Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : Partiel

Code UE : S14IN1I1

Année 2019-20

Documents et calculatrice interdits

Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre. Le sujet est long (et le barème de correction en tiendra compte) : vous pouvez donc passer une question vous semblant trop difficile, quitte à y revenir ensuite s'il vous reste du temps. Par ailleurs, toute réponse non correctement rédigée (avec explication précise et concise) ou non justifiée sera considérée comme fausse.

Solution : Tout d'abord, quelques remarques générales :

- Lorsqu'on donne une consigne de brièveté, en particulier avec une limite sur le nombre de lignes, on s'attend à ce qu'elle soit respectée : les longues réponses n'ont pas été pénalisées cette fois, mais sachez que vous vous pénalisez vous-mêmes puisque vous gâchez un temps précieux à écrire de longs commentaires...
- Attention aux calculs de complexité impliquant une boucle : ce n'est pas parce qu'un code contient une boucle qu'on l'exécute nécessairement n fois (si n est une des données du problème) ! Ce n'était en particulier pas le cas dans l'exercice 2, comme le montre les deux premières questions soulignant que le nombre de tours de boucle est égal au résultat qu'on retourne...
- Lorsqu'on demande de décrire ce que retourne un algorithme, on ne veut pas une paraphrase de la façon dont on construit le résultat, mais plutôt une phrase permettant de dire comment est relié la sortie de l'algorithme à son entrée. Il faut donc prendre un peu de recul pour comprendre ce que fait réellement l'algorithme...

Exercice 1 (codage des entiers)

1. Donnez le codage en binaire de l'entier naturel $(159)_{10}$, en précisant brièvement (5 lignes maximum) votre méthode.

Solution : Il existe 2 méthodes :

- On commence par retrouver les puissances de 2 successives :

n	0	1	2	3	4	5	6	7	8
2^n	1	2	4	8	16	32	64	128	256

  Dans $(159)_{10}$, on peut donc placer $(128)_{10} = 2^7$ comme plus grande puissance de 2. Il reste $(159-128)_{10} = (31)_{10}$ qui s'écrit $(16+8+4+2+1)_{10}$ (en effet, $(31)_{10} = (32-1)_{10}$ est un entier précédent une puissance de 2). Ainsi, l'écriture en binaire de l'entier naturel $(159)_{10}$ est $(10011111)_2$.
- On fait une division euclidienne de $(159)_{10}$ par 2, on note le quotient et le reste et on réitère le processus sur le quotient, jusqu'à obtenir un quotient égal à 0. Ensuite, il suffit d'écrire les restes dans l'ordre inverse de leur obtention. On a donc :

$$\begin{aligned}
(159/2)_{10} &= (79+\mathbf{1})_{10}\\
(79/2)_{10} &= (39+\mathbf{1})_{10}\\
(39/2)_{10} &= (19+\mathbf{1})_{10}\\
(19/2)_{10} &= (9+\mathbf{1})_{10}\\
(9/2)_{10} &= (4+\mathbf{1})_{10}\\
(4/2)_{10} &= (2+\mathbf{0})_{10}\\
(2/2)_{10} &= (1+\mathbf{0})_{10}\\
(1/2)_{10} &= (0+\mathbf{1})_{10},
\end{aligned}$$

  soit $(10011111)_2$.

2. Donnez l'entier naturel dont le code binaire est $(1001110)_2$, en précisant brièvement (5 lignes maximum) votre méthode.

Solution : On lit le nombre de la droite vers la gauche (des bits de poids faible vers les bits de poids fort). Le $i$ème chiffre le composant dans cet ordre correspond au coefficient de la $(i-1)$ème puissance de 2 qui entre dans la somme des puissances de 2 qui permet de calculer le nombre. Nommons $(x)_{10}$ le nombre en base 10 à trouver. On a donc :

$$\begin{aligned}(x)_{10} &= (1001110)_2 \\ &= (0\times 2^0 + 1\times 2^1 + 1\times 2^2 + 1\times 2^3 + 0\times 2^4 + 0\times 2^5 + 1\times 2^6)_{10} \\ &= (1\times 2^1 + 1\times 2^2 + 1\times 2^3 + 1\times 2^6)_{10} \\ &= (2+4+8+64)_{10} \\ &= (78)_{10}.\end{aligned}$$

Exercice 2 (logarithme discret) L'algorithme suivant calcule la partie entière du logarithme en base 2 de l'entier positif `n`, c'est-à-dire le plus grand entier ℓ tel que $2^\ell \leq$ `n`.

```
Fonction log2(d n: entier): entier
# n ∈ ℕ*
   ℓ, m : entier;
Début
   ℓ := 0;
   m := n;
   Tant que m > 1 faire
      ℓ := ℓ + 1;
      m := ⌊m/2⌋;     # quotient de la division euclidienne de m par 2
   Fin faire
   retour ℓ;
Fin
```

1. Exécutez l'algorithme `log2` sur l'entier $n = 8$, en montrant toutes les valeurs des variables pendant l'exécution, le résultat et en comptant le nombre de tours de boucle (ou étapes d'itération) effectués.

