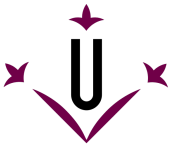


Universitat de Lleida

Universitat de Lleida
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Algorítmica i complexitat

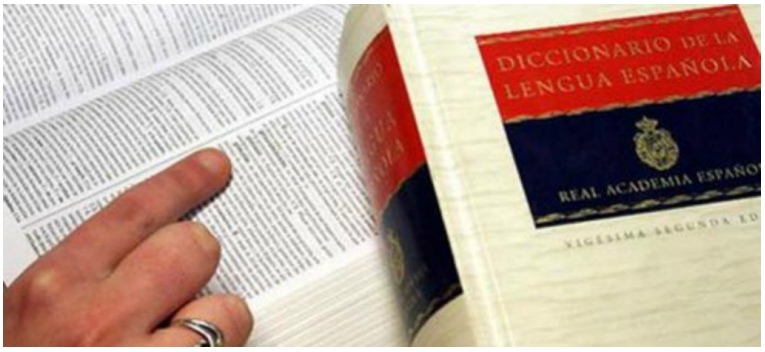
Pràctica 2
Algorítmica i Complexitat

Boris Llona Alonso
Robert Munné Mas
Grup GM2
19/05/2019
Curs 2018/2019

Universitat de Lleida

Plantejament de la pràctica:

Per a realitzar el que se'ns descriu a l'enunciat primer cal pensar com abordar la implementació dels problemes que ens podem trobar.
Creiem que es clau que hi hagi tres patrons de modificació de la paraula ja que ens hem de basar en aquests per actuar en conciencia. Aquests són eliminar, afegir o reemplaçar una lletra. Aquests patrons els podem fer tantes vegades com vulguem en una paraula i mitjançant un diccionari hem de comparar quina es la mes adient per a substituir-la.
És evident que per cada lletra mínim haurem de provar coincidències o discrepàncies amb totes les lletres del diccionari, ja que sinó ens podriem deixar una presumpta paraula millor que la anterior trobada.
Un altre tema a tenir en compte es la llargada dels diferents Strings a comparar ja que si comparem posició a posició de manera *standard* ens podrem trobar amb errors.
Per exemple tenim la paraula Algorítmica i afegim una X al principi tal que -> XAlgorítmica. Anomenant k a Algorítmica i j a XAlgorítmica. k[i] != j[i] on i es un iterador desde 0 fins la seva llargada.
Altrament, afegint-ho al final trobaria bé les coincidències en aquest cas. Però clar, es pot modificar tota lletra dins de la paraula.



Universitat de Lleida

Estratègia emprada:

La estratègia que hem pensat per a resoldre aquest problema és, recorrer cada paraula del arxiu original i per cadascuna d'elles mirar amb totes les del diccionari. Per cada paraula busquem les lletres iguals entre la paraula amb errors i la del diccionari. Tot seguit calculem les diferències i ho guardes a una llista d'on traurem la que tingui menys diferències.
És una manera simple i eficient de saber quina és la paraula que més s'assembla. Al final només cal concatenar aquesta llista i escriure-la al fitxer de sortida.

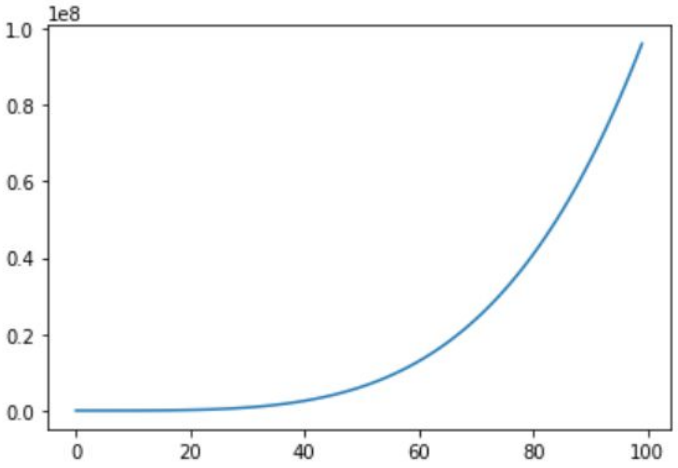
Disseny Iteratiu:

