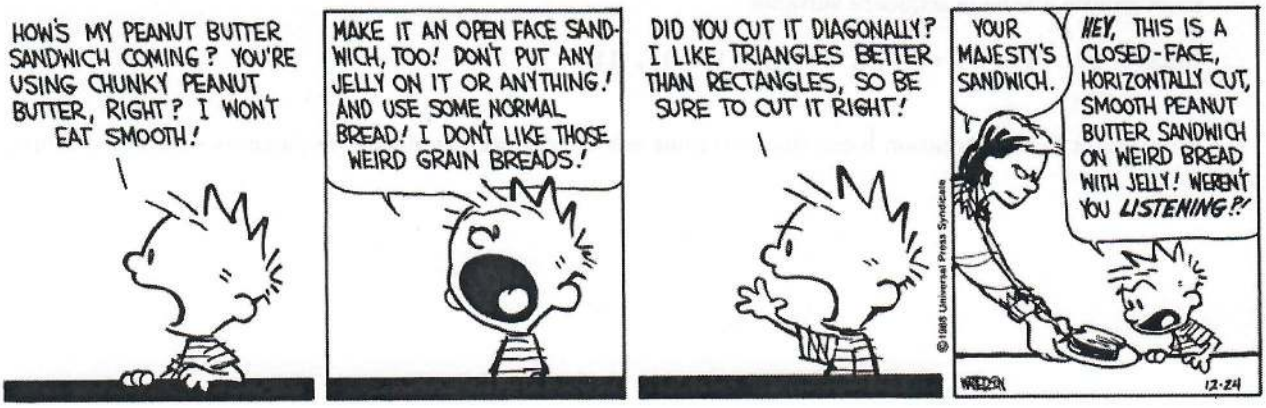
Algorithmique
Contrôle n° 2 (C2)

INFO-SUP S2
EPITA

28 février 2022 - 8 : 30

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 (Un peu de cours... – *4 points*)

Soit l'arbre général **A** représenté sous la forme binaire **premier fils - frère droit** :

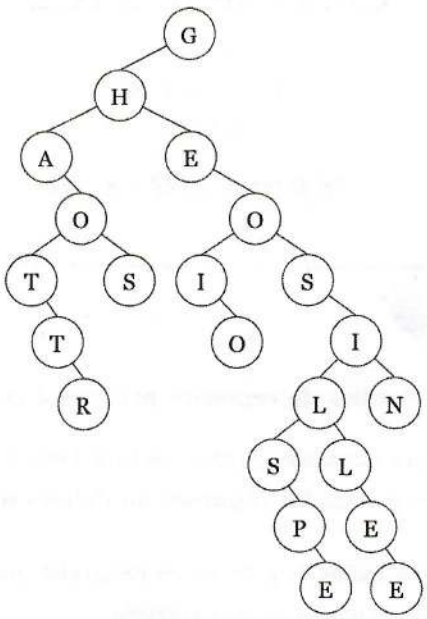


FIGURE 1 – Arbre général **A** sous forme binaire **premier fils - frère droit**

1. Quelle est la taille de l'arbre `A` ?
2. Quelle est la hauteur de l'arbre `A` ?
3. Quelle est la longueur de cheminement interne de l'arbre `A` ?
4. Quelle est la profondeur moyenne externe de l'arbre `A` ?
5. Donner la liste des nœuds de l'arbre `A` rencontré en ordre suffixe.

Exercice 2 (Arbre Binaire : Ordres et représentations – *2 points*)

Soit un arbre binaire représenté sous forme d'occurrences, dont le traitement *infixe* du parcours profondeur main gauche affiche la séquence suivante :

infixe : `00, 001, 0, ε, 100, 10, 1, 110, 11, 111`

En utilisant la numérotation hiérarchique comme valeurs des noeuds, donner le parcours suffixe de l'arbre.

Exercice 3 (Matrice renversée – *4 points*)

Pour cet exercice, on définit la *matrice renversée* de la manière suivante :

- la première ligne devient la dernière ligne, la seconde ligne devient l'avant dernière ligne, etc
- les éléments de chaque ligne sont inversés, le premier élément devient le dernier élément, le second élément devient l'avant-dernier, etc

Écrire la fonction `build_reverse(M)` qui construit et retourne la matrice renversée de M. La matrice M est considérée non vide.

