L2 Informatique - Algorithmique - Correction session 1

mardi 16 décembre 2014. 09h00-11h00. A 400.

La précision et la clarté de votre rédaction sont *fondamentales*. Documents interdits. Durée 2h00. Le barème indiqué est *approximatif*.

Exercice 1. [2pts] Quelle est la valeur de la somme des entiers compris entre 39 et 78 ? Soit n un entier strictement positif, calculez la somme

$$\sum_{i=0}^{n} 3^i.$$

Solution. La somme des n premiers entiers non-nuls est fournie par

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Il suffit donc de soustaire à la somme des entiers de 1 à 78, la somme des entiers de 1 à 38 :

$$\sum_{i=39}^{78} i = \frac{78 \times 79}{2} - \frac{38 \times 39}{2} = \frac{78 \times 79 - 38 \times 39}{2} = 4680.$$

D'autre part (programme de 1ère, suites géométriques de raison q)

$$S := \sum_{i=0}^{n} q^i = 1 + q + q^2 + \cdots + q^n$$

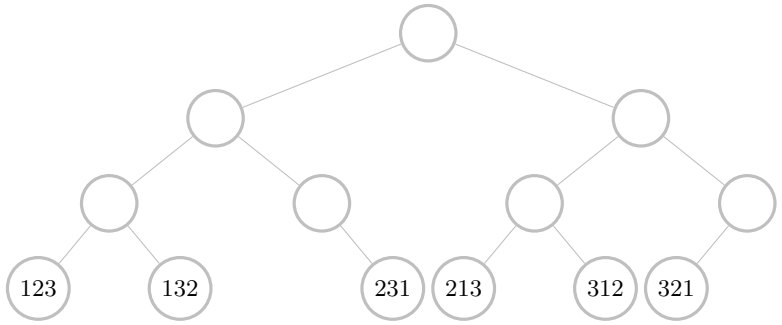
Donc $qS = q + q^2 + \cdots + q^{n+1}$ et $1 + qS - q^{n+1} = S$. Finalement

$$S = \frac{q^{n+1} - 1}{q - 1} \tag{1}$$

Il suffit de remplacer q par 3.

Exercice 2. [4pts] Dessinez l'arbre binaire de décision du tri par sélection par le minimum pour les instances de taille 3. Les branches droites correspondent aux comparaisons qui sont satisfaites, les feuilles doivent contenir la liste avant le tri. De manière plus générale, combien de feuilles y-a-t-il au minimum dans l'arbre de décision associé à un algorithme de tri pour une instance de taille n ?

Solution. Le tri sélection se fait en cherchant le minimum dans une liste L et c'est donc la comparaison $L[i] < \min$ qui est réalisée à chaque étape de la boucle, le minimum étant initialisé à la première valeur de la liste à chacune des $n-1$ étapes de recherche. Pour la permutation identité $\sigma = 123$, min est initialisé à 1 et on compare successivement min à la valeur 2 et à la valeur 3, le test $L[i] < \min$ échoue dans les deux cas ce qui nous fait descendre deux fois à gauche dans l'arbre. La valeur 1 est donc le minimum et reste en première position. On recommence sur la sous-liste constituée des deux derniers termes 2 et 3, le minimum est cette fois initialisé à la valeur 2 et le dernier test consiste à comparer min à la valeur 3 qui fait encore échouer le test et descendre une dernière fois à gauche dans l'arbre :



NB. Si vous avez considéré la comparaison $L[i] > \min$ au lieu de $L[i] < \min$, l'arbre obtenu est symétrique.

Exercice 3. [2pts] Écrivez un algorithme `MinMax(L):entier` qui renvoie le *premier* indice du terme le plus petit de la liste L ainsi que le *dernier* indice du terme le plus grand de la liste L. Par exemple pour la liste $[2, 1, 4, 3, 7, 9, 2, 1, 9, 3]$, l'algorithme renvoie les valeurs 2 et 9 (on suppose ici que l'indexation commence par 1). On veut que l'algorithme trouve ces deux indices *en une seule passe* sur la liste. Quelle est la complexité de cet algorithme ?

Solution. Il suffit d'initialiser deux variables qui contiendront ces deux indices à l'indice 1, balayer la liste et mettre ces indices à jours si nécessaire. Pour que l'indice de l'élément minimal soit lui aussi minimal, il suffit que la comparaison avec le terme courant soit stricte et pour que l'indice de l'élément maximal soit maximal, il faut cette fois que la comparaison soit large afin de mettre à jour y compris si l'on rencontre la même valeur maximale.

