Devoir maison TO2

(à rendre le 2 mai 2011)

Exercice 1 (Tri de crêpes)
Un cuisinier se trouve en face d'une pile de n crêpes, dont les diamètres varient (Figure 1 à gauche). Il tient à la main une spatule qu'il peut passer n'importe où dans sa pile de crêpes pour retourner tout le haut de sa pile (Figure 1 au milieu). Il aimerait trier sa pile pour voir toutes les crêpes rangées par diamètre croissant de haut en bas (Figure 1 à droite).

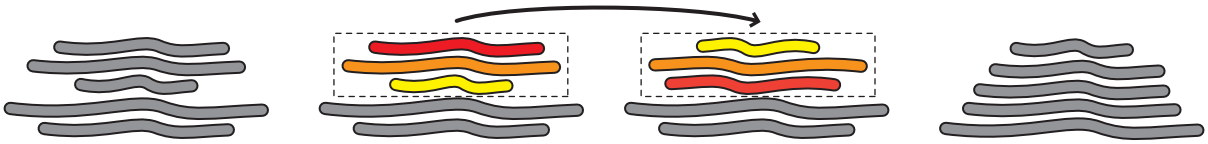

FIGURE 1 – Le retournement dans une pile de crêpes.

Sa stratégie nécessite n étapes : à la $p$ième étape, il replace la $p$ième plus grosse crêpe au $p$ième étage de la pile. Pour cela, il doit faire deux retournements :

(i) d'abord, il passe sa spatule en dessous de la $p$ième plus grosse crêpe et la fait passer tout en haut en retournant le haut de la pile,

(ii) ensuite, il passe sa spatule en dessous de la crêpe à l'étage p et retourne le haut de la pile : la $p$ième plus grosse crêpe se retrouve alors à l'étage p.

Sur la Figure 2, on a représenté la 2ième étape pour trier la pile de la Figure 1.

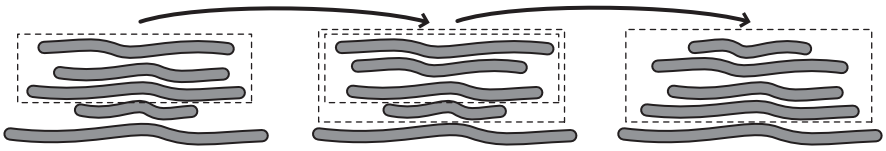

FIGURE 2 – La 2ième étape de la stratégie du cuisinier.

1. Écrire une classe `PileCrepes` avec :
   - un argument `private int[] diametres` représentant les diamètres des crêpes dans la pile (la crêpe du bas est représentée par `diametres[0]`),
   - un constructeur `public PileCrepes(int[] diametres)` qui construit une pile de crêpes en partant d'un tableau de diamètres,
   - une méthode `public void spatule(int position)` qui tourne tout le haut de la pile au dessus de position (échanger `diametres[position]` avec `diametres[diametres.length-1]`, puis `diametres[position+1]` avec `diametres[diametres.length-2]`, etc.),
   - une méthode `public int positionPlusGrosse(int etage)` qui renvoie la position de la plus grosse crêpe située au dessus de `etage`.
2. Écrire une méthode `cuisinier` qui trie la pile de crêpes courante avec la stratégie du cuisinier.
3. Décrire l'invariant qui assure que la méthode du cuisinier termine et trie sa pile de crêpes. Montrer que l'invariant est préservé lors du tri.
4. Quel est le nombre de retournements nécessaires au cuisinier dans le pire des cas ?

Exercice 2 (Arbres)
Un arbre est constitué d'une racine étiquetée par un entier, et d'un certain nombre (éventuellement zéro) de branches qui sont elles-même des arbres. Un exemple est présenté sur la figure 3. Pour représenter un arbre, on utilise donc la classe récursive suivante :

```
public class Arbre {
    public int     racine;    // étiquette de la racine
    public Arbre[] branches;  // branches de l'arbre
    ...
}
```

1. Écrire un constructeur `Arbre(int racine, Arbre[] branches)` qui initialise les attributs de l'arbre avec les arguments passés au constructeur. Pour l'instant, on ne veut pas copier quoi que ce soit (ni le tableau, ni les objets qui sont à l'intérieur).
2. Écrire une méthode `void parcours()` qui parcourt l'arbre courant en affichant les étiquettes de chacun de ses noeuds. Par exemple, on doit obtenir `3 1 4 3 2 5 4 7 7 6 1 5` pour l'arbre de la figure 3.
3. Écrire les méthodes `int nbNoeuds()` et `int nbFeuilles()` qui renvoient respectivement le nombre de noeuds et le nombre de feuilles (les noeuds qui n'ont pas de descendant).
4. Écrire une méthode `boolean egale(Arbre a)` qui teste si l'arbre `a` est égal à l'arbre courant. Écrire une méthode `boolean contient(Arbre a)` qui teste si l'arbre `a` est un sous-arbre de l'arbre courant.
5. Écrire une méthode `Arbre copie()` qui renvoie une copie en profondeur de l'arbre courant.

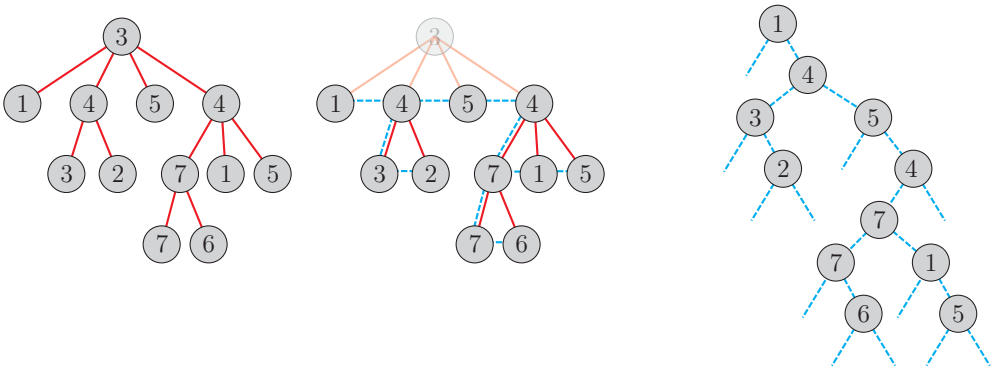

FIGURE 3 – Un arbre à 11 noeuds et 7 feuilles, et sa transformation en un arbre binaire.

On s'intéresse maintenant aux arbres binaires, dont tous les noeuds ont zéro ou deux descendants. On veut transformer un arbre quelconque A en un arbre binaire B. On utilise pour cela la stratégie présentée sur la figure 3 :
– les noeuds de B sont tous les noeuds de A sauf sa racine ;
– pour chaque noeud N, le fils gauche de N dans B est le premier fils de N dans A ;
– pour chaque noeud N, le fils droit de N dans B est le frère de N dans A (ie. le noeud qui est immédiatement à sa droite).

1. Écrire une méthode `boolean estBinaire()` qui teste si un arbre est binaire.
2. Écrire une méthode `Arbre toBinaire()` qui transforme un arbre quelconque en un arbre binaire avec la méthode décrite au-dessus.
3. Écrire une méthode `Arbre fromBinaire(int racine)` qui effectue la transformation inverse (en ajoutant `racine` pour la racine qui avait été oubliée dans la transformation).