Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 Levensthein

0.1.1 Teacher's guide

Emne Strenge, rekursion, tests, caching

Sværhedsgrad Svær

0.1.2 Introduction

Du skal i de følgende to opgaver arbejde med en funktion til at bestemme den såkaldte *Levenstheindistance* mellem to strenge a og b. Distancen er defineret som det mindste antal editeringer, på karakter-niveau, det er nødvendigt at foretage på strengen a før den resulterende streng er identisk med strengen b. Som editeringer forstås (1) sletninger af karakterer, (2) indsættelser af karakterer, og (3) substitution af karakterer.

Varianter af Levensthein-distancen mellem to strenge kan således benyttes til at identificere om studerende selv har løst deres indleverede opgaver eller om der potentielt set er tale om plagiatkode ;)

Matematisk set kan Levensthein-distancen $leven(a,b)$, mellem to karakterstrenge a og b, defineres som $lev_{a,b}(|a|,|b|)$, hvor $|a|$ og $|b|$ henviser til længderne af henholdsvis a og b, og hvor funktionen lev er defineret som følger:[1]

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

hvor $1_{(a_i \neq b_j)}$ henviser til *indikatorfunktionen*, som er 1 når $a_i \neq b_j$ og 0 ellers.

0.1.3 Exercise(s)

0.1.3.1: Implementér funktionen *leven* direkte efter den matematiske definition (ved brug af rekursion) og test korrektheden af funktionen på nogle små strenge, såsom "house" og "horse"(distance 1) samt "hi" og "hej" (distance 2).

0.1.3.2: Den direkte implementerede rekursive funktion er temmelig ineffektiv når strengene a og b er store. F.eks. tager det en del millisekunder at udregne distancen mellem strengene "dangerous house" and "danger horse". Årsagen til denne ineffektivitet er at en løsning der bygger direkte på den rekursive definition resulterer i en stor mængde genberegninger af resultater der allerede er beregnet.

For at imødekomme dette problem skal du implementere en såkaldt "caching mekanisme" der har til formål at sørge for at en beregning højst foretages en gang. Løsningen kan passende gøre brug af gensidig rekursion og tage udgangspunkt i løsningen for den direkte rekursive definition (således skal løsningen nu implementeres med to gensidigt rekursive funktioner `leven`

[1] See `https://en.wikipedia.org/wiki/Levenshtein_distance`.

og `leven_cache` forbundet med `and`). Som cache skal der benyttes et 2-dimensionelt array af størrelse $|a| \times |b|$ indeholdende heltal (initielt sat til -1).

Funktionen `leven_cache`, der skal tage tilsvarende argumenter som `leven`, skal nu undersøge om der allerede findes en beregnet værdi i cachen, i hvilket tilfælde denne værdi returneres. Ellers skal funktionen `leven` kaldes og cachen opdateres med det beregnede resultat. Endelig er det nødvendigt at funktionen `leven` opdateres til nu at kalde funktionen `leven_cache` i hver af de rekursive kald.

Test funktionen på de små strenge og vis at funktionen nu virker korrekt også for store input.

Det skal til slut bemærkes at den implementerede løsning benytter sig af $O(|a| \times |b|)$ plads og at der findes effektive løsninger der benytter sig af mindre plads ($O(\max(|a|, |b|))$). Det er ikke et krav at din løsning implementerer en af disse mere pladsbesparende strategier.