

|  |
| --- |
| **UNIVERSITAT DE LLEIDA** |
| **ESCOLA POLITÈCNICA SUPERIOR** |
| Algorítmica i Complexitat - Curs 2021-2022 |
| **Pràctica 2: Recursivitat i Anàlisis de Costos** |
| **Detector d’objectes marins** |
| **Informe presentat per:** Alexandru Cristian Stoia Y2386362B,  i Pol Triquell Lombardo 48054396J  **Grup PraLab:** GM3  **Professorat:** Aitor Corchero, Raúl Ariño i Ian Palacín  **Data d’entrega:** 10/06/2022  **Grau en Enginyeria Informàtica**  **Universitat de Lleida** |
|  |
|  |
|  |
|  |
|  |



**Índex de continguts**

[**I.** **CONTINGUTS** 3](#_Toc101536151)

[**A.** **El primer pensament** 3](#_Toc101536152)

[I. Resolució Iterativa 4](#_Toc101536153)

[II. Pseudocodi 4](#_Toc101536154)

[III. Anàlisis dels Costos. 5](#_Toc101536155)

[IV. Temp**s** d’execució 7](#_Toc101536156)

[**B.** **Solució més eficient en temps** 7](#_Toc101536157)

[I. Pseudocodi 7](#_Toc101536158)

[II. Versió iterativa 8](#_Toc101536159)

[III. Versiórecursiva 8](#_Toc101536160)

[IV. Costos 9](#_Toc101536161)

[V. Anàlisisdels Costos. 9](#_Toc101536162)

[**C.** **Conclusions** 10](#_Toc101536163)

[**II.** **Bibliografía** 11](#_Toc101536164)

# **CONTINGUTS**

## **El primer pensament**

Al presentar la pràctica i fer una primera llegida, vam pensar que la pràctica seria bufar i fer ampolles perquè simplement recorrent l’array que ens passaven per paràmetre en teníem prou però això no va ser així, recórrer l’array i mirar a veure si una posició no estava ocupada retornar aquell número no era una bona solució al problema inicial.

Analitzant l’enunciat, ens vam donar compte que en comptes d’un array, ens passaven tuples amb el nom de la persona que reservava una taula i el número d’aquella taula. Per implementar la solució correcta, no calia precisar del nom per saber quina taula estava lliure. Vam utilitzar el pensament inicial ja que, al nostre parer, era la solució més viable d’implementar, ordenar els números de les taules en un array i si l’índex no coincideix amb el valor que hi ha l’array significa que aquella posició no està ocupada. A continuació, podem apreciar un exemple, que retorna el número de taula 2.

1. Neteja de dades

Per aconseguir les imatges de la matriu determinades per píxels, els seus contorns, és a dir, els elements adjacents a la matriu, han de ser estrictament menors, per tant, donat l’exemple de l’informe podem determinar que els contorns tindran valor igual a -1. Així, tindrem que sí o sí seran estrictament menors perquè sinó el sonar ens estaria donant informació errònia.

Per fer el contorn, inicialitzem una nova imatge incrementant en dos posicions el tamany de la matriu i la inicialitzem tota a -1, així aconseguim una matriu amb el contorn del tamany correcte.

A continuació, copiem la imatge que teníem inicialment a aquesta imatge creada anteriorment amb el nou contorn.

### Resolució Iterativa

Per resoldre el problema plantejat hem utilitzat el pseudocodi i l’hem transformat en funcions reals de python: *createGoodArray(reserves)*

Amb aquesta funció agafem la llista de reserves i les passem a un nou array el qual conté simplement el número de la taula reservada (Sense el nom de la reserva ja que no ens interessa).

Una vegada el tenim, cridem la funció *sort*, d’aquesta forma l’array queda ordenat en ordre ascendent. Una vegada ordenat, només queda veure quina és la primera posició que no correspon amb la seva taula i retornar aquesta:

1. for i in range(0, len(array)):
2. idTable = i
3. if idTable != array[i]:
4. return idTable

Si arribem al final del array, vol dir que totes les taules estan reservades, per tant la següent taula és la n+1 (sent n el número de reserves fetes).

### Pseudocodi

Per resoldre el problema de forma recursiva, hem optat per una crida no final, aprofitant la funció que se’ns ha proporcionat.

Dins d’aquesta, preparem l’array per ser processat, d’igual forma que hem fet al mètode iteratiu. Una vegada tenim l’array preparat cridem a la funció recursivoNoFinal:

En aquesta funció passem un array i un número, aquest array es el array de les taules, el número és el id de la taula més propera.