1	2	8	4
5	-1	7	8

FIGURE 2 – `Mat1`

1	0
8	4
5	-3

FIGURE 3 – `Mat2`

9	1	4	-1
7	2	8	-2
0	5	6	10
-4	9	0	12

FIGURE 4 – `Mat3`

Exemples d'applications sur les matrices des figures 2, 3 et 4 :

```
>>> build_reverse(Mat1)
[[8, 7, -1, 5], [4, 8, 2, 1]]
>>> build_reverse(Mat2)
[[-3, 5], [4, 8], [0, 1]]
>>> build_reverse(Mat3)
[[12, 0, 9, -4], [10, 6, 5, 0], [-2, 8, 2, 7], [-1, 4, 1, 9]]
```

Exercice 4 (Plus longue diagonale triée – *5 points*)

Pour cet exercice on définit les *diagonales triées* d'une matrice de la manière suivante :

- les *diagonales* considérées seront uniquement celles qui vont de gauche à droite et de haut en bas partant d'un élément de la première ligne ou de la première colonne et qui vont en diagonale jusqu'à arriver sur un côté de la matrice.
- une *diagonale* est *triée* si tous ses éléments sont triés en ordre croissant.

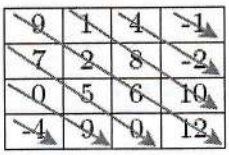


FIGURE 5 – *Diagonales* de `Mat3`

Dans la matrice `Mat3` de la figure 4, les *diagonales* à considérer sont (voir Figure 5) :

- -4
- 0, 9
- 7, 5, 0
- 9, 2, 6, 12
- 1, 8, 10
- 4, -2
- -1

Écrire la fonction `longest_sorted_diagonal(M)` qui retourne le nombre d'éléments de la plus longue *diagonale* triée de la matrice M considérée carrée et non vide.

Les deux *diagonales* triées sont (0, 9) et (1, 8, 10), la longueur de la plus longue *diagonale* triée de `Mat3` est donc 3.

Exercice 5 (Fils unique – *5 points*)

La fonction `get_singles(B)` prend en paramètre un arbre binaire B et retourne un couple `(leaves, singles)` où :

— `leaves` est le nombre de feuilles sans frères de B.
— `singles` est le nombre de points simples de B..

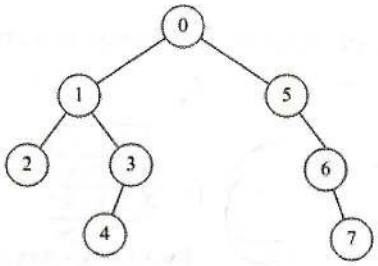


FIGURE 6 – $B1$

Exemples d'application :

```
>>> get_singles(B1)
(2, 3)
>>> get_singles(bintree.BinTree(42, None, None))
(1, 0)
>>> get_singles(None)
(0, 0)
```

Ecrire la fonction auxiliaire `__get_singles(B, sibling)` où B est un arbre binaire non vide, `sibling` une valeur booléenne indiquant si B a un frère et qui retourne un couple `(leaves, singles)` avec les mêmes spécifications que la fonction `get_singles(B)`.

La fonction `__get_singles(B, sibling)` sera appelée de la manière suivante :

```
def get_singles(B):
    if B == None:
        return (0,0)
    else:
        return __get_singles(B, False)
```

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

Fonctions et méthodes autorisées

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes ainsi que la fonction `range` :

```
>>> L = []

>>> for i in range(5):
        L.append(i)

>>> L
[0, 1, 2, 3, 4]

>>> len(L)
5

>>> for i in range(5, 10):
        L.append(i)
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> for i in range(9, -1, -1):
        print(i, end=" ")
9 8 7 6 5 4 3 2 1 0
```

Les fonctions `min` et `max`, mais uniquement avec deux valeurs entières !

Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).

Vos fonctions

Vous pouvez écrire des fonctions 'intermédiaires' / 'supplémentaires', dans ce cas vous devez donner 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.