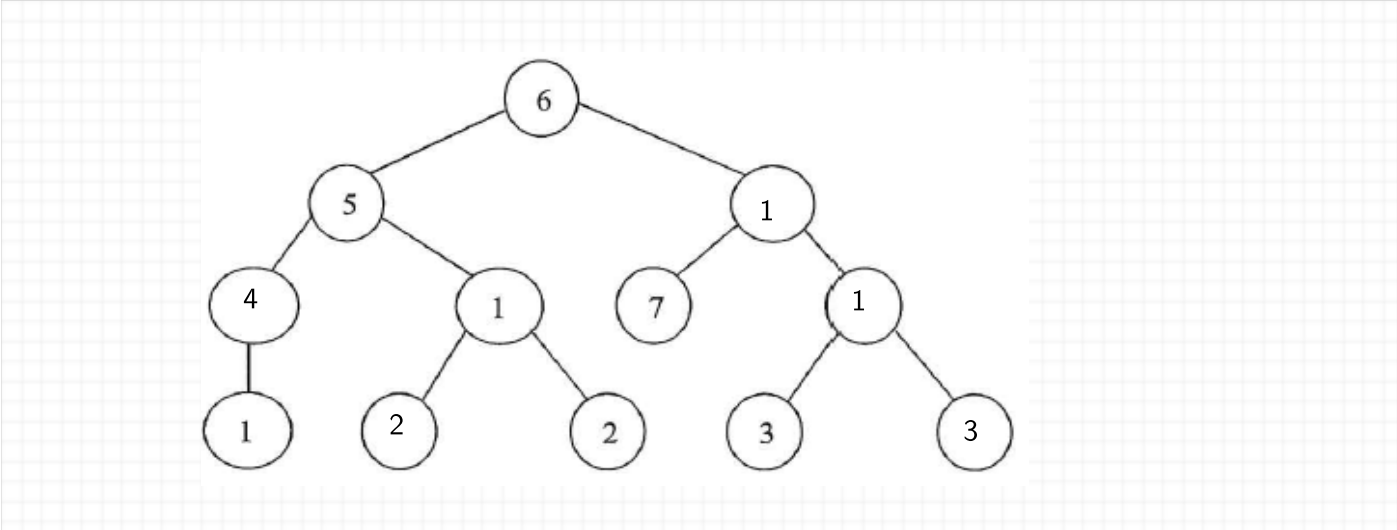
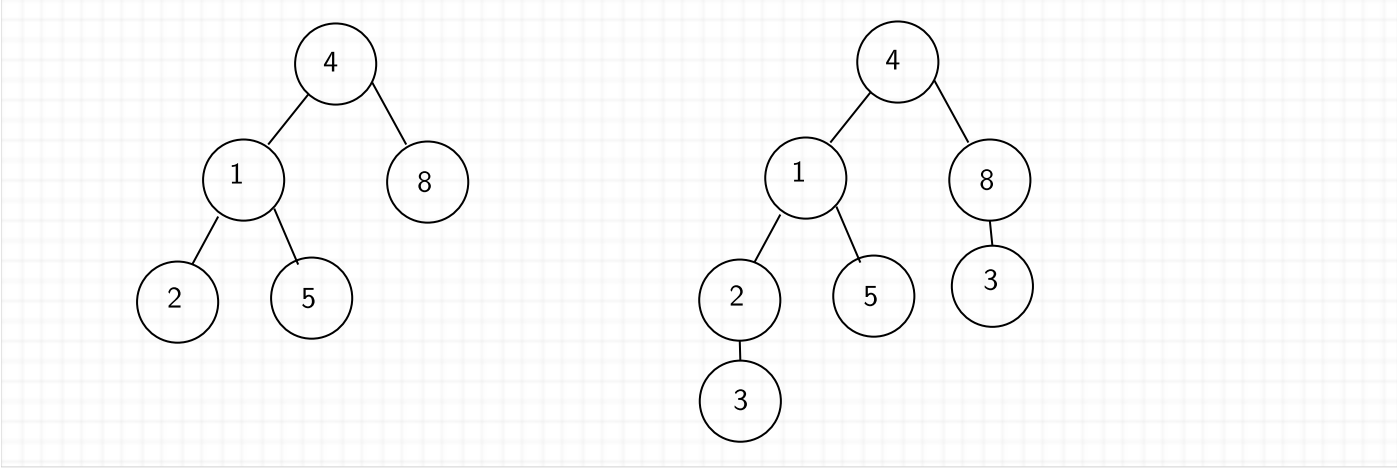
Corrigé DS arbres BAC

Remarques globales: Je n'ai jamais autant écrit "consigne" dans une correction. J'ai donc enlevé des points pour la consigne. Je ne sais pas vous apprendre à lire, je ne suis pas formé pour ça, il faut vous responsabiliser sur la lecture des énoncés. Un mot a un sens précis.
Bien poncer le corrigé des arbres mobiles.

