Examen 2I003

Jeudi 21 Juin 2018, 2 heures
aucun document autorisé

Exercice 1 – Problème de décomposition

Préliminaires

On dit qu'une suite d'entiers $(M_0, \ldots, M_{n-1})$ est *supercroissante* si tous ses éléments sont strictement positifs et si chaque élément est strictement supérieur à la somme de ses précédents : $M_i > M_0 + \ldots + M_{i-1}$ pour tout $i = 1, \ldots, n-1$.

Étant donné un entier naturel s et une suite $M = (M_0, \ldots, M_{n-1})$ supercroissante, on cherche à savoir s'il existe n valeurs binaires $x_0, \ldots, x_{n-1}$ ($x_i = 0$ ou 1) telles que $s = x_0 M_0 + \ldots + x_{n-1} M_{n-1}$.
Dans cet exercice on appellera ce problème : *problème de décomposition*.

Question 1

Pour chacune des suites ci-dessous, dire si elle est supercroissante, en justifiant la réponse :
a) $(1, 3, 4, 7, 10)$; b) $(2, 3, 8, 14, 31)$; c) $(1, 2, 4, 8)$.

Solution:
a) $(1, 3, 4, 7, 10)$ n'est pas supercroissante car $1 + 3$ n'est pas strictement inférieur à 4.
b) $(2, 3, 8, 14, 31)$ est supercroissante car $2 < 3$, $2+3 = 5 < 8$, $2+3+8 = 13 < 14$ et $2+3+8+14 = 27 < 31$.
c) $(1, 2, 4, 8)$ est supercroissante car $1 < 2$, $1 + 2 = 3 < 4$ et $1 + 2 + 4 = 7 < 8$.

Question 2

On considère la suite supercroissante $M = (1, 3, 5, 10, 20)$. Pour chacun des entiers suivants, dire si le problème de décomposition admet une solution :
$s_1 = 0$; $s_2 = 26$; $s_3 = 17$.
S'il existe une solution, donner les valeurs des x_i.

Solution:
$s_1 = 0$: oui, $(0, 0, 0, 0, 0)$; $s_2 = 26$: oui, $(1, 0, 1, 0, 1)$; $s_3 = 17$: non.

On considère la fonction :

```
def plusGrandInd(M,s):
    n = len(M)
    k = 1
    while (k < n) and M[k] <= s:
        k = k + 1
    return k - 1
```

où M est un tableau de nombres rangés en ordre croissant et s un nombre supérieur ou égal à $M[0]$.
On rappelle que `len(M)` est le nombre d'éléments du tableau M.

Question 3

On note k_i la valeur de k à la fin de l'itération i. Initialement, c'est-à-dire à la fin de l'itération 0, $k_0 = 1$.

1. Montrer, par récurrence sur i, qu'à la fin de l'itération i, on a $M[j] \leq s$ pour tout j tel que $0 \leq j < k_i$.
2. En déduire que `plusGrandInd(M, s)` retourne le plus grand indice p tel que $M[p] \leq s$.

Solution:

1. **Base** À la fin de l'itération 0, on a $k_0 = 1$. Puisque $M[0] \leq s$ par hypothèse, on a bien $M[j] \leq s$ pour tout j tel que $0 \leq j < k_0$.
   Induction Soit $i > 0$, supposons que la propriété soit vraie pour i et qu'il y ait une itération $i + 1$. Puisque la propriété est vraie pour i, on a $M[j] \leq s$ pour tout j tel que $0 \leq j < k_i$. Puisqu'il y a une itération $i + 1$, on a $M[k_i] \leq s$. Donc $M[j] \leq s$ pour tout j tel que $0 \leq j < k_i + 1 = k_{i+1}$.
   Conclusion La propriété est vraie pour tout $i \geq 0$ tel qu'il existe une itération i.
2. Il y a deux cas de sortie de la boucle `while` : ou bien $k = n$ ou bien ($k < n$ et $M[k] > s$).
   Si $k = n$ alors `plusGrandInd(M,s)` retourne $n - 1$ et, d'après la question précédente, $M[j] \leq s$ pour tout j tel que $0 \leq j < n$. Donc `plusGrandInd(M,s)` retourne le plus grand indice p tel que $M[p] \leq s$.
   Si $k < n$ et $M[k] > s$ alors, d'après la question précédente, on a $M[j] \leq s$ pour tout j tel que $0 \leq j < k$. `plusGrandInd(M,s)` retourne $k - 1$, c'est-à-dire le plus grand indice p tel que $M[p] \leq s$.

