Algorithmique
Partiel n° 2

S2# – Epita

D.S 306064.04 BW (24 Janvier 2017 - 09 :00)

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - ☐ Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - ☐ Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - ☐ Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - ☐ Aucune réponse au crayon de papier ne sera corrigée.

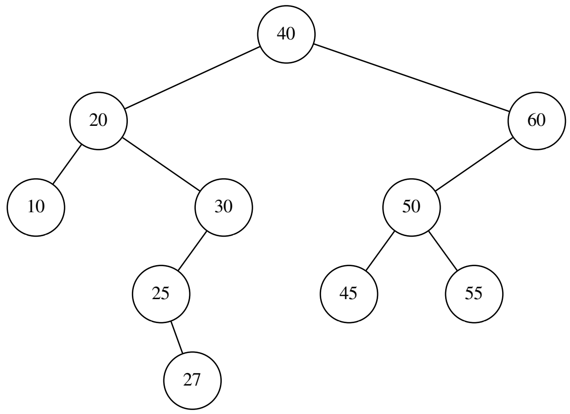
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.

- ☐ **Le code :**
  - ☐ Tout code doit être écrit dans le langage Python (pas de C, Caml, Algo ou autre).
  - ☐ **Tout code Python non indenté ne sera pas corrigé.**
  - ☐ Tout ce dont vous avez besoin (classes, fonctions, méthodes) est indiqué en **annexe** !
  - ☐ Vous n'avez le droit d'utiliser que ce qui a été vu en TD et autorisé en **annexe**
  - ☐ Vos fonctions doivent impérativement respecter les exemples d'applications donnés.

- ☐ Durée : 2h00 (May the force...)

Exercice 1 (Arbre Binaire de Recherche : Ajout racine - *1 point*)

Soit l'arbre binaire de recherche `B` défini ci-dessous :



Représenter graphiquement l'arbre `B` après ajouts en racine des valeurs `26` et `28`.

Exercice 2 (A-V.L. : Ajout - *4 points*)

Représenter graphiquement l'A-V.L. obtenu après ajouts des valeurs suivantes (dans l'ordre donné) dans un arbre vide au départ :

`50, 61, 25, 37, 58, 31, 33, 34, 28, 36, 35`

Exercice 3 (Arbre 2-3-4 : Ajout - *3 points*)

1. Représenter graphiquement l'arbre 2-3-4. obtenu après ajouts avec éclatements à la descente des valeurs suivantes (dans l'ordre donné) dans un arbre vide au départ :

   `13, 36, 3, 12, 27, 4, 24, 16, 28, 30, 6, 32, 17, 21, 22`
2. Représenter graphiquement l'arbre 2-3-4. obtenu précédemment sous sa représentation rouge-noir (les 3-nœuds représentés penchés à gauche).

Exercice 4 (Mystery - *2 point*)

Que retourne la fonction ci-dessous appelée avec B un arbre binaire et un entier $x \in [1, taille(B)]$?

```
fonction mystery(ArbreBinaire B, Entier x) : Element
variables
  Entier y
debut
  y ← taille(g(B)) + 1
  si x > y alors
    retourne mystery(d(B), x - y)
  sinon
    si x = y alors
      retourne contenu(racine(B))
    sinon
      retourne mystery (g(B) , x)
    fin si
  fin si
fin
```

Exercice 5 (Convert - *4 points*)

Le type `BinTreeParent` est une représentation d'un arbre binaire classique avec en plus le champ `parent` qui est un lien vers le père du nœud si celui-ci existe.

Écrire la fonction `convert` qui à partir d'un arbre binaire de type `BinTree` construit une copie de type arbre binaire avec lien de parenté de type `BinTreeParent` équivalent (avec le champ `parent` correctement rempli).

Attention :
- vous n'avez le droit d'écrire qu'**une seule fonction**,
- pas de fonction d'appel,
- la fonction ne prend qu'**un seul paramètre** : l'arbre que l'on souhaite convertir,
- vous n'avez pas le droit de modifier l'arbre passé en paramètre.

Exercice 6 (A-V.L. - Suppression du maximum - *7 points*)

Nous nous intéressons ici à la suppression du maximum dans un A-V.L. avec rééquilibrage.

1. Compléter le tableau donné afin qu'il indique pour chaque cas de déséquilibre (**uniquement** après la suppression du maximum) : quelle est la rotation à effectuer, ainsi que le changement éventuel de hauteur induit (0 si l'arbre ne change pas de hauteur après rotation, 1 sinon). Si des parties du tableau ne sont pas applicables dans notre cas, les rayer. Remplir des cases dans les cas non applicables est considéré comme une erreur.

2. Écrire la fonction récursive qui supprime la valeur maximum d'un A-V.L., retourne un triplet en résultat : la valeur de la clé supprimée, un booléen indiquant si l'arbre a changé de hauteur, et l'arbre.
   Vous pouvez utiliser les procédures `rg`, `rd`, `rgd`, `rdg` qui effectuent les rotations (avec mises à jour des déséquilibres). Le type pour les A-V.L. et les rotations sont rappelés en annexe.

Algorithmique

Type algébrique abstrait : arbre binaire

```
TYPE
    ArbreBinaire

UTILISE
    Noeud, Elément, Entier

OPERATIONS
    arbre_vide : → ArbreBinaire
    <_, _, _>  : Noeud × ArbreBinaire × ArbreBinaire → ArbreBinaire
    racine     : ArbreBinaire → Noeud
    contenu    : Noeud → Elément
    g          : ArbreBinaire → ArbreBinaire
    d          : ArbreBinaire → ArbreBinaire
    taille     : ArbreBinaire → Entier

PRECONDITIONS
    racine(B) est-défini-ssi B≠arbre_vide
    g(B) est-défini-ssi B≠arbre_vide
    d(B) est-défini-ssi B≠arbre_vide

AXIOMES
    racine(<o, B1, B2>)= o
    g(<o, B1, B2>)= B1
    d(<o, B1, B2>)= B2
    taille(arbre_vide) = 0
    taille(<o, B1, B2>) = 1 + taille(B1) + taille(B2)

AVEC
    o          : Noeud
    B, B1, B2  : ArbreBinaire
```

Python : classes, fonctions et méthodes autorisées

Arbre Binaire

`BinTree` : les arbres binaires manipulés ici sont les mêmes qu'en td.

- `None` est l'arbre vide.
- L'arbre non vide a 3 attributs : `key`, `left`, `right`.

```
>>> B = BinTree()
>>> B.key = 'une clef'
>>> B.left = None
>>> B.right = None
```

Arbre Binaire avec lien de parenté

`BinTreeParent` : Les arbres binaires avec lien de parenté :

- `None` est l'arbre vide.
- L'arbre non vide a 4 attributs : `key`, `left`, `right`, `parent`.

```
>>> P = BinTreeParent()
>>> P.key = 'une clef'
>>> P.left = None
>>> P.right = None
>>> P.parent = None
```

A-V.L.

`AVL` : les A-V.L. manipulés ici sont les mêmes qu'en td.

- `None` est l'arbre vide.
- L'arbre non vide a 4 attributs : `key`, `left`, `right`, `balance`.

```
>>> A = AVL()
>>> A.key = 47
>>> A.left = None
>>> A.right = None
>>> A.balance = 0
```

Les procédures de rotations `rg`, `rd`, `rdg` et `rgd` sont déjà implémentées avec mise à jour des déséquilibres, mais sans vérification de l'arbre passé en paramètre.

```
>>> A = AVL()
# do stuff on A
>>> rg(A)
>>> rd(A)
>>> rdg(A)
>>> rgd(A)
```