Aix*Marseille université

PORTAIL ET L1EN2ANS DESCARTES

Introduction à l'informatique : Examen
Code UE : SPO1U65

Année 2023-24

Documents et calculatrice interdits - 2 heures

Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre. Vous pouvez passer une question vous semblant trop difficile, quitte à y revenir ensuite. Par ailleurs, toute réponse non correctement rédigée (avec explication précise et concise) ou non justifiée sera considérée comme fausse.

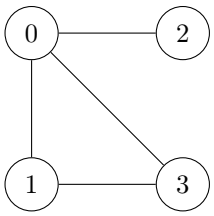
Exercice 1 (Étude d'un algorithme) Considérons l'algorithme suivant, qui prend en entrée un tableau `A` d'entiers :

```
def mystère(A):
    n = len(A)
    j = 1
    while j < n:
        for i in range(n):
            A[i] = A[i] + A[j]
        j = 2 * j
    return A
```

1. Exécuter l'algorithme `mystère` sur l'entrée $[4, 1, 3, 2]$ en remplissant une table qui montre l'évolution des valeurs de toutes les variables le long de l'exécution. Quel est le résultat renvoyé par l'algorithme sur cette entrée ?
2. Expliquer pourquoi l'algorithme `mystère` s'arrête sur n'importe quel tableau d'entiers en entrée.
3. Combien d'instructions (lignes de code) sont exécutées par l'algorithme `mystère` sur une entrée de longueur n ? Répondre en faisant d'abord un calcul détaillé et justifié du nombre d'instructions et, ensuite, en simplifiant avec la notation « grand O ».

Exercice 2 (Écriture d'un algorithme) Considérons dans cet exercice des graphes non orientés, représentés par leur matrice d'adjacence. *Dans tout cet exercice, on demande d'écrire des programmes en Python, sans utiliser les fonctions Python de base, en dehors de* `len` *et* `range` *: ainsi, par exemple, l'utilisation des fonctions* `sum`*,* `max`*,* `sqrt` *ou* `pow` *n'est pas autorisée.*

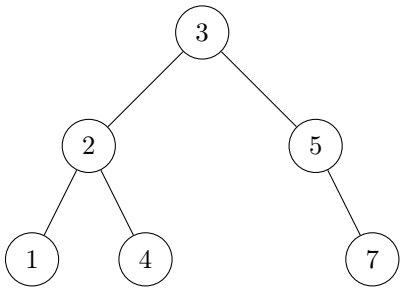
1. Donner la matrice d'adjacence représentant le graphe ci-dessous.



2. Écrire le code Python d'une fonction `degré(M, u)` qui prend en argument la matrice d'adjacence d'un graphe non orienté (*on ne vérifiera pas que la matrice d'adjacence vérifie la condition nécessaire pour représenter un graphe non orienté*) et un sommet de ce graphe et renvoie le degré de ce sommet dans le graphe, c'est-à-dire le nombre d'arêtes adjacentes à ce sommet. Par exemple, dans le graphe ci-dessus, le sommet 0 a degré 3, alors que le sommet 2 a degré 1.
3. En utilisant la fonction de la question précédente, écrire une fonction Python `degré_max(M)` qui calcule le degré maximal d'un sommet parmi les sommets du graphe représenté par la matrice d'adjacence fournie en argument.

Exercice 3 (Arbres binaires)

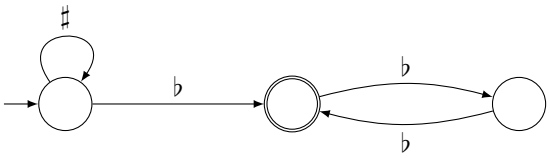
1. Donner (sans justification) l'ordre de parcours suffixe (ou postfixe) de l'arbre binaire ci-dessous :



2. L'arbre ci-dessus est-il un arbre binaire de recherche ? *Justifiez votre réponse.*
3. Dessiner un arbre binaire de recherche dont le parcours préfixe est 4, 2, 3, 6, 5, 7.

