CS1112-202/206/207–Spring 2010

Section 9 Solution

Friday, April 2

1 Reverse Complement

There are two tasks we have to do: reverse and complement. Let `strand` be the result of the reverse complement of `dna`, whose length is `len`. For character i in `strand`, we need to get character `len-i+1` in `dna` and take the complement of it. The resulting code is as follows:

```
function strand=rComplement(dna)
% Return the reverse complement of a DNA strand.
% Assume that dna contains only the letters 'A','T','C', and 'G'.
% If dna is the empty vector, return the empty vector.

len=length(dna);

strand='';

for i=1:len
    strand(i)=dna(len-i+1);

    if strand(i)=='A'
        strand(i)='T';
    elseif strand(i)=='T'
        strand(i)='A';
    elseif strand(i)=='C'
        strand(i)='G';
    else
        strand(i)='C';
    end
end
```

If you have a hard time understanding why the index needs to be `len-i+1`, try some examples. If $i = 1$, then it should be from len. If $i = 2$, then it should be from $len - 1$. Pick up the pattern and you will get a desired expression.

2 Counting a DNA Pattern

What do we have to do here? We know that we can use `strcmp` to compare two strings. Now, that function is useful only when the two strings we give it are of the same length. (How can two strings be the same if they have different lengths?) Hence, what we have to do is to extract all the possible substrings of `dna` that have the same length as that of `p`. How many such substrings are there? Let lp be the length of `p`. If the substring begins at position 1, then it ends at position lp. If the substring begins at position 2, then it ends at position $lp + 1$. Now, what position should the last substring begin so that it ends at position ld, where ld is the length of `dna`? Picking up the pattern from the two examples above, the last substring should begin at position $ln - lp + 1$.

Now that we have the number of substrings, we know how many times the loop should iterate. For iteration i, i.e., when the substring begins at position i, we still have to extract this substring correctly. Let us go back to the two examples above. We now ask a different question: If a substring begins at position i, which position should it ends? Again, picking up the pattern, we arrive at the answer $i + lp - 1$. Therefore, the substring is `dna(i:i+lp-1)`. We can now compare this substring to `p` using `strcmp`. If they match, then increment the count. That's it!

3 Counting a DNA Pattern—Challenge Edition!

It seems that the crux of the solution to the last problem is `strcmp(dna(i:i+lp-1),p)`. How can we write the same function without using `strcmp`? The answer is simple: Because `strcmp` is a built-in function, and now we can't use it, we just need to implement one ourselves. Therefore, we need to understand how `strcmp` works. It takes two strings. If their lengths are not the same, then simply return 0. Otherwise, we have to go through each character of both strings. We try to find a mismatch, i.e., a position such that the characters of both strings in that position are different. If there is such a mismatch, then the two strings are not equal, and we can return 0. If our trial fails, i.e., there is no mismatch, then we can declare that both strings are identical and return 1.

The attempt to find a mismatch can terminate as soon as we find one. This immediately suggests a `while` loop structure for the code, as a `for` loop will be inefficient. (Think about it: If the mismatch is at the very first character, and both strings are really long, why spend time checking the rest?) Moreover, because we are comparing two strings of the same length, we do not have to verify whether they have the same length.

Thus, the body of the `for` loop in the last problem will now look like the following:

```
d=dna(i:i+lp-1);  % the substring we are going to compare with p
pos=1;  % start comparing from the first position of both strings
while pos<=lp && d(pos)==p(pos)  % still have more to check
                                 % and no mismatch at this position
    pos=pos+1;  % move to next position
end

if pos>lp  % the while loop did not terminate because of a mismatch
           % so there is no mismatch after all
    count=count+1;  % pattern found; increment the count
end
```