Question 4

Calculer la complexité de `plusGrandInd` dans le meilleur cas et dans le pire cas, en précisant quel est le meilleur cas et quel est le pire cas pour un tableau de taille n.

Solution:

Dans le meilleur cas la complexité est en $\Omega(1)$, c'est le cas où $s < M[1]$. Dans le pire cas la complexité est en $O(n)$, c'est le cas où $s \geq M[n-1]$.

Question 5

Soit $M = (M_0, \ldots, M_{n-1})$ une suite supercroissante.

1. Quelle est la plus petite valeur non nulle de s pour laquelle le problème de décomposition admet une solution ?
2. Quelle est la plus grande valeur de s pour laquelle le problème de décomposition admet une solution ?

Solution:

1. La plus petite valeur non nulle de s pour laquelle le problème de décomposition admet une solution est $s = M_0$.
2. La plus grande valeur pour laquelle le problème de décomposition admet une solution est $M_0 + \ldots + M_{n-1}$.

On considère la fonction :

```
def existeDec(M, s):
    print ("Valeur␣de␣s␣:␣", s)
    if s == 0:
        return True
    if s < M[0]:
        return False
    p = plusGrandInd(M, s)
    print ("Valeur␣de␣p␣:␣", p)
    if s - M[p] >= M[p]:
        return False
    return existeDec(M, s - M[p])
```

où M est un tableau représentant une suite supercroissante et s un entier naturel.

Question 6

Exécuter l'appel de `existeDec(ExM, 42))`, avec `ExM = [2,3,8,14,31]`, en donnant les affichages successifs et le résultat final.

Solution:

```
Valeur de s :  42
Valeur de p :  4
Valeur de s :  11
Valeur de p :  2
Valeur de s :  3
Valeur de p :  1
Valeur de s :  0
```

La valeur retournée est `True`.

Question 7

Soit $M = (M_0, \ldots, M_{n-1})$ une suite supercroissante et s un entier naturel non nul. Soit p le plus grand indice tel que $M_p \leq s$.

1. Montrer que, si $s = x_0 M_0 + \ldots + x_{n-1} M_{n-1}$ avec $x_i \in \{0, 1\}$ pour tout $i \in \{0, \ldots n-1\}$, alors :
   (a) $x_j = 0$ si $j > p$,
   (b) $x_p = 1$.

   Indication : faire ces deux preuves en utilisant un raisonnement par l'absurde.
2. En déduire que :
   (a) si $s - M_p \geq M_p$ alors le problème de décomposition n'a pas de solution pour (M, s) (faire un raisonnement par la contraposée) ;
   (b) si $s - M_p < M_p$ alors le problème de décomposition a une solution pour (M, s) ssi il en a une pour $(M, s - M_p)$.

Solution:

1. (a) Soit $j > p$. Si $x_j = 1$ alors $s = x_0 M_0 + \ldots + M_j + \ldots + x_{n-1} M_{n-1}$ donc $M_j \leq s$, ce qui contredit le fait que p est le plus grand indice tel que $M_p \leq s$. Donc $x_j = 0$.
   (b) On a donc $s = x_0 M_0 + \ldots + x_p M_p$. Si $x_p = 0$ alors $s = x_0 M_0 + \ldots + x_{p-1} M_{p-1} \leq M_0 + \ldots + M_{p-1} < M_p$, ce qui contredit le fait que $M_p \leq s$. Donc $x_p = 1$.
2. (a) Si le problème de décomposition a une solution pour (M, s) alors $s = x_0 M_0 + \ldots + M_p$ donc $s - M_p = x_0 M_0 + \ldots + x_{p-1} M_{p-1} \leq M_0 + \ldots + M_{p-1} < M_p$. Par conséquent, si $s - M_p \geq M_p$ alors le problème de décomposition n'a pas de solution pour (M, s).
   (b) Supposons $s - M_p < M_p$.
   Si le problème de décomposition a une solution pour (M, s) alors $s = x_0 M_0 + \ldots + M_p$. Donc $s - M_p = x_0 M_0 + \ldots + x_{p-1} M_{p-1}$ et le problème de décomposition a une solution pour $(M, s - M_p)$.
   Si le problème de décomposition a une solution pour M et $s - M_p$ alors $s - M_p = x_0 M_0 + \ldots + x_{p-1} M_{p-1}$ (puisque $s - M_p < M_p$) donc $s = x_0 M_0 + \ldots + x_{p-1} M_{p-1} + M_p$ et le problème de décomposition a une solution pour (M, s).