Exercice 4 (Automates finis)

1. Donner l'alphabet ainsi que l'ensemble de toutes les séquences (ou mots) acceptées par l'automate ci-dessous.



2. Dessiner un automate qui reconnaît l'ensemble des séquences sur l'alphabet $\{0, 1\}$ contenant un nombre de 0 congru à 2 ou 3 modulo 5 (c'est-à-dire dont le reste dans la division euclidienne par 5 vaut 2 ou 3). Par exemple, les séquences qui contiennent un nombre de zéros égal à 2, ou 3, ou 7, ou 8, ou 12, ou 13, ou 17, ou 18, etc. doivent être acceptées. Les séquences peuvent contenir autant de 1 que possible, à toutes les positions.
3. Dessiner un automate qui reconnaît l'ensemble des séquences sur l'alphabet $\{a, b, c, d\}$ où chaque lettre a est directement suivie d'une lettre b et où chaque lettre c est directement suivie d'une lettre d (il n'y a donc pas de condition particulière pour les lettres suivants b et d). Par exemple, les séquences *abcd*, *cdddbab* et *dddbb* sont acceptées, mais pas les séquences *bacd* ou *cdcddda*. La séquence vide est également acceptée.

Aix*Marseille université

PORTAIL ET L1EN2ANS DESCARTES

Introduction à l'informatique : Examen

Code UE : SPO1U65

Année 2023-24

Documents et calculatrice interdits - 2 heures

Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre. Vous pouvez passer une question vous semblant trop difficile, quitte à y revenir ensuite. Par ailleurs, toute réponse non correctement rédigée (avec explication précise et concise) ou non justifiée sera considérée comme fausse.

Exercice 1 (Étude d'un algorithme) Considérons l'algorithme suivant, qui prend en entrée un tableau `A` d'entiers :

```
1 def mystère(A):
2     n = len(A)
3     j = 1
4     while j < n:
5         for i in range(n):
6             A[i] = A[i] + A[j]
7         j = 2 * j
8     return A
```

1. Exécuter l'algorithme `mystère` sur l'entrée $[4, 1, 3, 2]$ en remplissant une table qui montre l'évolution des valeurs de toutes les variables le long de l'exécution. Quel est le résultat renvoyé par l'algorithme sur cette entrée ?

Solution : Voici la table avec les valeurs des variables :

A	n	j	i
$[4, 1, 3, 2]$	4	1	0
$[5, 1, 3, 2]$			1
$[5, 2, 3, 2]$			2
$[5, 2, 5, 2]$			3
$[5, 2, 5, 4]$		2	0
$[10, 2, 5, 4]$			1
$[10, 7, 5, 4]$			2
$[10, 7, 10, 4]$			3
$[10, 7, 10, 14]$		4	

Le résultat de l'exécution de l'algorithme est la valeur finale de A, c'est-à-dire $[10, 7, 10, 14]$.

2. Expliquer pourquoi l'algorithme `mystère` s'arrête sur n'importe quel tableau d'entiers en entrée.

Solution : L'algorithme comporte deux boucles imbriquées ; il est donc nécessaire de justifier la terminaison de chacune des boucles en question.

- La boucle `for` intérieure est exécutée pour toute valeur de i dans l'intervalle $0, \ldots, n-1$, où n est la longueur de A et donc une valeur finie. Chaque exécution de cette boucle s'arrête donc après n itérations.
- La boucle `while` extérieure est exécutée à partir de $j = 1$ (donc une valeur strictement positive) et la valeur de j est doublée à chaque tour de la boucle (ligne 7). Comme la valeur de n ne change pas, à un moment donné la valeur de i va atteindre ou dépasser celle de n (En effet, si $n = 0$ ou $n = 1$ les instructions à l'intérieur de la boucle `while` ne sont jamais exécutées.)

Toute instruction en dehors de la boucle `while` est exécutée une seule fois, ce qui démontre la terminaison de l'algorithme.

