CS161 Spring 2013 Project

SUNet ID:	agiel	ankitk	hkader
Name:	Andrew Giel	Ankit Kumar	Hussain Kader

By turning in this assignment, We agree by the Stanford honor code and declare that all of this is our own work.

1

Given two strings A and B of length m and n, create a new letter not in the alphabet of either string, which we will denote $*$. Now, append a $min(m,n)+1 = l$-length string of $*$s to both strings to create A^* and B^*. We feed A^* and B^* into CLCS and receive Z^* as the returned sequence; i.e, Z^* is the matches in $CLCS(A^*, B^*)$. First, we claim that Z^* must contain at least one $*$. Proof: say this was not the case, and say Z^* was the longest valid $CLCS$, but it has no $*$s. Then Z^* contains only letters, and so Z^* can be at most size $min(m,n)$, as surely only $min(m,n)$ letters can be matched in the original string. But there were $min(m,n)+1 > min(m,n)$ $*$s added, and matching each of these $*$s to each other is possible; hence, we have a contradiction, as we assumed Z^* was the longest valid $CLCS$, but it is shorter than simply matching all $*$s. Now, note that Z^*, since it's a CLCS output, must correspond to some procedure which mimics the behavior of $LCS(cut(A^*, i), cut(B^*, j))$ for some i,j. Note that $CLCS(A^*, B^*)$ can be of length $LCS(A,B)+l$ by simply cutting at 0 and matching A to B and then all l stars to each other; therefore, any Z^* must be length greater than or equal to that. There are four ways that the strings A^*, B^* can look after a cut: *s, letters, *s; letters, *s, letters; *s, letters; or letters,*s. We claim that, in Z^*, by starting at the first letter after the first sequence of *s, and wrapping around the end of the string if necessary, we obtain $LCS(A,B)$. We will prove this by (many) cases.

1. The two cuts were such that A^*, B^* both looked like *s, letters. The only way for this to happen is if both were cut right before the *s; hence, the strings look like $l * s$ then the original A, and $l * s$ then the original B. Thus, the LCS of this cut is to match all the $*$s and then match the original strings A and B to each other. Therefore, starting after the last matched $*$, we obtain the LCS of A and B.

2. The two cuts were such that A^*, B^* both looked like letters, *s. Then both were cut at 0, and the LCS is to match the original A to the original B and then all the $*$s. Again, if we start at the the end of the $*$s (instantly wrapping around to the start), we obtain the LCS of A and B in order.

3. The two cuts were such A^*, B^* both looked like letters, *s, letters. Note that if any of the letters match to a letter in the other string that is on the other side of the *s, none of the *s can match to each other as there would then be a crossing of matches. The maximum length of matching in this case is $min(m,n)$, which is less than l, so this could not happen in a CLCS output. Therefore, letters on either side of the *s only match to each other, implying that all the $*$s match to each other, since this maximizes the LCS length. So, starting after the last * matched, we see that the matches correspond to matches between the start of A

and B, and then wrap around to matches between the end of A and B. Finally, since this is a CLCS output with l matching *s, the size of the matching between A and B must be $\geq LCS(A, B)$, but the matches are exactly in order and thus cannot be $> LCS(A, B)$, so in fact these matches are $LCS(A, B)$.

4. The two cuts were such A^*, B^* both looked like *s, letters, *s. If any of the *s on one side of the letters matched to any *s on the other side, none of the letters could match to each other; thus the size of the CLCS output would be at most l. This would imply that $LCS(A, B)$ is 0, as otherwise simply cutting at 0 and taking the output size $LCS(A, B) + l$ would be bigger. Thus, the LCS would be the empty string, which is what we'd return as there are no letters in the CLCS output. Otherwise, the *s on either side of the letters match to each other only, and the letters match to each other in the middle. But note that the letters are now perfectly in order as they were in A and B, so these matches are indeed $LCS(A, B)$, and we obtain them by again starting at the the last * in the first sequence of *s and reading the letters in the matching in order (note that there must be a match in *s before the first letter, else the size of the CLCS output would be $LCS(A, B) + l'$ where $l' < l$, since a * is not being matched at the beginning.