Una vegada agafat l’array, comprovem si la primera posició correspon amb la primera taula ocupada, en cas contrari retornem el id de la taula. En cas que coincideixin, vol dir que la taula es troba ocupada i passem a analitzar la següent, tornant a cridar la funció amb el mateix array sense la primera posició i augmentant una unitat el id de la taula.

En cas que no hi hagi array, retornem el id de la primera taula (ja que suposem que no hi ha cap reserva, per tant, el restaurant està buit).

### Anàlisis dels Costos.

Donades les dos funcions, iterativa i recursiva, podem observar els costos següents.

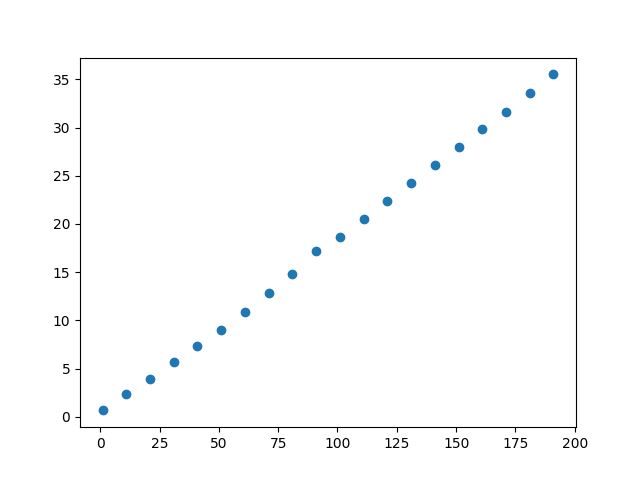


Figura 1: Gràfica de l’algorisme iteratiu

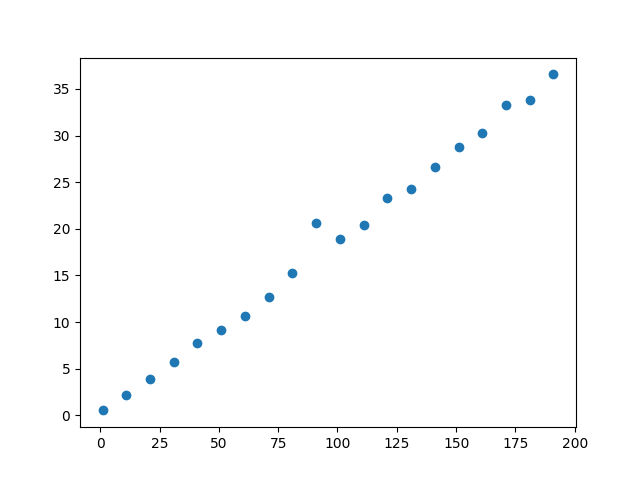


Figura 2: Gràfica de l’algorisme recursiu

Els costos a nivells de números podem veure que sortiria un cost de O(n) segons les gràfiques ja que desglossat seria de la següent forma:

* El primer recorregut que es fa, es per separar el noms de les taules, això comporta un cost O(n)
* Posteriorment, hem d’ordenar l’array de les taules, el qual fent una investigació per internet, hem trobat que la funció té un cost O(nLog(n)) *(Rusin, 2022)*.
* Una vegada el tenim ordenat tornem a recórrer l’array per trobar la primera taula lliure, que comporta una altra vegada un cost O(n)

Com a còmput final, la suma de les tres funcions, el cost seria O(n) + O(nLog(n)+ O(n) = O(2n) + O(nLog(n)), cosa que es pot simplificar a nivell de costos a O(nLog(n)).

Una de les coses que ens ha sorprès es la gràfica, ja que no mostra un comportament de nLog(n), sinó mes linial, és a dir n: Ho hem atribuït al fet que aquest cost de nLog(n) és gairebé insignificatiu en la nostra pràctica, es a dir, amb un número petit de mostres no es veu afectat el temps.

### Temp**s** d’execució

Després de passar els tests, hem provat a executar el *main* i ens hem sorprès al veure que tarda 20 minuts en mostrar els gràfics, en un primer moment hem pensat que era a causa dels prints infinits que fa el programa de les reserves, al treure els prints i tornar a executar el programa no millorava gaire.

## **Solució més eficient en temps**

### Pseudocodi

Després d’haver vist que el nostre cost era de O(n\*log(n)), vam decidir buscar una altra alternativa a veure si d’alguna forma ho podíem fer més eficient i ens va sorgir el següent codi:

Array\_reserves = agafar nomes números de les tuples

Reserva\_mes\_gran = màxim de Array\_reserves

Array\_taules = totes lliures (0)

for reserva in array\_reserves:

array\_taules[reserva] = ocupat (1)

for i in range (0, longitud array reserves):

si lliure

retorn i

retorn longitud array

D’aquesta forma agafem les reserves del restaurant, posteriorment creem un array amb les taules que necessitarem per gestionar les reserves. Una vegada ho tenim, comprovem quina és la primera taula lliure, en cas que no hi hagi cap lliure, retornarem la següent de l’ultima reserva.

