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

Les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.

Exercice 1 (codage des entiers)

1. Donnez le codage en binaire de l'entier naturel 159, en précisant brièvement (5 lignes maximum) votre méthode.
2. Donnez l'entier naturel dont le code binaire est 1001110, en précisant brièvement (5 lignes maximum) votre méthode.

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

```
fonction log2(n : entier positif):
    ℓ := 0
    m := n
    Tant que m > 1 faire
        ℓ := ℓ + 1
        m := ⌊m/2⌋      # quotient dans la division euclidienne de m par 2
    FinTantQue
    retourner ℓ
```

1. Exécutez l'algorithme `log2` sur l'entier `n` $= 8$, en donnant toutes les valeurs des variables pendant l'exécution, le résultat et en comptant le nombre de tours de boucle effectués.
2. Même question pour l'entier `n` $= 7$.
3. Expliquez pourquoi l'algorithme `log2` termine pour toute entrée `n` > 0.
4. Calculez, en justifiant, le nombre d'instructions effectuées par l'algorithme `log2` en fonction de la valeur de `n`, prise quelconque. Vous pourrez simplifier le résultat en utilisant la notation « grand $\mathcal{O}$ ».

Exercice 3 (algorithmes sur des tableaux) Considérez l'algorithme suivant :

```
fonction mystère(A : tableau d'entiers):
    n := longueur(A)
    B := tableau de longueur n rempli de 0
    B[0] := A[0]
    Pour i de 1 à n-1 faire
        B[i] := A[i] + B[i-1]
    FinPour
    retourner B
```

1. Exécutez l'algorithme `mystère` sur le tableau d'entrée `[2, 1, 3, 2, 1]` en donnant toutes les valeurs des variables pendant l'exécution et le résultat.
2. Décrivez en une phrase le résultat calculé par l'algorithme `mystère`.
3. Calculez le nombre d'instructions 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 $\mathcal{O}$ ».
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. Le résultat restera inchangé dans le cas où le tableau contient seulement des entiers positifs ou nuls.

5. **(bonus)** Écrire le pseudo-code d'un algorithme calculant la moyenne des valeurs d'un tableau `A` supposé non vide.

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 le 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 brièvement votre réponse, et vous pourrez vous appuyer sur des calculs approchés.
2. Même question dans le cas d'une connexion plus médiocre, permettant un débit de 250 mégabits par seconde.

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

Les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.

Solution : Quelques remarques d'ordre générale :

- Lorsqu'on donne une consigne de brièveté, en particulier avec une limite sur le nombres de lignes, on s'attend à ce qu'elle soit respectée : les longues réponses n'ont pas été pénalisées, mais vous vous pénalisez vous-même 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 donnant à l'idée 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, vis-à-vis de 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, en précisant brièvement (5 lignes maximum) votre méthode.

   Solution : 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, on peut donc placer $128 = 2^7$ comme plus grande puissance de 2. Il reste $159 - 128 = 31$ qui s'écrit $16 + 8 + 4 + 2 + 1$ (en effet, $31 = 32 - 1$ est un entier précédent une puissance de 2). Ainsi, l'écriture en binaire de l'entier naturel 159 est 10011111.

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

   Solution : On utilise la table de la question précédente pour sommer les puissances de 2 correspondantes :

   $$2^1 + 2^2 + 2^3 + 2^6 = 2 + 4 + 8 + 64 = 78$$

Exercice 2 (logarithme discret)

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

```
fonction log2(n : entier positif):
    ℓ := 0
    m := n
    Tant que m > 1 faire
        ℓ := ℓ + 1
        m := ⌊m/2⌋      # quotient dans la division euclidienne de m par 2
    FinTantQue
    retourner ℓ
```

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

Solution : Pour l'entrée `n` $= 8$, on a les valeurs suivantes des variables au début de chaque tour de boucle :

	ℓ	`m`
au début	0	8
après 1 tour de boucle	1	4
après 2 tours de boucle	2	2
après 3 tours de boucle	3	1

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 :

	ℓ	`m`
au début	0	7
après 1 tour de boucle	1	3
après 2 tours de boucle	2	1

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` ≤ 0*. On demande de montrer que l'algorithme retourne bien un résultat dès que son 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 définit 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 pas sortir de cette boucle, quelle que soit la valeur de l'entrée* `n`*.*

