Aix*Marseille université

PORTAIL RENÉ DESCARTES

Science informatique : Examen
2h

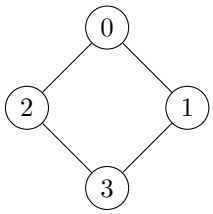
Année 2022-23

Documents et calculatrice interdits

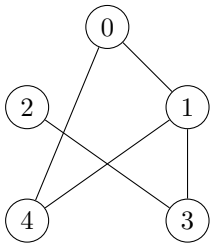
Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.

Exercice 1

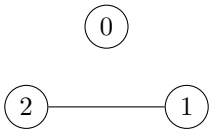
(G1)



(G2)



(G3)



1. Donnez le degré de chaque nœud de chacun des graphes ci-dessus.
2. Pour chacun des graphes ci-dessus, donnez la somme des degrés des nœuds du graphe et le nombre total d'arêtes du graphe.
3. Quelle relation y a-t-il entre le nombre total d'arêtes d'un graphe et la somme des degrés des nœuds de ce graphe ? Pouvez-vous l'expliquer ?
4. On considère maintenant un graphe avec n nœuds, où chaque nœud est de degré k. Donnez la somme $S(k, n)$ des degrés des nœuds du graphe en fonction de k et n.
5. En vous appuyant sur vos réponses aux questions précédentes, expliquez pourquoi dans un groupe de 17 personnes il n'est pas possible que chaque personne ait exactement 5 amis au sein du groupe (pour simplifier, vous pouvez supposer que les liens d'amitié sont toujours réciproques).

Exercice 2

Donnez un automate ayant pour alphabet `{a, b, c}` et reconnaissant toutes les séquences de caractères dans cet alphabet qui finissent par `baba`.

Exercice 3 Voici la table de transitions vue en cours pour une machine de Turing (machine `T1`) permettant d'incrémenter un entier naturel.

état	symbole	sens	nouveau symbole	nouvel état
D	0	$\rightarrow$	0	D
D	1	$\rightarrow$	1	D
D		$\leftarrow$		G
G	1	$\leftarrow$	0	G
G	0	$\leftarrow$	1	OK
G		$\leftarrow$	1	OK

L'entier à incrémenter est écrit sur la bande de la machine en notation binaire. Cette machine a D comme état initial et s'arrête si elle atteint l'état OK. Comme d'habitude, la tête de lecture est initialement placée sur la case contenant le premier symbole du mot d'entrée (ici le chiffre le plus à gauche dans la notation binaire du nombre à incrémenter).

1. Exécutez la machine de Turing `T1` sur l'entrée `101` en montrant à chaque étape le contenu du ruban, la position de la tête de lecture et l'état de la machine.
2. En vous inspirant de la machine `T1`, proposez une machine de Turing `T2` permettant de décrémenter un entier naturel strictement positif écrit sur la bande en notation binaire.
3. Voici une table de transitions pour une machine de Turing `T3` acceptant les entiers écrits en binaire différents de 0. L'état initial est Z.

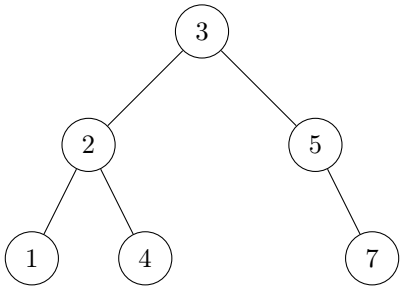
état	symbole	sens	nouveau symbole	nouvel état
Z	0	$\rightarrow$	0	Z
Z	1	$\rightarrow$	1	OK

Et voici une table de transitions pour une machine de Turing `T4` décalant un nombre écrit en binaire d'une case vers la gauche. L'état initial est DM.

état	symbole	sens	nouveau symbole	nouvel état
DM	0	$\rightarrow$	0	DM
DM	1	$\rightarrow$	1	DM
DM		$\leftarrow$		M
M	0	$\leftarrow$		$M0$
M	1	$\leftarrow$		$M1$
$M0$	0	$\leftarrow$	0	$M0$
$M0$	1	$\leftarrow$	0	$M1$
$M0$		$\rightarrow$	0	OK
$M1$	0	$\leftarrow$	1	$M0$
$M1$	1	$\leftarrow$	1	$M1$
$M1$		$\rightarrow$	1	OK

Décrivez comment vous pourriez vous inspirer de machines de Turing `T1`, `T2`, `T3` et `T4` pour construire une machine de Turing permettant de calculer la somme de deux entiers naturels écrit sur la bande en notation binaire et séparés par un symbole `+` (l'alphabet de la machine de Turing serait donc `{0,1,+}`). Par exemple, pour le calcul de $3 + 2$, la bande contiendrait initialement `11+10` et le résultat final serait `101`. On vous demande seulement de donner l'idée générale, pas besoin de donner une table de transition explicite pour cette question.

