ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Correction de Midterm 2 Novembre 2009

Problème 1 [10 points]. Donner une spécification formelle du problème suivant : décider si la somme des éléments d'un ensemble fini et non vide de nombres naturels est paire.

1. Quel est l'ensemble des entrées ?
2. Quel est l'ensemble des sorties ?
3. Quelle est la dépendance relationnelle ?

Solution.

1. L'input est un sous-ensemble fini et non vide de $\mathbb{N}_0$; l'ensemble I de tous les inputs possibles est donc l'ensemble de tous les sous-ensembles finis de $\mathbb{N}_0$, à l'exception de l'ensemble vide qui est exclu: $I = \mathrm{Pot}^*(\mathbb{N}_0)\backslash\{\emptyset\}$.
2. Si on avait une machine pour résoudre ce probléme, quand on lui donne un ensemble fini et non vide, elle nous répondrait soit "vrai" (si la somme des éléments de cet ensemble est paire), soit "faux" (si la somme de ces éléments est impaire). L'ensemble O des outputs possibles est donc $O = \{\text{vrai}, \text{faux}\}$.
3. On pose $S = \{s_0, \ldots, s_{n-1}\}$ avec tous les s_i dans $\mathbb{N}_0$. La dépendance relationnelle R est

$$\begin{aligned} R \quad = \quad & \left\{ (S, \text{vrai}) \in I \times O \;\middle|\; \exists k \in \mathbb{N}_0 : \sum_{s \in S} s = 2k \right\} \\ & \bigcup \left\{ (S, \text{faux}) \in I \times O \;\middle|\; \exists k \in \mathbb{N}_0 : \sum_{s \in S} s = 2k + 1 \right\} \end{aligned}$$

Problème 2 [20 points].

Répondre aux questions suivantes. Pas besoin de justifier.

1. Donner un élément de $\mathrm{Pot}(\{a,b,c\}) \times \{a,b\}$.

   Tout élément de $\mathrm{Pot}(\{a,b,c\})\times\{a,b\}$ est une paire formée d'un sous-ensemble de $\{a,b,c\}$ et d'un élément de $\{a,b\}$. Un exemple possible est $(\{a,b\},a)$.

2. Donner une relation symétrique sur $\{a,b,c\}$.

   Soit $S=\{a,b,c\}$. On rapelle qu'une relation R sur S est un sous-ensemble $R \subseteq S\times S$. L'inverse de R est
   $$R^{-1} = \{(x,y) \in S\times S | (y,x) \in R\}.$$
   R est symétrique si $R^{-1} = R$. Un exemple d'une relation symétrique sur S est $R=\{(a,b),(b,a),(c,c)\}$.

3. Donner un sous-ensemble de $\mathrm{Pot}(\{1,2,3\})$.

   La réponse doit être une collection de sous-ensembles de $\{1,2,3\}$. Par exemple, une réponse possible est $\{\{1,2\},\{3\}\}$.

4. Soit S un ensemble avec $|S|=n$. Que vaut $|\mathrm{Pot}(S)|$?

   Le nombre de sous-ensembles de S de taille k est $\binom{n}{k}$. Donc
   $$|\mathrm{Pot}(S)| = \sum_{k=0}^{n} \binom{n}{k} = 2^n.$$

5. $n = \theta\big(\log_2(3^n)\big)$. Vrai ou Faux?

   Puisque $\log_2(3^n) = n\log_2(3)$, il est clair que l'assertion est VRAIE.

6. $\sqrt{n} = O\big((\log_2 n)^2\big)$. Vrai ou Faux?

   Puisque la fonction log croît plus lentement que *toute* fonction polynomiale, $\log n$ croît plus lentement que $n^{1/4}$, ce qui implique que $(\log n)^2$ croît plus lentement que $\sqrt{n}$. Donc l'assertion est FAUSSE.

7. $2^{(\ln^2 n)} = O\big(n^2\big)$. Vrai ou Faux?

   On peut vérifier que $2^{(\ln^2 n)} = \big(n^{\ln n}\big)^{\ln 2}$. Puisque $\ln n$ croît plus rapidement qu'une constante, l'assertion est FAUSSE.

8. Soit $p(n)$ un ploynôme de degré 4. Classer les fonctions suivantes de telle façon que f soit avant g si $f(n) = o\big(g(n)\big)$:

$$\left(\frac{1}{2}\right)^n, e^{2\pi n}, p(n)^2, n^5, 1.61803, \ln^6(n)$$

   Noter d'abord que

$$\lim_{n\to\infty} \left(\frac{1}{2}\right)^n = 0,$$

   et donc que $\left(\frac{1}{2}\right)^n = o(1)$. En utilisant ce fait et le fait que la fonction log croît plus lentement que toute fonction polynomiale, qui elle-même croît plus lentement que toute fonction exponentielle, on obtient le classement suivant:

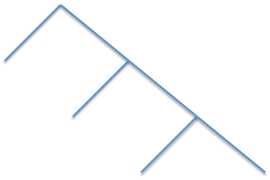
$$\left(\frac{1}{2}\right)^n, 1.61803, \ln^6(n), n^5, p(n)^2, e^{2\pi n}$$

9. Donner une fonction $g(n)$ avec $g(n) \neq n^2 + n + 1$, et $g(n) = \theta(n^2 + n + 1)$.

   $g(n)$ peut être toute fonction qui est $\theta(n^2)$, avec $g(n) \neq n^2 + n + 1$. Par exemple, une solution possible est $g(n) = n^2$.

