Partie C : classement à l'issue d'une étape

Chaque jour, à la fin de l'étape, on décide de construire un Arbre Binaire de Recherche (ABR) afin d'établir le classement des équipes. Chaque nœud de cet arbre est un objet de type `Equipe`.

Dans cet arbre binaire de recherche, en tout nœud :

- toutes les équipes du sous-arbre gauche sont strictement plus rapides que ce nœud ;
- toutes les équipes du sous-arbre droit sont moins rapides ou sont à égalité avec ce nœud.

Voici les temps, en heure et minute, relevés à l'issue de la première étape :

Temps à l'arrivée de la première étape											
Equipe	`eq1`	`eq2`	`eq3`	`eq4`	`eq5`	`eq6`	`eq7`	`eq8`	`eq9`	`eq10`	`eq11`
Temps	4h36	3h57	3h09	5h49	4h45	3h26	4h51	5h52	4h31	3h44	4h26

Dans l'arbre binaire de recherche initialement vide, on ajoute successivement, dans cet ordre, les équipes `eq1`, `eq2`, `eq3`, …, `eq11`, 11 objets de la classe `Equipe` tous construits sur le même modèle que l'objet `eq11` précédent.

8. Dans l'arbre binaire de recherche ci-dessous, les nœuds `eq1` et `eq2` ont été insérés. Recopier et compléter cet arbre en insérant les 9 nœuds manquants.

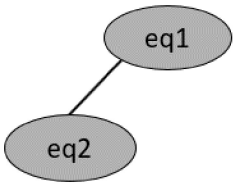

Figure 1. Premiers éléments de l'ABR

On donne ci-dessous la classe `Noeud`, permettant de définir les arbres binaires :

```
1 class Noeud:
2     def __init__(self, equipe, gauche = None, droit = None):
3         self.racine = equipe
4         self.gauche = gauche
5         self.droit = droit
```

On donne ci-dessous le code d'une fonction `construction_arbre` qui, à partir d'une liste d'éléments de type `Noeud` permet d'insérer successivement chaque nœud à sa place dans l'ABR.

```
1 def construction_arbre(liste):
2     a = Noeud(liste[0])
3     for i in range(1,len(liste)):
4         inserer(a, liste[i])
5     return a
```

La fonction `construction_arbre` fait appel à la fonction `inserer` qui prend pour paramètre `arb`, de type `Noeud`, et `eq`, de type `Equipe`. Cette fonction construit le nœud à partir de `eq` et l'insère à sa place dans l'ABR.

```
1  def inserer(arb, eq):
2      """ Insertion d'une équipe à sa place dans un ABR contenant
3          au moins un noeud."""
4      if convert(eq.temps_etape) < convert(arb.racine.temps_etape):
5          if arb.gauche is None:
6              arb.gauche = ...
7          else:
8              inserer(..., eq)
9      else:
10         if arb.droit is None:
11             arb.droit = Noeud(eq)
12         else:
13             ...
```

10. Expliquer en quoi la fonction `inserer` est une fonction récursive.

11. Recopier et compléter les lignes 6, 8 et 13 de la fonction `inserer`.

12. Recopier et compléter les lignes 3 et 5 de la fonction `est_gagnante` ci-dessous qui prend en paramètre un ABR `arbre`, de type `Noeud`, et qui renvoie le nom de l'équipe ayant gagné l'étape.

```
1 def est_gagnante(arbre):
2     if arbre.gauche == None:
3         return ...
4     else:
5         return ...
```

Partie D : classement général

On décide d'établir un classement général obtenu à partir du cumul des temps mis par chaque équipe pour parcourir l'ensemble des 9 étapes.

Sur le même principe que l'arbre de la partie précédente, on construit l'ABR ci-dessous qui permet, grâce au parcours d'arbre approprié, d'établir ce classement général des équipes.

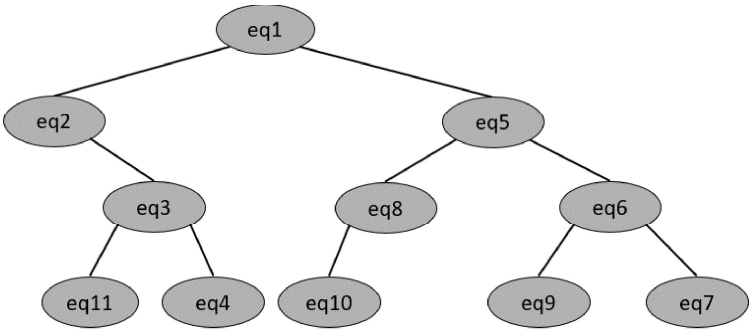

Figure 2. ABR du classement général

Le règlement prévoit la disqualification d'une équipe en cas de non-respect de celui-ci. Il s'avère que l'équipe 2 et l'équipe 5 doivent être disqualifiées pour manquement au règlement. Les nœuds `eq2` et `eq5` doivent donc être supprimés de l'ABR précédent.

Pour supprimer un nœud `N` dans un ABR, trois possibilités se présentent :

- le nœud `N` à supprimer est une feuille : il suffit de le retirer de l'arbre ;
- le nœud `N` à supprimer n'a qu'un seul fils : on relie le fils de `N` au père de `N` et on supprime le nœud `N` ;
- le nœud `N` à supprimer possède deux fils : on le remplace par son successeur (l'équipe qui a le temps immédiatement supérieur) qui est toujours le minimum de ses descendants droits.

13. Dessiner le nouvel arbre de recherche `a_final` obtenu après suppression des équipes `eq2` et `eq5` dans l'ABR correspondant au classement général.

L'organisateur souhaite disposer d'une fonction `rechercher` permettant de savoir si une équipe a été disqualifiée ou non. On donne les spécifications de la fonction `rechercher`, prenant en paramètre `arbre` et `equipe`.

```
def rechercher(arbre, equipe):
    """
    Paramètres
    ---------
    arbre : un ABR, non vide, de type Noeud, représentant le
            classement général.
    equipe : un élément, de type Equipe, dont on veut déterminer
             l'appartenance ou non à l'ABR arbre.
    Résultat
    ---------
    Cette fonction renvoie True si equipe est un nœud de arbre,
    False sinon.
    """
    ...
```

Pour cette fonction (`a_final` désigne l'arbre obtenu à la question 13, après suppression des équipes 2 et 5) :

- l'appel `rechercher(a_final, eq1)` renvoie `True` ;
- l'appel `rechercher(a_final, eq2)` renvoie `False`.

14. Écrire le code de la fonction `rechercher`.