Examen 2I003

Jeudi 2 Juin 2016, 2 heures
aucun document autorisé

Exercice 1 – Graphes non orientés

Dans cet exercice, $G = (V, E)$ est un graphe non orienté. On pose $n = |V|$ et $m = |E|$.

Question 1

Rappelez la définition du degré $d_G(x)$ pour $x \in V$.

Solution:

Le degré de tout sommet $x \in V$ est le nombre de sommets de V qui sont adjacents à x.

Question 2

Démontrez par récurrence sur le nombre m d'arêtes que, pour tout graphe non orienté $G = (V, E)$:

$$\sum_{x \in V} d_G(x) = 2m.$$

Solution:

La propriété à démontrer s'énonce ainsi, pour $m \geq 0$:

$$\Pi(m) : \text{pour tout graphe } G = (V, E) \text{ non orienté de } m \text{ arêtes, } \sum_{x \in V} d_G(x) = 2m.$$

La propriété se démontre par récurrence faible.

Base $m = 0$. Pour tout graphe G sans arête, $d_G(x) = 0$ donc $\Pi(0)$ est bien vérifiée.

Induction Soit $m > 0$ tel que $\Pi(m-1)$ soit vérifiée. Soit $G = (V, E)$ un graphe ayant m arêtes, $a = \{u, v\} \in E$, $E' = E \setminus \{a\}$ et $G' = (V, E')$. G' est un graphe qui a $m - 1$ arêtes donc, par hypothèse de récurrence :

$$\sum_{x \in V} d_{G'}(x) = 2(m-1)$$

D'autre part, on observe que $d_G(u) = d_{G'}(u) + 1$, $d_G(v) = d_{G'}(v) + 1$ et que $d_G(x) = d_{G'}(x)$ pour tout $x \in V \setminus \{u, v\}$. On en déduit que

$$\sum_{x \in V} d_G(x) = 2 + \sum_{x \in V} d_{G'}(x) = 2 + 2(m-1) = 2m$$

Donc $\Pi(m)$ est vérifiée.

Conclusion On en conclut que la propriété est vraie pour tout $m \in \mathbb{N}$.

Question 3

Rappelez la définition de la composante connexe C_x de x pour $x \in V$.

Solution:

La composante connexe de $x \in V$ est l'ensemble des sommets $y \in V$ tels qu'il existe une chaîne entre x et y.

Question 4

Dans cette question on suppose que tous les sommets de G sont de degré 1.

1. Démontrez que G a un nombre pair de sommets.
2. Démontrez par l'absurde que $|C_x| = 2$ pour $x \in V$. Calculez le nombre de composantes connexes en fonction de $n = |V|$.

Solution:

1. D'après la question précédente : $\sum_{x\in V} d_G(x) = 2m$. Puisque tous les sommets de G sont de degré 1, on a aussi $\sum_{x\in V} d_G(x) = n$. Donc $n = 2m$ et n est pair.
2. Puisque $d_G(x) = 1$, x a un sommet adjacent y donc C_x a au moins 2 éléments : x et y.

   Supposons qu'il existe un troisième élément z dans C_x. Alors il existe une chaîne $x_1 = x, x_2, \ldots, x_k = z$ entre x et z, dont les sommets sont deux à deux distincts. Puisque $d_G(x) = 1$, on a $x_2 = y$. Donc y a deux sommets adjacents : x et x_3, ce qui contredit le fait que $d_G(y) = 1$. Par conséquent il n'existe pas de troisième élément dans C_x et $|C_x| = 2$.

   Les composantes connexes forment une partition de V et sont toutes de cardinal 2, il y en a donc $\frac{n}{2}$.

Question 5

Dans cette question on suppose que G est connexe, que $n \geq 3$ et que tous les sommets de G sont de degré égal à 2. On veut montrer que l'on peut construire une chaîne élémentaire $x_1, x_2, \ldots, x_n$ telle que x_1 et x_n sont adjacents. Soit $x_1, x_2, \ldots, x_k$ une chaîne élémentaire de longueur maximale et x_{k+1} le deuxième sommet adjacent à x_k.

1. Montrez que x_{k+1} est l'un des sommets $x_1, x_2, \ldots, x_{k-2}$.
2. Montrez par l'absurde que $x_{k+1} = x_1$.
3. Montrez par l'absurde que $k = n$.
4. En déduire qu'il existe une chaîne élémentaire $x_1, x_2, \ldots, x_n$ telle que x_1 et x_n sont adjacents.

