CS1112-206/211/212–Fall 2009

Section 9 Solution

Friday, October 30

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., 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 Two-Dimensional Interpolation

This function is easy to implement if we know how to access particular parts of a matrix efficiently. One solution is posted on the course webpage. Another solution is attached below. Instead of creating an intermediate matrix, this version creates the final matrix right away and processes as if the matrix were the intermediate one when "calculating odd rows."

```
function newM=Interpolate2D(M)
% Perform 2-d interpolation on the real-valued data in nr-by-nc matrix M.
% The interpolated data are added between existing data points so newM is
%  (2*nr-1)-by-(2*nc-1).  Use the simple average as the interpolated value.

[nr,nc]=size(M);
nnr=2*nr-1;
nnc=2*nc-1;
newM=zeros(nnr,nnc);

% assign original values to odd rows, odd columns
for i=1:nr
    for j=1:nc
        newM(2*i-1,2*j-1)=M(i,j);
    end
end

% calculate odd rows, even columns
for i=1:2:nnr
    for j=2:2:nnc
        newM(i,j)=(newM(i,j-1)+newM(i,j+1))/2;
    end
end

% calculate even rows
for i=2:2:nnr
    for j=1:nnc
        newM(i,j)=(newM(i-1,j)+newM(i+1,j))/2;
    end
end
```