Mentre hi hagi paraules a original:
    Mentre hi hagi paraules al diccionari:
        Mentre lletra no hagi estat comparada amb les altres:
            Mirem diferències entre paraula a i b
    millorParaula = min(diferències)
    millor += millorParaula
output.writte(millor)

Anàlisi de costos:

El nostre algorisme iteratiu com he comentat conté un bucle per analitzar la paraula del fixer, un altre per comparar-la amb tots els del diccionari, i un parell per a veure les combinacions en la paraula i determinar la que té el millor ràtio.
Per tant incorre en uns costos en notació big oh de 0(N^4).

Universitat de Lleida



Disseny Recursiu:

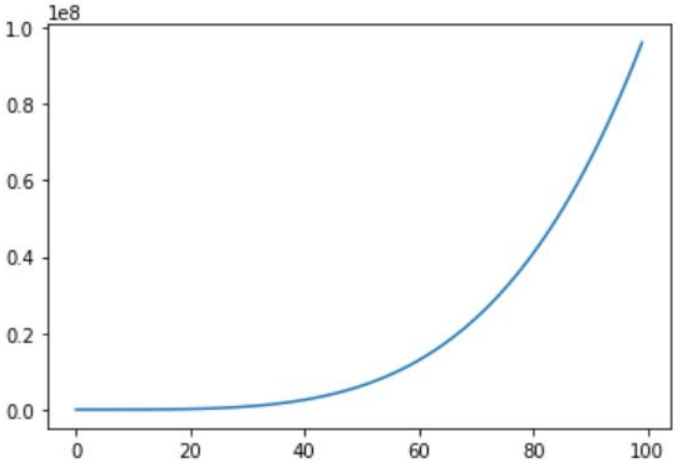
```
def compare(textList, dictList):
    global number, corrected
    new_text = ''
    total_differences = 0

    if len(textList) == 0:
        return 0

    i = textList[0].strip()
    pos = 0
    differences = []

    add_differences(dictList, differences, i, pos)

    min_differences = min(differences)
    total_differences += min_differences

    if new_text != '':
        new_text += ' '

    new_text += dictList[differences.index(min_differences)]

    differences.clear()
    compare(textList[1:], dictList)

    number.write(str(total_differences))
    corrected.write(new_text)
```

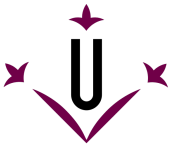
```
def add_differences(dictList, differences, i, pos):
    if len(dictList) == 0:
        return 0

    j = dictList[0].strip()
    differences.append(0)
    s = difflib.SequenceMatcher()
    s.set_seqs(i, j)

    differences[pos] = diff(s, len(i), len(j))
    pos += 1

    return add_differences(dictList[1:], differences, i, pos)
```

Com es pot apreciar en el codi recursiu, la forma que em seguit per a resoldre-ho es que fins que no acabi de comprobar una paraula del text amb totes les del diccionari no sortirà de la funció `add_differences()`. Per a que llegeixi la següent paraula ho fem acurtant la llista a cada iteració amb list[1:].

Análisi de costos:

L'algorisme recursiu té el mateix cost: 0(n^4) ja que realitza exactament el mateix procediment que l'iteratiu, però en comptes de realitzar els bucles en base a un for, ho fa cridant a la mateixa funció amb el diccionari adequat per analitzar sempre i quant hi hagi text per a analitzar.





Conclusions:

Resoldre aquesta segona pràctica considerem ha sigut molt bo per a la nostra formació en quant a programació. Sobretot hem après analitzant el codi de la llibreria **difflib.** Ja que un problema que a priori sembla complexe hem vist que al final la millor solució és la més intel·ligent no la més rebuscada.
També hem tingut en compte en tot moment el tema dels costos per a que sigui el més petit possible i això ens ha donat una visió mes amplia sobre el projecte i ens ha fet reflexionar encara més sobre com afrontar-ho.