Université d'Aix-Marseille - Master Informatique 1^{ere} année
UE Complexité - Correction Examen partiel - 2020-2021
Durée : 1h30 - Tous documents interdits

Préambule : Pour les questions où vous devez fournir des algorithmes, ceux-ci devront être écrits dans un langage algorithmique avec des instructions de genre "`si alors sinon`", "`tantque`", "`pour`" etc. mais il faudra les définir avec un minimum d'ambiguïté de sorte à ce que le correcteur puisse s'assurer de leur validité et que l'évaluation de leur complexité soit la plus précise possible (ces évaluations devront toujours être justifiées). Concernant celles-ci, sauf indication contraire, nous utiliserons le modèle de base pour lequel toutes les actions élémentaires, commme les opérations du type tests, affectations ou opérations arithmétiques, sont exécutables en temps constant.

Exercice 1. On considère ici des tableaux de n caractères, qui ne peuvent contenir que les caractères "a", "b", "c", "d" ou "B", sachant que les caractères "a", "b", "c" ou "d" ne peuvent apparaître que dans le début du tableau, et dès que le caractère "B" apparaît, toutes les cases suivantes contiennent le caractère "B" jusqu'à la fin du tableau. Il vous faut donner ici un algorithme qui prend en entrée un tel tableau et qui vérifie si ce tableau contient la séquence de caractères "abc", c'est-à-dire s'il existe un indice `i` tel que `t[i]="a"`, `t[i+1]="b"` et `t[i+2]="c"`. Donnez une évaluation de la complexité de l'algorithme.

Solution. (Algorithme sur 1,25 points) Il existe plusieurs solutions mais nous veillons dans ce corrigé à proposer un algorithme qui soit le plus simple possible. Le résultat est mémorisé dans la variable `reponse` qui vaut `Oui` en sortie si la séquence "abc" est présente dans le tableau, et `Non` sinon.

```
entree : t tableau de caracteres indice de 1 a n
sortie : reponse {Oui,Non}
debut
        si  ( n < 3 ) alors  reponse <- Non
        sinon
                reponse <- Non
                i <- 1
                tant que  ( reponse = Non ) et ( i <= n-2 ) et ( t[i] != 'B' ) faire
                    si ( t[i] = 'a' ) et ( t[i+1] = 'b' ) et ( t[i+2] = 'c' ) alors reponse <- Oui
                                                                                sinon i <- i+1
                fin tant que
        fin sinon
fin
```

Concernant la validité de l'algorithme, on note qu'il y a 3 possiblités d'arrêt de la boucle :

- (reponse = Oui) : ce qui signifie que t[i] = 'a', t[i+1] = 'b' et t[i+2] = 'c'
- (i $\leq$ n-2) est faux, i.e. (i > n-2), ce qui signifie que i=n-1 qu'il n'y a donc plus que 2 cases du tableau pour trouver l'occurrence de la séquence "abc" à partir de t[i], i.e. à partir de t[n-1].
- (t[i] = 'B') ce qui signifie qu'il n'y a plus de possibilité pour trouver l'occurrence de la séquence "abc" à partir de t[i] car le reste du tableau à partir de t[i] ne contient que le caractère 'B'.

Complexité (sur 0,75 points). Dans le pire des cas, tout le tableau peut être visité, soit que "abc" ne figure pas dans la tableau, soit qu'il y figure dans les 3 dernières cases t[n-2], t[n-1] et t[n]. Dans ces conditions, si la séquence "abc" n'est pas présente et qu'aucun 'B' n'a été trouvé, la dernière valeur de i sera égale à n-1. Ainsi, il y aura exactement n-2 passages dans la boucle `tantque` (pour i=1, i=2, ...et i=n-2). Et à chaque passage, il y a les 3 tests d'entrée, le 3 tests du `si`, et l'incrémentation de la variable i. On dénombre ainsi 8 opérations fondamentales (l'incrémentation contient 2 opérations fondamentales). On pourrait éventuellement rajouter les 3 opérations fondamentales (3 opérations arithmétiques) figurant dans les tests (i $\leq$ n-2), (t[i+1] = 'b') et (t[i+2] = 'c'). La boucle `tantque` conduit donc à :

- 8 $\times$ (n-2) opérations fondamentales
- 3 opérations fondamentales quand i=n-1(on peut supposer que les trois tests d'entrée sont exécutés lors de l'arrêt de la boucle).

