université PARIS-SACLAY

Correction interro n°2

Cours d'algorithmique

—Licence CFA / Troisième Année—

Les documents manuscrits, sujets de travaux pratiques et dirigés ainsi que les supports de cours sont autorisés. Tous les autres documents tels que livres, calculatrices, téléphones portables et ordinateurs sont interdits.

Les exercices sont indépendants. Si l'on ne sait pas justifier une question, on peut néanmoins utiliser la réponse dans la suite.

Le barème est indicatif et pourra changer à la correction.
Durée : 45 min

▸ **Exercice 1.** – 2 points.

Écrire une fonction récursive qui prend en paramètre un tableau de nombres de taille `N` et qui renvoie le maximum des nombres du tableau. Quelle en est la complexité ?

✂ ----------

Version courte :

```
float max_rec(float T[], int N) {
  if (N == 0) erreur('tableau vide');
  if (N == 1) return T[0];
  return max(T[N-1], max_rec(T, N-1));
}
```

Version récursive terminale :

```
float max_rec(float m, float T[], int N, int i) {
  // retourne le maximum de m et du contenu du tableau
  // a partir de la position i
  if (i == N) return m;
  return max_rec(max(m, T[i]), T, N, i+1);
}

float max_tab(float T[], int N) {
  if (N == 0) erreur('tableau vide');
  return max_rec(T[0], T, N, 1);
}
```

Les instructions de `max_rec` sont toutes en temps constant $O(1)$, sauf l'appel récursif. La fonction `max_rec` sera appelée N fois. On a donc une complexité en $O(N)$.

---------- ✂

▸ **Exercice 2. (Remplissage bit-retourné d'un tableau)** – 3 points.

On considère le code Java suivant :

```
1   public static void bitrev_rec(int T[], int min, int max, int n, int pow2) {
2     int mid;
3     if (min == max) {
4       T[min] = n;
5     } else {
6       mid = (min+max)/2;
7       bitrev_rec(T, mid+1, max, n, 2*pow2);
8       bitrev_rec(T, min, mid, n+pow2, 2*pow2);
9     }
10  }
11
12  public static void bitrev(int T[]) {
13    bitrev_rec(T, 0, T.length-1, 0, 1);
14  }
```

1. Quel est, après l'execution de

   ```
   int [] tab = new int[8];
   bitrev(tab);
   ```

   le contenu du tableau `tab` ? On donnera le détail de la suite des appels de la fonction `bitrev_rec`.

   Indication : le tableau contient les nombres de 0 à 7 dans un certain ordre.

   ✂ ----------

   Voici une trace de l'exécution de `bitrev`

   ```
   Appel de bitrev avec min=0, max=7, n=0, pow2=1
   Appel de bitrev avec min=4, max=7, n=0, pow2=2
   Appel de bitrev avec min=6, max=7, n=0, pow2=4
   Appel de bitrev avec min=7, max=7, n=0, pow2=8
   Affectation T[7] <- 0
   retour de bitrev
   Appel de bitrev avec min=6, max=6, n=4, pow2=8
   Affectation T[6] <- 4
   retour de bitrev
   retour de bitrev
   Appel de bitrev avec min=4, max=5, n=2, pow2=4
   Appel de bitrev avec min=5, max=5, n=2, pow2=8
   Affectation T[5] <- 2
   retour de bitrev
   Appel de bitrev avec min=4, max=4, n=6, pow2=8
   Affectation T[4] <- 6
   retour de bitrev
   retour de bitrev
   retour de bitrev
   Appel de bitrev avec min=0, max=3, n=1, pow2=2
   ```

```
Appel de bitrev avec min=2, max=3, n=1, pow2=4
Appel de bitrev avec min=3, max=3, n=1, pow2=8
Affectation T[3] <- 1
retour de bitrev
Appel de bitrev avec min=2, max=2, n=5, pow2=8
Affectation T[2] <- 5
retour de bitrev
retour de bitrev
Appel de bitrev avec min=0, max=1, n=3, pow2=4
Appel de bitrev avec min=1, max=1, n=3, pow2=8
Affectation T[1] <- 3
retour de bitrev
Appel de bitrev avec min=0, max=0, n=7, pow2=8
Affectation T[0] <- 7
retour de bitrev
retour de bitrev
retour de bitrev
retour de bitrev
7 3 5 1 6 2 4 0
```

On obtient donc `[7 3 5 1 6 2 4 0]`;

▸ **Exercice 3. (Chirurgie dans les listes)** – 5 points.

