ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Mi-parcours corrigé 15 Novembre 2010

Problème 1 [10 points].

Soit $A(n) = \sum_{k=1}^{n} k \cdot k!$.

1. Calculer $A(n)$ pour $n = 1, 2, 3, 4, 5$.
2. Comparer $A(n)$ et $n!$ pour $n = 1, 2, 3, 4, 5$.
3. Deviner une formule générale pour $A(n)$ et prouver cette formule par induction.

Corrigé :

1. *[2"]*

n	$A(n)$	$n!$
1	1	1
2	5	2
3	23	6
4	119	24
5	719	120

2. *[3"]*Il semblerait que

$$A(n) = (n+1)! - 1 \tag{1}$$

3. *[5"]*On veut montrer par induction que la formule (**??**) est juste pout tout $n \geq 1$.

   Base : Quand $n = 1$, on a $(1+1)! - 1 = 2 - 1 = 1 = A(1)$.

   Pas : Supponsons maintentant que (**??**) est vraie pour n. Nous voulons montrer qu'elle est ausi vraie pour $n+1$.

$$\begin{aligned}
(n+1) &= \textstyle\sum_{k=1}^{n+1} k \cdot k! \\
&= \textstyle\sum_{k=1}^{n} k \cdot k! + (n+1) \cdot (n+1)! \\
&= A(n) + (n+1) \cdot (n+1)! \\
&= (n+1)! - 1 + (n+1) \cdot (n+1)! \quad \text{(par hypothèse d'induction)} \\
&= ((n+1)+1) \cdot (n+1)! - 1 \\
&= (n+2)! - 1
\end{aligned}$$

   et donc par induction (**??**) est vraie pour tout $n \geq 1$.

Problème 2 [15 points]. Etant donné $S \subseteq \mathbb{N}$ et $x \in \mathbb{N}$, on considère le problème suivant : est-ce qu'il existe un ensemble $T \subseteq S$ tel que $x = \sum_{i \in T} i$? (Ce problème est appelé le problème du *subset sum*).

1. Donner la spécification formelle de ce problème.
2. On suppose que S est donné par $\{s_1, s_2, \ldots, s_n\}$ satifaisant la propriété suivante : $s_2 > s_1$, $s_3 > s_2 + s_1$, et en général $s_k > \sum_{i=1}^{k-1} s_i$ pour tout k. Donner un algorithme qui trouve la réponse en n additions au plus. Expliquer pourquoi l'algorithme marche.

Corrigé :

Le problème de la somme de sous-ensembles :

1. *[5"]* L'entrée I consiste en un sous-ensemble $S \subseteq \mathbb{N}$ et un entier $x \in \mathbb{N}$. Dire que S est un sous-ensemble de $\mathbb{N}$ est la même chose que de dire que S appartient à l'ensemble des parties de $\mathbb{N}$, i.e. $S \in \mathrm{Pot}(\mathbb{N})$. Ainsi, l'entrée est un couple $(S, x) \in I$, où $I = \mathrm{Pot}(\mathbb{N}) \times \mathbb{N}$.
   Ce problème est un problème de décision : la sortie peut être soit "oui" soit "non". Ainsi $O = \{\mathrm{Oui}, \mathrm{Non}\}$.
   Nous définissons à présent l'ensemble W de toutes les entrées pour lesquelles la sortie est "oui". Cet ensemble est
   $$W := \left\{ (S, x) \in I \;\middle|\; \exists T \in \mathrm{Pot}(S) \text{ tel que } \sum_{y \in T} y = x \right\}.$$
   En utilisant l'ensemble W, on peut écrire la relation de dépendance R comme suit :
   $$R := W \times \{\mathrm{Yes}\} \cup (I \setminus W) \times \{\mathrm{No}\}$$
2. *[10"]*On peut utiliser l'algorithme glouton suivant :
   Input: $x \in \mathbb{N}$, $S = \{s_1, \ldots, s_n\}$ avec $\sum_{i=1}^{k-1} s_i < s_k$ $1 \leq k \leq n$
   Output: Oui si $x = \sum_{y \in T} y$ pour un certain $T \subseteq S$, Non sinon.

```
i ← n
while i > 0 do
  if s_i ≤ x then
    x ← x − s_i
    if x = 0 then
      return Oui
    end if
  end if
  i ← i − 1
End
return Non
```

(Une suite s_i qui vérifie
$$s_k > \sum_{i=1}^{k-1} s_i$$

est appelée *supercroissante*. L'entrée S est supercroissante.)

