Semaine 3 : Série d'exercices sur les algorithmes

1 [N1] Quel est le bon algorithme ? – le retour

Lequel des quatre algorithmes suivants permet de calculer la somme des n premiers nombres pairs (par exemple : si $n = 4$, alors s doit valoir $2 + 4 + 6 + 8 = 20$) ?
Expliquez également pourquoi les autres ne fonctionnent pas.

a)

algo1
entrée : *n nombre entier positif*
sortie : s

Si $n = 0$
  Sortir : 0
Sortir : **algo1**$(n-2) + n$

b)

algo2
entrée : *n nombre entier positif*
sortie : s

Si $n = 0$
  Sortir : 0
Sortir : $2 \times$ (**algo2**$(n-1) + n$)

c)

algo3
entrée : *n nombre entier positif*
sortie : s

Si $n = 0$
  Sortir : 0
Sortir : **algo3**$(n-1) + 2n$

d)

algo4
entrée : *n nombre entier positif*
sortie : s

Si $n = 0$
  Sortir : 0
Sortir : **algo4**$(2n-2) + 2n$

2 [N2] Que font ces algorithmes ?

Pour chacun des quatre algorithmes ci-dessous, répondre aux questions suivantes :

a) Que se passe-t-il si on exécute l'algorithme avec la valeur d'entrée 6 ?

b) L'algorithme est-il récursif ou non ?

c) En général, que fait l'algorithme ?
***** Sauriez-vous le démontrer ?

d) Quelle est la complexité de l'algorithme ? (utiliser la notation $\Theta(\cdot)$)

algo1
entrée : *entier naturel n*
sortie : *? ?*

Si $n = 0$
  Sortir : 0
$k \longleftarrow 0$
Pour j allant de 1 à $2n - 1$
  Si j est impair
    $k \longleftarrow k + j$
Sortir : k

algo2
entrée : *entier naturel n*
sortie : *? ?*

Sortir : n^2

algo3
entrée : *entier naturel n* sortie : *? ?*
Si $n = 0$ **Sortir :** 0 **Sortir :** $2n - 1 +$ **algo3**$(n-1)$

algo4
entrée : *entier naturel n* sortie : *? ?*

Si $n = 0$
  Sortir : 0
Si $n = 1$
  Sortir : 1
Sortir : $n +$ **algo4**$(n-2)$

Voici maintenant quatre autres algorithmes.

e) Un seul de ces algorithmes fonctionne correctement : lequel ?

f) Et celui qui fonctionne a un gros défaut : lequel ?

g) Question subsidiaire : que calcule-t-il ?

algo5
entrée : *entier naturel n* sortie : *? ?*
Si $n = 0$ **Sortir :** 0 **Sortir :** $n +$ **algo5**(n)

algo6
entrée : *entier naturel n* sortie : *? ?*
Si $n = 0$ **Sortir :** 0 **Sortir :** $n +$ **algo6**$(n+1)$

algo7
entrée : *entier naturel n* sortie : *? ?*

Si $n = 0$
  Sortir : 0
Si $n = 1$
  Sortir : 1
Sortir : $2 \cdot$ **algo7**$(n-1) -$ **algo7**$(n-2)$

algo8
entrée : *entier naturel n* sortie : *? ?*

Si $n = 0$
  Sortir : 0
Si $n = 1$
  Sortir : 1
Sortir : **algo8**$(2n) +$ **algo8**$(2n-1)$

3 [N1] Au temps des Grecs

Proposez une version récursive de l'algorithme d'Euclide, qui calcule le plus grand diviseur commun de deux entiers naturels a et b [1] :

Algorithme d'Euclide
entrée : *a, b deux entiers naturels* sortie : *pgcd(a,b)*

Tant que $b > 0$
  $r \longleftarrow a \mod b$
  $a \longleftarrow b$
  $b \longleftarrow r$
Sortir : a

Remarque : « $a \mod b$ » dénote le reste de la division euclidienne de a par b.

1. **Note :** par convention, si b est nul, on sort a ; a par contre est non nul par hypothèse.

