Corrigé du TD 8

Exercice 1.

Question 1. *Écrire une procédure qui recherche une valeur* `x` *dans un ABR et retourne un pointeur vers le nœud correspondant, ou* `None` *si* `x` *n'est pas présent.*
Voici une version récursive :

```
rechercheABR(A : arbreBinaire, x : entier) : pointeur sur Noeud
    si A == None alors retourner None
    si A->valeur == x alors retourner A
    si x < A->valeur alors retourner rechercheABR(A->gauche, x)
    sinon retourner rechercheABR(A->droite, x)
```

Comme la recherche parcourt une unique branche de l'arbre, on peut aussi donner une version itérative :

```
rechercheABRit(A : arbreBinaire, x : entier) : pointeur sur Noeud
    P : pointeur sur Noeud ; P = A
    tant que P <> None et P->valeur <> x faire
        si x < P->valeur alors P = P.gauche
        sinon P = P.droit
    retourner P
```

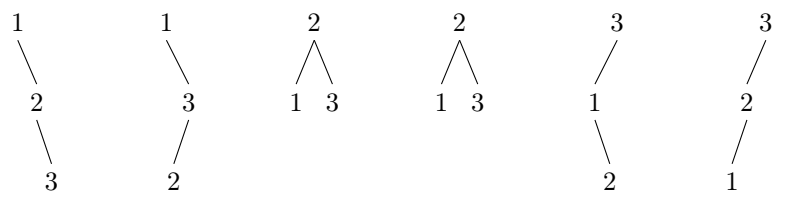
La complexité dans le pire des cas est égale à la hauteur h de l'arbre, atteinte quand la valeur recherchée est présente sur une feuille de profondeur maximale. Si n est le nombre de nœuds de l'arbre, on a $h + 1 \leq n \leq 2^{h+1} - 1$. La complexité dans le pire des cas, en fonction de n, est donc égale à $n - 1$, atteinte quand l'arbre est dégénéré et `x` est présente sur l'unique feuille. Si on suppose l'arbre complet, la complexité dans le pire des cas est $\log_2(n + 1) - 1$, soit $\Theta(\log n)$.

Question 2. Écrire une procédure qui insère la valeur `x` comme nouvelle feuille d'un ABR. Comparer la complexité de cette procédure à celle de la question 1.

```
insereFeuille(A : arbreBinaire, x : entier) : arbreBinaire
    si A == None alors
        N : pointeur sur noeud ; N = Nouveau(noeud)
        N->valeur = x ; N->gauche = None ; N->droit = None
        retourner N
    si x <= A->valeur alors
        A->gauche = insereFeuille(A->gauche, x)
    si x > A->valeur alors
        A->droit = insereFeuille(A->droit, x)
    retourner A
```

Question 3. *Construire tous les ABR possibles en insérant aux feuilles des nœuds étiquetés avec les valeurs de l'ensemble* $E_3 = \{1, 2, 3\}$.

Voici les ABR obtenus en insérant 1, 2, 3 dans les ordres $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$, $(3, 2, 1)$.



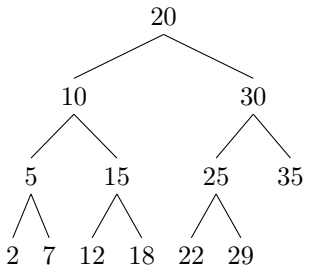
Question 4. *Écrire une procédure qui affiche les valeurs d'un ABR par ordre croissant, puis par ordre décroissant.*

Il s'agit simplement de l'affichage infixe.

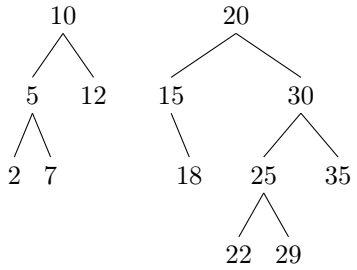
```
afficheCroissant(A : ABR)
    si A == None alors retourner
    afficheCroissant(A->gauche)
    afficher A->valeur
    afficheCroissant(A->droit)
```

Pour l'ordre décroissant il suffit d'échanger les deux appels récursifs.

Question 5. *Couper l'ABR suivant à la valeur* 13.



Réponse :



Question 6. *Écrire une procédure qui retourne une coupe d'un ABR* `A` *selon une valeur* `x`.

```
coupeSelon(A : ABR, x : entier) : coupe
    C : coupe ; D : coupe
    si A == None alors
        C.inf = None ; C.sup = None
        retourner C
    si x == A->valeur alors
        C.inf = A->gauche
        C.sup = A->droit
    si x < A->valeur alors
        D = coupeSelon(A->gauche, x)
        C.sup = A
        C.sup->gauche = D.sup
        C.inf = D.inf
    si x > A->valeur alors
        D = coupeSelon(A->droit, x)
        C.inf = A
        C.inf->droit = D.inf
        C.sup = D.sup
    retourner C
```

Question 7. *Déterminer le coût de la procédure* `coupeSelon`.

On descend d'un niveau dans l'arbre à chaque appel récursif. La coût dans le pire des cas est donc $\Theta(h)$ et $\Theta(n)$ (cas d'un arbre dégénéré). Si l'arbre est complet, le coût est $\Theta(\log n)$.

Question 8. *Écrire une procédure qui insère un nœud de valeur* `x` *à la racine d'un ABR.*

```
insèreRacine(A : ABR, x : entier) : ABR
    C : coupe ; C = coupeSelon(A, x)
    B : ABR ; B = Nouveau(noeudBinaire)
    B->valeur = x
    B->gauche = C.inf
    B->droit = C.sup
    retourne B
```

Question 9. *Déterminer le coût de la procédure* `insèreRacine`.

Comme précédemment, le coût dans le pire des cas est $\Theta(h)$ et, pour un arbre complet, $\Theta(\log n)$.

Question 10. *Écrire une procédure qui réalise la fusion de deux ABR* `A` *et* `B`*, sans création de nouveaux nœuds.*

```
fusionABR(A, B : ABR) : ABR
    si A == None alors retourner B
    si B == None alors retourner A
    C = coupeSelon(B, A->valeur)
    A->gauche = fusionABR(A->gauche, C.inf)
    A->droit = fusionABR(A->droit, C.sup)
    retourner A
```

Question 11. La valeur à la racine de `A` est comparée à chaque valeur d'une branche de `B` lors de la coupe. La procédure est appelée au plus une fois pour chaque nœud de `A`. Le coût de la procédure est donc majoré par $n_A \times h_B$.