string we want to decode

//answer is where the decoded answer will be kept

MorseCode(code, answer)

if (code.length() > 0) { //This checks if we are done with all decoding

for i=1 -> 5 { //Counter ensures that code segments go up to length of 5

if (code.length() >= i) { //This makes sure that there is enough code left

Currentcode = Currentcode + code[i]

if (decode (Currentcode) != "")

tempAnswer = answer + decode(Currentcode)

MorseCode(code.substring(i+1, code.length()), tempAnswer)

tempAsnwer = "" //reset tempAnswer

}else {break}

}

}else {print answer} //step back up from recursion

Example 1: Use case 4 from attempt #1

I will be going through the algorithm methodically and show each step.

The underline section will represent the section that has been decoded.

The **bolded** section represents what is being sent into the next recursive call.

The spaces will represent how the codes are separated in each recursive call.

Let O represent dOt and A represent dAsh.

Decode OOOO

O **OOO**

O O **OO**

O O O **O**

O O O O -> Print “E E E E”

O O OO -> Print “E E I”

O OO **O**

O OO O -> Print “E I E”

O OOO -> Print “E S”

OO **OO**

OO O **O**

OO O O -> Print “I E E”

­OO OO -> Print “I I”

OOO **O**

OOO O -> Print “S E”

OOOO -> Print “H”

Decode OOOOO

O **OOOO**

O O **OOO**

O O O **OO**

O O O O **O**

O O O O O -> Print “E E E E E”

O O O OO -> Print “E E E I”

O O OO **O**

O O OO O -> Print “E E I E”

O O OOO -> Print “E E S”

O OO **OO**

O OO O **O**

O OO O O -> Print “E I E E”

­ O OO OO -> Print “E I I”

O OOO **O**

O OOO O -> Print “E S E”

O OOOO -> Print “E H”

OO **OOO**

OO O **OO**

OO O O **O**

OO O O O -> Print "I E E E"

OO O OO -> Print "I E I"

OO OO **O**

OO OO O -> Print "I I E"

OO OOO -> Print "I S"

OOO **OO**

OOO O **O**

OOO O O -> Print "S E E"

OOO OO -> Print "S I"

OOOO **O**

OOOO O -> Print "H E"

OOOOO -> Print "5"

Does the new algorithm satisfy the 2 conditions listed?

1. The new algorithm does keep track of which segment of Morse code should be set to the simplest case. Every time the function MorseCode() is recursively called, it automatically sets the existing code to the simplest case.
2. The algorithm is able to step back through the recursion and increment the code length. By including the "if (code.length() > 0)", it will print the current decoded string and go back up recursively to the 2nd to last MorseCode() and increment the counter/code length.

Both of these conditions are necessary in order to have all possible solutions. This algorithm satisfies both conditions.

Let S be the solution set for all possible combinations, and B the subset with the solutions generated by the Brute-Force algorithm. B contains all the sequences that in total add up to the length of the input, and were generated in order by adding one at a time while maintaining the total sum equal to the length of the original input. For S to have a solution not included in B, such solution must not add up to the total length of the input, and thus, by contradiction, we prove that S and B are equivalents

Analyzing the Algorithm

Now that the given algorithm will produce all results, what is the time complexity?

I am making some assumptions when calculating time complexity.

1. The decoding of individual characters is O(1). In other words, the decode() function from the pseudocode will run in constant time.

By looking at the pseudocode, we can build a recurrence relation.

Let n be the length of the Morse code that we are decoding and D be our function.

**D(n) = D(n-1) + D(n-2) + D(n-3) + D(n-4) + D(n-5) + n**

This section "**D(n-1) + D(n-2) + D(n-3) + D(n-4) + D(n-5)**" represents the for loop. The for-loop iterates 5 times and takes 1 "code-length" away for each iteration.

D(0) = 0, D(1) = 1, D(2) = 3, D(3) = 7, D(4) = 15, D(5) = 31

By using trial and error, I have deduced that **D(n) = 2^n - 1**.

Unanswered Questions

In the recurrence relation, D(n) = D(n-1) + D(n-2) + D(n-3) + D(n-4) + D(n-5) + n

I have yet to figure out how "n" is added to the calculation. This recurrence relation has not yet been proven correct either. It has been based off the amount of steps required from all the examples used. Decoding OOOO took 15 steps. Decoding OOOOO took 31 steps.

**Solving the Recurrence**

First, we should find out how many different solutions there are for each length of Morse code. This should tell us the necessary amount of iterations that are required to make this program work properly. I only went up to Morse code length 5 in my first analysis. According to the trend, the maximum amount of solutions to any given Morse code length is 2^(n-1), where n is the length of the Morse code. However, does the trend hold even after Morse code length 5?

For this example, I will only be using dots because O, OO, OOO, OOOO, and OOOOO are all decodable. This guarantees that every recursive step in the program will decode a legitimate segment of Morse code. The purpose of this is to explore the maximum amount of solutions in the worst case scenario.



The results starts diverging at n=6.

And the gap gets larger as we go on.

