

|  |
| --- |
| **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](#_Toc105771061)

[**A.** **El primer pensament** 3](#_Toc105771062)

[**I.** **Preparació de dades** 3](#_Toc105771063)

[**B.** **El codi** 3](#_Toc105771064)

[**I.** **Resolució Iterativa** 3](#_Toc105771065)

[**II.** **Resolució Recursiva** 4](#_Toc105771066)

[**III.** **Funció isBigger** 4](#_Toc105771067)

[**C.** **Anàlisis dels Costos** 5](#_Toc105771068)

[**D.** **Conclusions** 7](#_Toc105771069)

# **CONTINGUTS**

## **El primer pensament**

Al començar a llegir l’informe, poc a poc hem anat pensant diverses formes de resoldre el problema, alguns dels quals van funcionar i altres que en la teoria funcionaven però finalment tenien algun petit problema.

Una de les primeres propostes va ser fer un cercador de les posicions adjacents en forma de creu, per veure si els valors eren menors que el valor de la posició que estem analitzant de la imatge.

### **Preparació 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, estalviem un pas de crear subfuncions específiques per casos particulars.

Per fer el contorn, inicialitzem una nova imatge incrementant en dos posicions la mida de la matriu i la inicialitzem tota a -1, així aconseguim una matriu amb la qual poder treballar i no tenir errors en els casos de les cantonades.

A continuació, copiem la imatge que teníem inicialment a aquesta imatge creada anteriorment amb el nou contorn, obtenint un marc envoltant la imatge original.

## **El codi**

### **Resolució Iterativa**

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

Amb aquesta funció agafem la imatge del sonar, la adaptem amb la nostra funció *setBorders(image).*

Una vegada tenim la imatge per analitzar, creem un array buit per poder guardar els resultats.

La funció en si, recorre el array element per element, comprovant en creu (dalt, baix, esquera i dreta) si l’element analitzat és el més gran de tots, amb l’ajuda de la subfunció *isBigger*. Si aquesta retorna *true* fa un *append* a la llista dels resultats, en cas contrari, passa al següent element per analitzar-lo, així fins haver analitzat tots els elements del array (Els -1’s no conten com elements a analitzar, per fer-ho més òptim).

1. image = setBorders(image)
2. results = []
3. for i in range(1, len(image) - 1):
4. for j in range(1, len(image) - 1):
5. if isBigger(i, j, image):
6. results.append([i - 1, j - 1])

Si arribem al final del array, retornem el array de solucions.

### **Resolució Recursiva**

D’igual forma que fem en la resolució iterativa, en primer lloc adaptem la imatge per al nostre codi, creant el marc de -1’s i el array per guardar el resultat, posteriorment aquesta imatge la utilitzem en l’algorisme recursiu:

1. if i == len(image) - 2 and j == len(image) - 1:
2. return positions
3. if j == len(image) - 1:
4. return lateralesRecursivo(image, positions, i + 1, 1)
6. if isBigger(i,j,image):
7. positions.append([i - 1, j - 1])
9. return lateralesRecursivo(image, positions, i, j + 1)

L’algorisme es va movent per files, fent la crida recursiva sobre la següent posició de la fila, en cas d’haver arribat al final d’aquesta, passa a la fila següent. Una vegada arribat al final es retorna el resultat. L’anàlisi del entorn per determinar si s’ha d’afegir també es fa amb la funció auxiliar *isBigger.*

### **Funció isBigger**

Aquesta és la funció que comprova si les posicions adjacents a la analitzada són més petites que la central:

La funció rep com paràmetres d’entrada una “i” i una “j”, les quals ens serviran per saber quines posicions hem d’analitzar i també ens entra el array que correspon.

Fem un check utilitzant and’s per comprovar que la posició central de la creu es més gran que totes i cadascuna de les posicions adjacents. Si aquesta ho compleix retorna true, en cas contrari false.

## **Anàlisis dels Costos**

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

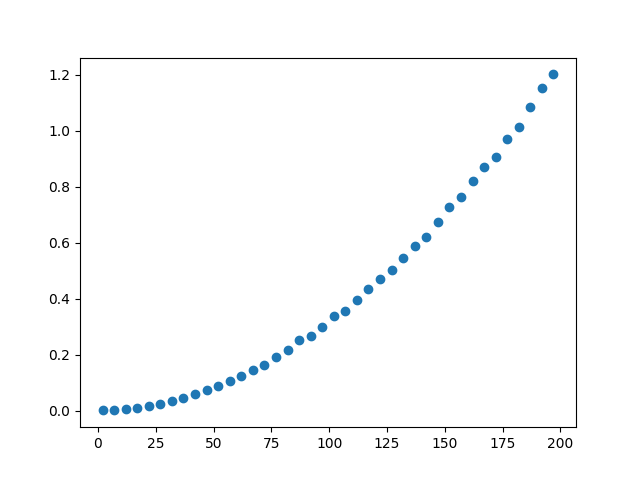


Figura 1: Gràfica de l’algorisme iteratiu

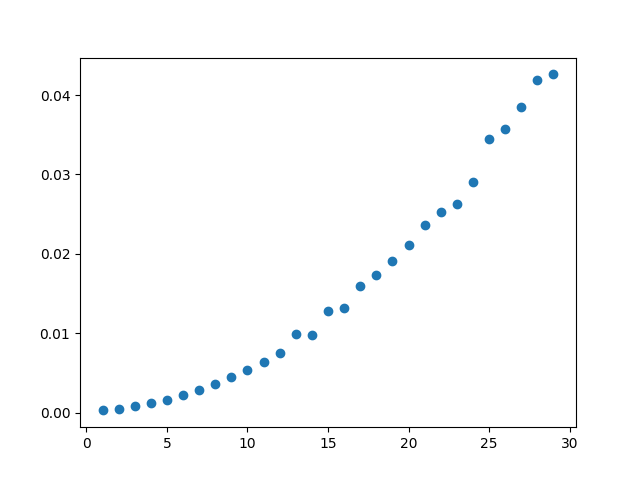


Figura 2: Gràfica de l’algorisme recursiu

Els costos a nivells de números podem veure que sortiria un cost de O(n2) segons les gràfiques, un cop desglossades serien de la següent forma:

* El primer recorregut que es fa es per crear el array de -1’s, el qual representa un cost de O(n2) al ser una matriu de 2 dimensions quadrada.
* Posteriorment, hem d’analitzar les posicions de l’array una per una: O(n2) .

Com a còmput final, la suma de les funcions, el cost seria O(n2) + O(n2) = O(2n2), cosa que es pot simplificar a nivell de costos a O(n2).

Com podem veure a les gràfiques, ens confirmen aquest cost quadràtic del algorisme.

## **Conclusions**

Després d’haver realitzat la pràctica, hem vist un cas real d’aplicació de la visió per ordinador, on ens donen un mapa amb números i nosaltres hem d’analitzar i donar-li sentit al resultat.

Com hem vist, la optimització del codi és molt important, ja que si tenim una imatge més gran, el cost se’ns dispara.

Analitzar el problema i pensar una solució a vegades resulta senzill, però la part realment complicada i en la que ens haurem de centrar serà l’optimització d’aquest codi, ja que no només ha de funcionar sinó que ha de funcionar amb el menor cost possible.