Exercice 4 Un parcours alternatif pour les arbres (binaires) est le *parcours par niveaux*, dans lequel les nœuds sont traités (par exemple affichés) niveau par niveau, de la racine vers les feuilles, et de gauche à droite. Par exemple, considérons l'arbre suivant :



Dans un parcours par niveaux, ses nœuds sont affichés dans l'ordre : 3, 2, 5, 1, 4, 7.

1. En supposant de disposer d'un algorithme `afficher_niveau(arbre, i)`, qui affiche le niveau i de l'arbre (où 0 est le niveau de la racine, 1 le niveau des enfants de la racine, etc.) et d'un algorithme `hauteur(arbre)` pour calculer la hauteur, on peut écrire le parcours par niveaux comme suit :

```
def parcours_par_niveaux(arbre):
    h = hauteur(arbre)
    for i in range(h + 1):
        afficher_niveau(arbre, i)
```

   Écrivez l'algorithme `afficher_niveau(arbre, i)` de façon récursive. Votre algorithme devra manipuler les arbres binaires uniquement par le biais des fonctions suivantes (vues en cours) : `nouveau_nœud(valeur, enfant_gauche, enfant_droit)`, `valeur(nœud)`, `enfant_gauche(nœud)`, `enfant_droit(nœud)` et `est_arbre_vide(nœud)`.

2. L'algorithme `parcours_par_niveaux(arbre)` ainsi obtenu a le défaut que chaque nœud (en dehors des feuilles) est traversé plusieurs fois, puisque pour accéder aux nœuds d'un certain niveau il est toujours nécessaire de passer par les nœuds des niveaux précédents. Combien de fois traverse-t-on la racine dans le pire des cas pour un arbre de n nœuds ? Combien de fois traverse-t-on un nœud de niveau i dans le pire des cas ? En déduire (en justifiant votre réponse) que le nombre total de traversées de nœuds effectuées par `parcours_par_niveaux` est en $O(n^2)$ dans le pire des cas.

3. Il est possible d'écrire un algorithme plus efficace pour le parcours par niveaux, qui traverse chaque nœud une seule fois, en utilisant une file comme structure auxiliaire (on manipulera les files uniquement par le bias des algorithmes de base vus en cours `nouvelle_file()`, `enfiler(F, x)`, `défiler(F)` et `est_file_vide(F)`). L'idée est d'insérer les enfants du nœud courant dans la file après l'avoir affiché. Écrivez cet algorithme.

Aix*Marseille université

Portail René Descartes

Science informatique : Examen
2h

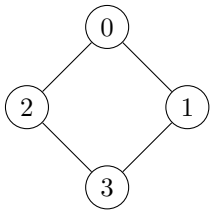
Année 2022-23

Documents et calculatrice interdits

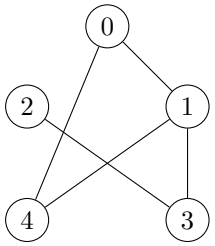
Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre.

Exercice 1

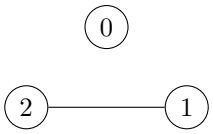
(G1)



(G2)



(G3)



1. Donnez le degré de chaque nœud de chacun des graphes ci-dessus.

   Solution : Pour G1, tous les nœuds sont de degré 2. Pour G2, le nœud 2 est de degré 1, les nœuds 0, 3 et 4 sont de degré 2 et le nœud 1 est de degré 3. Pour G3, le nœud 0 est de degré 0 et les nœuds 1 et 2 sont de degré 1.

2. Pour chacun des graphes ci-dessus, donnez la somme des degrés des nœuds du graphe et le nombre total d'arêtes du graphe.

   Solution : Pour G1, la somme des degrés des nœuds est 8 et le nombre d'arêtes est 4. Pour G2, la somme des degrés des nœuds est 10 et le nombre d'arêtes est 5. Pour G3 la somme des degrés des nœuds est 2 et le nombre d'arêtes est 1.

3. Quelle relation y a-t-il entre le nombre total d'arêtes d'un graphe et la somme des degrés des nœuds de ce graphe ? Pouvez-vous l'expliquer ?

   Solution : La sommme des degrés des nœuds est égale au double du nombre d'arêtes. Cela s'explique intuitivement parce que chaque arête compte pour une unité dans le décompte des degrés de deux nœuds différents (ou pour deux unités dans le décompte du degré d'un seul nœud dans le cas d'une boucle).

