Complexité

9 octobre 2023

Plan

Retour sur la semaine dernière

Complexité

Retour sur la semaine dernière

Rendu des DM

- Toutes les questions étaient sur 4 sauf les questions de typage sur 2 ;
- Notes sur 60 ramenées sur 20.

Réponse dans le mauvais sens (1)

```
let filtrer_positifs l =
    let rec aux l acc = match l with
    | [] -> acc
    | p::q when p>=0 -> aux q (p::acc)
    | _::q -> aux q p
    in aux l []
```

```
filtrer_positifs [1; -1 ; 2] (* [2; 1] *)
```

Réponse dans le mauvais sens (1)

```
let filtrer_positifs l =
    let rec aux l acc = match l with
    | [] -> acc
    | p::q when p>=0 -> aux q (p::acc)
    | _::q -> aux q p
    in aux l []
```

```
filtrer_positifs [1; -1 ; 2] (* [2; 1] *)
```

```
let rec filtrer_positifs l = match l with
| [] -> []
| p::q when p>= 0 -> p::(aux q)
| _:: q -> aux q
```

Réponse dans le mauvais sens (2)

```
let rec iterateur2 n x = match n with
| 0 -> x
| _ -> iterateur2 f (n-1) (f (n-1) x)
```

```
iterateur2 n x (* f 0 (f 1 (.... ( f (n-1) x) ...))
    *)
```

Réponse dans le mauvais sens (2)

```
let rec iterateur2 n x = match n with
| 0 -> x
| _ -> iterateur2 f (n-1) (f (n-1) x)
```

```
iterateur2 n x (* f 0 (f 1 (.... ( f (n-1) x) ...))
    *)
```

```
>let rec iterateur2 n x = match n with
| 0 -> x
| _ -> f (n - 1) iterateur2 f (n-1) x
```

Confusion dans le typage

Si vous voulez calculer n^2, vous n'avez pas besoin de considérer n comme un flottant et utiliser $**$, vous pouvez simplement écrire `n * n`.
Pas besoin de faire des restrictions sur le typage de vos fonctions.
Attention à l'ordre des arguments.

Cas de base, décalage

```
1 let rec iterateur2 n x = match n with
2     | 0 -> f 0 x
3     | _ -> f (n-1) (iterateur2 f (n-1) x)
```

Cas de base, décalage

```
let rec iterateur2 n x = match n with
    | 0 -> f 0 x
    | _ -> f (n-1) (iterateur2 f (n-1) x)
```

```
let rec iterateur2 n x = match n with
    | 1 -> f 0 x
    | _ -> f (n-1) (iterateur2 f (n-1) x)
```

Cas de base, décalage

```
let rec iterateur2 n x = match n with
    | 0 -> f 0 x
    | _ -> f (n-1) (iterateur2 f (n-1) x)
```

```
let rec iterateur2 n x = match n with
    | 1 -> f 0 x
    | _ -> f (n-1) (iterateur2 f (n-1) x)
```

```
let rec iterateur2 n x = match n with
        | 0 -> x
        | _ -> f (n-1) (iterateur2 f (n-1) x)
```

Problèmes de complexités

Quelques soucis de complexité dans votre code qui n'ont pas été sanctionnés, mais qui le seront après ce chapitre : si votre code est drastiquement moins bon qu'une solution raisonnable, vous pouvez être sanctionné sur le sujet.

Quelques recommendations

- Une petite phrase avant votre code pour expliquer ce que vous faites peut aider la personne qui corrige ;
- L'encre qui s'efface avec la chaleur est une très mauvaise idée ;
- Écrivez dans la mesure du possible chaque code sur une seule page.

