Homework Assignment 1

Matthieu Boyer

3 octobre 2023

Table des matières

1 Exercise 1 - [Edit Distance/Levenshtein Distance]

1.1 Question 1

Proposition 1.1.1 (Complexity and Correction)**.** *If we denote C_f the complexity of f, this algorithm has time complexity $\mathcal{O}\left(C_f\left(\frac{n}{t}\right)^2\right)$. This algorithm is correct.*

Démonstration. — Moreover, it is clear this algorithm is correct as it only just applies the dynamic programming algorithm for the Levenshtein distance by steps.

— This algorithm complexity comes from the fact it has two while loops for which the commands are executed at most $\lceil n/t \rceil$ times. The commands in both *while* loops are executed in $\mathcal{O}(C_f)$. The *for* loops inside the *left while* loop are equivalent to loops for i between left and left + $t-1$ and thus are disjoint. The sum of their complexity over the *left while* loop is then n. The number of operations inside the *up while* loop is then in $\mathcal{O}\left(C_f \frac{n}{t}\right)$ and thus the total complexity is, as announced, in $\mathcal{O}\left(C_f\left(\frac{n}{t}\right)^2\right)$

■

1.2 Question 2

By the recurrence formula : $\mathbf{D}[i][j] = \max \begin{cases} \mathbf{D}[i-1][j]+1 \\ \mathbf{D}[i][j-1]+1 \\ \mathbf{D}[i-1][j-1]+1 if S[i] \neq T[j] \text{ else } 0 \end{cases}$ we see that $\mathbf{D}[i][j]$ is at most 1 plus one of its left neighbour, upper neighbour or upper left corner neighbour.

1.3 Question 3

By recurrence formula, if we substract from all the values in A, B, C a certain integer k, then we get for the new matrix F, the one we would have gotten with A, B, C with k substracted to all values. Thus, if the values in A, A', B, B' and C, C' all differ from a common integer, the resulting

Algorithme 1 Question 1 - Levenshtein Distance with f

Input S, T, f, t ▷ Two Strings, the function f computing the values and the step t

$\mathbf{D} = \text{zeros}(n+1, n+1)$ ▷ $len(S) = len(T) = n$
for $i \leftarrow 0$to $n+1$ **do**
  $\mathbf{D}[i][0] \leftarrow i$
end for

for $j \leftarrow 0$ to $n+1$ **do**
  $\mathbf{D}[0][j] \leftarrow j$
end for

$\text{up}, \text{left} \leftarrow 0, 0$
while $\text{up} < n$ **do**
  $\text{left} \leftarrow 0$
  while $\text{left} < n$ **do**
    $down \leftarrow \min(n - \text{up}, t)$
    $right \leftarrow \min(n - \text{left}, t)$

    $b \leftarrow \mathbf{D}[\text{up}][\text{left}]$
    $a \leftarrow \mathbf{D}[\text{up}+1 \rightarrow \text{up}+1+down][\text{left}]$
    $c \leftarrow \mathbf{D}[\text{up}][\text{left}+1 \rightarrow \text{left}+1+right]$

    $f(a, b, c, d, e)$ ▷ We can suppose here that f modifies only the last line and column of F in $\mathbf{D}$ with side-effect.
    $\text{left} \leftarrow \text{left} + right$
    for $i \leftarrow 1$ to $down - 1$ **do**

$$\mathbf{D}[\text{up}+i][\text{left}] \leftarrow \min \begin{cases} \mathbf{D}[\text{up}+i][\text{left}-1] + 1 \\ \mathbf{D}[\text{up}+i-1][\text{left}] + 1 \\ \mathbf{D}[\text{up}+i-1][\text{left}-1] + \mathbb{1}_{\{S[up+i]=T[left]\}} \end{cases}$$

      ▷ We update the first Column of the block we consider.
    end for
  end while
  $\text{up} \leftarrow \text{up} + d$
  for $i \leftarrow 1$ to n **do**

$$\mathbf{D}[\text{up}][i] \leftarrow \min \begin{cases} \mathbf{D}[\text{up}][i-1] + 1 \\ \mathbf{D}[\text{up}-1][+i] + 1 \\ \mathbf{D}[\text{up}-1][i-1] + \mathbb{1}_{\{S[up+i]=T[left]\}} \end{cases}$$

    ▷ We update the first line of the blocks we will consider.
  end for
end while
return $\mathbf{D}[n][n]$

values after applying f will differ from this same value. Thus, $F' = F + (A^{'} - A)$ and $A^{'} - A$ is a matrix with all values equal.

