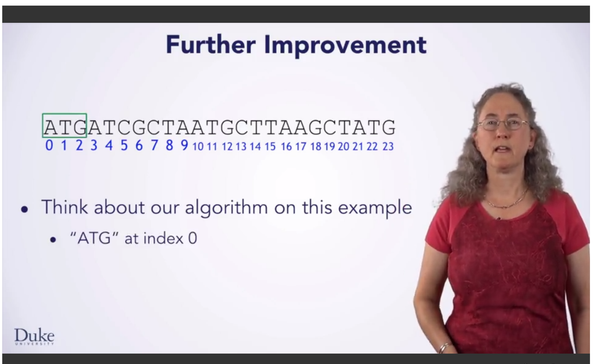

While Loops

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
As you have been learning about strings you have been improving on your algorithm to find genes in DNA. However, lets take a moment to think about what your algorithm would do on this string.

0:15
You will find the start codon ATG at index 0, then it will find the stop codon, TAA at index 8. It will then check if the distance between them, which is eight, is a multiple of three. Because eight is not a multiple of three, your algorithm will conclude that this is not a valid gene. Between the ATG and TAA there is one full codon, ATC, and two-thirds of another code on, GC. But if you were to keep looking past this TAA you would find another TAA at index 15. Now the distance between the ATG and TAA is 15 which is a multiple of three, so this is a valid gene.

1:01
The first TAA that we found was not actually a codon, but rather pieces of two adjacent codons, the T from GCT, and the AA from an AAT. Your next improvement to the gene finding algorithm is to add this functionality, to make the algorithm keep looking until it finds a stop codon that is a multiple of three away from the start codon. Having just worked that example let us now do step two of the seven step process and write down what we just did.

1:32
The first thing we did was to find the ATG. Then we found the first occurrence of TAA after the ATG which was right here at index eight. Then we checked that the distance between them was a multiple of three or not.

1:51
In this case, it was not. So we found the next TAA after this first one. The second one is right here at index15. Then we checked if the distance between this TAA and the start codon was a multiple of three. It was, so that all of the sub string from 0 up to 18 was our answer. In this particular set of steps, we checked in two places to see if the distance was a multiple of three. If this works in the general case, you can just implement this algorithm with familiar if else statements. However, do we always only need to check twice?

2:32
Let's look at a different DNA string. With this DNA string we would need to check three times. The first two TAAs are not a multiple of three away from the start codon but the third one is. Would checking three times be enough?

2:53
Could we have to check 4 times, 5, 10, 50 times? This raises the question of how many times we have to check in general. And the answer is that we cannot think of a particular number of times. Even if you wrote 50 if else statements, we could come up with a DNA string that has more than 50 TAAs that are not a multiple of three away from the start codon before finding a valid one.

3:20
Instead we need to write our algorithm so that it repeats the checking however many times it needs to.

3:28
As you have seen before, repetition in your algorithm will turn into a loop when you translate the algorithm into code.

3:38
To express your algorithm with repetition, you will need to make the repetitive steps the same and figure out what to loop over.

3:52
Previously you have seen four loops, which iterate over the elements in some interval such as pixels in an image.

4:01
Now, you're going to learn about a new kind of loop known as a While Loop, which lets you iterate as long as some condition holds.

4:11
Before we try to generalize these steps by finding repetition, let's be a bit more precise about what we did.

4:19
We found the first ATG at index zero. For the first TAA we started looking at index three and found it at index eight.

4:32
We checked if eight was a multiple of three, it wasn't. So we started looking at index nine for the second TAA and we found it at index 15. We checked if 15 was a multiple of 3, it was, so everything between was our answer.

4:53
Now, let us take these steps and generalize them.

4:59
We looked for ATG here, why was that? We always want to look for ATG because it is the start codon. What about the fact that we found it at index zero? We're not always going to find it at index zero, however we are going to want to use that information, so let's give that a name. When we turn this into code, what will be a variable that we will assign to this in this line and then use later?

5:35
In particular, we'll call it StartIndex.

5:40
What about looking for TAA?

5:43
We will always want to look for TAA, since that is the stop codon. Will we always start at index three? Probably not. Why did we start at index three here? We started there because it was right after the start codon that we found. In the general case, this would be startIndex + 3. We won't always find it at index eight either, so let's give that a name too.

6:16
We'll call it currIndex. Let's also be a bit more specific about the distance between them. It is currIndex minus startIndex.

6:32
Which happens in both of these steps.

6:38
Next, you aren't always going to start looking from index nine, but why do we start at nine here?

6:46
If you look back at where we work the problem and wrote down our steps, we started at nine because the previous one started at eight. In our generalized algorithm, we named the previous location currIndex, so we can start from currIndex + 1. We also should name the location where we found it.

7:08
Should we give it a new name such as nextIndex? Or should we just update an existing name such as, currIndex?

7:23
In this case, we want to update currIndex since that represents where we have found the most recent TAA. If you did not realize this right away and gave it a different name, you would realize it later on as you try to make the steps uniform so that you can express the repetition. Finally, we'll generalize the last step to just indicate that the text between them is the text from start index to currIndex + 3.

7:55
Now these steps look repetitive.

7:59
The repetition maybe a bit hard to see since it only happens twice. But if you wrote down the steps for strings with more TAAs that dont work you would see that you do these steps over and over again.

8:14
To make this repetitive let's write it down like this. Notice that steps four, five, and six are what we will repeat. We've slightly adjusted the steps from before to reflect the choice we were making in step four and the two possible outcomes in steps five, and six. However, we have left the conditions under which we'll repeat these steps blank here.

8:36
How do we know when to stop repeating them?

8:40
Also what would you do after you stopped repeating this loop?

8:45
We would stop if we run out of TAAs. If that happened currIndex would be minus 1, which you know from having learned that you get minus 1 when you cannot find something in a string.

8:59
If you were to encounter this case, it would mean that there is no valid gene in the string.

9:05
So you should give an answer of the empty string.

9:09
If you did not see this right off, what could you do to figure it out? You should work more examples until you understand the pattern.

9:19
Now, your algorithm is generalized.

9:23
But you'll need to learn about while loops before you can translate this into code, thank you.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?