5. The two cuts were such A^*, B^* looked like letters, *s and *s, letters. Note that all letters are in order of the original A, B here. Thus, if we match NO *s, the best we can get is $LCS(A, B)$, as we'd be matching the letters in order. However, this is surely less than $LCS(A, B) + l$, so this wouldn't happen. Therefore, we must match the *s, and subsequently we cannot match any of the letters, as there would then be a crossing of matches. Therefore, in order for this to be a valid CLCS output, $LCS(A, B)$ must equal 0, and thus the LCS is the empty string, as we would return.

6. The two cuts were such A^*, B^* looked like letters, *s and *s, letters, *s. If any of the first *s in the second string matched to the last *s in the first, none of the letters would be able to match, implying that $LCS(A, B)$ equals and that the LCS is the empty string, which we'd return. Otherwise, this formation could not correspond to a CLCS output, as the best matching we could get is A matched with B and then the final $l' < l$ stars matched to each other, giving a length of $LCS(A, B) + l' < LCS(A, B) + l$.

7. The two cuts were such A^*, B^* looked like letters, *s, and letters, *s, letters. If any *s are matched to each other, only the letters at the beginning of both cuts could be matched to each other. But note these blocks of letters are sequential as they appear in A, B, so the length of this match is $\leq LCS(A, B)$, as $LCS(A, B)$ could still take these matches. If that $\leq$ is equality, then we have a valid $LCS(A, B)$ as the matches are sequential as they appear in A, B, and starting at the end of the sequence of matched *s (instantly wrapping around), we'll obtain this LCS. Otherwise, the $\leq$ is a $<$, and therefore this isn't a valid CLCS output, as the size is less than $LCS(A, B) + l$. If, on the other hand, none of the *s are matched to each other, we can only match the letters giving a maximum match length of $min(m, n)$, which is $< l$, so again this isn't a valid CLCS output.

8. The two cuts were such A^*, B^* looked like *s, letters and *s, letters, *s If any of the last block of *s in the second match to the *s in the first cut, none of the letters can match, implying $LCS(A, B)$ equals 0 and the LCS is the empty string, which we would return.

Otherwise, this formation couldn't lead to a valid CLCS, as only $l' < l$ *s would match, and other than that the letters would match sequentially to each other, leading to a match size of $LCS(A, B) + l'$, which is less than $LCS(A, B) + l$.

9. The two cuts were such A^*, B^* looked like *s, letters, and letters, *s, letters. If the *s match to each other, the only other matches can be the second set of letters in the second cut (which is the beginning of the string B) and the letters in A. This matching is $\leq LCS(A, B)$, as in case 7. Again, if we have equality, we'll return a valid LCS by starting after the end of the first sequence of *s; otherwise, it is strictly less than, and therefore the output would not be a valid CLCS output, as the length would be $< LCS(A, B) + l$.

10. The two cuts were such A^*, B^* looked like letters, *s, letters and *s, letters, *s. If any of the first sequence of *s in the second cut matches to the *s in the first cut, none of the letters can match and therefore $LCS(A, B)$ would have to be 0 for this to be a valid output, and we would return the empty string as required. Otherwise, none of these first *s match, and thus the output length is $< LCS(A, B) + l$, and so this would not be a valid CLCS output.

We need not consider any other cases, as they would be the symmetric opposite of one of the above cases and thus fall under the same proof. Note that this proof is certainly cumbersome and there is probably a more elegant way. However, this way is correct. We want to clarify that the strategy of this proof is as follows: given an output Z^* from the black-box CLCS implementation, we claim that we can obtain the LCS of A,B from starting at the first letter after the first sequence of *s in Z^*, and wrapping around if necessary, then stopping at the next *. To prove this claim, we note that Z^* MUST correspond to $LCS(cut(A^*, i), cut(B^*, j))$. We then show that for any combination of cuts, starting at the first letter after the first sequence of *s in the matching and wrapping around if necessary correctly finds the LCS. In these cases, we often use the fact that Z^* is a valid CLCS output and thus it's length must be $\geq LCS(A, B) + l$.