1. Exercice sur les arbres mobiles:





Ici, on vous demande d'ajouter des instructions, pas de réécrire le code en le bidouillant. On est sur une structure mutable, le type list, qui justement possède des méthodes en place pour muter.

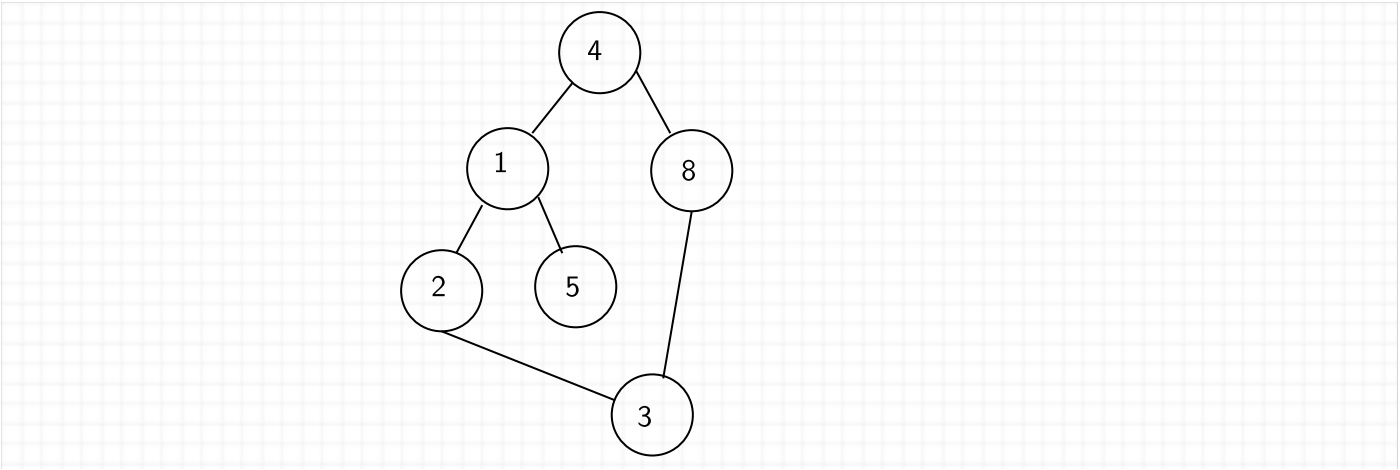
```
n2[1].append([3, []])
n8[1].append([3, []])
```

Voici ce qu'il en faut pas faire:

```
n3 = [3, []]
n2[1].append(n3)
```

```
n8[1].append(n3)
```

Car voici ce qui est généré:



Pour rappel, en python, les types mutables, comme les listes ou les objets que vous créez, sont manipulés par adresse mémoire.

A partir de maintenant, sur notre brouillon, on sait que pour un arbre a, a[0] est sa cle, a[1] représente ses enfants. si a[1] est la liste vide, a est une feuille. a[1][0] est son premier enfant s'il existe, a[1][1] est son deuxième enfant s'il existe. Il est bon de lister les accesseurs d'une structure afin de pouvoir s'y référer pendant l'exercice.

Application du parcours en largeur à une simple accumulation.

defiler n'a qu'un paramètre, c'est en plus rappelé dans l'énoncé. Je ne comprends (vraiment) pas pourquoi certains mettent deux paramètres. La boite elle a qu'un trou et toi tu respectes pas, tu bourres dans un autre trou qui n'existe pas.

```
7. while not est_vide(file):
8. arbre = defiler(file)
11. p = p + arbre[0]
```