4. On considère maintenant un graphe avec n nœuds, où chaque nœud est de degré k. Donnez la somme $S(k, n)$ des degrés des nœuds du graphe en fonction de k et n.

   Solution : On a $S(k, n) = nk$.

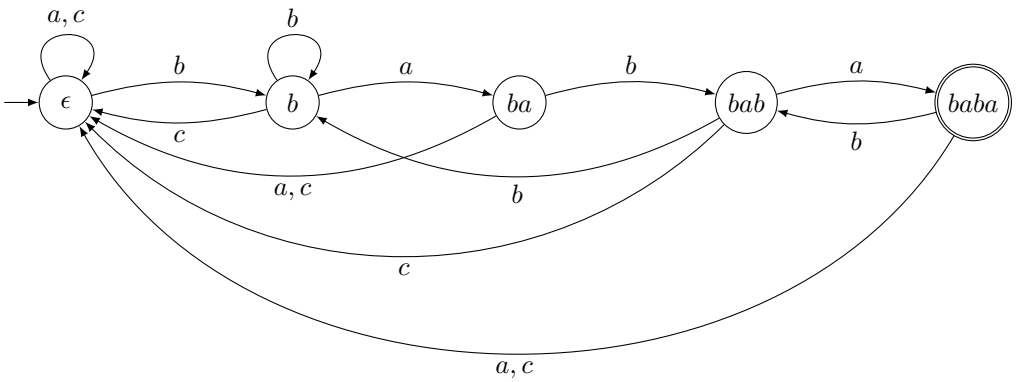
5. En vous appuyant sur vos réponses aux questions précédentes, expliquez pourquoi dans un groupe de 17 personnes il n'est pas possible que chaque personne ait exactement 5 amis au sein du groupe (pour simplifier, vous pouvez supposer que les liens d'amitié sont toujours réciproques).

   Solution : On peut modéliser la situation décrite par un graphe non dirigé où les nœuds représentent les personnes et les arêtes les liens d'amitié. On raisonne par l'absurde. Si les 17 personnes avaient chacune 5 amis, alors, d'après la réponse à la question 4, la somme des degrés des nœuds du graphe décrivant la situation serait de $17*5 = 85$. On aurait alors, d'après la réponse à la question 3, $85 = 2m$ avec m, le nombre d'arête du graphe, un nombre entier. Comme 85 est impair on obtient une contradiction. Il n'est donc pas possible que dans un groupe de 17 personnes chaque personne ait exactement 5 amis au sein du groupe.

Exercice 2

Donnez un automate ayant pour alphabet `{a, b, c}` et reconnaissant toutes les séquences de caractères dans cet alphabet qui finissent par `baba`.

Solution :



Exercice 3 Voici la table de transitions vue en cours pour une machine de Turing (machine `T1`) permettant d'incrémenter un entier naturel.

état	symbole	sens	nouveau symbole	nouvel état
D	0	$\rightarrow$	0	D
D	1	$\rightarrow$	1	D
D		$\leftarrow$		G
G	1	$\leftarrow$	0	G
G	0	$\leftarrow$	1	OK
G		$\leftarrow$	1	OK

L'entier à incrémenter est écrit sur la bande de la machine en notation binaire. Cette machine a D comme état initial et s'arrête si elle atteint l'état OK. Comme d'habitude, la tête de lecture est initialement placée sur la case contenant le premier symbole du mot d'entrée (ici le chiffre le plus à gauche dans la notation binaire du nombre à incrémenter).

1. Exécutez la machine de Turing `T1` sur l'entrée `101` en montrant à chaque étape le contenu du ruban, la position de la tête de lecture et l'état de la machine.

Solution :

	D			
|	1 |	0 |	1 |	
		D		
|	1 |	0 |	1 |	
			D	
|	1 |	0 |	1 |	
				D
|	1 |	0 |	1 |	
			G	
|	1 |	0 |	1 |	
		G		
|	1 |	0 |	0 |	
	OK			
|	1 |	1 |	0 |	

2. En vous inspirant de la machine `T1`, proposez une machine de Turing `T2` permettant de décrémenter un entier naturel strictement positif écrit sur la bande en notation binaire.

Solution :

état	symbole	sens	nouveau symbole	nouvel état
$D\_$	0	$\rightarrow$	0	$D\_$
$D\_$	1	$\rightarrow$	1	$D\_$
$D\_$		$\leftarrow$		$G\_$
$G\_$	0	$\leftarrow$	1	$G\_$
$G\_$	1	$\rightarrow$	0	OK

L'état initial est $D\_$.

