Exercices pour s'entraîner

Exercice 1 :

On suppose que "&" est l'opérateur de concaténation de tableaux. Soit la fonction suivante :

```
type TABLEAU = array of CARDINAL;

function x (n : CARDINAL) : TABLEAU;
var
  t : TABLEAU (1..1);
begin
  if (n = 0) or (n = 1) then begin
    t[1] := n;
    x := t;
  end else begin
    if n mod 2 /= 0 then begin
      t[1] := 1;
    end {if};
    x := x(n div 2) & t[1];
  end {if};
end {x};
```

Q 1 . Que fait la fonction `x` ?

Q 2 . Quelle est la complexité en espace de `x` en fonction de `n` ? Justifier.

Exercice 2 : La fonction inconnue

Soit la déclaration suivante :

```
type TABLEAU = array of CARDINAL;
```

et la fonction suivante :

```
procedure inconnue (t : var TABLEAU; q : var CARDINAL);
begin
   q := low(t) - 1;
   for j in low(t) to high(t) do begin
      if t[j] <= t[high(t)] then begin
         q := q + 1;
         echanger(t,q,j);
      end {if};
   end {loop};
   if q < high(t) then begin
      q := q + 1;
   end {if};
end {inconnue};
```

Q 1 . Déroulez la fonction `inconnue` à la main sur le tableau `2 2 6 4 3 2 6 1 5 3`.

Q 2 . Que fait la fonction `inconnue`, que représente la variable `q` ?

Q 3 . Donnez la complexité en temps de la fonction `inconnue` en fonction de la taille du tableau `t`.

Exercice 3 : Mais comment faire ?

Décrire un algorithme en $\mathcal{O}(n \log n)$ qui, à partir d'un ensemble S de nombres réels et d'un autre nombre réel x, détermine si il existe deux nombres réels de S (pas nécessairement différents) tels que leur somme soit égale à x.

Exercice 4 : Récursivité

Soit la fonction f définie récursivement pour i et j strictement positifs ainsi :

$$
\begin{array}{lcll}
f(i,j) & = & 0 & j > i+1 \\
f(1,j) & = & 1 & 1 \leq j \leq i \\
f(i,1) & = & 1 & i \geq 1 \\
f(i,i+1) & = & 1 & i \geq 1 \\
f(i,j) & = & f(i-1,j) + f(i-1,j-1) & \text{sinon}
\end{array}
$$

Q 1 . Donner sans le justifier les valeurs et le nombre d'additions et d'appels à f nécessaires au calcul de $f(5,j)$ pour tout j dans 1..6.

On s'intéresse maintenant au calcul de $\sum_{j=1}^{i+1} f(i,j)$.

Q 2 . En utilisant un espace mémoire en $\mathcal{O}(i)$, écrire une fonction non récursive capable de calculer la somme $\sum_{j=1}^{i+1} f(i,j)$. Quelle est la complexité en temps de votre algorithme ?

Q 3 . Est-ce mieux que si l'on avait calculé f grâce à sa définition récursive ?

Exercice 5 :

On souhaite effectuer un tri de matrice de taille $n \times m$. On veut que tous les éléments soient rangés, du plus petit au plus grand et d'en haut à gauche à en bas à droite. Il faut donc vérifier la propriété suivante:

$$case(i,j) \leq case(i+1,j) \text{ et } case(n,j) \leq case(1,j+1)$$

Par exemple : $\begin{array}{cccc} 1 & 3 & 5 & 7 \\ 9 & 2 & 1 & 3 \\ 1 & 5 & 2 & 2 \\ 4 & 5 & 6 & 8 \end{array}$ donnera $\begin{array}{cccc} 1 & 1 & 1 & 2 \\ 2 & 2 & 3 & 3 \\ 4 & 5 & 5 & 5 \\ 6 & 7 & 8 & 9 \end{array}$

Q 1 . Décrivez (en français) une méthode permettant de faire le tri d'une matrice de taille $n \times m$ dont la complexité en espace est $\Theta(nm)$ et la complexité en temps $\mathcal{O}(nm \log nm)$.

Q 2 . Sachant maintenant que tous les nombres dans la matrice sont inférieurs ou égaux à un certain nombre M, donnez un algorithme dont la complexité en espace est $\Theta(M)$ et la complexité en temps $\mathcal{O}(nm)$. Justifiez les résultats de complexité.

Exercice 6 :

Soit la fonction suivante :

```
function koic (n : CARDINAL) : CARDINAL;
var
   i : CARDINAL
   t : array(1..n div 2+2) of CARDINAL;
begin
   for i := low(t) to high(t) do
     t[i] := 1;
   for i in 2..n do begin
      for j in reverse 2..i/2+1 do begin
         t[j] := t[j-1]+t[j];
      end {loop};
      if i mod 2 = 1 then begin
         t[i/2+2] := t[i/2+1];
      end {if};
      write(i);  write(':');
      for j := low(t) to high(t) do begin
         write(t[j]);
      end {loop};
      writeln;
   end {loop};
   koic := t[n div 2+1];
end {koic};
```

Q 1 . Donner ce qu'imprime à l'écran un appel à la fonction `koic` avec `n=4` et `n=5`. Quel est le résultat retourné par la fonction ? Que calcule la fonction `koic` ?

