CS161 Programming Project

Jason van der Merwe – Jasonvdm@stanford.edu – 05719899
Bridge Eimon – beimon@stanford.edu – 05716372

May 27, 2014

Theory

(a) We can modify the input strings given to CLCS such that CLCS outputs the solution to the LCS problem. Because CLCS is cylical in nature, that is, the end of the string flows through to the beginning of the string, we have to make sure that when solving for the LCS, CLCS doesn't cycle. We can achieve this by modifying our alphabet so that we label each instance of a letter with the order of occurence. For instance, when we see the first letter a, we label it a_1, if the a we see is the third occurence of a, then we label it a_3. The string $asdaad$ would be labeled $a_1s_1d_1a_2a_3d_2$. This ensures that every letter in our input strings is unique, even if there are multiple occurences of the base letter. The labeling takes $O(m+n)$ time, since we iterate through the length of each input string. And since we are calling CLCS, we have a run time of $o(mn)$ for CLCS. The time it takes to label the strings will be dwarfed by the run time of CLCS. Thus, LCS can also be solved in O(T(m,n)) time.

(b) We will show that this modified algorithm still works because:
$\forall i, j\ \exists k\ LCS\big(cut(A,i), cut(B,j)\big) \leq LCS\big(cut(A,k), cut(B,0)\big)$.
For $LCS\big(cut(A,i), cut(B,j)\big)$, we have three options. $LCS\big(cut(A,i), cut(B,j)\big)$ can be 0, 1, or greater than one. In the first case, if the longest common subsequence is 0, there are no common characters in the two strings, so $\forall k\ LCS\big(cut(A,k), cut(B,0)\big) = 0$. In the second case where the longest common subsequence between $cut(A,i)$ and $cut(B,j)$ is 1, there is at least one character in common between the two strings, so $\forall k\ LCS\big(cut(A,k), cut(B,0)\big) \geq 1$ since at least the one character the two strings share will be a common subsequence, and there could be more depending on the cut (i.e. the longest common subsequence between AB and BA is one, but the longest common subsequence between AB and AB is two).
If $LCS\big(cut(A,i), cut(B,j)\big) > 1$, then there exist a sequence $< X_1, ..., X_s >$ in both $cut(A,i)$ and $cut(B,j)$ which is the longest common subsequence. If $cut(B,j)$ preserves the order of $< X_1, ..., X_s >$, then we know that $LCS\big(cut(A,i), cut(B,j)\big) \leq LCS\big(cut(A,i), cut(B,0)\big)$ since the longest common subsequence $< X_1, ..., X_s >$ is in both $cut(A,i)$ and $cut(B,0)$ in the same order. If $cut(B,j)$ does not preserve the order

of $< X_1, ..., X_s >$, then since it is a different cut of B, we must have that $cut(B, 0)$ has the elements $< X_{cut+1}, ..., X_s, X_1, ...X_{cut} >$. In order to get a common subsequence in a cut of A of at least the size of $LCS\big(cut(A, i), cut(B, j)\big)$, we must ensure that elements $< X_1, ..., X_s >$ are also in the order $< X_{cut+1}, ..., X_s, X_1, ...X_{cut} >$. We can do this by finding the index of X_{cut} in $cut(A, i)$ and cutting there to ensure that X_{cut+1} is the first element in the common subsequence in the new array. Thus, in the array $cut(cut(A, i), A.index(X_{cut}))$ the key elements are in order $< X_{cut+1}, ..., X_s, X_1, ...X_{cut} >$. We can rewrite $cut(cut(A, i), A.index(X_{cut})) = cut(A, i + C.index(X_{cut}))$ where $C = cut(A, i)$. Thus, there is a value for k such that $< X_{cut+1}, ..., X_s, X_1, ...X_{cut} >$ is a common subsequence, so $LCS\big(cut(A, i), cut(B, j)\big) \leq LCS\big(cut(A, i + C.index(X_{cut}), cut(B, 0)\big)$ since all the elements in $LCS\big(cut(A, i), cut(B, j)\big)$ are in the same order in $cut(A, i + C.index(X_{cut})$ and $cut(B, 0)$.
Lastly, we just need to prove that:
$\forall i, j\ \exists k\ LCS\big(cut(A, i), cut(B, j)\big) \leq LCS\big(cut(A, k), cut(B, 0)\big)$ is sufficient to prove that it is enough just to compare $LCS\big(cut(A, k), cut(B, 0)\big)\ \forall k$. Since we are only looking for the largest common subsequence, and we know that for every pair of $cut(A, i)$ and $cut(B, j)$ there is a pair $cut(A, k)textandcut(B, 0)$ which has a common subsequence of the same length or greater length. Thus, the longest common subsequence can be found from a $cut(B, 0)$, so all we need to compare is $LCS\big(cut(A, k), cut(B, 0)\big)\ \forall k$.

(c) First, we want to show that given a common subsequence of the strings, we can show how to create a corresponding path in the graph. This is fairly easy to do. We begin at the bottom right corner of the graph (m,n) and use left-move precedence, meaning that we move left in the cases of no given choices of where to move. We move left until we enter into a column which matches the current character in our subsequence (we start with the last character of the string). We then move up until we find the row that has the matching letter. Once we have matched the both the row and the column to the letter are looking, we move diagonally up and to the left. Then, we repeat the process with the next character. Once we finish looking at all the characters in our subsequence, we move left until we hit the 0th column, and then up until we hit the origin. We then simply reverse this path from (0,0) to (m,n) to find our actual path.

Second, we want to show that given a path from the top left to the bottom right, we can recover a common subsequence. The algorithm for this is simple. We start at (0,0) and move along the path. If we move diagonally along the path, we add the corresponding letter from the box we arrive at after traveling along the diagonal. We don't add any letters from lateral or vertical movements. This will recover a common subsequence.

Third, we want to that the SHORTEST paths in the graph correspond to the LONGEST common subsequences. The shortest path in any graph is constant diagonal movement from (0,0) to (m,n), since it takes two movements, down and right, to cover the same ground that a diagonal movement takes. The max number of diagonal movements in a graph is min(m,n). We cannot have more diagonals since a path cannot go back on itself. However, we also just showed that we add a letter to our common subsequence for every

diagonal movement. So the longer the subsequence, the more diagonal movements we make, which corresponds to the shorter the path. So the shortest paths in the graph correspond to the longest common subsequences.

(d) Since $i < j$, $(j, 0)$ is necessarily in $G_{L_i} \cup p_i$ since $(j, 0)$ is directly below $(i, 0)$. Also, $(m + j, n)$ is necessarily in $G_{L_i} \cup p_i$ because it is directly below $(m + i, n)$. Since we start and end in $G_{L_i} \cup p_i$, in order to include a node in G_{U_i} we would have an initial node in p_i and then cross back over p_i to get to our node $(m + j, n)$. This means that if we have any nodes in G_{U_i}, they are they are surrounded by nodes in p_i and we have a subpath $p_1 \Rightarrow u_1 \Rightarrow ... \Rightarrow u_n \Rightarrow p_2$ where p_x is a node in p_i and u_x is a node in G_{U_i}. By definition, though, p_i is the shortest path, and so the subpath between any two nodes in p_i is also the shortest path between them. This means that it is shorter or equal to use the shortest path to get from p_1 to p_2 rather than taking the path $u_1 \Rightarrow ... \Rightarrow u_n$. It may be quicker never to use p_i at all from $(j, 0)$ to $(m + j, n)$, but there is always a shorter path entirely in G_{L_i} than using any nodes in G_{U_i}.