En récapitulant, il faut compter le test du premier `si` (i.e. n < 3), les initialisations des variables `reponse` et `i`, et on obtient au total 3 + (8 $\times$ (n-2)) + 3 = 8n - 10 opérations fondamentales. Notons que la prise en compte des opérations fondamentales figurant dans les tests (i $\leq$ n-2), (t[i+1] = 'b') et (t[i+2] = 'c') ne modifie pas la complexité de l'algorithme. Celle-ci est donc en $\Theta(8n-10) = \Theta(n)$. Si l'usage ne la notation O ne serait pas erronée, celle-ci serait tout simplement imprécise et il est donc ici impératif d'utiliser Θ.

Exercice 2. Définissez un programme pour Machine de Turing Déterministe qui reconnaît le langage L défini sur l'alphabet $\{a,b,c,d\}$ et constitué des mots qui contiennent comme facteur le mot "abc". On rappelle qu'un mot v est facteur d'un mot m s'il existe deux mots u et w, facteurs de m et tels que $m = uvw$. On rappelle que pour définir un programme pour Machine de Turing Déterministe, il faut notamment préciser ses états, son alphabet d'entrée ainsi que son alphabet de ruban, et la fonction de transition (vous pouvez fournir celle-ci par un dessin comme dans le corrigé du TD 3). Donnez une évaluation de la complexité de votre programme. À quelle classe de complexité appartient ce langage L? Justifiez votre réponse.

Solution. Programme (sur 2 points). Ce programme pour machine de Turing déterministe est défini sur l'alphabet d'entrée $\Sigma = \{a,b,c,d\}$ et l'alphabet de ruban $\Gamma = \{a,b,c,d,\beta\}$ où β correspond au (symbole) blanc d'une case vide. Elle fonctionne en déplaçant systématiquement la tête de lecture/écriture de la gauche vers la droite d'une case à chaque transition. La machine est définie avec les états suivants:

- q_{oui} et q_{non}
- l'état initial q_0 qui sert aussi à trouver un symbole a
- l'état q_1 qui signifie que le dernier symbole lu est un a et qui vérifie si le symbole suivant est un b
- l'état q_2 qui signifie que les deux derniers symboles lus sont un a suivi d'un b et vérifie si le symbole suivant est un c

Dans l'état q_0, dès que le symbole courant est un a, la machine passe dans l'état q_1, et si c'est β elle s'arrête en transitant en q_{non}. Si dans q_1, le symbole courant est un a, la machine reste dans cet état, si le symbole courant est un b, elle passe dans l'état q_2, sinon, si c'est c ou d, elle revient dans l'état q_0, et si c'est β elle s'arrête en transitant en q_{non}. Si dans q_2, le symbole courant est un c, elle s'arrête en transitant dans l'état q_{oui}, sinon, si c'est un a, la machine repasse dans létat q_1, si le symbole courant est un b ou un d, elle revient dans l'état q_0, et si c'est β elle s'arrête en transitant en q_{non}.

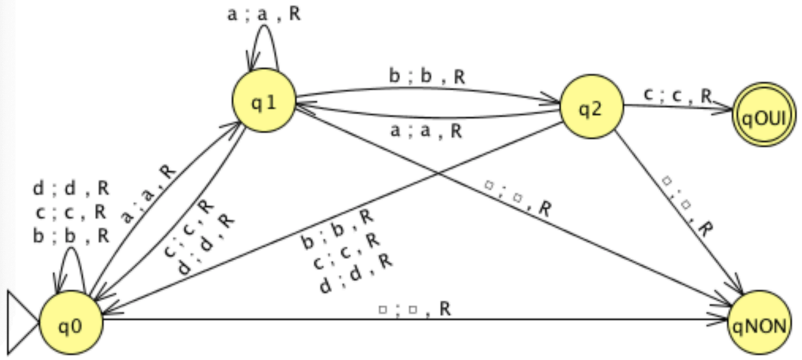

FIG. 1 – *Machine de Turing dessinée avec JFLAP (R pour déplacement à droite et □ pour le symbole β).*

Complexité (sur 1 point). Dans le pire des cas, le mot m en entrée ne figure pas dans le langage L, et le mot est lu intégralement et donc, comme à chaque transition, la tête de lecture/écriture se déplace d'une case vers la droite, il y aura exactement $n+1$ transitions, avec $n = |m|$ (n transitions pour arriver sur la première case vide, et une dernière transition pour passer dans q_{non}). Si le mot en entrée appartient au langage L, dans le pire des cas, le facteur *abc* figure à la fin du mot m et donc, on a exactement n transitions. La complexité est donc en $\Theta(|m|) = \Theta(n)$.