La complexité de l'algorithme est très simple à évaluer, il n'y a qu'une seule boucle dont l'étendue est la longueur n de la liste à parcourir et les comparaisons ainsi que l'incrémentation de la variable de boucle, se font en $\Theta(1)$. Ce dernier temps n'est pas à strictement parler constant puisque la mise à jour

des variables *imin* et *imax* n'est pas systématique mais il est borné par deux constantes. Finalement $T(n) = \Theta(n)$.

```
Algorithme MinMax(L) : couple
données
    L : liste de valeurs
variables
    imin, imax, i : entiers
---
imin ← 1
imax ← 1
i ← 2
tantque (i ≤ #L) faire
    si (L[i] < L[imin]) alors
        imin ← i
    sinon
        si (L[i] >= L[imax]) alors
            imax ← i
        fsi
    fsi
    i ← i + 1
ftq
renvoyer(imin, imax)
```

Algo 1. Recherche des indices du min et du max.

Exercice 4. [2.5pts] Écrivez l'expression arithmétique suivante sous forme postfixée :

$$(2 + 3) \times (7 - 1) + 2$$

Évaluez cette expression à l'aide d'une pile. On rappelle qu'on lit les termes de l'expression postfixée et que, selon qu'il s'agisse d'un opérande x ou d'un opérateur $\star$, on empile x ou on dépile les deux valeurs a et b au sommet de la pile et on empile $a \star b$.

Faites un tableau contenant les différents états de la pile au fur et à mesure de l'évaluation de l'expression postfixée que vous aurez trouvée. Quel est le résultat final dans la pile ?

Solution. L'expression postfixée correspondante est l'expression

$$2, 3, +, 7, 1, -, \times, 2, +$$

				1				
	3		7	7	6		2	
2	2	5	5	5	5	30	30	32
2	3	+	7	1	−	×	2	+

Le résultat final est bien sûr 32, que vous ayez réussi ou non à faire la pile.

Exercice 5. [6pts] Soient u et v deux séquences de valeurs dans un ensemble E de longueurs respectives n et m avec $n \leq m$. On dit que u est une *sous-séquence* de v s'il existe une application strictement croissante $\sigma : [1; n] \to [1; m]$ telle que $\forall i \in [1; n],\ u_i = v_{\sigma(i)}$. Par exemple le mot "cas" est une sous-séquence du mot "carnages" (ou encore "carnages"), ainsi que le mot "rage" ("carnages"). La séquence vide est toujours considérée comme une sous-séquence de n'importe quelle séquence.

En supposant que la séquence v est constituée de n valeurs distinctes, combien peut-on construire de sous-séquences u de v ?

Écrivez un algorithme `EstSousSeq(u,v):booleen` qui renvoie `vrai` si u est une sous-séquence de v et `faux` sinon. On notera $u[i]$ le i-ème terme de la séquence u. Estimez la complexité de cet algorithme dans le meilleur des cas et dans le pire des cas en fonction des longueurs n et m des séquences u et v.

Solution. Soit $u = u_1 u_2 \dots u_n$ une séquence de longueur n et $b = b_1 b_2 \dots b_n$ une séquence *binaire* de même longueur. On associe à b la sous-séquence de u constituée par les termes u_i tels que $b_i = 1$. À chaque sous-séquence de u correspond une unique séquence binaire et réciproquement, il y a donc autant de sous-séquences de u que de séquences binaires de longueur n, soit 2^n.

On se donne deux indices i et j pour indexer les séquences u et v respectivement, tous deux initialisés au premier indice 1. La boucle principale sur j consiste à parcourir v en incrémentant j. On compare son terme courant $v[j]$ à $u[i]$ et s'ils sont égaux c'est que $u[i]$ fait partie d'une sous-séquence de v, on incrémente i dans ce cas. Il faut bien sûr que ces indices ne dépassent pas les longueurs de leurs séquences respectives. D'autre part, il faut toujours qu'il reste au moins autant de termes à parcourir dans v qu'il en reste dans u à comparer, sans quoi u ne peut plus être une sous-séquence de v

La complexité est simple à calculer, dans le meilleur des cas u est un préfixe de v et n comparaisons auront suffit à le décider, donc $\check{T}(n, m) = n$. Dans le pire des cas u est un suffixe de v, il aura donc fallu m comparaisons, soit $\hat{T}(n, m) = m$.

Algorithme EstSsSeq(u, v) : booléen
données
    u, v : mots
variables
    $i,\ j$: entiers

$i \leftarrow 1$
$j \leftarrow 1$
tantque $((i \leq \#u) \ \wedge \ (j \leq \#v) \ \wedge \ (\#u - \#v \leq i - j))$ **faire**
    si $(u[i] = v[j])$ **alors**
        $i \leftarrow i + 1$
    fsi
    $j \leftarrow j + 1$
ftq
renvoyer$(i > \#u)$

Algo 2. Test de sous-séquences