Examen 2 de Programmation en Java

Christian Rinderknecht

Mardi 29 Juin 2004

Durée : deux heures. Les documents et les calculatrices ne sont pas autorisés.

1 Questionnaire à choix multiples

Attention : il peut y avoir plusieurs bonnes réponses par question, par conséquent une mauvaise réponse entraine une fraction de point négative. Néanmoins, ces points négatifs ne sont pas comptabilisés hors de cette partie.

1. Quelle est la valeur de `values[2][1]` dans le tableau suivant ?

   ```
   double[][] values =
     {{1.2,9.0,3.2},{9.2,0.5,1.5,-1.2},{7.3,7.9,4.8}}
   ```

   ☐ `7.3`
   ☐ `7.9`
   ☐ `9.2`
   ☐ Il n'y a pas d'élément correspondant à ces indices.

2. Quelle est la valeur retournée par `things.length` ?

   ```
   double[][] things = {{1.2,9.0},{9.2,0.5,0.0},{7.3,7.9,1.2,3.9}}
   ```

   ☐ `2`
   ☐ `3`
   ☐ `4`
   ☐ `9`

3. Étant donné la déclaration de variable suivante : `long[][] stuff;` Quelles instructions ci-dessous construisent un tableau de 5 lignes et 7 colonnes et l'affecte à `stuff` ?

   ☐ `stuff = new stuff[5][7]`
   ☐ `stuff = new long[5][7]`
   ☐ `stuff = long[5][7]`
   ☐ `stuff = long[7][5]`

4. Étant donnée la déclaration

```
int[][] items = {{0,1,3,4},{4,4,99,0,7},{3,2}};
```

Quelles boucles affichent tous les éléments de `items` ?

☐
```
int row=0; int col = 0;
for (row=0; row < items.length; row++) {
  System.out.println();
  for (col=0; col < items.length; col++)
    System.out.print (items[row][col] + " ");
}
```

☐
```
int row=0; int col = 0;
for (row=0; row < items.length; row++) {
  System.out.println();
  for (col=0;col < items[col].length; col++)
    System.out.print (items[row][col] + " ");
}
```

☐
```
int row=0; int col = 0;
for (row=0; row < items.length; row++) {
  System.out.println();
  for (col=0; col < items[row].length; col++)
    System.out.print (items[row][col] + " ");
}
```

☐
```
int row=0; int col = 0;
for (row=0; row < items.length; row++) {
  for (row=0; row < items[row].length; row++)
    System.out.print (items[row][col] + " ");
  System.out.println();
}
```

5. Soit la déclaration

```
int[][] items = {{0,1,3,4},{4,3,99,0,7},{3,2}}
```

Quelles instructions remplacent entièrement les valeurs de la première ligne de `items` ?

☐
```
items[0][0] = 8;
items[0][1] = 12;
items[0][2] = 6;
```

☐
```
items[0] = new {8,12,6};
```

☐
```
int[] temp = {8,12,6};
items[0] = temp;
```

☐
```
items[0] = {8,12,6};
```

6. Qu'affiche l'extrait de programme suivant ?

```
int[] tab = {1,4,3,6,8,2,5};
int what = tab[0];
for (int index=0; index < tab.length; index++) {
  if (tab[index] < what) what = tab[index];
}
System.out.println (what);
```

☐ `1`
☐ `5`
☐ `1 4 3 6 8 2 5`
☐ `8`

2 Nombres de Armstrong

Un nombre de Armstrong est un nombre entier positif égal à la somme des cubes des chiffres qui le composent en base dix. Par exemple :

$$153 = 1^3 + 5^3 + 3^3$$

Écrivez un programme Java nommé Armstrong qui affiche tous les nombres de Armstrong inférieurs à 1000.

3 Recherche par dichotomie

Supposons que nous disposions d'un tableau t d'entiers triés par ordre croissant ainsi que d'un entier n. Nous voulons déterminer efficacement si n est dans t. Une façon inefficace consiste à parcourir tout le tableau et comparer n à chaque élément. Dans le pire des cas il faut donc parcourir tout le tableau, ce qui rend le temps de recherche proportionnel à la taille de t. Une façon efficace consiste à comparer n avec l'élément au milieu du tableau. Si n est égal à cet élément, alors on conclut ; s'il est strictement plus grand alors on relance la recherche dans la partie supérieure, sinon on relance dans la partie inférieure. Si on doit chercher dans un sous-tableau vide, alors on conclut que n est absent du tableau. C'est la *recherche par dichotomie*. Dans le pire des cas on montre que le temps de recherche est proportionnel au logarithme de la taille du tableau (ce qui est inférieur à la recherche linéaire naïve décrite précédemment).

Complétez la méthode `lookup` dans le programme suivant pour quelle effectue la recherche par dichotomie. Un sous-tableau de `t` est modélisé par la donnée de `t` lui-même ainsi que de deux indices dans `t` correspondant à la borne inférieure et supérieure du sous-tableau. Ainsi le premier paramètre de `lookup`, `n`, est l'entier recherché ; le second, `t`, est le tableau initial dans lequel on le recherche ; le troisième, `low`, est l'indice dans `t` de la borne inférieure du sous-tableau ; le quatrième, `high`, est la borne supérieure du sous-tableau. Par exemple `lookup (5,t,1,4)` signifie qu'on recherche `5` dans le sous-tableau compris entre les indices `1` et `4` du tableau `t`. Le type de retour est de type `boolean` : `true` si `n` est dans le sous-tableau de `t`, `false` sinon.

Un tableau (ou sous-tableau) vide est caractérisé par une borne inférieure qui est strictement supérieure à sa borne supérieure. Pour qu'un tableau soit un sous-tableau d'un autre il faut vérifier quelques contraintes sur ses bornes inférieures et supérieures. L'indice milieu du tableau sera `(low + high)/2`.

La méthode `lookup` devra être programmée sans boucles. Elle peut être écrite en cinq ou sept lignes seulement !

```
public class Dicho {
    public static boolean lookup (int n, int[] t, int low, int high) {









    }

    public static void print (int[] t) {
        int i = 0;
        for (i = 0; i < t.length; i++)
          System.out.print (t[i] + " ");
        System.out.println ();
    }

    public static void main (String[] args) {
        if (args.length == 1) {
          int n = Integer.parseInt (args[0]);
          int[] t = {3,7,8,13,15,100};
          print (t);
          if (lookup (n,t,0,t.length-1))
            System.out.println ("Présent.");
          else System.out.println ("Absent.");
        }
        else System.out.println ("usage: java Dicho <num>");
    }
}
```