Classe de complexité du langage L (sur 1 point). Du fait de l'existence du programme pour machine de Turing déterministe précédent, et de sa complexié en $\Theta(n)$, donc polynomiale sur modèle de calcul déterministe, on peut affirmer que ce langage appartient à la classe de complexité **P**.

Exercice 3. Un *ensemble dominant* dans un graphe non-orienté $G = (S,A)$, est un sous-ensemble D de S, i.e. $D \subseteq S$ tel que pour tout sommet x de S qui n'est pas D, i.e. $x \in S \backslash D$, il existe au moins un sommet $y \in D$ tel que l'arête $\{x,y\} \in A$. En d'autres termes, tout sommet x de S qui n'est pas D doit avoir au moins un sommet voisin y qui est dans D. À partir de cette notion d'ensemble dominant, il est possible de définir le problème de décision suivant :

ENSEMBLE DOMINANT
Donnée : Un graphe non-orienté $G = (S,A)$ et un entier k.
Question : G possède-t-il un ensemble dominant de taille k ou moins ?

Question 1. Dessinez un graphe non-orienté G de 6 sommets et 6 arêtes tel que le couple (G,3) (i.e. $k = 3$) est une instance positive de *ENSEMBLE DOMINANT* et le couple (G,2) (i.e. $k = 2$) est une instance négative de *ENSEMBLE DOMINANT*. Dans chaque cas, il vous faut justifier votre réponse.

Solution (sur 1 point avec 0,5 point par instance). Comme précisé dans la question, une instance du problème de décision *ENSEMBLE DOMINANT* est un couple constitué d'un graphe non-orienté G et d'un entier k. Nous proposons donc un unique graphe G, mais en considérant un couple (G,3) (i.e. $k = 3$) et un couple (G,2) (i.e. $k = 2$), nous allons en fait proposer avec un même graphe deux instances. Voici donc le graphe G :

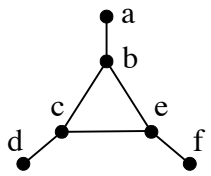

FIG. 2 – *Graphe G de 6 sommets et 6 arêtes permettant de définir une instance positive et une instance négative.*

Pour la première instance avec $k = 3$, donc ce graphe G et l'entier 3, on constate que l'ensemble de sommets $D = \{b,c,e\}$ est ensemble dominant de taille $k = 3$ puisque soit les sommets sont dan D, soit ils possèdent un voisin qui est dans D. Par conséquent, (G,3) est bien une instance positive du problème de décision *ENSEMBLE DOMINANT*. Pour la deuxième instance avec $k = 2$, on peut constater qu'aucun ensemble de 2 sommets n'est un ensemble dominant de G. En effet, il faut nécessairement dans cet ensemble avoir à la fois soit a ou b, soit c ou d, soit e ou f, ce qui impose de disposer de 3 sommets dans D. Il n'existe donc aucun ensemble dominant de taille $k = 2$ pour ce graphe G et par conséquent, (G,2) est bien une instance négative du problème de décision *ENSEMBLE DOMINANT*.

Question 2. À partir du problème de décision *ENSEMBLE DOMINANT*, définissez le problème de recherche associé, puis celui d'optimisation (ici on considère bien sûr comme critère d'optimisation la taille de l'ensemble dominant), celui de comptage et enfin, celui d'énumération.

Solution. (Sur 2 points, soit 0,5 point par définition de problème)

ENSEMBLE DOMINANT RECHERCHE
Donnée : Un graphe non-orienté $G = (S,A)$ et un entier k.
Question : Si G possède un ensemble dominant de taille k ou moins, le trouver.

ENSEMBLE DOMINANT OPTIMISATION
Donnée : Un graphe non-orienté $G = (S,A)$.
Question : Trouver un ensemble dominant de G de taille minimum.

ENSEMBLE DOMINANT COMPTAGE
Donnée : Un graphe non-orienté $G = (S,A)$ et un entier k.
Question : Donner le nombre d'ensembles dominants de G de taille k.

ENSEMBLE DOMINANT ÉNUMÉRATION
Donnée : Un graphe non-orienté $G = (S,A)$ et un entier k.
Question : Donner tous les ensembles dominants de G de taille k.