4 [N2] Au temps des Egyptiens, deuxième partie

Proposez une version récursive de l'algorithme vu à l'exercice 3 de la semaine 2 :

multiplication égyptienne

entrée : *a, b deux entiers naturels non nuls*
sortie : *? ?*

```
x ⟵ a
y ⟵ b
z ⟵ 0
Tant que y ≥ 1
    Si y est pair
        x ⟵ 2 x
        y ⟵ y/2
    Sinon
        z ⟵ z + x
        y ⟵ y − 1
Sortir : z
```

5 [N3] Création d'algorithmes

a) Soit A une chaîne de caractères formée uniquement de lettres usuelles et d'espaces. Par exemple : $A =$« Le seigneur des anneaux ».

Ecrivez un algorithme dont l'entrée est A et dont la sortie est le nombre de « mots » de la chaîne (au sens séquence de lettres, 4 dans l'exemple). Si cela vous facilite la tâches, vous pouvez commencer par supposer qu'il n'y a qu'une seule espace [2] entre deux mots et qu'il n'y en a pas ni au début ni à la fin. Vous pouvez ensuite essayer de modifier votre algorithme pour vous affranchir de ces hypothèses (comme par exemple avec $A =$« Le seigneur des anneaux »).

Quelle est la complexité de votre algorithme ?

b) Soit L une liste d'au moins un nombre entier positif (par exemple, $L = (3, 43, 17, 22, 16)$). Ecrivez un algorithme dont l'entrée est L et dont la sortie est le plus grand nombre de la liste (dans l'exemple : 43). Essayez d'écrire une version itérative (= non récursive) et une version récursive.

Quelles sont les complexités de vos algorithmes ?

c) Soit L une liste d'au moins deux nombres entiers positifs (par exemple, $L = (3, 43, 17, 22, 16)$). Ecrivez un algorithme dont l'entrée est L et dont la sortie est le produit des deux plus grands nombres de la liste (dans l'exemple : $43 \times 22 = 946$).

Quelle est la complexité de votre algorithme ?

Si ce n'est pas déjà le cas, essayez de trouver un algorithme de complexité linéaire.

2. L'espace de l'imprimeur est en effet féminine !

6 [N3] Taille de liste

Soit `L` une liste d'entiers (pas forcément ordonnée et avec de possibles répétitions), comme par exemple `{19, 31, 15, 21}`.

On cherche ici à écrire un algorithme **taille**(L) qui retourne le nombre d'éléments d'une liste L donnée en entrée.

On suppose que l'on dispose d'un autre algorithme **a_element**(L, i) qui nous dit (vrai ou faux) s'il existe un i^{e} élément dans la liste : il répond donc « vrai » si i est inférieur ou égal à la taille de la liste et « faux » sinon. [3]

1. Comment utiliser l'algorithme **a_element**(L, i) pour calculer la taille de L en un temps linéaire (par rapport à cette taille) ?

   Ecrivez un algorithme pour le faire.

2. En vous inspirant de la recherche dichotomique (cf. cours), pourrait-on avoir une complexité moindre que linéaire ?

   Si oui, quelle serait cette complexité et expliquez intuitivement comment procéder.

   Pour les plus motivés : essayez d'écrire un tel algorithme.

Pour aller plus loin

7 [N2] Devinette

Que fait l'algorithme de la page suivante, où L est une liste de nombres *ordonnée* (par ordre croissant), x un nombre de même nature que ceux de L, et a, b deux entiers naturels non nuls ? La notation $\lfloor x \rfloor$ représente la « partie entière par défaut » de x, c.-à-d. le plus grand entier inférieur ou égal à x. Par exemple $\lfloor 1.5 \rfloor = 1$ et $\lfloor 1 \rfloor = 1$.

Cet algorithme est-il correct dans tous les cas ? Quels problèmes pourraient se produire ?

