Problem Statement

Alice is trying to send a message of length n to Bob. However, alice knows that if she sends a message, k packets will be corrupted. Alice knows that by Berlekamp-Welch, she should send $n + 2k$ packets in to ensure that the message can be decoded. Show that $n + 2k$ is the minimal number of packets to send to be able to decode the message (in other words, if Alice sends fewer than $n+2k$ packets, then the message could potentially not be decoded)

Solution

First, I'll start by noting that it's easy to get a problem like this and immediately feel stuck because although you clearly know that the problem involves the Berlekamp-Welch algorithm in some way, the problem doesn't really tell us how it's supposed to be used.

Therefore, the best way to go about solving this is to go through every step of the Berlekamp-Welch algorithm, and see where exactly do the number of packets actually matter. This will be the key to the solution, and in the meantime we also gain useful insight into how the algorithm works, so hopefully similar problems won't be as hard in the future. As I'll get into later, this is also the advice I'd give to literally any problem you encounter.

Starting off, we know that $P(x)$ is a degree $n-1$ polynomial with n unknown coefficients. Then, Alice sends out packets $P(1) = r_1, P(2) = r_2, \dots, P(m) = r_m$ to Bob. (suppose we don't know that $n + 2k$ packets are required at this stage, so we'll say Alice sends m packets.)

Then, Berlekamp-Welch defines the *error locator polynomial* $E(x)$, which is defined as:

$$E(x) = (x - e_1)(x - e_2)\cdots(x - e_k)$$

where $e_1, e_2, \dots, e_k$ are the locations of the errors. Since there are k parentheses here (one for each error), then this means that $E(x)$ is a degree k polynomial.

Comment: One thing you might question at this point is the purpose of the error locator polynomial to begin with: why are we defining something that we literally have no information about? The answer is that $E(x)$ has a very special property: it will return 0 whenever the packet sent is corrupted, which is crucial in the next step.

A natural follow up question to this is: how do we know that $E(x) = 0$ at an error? Well, that's just how we define it! The reason we are allowed to define such a polynomial is that this form of the polynomial makes no assumption of the errors. In other words, this form is just as general as writing $E(x) = b_k x^k + b_{k-1}x^{k-1} \cdots + b_0$, then imposing the condition of $E(x) = 0$ at the errors, but once we do so we know that it can be factored into the above form *because* the errors are the zeroes.

This explanation might be confusing at first, but it is arguably one of the most important parts of the Berlekamp-Welch algorithm.

Now for the most crucial step, and the one that tells us why $n + 2k$ packets are required: Berlekamp-Welch claims that

$$P(i)E(i) = r_i E(i).$$

Before I go any further with the solution, I want to take a minute to actually look at why this equation is even valid to begin with; I believe this to be a valuable step in understanding the algorithm, and will no doubt be useful to solving future problems. To show its validity, we consider two cases – whether the received packet i is an error:

- If the packet i is an error, then we know that $E(i)$ will be 0 by the definition of $E(x)$, and since $E(i)$ exists on both sides of the equation, then we have $0 = 0$, so equality is preserved.
- If the packet i is not an error, then $E(i)$ will return the same value on both sides, and so they will cancel, leaving us with $P(i) = r_i$, which is clearly true.

This covers all possible cases, and since equality holds in both of them, this equation does make sense for all packets i.

Now that we fully understand what this equation tells us, now let's examine the left hand side $P(i)E(i)$ more closely. We know from earlier that $P(x)$ is a polynomial of degree $n-1$, and $E(x)$ is a polynomial of degree k. Now comes the first of two crucial observations to solve this problem: **when we multiply two polynomials together, the combined polynomial has a degree equal to the sum of the two individual degrees.** To see why this is true, we know that the largest power of x in $P(x)$ is x^{n-1} (degree $n-1$), and in $E(x)$ is x^k (degree k), so when these two terms multiply together, then we get an x^{n+k-1} term, and so therefore the degree of the product is $n+k-1$.

Now, since the left hand side is a polynomial of degree $n+k-1$, so we can write it as:

$$P(x)E(x) = a_{n+k-1}x^{n+k-1} + a_{n+k-2}x^{n+k-2} + \cdots + a_1 x + a_0$$

If we count these terms, there are a total of $n+k$ unknown coefficients, since there is an a_0 term. These coefficients are still unknown to us since we don't know $E(x)$, so we'll have to find a way to solve them.