Signature des fonctions

Attention : quand on demande de modifier un tableau, il ne faut pas renvoyer un tableau, ni créer un nouveau tableau :

```
1 let modifier t x =
2     let n = Array.length t in
3     Array.make n x
```

Signature des fonctions

Attention : quand on demande de modifier un tableau, il ne faut pas renvoyer un tableau, ni créer un nouveau tableau :

```
let modifier t x =
    let n = Array.length t in
    Array.make n x
```

```
let modifier t x =
    let n = Array.length t in
    for i = 0 to n - 1 do
        t.(i) <- x
    done
```

Renvoyer quelque chose dans la boucle for

On ne peut pas renvoyer quoi que ce soit dans la boucle for, on doit procéder par effet de bord :

```
let palyndrome t =
    let n = Array.length t in
    for i = 0 to n-1 do
        if t.(i)<>t.(n-1-i) then false
    done
```

Renvoyer quelque chose dans la boucle for

On ne peut pas renvoyer quoi que ce soit dans la boucle for, on doit procéder par effet de bord :

```
let palyndrome t =
    let n = Array.length t in
    for i = 0 to n-1 do
        if t.(i)<>t.(n-1-i) then false
    done
```

```
let palyndrome t =
    let n = Array.length t in
    let resultat = ref true in
    for i = 0 to n-1 do
        if !resultat then
            resultat := t.(i)= t.(n-i-1)
    done ;
    !resultat
```

Exercices

S'il y a des exercices que vous voulez traiter de manière spécifique dans les TD, envoyez-moi un e-mail.
Il y aura une correction en cours ou en TD, ou bien j'enverrai une correction par e-mail à la classe.

Complexité

Entrée

Les **entrées** d'un programme ou d'une fonction sont les arguments avec qui ils sont appelés. En faisant varier ces entrées, on peut étudier le comportement du programme asymptotiquement.

Taille des entrée

On regroupe certaines entrées possibles par taille. Cette taille peut varier selon le type des entrées, et selon le problème qui nous intéresse :

- Pour un tableau ou une liste, il s'agit souvent de sa longueur ;
- Pour un entier n, il peut s'agir de sa valeur n, ou du nombre de chiffres dans sa représentation binaire $\log n$;
- Pour une matrice de taille $n \times n$, il peut s'agir de son nombre total d'éléments n^2 ou simplement du nombre de colonnes ou de lignes n.

Opérations de bases

Lors de l'étude de la complexité, on a besoin d'utilisé une unité, des opérations dont le coût est sensé être constant. Cela peut dépendre du problème :

- La plupart du temps, on considère que toutes les opérations de bases du langages (arithmétique, accéder à un élément dans un tableau, vérifier une condition, faire un appel de fonctions...) sont en temps constant ;
- Parfois, on ne s'intéresse qu'à certaines opérations, soit par simplification de l'étude de complexité (par exemple les échanges dans un tableau pour un algo de tri), soit parce que ces opérations sont beaucoup plus couteauses (accès à une ressource sur le disque).

Complexité

La **Complexité temporelle**, ou simplement **complexité**, d'un programme est le nombre d'opérations de base $C(p)$ d'un programme sur une entrée p.

Complexité dans le pire des cas

Définition 1 : Complexité dans le pire des cas

La complexité **dans le pire des cas** est le maximum des complexité sur des entrées d'une taille donnée.

$$C_{pire}(n) = \max_{p,|p|=n} C(p)$$

Complexité dans le meilleur des cas

Définition 2 : Complexité dans le meilleur des cas

La complexité **dans le meilleur des cas** est le minimum des complexité sur des entrées d'une taille donnée.

$$C_{meilleur}(n) = \min_{p,|p|=n} C(p)$$

Complexité en moyenne

Définition 3 : Complexité en moyenne

La complexité **en moyenne** est l'espérance de la complexité sur l'ensemble des entrées d'une taille donnée.

$$C_{moyenne}(n) = \frac{1}{|\{p| \, |p| = n\}|} \sum_{p,|p|=n} C(p)$$

Complexités

On s'intéressera souvent au pire des cas, et parfois au cas moyen lorsque celui-ci est facile à calculer.