Question 8

Prouver, par récurrence forte sur s, que `existeDec(M,s)` se termine et renvoie la valeur `True` si s admet une décomposition et la valeur `False` sinon.

Indication : pour la base, on étudiera le cas où $s = 0$ et le cas où $0 < s < M[0]$.

Solution:

Base Si $s = 0$, `existeDec(M,s)` se termine et renvoie la valeur `True`, ce qui est correct puisque 0 admet une décomposition.
Si $0 < s < M[0]$, `existeDec(M,s)` se termine et renvoie la valeur `False`, ce qui est correct puisque la plus petite valeur non nulle de s pour laquelle le problème de décomposition admet une solution est $s = M[0]$.

Induction Soit $s > M[0]$, supposons que `existeDec(M,r)` se termine et est correct pour tout entier naturel $r < s$.
Si $s - M[p] \geq M[p]$ alors `existeDec(M,s)` se termine et renvoie la valeur `False`, ce qui est correct puisque, dans ce cas, le problème de décomposition n'a pas de solution pour (M, s).
Si $s - M[p] < M_p$ alors `existeDec(M,s)` fait un appel à alors `existeDec(M,s-M[p])`, qui se termine (par hypothèse de récurrence) donc `existeDec(M,s)` se termine. `existeDec(M,s)` renvoie le même résultat que `existeDec(M,s-M[p])`, ce qui est correct puisque, dans ce cas, le problème de décomposition a une solution pour (M, s) ssi il en a une pour $(M, s - M[p])$.

Conclusion `existeDec(M,s)` se termine et renvoie la valeur `True` si s admet une décomposition et la valeur `False` sinon.

Question 9

Calculer la complexité de `existeDec` dans le meilleur cas et dans le pire cas, en précisant quel est le meilleur cas et quel est le pire cas pour un tableau de taille n. Justifier la réponse.

<u>**Solution**</u>:

Dans le meilleur cas la complexité est en $\Omega(1)$, c'est le cas où $s < M_0$. En effet, dans ce cas il n'y a aucun appel récursif.

Le pire cas est celui où $s = M_0 + \ldots + M_{n-1}$. Notons c_k la complexité de l'algorithme pour $t = M_0 + \ldots + M_{k-1}$. Pour un tel t, la complexité de `plusGrandInd` est égale à k, donc $c_k = k + c_{k-1}$. Pour $k = 0$, on a $c_0 = 1$. D'où $c_n = n + (n-1) + (n-2) + \ldots 1 = \dfrac{n(n-1)}{2}$. Dans le pire cas on a donc une complexité en $O(n^2)$.

Exercice 2 – Arbres Binaires, Arbres Binaires de Recherche

Question 1 – Question de cours

Soit T un arbre binaire.

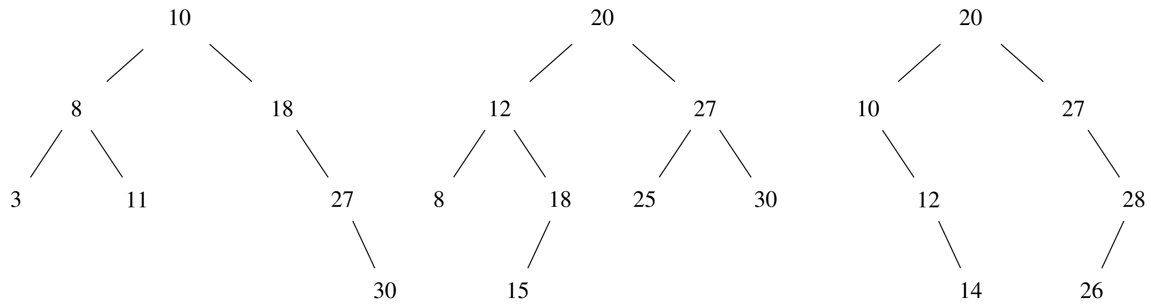
1. Rappelez les définitions inductives d'un arbre binaire étiqueté sur un ensemble E, de sa hauteur $h(T)$ et de son nombre de nœuds $n(T)$.
2. Montrez par <u>induction structurelle</u> que $h(T) \leq n(T) \leq 2^{h(T)} - 1$.
3. Quels sont les arbres binaires tels que $h(T) = n(T)$? Même question pour $n(T) = 2^{h(T)} - 1$.