Notons que pour les 2 derniers problèmes, *ENSEMBLE DOMINANT COMPTAGE* et *ENSEMBLE DOMINANT ÉNUMÉRATION* on peut aussi considérer tous les ensembles dominants de G de taille k ou moins.

Question 3. Donnez un algorithme qui prend en entrées un graphe non-orienté G et un ensemble de sommets D représenté par un tableau de booléens, et qui vérifie si D est un ensemble dominant de G. Donnez une évaluation de la complexité de votre algorithme. Nous vous recommandons d'utiliser à partir de cette question, une représentation des graphes par matrices d'adjacence même si cela engendre une (petite) dégradation de l'efficacité de vos algorithmes.

Solution. (Algorithme sur 2 points) Tout d'abord, il faut rappeler qu'un ensemble de sommets D représenté par un tableau de booléens est un tableau indicé de 1 à n, tel que si D[x] = 1, alors le sommet x est dans l'ensemble D et si D[x] = 0, alors x n'appartient pas à l'ensemble. Il s'agit ici de vérifier que tout sommet qui n'est pas dans D possède au moins un voisin qui appartient à D. Ainsi, tous les sommets doivent être visités. Pour un sommet x qui n'est pas

dans D, il faut visiter ses voisins pour vérifier qu'il possède au moins un voisin appartenant à D, c'est-à-dire qu'il existe un sommet y dans D (i.e. avec D[y] = 1) tel que l'arc $\{x,y\}$ figure dans le graphe (i.e. tel que G.A[x][y] =1 puisque le graphe est représenté par une matrice d'adjacence). L'algorithme va donc parcourir les n sommets de G et pour chaque sommet x qui n'est pas dans D, rechercher dans la matrice G.A, à la ligne d'indice x, si une case G.A[x][y] a pour valeur 1 avec D[y] = 1. Si pour un sommet x, ce n'est pas le cas, alors la réponse doit être négative.

```
algorithme Ensemble-dominant
entree G : graphe de n sommets represente par une matrice d'adjacence
entree D : tableau de booleens indice de 1 a G.n
sortie : reponse (Oui,Non)
variables x,y : sommet
variable trouve : booleen
debut
        reponse <- Oui
        x <- 1
        tant que ( reponse = Oui ) et ( x <= G.n ) faire
                si ( D[x] = 0 ) alors
                        y <- 1
                        trouve <- faux
                        tant que ( trouve = faux ) et ( y <= G.n ) faire
                                si ( G.A[x][y] = 1 ) et ( D[y] = 1 ) alors trouve <- vrai
                                                                     sinon y <- y+1
                        si ( trouve = faux ) alors reponse <- Non
                fin si
        fin tant que
fin
```

Complexité (sur 1 point). Dans le pire des cas, la boucle sur x va être exécutée n fois, et celle sur y aussi. Les autres opération s'exécutent en temps constant. Ainsi, la complexité est en $\Theta(n^2)$.

Question 4. Donnez un algorithme qui prend en entrées un graphe non-orienté G et un ensemble de sommets D, et qui vérifie si D est un ensemble dominant minimal de G, c'est-à-dire que D ne possède aucun sous-ensemble strict qui soit aussi un ensemble dominant de G. Donnez une évaluation de la complexité de votre algorithme.

Solution. (Algorithme sur 2 points) En fait, tout d'abord, il faut vérifier que l'ensemble de sommets D est bien un ensemble dominant de G. Une fois ceci réalisé, il suffit de s'assurer qu'aucun sous-ensemble strict de D n'est un ensemble dominant de G. Pour cela, on regarde si en enlevant un sommet x de D, on dispose d'un ensemble dominant de G. Et il suffit d'en trouver un pour constater que D n'est pas un ensemble dominant minimal de G. Pour faciliter le traitement, on va utiliser l'algorithme `Ensemble-dominant` proposé dans la réponse à la question précédente.

```
entree G : graphe de n sommets represente par une matrice d'adjance
entree D : tableau de booleens indice de 1 a n
sortie reponse : {Oui,Non}
variable x : sommet
variable minimal : booleen
debut
        si ( Ensemble-dominant(G,D) = Oui ) alors
                minimal <- vrai
                x <- 1
                tant que ( minimal ) et ( x <= G.n ) faire
                        si ( D[x] = 1 ) alors
                                D[x] <- 0
                                si ( Ensemble-dominant(G,D) = Oui ) alors minimal <- faux
                                                                    sinon x <- x+1 ; D[x] <- 1
                        fin si
                fin tant que
                si minimal alors reponse <- Oui sinon reponse <- Non
        sinon reponse <- Non
fin
```

