Université Paris Cité

L2 informatique - Année 2022–2023

Contole d'Éléments d'Algorithmique n° 2 - Groupe 5 (Correction)

Exercice 1. *Suite Récurrente.*
On considère la suite définie par :

$$a_0 = 2;$$
$$a_n = 3 * a_{n-1} + 4 \qquad (n \geq 1).$$

1. Écrivez une fonction récursive non terminale (en JAVA ou en pseudo-code) qui calcule a_n en appliquant directement la définition ci-dessus.
2. Détaillez les états de la pile lorsqu'on lance votre algorithme avec $n = 4$.
3. Écrivez une fonction récursive terminale qui calcule a_n.
4. Montrez que pour tout n , $a_n = 4 * 3^n - 2$.

Correction :

1.
```
Fonction Suite ( n: entier) : entier
   Si (n=0)  alors retourner (2)
             sinon retourner ( 3*Suite( n-1) +4)
```

2. La pile s'exécution contient Suite (4) puis Suite (3) est empilé puis Suite (2) puis Suite (1) puis Suite(0).
   Suite (0) retourne 2 àSuite (1) qui est alors dépilé et fait le calcul 3*2+4 = 10 qui est retourné à Suite (2)
   Suite (2) calcule 3*10+4 = 34 et retourneà Suite (3)
   Suite (3) calcule 3*34+4 = 106 et retourne à Suite (4)
   Suite (4) calcule 3*106+4 = 322 et retourne au programme appelant
3.
```
Suite_ter( n) : entier
   retourne Suite_aux(n,2)

Suite_aux( n, a) :entier
  si n=0 alors retourne a
            sinon retour Suite\_aux( n-1, 3a+4)
```

4. On montre que $a_n = 4 * 3^n - 2$ par récurrence.
   Cas de base : n=0 par définition a_0=2 et par la formule $a_0 = 2$
   Induction : on suppose que $a_n = 4 * 3^n - 2$. Par definition $a_{n+1} = 3 * a_n + 4$, par hypothése de récurrence $a_n = 4 * 3^n - 2$. Donc $a_{n+1} = 4 * 3^{n+1} - 2$. a_{n+1} satisfait bien la formule donné.

Exercice 2. *Fonction mystère sur les listes.*

1. Déroulez l'algorithme sur la liste chaînée suivante. A chaque affectation d'une des variables, vous montrerez où pointe chaque élément.

$$lis \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow \texttt{null}$$

Algorithm 1 Algorithme 1

Entrée : Liste chaînée contenant des entiers triés `lis`

1: **fonction** MYSTERE(lis)
2: $tmp \leftarrow lis.head$
3: **tant que** $tmp.next \neq$ `null` & $tmp.key < tmp.next.key - 1$ **faire**
4: $c \leftarrow new\ Cellule(tmp.key + 1, tmp.next)$
5: $tmp.next \leftarrow c$
6: $tmp \leftarrow tmp.next$
retourne lis

2. Que fait cet algorithme ?
3. Transformer cet algorithme en algorithme récursif. Vous n'avez pas le droit d'utiliser de While.

Correction : Si la liste a au moins deux élélements x et y, la fonction rajoute entre x et y les éléments x+1, x+2,...y-1.

```
Fonction Mystere_rec(lis) : liste.
  tmp:=lis.head;
  si tmp.next =nil alors return lis
  si tmp.key= tmp.next.key+1 alors retourne lis
  si tmp.key>= tmp.next.key+2 alors
                              c:= new Cellule (tmp.key.next-1,tmp.next); tmp.next:=c ;
                              retourne(Mystere_rec(lis));
```

Exercice 3. *Changement de parité.*
On a un tableau de n entiers où les nombres pairs se trouvent avant les nombres impairs. La case pivot de ce tableau correspond à la première position où se trouve une valeur impaire (s'il n'y a que des entiers pairs, la case pivot correspond à n).

1. Proposez un algorithme linéaire qui renvoie l'indice de la case pivot.
2. Proposez un algorithme en temps logarithmique qui renvoie l'indice de la case pivot.

Correction :

1.
```
Pivot(tableau(T) de taille n): entier
 entier i=0
 tant que i<n faire
       si T[i] est paire alors i++ sion retouner (i)
 retourne (n).
```

2.
```
public  static int PariteLog( int[] T){
  int deb=0;int fin=T.length-1; int m;
  if (T[fin]%2==0) return ( fin);
  if (T[deb]%2==1) return ( deb);
   //le tableau commence par un entier pair et termine par un entier impair
   while ((fin-deb)>1) {
       m=deb+(fin-deb)/2;
       if (T[m]%2==0) deb=m;
           else  fin = m;}
      return(fin);
 }
```