1.4 Question 4

We will show here that we can pre-compute all $t \times t$ matrices in $\mathcal{O}\left(3^{2t}\sigma^{2t}t^2\right)$.
First, as $\mathbf{D}[i][j]$ here is the minimal number of elementary operations to go from string $S[0:i]$ to string $T[0:j]$. We can thus interpret the submatrix of $\mathbf{D}$ between (i,j) and $(i+t, j+t)$ as the dynamic programming matrix for minimum number of operations to go from string $S[i:i+t+1]$ to string $T[j:j+t+1]$ to which we added a first line and a first column.
We thus see that those matrices can be fully determined by their first line, first column and by two words.
As the values in $\mathbf{D}$ are bounded by 0 and $2n$ (we can always remove all letters in S and add all letters in T), and by question 2., as values along a line or a column differ by an integer in $-1, 0, 1$, the number of first lines and columns is $\mathcal{O}(n3^{2t})$. However, from question 3., if we allow negative values for f (which we have no reason not to do), we can always substract from the first line and column the value in the top left corner, and re-add it in $\mathcal{O}(t^2)$ after pre-processing.
Moreover, there are σ^t words of length t over the alphabet so the number of submatrices is $\mathcal{O}(3^{2t}\sigma^{2t})$.
We can then use the recurrence equation to derive the values on the submatrix in $\mathcal{O}(t^2)$.
Finally, we can pre-compute all $t \times t$ submatrices in $\mathcal{O}(3^{2t}\sigma^{2t}t^2)$.
Then, to access the values of the submatrix from $(up, left)$ to $(up+t, left+t)$, we need to identify the preprocessed corresponding matrix and thus we need to go through $S[up:up+t]$, $T[left:left+t]$, the first column and row of this submatrix to which we substracted the upper left value, which all are done in $\mathcal{O}(t)$.

1.5 Question 5

From Question 4, we have got an algorithm that allows us to compute the result in $\mathcal{O}(3^{2t}\sigma^{2t}t^2 + \left(\frac{n^2}{t}\right))$.
Indeed, after pre computing, we only need to check in $\mathcal{O}(1)$ the $\mathcal{O}\left(\left(\frac{n}{t}\right)^2\right)$ $t \times t$ submatrices in $\mathbf{D}$.
Thus, if we take $t = \lfloor \log_{(3\sigma)}(\sqrt{n}) \rfloor$, we have complexity in $\mathcal{O}\left((3\sigma)^{\log_{3\sigma}(n)} \log_{3\sigma}(\sqrt{n}) + \left(\frac{n^2}{\log_{3\sigma}(\sqrt{n})}\right)\right)$.
As $\log_{3\sigma}(\sqrt{n}) = \Theta(\log(n))$ and $(3\sigma)^{\log_{3\sigma}(n)} \log_{3\sigma}(\sqrt{n}) = \mathcal{O}(n\log(n)^2) = o(\frac{n^2}{\log(n)})$, this algorithm has complexity in $\mathcal{O}\left(\frac{n^2}{\log n}\right)$

2 Exercice 2

2.1 Question 1

We will store in an array A the result of the rank queries for value 1 : $A[i] = rank_1[i]$. Then $rank_0[i] = i - A[i]$.
To store the integer k, $\lceil \log_2(k) \rceil$ bits are needed. Thus, this array uses :

$$\sum_{k=1}^{n} \lfloor \log_2(k) \rfloor =$$