<u>**Solution**</u>:

1. Voir le cours.
2. Soit T un arbre binaire.
   Base Si $T = \emptyset$ alors $h(T) = 0, n(T) = 0$ et $2^{h(T)} - 1 = 0$ donc la propriété est vraie.
   Induction Si $T = (x, G, D)$ où G et D sont deux arbres binaires. On suppose par induction structurelle que propriété est vraie pour G et D.
   Alors, $h(T) = 1 + max(h(G), h(D)) \leq 1 + max(n(G), n(D)) \leq 1 + n(G) + n(D) = n(T)$, et donc $h(T) \leq n(T)$.
   De même, $n(T) = 1 + n(G) + n(D) \leq 1 + 2^{h(G)} - 1 + 2^{h(D)} - 1 \leq 2.2^{max(h(G),h(D))} - 1 = 2^{1+max(h(G),h(D))} - 1 \leq 2^{(h(T)} - 1$ et donc $n(T) \leq 2^{(h(T)} - 1$.
   Conclusion La propriété est vraie pour tout arbre binaire.
3. Tous les arbres binaires réduits à un chemin vérifient $h(T) = n(T)$. Les arbres tels que $n(T) = 2^{h(T)} - 1$ sont les arbres parfaits dont le dernier niveau est complètement ≪ plein ≫.

Question 2 – Question de cours

1. Rappelez la définition inductive d'un arbre binaire de recherche.
2. Est-ce que les arbres suivants sont des arbres binaires de recherche. Dans la négative, justifiez votre réponse.



B_1 B_2 B_3

Solution:

1. Voir le cours.
2. B_1 n'est pas un arbre binaire de recherche car le sommet 11 est dans le sous-arbre gauche du sommet 10. B_2 est un arbre binaire de recherche. B_3 n'est pas un arbre binaire de recherche car le sommet 26 est dans le sous-arbre droit du sommet 27.

Question 3

Qu'affiche l'algorithme `mystere(B2,8,15)` où B_2 est l'arbre binaire de la question 2 ? Vous préciserez l'arbre des appels et les valeurs retournées à chaque appel.

La fonction `estABRvide(T)` retourne `true` si T est un arbre vide. $L_1 + L_2$ désigne la concaténation des listes L_1 et $L2$.

```
def mystere (T, a, b):
        if estABRvide(T):
                print "appel␣mystere␣(vide,",  a, ",",b, ")"
                print "appel␣mystere␣(vide,",  a, ",",b, ")␣retourne␣[]"
                return []
        L=[]
        print "appel␣mystere␣(", T.clef, ",", a,",", b, ")"
        if (a<T.clef):
                L=mystere(T.gauche, a,b)
        if (a<= T.clef) and (T.clef<=b):
                L=L+[T.clef]
        if (T.clef<b):
                L=L+mystere(T.droit, a,b)
        print  "l'appel␣mystere␣(", T.clef, a,b, ")␣retourne", L
        return L
```

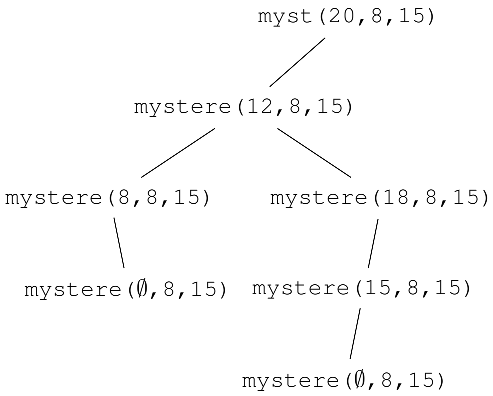
Solution:

On obtient les affichage suivants :

```
>>> mystere(T,8,15)
appel mystere ( 20 , 8 , 15 )
appel mystere ( 12 , 8 , 15 )
appel mystere ( 8 , 8 , 15 )
appel mystere (vide, 8 , 15 )
appel mystere (vide, 8 , 15 ) retourne []
```

```
l’appel␣mystere␣(␣8␣8␣15␣)␣retourne␣[8]
appel␣mystere␣(␣18␣,␣8␣,␣15␣)
appel␣mystere␣(␣15␣,␣8␣,␣15␣)
appel␣mystere␣(vide,␣8␣,␣15␣)
appel␣mystere␣(vide,␣8␣,␣15␣)␣retourne␣[]
l’appel mystere ( 15 8 15 ) retourne [15]
l’appel␣mystere␣(␣18␣8␣15␣)␣retourne␣[15]
l’appel mystere ( 12 8 15 ) retourne [8, 12, 15]
l’appel␣mystere␣(␣20␣8␣15␣)␣retourne␣[8,␣12,␣15]
[8,␣12,␣15]
```

La figure suivante représente l’arbre des appels. Le tableau donne les valeurs retournées.



Appel	Liste retournée
`mystere(20,8,15)`	`[8,12,15]`
`mystere(12,8,15)`	`[8,12,15]`
`mystere(8,8,15)`	`[8]`
`mystere(∅,8,15)`	`[]`
`mystere(18,8,15)`	`[15]`
`mystere(15,8,15)`	`[15]`
`mystere(∅,8,15)`	`[]`

Question 4

Pour tout arbre binaire de recherche T, soit la propriété $\Pi(T)$ suivante :

$\Pi(T)$: pour tout couple $(a, b) \in \mathbb{N}^2$ avec $a \leq b$, mystere(T, a, b) se termine et retourne toutes les clefs de T en ordre croissant qui sont dans l’intervalle $[a, b]$.

Démontez $\Pi(T)$ par induction structurelle sur l’ensemble des ABR.

Solution:

Base : Si $T = \emptyset$, la fonction retourne la liste vide. Elle se termine donc, et aucune clef de T n’est dans l’intervalle $[a, b]$. La propriété est donc vérifiée dans ce cas.

Induction : Soit maintenant $T = (c, G, D)$ un ABR tel que $\Pi(G)$ et $\Pi(D)$ sont vérifiées.

- Par hypothèse d’induction, les appels $\Pi(G)$ et $\Pi(D)$ se terminent. Donc, $\Pi(T)$ se termine.
- Par hypothèse sur les ABR, tous les éléments de G sont strictement inférieurs à c. De plus, si $c \leq a$, alors aucun élément de G n’est dans l’intervalle $[a, b]$. On en déduit que il peut y avoir des éléments de G dans $[a, b]$ uniquement si $c > a$. Soit alors L_G la liste ordonnée de tout ces éléments qui est renvoyée par l’appel mystere(G, a, b). Dans le cas contraire, L_G est la liste vide.
  De même, tous les éléments de D sont strictement inférieurs à c. Si $b \leq c$, alors aucun élément de D n’est dans l’intervalle $[a, b]$. On en déduit que il peut y avoir des éléments de D dans $[a, b]$ uniquement si $b > c$. Soit alors L_D la liste ordonnée de tout ces éléments qui est renvoyée par l’appel mystere(D, a, b). Dans le cas contraire, L_D est la liste vide.
  Au final, c est inférieur aux éléments de G et supérieurs à ceux de D. Si $c \in [a, b]$, la fonction retourne $L_G + [c] + L_D$, sinon $L_G + L_D$. Ces listes sont croissantes dans les deux cas, et contiennent tous les élément de T dans $[a, b]$.

Conclusion : La propriété est donc vérifiée par induction sur l’ensemble des ABR.

Question 5

On souhaite implémenter la fonction `mystere` en utilisant des listes doublement chaînées circulaires.

1. Quelle est la complexité de la fusion de deux listes doublement chaînées circulaires ?
2. En déduire la complexité de la fonction `mystere` dans le pire cas et le meilleur des cas. Justifiez vos réponses.

Solution:

1. La fusion de deux listes doublement chaînées circulaires est en $\Theta(1)$.
2. Dans le meilleur des cas, $a = b = c$. Il n'y a alors pas d'appels récursifs, la fonction contient un nombre borné d'instructions et est alors en $\Omega(1)$.

   Dans le pire des cas, toutes les clefs de T sont dans l'intervalle $[a, b]$. Il y a alors au plus 3 appels par noeuds (en comptant les appels sur des noeuds vides). Chaque appel pris séparément contient un nombre borné d'instruction, la fusion des listes étant en $\Theta(1)$. On en déduit que la complexité est en $\mathcal{O}(n)$, où n est la taille de T.