### Versió iterativa

Següint el pseudocodi i adaptant algun funcionament junt amb la utilització de subfuncions la nostra solució ha resultat vàlida passant tots els tests:

1. def bookerineManagement\_iterativo(reserves):
3. reserves\_numbers = reserves\_cleaned(reserves) #Nos quedamos solo con los números de la reserva
4. highest\_table = max(reserves\_numbers) #Vemos la mesa más alejada
5. large\_array = [0] \* (highest\_table +1) #Creamos un array que mide el número de mesas +1
7. for i in reserves\_numbers: #"Ocupamos" las mesas que estan reservadas
8. large\_array[i] = 1
10. for i in range (0,len(large\_array)): #Si encontramos una mesa libre devolvemos ese número
11. if large\_array[i] == 0:
12. return i
14. return len(large\_array) #En caso que estén todas llenas, devolvemos la última mesa disponible

### Versiórecursiva

Una vegada hem tingut la versió iterativa, hem transformat la mateixa en una versió recursiva, per fer-ho utilitzem una segona funció auxiliar també recursiva, però no final. D’aquesta forma li passem per paràmetre la taula i cada vegada actualitza el valor.

Aquesta funció auxiliar l’hem utilitzat ja que és mes senzill i resulta més comode passar un array ja net i tractat enlloc del que ens ve d’origen.

### Costos

Figura 1: Gràfica de l’algorisme iteratiu Figura 2: Gràfica de l’algorisme recursiu

Si analitzem les gràfiques podem veure com son O(n), no obstant la recursiva té un comportament un tant curiós, hem atribuit aquest comportament a les reserves que es fan, no hem decidit fer més proves ja que per cada prova el ordinador triga una hora.

### Anàlisisdels Costos.

En aquesta solució hem primat la eficiència a nivell de cost en temps, ja que el crear un array de n posicions és menys eficient a nivell de memòria però té un cost O(n):

* En el primer pas, igual que en la primera versió agafem els números de les reserves, el qual té un cost O(n).
* Posteriorment comprovem quina és la reserva més gran, el qual comporta també un cost O(n).
* El crear el array amb les posicions ja inicialitzades es O(1) *(Rusin, 2022)*
* Una vegada creat hem de plenar el array recorrent l’array de reserves, que torna a ser O(n).
* Finalment, trobar la primera taula lliure comporta tornar a recórrer l’array final, per tant també és O(n)

Si fem el càlcul final dels costos ens resulta en O(4n) + O(1), el qual es O(n).

## **Conclusions**

Desprès d’haver fet dues versions del codi, la primera sent més eficient a nivell de memòria i la segona més eficient a nivell de cost de computació hem vist diferents aspectes:

1. La nostra primera aproximació té un cost de nLog(n), no obstant, com en aquesta pràctica les reserves que es duuen a terme son poques, aquest cost no es veu reflexat en un temps d’execució excessiu.
2. La segona versió tot i teòricament ser més eficient, els gràfics generats indiquen un cost més elevat, suposant que l’eix de les Y’s és el cost, ja que els gràfics no disposen de llegenda.
3. El comportament del punt anterior, suposem que es deu a les reserves que s’han utilitzat per generar el gràfic, per una millor idea del cost s’hauria d’utilitzar la mateixa mostra de reserves per als dos casos i preferentment una mostra bastant ampla.
4. Els gràfics de la primera versió mostren un comportament de O(n) tot i sent O(nLog(n)), hem arribat a la conclusió que la funció sort de python que és la que fa augmentar el cost, casi no és perceptible degut una altra vegada a la poca quantitat de reserves que es produeixen.
5. Els tests en la versió O(n) triguen entre 0,006 i 0,018s, mentre que en l’altra versió son lleugerament superiors.

Una vegada acabada la pràctica hem après millor el procés de transformació de iteratiu a recursiu, entenent els avantatges de les diferents versions, tant la final com la no final. També ens ha confirmat que un programa no pot ser igual d’eficient a nivell de memòria i a nivell de temps, es a dir: si vols una millora en una, comporta un petit augment en l’altra.

# **Bibliografía**

Rusin, R. (2022). *Python.org*. Obtenido de https://wiki.python.org/moin/TimeComplexity