Solution:

1. x_{k+1} est le deuxième sommet adjacent à x_k donc il n'est pas égal à x_k ni à x_{k-1}. Si x_{k+1} n'est égal à aucun des sommets $x_1, x_2, \ldots, x_{k-2}$ alors $x_1, x_2, \ldots, x_{k-2}, x_{k-1}, x_k, x_{k+}$ est une chaîne élémentaire de longueur supérieure à celle de $x_1, x_2, \ldots, x_k$, ce qui contredit l'hypothèse. Donc x_{k+1} est l'un des sommets $x_1, x_2, \ldots, x_{k-2}$.
2. Supposons que $x_{k+1} \neq x_1$. Alors x_{k+1} est égal à l'un des x_i, avec $2 \leq i \leq k-2$ et ce sommet x_i a trois sommets adjacents différents : x_{i-1}, x_{i+1} et x_k, ce qui est contraire à $d_G(x_i) = 2$. Donc $x_{k+1} = x_1$.
3. Supposons que $k < n$. Alors il existe (au moins) un sommet $y \in V \setminus \{x_1, x_2, \ldots, x_k\}$. Puisque C est connexe il existe une chaîne entre x_1 et y. Cette chaîne contient nécessairement une arête $\{x_i, z\}$ avec $z \in V \setminus \{x_1, x_2, \ldots, x_k\}$, donc x_i a trois sommets adjacents différents, ce qui est contraire à $d_G(x_i) = 2$. Donc $k = n$.
4. Il existe une chaîne élémentaire $x_1, x_2, \ldots, x_k$ de longueur maximale (puisque la longueur d'une chaîne élémentaire est toujours inférieure ou égale à $n - 1$). On a montré dans les deux questions précédentes que $k = n$ et que x_1 est le deuxième sommet adjacent à x_n.

Exercice 2 – Gestion d'intervalles disjoints

Le but de cet exercice est d'étudier des structures de données pour stocker un ensemble d'intervalles fermés disjoints. Deux intervalles $I = [I.min, I.max]$ et $J = [J.min, J.max]$ sont disjoints si $I \cap J = \emptyset$. A titre d'exemple, $[5,7] \cap [7,10] = \{7\}$, ces deux intervalles ne sont donc pas disjoints. Par contre, $[5,7] \cap [8,10] = \emptyset$ et ces deux intervalles sont disjoints. Pour insérer un nouvel intervalle, il faut alors s'assurer qu'il n'intersecte pas un intervalle déja existant. Si c'est le cas, **on refuse l'insertion**.

Question 1

Ecrire la fonction `IntersectionVide(I,J)` qui retourne vrai si $I \cap J = \emptyset$. Quelle est sa complexité ?

Solution:

```python
def IntersectionVide(I,J):
        return (I.max<J.min) or (J.max<I.min)
```

La complexité de cette fonction est en $\Theta(1)$.

Question 2

On suppose dans cette question uniquement que l'on utilise un tableau non trié T pour stocker les intervalles. Quelle est la complexité dans le meilleur et le pire des cas d'un algorithme qui insère (si c'est possible) un intervalle I dans T ? Justifiez votre réponse.

Solution:

Dans le meilleur des cas, le premier élément du tableau intersecte l'intervalle que l'on souhaite insérer. La complexité dans le meilleur des cas est donc en $\Omega(1)$. Dans le pire des cas, aucun élément du tableau n'intersecte l'élément que l'on souhaite insérer. La complexité est alors en $\mathcal{O}(n)$.

Question 3

On suppose dans cette question uniquement que l'on utilise un tableau trié en ordre croissant T pour stocker les intervalles.

1. Quel est le tableau obtenu en effectuant les insertions successives des intervalles suivants : $[12, 15]$, $[5, 7]$, $[8, 9]$, $[27, 30]$, $[6, 10]$, $[20, 22]$ et $[2, 4]$?
2. Quelle est la complexité dans le pire des cas et le meilleur des cas de la fonction de recherche de la place éventuelle d'un intervalle I en utilisant une recherche séquentielle ? Justifiez votre réponse.
3. Quelle est la complexité dans le pire des cas et le meilleur des cas de la fonction de recherche de la place éventuelle d'un intervalle I en utilisant une recherche dichotomique ? Justifiez votre réponse.
4. Quelle est la complexité dans le pire des cas d'un algorithme qui insère (si c'est possible) un intervalle I dans T triée en utilisant la recherche dichotomique ? Justifiez votre réponse.

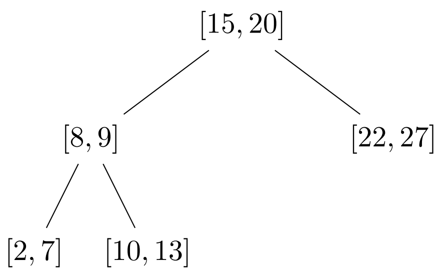
Solution:

1. On obtient le tableau $T = [[2, 4], [5, 7], [8, 9], [12, 15], [20, 22], [27, 30]]$;
2. Dans le meilleur des cas, le premier élément du tableau intersecte l'intervalle I que l'on souhaite insérer. La complexité dans le meilleur des cas est donc en $\Omega(1)$. Dans le pire des cas, I intersecte le dernier élément du tableau, la complexité est donc en $\mathcal{O}(n)$;
3. Dans le meilleur des cas, le premier élément du tableau testé (au milieu du tableau) intersecte l'intervalle I que l'on souhaite insérer. La complexité dans le meilleur des cas est donc en $\Omega(1)$. Dans le pire des cas, I intersecte le dernier élément du tableau testé, la complexité est donc en $\mathcal{O}(\log n)$;
4. Dans le pire des cas, l'éément est inséré en début du tableau, la complexité est donc en $\mathcal{O}(n)$.