   Solution : Pour l'entrée $n = 8$ on a les valeurs suivantes des variables pendant l'exécution :

n	ℓ	m
8	0	8
	1	4
	2	2
	3	1
		0

   avec 3 tours de boucle effectués, ce qui donne 3 comme résultat.

   Attention, on s'arrête bien une fois que m prend la valeur 1, puisque la condition de la boucle `Tant que` *demande à ce qu'on continue tant que m est strictement supérieur à 1 : on s'arrête donc dès lors que ce n'est plus vrai, à savoir quand m est inférieur ou égal à 1.*

2. Même question pour l'entier `n` $= 7$.

   Solution : Pour l'entrée `n` $= 7$ on a :

n	ℓ	m
7	0	7
	1	3
	2	1
		0

   avec 2 tours de boucle effectués, ce qui donne 2 comme résultat.

3. Expliquez pourquoi l'algorithme `log2` termine pour toute entrée $n > 0$.

Solution : La variable m a initialement la valeur n, donc un entier strictement positif. Cette valeur est divisée par 2 en prenant l'entier inférieur à chaque tour de boucle, ce qui assure que la nouvelle valeur est toujours un *entier* strictement inférieur. Cela nous garantit que la valeur de m arrivera tôt ou tard à 1, ce qui terminera l'exécution de la boucle et donc de l'algorithme.

Cette question n'a pas été comprise par beaucoup d'entre vous. On ne demande pas de montrer que l'algorithme ne fonctionne pas dans le cas où $n \leq 0$. On demande de montrer que l'algorithme retourne bien un résultat (et donc se termine) dès que son paramètre (ou argument) est strictement positif. On n'a évidemment pas le droit de justifier cela par le fait que l'algorithme calcule le logarithme, qui est bien défini pour $n > 0$, puisqu'alors il faudrait justifier cela... La raison pour laquelle l'algorithme pourrait ne pas retourner un résultat, c'est la boucle `Tant que` *dont on pourrait ne jamais sortir. Il s'agit donc de trouver un argument montrant qu'on finit toujours par sortir de cette boucle, quelle que soit la valeur de l'entrée positive n.*

4. Calculez, en justifiant, le nombre d'opérations élémentaires effectuées par l'algorithme `log2` en fonction de la valeur de n, prise quelconque. Vous pourrez simplifier le résultat en utilisant la notation « grand $\mathcal{O}$ ».

Solution : On exécute les lignes « ℓ `:= 0;` », « m `:=` n`;` » et « `retour` ℓ`;` » seulement une fois chacune, pour un total de 3 opérations élémentaires (deux affectations et un retour). Comme la valeur de m est divisée par 2 à chaque tour de boucle, elle arrive à 1 en $\log_2(n)$ tours de boucle (arrondi à l'entier inférieur), donc les lignes « ℓ `:=` $\ell + 1$`;` » et « m `:=` $\lfloor m/2 \rfloor$`;` » sont exécutées $\log_2 n$ fois (arrondi à l'entier inférieur). Notons que ces deux lignes contiennent 5 opérations élémentaires (deux affectations, une addition, une division, une troncature – ou calcul de la partie entière inférieure). La ligne « `Tant que` $m > 1$ `faire` » est exécutée une fois de plus, quand la condition devient fausse, et on sort de la boucle, donc $\log_2 n + 1$ fois.
Le nombre d'opérations élémentaires exécutées est donc $3 + 5 \log_2 n + (\log_2 n + 1) = 6 \log_2 n + 4$, ce qui peut être simplifié en $\mathcal{O}(\log_2 n)$.

Exercice 3 (algorithmes sur des tableaux) On considère dans cet exercice l'algorithme suivant :

```
Fonction mystère(d A[0...n − 1]: tab_entiers): tab_entiers
    i: entier;
    B[0...n − 1]: tab_entiers;
Début
    Pour i de 0 à n − 1 faire
       B[i] := 0;
    Fin faire
    B[0] := A[0];
    Pour i de 1 à n − 1 faire
       B[i] := A[i] + B[i − 1];
    Fin faire
    retour B;
Fin
```

1. Exécutez l'algorithme `mystère` sur le tableau d'entrée `[2, 1, 3, 2, 1]` en montrant toutes les valeurs des variables pendant l'exécution et le résultat.

