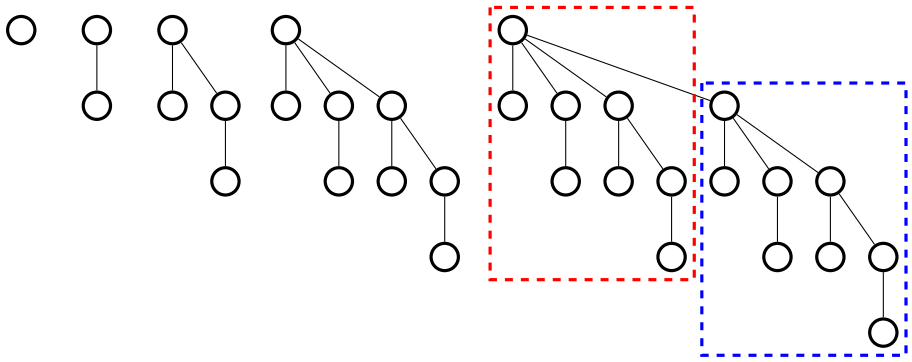
université PARIS DIDEROT PARIS 7

INITIATION À LA PROGRAMMATION (IP2)

TD12 : Tas binomial

Ce TD est consacré au tas binomial, qui est essentiellement une liste d'arbres spéciaux qui nous permet d'extraire rapidement le plus petit élément.
Un *arbre binomial* est un arbre d'arité arbitraire construit récursivement. Les arbres binomiaux sont regroupés par leur *type*. Un arbre binomial de type 0 est un sommet sans feuille. Pour $k \geq 0$, un arbre binomial de type $k+1$ se construit en attachant la racine d'un arbre binomial de type k comme l'enfant d'un autre arbre binomial de type k. Voici quelques exemples d'arbre binomial dans la figure suivante. Nous pouvons constater que l'arbre de type 4 est bien construit par attacher un arbre de type 3 (rectangle bleu) à la racine d'un autre arbre de type 3 (rectangle rouge).



Voici une classe `ArbreB` qui modélise un arbre binomial dont chaque noeud contient un entier.

```
class ArbreB{
    public final int type;
    private final ArbreB[] enfants;
    public final int val;
}
```

Exercice 1 Commençons par la manipulation des arbres binomiaux.

1. Combien de sousarbres un arbre binomial de type k possède-t-il ? Combien de noeuds un arbre binomial de type k possède-t-il ?
2. Ecrire un constructeur `ArbreB(int val)` qui crée un arbre binomial de type 0 contenant la valeur `val`. Ecrire un constructeur normal `ArbreB(int type, ArbreB[] enfants, int val)` ;
3. Ecrire une méthode `public ArbreB fusion(ArbreB a)` qui a la fonctionnalité suivante : si `this` et `a` ont le même type k, alors construire un nouvel arbre de type $k+1$ en attachant l'arbre dont la racine porte une valeur plus grande à la racine de l'autre. Tous les arbres ainsi construits sont croissants sur toutes les branches. Pourquoi ?
4. Ecrire une méthode `void affiche()` qui affiche les noeuds en ordre préfixe.
5. Ecrire une méthode `int somme()` qui renvoie la somme de tous les valeurs dans l'arbre.

Un *tas binomial* est tout simplement une liste d'arbres binomiaux du type 2 à 2 distinct. Il peut être modélisé par la classe `TasB` suivante, venue avec un constructeur vide.

```
class TasB{
    private ArbreB[] element = new ArbreB[64];
    // element[k] est l'arbre de type k, s'il existe
}
```

Exercice 2 Continuons par la manipulation des tas binomiaux.

1. Quel est le nombre maximal de noeuds pour un tas binomial représenté par la classe `TasB` ? Combien de mémoire qu'il prendra dans ce cas ? Justifier pourquoi on peut supposer qu'il n'y aura pas de débordement du tableau `element[64]`.
2. Ecrire un constructeur normal `TasB(ArbreB[] e)` qui prend les premiers 64 éléments du tableau (s'il y en a). Ecrire un constructeur vide `TasB()`.
3. Ecrire une méthode `void ajout(ArbreB a)` qui ajoute l'arbre `a` dans le tas binomial courant en conservant la propriété du tas, c'est-à-dire les types des arbres 2 à 2 distincts. A chaque fois qu'il y a deux arbres du même type, il faut les fusionner. L'ajout d'un arbre peut nécessiter plusieurs fusions.
4. Ecrire une méthode `void ajout(int val)` qui ajoute la valeur `val` au tas courant. Nous pouvons utiliser les méthodes précédentes.
5. Ecrire une méthode `void ajout(TasB t)` qui ajoute tous les arbres du tas `t` au tas courant.
6. Dans la classe `ArbreB`, écrire une méthode `TasB sousArbres()` qui renvoie un tas binomial formé par les sous-arbres de l'arbre courant.
7. Ecrire une méthode `int retireMinimal()` qui retire le noeud avec la valeur minimale du tas en conservant la propriété du tas binomial. Nous pouvons d'abord chercher l'arbre dont la racine contient la valeur minimale (pourquoi ?), retirer cet arbre du tas, construire un autre tas contenant tous les sous-arbres mais pas la racine, puis le fusionner avec le tas original.
8. Observer le nombre d'opérations dans chaque méthode. Pourquoi le tas binomial est efficace ?
9. Ecrire une méthode **statique** `int[] tri(int[] tab)` qui renvoie un tableau trié en ordre croissant en utilisant un tas binomial.