Voyons à présent pourquoi l'algorithme renvoie la bonne réponse. Informellement, nous pouvons dire que l'algorithme choisit toujours le plus grand s_i inférieur à x. Cette stratégie est bonne puisque si s_i n'est pas utilisé, il sera impossible d'obtenir une somme suffisamment grande, étant donné que la somme de tous les éléments plus petits est plus petite que s_i et donc aussi que x.

Essayons de montrer cela plus formellement. Soit $x \in \mathbb{N}, S \subseteq \mathbb{N}, s \in S$ avec s tel que $\sum_{y \in S \setminus \{s\}} y < x$. Considérons les deux assertions suivantes

(1) Il existe $T \subseteq S$ tel que $x = \sum_{y \in T} y$ et $s \in T$.

(2) Il existe $T \subseteq S$ tel que $x = \sum_{y \in T} y$.

Il est clair que (1) $\Longrightarrow$ (2). Supposons que (2) soit vrai. Nous voulons montrer que (1) est aussi vrai. Supposons par l'absurde que $s \notin T$. Alors puisque $T \subseteq (S \setminus \{s\})$,

$$\sum_{y \in T} y \leq \sum_{y \in (S \setminus \{s\})} y < x,$$

et donc T ne vérifie pas l'hypothèse $\sum_{y \in T} y = x$, d'où la contradiction. Ainsi $s \in T$ et donc (1) est vrai.

Il s'ensuit que dans notre contexte, les assertions (1) et (2) sont équivalentes. Or l'assertion (1) ci-dessus est aussi équivalente à

$$(x - s) \text{ est une somme de sous-ensemble de } S \setminus \{s\}. \tag{2}$$

Puisqu'à chaque itération, s_i vérifie la propriété

$$s_i > \sum_{k=1}^{i-1} s_k,$$

on peut appliquer (**??**) à chaque itération, ce qui établit la correction de l'algorithme.

Remarquez que l'hypothèse que la suite s_i est supercroissante est nécessaire au bon fonctionnement de l'algorithme. Considérons l'ensemble $S = \{3, 4, 5\}$ qui n'est pas supercroissant et posons $x = 7$. L'algorithme glouton ci-dessus commencerait par prendre 5 et vérifierait ensuite si $2 = 7 - 5$ est une somme de sous-ensemble de $\{3, 4\}$, pour finalement renvoyer Non comme réponse. Mais $7 = 3 + 4$ est clairement une somme de sous-ensemble de S.

Problème 3 [25 points].

1. Supposons que nous disposons uniquement de la structure de données stack. Est-il possible de réaliser une file d'attente en utilisant deux stacks ? Si oui, donner les algorithmes correspondants pour les opérations de la file d'attente.
2. Nous voulons de nouveau réaliser une file d'attente en utilisant deux stacks, mais avec la condition de plus que le temps de calcul des opérations est similaire à celui d'une implémentation classique de file d'attente.

   Plus précisément, supposons que les opérations sur les stacks se font en temps $O(1)$ (i.e. ne dépendent pas de la taille du stack). Donner alors des algorithmes pour les opérations d'une file d'attente basée sur deux stacks qui ont la propriété que, si l'on commence avec une file vide, pour faire les m premières opérations de la file d'attente, on a besoin de $O(m)$ opérations du stack.

Corrigé :

1. *[10"]* Soient S_1 et S_2 deux stacks. Nous aimerions réaliser une file d'attente Q en utilisant S_1 et S_2.

   Nous pouvons réaliser une telle file d'attente qui contient à $top[S_1]$ l'élément le plus récemment ajouté, et la file continue alors jusqu'à la fin de S_1, continue à la fin de S_2 jusqu'à $top[S_2]$ qui contient l'élément le plus ancien.