Solution : Tout d'abord, il convient de voir que la première boucle `Pour` a simplement pour but d'initialiser toutes les cases du tableau `B` à la valeur 0. Nous allons donc uniquement détailler l'exécution de cet algorithme à partir du moment où l'on est sorti de cette boucle. Les valeurs des variables pendant l'exécution après la 1ère boucle sont les suivantes :

`A`	n	`B`	i
[2, 1, 3, 2, 1]	5	[0, 0, 0, 0, 0]	
		[2, 0, 0, 0, 0]	1
		[2, 3, 0, 0, 0]	2
		[2, 3, 6, 0, 0]	3
		[2, 3, 6, 8, 0]	4
		[2, 3, 6, 8, 9]	5

ce qui donne $[2, 3, 6, 8, 9]$ comme résultat.

2. Décrivez en une phrase le résultat calculé par l'algorithme `mystère`.

Solution : Cet algorithme donne comme résultat un tableau `B` où le i-ème élément est la somme des i premiers éléments du tableau d'entrée `A`, c'est-à-dire

$$B = [A[0], A[0] + A[1], A[0] + A[1] + A[2], \ldots, A[0] + A[1] + \cdots + A[n-1]]$$

C'est un tableau stockant les effectifs cumulés, si on parle en termes statistiques.

3. Calculez le nombre d'opérations élémentaires effectuées par l'algorithme `mystère` en fonction de la longueur n du tableau `A` donné en entrée. Vous pourrez simplifier le résultat en utilisant la notation « grand $\mathcal{O}$ ».

Solution : La ligne de code « `B[`i`] := 0;` » contient une affectation, soit une opération élémentaire. Elle est contenue dans une boucle qui est exécutée n fois et qui, même si cela est « caché », exécute une incrémentation de i à chaque étape, soit 2 opérations élémentaires (une affectation et une addition). De plus, le test pour savoir si l'on reste dans la boucle est réalisé $n + 1$ fois. Donc cette première boucle effectue en tout $4n + 1$ opérations élémentaires.
La ligne « `B[0] := A[0];` » est une affectation réalisée une fois et compte donc pour une opération élémentaire. De même pour la ligne `retour B;`. Donc, en dehors de toute boucle, l'algorithme exécute 2 opérations élémentaires.
La ligne « `B[`i`] := A[`i`] + B[`$i-1$`];` » réalise une affectation, une addition et une soustraction, soit 3 opérations élémentaires. Elle est dans une boucle qui réalise $n-1$ étapes et qui incrémente i à chaque étape, soit 2 opérations élémentaires en plus par étape. De plus, le test pour savoir si l'on reste dans la boucle est réalisé n fois. Cette seconde boucle réalise donc $5(n-1) + n$ opérations élémentaires.
Pour récapituler, on a donc $2 + 4n + 1 + 5(n-1) + n = 3 + 5n + 5n - 5 = 10n - 2$ opérations élémentaires exécutées par cet algorithme, ce qui donne une complexité en $\mathcal{O}(n)$.

4. Modifiez l'algorithme `mystère` pour qu'il s'arrête et signale une erreur (en retournant le tableau vide `[]` comme résultat) si le tableau d'entrée `A` contient un entier strictement négatif.

Solution : La solution la plus simple consiste à faire le test de positivité au fur et à mesure de la visite du tableau `A` :

```
Fonction mystère(d A[0...n − 1]: tab_entiers): tab_entiers
    i: entier;
    B[0...n − 1]: tab_entiers;
Début
    Pour i de 0 à n − 1 faire
       B[i] := 0;
    Fin faire
    B[0] := A[0];
    Pour i de 1 à n − 1 faire
       Si (A[i] < 0) alors
          retour [];
       Sinon
          B[i] := A[i] + B[i − 1];
       Fin si
    Fin faire
    retour B;
Fin
```

Une autre solution intéressante consiste à vérifier dans un premier temps que toutes les cases de `A` contiennent un entier positif ou nul : on écrit donc une fonction `tous_positifs` qui renvoie vrai si et seulement si c'est le cas (et faux sinon).

```
Fonction tous_positifs(d A[0...n − 1]: tab_entiers): booléen
    i: entier;
Début
    Pour i de 0 à n − 1 faire
       Si A[i] < 0 alors
          retour faux;
       Fin si;
    Fin faire;
    retour vrai;
Fin
```