3. Notez qu'il n'est pas évident de toujours avoir un tel algorithme (sans avoir `taille`, bien sûr !). En realité cela dépend de la représentation effective de L. Mais nous faisons ici l'hypothèse qu'un tel algorithme existe pour L.

devinette

entrée : L, x, a, b
sortie : *? ?*

Si a et b sont égaux
  Si x et le a-ième nombre de L sont identiques
    Sortir : a
  Sinon
    Sortir : 0
$c \longleftarrow a + \left\lfloor \frac{b-a}{2} \right\rfloor$
Si x et le c-ième nombre de L sont identiques
  Sortir : c
Sinon, **si** le c-ième nombre de L est plus petit que x
  Sortir : devinette(L, x, $c+1$, b)
Sinon
  Sortir : devinette(L, x, a, $c-1$)

8 [N3] Deviner l'affichage

Soit l'algorithme suivant opérant sur un message (« chaîne de caractères ») et un entier :

yfèkoi

entrée : *message m, entier naturel n*
sortie : *(rien)*

Si $n = 0$
  afficher : m
Pour i de n à 1 (en décroissant)
  yfèkoi(« m «,» i », $n - i$)

où « m «,» i » désigne le message composé de m suivi d'une virgule suivie de l'écriture de i en décimal. Par exemple, si m est le message « *2,3* » et que i vaut 5, alors « m «,» i » sera le message « *2,3,5* ».

1. Donnez l'affichage produit par cet algorithme lorsqu'il est lancé avec les entrées suivantes :
   (a) $m =$ « 3 : » et $n = 3$
   (b) $m =$ « 4 : » et $n = 4$
2. A quoi correspond cet algorithme (par rapport à l'entier n fourni en paramètre) ?

Pour le fun...

Savez-vous démontrer par réccurence que tous les crayons d'une boîte de crayons de couleur sont forcément tous de la même couleur ?

Soit $P(n)$ la proposition : « tous les crayons d'une boîte comptant n crayons de couleur sont de la même couleur. »

$P(1)$ est clairement vraie.

Montrons que $P(n) \implies P(n+1)$:

- soit une boîte contenant $n+1$ crayons de couleur ; ouvrons-la et sortons un crayon ; on la referme et on a donc une boîte comptant n crayons de couleur ;
- par hypothèse de récurrence [4] tous les crayons dans la boîte sont donc de la même couleur ;
- on rouvre la boîte, on y remet le crayon que l'on avait enlevé et on en retire un autre (qui est donc de la même couleur que ceux restés dans la boîte), puis l'on referme la boîte ;
- on se retrouve à nouveau avec une boîte contenant n crayons de couleur, qui sont donc, par hypothèse de récurrence, tous de la même couleur ;
- or le crayon que l'on a sorti est aussi de la même couleur que les autres restés dans la boîte,
- donc tous les $n+1$ crayons sont bien tous de la même couleur, c.-à-d. $P(n+1)$ est vraie.

Trouvez l'erreur dans ce raisonnement (car il y en a une ! `;-)`).

Cours ICC : liens théorie $\longleftrightarrow$ Programmation

Nous avons créé plusieurs exercices de programmation (C++) spécifiques à ce cours pour mieux faire le lien avec la partie théorie.
Il est pour le moment un peu tôt pour vous les proposer (pas encore assez de notions en programmation), mais nous les listons déjà ici pour que dès la semaine 6 vous pensiez à les faire :

- l'exercice 2 de la série 6 de programmation vous proposera de coder une version récursive du calcul de la factorielle d'un nombre ;
- l'exercice 3 de la même série abordera la suite de Fibonacci (récursive) ;
- et l'exercice 6 de cette série reviendra sur la recherche par dichotomie (récursivité) ;
- l'exercice 6 de la série 7 de programmation vous proposera de comparer (au moins) deux méthodes de tri ;
- l'exercice 4 de la série 8 de programmation vous proposera de programmer les tours de Hanoï.

Retrouvez tous les exercices de programmation liés à la partie théorie du cours sous le lien « Exercices de C++ spécifiques à ICC (lien programmation - théorie) » en bas de la page Moodle du cours, dans la section « Ressources complémentaires / Références ».

4. Le but est bien de montrer ici que $P(n) \implies P(n+1)$, c-.à-d. lorsque $P(n)$ est vraie alors $P(n+1)$ l'est aussi.