CS 180 HW4

Hanna Co

Due: June 5, 2021

1.

Initialize a nxn array, called OPT
for $i = 0 \sim n$
    for $j = i + 1 \sim n$
        if str[i] == str[j]
            OPT[i][j] = OPT[i+1][j-1]
        else
            OPT[i][j] = min(OPT[i+1][j], OPT[i][j-1]) + 1
return OPT[0][n-1]

Proof of correctness:

This algorithm keeps track of how many insertions are required for substrings of str, and returns how many insertions are required for the total substring. Let's say we have some string of length n that requires say, 3 insertions, but the algorithm returns 4. This would mean that we did not choose the optimnal number of insetions. However, out algorithm ensures that it always chooses the minimum number, by using the min() function and checking for equality. Therefore, out algorithm will always give the minimum number of insertions.

Proof of time complexity:

The outer loop runs from 0 to n, and so does the inner loop, so the total time complexity of $O(n^2)$

2.

3-SAT can be polynomial time reduced to Approx. 3-SAT. Given an instance of Approx. 3-SAT with n clauses, Approx. 3-SAT will find a solution that satisfies $n-1$ clauses. Now lets say we are given an instance of 3-SAT with n clauses. We can add a clause C_f that we know is guaranteed to be false. We then run Approx. 3-SAT on the $n+1$ clauses. If Approx 3-SAT returns true, then we know that the one clause that could not be satisfied was C_f. If it returns false, then we know that there is another clause besides C_f that cannot be satisfied with any assignment. Thus, we can use Approx. 3-SAT to solve 3-SAT. Therefore, 3-SAT $\leq_p$ Approx. 3-SAT.