So what is causing this divergence and why is it occurring for n>5?

Let's first understand the idea of counting the number of solutions by hand. When searching for all Morse code solutions, you have to build all possible solutions from previous solutions. Let me explain with a small example.

The total amount of solutions is determined by how many different ways the smaller segments can be decoded. The way the algorithm works is by taking a code segments of size 1-5, decoding them, and then sending the remaining Morse code into the next recursive step. So depending on the size of the original code segment, the number of different solutions each original code segment varies.

For example : Decode OOOO

O **OOO 1st Morse code segment is size 1 and the rest of code is size 3**

O O **OO**

O O O **O**

O O O O -> Print “E E E E”

O O OO -> Print “E E I”

O OO **O**

O OO O -> Print “E I E”

O OOO -> Print “E S” **Produced 4 different results.**

OO **OO 2st Morse code segment is size 2 and the rest of code is size 2**

OO O **O**

OO O O -> Print “I E E”

­OO OO -> Print “I I” **Produced 2 different results**

OOO **O 3st Morse code segment is size 3 and the rest of code is size 1**

OOO O -> Print “S E” **Produced 1 different result**

OOOO -> Print “H” **4th Morse code segment is size 4 and produced 1 different result by being uniquely decodable. Did not send anything into the next recursive call.**

We should note that the total number of different solutions is dependent on 2 things.

1. The size of the remaining Morse code after the first cuts.
2. If the entire Morse code can be decoded into one single character.

Use OOOOO. Let the underlined section represent the first code segment and the **bolded** section be the Morse code sent into the next recursive step.

O**OOOO** : The bolded section has 8 ways of being decoded.

OO**OOO** : The bolded section has 4 ways of being decoded.

OOO**OO** : The bolded section has 2 ways of being decoded.

OOOO**O** : The bolded section has 1 way of being decoded.

OOOOO : The entire Morse code can be decoded into one single character and still produces one way of being uniquely decoded.

So the total different ways of decoding OOOOO is 16.

Let's try to decode OOOOOO.

Remember that the first cuts must only be of size 1-5.



Notice that we did not make a cut OOOOOO. Remember, the first cuts can only be of sizes 1-5. Therefore OOOOOO was never made. Because of this, we cannot add 1 for being uniquely decodable to 1 character. That is why the result is 31 different solutions instead of 32 different solutions.

There is another way to think about the total number of solutions. Once the length of the Morse code is beyond 5, we only have to worry about the first condition, the size of the remaining Morse code after the first cuts. Since there are only 5 cuts from size 1 to 5, then we can represent the total number of solutions by the size of the rest of the Morse code after the 5 initial cuts.

Let **S** be the function that tells us the maximum number of ways a Morse code length **n** can be decoded.

Base cases :

**S(1) = 1, S(2) = 2, S(3) = 4, S(4) = 8, and S(5) = 16**

Recurrence Relation :

**S(n) = S(n-1) + S(n-2) + S(n-3) + S(n-4) + S(n-5) for n>5**



To get a feel for where the lower bound could be, I did an exponential regression on the number of solutions for n > 5. I ended up with θ(1.96605^n). However, when I actually tried to solve this recurrence using the characteristic equation, there was only one real root of approximately 1.96595.

Therefore, the non-trivial lower bound is **Ω(1.96595^n)**. This represents the minimum amount of iterations needed for any given Morse code decoder to print out the solutions.

**Testing my algorithm**

When testing my algorithm, I was concerned with the total amount of recursive calls needed to complete the program. If the number of recursive calls O(2^n) and Ω(1.96595^n), then my program meets the requirements.

Let's go back to my original time complexity equation.

Let D be the function of n that returns the amount of recursive call used in my program.

**D(n) = D(n-1) + D(n-2) + D(n-3) + D(n-4) + D(n-5) + n**

Base cases : D(1)=1, D(2)=3, D(3)=7, D(4)=15, D(5)=31

Therefore D(n) = O(2^n)

If I can show that the number of recursive calls is less than or equal to the above recurrence relation, then the program that I have made correlates very closely to the lower bound.



After testing my program to Morse codes length 22, my data shows that the number of recursive calls is below D(n) and continues to diverge. Therefore, my program runs in O(2^n) Since I previously had shown that the lower bound for this algorithm is Ω(1.96595^n), my brute force algorithm is Ω(1.96595^n) and O(2^n).

**Possible Improvements**

The one major improvement that could be implemented is a look-up table. This table would hold all previous decoded segments of Morse code so that it would not have to decode the same section of Morse code every time. This would decrease the total amount of recursive calls, but increase the amount of memory used at any one given time. Memory space would have to be dedicated for holding all the possible decoded messages for all the different Morse code segments. However, the time complexity would remain the same because the lower limit on the number of different decoded messages remains the same.

I can also expand upon the project by trying out much longer codes. Of course I have to consider the implication that these longer codes will result in a much longer computation. But I believe that if the codes are always cut to a sufficiently small enough length, then I can still get the answers i need in a time/space efficient manner. Let 20 be the largest code size for now.