Групове завдання 24

Є функція, що повертає список слів.

Побудувати «фільтри» у вигляді декораторів, які «псують» слова з цього списку наступним чином:

1. Додають до коду 1 символу слова деяке задане число.
2. Видаляють 1 символ слова
3. Вставляють 1 символ у слово

Отримати випадковий рядок з файлу “text.txt”, запам’ятати його в окремому текстовому файлі, перетворити його у список слів та пропустити через «фільтри». Кожне слово word може проходити через фільтри не більше len(word) // 3 разів.

Утворити зі списку відфільтрованих слів рядок (слова мають розділятись пропусками) та передати іншій команді, вказавши окрім рядка назву своєї команди (A,B, C, D, E) та порядковий номер фільтрованого повідомлення.

Інша команда має спробувати відтворити все або максимальну частину первинного повідомлення.

Відтворення треба робити за допомогою алгоритму, що обчислює відстань Левенштейна між 2 рядками. Для цього з файлу “text.txt” отримати усі різні слова та знайти для кожного слова повідомлення слово з файлу з мінімальною відстанню Левенштейна.

Перемагає команда, яка відтворить максимальну кількість повідомлень інших команд.

Для розрахунку відстані Левенштейна найчастіше застосовують простий алгоритм, в якому використовується матриця розміром (n + 1) * (m + 1), де n і m - довжини порівнюваних рядків. Окрім цього вартість операцій вилучення, заміни та вставки вважається однаковою. Для конструювання матриці використовують таке рекурсивне рівняння:

$D_{0,0}=0$

$$D_{i,j}=\min\begin{cases} D_{i-1,j-1} + 0 \text{ (equal)} \\ D_{i-1,j-1} + 1 \text{ (replace)} \\ D_{i-1,j} + 1 \text{(insert)} \\ D_{i,j-1} + 1 \text{(delete)} \end{cases}$$

У псевдокоді алгоритм виглядає так:

```
int LevenshteinDistance(char str1[1..lenStr1], char str2[1..lenStr2])
  // d таблиця кількість рядків = lenStr1+1 та кількість стовпців = lenStr2+1
  declare int d[0..lenStr1, 0..lenStr2]
  // i та j використовуються для індексування позиції у str1 та у str2
  declare int i, j, cost

  for i from 0 to lenStr1
     d[i, 0] := i
  for j from 0 to lenStr2
     d[0, j] := j
```

```
for i from 1 to lenStr1
   for j from 1 to lenStr2
      if str1[i] = str2[j] then cost := 0       //однакові
                           else cost := 1       //заміна
      d[i, j] := minimum(
                           d[i-1, j  ] + 1,     // вилучення
                           d[i  , j-1] + 1,     // вставка
                           d[i-1, j-1] + cost   // заміна або однакові
                         )

return d[lenStr1, lenStr2] //значення відстані Левенштайна в останній клітинці матриці
```