3. Combien d'instructions (lignes de code) sont exécutées par l'algorithme `mystère` sur une entrée de longueur n ? Répondre en faisant d'abord un calcul détaillé et justifié du nombre d'instructions et, ensuite, en simplifiant avec la notation « grand O ».

Solution : On sait que l'algorithme termine pour n'importe quel tableau d'entiers en entrée, donc les lignes 2, 3 et 8 sont exécutées exactement une fois chacune, ce qui donne 3 instructions.
Comme la valeur de j est doublée à chaque itération jusqu'à atteindre ou dépasser n, l'intérieur de la boucle `while` est exécuté $\log_2 n$ fois (plus précisément, $\lceil \log_2 n \rceil$ fois si $n > 0$ et 0 fois si $n = 0$), la ligne 4 une fois de plus (correspondant au moment quand la condition $j \geq n$ interrompt l'exécution de la boucle), donc $\log_2 n + 1$. Pour ce qui concerne l'intérieur de la boucle `while`, à chaque itération on a :

— n exécutions de la ligne 6, ainsi que $n + 1$ de la ligne 5 ;

— une exécution de la ligne 7.

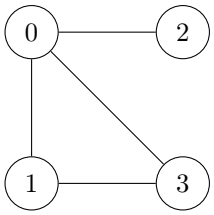
Donc, à chaque tour de la boucle `while` on exécute $2n + 2$ instructions, pour un total de $(2n + 2) \log_2 n$. En ajoutant les exécutions des lignes 2, 3, 4 et 8 on obtient donc

$$1 + 1 + (\log_2 n + 1) + (2n + 2) \log_2 n + 1 = 2n \log_2 n + 3 \log_2 n + 4$$

instructions, ce qui est $O(n \log_2 n)$.

Exercice 2 (Écriture d'un algorithme) Considérons dans cet exercice des graphes non orientés, représentés par leur matrice d'adjacence. *Dans tout cet exercice, on demande d'écrire des programmes en Python, sans utiliser les fonctions Python de base, en dehors de* `len` *et* `range` *: ainsi, par exemple, l'utilisation des fonctions* `sum`*,* `max`*,* `sqrt` *ou* `pow` *n'est pas autorisée.*

1. Donner la matrice d'adjacence représentant le graphe ci-dessous.



Solution : La matrice d'adjacence du graphe ci-dessus est

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

2. Écrire le code Python d'une fonction `degré(M, u)` qui prend en argument la matrice d'adjacence d'un graphe non orienté (*on ne vérifiera pas que la matrice d'adjacence vérifie la condition nécessaire pour représenter un graphe non orienté*) et un sommet de ce graphe et renvoie le degré de ce sommet dans le graphe, c'est-à-dire le nombre d'arêtes adjacentes à ce sommet. Par exemple, dans le graphe ci-dessus, le sommet 0 a degré 3, alors que le sommet 2 a degré 1.

Solution :

```
def degré(M, u):
    n = len(M)
    d = 0
    for v in range(n):
        if M[u][v] == 1:
            d += 1
    return d
```

3. En utilisant la fonction de la question précédente, écrire une fonction Python `degré_max(M)` qui calcule le degré maximal d'un sommet parmi les sommets du graphe représenté par la matrice d'adjacence fournie en argument.

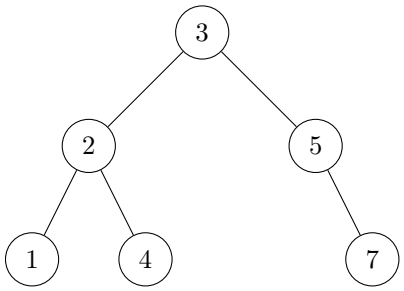
Solution :

```
def degré_max(M):
    n = len(M)
    d_max = 0
    for u in range(n):
        d = degré(M, u)
        if d > d_max:
            d_max = d
    return d_max
```

Page 1 sur 2...