4. Calculez, en justifiant, le nombre d'instructions 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 « `retourner` ℓ » seulement une fois chacune, pour un total de 3 instructions. Comme la valeur de m est divisée par 2 à chaque tour de boucle, elle arrive à 1 en $\lfloor \log_2 n \rfloor$ tours de boucle (arrondi à l'entier inférieur, $\lfloor x \rfloor$ étant une notation pour la partie entière de x), donc les lignes « ℓ `:= ` ℓ `+ 1` » et « `m := ⌊ m/2 ⌋` » sont exécutées $\lfloor \log_2 n \rfloor$ fois. 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 $\lfloor \log_2 n \rfloor + 1$ fois.
Le nombre de lignes de code exécutées est donc $3 + 2\lfloor \log_2 n \rfloor + (\lfloor \log_2 n \rfloor + 1) = 3\lfloor \log_2 n \rfloor + 4$, ce qui peut être simplifié en $\mathcal{O}(\log_2 n)$, puisque $3\lfloor \log_2 n \rfloor + 4 \leq 7 \log_2 n$ dès lors que $n \geq 2$.

Exercice 3 (algorithmes sur des tableaux) Considérez l'algorithme suivant :

```
fonction mystère(A : tableau d'entiers):
    n := longueur(A)
    B := tableau de longueur n rempli de 0
    B[0] := A[0]
    Pour i de 1 à n-1 faire
        B[i] := A[i] + B[i-1]
    FinPour
    retourner B
```

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

Solution : Les valeurs des variables pendant l'exécution sont les suivantes :

	`A`	`n`	`B`	`i`
après l'initialisation de la variable `B`	[2, 1, 3, 2, 1]	5	[0, 0, 0, 0, 0]	
au début de la boucle `Pour`			[2, 0, 0, 0, 0]	1
après 1 tour de boucle `Pour`			[2, 3, 0, 0, 0]	2
après 2 tour de boucle `Pour`			[2, 3, 6, 0, 0]	3
après 3 tour de boucle `Pour`			[2, 3, 6, 8, 0]	4
après 4 tours de boucle `Pour`			[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 terme statistiques.

3. Calculez le nombre d'instructions 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 : Les lignes de code « `n := longueur(A)` », « `B := tableau de longueur n rempli de 0` », « `B[0] := A[0]` » et « `retourner B` » sont exécutées une fois chacune, pour un total de 4 instructions. La ligne « `B[i] := A[i] + B[i-1]` » est exécute pour chaque valeur de `i` de 1 à `n` $- 1$, donc `n` $- 1$ fois. La ligne « `Pour i de 1 à n-1 faire` » est exécutée une fois de plus (quand `i` atteint la valeur `n` et on sort de la boucle), donc `n` fois. Cela donne un total de $4 + (n-1) + n = 2n - 3$ instructions, ce qui est $\mathcal{O}(n)$.

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. Le résultat restera inchangé dans le cas où le tableau contient seulement des entiers positifs ou nuls.

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(A : tableau d'entiers):
    n := longueur(A)
    B := tableau de longueur n rempli de 0
    Si A[0] < 0 alors
        retourner []
    FinSi
    B[0] := A[0]
    Pour i de 1 à n-1 faire
        Si A[i] < 0 alors
            retourner []
        Sinon
            B[i] := A[i] + B[i-1]
        FinSi
    FinPour
    retourner B
```

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(A : tableau d'entiers):
    n := longueur(A)
    Si A[0] < 0 alors
        retourner faux
    FinSi
    retourner vrai
```

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

```
fonction mystère(A : tableau d'entiers):
    Si tous_positifs(A) alors  # équivaut au test tous_positifs(A)=vrai
        n := longueur(A)
        B := tableau de longueur n rempli de 0
        B[0] := A[0]
        Pour i de 1 à n-1 faire
            B[i] := A[i] + B[i-1]
        FinPour
        retourner B
    Sinon
        retourner []
    FinSi
```

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

5. **(bonus)** Écrire 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 précédente, on peut simplement modifier la dernière ligne :

```
fonction moyenne(A : tableau d'entiers):
    n := longueur(A)
    B := tableau de longueur n rempli de 0
    B[0] := A[0]
    Pour i de 1 à n-1 faire
        B[i] := A[i] + B[i-1]
    FinPour
    retourner B[n-1] / n    # ligne modifiée
```

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

```
fonction moyenne(A : tableau d'entiers):
    n := longueur(A)
    somme := 0
    Pour i de 0 à n-1 faire
        somme := somme + A[i]
    FinPour
    retourner somme / n
```

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 le 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 brièvement 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 image. Le fournisseur optera donc pour la haute définition.
   Ici, outre le calcul qui est intéressant en soi pour des étudiants scientifiques comme vous, il est à noter qu'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 par le correcteur : il vaut mieux réfléchir sur un brouillon pour ensuite ne donner que les calculs utiles pour répondre à la question simplement.

2. Même question dans le cas d'une connexion plus médiocre, 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.