Université de Lille 1 SCIENCES ET TECHNOLOGIES

Info 204 - ASD

2013-2014
Licence STS - Semestre 4

DS1 - documents de cours, TD, TP autorisés

Note : lorsque, dans une question, est demandée la complexité, c'est une fonction de la taille de la donnée qui est attendue. Lorsque c'est le comportement asymptotique, la réponse attendue est soit une notation $\mathcal{O}$, soit une notation Θ soit une notation Ω.

Vous pourrez utiliser toute fonction vue en cours, TD, TP à la condition expresse de spécifier correctement les entrées et les sorties.

Exercice 1 : Questions de cours [3 points] (réponses sans justification)

Q 1.1 [0.75 point] Si je prouve qu'un algorithme est en $\mathcal{O}(n^2)$, peut-il être en $\Theta(n)$ sur certains exemplaires ? Répondre par oui ou non.

Q 1.2 [0.75 point] Si je prouve qu'un algorithme est en $\Omega(n \log n)$, peut-il être en $\mathcal{O}(n)$ sur certains exemplaires ? Répondre par oui ou non.

Q 1.3 [0.75 point] Le coût d'un ajout en queue dans une liste simplement chaînée contenant n éléments est (une seule réponse possible) :

(a) $\mathcal{O}(n^2)$ (b) $\mathcal{O}(n)$ (c) $\Theta(n)$ (d) $\Theta(1)$?

Q 1.4 [0.75 point] Le coût d'un ajout en queue dans une liste avec sentinelle contenant n éléments est (une seule réponse possible) :

(a) $\mathcal{O}(n)$ (b) $\mathcal{O}(1)$ (c) $\Theta(\log n)$ (d) $\Theta(1)$?

Exercice 2 : Multiplication de vecteurs de bits [6 points]

On souhaite réaliser la multiplication de deux nombres x et y représentés chacun sur n bits.

Une stratégie pour faire cela efficacement consiste à décomposer chacun des nombres en deux vecteurs de bits : ceux de poids fort et ceux de poids faible et d'utiliser uniquement les opérations de somme sur les entiers (`+` et `-`), le décalage binaire noté `<<` et le ou binaire noté `|`.

On notera x_L le vecteur de taille $\frac{n}{2}$ représentant les bits de poids fort de x et x_R le vecteur de taille $\frac{n}{2}$ représentant les bits de poids faible de x. Par exemple, si $x = \texttt{10011100}$ alors $x_L = \texttt{1001}$ et $x_R = \texttt{1100}$.

En utilisant la propriété suivante :

$$xy = (2^{n/2}x_L + x_R)(2^n/2y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

on aboutit au pseudo-code suivant :

```
/* fastmultiply : vecteur de bits, vecteur de bits -> vecteur de bits */
function fastmultiply(x, y)
  n = max(size of x, size of y)
  if n = 1 then
     return xy
  else
    xL, xR = leftmost n/2, rightmost n/2 bits of x
    yL, yR = leftmost n/2, rightmost n/2 bits of y
    P1 = fastmultiply(xL,yL)
    P2 = fastmultiply(xR,yR)
    P3 = fastmultiply(xL + xR,yL + yR)
    return P1 << 2^n | (P3 - P1 - P2) << 2^(n/2) | P2
```

Q 2.1 [1 point] Donner l'équation de récurrence donnant le nombre d'appels récursifs de cet algorithme.

Q 2.2 [1.5 points] En déduire le comportement asymptotique de l'algorithme `fastmultiply`.

On souhaite maintenant convertir un entier **décimal** 10^n (1 suivi de n 0, par exemple $10^3 = 1000$ soit mille) en binaire. On propose l'algorithme suivant pour $n = 2^p$:

```
/* pwr2bin : decimal -> vecteur de bits */
function pwr2bin(n)
  if n = 1 then
    return 1010 (* dix en binaire *)
  else
    z = ???
    return fastmultiply(z, z)
```

Q 2.3 [1 point] Quelle valeur doit-on assigner à z ?

Q 2.4 [1 point] Donner l'équation de récurrence donnant le nombre d'appels récursifs de cet algorithme.

Q 2.5 [1.5 points] En déduire le comportement asymptotique de l'algorithme `pwr2bin`.