   L'implémentation de `Enqueue`(x) est triviale :

   1: `Push`(S_1, x)

   L'idée de l'algorithme pour `Dequeue`(x) est d'enlever un élément de S_2. S'il n'y en a plus, on bouge le contenu de S_1 dans S_2 et on ressaye. Le voici :

   1: $r \leftarrow$ `Pop`(S_2)
   2: **if** $r =$ underflow **then**
   3: $\quad r \leftarrow$ `Pop`(S_1)
   4: $\quad$ **while** $r \neq$ underflow **do**
   5: $\quad\quad$ `Push`(S_2, r)
   6: $\quad\quad r \leftarrow$ `Pop`(S_1)
   7: $\quad$ **End**
   8: $\quad r \leftarrow$ `Pop`(S_2)
   9: **end if**
   10: **return** r

2. *[15"]* Avec les algorithmes comme présentés sous le point a) ci-dessus, chaque élément est enlevé au plus deux fois et inséré au plus deux fois dans stack, ce qui fait que la propriété voulue est vérifiée.

Problème 4 [15 points].

Soit T un arbre binaire AVL de recherche, de hauteur h (on rapelle que bal(k) est défini comme la hauteur du sous-arbre de gauche de k moins la hauteur du sous-arbre de droite de k ; dans un arbre AVL, pour tout sommet k, bal(k) $\in \{0, +1, -1\}$). Un élément est inséré dans T, et un arbre binaire de recherche T' est obtenu. Répondre aux questions suivantes concernant T' *avant qu'aucune rotation soit effectuée.*

1. Quel est le *plus petit* nombre possible de sommets avec bal(k) $\in \{+2, -2\}$? Justifier brièvement la réponse.
2. Quel est le *plus grand* nombre possible de sommets avec bal(k) $\in \{+2, -2\}$? Montrer un exemple avec ce nombre et expliquer pourquoi c'est le plus grand nombre possible.
3. Soit s un sommet de T', et soit t son fils de gauche. Est-il possible d'avoir bal(t) $= +2$ et bal(s) $= -1$? Si oui, donner un exemple. Si non, expliquer pourquoi.

Corrigé :

1. *[5"]*Le plus petit nombre de sommets possible satisfaisant bal(k) $\in \{+2, -2\}$ est 0. Par exemple, si T est un arbre binaire complet, alors aucune insertion ne peut conduire à bal(k) $= 2$ à quelque sommet que ce soit de T'.
2. *[5"]*Le plus grand nombre de sommets possible avec bal(k) $\in \{+2, -2\}$ est h. *Cf.* par exemple la figure ci-dessous pour constater que h peut être atteint. Pour voir que h ne peut pas être dépassé : il y a au plus $h + 1$ sommets de T dont la hauteur a changé à cause de l'insertion. Si $h + 1$ sommets voient leur hauteur changer, alors nécéssairement l'élément a été inséré sous une feuille de T, qui ne peut pas passer de bal(k) $= 0$ à bal(k) $= \pm 2$.

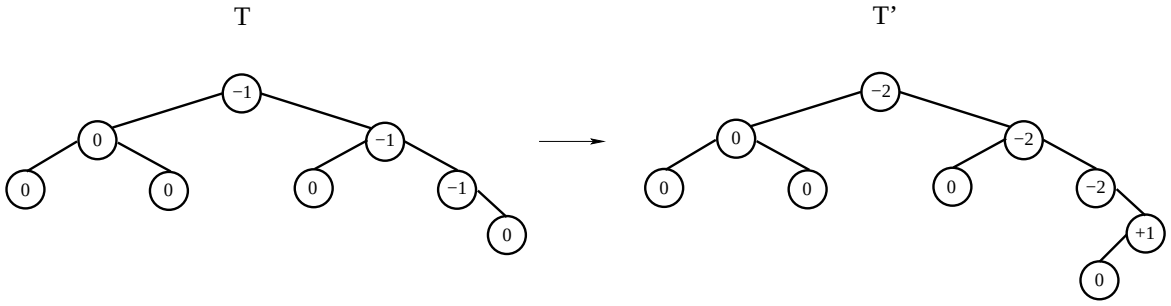

FIG. 1 – Exemple où une insertion peut entraîner bal=-2 sur h sommets. (h=3 dans cette figure)

3. *[5"]*Non. Ce n'est pas possible. Si bal(t) $= +2$, cela implique que la hauteur de t a augmenté à cause de l'insertion. Mais alors la hauteur du sous-arbre à gauche de s a augmenté aussi. Et alors bal(s) $= -1$ dans T' impliquerait que bal(s) $= -2$ dans T, contredisant la propriété AVL.