On rappelle les déclarations du type `LinkedList` vues en cours, où `[...]` signale qu'une partie du code a été coupée.

```
1   public class LinkedList<Elem> {
2       static class RefCell<Elem> {
3           public Cell<Elem> next;
4           public RefCell() [...];
5           public RefCell(Cell<Elem> cell) [...];
6       }
7       static class Cell<Elem> extends RefCell<Elem> {
8           public Elem value;
9           public Cell(Elem value, Cell<Elem> next) [...]
10          public Cell(Elem value) [...]
11      }
12      private RefCell<Elem> begin;
13      public LinkedList() { begin = new RefCell<Elem>(); }
14  }
```

Dans les questions suivantes, on demande

— de travailler en place, sans allouer ni désallouer de cellule,

— d'avoir un algorithme dont la complexité est $O(\text{Long}(\texttt{L}))$. On justifiera brièvement cette complexité.

1. Écrire une méthode `rev` qui inverse, en place, la liste chaînée. Par exemple, si avant l'appel de méthode la liste vaut $\langle 1, 5, 4, 6\rangle$, on doit avoir après l'appel $\langle 6, 4, 5, 1\rangle$. Quelle est la complexité de cet algorithme ? Justifier brièvement votre réponse.

✂ ----------

```
void LinkedList::rev() {
  Cell todo = this.begin.next;
  this.begin.next = null;
  while (todo != null) {
    Cell tmp = todo;
    todo = todo.next;
    tmp.next = this.begin.next;   // insere tmp
    this.begin.next = tmp;        // en tête de this
  }
}
```

Variante : on suppose écrite les deux méthodes suivantes :
— `LL.insert_head_cell(c)` insère la cellule en tête
— `LL.remove_head cell(c)` supprime la cellule de tête et la retourne
Note : on a écrit des méthodes similaires en TD mais où l'on passait en paramètre une valeur et non pas une cellule. Il suffit de supprimer la commande qui alloue la cellule.

```
void LinkedList::rev() {
  // transfert les cellules de this dans une LL tmp en gardant l'ordre
  LinkedList tmp = new LinkedList()
  tmp.begin.next = this.begin.next;
  this.begin.next = null;
  // transfert les cellules de tmp dans this une à une
  while (not tmp.is_empty()) {
     this.insert_head_cell(tmp.remove_head cell());
  }
}
```

---------- ✂

2. Écrire un algorithme qui, étant donnée une liste chaînée, ne garde dans la liste que les éléments positifs. Les éléments coupés seront retournés dans une nouvelle liste en conservant leur ordre.

   Par exemple pour $\langle 2, -1, -3, 6, 7, -2, 1\rangle$ on ne gardera que $\langle 2, 6, 7, 1\rangle$ et l'on retournera $\langle -1, -3, -2\rangle$.

✂ ----------

```
LinkedList LinkedList::garde_positif() {
  LinkedList res = new LinkedList();
  RefCell curthis = this.begin;
  refCell endres = res.begin; // pointeur sur la fin de res;
  while (curthis.next != null) {
    if (curthis.next.val < 0) {
      endres.next = curthis.next;       // place la cellule après endres
```

```
        endres = endres.next;              // avance endres
        curthis.next = curthis.next.next; // supprime la cellule de curthis
      }
      else {
        curthis = curthis.next;           // avance curthis
      }
    }
    endres.next = null;
    return res;
  }
```