Aside: Berlekamp-Welch defines $Q(x) \equiv P(x)E(x)$, which is a good definition to have if we needed to actually solve for $P(x)$, but as we'll see it's not really required in this problem, though I'll mention it at the end.

Comment: One common mistake to make here is to forget that because we have a constant a_0 term, there are $n+k$ coefficients instead of $n+k-1$ of them. One way to avoid this is to look explicitly at where you're starting the counting and where you're ending.

Now that you've seen me do this with the left hand side, I encourage you to try and do the same thing with the right hand side: write down $E(x)$ in its most general polynomial form, then count the number of unknown coefficients.

Now, let's look at the right hand side, $r_i E(i)$, and specifically $E(x)$. Since we know that $E(x)$ is a degree k polynomial from before, then we know that we can write:

$$E(x) = b_k x^k + b_{k-1}x^{k-1} + \cdots + b_1 x + b_0$$

which gives us another set of $k+1$ unknown coefficients, but since we know that $E(x)$ is also written as $E(x) = (x-e_1)\cdots(x-e_k)$, then we can see that the coefficient of the x^k term is equal to 1. The way we see this is that the only way to obtain an x^k term is by multiplying all the x's together, but since they all have a leading coefficient of 1 then the resulting polynomial will also have a coefficient of 1. Because of this, there's one less coefficient we need to solve, giving us a total of k unknown coefficients here. Putting this together with the left hand side, there are now a total of $n+2k$ unknown coefficients. And here we see where the $n+2k$ appears.

Comment: Here, ponder about how we might be able to determine these $n+2k$ coefficients. What information does Bob have that we haven't really used yet?

You should realize here that while we have used the symbolic form of all the equations Bob needs, we haven't actually used the packets themselves! That is the missing piece of the puzzle.

Now, putting it all together, we input each packet we obtained (i and r_i) into $P(i)E(i) = r_iE(i)$. Then, we make the second crucial observation: **each packet that we plug in will return us an equation consisting of the unknown coefficients,** meaning that we now have a large system of equations with $n + 2k$ unknowns.

As we know, for a system of $n+2k$ unknown coefficients, we require at least $n+2k$ equations to solve. Since each packet we receive gives us one of these equations, then we require at least $n + 2k$ packets in order to solve for the value of all these coefficients.

At this point, I would basically consider the problem done, since we've essentially shown why $n+2k$ packets are needed – specifically, **we need this many because it's the minimum number of packets required to guarantee that we can solve for all $n+2k$ coefficients.** Then, after these coefficients are found, we can divide the polynomial on the left hand side by $E(x)$ to get $P(x)$, which finally allows us to decode the message. In the algorithm, this division is formally written as the step:

$$P(x) = \frac{Q(x)}{E(x)} = \frac{P(x)E(x)}{E(x)}$$

this is where we see $Q(x) \equiv P(x)E(x)$ pop back up again. If we *didn't* have all the coefficients determined (i.e. less than $n+2k$ packets), then this polynomial division step would be impossible since there are unknown coefficients, and thus we likely won't be able to decode the message. That isn't to say it's impossible – you could theoretically get a system that requires fewer equations to solve – but this isn't true in general.

Takeaways

Hopefully this problem gives you a deeper understanding of Berlekamp-Welch as an algorithm, making future problems pertaining to this algorithm will come easier as well. As a guideline, if a problem asks you to prove something about Berlekamp-Welch, just run through the algorithm exactly in the way described above, and you'll likely stumble upon the solution naturally. If it asks you to execute the algorithm directly, the steps are also described above, albeit symbolically.

As a general rule of thumb for basically any problem you encounter in CS70 however: Try to distill the problem down into a mathematical proposition if one isn't already given to you, then try to prove that proposition true. In this case, we weren't asked to prove an explicit formula, but with some work we turned the problem statement of requiring $n + 2k$ into a mathematical problem of solving systems of equations. Then from there, we were able to see why $n + 2k$ packets are required. This approach will help you solve all kinds of proofs you encounter not just in CS70, but in future CS courses as well.