Rechercher de l'indice dans un tableau (1)

```
let mem t x =
    let n = Array.lenth t in
    let rec aux i =
        if i>=n then failwith "Element absent du
    tableau"
        else
            if t.(i) = x then i
            else aux (i+1)
    in aux 0
```

Rechercher de l'indice dans un tableau (1)

```
let mem t x =
    let n = Array.lenth t in
    let rec aux i =
        if i>=n then failwith "Element absent du
    tableau"
        else
            if t.(i) = x then i
            else aux (i+1)
    in aux 0
```

En fonction de la taille n du tableau, quelle est la complexité dans le meilleur des cas en terme d'accès au tableau ? Dans le pire des cas ?

Rechercher de l'indice dans un tableau (1)

```
1 let mem t x =
2     let n = Array.lenth t in
3     let rec aux i =
4         if i>=n then failwith "Element absent du
    tableau"
5         else
6             if t.(i) = x then i
7             else aux (i+1)
8     in aux 0
```

En fonction de la taille n du tableau, quelle est la complexité dans le meilleur des cas en terme d'accès au tableau ? Dans le pire des cas ?
Dans le meilleur des cas, l'élément que l'on cherche est là dès le début :

$$C_{meilleur} = 1$$

Rechercher de l'indice dans un tableau (1)

```
let mem t x =
    let n = Array.lenth t in
    let rec aux i =
        if i>=n then failwith "Element absent du
    tableau"
        else
            if t.(i) = x then i
            else aux (i+1)
    in aux 0
```

En fonction de la taille n du tableau, quelle est la complexité dans le meilleur des cas en terme d'accès au tableau ? Dans le pire des cas ?
Dans le meilleur des cas, l'élément que l'on cherche est là dès le début :

$$C_{meilleur} = 1$$

Dans le pire des cas, on doit regarder tous les léméents du tableau :

$$C_{pire} = n$$

Rechercher de l'indice dans un tableau (2)

En supposant que l'élément est aléatoirement placé dans le tableau de manière équiprobable, quelle est la complexité en moyenne ?

Rechercher de l'indice dans un tableau (2)

En supposant que l'élément est aléatoirement placé dans le tableau de manière équiprobable, quelle est la complexité en moyenne ?

$$\begin{aligned} C_{moyenne}(n) &= \frac{1}{n} \sum_{k=1}^{n} k \\ &= \frac{1}{n} \frac{n(n+1)}{2} \\ &= \frac{n+1}{2} \end{aligned}$$

Recherche dichotomique

```
let recherche t x =
    let n = Array.length t in
    let rec aux i j =
        if i>j then false
        else let k = (i+j)/2 in
            if x = t.(k) then true
            else
            if x> t.(k) then aux (k+1) j
            else aux i (k-1)
    in aux 0 (n-1)
```

Recherche dichotomique

```
let recherche t x =
    let n = Array.length t in
    let rec aux i j =
        if i>j then false
        else let k = (i+j)/2 in
            if x = t.(k) then true
            else
            if x> t.(k) then aux (k+1) j
            else aux i (k-1)
    in aux 0 (n-1)
```

On nomme (i_k), et (j_k) les deux suites finies des valeurs successives de i et j. Ainsi $i_0 = 0$ et $j_0 = n$ et à la fin $i_l = j_l + 1$ dans le pire des cas.

Recherche dichotomique

```
let recherche t x =
    let n = Array.length t in
    let rec aux i j =
        if i>j then false
        else let k = (i+j)/2 in
            if x = t.(k) then true
            else
            if x> t.(k) then aux (k+1) j
            else aux i (k-1)
    in aux 0 (n-1)
```

On nomme (i_k), et (j_k) les deux suites finies des valeurs successives de i et j. Ainsi $i_0 = 0$ et $j_0 = n$ et à la fin $i_l = j_l + 1$ dans le pire des cas.

On remarque qu'à toutes les étapes sauf la dernière, $j_k - i_k \leqslant \frac{j_{k-1} - i_{k-1}}{2}$.

