Université Lille1 Sciences et Technologies

Info 204 - ASD

16 mai 2014
Licence STS - Semestre 4

DS2 - 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 : Parcours d'arbre [3 points]

Compétence évaluée : utliliser une structure de donnée arborescente.

Avec la définition de type suivante :

```
type arbre = Vide | Cons of noeud
and noeud = {
  valeur : int;
  fils   : arbre list; /* la liste ordonnee des fils */
}

let le_noeud a =
  match a with
   | Vide -> raise ArbreVide
   | Cons n -> n
```

Ecrire une fonction en CAML `plus_profonde : arbre -> int` qui retourne la valeur associée à la feuille de profondeur maximale (si il y a plusieurs feuilles à cette profondeur maximale, on retournera celle concernant la feuille la plus à droite).

Vous pouvez (devez) écrire des sous-fonctions.

Corrigé

Ici, on peut faire le choix de faire un parcours en profondeur (avec un efile) ou en largeur en utilisant la récursivité. La solution fournie ici est réalisée avec un parcours en profondeur. Remarquez qu'on a choisi de passer en paramètre, et par adresse, les variables où stocker le résultat. On aurait pu ici se contenter d'utiliser directement les variables `v` et `p` parce qu'on peut, en CAML, déclarer des sous-fonctions dans des fonctions.

```
let plus_profonde a =
  let p = ref (-1)
  and v = ref 0
  in
  let rec aux a pcur pmax vmax  =
    if a = Vide then
      raise ArbreVide;
    let n = le_noeud a
    in
    let l = ref n.fils
    in
    if !l = [] then begin
      if pcur >= !pmax then begin
        pmax := pcur;
        vmax := n.valeur;
      end;
    end else begin
        while !l <> [] do
        aux (List.hd !l) (pcur+1) pmax vmax;
        l := List.tl !l;
      done;
    end;
  in
  aux a 0 p v;
  !v
```

Exercice 2 : Structure de données[7 points]

Compétence évaluée : réinvestir les connaissances du cours pour répondre à un nouveau problème proche d'un problème connu.

Un utilisateur souhaite disposer d'une structure de donnée lui permettant :
— d'ajouter au fur et à mesure de nouveaux entiers à cet ensemble,
— d'être capable d'en extraire le minimum.
Un exemple d'utilisation pourrait être le suivant (`s` est la structure de données) :

```
# ajouter s 2;;
# ajouter s 5;;
# ajouter s 3;;
# minimum s;;
2
# ajouter s 1;;
# minimum s;;
1
# minimum s;;
3
```

On souhaite que les opérations d'ajout (procédure `ajouter`) et d'extraction (fonction `minimum`) s'exécutent en temps $\mathcal{O}(\log n)$ où n est le nombre d'élements contenus dans la structure au moment de l'ajout ou de l'extraction. On supposera qu'on n'ajoute pas deux fois le même entier.

Q 2.1 Expliquer la structure de données que vous allez utiliser.

Corrigé

On utilisera un AVL.

Q 2.2 La structure de données sera nommée `sd`. Donner la définition des types nécessaires pour définir `sd` en CAML.

Corrigé

Voir cours.

Q 2.3 Décrire en français ou pseudocode l'algorithme de la procédure d'ajout.

Corrigé

`t` est manipulé comme un AVL, se référer au cours.

Q 2.4 Décrire en français ou pseudocode l'algorithme de la fonction d'extraction du minimum.

Corrigé

Idem.

Q 2.5 Justifier de la complexité en temps en $\mathcal{O}(\log n)$ de la procédure d'ajout.

Corrigé

C'est un parcours sur une branche sur un arbre équilibré et donc de hauteur $\mathcal{O}(\log n)$.

Q 2.6 Justifier de la complexité en temps en $\mathcal{O}(\log n)$ de la fonction d'extraction.

Corrigé

Pour extraire : on enlève l'élément le plus en bas à gauche, en $\mathcal{O}(\log n)$, puis on rééquilibre.

Exercice 3 : Dérécursivation [5 points]

Compétence évaluée : réfléchir sur un problème pour proposer une solution algorithmique efficace.

On fait face à un problème de calcul modélisé par l'équation de récurrence suivante :

$$f(i,j) = \begin{cases} i+j & \text{si } i=0 \text{ ou } j=0 \\ \max_{0\leq k\leq l<\min(i,j)} f(k,l) & \text{sinon} \end{cases}$$

Q 3.1 Ecrire une fonction **récursive** en CAML permettant le calcul de f.

Corrigé

```
let rec f1 i j =
  if i = 0 || j = 0 then
    i + j
  else
    let m = ref 0
    in
    for l = 0 to (min i j) - 1 do
      for k = 0 to l do
        m := l + k + max !m (f1 k l)
      done;
    done;
    !m
```

Q 3.2 Cette fonction est-elle récursive terminale ?

Corrigé

Non puisque l'appel récursif n'est pas la dernière instruction réalisée.

En gardant une version récursive, on souhaite réaliser une modification de la fonction pour limiter le nombre d'appels récursifs tout en occupant un espace mémoire le plus faible possible. Pour répondre à ce problème :

Q 3.3 Expliquer comment réduire le nombre d'appels récursifs.

Corrigé

On remplace les appels récursifs par l'accès à une structure de donnée qui conserve les éléments déjà calculés.