Q 2 . Quelle est la complexité en espace de `koic` en fonction de `n` ?

Q 3 . Quelle est la complexité en temps de `koic` en fonction de `n` ? Justifiez.

<u>Exercice 7</u> :

Un arbre binaire est un $F-$arbre si en chaque noeud, les hauteurs des sous-arbres gauche et droit diffèrent au plus de 2 et que chaque noeud possède exactement 0 ou 2 fils. Pour simplifier un peu, nous supposerons que le sous-arbre gauche n'est jamais plus haut que le sous-arbre droit correspondant.

Le but de l'exercice est d'évaluer les nombres minimum ($\min(h)$) et maximum ($\max(h)$) de noeuds dans un $F-$arbre de hauteur h donnée.

Q 1 . Dessiner les $F-$arbres de hauteur 0, 1 et 2 comportant un nombre maximal de noeuds. Exprimer $\max(h)$ en fonction de h.

On appelle F_h le $F-$arbre de hauteur h le plus creux (possèdant un nombre minimal de noeuds).

Q 2 . Dessiner F_1, F_2, F_3, F_4.

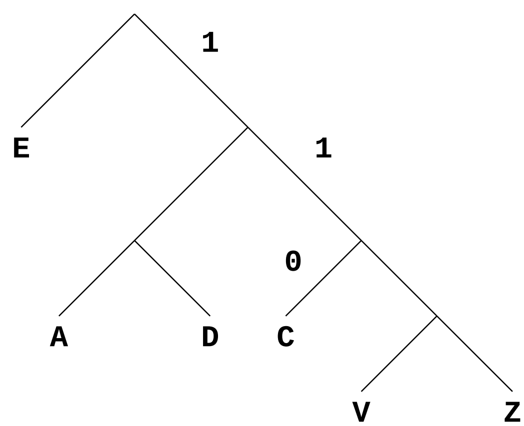
Q 3 . Montrer comment on peut construire F_h à l'aide de certains F_i avec $i < h$.

Q 4 . En déduire une expression de $\min(h)$ en fonction de certains $\min(i)$ avec $i < h$.

<u>Exercice 8</u> :

Il est possible d'associer à chaque arbre binaire un chemin. La procédure est simple, en partant de la racine on crée une chaîne de caractères à laquelle on concatène un 0 si on emprunte le fils gauche et un 1 si on emprunte le fils droit. On obtient ainsi un code associé à chaque feuille de l'arbre.

Nous nous intéressons à des arbres dont les feuilles contiennent une lettre et aucune feuille ne contient la même lettre. En suivant le chemin menant de la racine à une feuille et la procédure de codage décrite, on obtient le code associé à cette feuille. Sur l'exemple suivant, la lettre `C` aura pour code `110` :



Q 1 . Donnez le code associé à chaque feuille de l'arbre ci-dessus.

Il est facile de voir qu'il existe une bijection entre un code et un caractère. Cela signifie qu'à partir d'une suite de 0 et de 1 on obtient une et une seule lettre (et réciproquement). Une chaîne de caractères peut donc être codée de manière unique en associant à chaque lettre son code et en concaténant les codes.

Q 2 . Donnez le mot associé au codage `101100101100`.

Q 3 . Donnez le code associé au mot `EVADE`.

Q 4 . Ecrire une fonction `function decode (s : CODE, a : ARBRE) : STRING;` qui transforme la chaîne `s` formée de `0` et de `1` en le mot associé suivant les codes donnés par l'arbre `a`.

Q 5 . Quelle est la complexité en temps de cette fonction en fonction de la longueur de `s` ?

Q 6 . Proposez une méthode la plus efficace possible en temps (sans écrire le code) pour coder une chaîne de caractères `s` en une suite de `0` et de `1` en utilisant un arbre `a` qui ne contient que les lettres du mot à coder. Quelle est sa complexité en temps ?

On définit le code le plus court d'une feuille d'un arbre comme celui de longueur la plus courte.

Q 7 . Ecrire une fonction `function le_plus_court (a : ARBRE) : CHAR;` qui retourne la lettre dont le code est le plus court et qui se trouve la plus à gauche dans l'arbre.

Q 8 . Quelle est la complexité en temps de cette fonction en fonction de la taille de `a` ?

Q 9 . Ecrire une fonction `function les_plus_courts (a : ARBRE) : STRING;` qui retourne la chaîne composée d'une occurrence de chaque lettre possédant un code de même longueur et le plus court.

Q 10 . Si l'arbre donné est un arbre équilibré à n feuilles, quelle est la longueur maximale du code d'une lettre.