Question 4

Par la suite on utilise les arbres d'intervalles qui constituent une extension des arbres binaires de recherche. Tout sommet T de l'arbre possède un champ `T.Int` correspondant à un intervalle dont les bornes sont notées `T.Int.min` et `T.Int.max`. La structure de l'arbre est alors un arbre binaire de recherche avec comme clefs les valeurs `T.Int.min`. Plus précisément :

1. Pour tout sommet T, pour tout élement Y du sous-arbre gauche de T, on a `Y.Int.min<T.Int.min`.
2. Pour tout sommet T, pour tout élement Y du sous-arbre droit de T, on a `Y.Int.min>T.Int.min`.
3. Tous les intervalles stockés dans l'arbre sont disjoints.



Vérifiez que l'arbre représenté est bien un arbre d'intervalles.

Solution:

On observe que, si on ne considère que les valeurs minimales de chaque intervalle, on obtient bien un arbre binaire de recherche. De plus, tous les intervalles sont disjoints. On en déduit que l'on a bien un arbre d'intervalles.

Question 5

On considère maintenant le programme suivant :

```
def ABRIcherche(I,T):
    if estABIvide(T):
        print "L␣intervalle␣pas␣dans␣l␣arbre"
        return False
    print "Etude␣de␣l␣intervalle␣[", T.Int.min, T.Int.max, "]"
    if not IntersectionVide(I,T.Int):
        print "L␣intervalle␣intersecte␣un␣element␣de␣l␣arbre"
        return True
    if I.max<T.Int.min:
        return ABRIcherche(I,T.gauche)
    return ABRIcherche(I,T.droit)
```

Exécutez deux fois la fonction pour l'arbre `T` de la question précédente et les intervalles `I1 =[11,12]` et `I2 =[14,14]`.

Que fait cette fonction dans le cas général (pour tout arbre T et tout intervalle I) ?

Solution:

```
>>> ABRIcherche(I1,T)
Etude de l'intervalle␣[␣15␣20␣]
Etude␣de␣l'intervalle [ 8 9 ]
Etude de l'intervalle␣[␣10␣13␣]
L'intervalle intersecte un element de l'arbre
True
>>>␣ABRIcherche(I2,T3)
Etude␣de␣l'intervalle [ 15 20 ]
Etude de l'intervalle␣[␣8␣9␣]
Etude␣de␣l'intervalle [ 10 13 ]
L'intervalle␣pas␣dans␣lÕarbre
False
```

La fonction retourne `True` si et seulement si I intersecte un élément de l'arbre d'intervalle T.

Question 6

Démontrez par induction sur la structure de l'arbre la terminaison et la validité de la fonction `ABRIcherche`. Que peut-on dire de l'intervalle I dans le dernier cas ?

Solution:

On montre que, pour tout arbre d'intervalles T et tout intervalle I, la fonction se termine. De plus, la fonction retourne vraie si et seulement si l'intervalle I intersecte un sommet de l'arbre.

Base : Si T est l'arbre vide, la fonction retourne `False`. On en déduit qu'elle se termine dans ce cas et qu'elle est valide.

Induction : Supposons maintenant que T soit non vide. Il y a alors 3 cas :

1. Si $T.Int \cap I \neq \emptyset$, la fonction se termine, et renvoie `True`. Elle est donc valide dans ce cas.
2. Par hypothèse d'induction, l'appel `ABRIcherche(I,T.gauche)` se termine et retourne `True` si et seulement si l'intervalle I intersecte un sommet du sous-arbre gauche de T. De plus, tous les éléments situés à gauche de T sont des intervalles inclus dans $[0, T.Int.min[$. Donc, si $I.max < T.Int.min$, `ABRIcherche(I,T)` est vrai si et seulement si `ABRIcherche(I,T)` l'est, et l'appel se termine.
3. De la même manière, tous les éléments situés à droite de T sont situés dans l'intervalle $]T.Int.min, +\infty[$. Dans le dernier cas, on a forcément que $I.min > T.Int.max$, `ABRIcherche(I,T)` est vrai si et seulement si `ABRIcherche(I,T)` l'est, et l'appel se termine.

Dans le dernier cas, on a forcément que $I.min > T.Int.max$.

Question 7

Quelle est la complexité dans le meilleur et le pire des cas de la fonction `ABRIcherche(I,T)` ?

Solution:

Dans le meilleur des cas I intersecte la racine, la complexité est donc en $\Omega(1)$. Dans le pire des cas, on parcourt le plus long chemin de la racine à une feuille. Chaque appel étant en $\Theta(1)$, la complexité globale est alors en $\mathcal{O}(h)$, où h est la hauteur de l'arbre.

Question 8

Construire l'arbre obtenu en insérant successivement si possible les intervalles suivants de l'arbre d'intervalle de la question 4 : $[14, 14]$, $[30, 35]$, $[21, 23]$ et $[28, 29]$.

Solution:

L'intervalle $[21, 23]$ ne peut pas être inséré.