CS161 Programming Project

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

May 26, 2014

Theory

1. 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.

2. We will show that this modified algorithm still works because for all i and j there exists a k such that $LCS\big(cut(A,i), cut(B,j)\big) \leq LCS\big(cut(A,k), cut(B,j)\big)$

3. 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.