Exercice 3 : Listes circulaires [3 points]

On considère des listes circulaires avec les définitions de type suivantes :

```
type liste = Vide | Cons of cellule
and cellule = {
  valeur : int;
  mutable suivant : liste;
  mutable precedent : liste;
}
```

On considère disposer d'une fonction `la_cellule` qui étant donnée une liste retourne la cellule correspondante, lève une exception `ListeVide` sinon.

Q 3.1 [0.5 point] Dessiner une représentation de la liste avec un seul élément.

Q 3.2 [0.5 point] Dessiner une représentation de la liste avec deux éléments.

Q 3.3 [2 points] Ecrire la fonction d'ajout en tête d'un élément e à une liste l (la liste l peut être vide).

Exercice 4 : CycleSort [8 points]

L'algorithme CycleSort a été proposé en 1990 pour trier des tableaux d'objets auxquels on pouvait associer à chacun une clé entière dans l'intervalle $[1..n]$. Le problème du tri de tableaux d'objets peut alors se réduire au tri de tableaux d'entiers : en réalisant les même opérations d'échange de valeurs sur le tableau d'objets que sur le tableau d'entiers correspondant, on aura trié les objets.

Si il existe une bijection entre les objets et l'intervalle $[1..n]$, on a alors à trier des tableaux particuliers que sont les permutations. On rappelle qu'une permutation de taille n est la suite des éléments de 1 à n dans n'importe quel ordre.

Ici, et afin de pouvoir manipuler les permutations dans des tableaux, on définira une permutation à n éléments comme la suite des éléments de 0 à $n-1$ dans n'importe quel ordre. Par exemple si $n = 10$: `[|2; 4; 7; 6; 3; 9; 8; 0; 5; 1|]` est une permutation.

CycleSort est basé sur la remarque que dans une permutation on peut détecter des cycles. Un cycle de taille m est une suite d'éléments $t.(i_1), t.(i_2), \ldots, t.(i_m)$ qui sont tels que $t.(i_1) = i_2, t.(i_2) = i_3, ..., t.(i_m) = i_1$. Par exemple dans la permutation ci-dessus $(2, 7, 0)$ est un cycle puisque $t.(0) = 2, t.(2) = 7, t.(7) = 0$.

Réorganisation d'un cycle Pour un cycle donné, on peut assez simplement remettre ses éléments à la bonne place dans la permutation triée : il suffit de faire « tourner » les éléments du cycle. Sur l'exemple on passera t.(0) = 2 dans t.(2) (il sera donc bien rangé), t.(2) = 7 dans t(7) et t.(7) = 0 dans t.(0).

Q 4.1 [1 point] Indiquer tous les cycles de la permutation `[|4; 7; 2; 1; 3; 8; 9; 0; 5; 6|]`.

Q 4.2 [2 points] Donner le code en CAML d'une procédure `reorganiser` qui étant donné un tableau t et un indice i réorganise le cycle qui passe par la position i. Par exemple :

```
# let t = [|2; 4; 7; 6; 3; 9; 8; 0; 5; 1|];;
# reorganiser t 0;;
# t;;
- : int array = [|0; 4; 2; 6; 3; 9; 8; 7; 5; 1|]
```

Q 4.3 [0.5 point] Si m est la taille d'un cycle, combien d'affectations de valeurs seront nécessaires pour réorganiser un cyle ? Justifier.

Q 4.4 [0.5 point] Combien y a-t-il de cycles au maximum dans une permutation de taille n ?

Q 4.5 [0.5 point] Combien y a-t-il de cycles au minimum dans une permutation de taille n ?

Processus de tri Une procédure de tri utilisant cette propriété pourrait être :

```
pour chaque cycle c faire
  reorganiser le cycle c
fin pour
```

Q 4.6 [1 point] Est-ce un tri sur place ? Justifier.

Q 4.7 [0.5 point] Indiquez dans votre code de la procédure reorganiser entre quelles lignes il faudra ajouter un incrément de compteur pour calculer le nombre d'affectations.

Q 4.8 [1 point] Quel est le meilleur des cas pour le nombre d'affectations. Expliquer.

Q 4.9 [1.5 point] En identifiant le pire des cas, donner le comportement asymtotique de l'algorithme pour le nombre d'affectations ?