Domination

Définition 4 : Domination

Soient f et g deux fonctions de la variable réelle à valeur dans $\mathbb{R}$. On dit que f est **dominée** par g en $+\infty$ lorsqu'il existe des constantes $M \in \mathbb{R}$ et $K > 0$ tels que :

$$\forall x > M, |f(x)| \leqslant K|g(x)|$$

Cette définition dit que la fonction f est asymptotiquement au plus égale à g à un facteur multiplicatif près.
Vous verrez cela plus en détail en mathématiques.

Exemples de domination

- $3n^2 + n^3 = O(n^3)$
- $1 + n^2 = O(n^2)$
- $2^n + n^2 = O(2^n)$
- $2^n + 3^n = O(3^n)$
- $2 + 2^{-n} = O(1)$

Exemples de domination

De manière générale :

- $\log n = O(n)$
- $\forall \alpha \leqslant \beta, n^\alpha = O(n^\beta)$
- $\forall C > 1, \forall \alpha, n^\alpha = O(C^n)$

Exemples de complexités

- $O(1)$: Temps constant ;
- $O(\log n)$: Complexité logarithmique ;
- $O(n)$: Complexité linéaire ;
- $O(n \log n)$: Complexité quasi-linéaire ;
- $O(n^2)$: Complexité quadratique ;
- $O(n^\alpha)$: Complexité polynomiale ;
- $O(C^n)$: Complexité exponentielle.

Abus de notation en informatique

En informatique, on s'autorise parfois à faire dépendre une complexité asymptotique de deux variables. Par exemple $O(f(n, m))$.
En informatique, nous utiliserons cette notation pour indiquer que si l'on fixe l'une des deux variables, la complexité est en $O(f(n, m))$ quand l'autre variable tend vers $+\infty$.

Complexité asymptotique d'un programme

Grâce à l'étude asymptotique par domination de la complexité, on peut se passer des détails.

Complexité asymptotique d'un programme

Grâce à l'étude asymptotique par domination de la complexité, on peut se passer des détails.

```
let produit_matrice mat =
    let n = Array.length mat in
    if n= 0 then 1. else
    let m = Array.length mat.(0) in
    let res = ref 1. in
    for i = 0 to n-1 do
        for j = 0 to m-1 do
            res:= !res * mat.(i).(j)
        done
    done ; !res
```

Complexité asymptotique d'un programme

Grâce à l'étude asymptotique par domination de la complexité, on peut se passer des détails.

```
let produit_matrice mat =
    let n = Array.length mat in
    if n= 0 then 1. else
    let m = Array.length mat.(0) in
    let res = ref 1. in
    for i = 0 to n-1 do
        for j = 0 to m-1 do
            res:= !res * mat.(i).(j)
        done
    done ; !res
```

On sait que l'on va parcourir tous les éléments une fois, et réaliser une opération constante en $O(1)$, on a donc une complexité en $O(nm)$ pour une matrice de taille $n \times m$.

Complexités et temps de calcul

	1e2	1e3	1e4	1e5	1e6	1e9
$O(\log n)$	<1ms	<1ms	<1ms	<1ms	<1ms	<1ms
$O(n)$	<1ms	<1ms	<1ms	<1ms	<1ms	1ms
$O(n \log n)$	<1ms	<1ms	<1ms	<1ms	<1ms	30ms
$O(n^2)$	<1ms	<1ms	<1ms	0.01sec	1sec	11 jours
$O(n^3)$	<1ms	1ms	1 sec	17min	11j	31ma
$O((1.5)^n)$	2h	*non*				

Complexité Spatiale

Définition 5 : Complexité Spatiale

La **Complexité Spatiale** d'un programme sur une entrée donnée est l'espace mémoire occupé par le programme sur cette entrée.

Tout comme la complexité temporelle, on a besoin d'une unité qui sera souvent la place occupée par une variable. Généralement, on ignore la place mémoire occupée par les entrées elle-même.