3. Voici une table de transitions pour une machine de Turing `T3` acceptant les entiers écrits en binaire différents de 0. L'état initial est Z.

état	symbole	sens	nouveau symbole	nouvel état
Z	0	$\rightarrow$	0	Z
Z	1	$\rightarrow$	1	OK

Et voici une table de transitions pour une machine de Turing `T4` décalant un nombre écrit en binaire d'une case vers la gauche. L'état initial est DM.

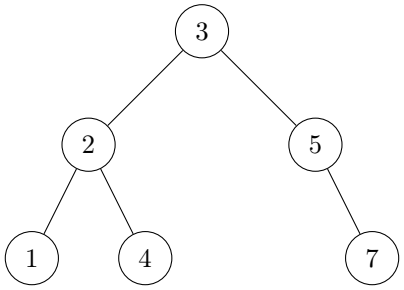
état	symbole	sens	nouveau symbole	nouvel état
DM	0	$\rightarrow$	0	DM
DM	1	$\rightarrow$	1	DM
DM		$\leftarrow$		M
M	0	$\leftarrow$		$M0$
M	1	$\leftarrow$		$M1$
$M0$	0	$\leftarrow$	0	$M0$
$M0$	1	$\leftarrow$	0	$M1$
$M0$		$\rightarrow$	0	OK
$M1$	0	$\leftarrow$	1	$M0$
$M1$	1	$\leftarrow$	1	$M1$
$M1$		$\rightarrow$	1	OK

Décrivez comment vous pourriez vous inspirer de machines de Turing `T1`, `T2`, `T3` et `T4` pour construire une machine de Turing permettant de calculer la somme de deux entiers naturels écrit sur la bande en notation binaire et séparés par un symbole `+` (l'alphabet de la machine de Turing serait donc `{0,1,+}`). Par exemple, pour le calcul de $3 + 2$, la bande contiendrait initialement `11+10` et le résultat final serait `101`. On vous demande seulement de donner l'idée générale, pas besoin de donner une table de transition explicite pour cette question.

Solution : On peut utiliser la stratégie suivante : tester si l'entier le plus à gauche est égal à zéro, si c'est le cas effacer le `+` et terminer, sinon décrémenter l'entier le plus à gauche, incrémenter l'entier le plus à droite, en décalant l'entier à gauche d'une case vers la gauche dans le cas où l'incrémentation de l'entier à droite augmente le nombre de chiffres de son écriture binaire, et revenir au test initial.
Pour implémenter cette stratégie, on peut réutiliser l'essentiel des transitions des machines `T1`, `T2`, `T3` et `T4`, en adaptant un peu pour traiter correctement la présence du `+` et s'assurer qu'on traite le bon entier à chaque fois (celui de gauche ou celui de droite selon les cas) et en remplaçant les états finaux des machines par les états initiaux appropriés pour enchaîner les opérations dans l'ordre désiré.
Ce n'était pas demandé, mais voici, pour le lecteur curieux, une table de transitions possible pour la machine désirée. L'état initial est Z. On a introduit un état en plus de ceux des machines `T1`, `T2`, `T3` et `T4` : l'état GG qui sert à parcourir la bande jusqu'à atteindre l'extrémité gauche dans le cas où l'incrémentation du nombre de droite ne change pas le nombre de chiffres de son écriture binaire (les états G et G_- ne permettant pas de le faire).

état	symbole	sens	nouveau symbole	nouvel état
Z	0	$\rightarrow$		Z
Z	+	$\rightarrow$		OK
Z	1	$\rightarrow$	1	D_-
D_-	0	$\rightarrow$	0	D_-
D_-	1	$\rightarrow$	1	D_-
D_-	+	$\leftarrow$	+	G_-
G_-	0	$\leftarrow$	1	G_-
G_-	1	$\rightarrow$	0	D
D	0	$\rightarrow$	0	D
D	1	$\rightarrow$	1	D
D	+	$\rightarrow$	+	D
D		$\leftarrow$		G
G	1	$\leftarrow$	0	G
G	0	$\leftarrow$	1	GG
G	+	$\leftarrow$	1	M
GG	0	$\leftarrow$	0	GG
GG	1	$\leftarrow$	1	GG
GG	+	$\leftarrow$	+	GG
GG		$\rightarrow$		Z
M	0	$\leftarrow$	+	$M0$
M	1	$\leftarrow$	+	$M1$
$M0$	0	$\leftarrow$	0	$M0$
$M0$	1	$\leftarrow$	0	$M1$
$M0$		$\rightarrow$		Z
$M1$	0	$\leftarrow$	1	$M0$
$M1$	1	$\leftarrow$	1	$M1$
$M1$		$\rightarrow$	1	D_-

Exercice 4 Un parcours alternatif pour les arbres (binaires) est le *parcours par niveaux*, dans lequel les nœuds sont traités (par exemple affichés) niveau par niveau, de la racine vers les feuilles, et de gauche à droite. Par exemple, considérons l'arbre suivant :



Dans un parcours par niveaux, ses nœuds sont affichés dans l'ordre : 3, 2, 5, 1, 4, 7.