10. Dessiner un arbre binaire dans lequel tous les sommets ont degré 0 ou 2, mais qui n'est pas un arbre AVL.

    Un exemple d'un tel arbre figure ci-dessous:



11. Supposons qu'on a un arbre binaire de recherche T de hauteur h, et qu'on efface un élément (avec l'algorithme du cours) pour obtenir un nouvel arbre binaire de recherche T'. Quelles sont les valeurs possibles pour la hauteur h' de T'?

On peut vérifier que l'algorithme du cours peut soit préserver la hauteur de l'arbre, soit la réduire par 1. Donc $h' = h$ ou $h' = h - 1$.

12. Soit T un arbre avec n sommets. Combien T a-t-il d'arêtes?

    Par la formule d'Euler, T a $n - 1$ arêtes.

13. Soit T un arbre binaire avec n sommets. Quel est la plus petite hauteur que peut avoir T? Quel est la plus grande hauteur que peut avoir T?

    Intuitivement, T a une hauteur minimale quand il est aussi proche que possible d'un arbre binaire complet. On peut vérifier que ceci correspond à une hauteur minimale de $\lfloor \log_2 n \rfloor$. T a une hauteur maximale quand tous les sommets on degré 1, sauf les feuilles, ce qui résulte en une hauteur de $n - 1$.

14. Combien faut-il de mémoire pour représenter un graphe orienté à n sommets par sa matrice d'adjacence?

    Comme on l'a vu en classe, il faut n^2 éléments de mémoire.

15. Soit G un graphe non-orienté avec n sommets représenté par des listes d'adjacence. Si la première liste contient $n - 1$ éléments alors G est connexe. Vrai ou Faux?

    Si la première liste contient $n - 1$ éléments, alors il existe une arête entre le sommet 1 et tout autre sommet du graphe. On peut donc trouver un chemin de longueur 2 entre chaque paire de sommets du graphe qui passe par le sommet 1. Le graphe est donc connexe, et l'assertion est VRAIE.

16. Soit T un arbre binaire de recherche avec n sommets. Pour faire une recherche dans T, combien faut-il de comparaisons dans le pire des cas (en notation θ)? En moyenne, quel est l'ordre du nombre de comparaisons nécessaires (en notation θ)?

    Comme on l'a vu en classe, la complexité dans le pire des cas est $\theta(n)$, et la complexité moyenne est $\theta(\log n)$.

17. Soit T un arbre AVL avec n sommets. Pour faire une recherche dans T, quel est l'ordre du nombre de comparaisons nécessaires dans le pire des cas (en notation θ)?

    Par un théorème prouvé en classe concernant la hauteur des arbres AVL, la complexité de la recherche dans un arbre AVL est dans le pire des cas $\theta(\log n)$.

18. Quel est l'ordre du nombre d'opérations nécessaires pour multiplier deux matrices $n \times n$ avec l'algorithme naïf (en notation θ)?

    L'algorithme naïf utilise n^3 multiplications et $n^2(n-1)$ additions. Le nombre d'opérations est donc $\theta(n^3)$.

19. L'algorithme de Strassen permet de multiplier deux matrices $n \times n$ en utilisant $O(n^2)$ opérations. Vrai ou Faux?

    L'algorithme de Strassen utilise $O(n^{\log_2(7)})$ operations, donc l'assertion est FAUSSE.

20. Soit un tableau de hachage contenant n éléments. Quel est le temps pour la recherche d'un élément dans le pire des cas (en notation θ)?

    Dans le pire des cas, tous les éléments pourraient avoir la même adresse de hachage. Dans ce cas on aura besoin de passer sur tous les éléments pour chercher un élément donné. La complexité dans le pire des cas est donc $\theta(n)$.

Problème 3 [15 points]. Soit $A(n) = \sum_{k=1}^{n} k \cdot k!$.

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

---

1. **Corrigé:**(2 points)

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

2. **Corrigé:**(3 points)
Il semblerait que

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

3. **Corrigé:**(10 points)
On veut montrer par induction que la formule (1) 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 (1) est vraie pour n. Nous voulons montrer qu'elle est ausi vraie pour $n+1$.

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

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

Problème 4 [20 points].

