Construction d'un Arbre Binaire de Recherche

Objectifs :

- Savoir construire un arbre binaire de recherche par insertion de valeurs.
- Savoir évaluer le coût en temps de sa construction dans le pire cas.

1 Tri du bijoutier

Considérons le problème du bijoutier voulant trier par grosseur un tas de diamants : pour faire cette opération il se sert d'un tamis ce qui lui permet de séparer le tas initial en deux tas, et il recommence avec de nouveaux tamis pour chaque tas.

- Pour classer plus facilement les tas obtenus, il place toujours le tas des plus petits diamands obtenu à gauche et celui des plus grands à droite.
- Le bijoutier utilise des tamis dont l'ouverture de maille indique la taille minimale des diamants qui ne passent pas à travers la grille de maillage.
  Par exemple, le tamis d'ouverture de maille 1mm retient tous les diamands d'une taille supérieure ou égale à 1mm.

Travail à faire

1. Représente sur un même arbre binaire tous les tas de diamands obtenus en utilisant successivement des tamis d'ouverture de maille $7mm$, puis $9mm$, puis $3mm$, puis $5mm$, le tas initial étant à la racine.

tas initial

•

2. Indique sur chaque arête l'inégalité que vérifie la taille des diamands ainsi triés.
3. En déduire, pour un tas quelconque de l'arbre de tri, une relation entre la taille des diamands de son sous-arbre gauche avec celle de son du sous-arbre droit ?

2 Arbre Binaire de Recherche

À l'instar du bijoutier qui trie ses diamands, nous allons nous servir de cet algorithme pour construire un arbre binaire particulier, appelé *arbre binaire de recherche*, dont la construction nous permettra de trier les valeurs d'une liste.

On considère la liste de nombres `liste = [7, 9, 3, 5, 4, 1]` et la fonction `inserer(x, a)` suivante :

```
def inserer(x, a):
    '''ajoute x à l'arbre a, renvoie un nouvel arbre'''
    if a is None:
        return Noeud(None, x, None)
    elif  x < a.valeur:
        return Noeud(inserer(x, a.gauche), a.valeur, a.droit)
    else:
        return Noeud(a.gauche, a.valeur, inserer(x, a.droit))
```

On rappelle la définition de la classe `Noeud`, introduite précédemment comme :

```
class Noeud:
    def __init__(self, g, d, v):
        self.gauche = g
        self.droit = d
        self.valeur = v
```

Le script ci-dessous permet de construire un arbre de type `Noeud` en utilisant la fonction `inserer` et stocke sa référence dans une variable `a`.

```
a = None
for v in liste:
    a = inserer(v, a)
```

Travail à faire

1. Exécute pas à pas ce script et compléte la représentation l'arbre `a` obtenue à la fin de chaque itération.
2. L'un des parcours, postfixe, infixe ou préfixe, de l'arbre précédemment obtenu trie la liste. Lequel ?

On considère maintenant la liste triée `[1, 3, 4, 5, 7, 9]` contenant les mêmes valeurs.

3. (a) Obtient-on le même arbre à partir de cette nouvelle liste ? Est-il trié avec le même parcours ?
   (b) Détermine le nombre total de comparaions effectuées pour le construire.
   (c) En déduire le nombre de comparaisons effectuées dans la construction de l'arbre pour une liste de n éléments, dans le pire cas.
4. Trouve une liste, contenant les mêmes valeurs, qui permet de construire un arbre en effectuant le moins de comparaison possible.
5. (a) Dans chacun des deux cas précédent, quel lien peut-on établir entre la hauteur h de l'arbre et le nombre de comparaisons effectuées pour ajouter un noeud supplémentaire dans l'arbre, dans le pire cas.
   (b) Sers-toi de la cette relation pour expliquer quelle forme d'arbre sera optimale avec cet algortihme d'insertion.
6. PLUS ULTRA Détermine le nombre total de comparaisons effectuées pour construire un arbre plein de taille n, c'est à dire un arbre dont tous les niveaux sont remplis.