On peut ensuite l'utiliser dans la fonction `mystère` modifiée :

```
Fonction mystère(d A[0...n − 1]: tab_entiers): tab_entiers
    i: entier;
    B[0...n − 1]: tab_entiers;
Début
    Si (tous_positifs(A) = faux) alors
       retour [];
    Fin si     Pour i de 0 à n − 1 faire
       B[i] := 0;
    Fin faire
    B[0] := A[0];
    Pour i de 1 à n − 1 faire
       Si (A[i] < 0) alors
          retour [];
       Sinon
          B[i] := A[i] + B[i − 1];
       Fin si
    Fin faire
    retour B;
Fin
```

Cela a le désavantage de parcourir une autre fois le tableau, mais cela a l'avantage de découper le problème « complexe » en deux sous-problèmes plus simples... C'est aussi une solution plus réutilisable puisque la fonction `tous_positifs` est intéressante en soi.

5. Écrivez le pseudo-code d'un algorithme calculant la moyenne des valeurs d'un tableau `A` supposé non vide.

Solution : Si on choisit de modifier la fonction donnée par l'exercice, on peut simplement modifier le prototype et la dernière ligne :

```
Fonction moyenne(d A[0...n − 1]: tab_entiers): réel
    i: entier;
    B[0...n − 1]: tab_entiers;
Début
    Pour i de 0 à n − 1 faire
       B[i] := 0;
    Fin faire
    B[0] := A[0];
    Pour i de 1 à n − 1 faire
       B[i] := A[i] + B[i − 1];
    Fin faire
    retour B[n − 1] /n;
Fin
```

On peut aussi choisir de ne pas stocker les effectifs cumulés dans un tableau, mais simplement dans une variable entière unique :

```
Fonction moyenne(d A[0...n − 1]: tab_entiers): réel
    i: entier;
    s: réel;
Début
    s := 0;
    Pour i de 0 à n − 1 faire
       s := s + A[i];
    Fin faire
    retour s/n;
Fin
```

Exercice 4 (codage de l'information) On se place dans la peau d'un fournisseur de vidéo à la demande qui doit adapter la définition (haute, moyenne, faible) de la vidéo diffusée en fonction de la qualité (haut débit, faible débit) de la connexion de l'utilisateur. Concrètement, supposons que le fournisseur propose de diffuser ses vidéos à raison de 25 images par seconde (on oublie dans cet exercice la présence de son dans les vidéos et on suppose que les vidéos ne sont pas compressées), avec les définitions suivantes :

— Haute définition : chaque image possède 1280×720 pixels (environ 1 million de pixels)
— Moyenne définition : chaque image possède 640×360 pixels (environ 250 000 pixels)
— Faible définition : chaque image possède 320×180 pixels (environ 50 000 pixels)

Chaque pixel d'une image est codé sur 12 octets.

1. Quelle est la meilleure définition possible que le fournisseur peut garantir à un utilisateur disposant d'une excellente connexion, permettant un débit de 3 gigabits par seconde ? Vous justifierez votre réponse, et vous pourrez vous appuyer sur des calculs approchés.

Solution : Chaque pixel nécessite $12 \times 25 = 300$ octets par seconde pour être diffusé, soit $300 \times 8 = 2400$ bits, soit environ 2,5 kilobits. Avec un débit de 3 gigabits par seconde, soit 3 millions de kilobits par seconde, on peut donc diffuser $3 \times 10^6 / 2{,}5 = 3 \times 10^6 \times 4/10 = 1\,200\,000$ pixels par seconde. Le fournisseur optera donc pour la haute définition.

Ici, outre le calcul qui est intéressant en soi pour des étudiants scientifiques comme vous, on s'attend à ce que vous sachiez que :

- *1 octet équivaut à 8 bits ;*
- *le préfixe giga équivaut à 10^9 ;*
- *le préfixe méga équivaut à 10^6.*

Toute réponse de plus de 10 lignes est souvent difficile à suivre pour le correcteur : il vaut bien réfléchir (en utilisant un brouillon par exemple), pour ensuite ne donner que les calculs utiles pour répondre à la question simplement et précisément.

2. Même question dans le cas d'une connexion moins performante, permettant un débit de 250 mégabits par seconde.

Solution : Avec un débit de 250 mégabits par seconde, soit 250 000 kilobits par seconde, on peut donc diffuser $250\,000 \times 4/10 = 100\,000$ pixels par seconde. Le fournisseur optera donc pour une faible définition.