Course
CS-E3190
Course materials
Your points

« 7.1 Implementing dynamic programming algorithms Course materials Lecture and Exercise Set 4 - Local Search »
CS-E3190 / Programming Exercise 2 - DP / 7.2 String edit distance

String edit distance

Exercise: Edit distance

This exercise asks you to implement an algorithm that computes the edit distance (Levenshtein distance) between two given strings. More precisely, for two strings a and b, the *Levenshtein distance* $D(a, b)$ is the minimum length of a sequence of edit operations that transforms the string a into the string b, where each edit operation in the sequence is one of the following:

> 1. deletion of one character from the string,
> 2. insertion of one character into the string, or
> 3. changing of one character in the string into another character.

For example, the Levenshtein distance satisfies $D(a, a) = 0$ for all strings a, since a sequence of zero edit operations suffices to transform a into a. Similarly, one can show that $D(a, b) = D(b, a)$ and $D(a, c) \leq D(a, b) + D(b, c)$ for all strings a, b, c. That is, D is a metric in the space of all strings.

Your task in this exercise is to complete the subroutine

> `void solver(int n, int m, const char *a, const char *b, int &d)`,

which takes as input a string `a` of length `n` and a string `b` of length `m`. The subroutine should compute the Levenshtein distance $D(a, b)$ and store it into `d`. To locate the subroutine quickly, you can search for "`???`" in the source file. You may assume that $n, m \geq 0$ and $nm \leq 4294967296$. Furthermore, you may assume that $a[n] = b[m] = 0$.

Grading. This exercise awards you up to 10 points in the course grading. The number of points awarded is the maximum points times the number of tests passed over the total number of tests, rounded up. To successfully complete a test, your implementation must use no more than 15 seconds of wall clock time and 10 MiB of memory. Each test will in general require the successful solution of one or more problem instances. In each batch of scaling tests, the first failed test will cause all subsequent tests in the batch to be skipped.

Points **10 / 10** My submissions **1 / 10**
Deadline Tuesday, 11 October 2022, 20:00
To be submitted alone

This course has been archived (Friday, 30 December 2022, 16:00).

Edit distance

1. Download the template source file edit-distance.cpp.
2. Study the problem description above.
3. Complete your solution in the template.
4. Test your solution with the unit tests available in the template.
   Recall that the Scratchpad is available for compiling and running your solution.
5. Submit your solution for grading using the dialog below.
   See here for a quick guide to the grader output.

edit-distance.cpp

Choose File No file chosen

Submit

« 7.1 Implementing dynamic programming algorithms Course materials Lecture and Exercise Set 4 - Local Search »