Complexité (sur 1 point). D'entrée, le test `( Ensemble-dominant(G,D) = Oui )` est en $\Theta(n^2)$. Si celui-ci renvoie Oui, le traitement se poursuit avec l'exécution du "`alors`". Dans le pire des cas, la boucle sur x va être exécutée n fois, et les tests (D[x] = 1) peuvent être au pire de l'ordre de n fois positifs. Dans ce cas, le test suivant `( Ensemble-dominant(G,D) = Oui )` va donc être exécuté et comme son coût est en $\Theta(n^2)$, cela conduit à un coût global en $\Theta(n^3)$.

Question 5. Donnez un algorithme qui prend en entrée un graphe non-orienté G et qui fournit en résultat un ensemble dominant de G. Donnez une évaluation de la complexité de votre algorithme.

Solution. (Algorithme sur 0,5 point) En fait, on sait que tout ensemble contenant tous les sommets est un ensemble dominant de G. Il suffit donc tout simplement de construire un ensemble D contenant tous les sommets.

```
entree G : graphe de n sommets represente par une matrice d'adjance
sortie D : tableau de booleens indice de 1 a n
variable x : sommet
debut
        pour x=1 a G.n faire D[x] <- 1
fin
```

Complexité (sur 0,5 point). Tout simplement en $\Theta(n)$ car il y a de l'ordre de n exécutions d'instructions réalisables chacune en temps constant.

Question 6. Donnez un schéma d'algorithme (il n'est pas nécessaire de donner votre algorithme dans le détail) qui prend en entrées un graphe non-orienté G et un entier k, et teste si le graphe G possède un ensemble dominant de taille k ou moins. Donnez une évaluation de la complexité de votre algorithme.

Solution. (Schéma sur 2 points + 1 point pour la complexité) En fait, une solution simple consiste à tester tous les sous-ensembles de sommets de taille k, sachant que pour chacun, pour vérifier s'il s'agit d'un ensemble dominant de G, on peut utiliser l'algorithme proposé dans la réponse à la question 3 dont le coût est en $\Theta(n^2)$. Il suffit pour cela de développer une arborescence de recherche (déterministe) qui va les construire un à un si nécessaire (si on en trouve 1, le traitement peut cependant s'arrêter). La difficulté ici étant que leur nombre est égal a $C_n^k = \frac{n!}{k!(n-k)!}$. On a vu dans le TD 2 (Exercice 2, question 4), que si $k = n/2$, alors $C_n^k = 2^{n/2}$. Il s'agit donc d'explorer un espace de recherche qui est de taille exponentielle par rapport à la taille n^2 de la donnée en entrée. Concernant cette exploration on peut envisager le développement (récursif) d'une arborescence binaire qui à chaque niveau va affecter de 2 façons un tableau D. À la racine de l'arborescence, on va considérer D[1], qui sera affecté sur la branche gauche à 0, et sur la branche droite à 1. Et à un niveau i, on procèdera de même mais bien sûr pour D[i]. L'ensemble des 2^n feuilles de cette arborescence binaire correspond aux 2^n sous-ensembles possibles des sommets du graphe. Et pour chacune des feuilles, on examine d'abord la taille du sous-ensemble associé au tableau de boolén D pour s'assurer que sa taille vaut k, et quand c'est le cas, il suffit alors d'utiliser l'algorithme `Ensemble-dominant` proposé dans la réponse à la question 3 pour vérifier s'il représente bien un ensemble dominant. On arrive alors à un algorithme dont la complexité est en $\Theta(n^2.2^n)$, où n^2 est le coût de l'appel à l'algorithme `Ensemble-dominant`, soit une complexité exponentielle.

Question 7. Pouvez-vous affirmer que le problème de décision *ENSEMBLE DOMINANT* appartient à la classe de complexité **P** ? Justifiez votre réponse.

Solution. (Sur 1 point) En fait, le schéma d'algorithme proposé dans la question précédente correspond à un algorithme qui résout ce problème et dont la complexité est exponentielle. Cela étant, nous ne pouvons prouver que c'est le meilleur algorithme pour résoudre le problème de décision *ENSEMBLE DOMINANT*. Aussi, nous ne pouvons affirmer avec ce qui précède si le problème de décision *ENSEMBLE DOMINANT* appartient à la classe de complexité **P**, ni même son contraire.