2

Take any i, j. We claim $\exists k$ such that $LCS(cut(A, i), cut(B, j)) \leq LCS(cut(A, k), cut(B, 0))$. Label the letters in the original arrangement of A, B as $a_1, a_2, ..., a_m$ and $b_1, b_2, ..., b_n$. Label the matches of $LCS(cut(A, i), cut(B, j))$ as (a_x, b_y) for $x \in \{1, ...m\}$, $y \in \{1, ...n\}$. Split the strings $cut(A, i)$ and $cut(B, j)$ into 2 regions each: $A_1 = \{a_1, ...a_i\}, A_2 = \{a_i + 1, ...a_m\}$ and $B_1 = \{b_1, ...b_j\}, B_2 = \{b_i + 1, ...b_n\}$ Note that in $cut(A, i)$ there is, sequentially, A_2 then A_1, and in $cut(B, j)$ there is, sequentially, B_2 then B_1. There are three cases; in all of them, we prove that there is a k such that all the $LCS(cut(A, i), cut(B, j))$ matches are valid:

1. In $LCS(cut(A, i), cut(B, j))$, there are only matches between A_1 and B_1 and between A_2 and B_2. Then all of these matches are valid in $LCS(cut(A, 0), cut(B, 0)$, as these two regions are sequential in A and B, so these matches will have no crosses. Thus $LCS(cut(A, 0), cut(B, 0)) \geq LCS(cut(A, i), cut(B, j))$

2. There are matches between B_2 and A_1, B_2 and A_2, and B_1 and A_1. Then let (a^*_x, b^*_y) by the match with the largest value $y*$ given that $x* \in A_1$ and $y* \in B_2$. Let $k = x* + 1$. Then

note that $cut(A,k)$ has, sequentially, $a_x \in A_1$ such that $x > x*$, then A_2, then the rest of A_1. B has, sequentially, B_1 then B_2. Note that any letter in B_1 can only match to letters $a_x \in A_1$ such that $x > x*$, else there would be a crossing of matches in $LCS(cut(A,i), cut(B,j))$. Letters in B_2 sequentially match with A_2, then $a_x \in A_1$ such that $x < x*$ as this is how the matches must have been in $LCS(cut(A,i), cut(B,j))$. Thus, all original matches are valid, so $LCS(cut(A,k), cut(B,0)) \geq LCS(cut(A,i), cut(B,j))$.

3. There are matches between A_2 and B_1, B_2 and A_2, and B_1 and A_1. Then let (a_x^*, b_y^*) by the match with the smallest value $y*$ given that $x* \in A_2$ and $y* \in B_1$. Let $k = x* - 1$. Then note that $cut(A,k)$ has, sequentially, $a_x \in A_2$ such that $x \geq x*$, then A_1, then the rest of A_2. B has, sequentially, B_1 then B_2. Note that letters in B_1 match sequentially to the letters $a_x \in A_2$ such that $x \geq x*$, then letters in A_1, sequentially, as this is how it was in $LCS(cut(A,i), cut(B,j))$. Then, letters in B_2 must only match to letters $a_x \in A_2$ such that $x < x*$, else there'd be a crossing of matches in $LCS(cut(A,i), cut(B,j))$. Thus, again, all original matches are valid, so $LCS(cut(A,k), cut(B,0)) \geq LCS(cut(A,i), cut(B,j))$

Finally note that we cannot have matches between A_2 and B_1 and matches between A_1 and B_2, else there'd be a crossing of matches. Thus the above 3 cases are all the possible cases, and in each we may find a k such that $LCS(cut(A,k), cut(B,0)) \geq LCS(cut(A,i), cut(B,j))$. Thus, it suffices to only try cuts of the form $cut(A,k), cut(B,0)$, as CLCS returns the largest possible $LCS(cut(A,i), cut(B,j))$, and for every $LCS(cut(A,i), cut(B,j))$ such that $j \neq 0$, there is a $LCS(cut(A,k), cut(B,0))$ greater than or equal to it.

3

Given a path $(m_0, n_0), ..., (m_x, n_x)$ such that $(m_0, n_0) = (0,0)$ and $(m_x, n_x) = (m,n)$, the corresponding common subsequence is $A[m_x + 1]$ for all pairs such that $(m_x + 1, n_x + 1) = (m_{x+1}, n_{x+1})$ (i.e all diagonal edges).

