Exercises — Levenshtein distance

version #

Assistants C/Unix 2022 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.assistants.epita.fr

1 Levenshtein distance

Files to submit:

- levenshtein/levenshtein.c

Provided files:

- levenshtein/levenshtein.h

Authorized functions: You are only allowed to use the following functions:

- malloc(3)
- free(3)

Authorized headers: You are only allowed to use the functions defined in the following headers:

- errno.h
- err.h
- stddef.h
- assert.h

1.1 Goal

The Levenshtein distance represents the edition distance between two strings, meaning the number of atomic changes to do in order to get from one string to the other.

The three possible operations are:

- Insertion of a character (Wherever in the string)
- Deletion a character
- Substitution of a character

You have to code the function `levenshtein` with the following prototype:

```
size_t levenshtein(const char *s1, const char *s2);
```

Mathematically, the Levenshtein distance between the `i` first characters of a string `a` and the `j` first characters of a string `b` is defined as:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

where:

$$1_{(a_i \neq b_j)} = \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases}$$

The tree elements in the minimum correspond respectively to deletion (first element), insertion (second element) and match or mismatch (third element).

The recursive implementation is not compulsory. Your code won't be tested with `NULL` strings.

1.2 Examples

For example, `levenshtein("doom", "tools") = 3`, the 3 changes are:

1. doom → toom (substitution of *d* to *t*)
2. toom → tool (substitution of *m* to *l*)
3. tool → tools (insertion of *s*)

It is my job to make sure you do yours.