Q 3.4 Expliquer comment conserver un espace mémoire faible.

Corrigé

On peut utiliser une table de hachage.

Q 3.5 Quelle est la complexité en espace de la nouvelle version ? Justifier en calculant exactement le nombre de valeurs calculées.

Corrigé

Pour répondre à cette question il faut évaluer le nombre exact de couples (k, l) pour lesquels il faut calculer la valeur de f.
k varie entre 0 et l. Par conséquent il faudra calculer $l + 1$ valeurs pour chaque l.
l varie entre 0 et $\min(i, j) - 1$, on aura donc

$$\sum_{0}^{\min(i,j)-1} l + 1 = \sum_{1}^{\min(i,j)} l = \frac{\min(i, j)(\min(i, j) + 1)}{2} = \Theta(\min(i, j)^2)$$

Par exemple, avec ce code :

```
let rec f2 i j =
  if i = 0 || j = 0 then
    i + j
  else
    let m = ref 0
    in
    for l = 0 to (min i j) - 1 do
      for k = 0 to l do
          try
            m := l + k + max !m (Hashtbl.find t (k,l));
          with
            | _ ->
              Hashtbl.add t (k,l) (f2 k l);
              m := l + k + max !m (Hashtbl.find t (k,l));
      done;
    done;
    !m
;;
f2 18 17;;
Hashtbl.stats t;;
```

On obtient :

```
# f2 18 17;;
- : int = 2449
# Hashtbl.stats t;;
- : Hashtbl.statistics =
{Hashtbl.num_bindings = 153; num_buckets = 128; max_bucket_length = 5;
 bucket_histogram = [|39; 45; 31; 7; 5; 1|]}
```

Soit $\frac{17\times18}{2} = 153$ valeurs calculées.
Si on avait utilisé un tableau, sa taille aurait été de $(\min(i, j) + 1)^2$ ce qui est moins efficace.

Exercice 4 : Programmation dynamique [5 points]

Compétence évaluée : appliquer les méthodologies apprises en cours.

On s'intéresse ici au problème du calcul de la longueur de la plus longue sous-séquence croissante d'un tableau d'entiers définie comme étant le plus grand nombre d'entiers $x_1, \ldots, x_m$ du tableau tels que :

— $x_i \leq x_{i+1} \quad \forall i \in 1..m - 1$ et
— $p(x_i) < p(x_{i+1}) \quad \forall i \in 1..m - 1$ où $p(x_i)$ est l'indice de x_i dans le tableau.

Par exemple, avec le tableau `[|3;1;2;6;5;7;9;8|]`, la longueur de la plus longue sous-séquence croissante est 5 (il y a plusieurs sous-séquences correspondantes : 1,2,6,7,9 ; 1,2,6,7,8 ; 1,2,5,7,9 ; 1,2,5,7,8 ; 1,2,3,7,9 ; 1,2,3,7,8).

Si on note $L(i)$ la longueur de la plus longue sous-séquence croissante incluant l'entier en position i dans le tableau, alors le problème s'exprime récursivement ainsi :

$$L(i) = \begin{cases} 1 + \max_{0 \leq k < i} L(k) & \text{si il existe } k \text{ tel que } t.(k) \leq t.(i) \\ 1 & \text{sinon} \end{cases}$$

Pour trouver la solution au problème initial, il suffit de rechercher le maximum parmi les L. On supposera disposer du tableau t.

Q 4.1 Donner les valeurs de $L(0), L(1), L(2)$ pour l'exemple.

Q 4.2 Donner les instructions CAML permettant de créer (pas remplir) la table de programmation dynamique permettant le calcul de L.

Corrigé

```
let plsc t =
  let n = Array.length t
  in
  let table = Array.make n 0
  and r = ref 0;
  in
```

Q 4.3 Donner les instructions CAML permettant d'initialiser la table L.

Corrigé

```
  (* initialisation *)
  for i = 0 to n-1 do
    table.(i) <- 1;
  done;
```

Q 4.4 Donner les instructions CAML permettant de remplir la table L.

Corrigé

```
  (* remplissage *)
  for i = 0 to n-1 do
    for k = 0 to i-1 do
      if t.(k) <= t.(i) then
        table.(i) <- max table.(i) (1 + table.(k));
    done;
  done;
```

Q 4.5 Donner les instructions CAML permettant d'obtenir le résultat.

Corrigé

```
  (* solution *)
  for i = 0 to n-1 do
    r := max !r table.(i)
  done;
  !r
```

Q 4.6 Quelle est la complexité en espace ? Justifier.

Corrigé

La table est de taille n (taille du tableau).

Q 4.7 Quelle est la complexité en temps ? Justifier.

Corrigé

On compte les appels à `max`.
Dans le pire des cas (tableau croissant) :

$$\sum_{i=0}^{n-1}\sum_{k=0}^{i} 1 + \sum_{i=0}^{n-1} 1 = \sum_{i=0}^{n-1}(i+1) + n = \frac{n(n+1)}{2} + n$$

Dans le meilleur des cas (tableau décroissant) :

$$\sum_{i=0}^{n-1}\sum_{k=0}^{i} 0 + \sum_{i=0}^{n-1} 1 = n$$

L'algorithme est donc en $\mathcal{O}(n^2)$ et en $\Omega(n)$.