

---

**NUMERO d'ANONYMAT :**

**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  $x \in V$ .

**Question 2**

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

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

---

**Question 3**

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

**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  $n = |V|$ .

---

### 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  $x_1, x_2, \dots, x_n$  telle que  $x_1$  et  $x_n$  sont adjacents.

Soit  $x_1, x_2, \dots, x_k$  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  $x_1, x_2, \dots, x_{k-2}$ .
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  $x_1, x_2, \dots, x_n$  telle que  $x_1$  et  $x_n$  sont adjacents.

---

## 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 déjà 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 ?

### 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.

---

### Question 3

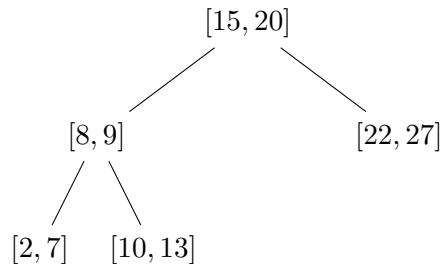
On suppose dans cette question uniquement que l'on utilise un tableau trié en ordre croissant  $T$  pour stocker les intervalles.

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.

#### 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 élément  $Y$  du sous-arbre gauche de  $T$ , on a  $Y.Int.min < T.Int.min$ .
2. Pour tout sommet  $T$ , pour tout élément  $Y$  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.

#### Question 5

On considère maintenant le programme suivant :

```
def ABRIcherche(I, T) :  
    if estABIdive(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  $I2 = [14, 14]$ .

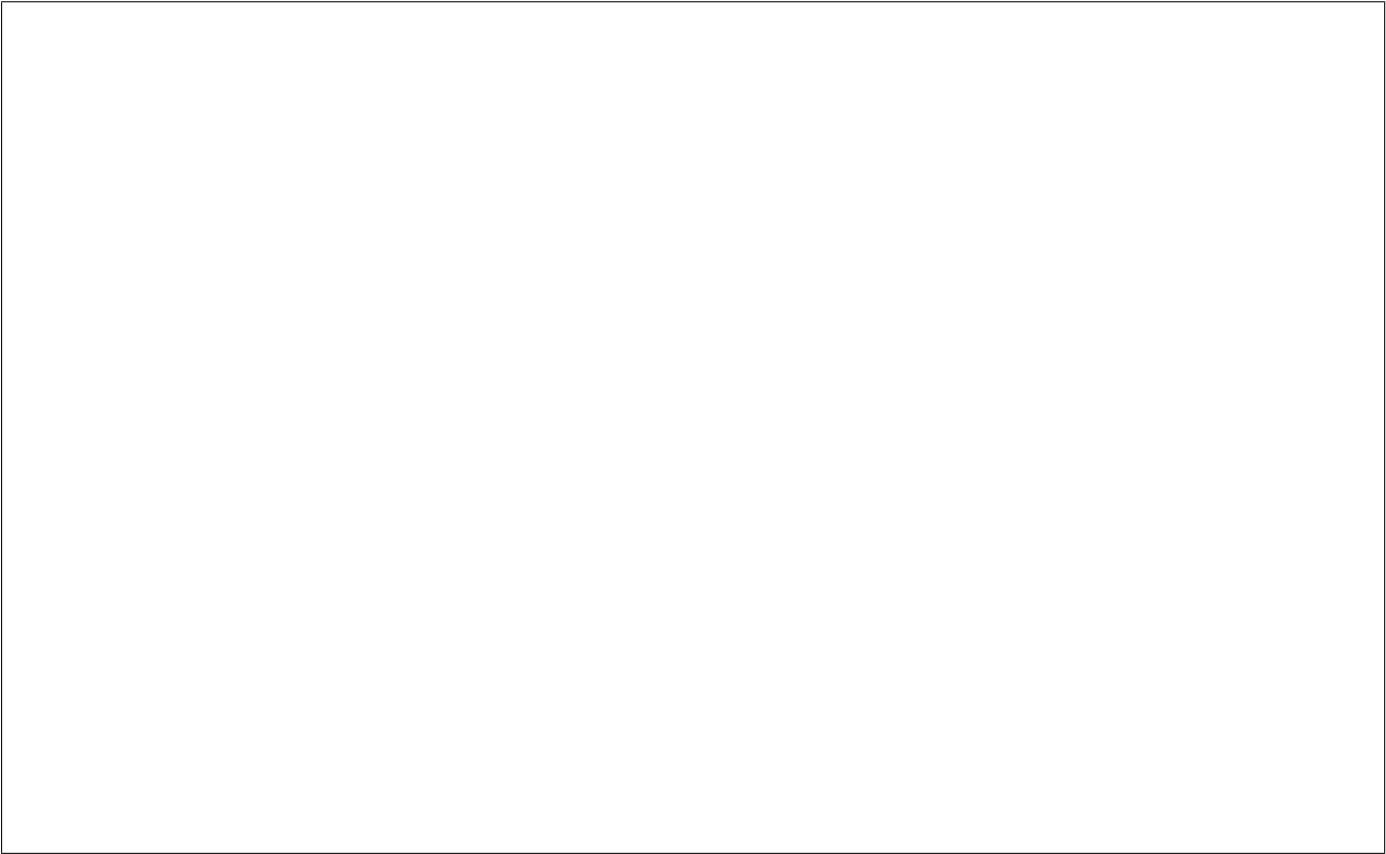
Que fait cette fonction dans le cas général (pour tout arbre  $T$  et tout intervalle  $I$ ) ?

---



**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 ?



---

### Question 7

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

### Question 8

Donnez 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  $[28, 29]$ .