Algorithmique
Partiel n° 2 (P2)

INFO-SUP S2
EPITA

7 juin 2021 - 8h30-10h30

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** !
- Durée : 2h00

Exercice 1 (Arbres de recherche – *4 points*)

1. Combien ?
   (a) Combien d'arbres binaires de recherche différents* peut-on construire avec les valeurs 1, 2, 3 et 4 ?
   (b) Combien d'A-V.L. différents* peut-on construire avec les valeurs 1, 2, 3 et 4 ?
   (c) Combien d'arbres 2-3-4 différents* peut-on construire avec les valeurs 1, 2, 3 et 4 ?
   *différents = structure et distribution des clés
2. Quoi ?
   Lesquels des arbres suivants sont des arbres 2-3-4 ?

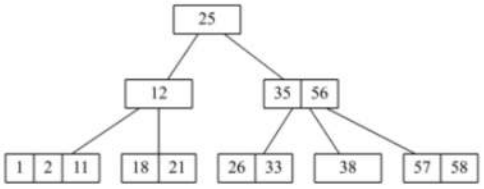

FIGURE 1 – B_1

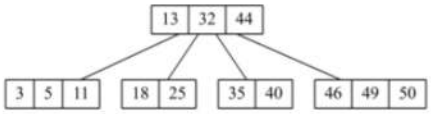

FIGURE 2 – B_2

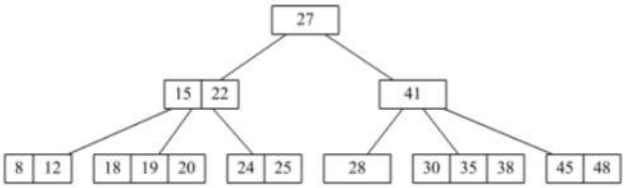

FIGURE 3 – B_3

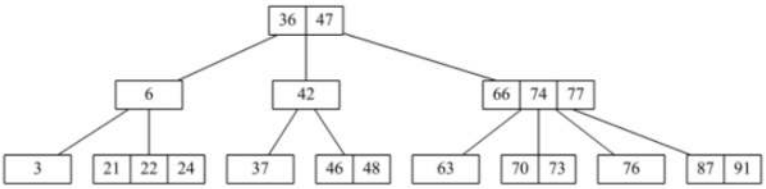

FIGURE 4 – B_4

Exercice 2 (Dessins – *4 points*)

1. Construire l'AVL correspondant aux ajouts successifs des valeurs $\{13, 10, 17, 15, 20, 1, 5, 25, 18, 19\}$ à partir de l'arbre vide.
   — Vous dessinerez 2 arbres : l'arbre après l'insertion de 5 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 $rg(42)$).
2. A partir de l'AVL de la figure 5, construire l'AVL résultant de la destruction de la clé 23.
   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 $rg(42)$).
   Lors de la suppression dans un point double, on prendra forcement l'élément maximum du sous-arbre gauche.

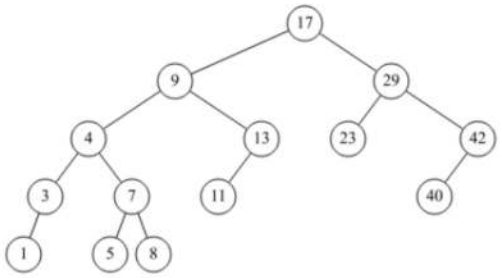

FIGURE 5 – A-V.L. pour les suppressions

Exercice 3 (Test – *4 points*)

On veut vérifier si un arbre binaire est bien un arbre binaire de recherche. Pour cela, écrire la fonction `__testBST(B, inf, sup)` qui vérifie si l'arbre B est bien un arbre binaire de recherche avec ses valeurs dans l'intervalle $]inf, sup]$. La fonction sera appelée de la manière suivante :

```
1         def testBST(B):
2             return __testBST(B, -infty, infty)
```

avec `infty` valeur définie qui représente une valeur "infinie", c'est à dire supérieure à n'importe qu'elle valeur numérique !

Exercice 4 (Génération – *5 points*)

Dans un arbre binaire, deux valeurs sont de même génération si elles sont à la même profondeur.

Écrire la fonction `generation(B, x, y)` qui vérifie si 2 valeurs x et y différentes sont présentes et de même génération dans l'arbre binaire de recherche B dont les valeurs sont toutes distinctes (vous pouvez écrire des fonctions intermédiaires).

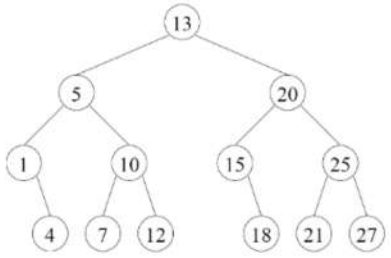

FIGURE 6 – Arbre binaire de recherche `B`

Exemples d'application avec l'arbre binaire de recherche de la Figure 6 :

```
1   >>> generation(None, 42, 4)
2   False
3   >>> generation(B, 5, 4)
4   False
5   >>> generation(B, 21, 27)
6   True
7   >>> generation(B, 7, 27)
8   True
9   >>> generation(B, 20, 15)
10  False
11  >>> generation(B, 2, 13)
12  False
13  >>> generation(B, 2, 50)
14  False
```

Exercice 5 (What is this ? – *3 points*)

Soient les fonctions suivantes :

```
1     def __what(L, left, right):
2
3         if left >= right:
4             return (None, 0)
5         else:
6             mid = left + (right-left) // 2
7             B = BinTreeMyst(L[mid], None, None, 1)
8             (B.left, ml) = __what(L, left, mid)
9             (B.right, mr) = __what(L, mid + 1, right)
10            B.myst += ml + mr
11            return (B, B.myst)
12
13    def mystery(L):
14        (B, m) =  __what(L, 0, len(L))
15        return B
```

La classe `BinTreeMyst` est similaire à la classe `BinTree`.

- L'arbre vide est `None`
- L'arbre non vide est (une référence sur) un objet de la classe `BinTreeMyst` avec 4 attributs : `key, left, right, myst`.
  - `B` : classe `BinTreeMyst`
  - `B.key` : contenu du noeud racine
  - `B.left` : le sous-arbre gauche
  - `B.right` : le sous-arbre droit
  - `B.myst` : attribut mystère

1. Dessiner l'arbre retourné par `mystery`$([1, 1, 10, 32, 8, 6, 50, 7, 32])$ en indiquant pour chaque noeud la valeur de l'attribut `myst`.
2. Quelle propriété doit avoir la liste L pour que l'arbre résultat soit un arbre binaire de recherche ?
3. L'arbre résultat est-il h-équilibré ? Pourquoi ?

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

Vos fonctions

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

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