Assignment 4 is due Sunday, December 24, 23:30.

Given a sequence X of n pairs of nonnegative integers, i.e.,

$$X = (x_1, x'_1)(x_2, x'_2)...(x_{i-1}, x'_{i-1})(x_i, x'_i)(x_{i+1}, x'_{i+1})...(x_n, x'_n)$$

such that $0 < x_i \leq x'_i$, consider the following **tuple operations** on X:

- *insert*$((z, z'), i)$ inserts tuple (z, z') after (x_i, x'_i) (or before (x_{i+1}, x'_{i+1})),
- *delete*(i) deletes (x_i, x'_i),
- *replace*$(i, (z, z'))$ replaces (x_i, x'_i) with tuple (z, z').

Given two sequences X (of length n) and Y (of length m), **the update distance problem** asks for the minimum number of operations to turn X into Y.

For instance, for X=$(1,1)(3,6)(2,5)(4,4)$ and Y=$(1,6)(3,4)(2,5)$, the update distance is 3 since X can be transformed into Y with 3 operations:

$$(1,1)(3,6)(2,5)(4,4) \xrightarrow{\textit{replace}(1,(1,6))} (1,6)(3,6)(2,5)(4,4) \xrightarrow{\textit{delete}(4)}$$
$$(1,6)(3,6)(2,5) \xrightarrow{\textit{replace}(2,(3,4))} (1,6)(3,4)(2,5)$$

Your task is to design, analyze, implement (in Python) and evaluate an algorithm using dynamic programming to solve the update problem.

Submit Report (PDF file) Make sure that your report includes the following:

(a) **Recursive formulation** of the update problem: assuming that $c[i, j]$ denotes the update distance between the prefix $X[1..i]$ (i.e., the first i tuples of X) and the prefix $Y[1..j]$ (i.e., the first j tuples of Y), 1) define $c[i, j]$ recursively and prove the optimal substructure property, 3) draw the recursion tree for $c[n, m]$ and show overlapping computations, and 4) discuss the existence of a topological ordering of the subproblems.

(b) **Pseudocode** of an algorithm designed using dynamic programming, based on the recursive formulation.

(c) **Best worst-case asymptotic time complexity** analysis of your algorithm.

(d) Results of **performance evaluations** of your algorithm: **plot** the experimental results in a graph, and **discuss** the results in terms of scalability in time.

Submit Python Code, and Benchmarks A and B (ZIP file)

(e) **Implement in Python** your algorithm designed using dynamic programming, to solve the update problem. Make sure that your implementation not only returns the solution to the update problem but also displays the operations to turn X into Y. Include your implementation in the zip file.

(f) Create a benchmark suite A of at least 10 instances to **test the correctness** of your Python program: take into account the **functional testing** methods while constructing the instances (e.g., 5 instances for white box testing and 5 instances for black box testing), and test your programs with these instances. Include both the instances and their outputs in the zip file.

(g) Create a benchmark suite B to **test the performance** of your Python program: construct at least 1000 total instances of at least 10 different sizes, measure the performance of your program over these instances in terms of computation time, and gather all the results in a single csv/excel/txt file. Include the instances and the results in the zip file.

Demos Provided that both the report and the zip file described above are submitted by the deadline, demonstrate that your Python program correctly computes solutions for your benchmark instances, and for the instances that will be provided by us. Demo day will be announced.