1. En supposant de disposer d'un algorithme `afficher_niveau(arbre, i)`, qui affiche le niveau i de l'arbre (où 0 est le niveau de la racine, 1 le niveau des enfants de la racine, etc.) et d'un algorithme `hauteur(arbre)` pour calculer la hauteur, on peut écrire le parcours par niveaux comme suit :

```
1 def parcours_par_niveaux(arbre):
2     h = hauteur(arbre)
3     for i in range(h + 1):
4         afficher_niveau(arbre, i)
```

   Écrivez l'algorithme `afficher_niveau(arbre, i)` de façon récursive. Votre algorithme devra manipuler les arbres binaires uniquement par le biais des fonctions suivantes (vues en cours) : `nouveau_nœud(valeur, enfant_gauche, enfant_droit)`, `valeur(nœud)`, `enfant_gauche(nœud)`, `enfant_droit(nœud)` et `est_arbre_vide(nœud)`.

   Solution : On descend dans l'arbre jusqu'à arriver au niveau souhaité, en s'arrêtant sur les sous-arbres vides :

```
def afficher_niveau(arbre, i):
    if not est_arbre_vide(arbre):
        if i == 0:
            print(valeur(arbre))
        else:
            afficher_niveau(enfant_gauche(arbre), i - 1)
            afficher_niveau(enfant_droit(arbre), i - 1)
```

2. L'algorithme `parcours_par_niveaux(arbre)` ainsi obtenu a le défaut que chaque nœud (en dehors des feuilles) est traversé plusieurs fois, puisque pour accéder aux nœuds d'un certain niveau il est toujours nécessaire de passer par les nœuds des niveaux précédents. Combien de fois traverse-t-on la racine dans le pire des cas pour un arbre de n nœuds ? Combien de fois traverse-t-on un nœud de niveau i dans le pire des cas ? En déduire (en justifiant votre réponse) que le nombre total de traversées de nœuds effectuées par `parcours_par_niveaux` est en $O(n^2)$ dans le pire des cas.

   Solution : Dans le pire des cas l'arbre est une chaîne de n nœuds de hauteur $n-1$ (donc avec n niveaux). Par conséquence, on traverse la racine n fois, une fois par niveau. Un nœud générique du niveau i est traversé $n-i$ fois dans le pire des cas. En total cela donne $\sum_{i=0}^{n}(n-i) = \sum_{j=1}^{n} j = \frac{1}{n}n(n+1)$ traversées de nœuds, ce qui est $O(n^2)$.

3. Il est possible d'écrire un algorithme plus efficace pour le parcours par niveaux, qui traverse chaque nœud une seule fois, en utilisant une file comme structure auxiliaire (on manipulera les files uniquement par le bias des algorithmes de base vus en cours `nouvelle_file()`, `enfiler(F, x)`, `défiler(F)` et `est_file_vide(F)`). L'idée est d'insérer les enfants du nœud courant dans la file après l'avoir affiché. Écrivez cet algorithme.

Solution : Cet algorithme est essentiellement un parcours en largeur à partir de la racine :

```
def parcours_par_niveaux(arbre):
    F = nouvelle_file()
    enfiler(F, arbre)
    while not est_file_vide(F):
        nœud = défiler(F)
        if not est_vide(nœud):
            print(valeur(nœud))
            enfiler(F, enfant_gauche(nœud))
            enfiler(F, enfant_droit(nœud))
```

Si on veut éviter d'insérer des arbres vides dans la file, il est nécessaire de faire des tests supplémentaires :

```
def parcours_par_niveaux(arbre):
    if not est_arbre_vide(arbre):
        F = nouvelle_file()
        enfiler(F, arbre)
        while not est_file_vide(F):
            nœud = défiler(F)
            print(valeur(nœud))
            if not est_arbre_vide(enfant_gauche(nœud)):
                enfiler(F, enfant_gauche(nœud))
            if not est_arbre_vide(enfant_droit(nœud)):
                enfiler(F, enfant_droit(nœud))
```