1.

$$T_0 = [1]\,.$$

$$T_1 = \begin{bmatrix} T_0 & T_0 \\ T_0 & -T_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$$T_2 = \begin{bmatrix} T_1 & T_1 \\ T_1 & -T_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

$$T_3 = \begin{bmatrix} T_2 & T_2 \\ T_2 & -T_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

2.

$$xT_m = \begin{bmatrix} x_1 & x_2 \end{bmatrix} T_m = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} T_{m-1} & T_{m-1} \\ T_{m-1} & -T_{m-1} \end{bmatrix} =$$

$$\begin{bmatrix} x_1 T_{m-1} + x_2 T_{m-1} & x_1 T_{m-1} - x_2 T_{m-1} \end{bmatrix} = \begin{bmatrix} y_1 + y_2 & y_1 - y_2 \end{bmatrix}.$$

3. L'algorithme suivant, de type diviser-pour-règner, calcule le produit xT_m, en $O(m2^m)$.

Algorithme 1 MATRIX(m, x)

```
1: if m = 0 then
2:     return x
3: else
4:     x_1 ← first_half(x)
5:     x_2 ← second_half(x)
6:     y_1 ← MATRIX(m − 1, x_1)
7:     y_2 ← MATRIX(m − 1, x_2)
8:     y = [y_1 + y_2   y_1 − y_2]
9:     return y
10: end if
```

La taille du problème est donnée par celle de x, qui est $n = 2^m$. Le problème est divisé en deux sous problèmes ($c = 2$), et chaque sous problème a une taille $\frac{n}{2}$ ($a = 2$). Le temps de parcours de la combinaison des sous problèmes est linéaire en n ($b = 1$). Du fait que $a^b = 2 = c$, nous obtenons ainsi un temps de parcours $O(n^{\log_2 2}.\log_2(n)) = O(m2^m)$.

Problème 5 [15 points]. Quel temps de calcul faut-il compter asymptotiquement pour chacun des algorithmes suivants ?

1. L'algorithme A divise le problème en 5 sous-problèmes de taille moitié et combine leur solution en un temps linéaire.
2. L'algorithme B résout par récurrence un problème de taille n en résolvant deux sous-problèmes de taille $n-1$ puis en combinant leur solution en un temps constant.
3. L'algorithme C divise le problème de taille n en 9 sous-problèmes de taille $n/3$ chacun, puis combine les solutions en un temps quadratique en n.

Solution.

1. Soit $T(n)$ le temps de parcours de l'algorithme pour des inputs de taille n. On a alors la récurrence $T(2n) = 5T(n)+O(n)$. On peut appliquer le théorème 2.1 du cours (page 39) avec $a = 2$, $c = 5$, et $b = 1$ pour obtenir (en utilisant le cas (1)) $T(n) = O(n^{\log_2 5})$.
2. La récurrence est $T(n) = 2T(n-1) + O(1)$, ce qui signifie qu'il existe un $a \in \mathbb{R}$ positif tel que $T(n) \leq 2T(n-1) + a$. Donc

$$T(n) \leq 2T(n-1) + a \leq 4T(n-2) + 3a \leq 8T(n-3) + 7a \leq \cdots$$

   et en général $T(n) \leq 2^i T(n-i) + (2^i - 1)a$. En prenant $i = n-1$, on en conclut que $T(n) = O(2^n)$, en utilisant le fait que $T(1) = O(1)$.
3. On a: $T(3n) = 9T(n) + O(n^2)$. On peut appliquer à nouveau le théorème 2.1 (cas (3)) pour obtenir $T(n) = O(n^2 \log n)$.

Problème 6 [20 points]. Étant donné une suite strictement croissante $(a_1, \ldots, a_n)$ de n entiers, concevez un algorithme de type diviser-pour-régner qui détermine s'il existe un indice i tel que $a_i = i$ et qui s'exécute en un temps $O(\log n)$.

Solution. Remarquons que $a_{i+1} \geq a_i + 1$ pour tout $i \in \{1, \cdots, n\}$. Ainsi, si $a_i < i$, alors $a_j < j$, pour tout $j < i$ et l'indice i recherché ne peut se retrouver à gauche de a_i. Similairement, si $a_i > i$, alors $a_j > j$, pour tout $j > i$. Ainsi, nous pouvons chercher la valeur de l'indice i en utilisant le concept de l'algorithme de recherche binaire. L'algorithme `Point-Fixe` suivant, de type diviser-pour-régner, détermine s'il existe un indice i tel que $a_i = i$ (l'appel initial à cet algorithme serait `Point-Fixe`$(a, 1, n)$):

```
1: Point-Fixe(a, ℓ, r)
2: i ← ⌊(ℓ + r)/2⌋.
3: if a_i = i then
4:    return i
5: else if ℓ = r then
6:    return "no match"
7: else if a_i < i then
8:    return Point-Fixe(a, i + 1, r)
9: else
10:   // a_i > i
11:   return Point-Fixe(a, ℓ, i − 1)
12: end if
```

Le temps de parcours de l'algorithme `Point-Fixe` peut s'écrire $T(n) = T(n/2) + O(1)$ (car à chaque appel récursif, la recherche se réduit de moitié). Nous obtenons ainsi $T(n) = O(\log n)$.