Thus, given a shortest path from $(0,0) \to (m,n)$, the corresponding common subsequence as defined above will be the longest common subsequence as long as we can prove that the shortest path from $(0,0) \to (m,n)$ contains the maximum number of diagonal edges. To see this, note that there is never an edge that goes back, i.e there is no edge $(m_1, n_1) \to (m_2, n_2)$ such that $m_1 > m_2$ or $n_1 > n_2$. Further note that no edge has length more than 1; i.e, given an edge (m_1, n_1) to (m_2, n_2), $m_2 = m_1$ or $m_1 + 1$, and $n_2 = n_1$ or $n_1 + 1$. Therefore, the length of any path from $(0,0) \to (m,n)$ will be $m + n - d$, where d is the number of diagonal edges, as every time there is a diagonal edge we save one move, and otherwise we must go from 0 to m and 0 to n in 1 move increments. Thus, the shortest path will maximize d and thus have the maximum number of diagonal edges, and therefore be a longest common subsequence.

4

By contradiction. Let $P_1, ..., P_x$ be p_i, so $P_1 = (i, 0)$ and $P_x = (m+i, n)$. Let $s = s_1, ..., s_y$ be the shortest path from $(j, 0) \to (m+j, n)$ in $G_{L_i} \cup p_i$, and let $\rho = \rho_1, ..., \rho_z$ be a shorter path that goes through $x \in G_{U_i}$. Since ρ starts in $G_{L_i} \cup p_i$ and touches $x \in G_{U_i}$, there must be some $\rho_a, ...\rho_{a+q}$ such that $\rho_a = P_k$ and $\rho_{a+q} = P_{k+w}$ for some k, w, as p_i is the boundary between the two regions. Now, since p_i is itself a shortest path, it must be that $w \leq q$. Thus, consider the new path $\gamma = \rho_1, ...\rho_{a-1}, P_k, P_{k+1}, ..., P_{k+w}, \rho_{a+q+1}..., \rho_z$. Since $w \leq q$, $len(\gamma) \leq len(\rho)$. But note that now γ does not go through x; in fact it is contained strictly in $G_{L_i} \cup p_i$, and thus by s a shortest path, $len(s) \leq len(\gamma) \leq len(\rho) \implies len(s) \leq len(\rho)$, a contradiction.

5

$SINGLESHORTESTPATH(A, B, p, p[l], p[u])$ uses DP to compute the shortest path in region bounded by $p[l]$ and $p[u]$. The number of nodes in this region is not $(u-l)n$, as the paths $p[l]$ and $p[u]$ can look many different ways. However, note that the two calls to $SINGLESHORTESTPATH$ in the two recursive calls to $FINDSHORTESTPATHS$ inside a call to $FINDSHORTESTPATHS(A, B, p, p[l], p[u])$ can look at at most the number of nodes in the region bounded py $p[l]$ and $p[u]$, plus the number of nodes in the path $p[mid]$, as the two calls will be $SINGLESHORTESTPATH(A, B, p, p[l], p[mid])$ and $SINGLESHORTESTPATH(A, B, p, p[mid], p[u])$, and thus the nodes looked at are bounded by $p[l]$ and $p[u]$, with the path $p[mid]$ being double counted. In particular, this implies that every level of recursion looks at at most the number of nodes looked in the level of recursion above it plus the $\frac{x}{2}(m+n)$, where x is the number of nodes at that level of recursion. This is because there are $\frac{x}{2}$ double-countings of paths, and every path is at most size $m+n$. This leads to the following sum for the total amount of work done by $SINGLESHORTESTPATH$: $\sum_{i=0}^{\lg m} 2mn + 2^{i-1}(m+n) = \sum_{i=0}^{\lg m} 2mn + \sum_{i=0}^{\lg m} 2^{i-1}(m+n) = 2mn \lg m + O(m^2 + mn)$, as there are $\lg m$ levels and 2^{i-1} double countings for level i (with root $= 0$), and the amount of work done at level 0 is at most $2mn$. Finally, note that $m \leq n$, so $O(m^2 + mn) = O(mn)$, so the overall work done is $O(2mn \lg m + O(mn)) = O(mn \lg m)$ as required.