Séance d'exercice 2
Complexité

CPUMons

16 décembre 2020

1 En théorie

La complexité algorithmique peut faire peur à première vue mais ne vous inquiétez pas il suffit de connaitre ces quelques règles et le tour est joué.

1.1 Opération élémentaire

Tout d'abord, il est bon d'apprendre à reconnaitre une opération élémentaire. En effet celle-ci seront toujours en O(1) :

- Affectation
- Opération arithmétique et booléenne
- Accéder à un élement d'un tableau via un indice donné
- Accès aux variables d'instance, aux méthodes d'un objet
- Appeler une méthode ou une fonction
- Retourner une valeur
- Lire / écrire une valeur

Remarque :

- Appeler une fonction $\neq$ exécuter cette fonction
- Lire une valeur $\neq$ lire un fichier contenant n données
- Accéder à un élément d'un tableau n'est une opération élémentaire que si on connaît son indice
- Une opération arithmétique ou booléenne peut contenir un appel à une fonction dont le temps d'exécution doit être analysé.

1.2 Taille de données

Hormis les opérations élementaires, il existe d'autre opérations qui vont demander qu'on exprime le temps d'exécution en fonction de la taille des données. Mais cela peut poser problème car pour une même taille, on peut avoir des temps d'exécution différents.
C'est pourquoi on va vouloir couvrir toutes les données possibles et d'intéresser au temps d'exécution dans le pire des cas.

1.3 Temps d'exécution

Voici la plupart des temps d'exécution que vous pourrez retrouver quand vous codez (dans l'ordre chronologique) :

- O(1)
- O(log n)
- O(n)
- O(n log n)
- O(n^2)
- O(n^3)
- O(n^k)
- O(a^n)
- O(n!)

1.4 Règles de calcul pour la notation grand-O

Nous entrons vraiment dans le calcul de complexité. Il est bon de connaitre ces quelques règles :

1. Les facteurs constants ne sont pas importants
2. Les termes d'ordre inférieur sont négligeables
3. Les termes dont la croissance est inférieure sont négligeables

1.5 Règles de calcul d'algo itératif

1.5.1 Opérations élémentaire

O(1) + O(1) = O(1).
un bloc d'instructions ne contenant que des opérations élémentaires sera évalué en O(1).

1.5.2 Instructions conditionnelles

Si <condition en O(f1(n))> alors <instructions en O(f2(n))>
La règle de calcul revient à calculer O(f1(n) +f2(n)).
On considère le pire des cas où la condition est toujours vraie.

Si <condition en O(f1(n))> alors <instructions en O(f2(n))>, sinon <instructions en O(f3(n)>
On le calculuera de la manière suivante : O(f1(n) + max(f2(n),f3(n))).

1.5.3 Instructions itératives

Tant que <condition en O(f1(n))> alors <instructions en O(f2(n))>
On aura O(g(n)×[f1(n) +f2(n)]) + O(f1(n)).
Il faut évaluer le nombre de passages dans la boucle : O(g(n)) (dans le pire des cas) et il ne faut pas oublier que la condition est testée pour sortir de la boucle. Elle est donc toujours testée au moins une fois.

Pour <déf. incrément en O(f1(n))> faire <instructions en O(f2(n))>
On aura O(g(n)×[f1(n) +f2(n)] + O(f1(n))) (avec g(n) le nbre d'itération)

2 Exercice 1

Supposons que P(n) et Q(n) soient des algorithmes retournant des booléens ; et que A(n), B(n) et C(n) soient trois algorithmes. Considérons le morceau d'algorithme ci-dessous.

```
if P(n) then
  A(n)
else
  if Q(n) then
    B(n)
  else
    C(n)
  end if
end if
```

Quelle est sa complexité si :

1. P(n), Q(n) et C(n) sont en O(n), A(n) en O(n^2) et B(n) en O(1)
2. P(n) et C(n) sont en O(n), B(n) en O(1) et Q(n) et A(n) en O(n^3)

3 Exercice 2

Soit l'algo du tri à bulle ci-dessous :

Require: Un tableau A de n ≥ 1 entiers indicés de 0 à n - 1
Ensure: /
  return Le tableau A est trié par ordre croissant.
  i $\leftarrow$ n - 2
  madeSwap $\leftarrow$ Vrai
  while madeSwap **do**
    madeSwap $\leftarrow$ Faux
    for j allant de 0 à i **do**
      if $A[j] > A[j+1]$ **then**
        échanger A[j] et A[j+1]
        madeSwap $\leftarrow$ Vrai
      end if
    end for
    i$\leftarrow$ i-1
  end while

— expliquez, en français, l'idée utilisée pour trier un tableau
— Analysez sa complexité dans le pire des cas