Brenden Nelson
ID# 11635716

350 HW 5

① Given: 2 strings α and β. Let γ be the longest word satisfying all conditions:
~~①~~ γ is a subsequence of α
~~②~~ γ is a subsequence of β
~~③~~ γ does not contain abb

Question: Design an algorithm that finds such a γ for any given α and β. Also analyze its complexity.

My Algorithm

- I can start by creating a Finite Automata to represent string α (M_α) and another Finite Automata to represent string β (M_β). I can then create a third FA to represent all strings that don't contain abb. I can run cartesian product on the three FA's I just created to obtain a Finite Automata M satisfying all conditions ①, ②, and ③. Since M is a FA, also can be represented as a graph, I run longest path algorithm on G. (Longest path is just the shortest path algorithm but replaces distances from Positive to negative values) I run this longest path algorithm on graph G from initial to accepting states, and collect the word on the path, which is the resulting string satisfying all conditions for γ.
- This Algorithm can be summarized in 3 steps.
① Enumerate all subsequences of α, β, no abbs to 3 FA's.
② Run cartesian product on 3 FA's to find FA/Graph M.
③ Run modified shortest path algorithm (Longest Path) on the resulting FA M to obtain the result.

Time complexity of my Algorithm
① $O(3n^3)$ to enumerate 3 FA's
② $O(n^3)$ to run cartesian product on 3 FA's.
③ $O(V^2)$ to run Djikstras "longest path" algorithm.

Thus total time complexity would be $O(n^3+V^2)$ with n being the total number of FA's and V being the total number of nodes(verticies) in the graph(M) created by running cartesian product. This algorithm is effecient considering the constraints we had on Y and would become less effecient for every new constraint added.

② $d(\alpha,\beta)$ denotes length of $LCS(\alpha,\beta)$. L_1 & L_2 are regular. Design an algorithm to compute $D = \max_{\alpha\in L_1, \beta\in L_2} d(\alpha,\beta)$.
// L_1 & L_2 could be infinite, therefore D would be infinite.

MyAlg
Step 1 - Enumerate α and β as FA's M_α and M_β.
Step 2 - run $LCS(\alpha,\beta)$ to find an Y.
Step3 - run $d(\alpha,\beta)$ to find the length of Y.
Step 4 - // Run $D = \max_{\alpha\in L_1, \beta\in L_2} d(\alpha,\beta)$
- if L_1 is infinite and L_2 is infinite,
then $D = \infty$.
- else
- run cartesian product on 2 FA's.
- run djikstras longest path algorithm to find longest common subsequence(s) between α and β.
- Let D equal the length of Y.

This algorithm would compute $D = \max_{\alpha\in L_1, \beta\in L_2} d(\alpha,\beta)$ to find the integer number of the length of the maximum LCS of $\alpha+\beta$.

(3) Is there any locality sensitive hash scheme for strings?

Yes. In 1965, Vladmir Levenshtein created a distance algorithm to determine how many edits it would take to transform one string into another. This number would measure similarity between strings and thus could store similar strings with similar hash values. this algorithm is very effecient if the words given have something in common (are similar) but becomes less effective as the given words become randomized.