Exercice 3 (Arbres binaires)

1. Donner (sans justification) l'ordre de parcours suffixe (ou postfixe) de l'arbre binaire ci-dessous :



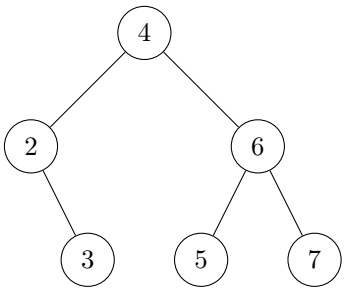
Solution : L'ordre de parcours suffixe de l'arbre binaire donné est 1, 4, 2, 7, 5, 3.

2. L'arbre ci-dessus est-il un arbre binaire de recherche ? *Justifiez votre réponse.*

Solution : L'arbre donné n'est pas un arbre binaire de recherche car le sous-arbre gauche de la racine étiqueté par la valeur 3 contient un nœud étiqueté par la valeur 4, alors que $4 > 3$ et 4 devrait donc appartenir au sous-arbre droit de la racine.

3. Dessiner un arbre binaire de recherche dont le parcours préfixe est 4, 2, 3, 6, 5, 7.

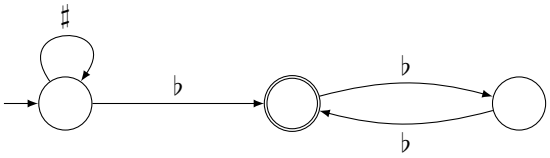
Solution : Un arbre binaire de recherche possible est le suivant :



Notons qu'on n'a en fait pas le choix. Le fait que le parcours préfixe commence par 4 *implique que la racine de l'arbre binaire de recherche doit être étiquetée par* 4*. Ensuite, les éléments* 2 *et* 3 *doivent donc appartenir au sous-arbre gauche de la racine, et les éléments* $7, 5, 6$ *au sous-arbre droit. Le même raisonnement permet de construire récursivement de façon unique chacun des deux sous-arbres.*

Exercice 4 (Automates finis)

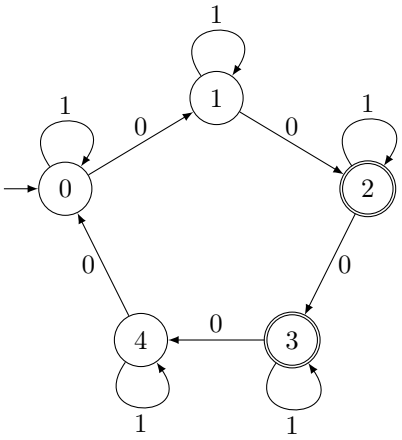
1. Donner l'alphabet ainsi que l'ensemble de toutes les séquences (ou mots) acceptées par l'automate ci-dessous.



Solution : L'automate a pour alphabet $\{\sharp, \flat\}$. Il reconnaît l'ensemble des séquences contenant d'abord un nombre quelconque de $\sharp$, suivi d'un nombre impair de $\flat$.

2. Dessiner un automate qui reconnaît l'ensemble des séquences sur l'alphabet $\{0, 1\}$ contenant un nombre de 0 congru à 2 ou 3 modulo 5 (c'est-à-dire dont le reste dans la division euclidienne par 5 vaut 2 ou 3). Par exemple, les séquences qui contiennent un nombre de zéros égal à 2, ou 3, ou 7, ou 8, ou 12, ou 13, ou 17, ou 18, etc. doivent être acceptées. Les séquences peuvent contenir autant de 1 que possible, à toutes les positions.

Solution :



3. Dessiner un automate qui reconnaît l'ensemble des séquences sur l'alphabet $\{a, b, c, d\}$ où chaque lettre a est directement suivie d'une lettre b et où chaque lettre c est directement suivie d'une lettre d (il n'y a donc pas de condition particulière pour les lettres suivants b et d). Par exemple, les séquences *abcd*, *cdddbab* et *dddbb* sont acceptées, mais pas les séquences *bacd* ou *cdcddda*. La séquence vide est également acceptée.

Solution :