Disjonction de cas sur l'arbre et ses enfants.
L'arbre est vide, sinon il a 0 enfants, sinon il a 1 enfant, sinon il a 2 enfants. On est obligés d'arriver à cette disjonction car sinon, on se retrouve à faire des calculs sur des enfants qui n'existent pas nécessairement.

```
def est_mobile(arbre):
  if arbre==[]:              # Arbre vide
    return True
  elif len(arbre[1]) == 0:   # Feuille
    return True
  elif len(arbre[1]) == 1:   # 1 enfant
    return est_mobile(arbre[1][0])
  else:                      # 2 enfants
    return (
     poids(arbre[1][0])==poids(arbre[1][1])
     and est_mobile(arbre[1][0]) and est_mobile(arbre[1][1])
    )
```

Pour aller plus loin:

réécriture booléenne en utilisant le caractère lazy des opérateurs booléens qui nous permet de recréer la structure conditionnelle.
si a est True, b ne sera pas évalué dans a or b, et True sera renvoyé.
si a est False, b ne sera pas évalué dans a and b, et False sera renvoyé.

```
def est_mobile(arbre):
    return (
              arbre==[]
          or len(arbre[1]) == 0)
          or (len(arbre[1]) == 1 and est_mobile(arbre[1][0]))
          or (
                  poids(arbre[1][0])==poids(arbre[1][1])
                and est_mobile(arbre[1][0]) and est_mobile(arbre[1][1])
              )
           )
```

Généralisation à un nombre quelconque d'enfants (fonctionne donc dans notre cas)

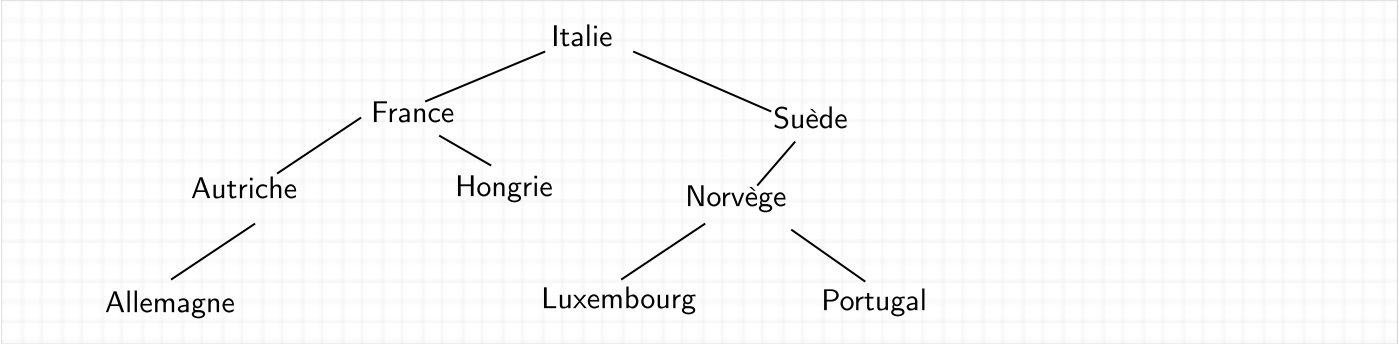
```
def est_mobile(arbre):
    return (
                arbre==[]
            or len(arbre[1]) == 0)
            or (
                    all([poids(a)==poids(arbre[1][0]) for a in arbre[1]])
                  and all([est_mobile(a) for a in arbre[1]])
                )
           )
```

La fonction all renvoie True ssi tous les éléments de l'itérable sont True.
(Il y a aussi la fonction any qui renvoie True si au moins un des éléments de l'itérable est True.)

2. Exercice ABR:

Les questions 1 et 2 servent à ce que tout le monde ait des points. Ne pas avoir bon est inquiétant.

1.a Hauteur 3
1.b True (A est avant P.....)
1.c



2. Italie, France, Suède, Autriche, Hongrie, Norvège

3.
Ici, il faut avoir compris comment fonctionnent les ABR.
On n'a besoin d'aller regarder que d'un seul côté de l'arbre selon que ce qu'on cherche est plus petit ou plus grand que la clé. C'est tout l'intérêt de l'ABR, de diviser la taille du problème par 2 à chaque fois, pour obtenir un algorithme en O(log(n)).

Ligne 2 - "Renvoie True si l'étiquette val appartient à l'arbre arb, False sinon"
Ligne 6 - return True
Ligne 7 - if val < racine(arb)
Ligne 10 - return recherche(droite(arb), val)

4. Non corrigé car corrigé de multiples fois.