Algorithmique
Partiel n° 2 (P2)

INFO-SUP S2
EPITA

16 mai 2023 - 08h30-10h30

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 (fonctions, méthodes) est indiqué en **annexe** !
  - — Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
  - — Comme d'habitude **l'optimisation est notée**. Si vous écrivez des fonctions non optimisées, vous serez noté sur moins de points. [1]
- ☐ Durée : 2h00

1. Des fois, il vaut mieux peu de points que pas de point.

Exercice 1 (Arbre 2-3-4 : Ajout - *2 points*)

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 :

```
41, 33, 7, 30, 31, 23, 22, 10, 45
```

Exercice 2 (Dessins – *4 points*)

1. Construire l'AVL correspondant aux ajouts successifs des valeurs $\{11, 30, 42, 32, 45, 23, 17, 5, 8, 1\}$ à partir de l'arbre vide.
   — Vous dessinerez 2 arbres : l'arbre après l'insertion de 17 puis l'arbre final
   — Indiquez quelles rotations ont été effectuées, dans l'ordre (par exemple si une rotation gauche a été effectuée sur l'arbre de racine 42, indiquer $rg(42)$ et si une rotation droite-gauche a été effectuée sur l'arbre de racine 42, indiquer $rdg(42)$).
2. A partir de l'AVL de la figure 1, construire l'AVL résultant de la destruction de la clé 21.
   Indiquez quelles rotations ont été effectuées, dans l'ordre (par exemple si une rotation gauche a été effectuée sur l'arbre de racine 42, indiquer $rg(42)$ et si une rotation droite-gauche a été effectuée sur l'arbre de racine 42, indiquer $rdg(42)$).
   Lors de la suppression dans un point double, on prendra forcement l'élément maximum du sous-arbre gauche.

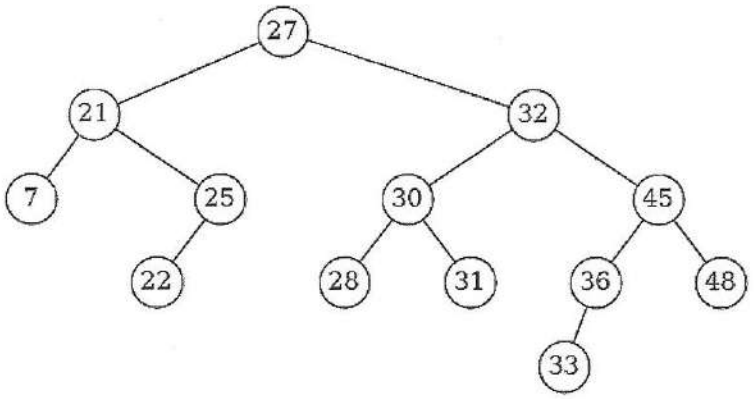

FIGURE 1 – A-V.L. pour les suppressions

Exercice 3 (Ajout avec mise à jour de la taille– *5 points*)

Écrire la fonction `addwithsize(`B`,` x`)` qui ajoute la valeur x en feuille dans l'arbre binaire de recherche B de type `BinTreeSize`, sauf si x est déjà présente. La fonction retourne l'arbre ainsi qu'un booléen indiquant si l'insertion a eu lieu.
L'arbre est représenté par le type `BinTreeSize`, il faut donc mettre à jour, lorsque nécessaire, le champ *size* en certains nœuds de l'arbre.

Exercice 4 (Plus proche ancêtre commun – *5 points*)

Le plus proche ancêtre* commun de deux noeuds d'un arbre binaire de recherche est le noeud le plus bas dans l'arbre binaire de recherche (le plus profond) ayant ces deux noeuds pour descendants.

* On considère qu'un noeud peut être son propre ancêtre.

Écrire la fonction `lca(B, x, y)` avec :
- B, un arbre binaire de recherche supposé non vide et dont toutes les clés sont uniques
- x et y, deux entiers qui correspondent aux clés de deux noeuds de B

et qui retourne la clé du plus proche ancêtre commun des noeuds contenant x et y.

On suppose de plus que l'arbre binaire de recherche B **contient** x **et** y **et que** $x \neq y$.

Remarques :
- `lca(B, x, y) = lca(B, y, x)`
- il est possible que `lca(B, x, y) = x` ou `lca(B, x, y) = y` (voir les exemples ci-dessous)

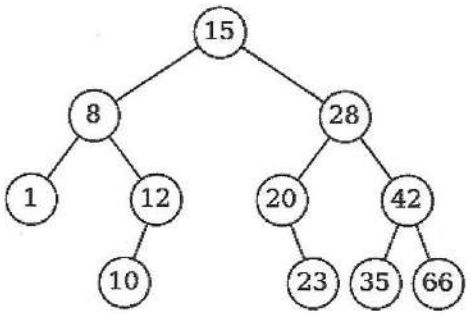

FIGURE 2 – Arbre binaire de recherche B1

Exemples d'applications avec B1 l'arbre binaire de recherche de la Figure 2 :

```
>>> lca(B1, 10, 15)
15
>>> lca (B1, 20, 66)
28
>>> lca(B1, 8, 10)
8
>>> lca(B1, 10, 42)
15
```

Exercice 5 (Mystery – *4 points*)

Soit l'opération suivante :

SORTE
ABR

UTILISE
élément

OPÉRATIONS
mystery: ABR × élément × élément → ABR

PRÉCONDITIONS
mystery(B, x, y) est-défini-ssi $x < y$

AXIOMES
mystery(arbre-vide, x, y) = arbre-vide
$r < x < y \Rightarrow$ *mystery*(<r, L, R>, x, y) = *mystery*(R, x, y)
$x \leq r \leq y \Rightarrow$ *mystery*(<r, L, R>, x, y) = <r, *mystery*(L, x, y), *mystery*(R, x, y)>
$x < y < r \Rightarrow$ *mystery*(<r, L, R>, x, y) = *mystery*(L, x, y)

AVEC
r, x, y : élément
B, L, R : ABR

1. Dessinez l'arbre résultat de `mystery(`$B1$`, 13, 22)` avec $B1$ l'arbre de la Figure 2.
2. Dessinez l'arbre résultat de `mystery(`$B1$`, −2, 13)` avec $B1$ l'arbre de la Figure 2.
3. Implémenter l'opération `mystery(B, x, y)` en Python avec B arbre binaire de recherche et x et y deux entiers tels que $x < y$. La fonction modifie l'arbre B.

Annexes

Les arbres binaires

Les arbres binaires manipulés ici sont les mêmes qu'en td.

- L'arbre vide est `None`
- L'arbre non vide est (une référence sur) un objet de la class `BinTree` avec 3 attributs : `key`, `left`, `right`.
  - `B` : classe `BinTree`
  - `B.key` : contenu du nœud racine
  - `B.left` : le sous-arbre gauche
  - `B.right` : le sous-arbre droit

class `BinTreeSize`

Les arbres de la classe `BinTreeSize` : les mêmes que ceux de la classe `BinTree` avec en plus le *champ* `size` qui indique en chaque nœud la taille de l'arbre dont il est racine.

Donc si `BS` représente un arbre non vide, c'est (une référence sur) un objet de la classe `BinTreeSize` avec les 3 attributs habituels (`BS.key`, `BS.left`, `BS.right`) et en plus `BS.size`.

Pour créer un nouveau nœud contenant la clé `k`, de sous-arbres gauche `L` et droit `R`, racine d'un arbre de taille `s` :

```
1  >>> N = BinTreeSize(k, L, R, s)
```

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas vous devez donnez leurs spécifications : on doit savoir ce qu'elles font, que signifient les paramètres.

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.