CMPS 102 — Fall 2018 – Homework 4

Alyssa Melton

I have read and agree to the collaboration policy.

Collaborators: none

Solution to Problem 2

Say we are given two sequences, A and B, with A have m edges and B having n edges.
For simplicity, let's say A is of the same length, or longer than B.
We will store the elements of A and B in arrays the size of the longest sequence, m.
If one array is smaller than m, fill the end values with $null$.

For two sequences s and t, we say t is a subsequence of s if all elements in t are in s and these elements appear in t in the same order the do in s.
We want to find a subsequence that is shared by A and B, and is of maximal length.

To do this, we will keep an array called $subsequence$ of size n (the smaller of the two sequences).
Keep another array called $tempsequence$ of size n. This will be used for comparison.

for i (0 to m-1): {
    for j (0 to n-1): {
        $index = (i + j)\%m$. // i is the place in B that the original B starts.
        if $index = 0$: { // cannot have subsequences out of order.
            if (# non-null elements in $tempsequence$) > (# non-null elements in $subsequence$): {
                $subsequence = tempsequence$.
            }
            reset all elements of $tempsequence$ back to $null$.
        }
        if $A[index] = B[index]$: add this element to the back of $tempsequence$.
    }
    if (# non-null elements in $tempsequence$) > (# non-null elements in $subsequence$): {
        $subsequence = tempsequence$.
    }
    reset all elements of $tempsequence$ back to $null$.
    Rotate all elements of B to the right by 1.
    The element the falls off the end becomes the $B[0]$.
}
return $subsequence$.

A and B in each iteration of the while loop will look like this:

A:	A	A	T	A	G	T	T	G	G	T	A	G
B:	A	A	T	A	G	T	T	G	G	null	null	null
B:	null	A	A	T	A	G	T	T	G	G	null	null
B:	null	null	A	A	T	A	G	T	T	G	G	null
B:	null	null	null	A	A	T	A	G	T	T	G	G
B:	G	null	null	null	A	A	T	A	G	T	T	G
B:	G	G	null	null	null	A	A	T	A	G	T	T

We are checking vertically for matches, and the most number of matches give the longest substring.

Claim 1. *This algorithm correctly outputs the maximal substring.*

Proof. Assume the algorithm did not output the maximal substring. Then there must be some substring s that was not considered. This however could not be, since the algorithm literally checks every possible shift while keeping the elements in order. It could output a substring longer than one that is valid. The only time that could happen is when shifting and wrapping back to the beginning. However, this can not happen either, because the outer loop keeps track of where to start in the newly shifted array so that we start at the beginning of the shorter sequence. When wrapping to the beginning, there is a check for when index=0, as to reset the subsequence array. □

Claim 2. *The run time is $O(mn)$ and the space-complexity is $O(m)$*

Proof. The outer loop runs through 0 to m, and the inner loop runs 0 to n, hence, $O(mn)$. □