EXERCICE 5 (4 points)

Cet exercice traite du thème « algorithmique », et principalement des algorithmes sur les arbres binaires.

On manipule ici les arbres binaires avec trois fonctions :

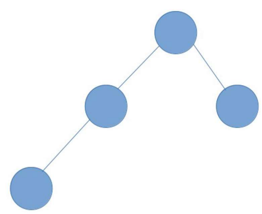
- `est_vide(A)` qui renvoie `True` si l'arbre binaire `A` est vide, `False` s'il ne l'est pas ;
- `sous_arbre_gauche(A)` qui renvoie le sous-arbre gauche de l'arbre binaire `A` ;
- `sous_arbre_droit(A)` qui renvoie le sous-arbre droit de l'arbre binaire `A`.

L'arbre binaire renvoyé par les fonctions `sous_arbre_gauche` et `sous_arbre_droit` peut éventuellement être l'arbre vide.

On définit la **hauteur** d'un arbre binaire non vide de la façon suivante :

- si ses sous-arbres gauche et droit sont vides, sa hauteur est 0 ;
- si l'un des deux au moins est non vide, alors sa hauteur est égale à 1 + M, où M est la plus grande des hauteurs de ses sous-arbres (gauche et droit) non vides.

1. a. Donner la hauteur de l'arbre ci-dessous.



b. Dessiner sur la copie un arbre binaire de hauteur 4.

La hauteur d'un arbre est calculée par l'algorithme récursif suivant :

```
Algorithme hauteur(A):
  test d'assertion : A est supposé non vide
  si sous_arbre_gauche(A) vide et sous_arbre_droit(A) vide:
    renvoyer 0
  sinon, si sous_arbre_gauche(A) vide:
    renvoyer 1 + hauteur(sous_arbre_droit(A))
  sinon, si ... :
    renvoyer ...
  sinon:
     renvoyer 1 + max(hauteur(sous_arbre_gauche(A)),
                         hauteur(sous_arbre_droit(A)))
```

2. Recopier sur la copie les lignes 7 et 8 en complétant les points de suspension.

3. On considère un arbre binaire R dont on note G le sous-arbre gauche et D le sous-arbre droit. On suppose que R est de hauteur 4 et G de hauteur 2.

   a. Justifier le fait que D n'est pas l'arbre vide et déterminer sa hauteur.

   b. Illustrer cette situation par un dessin.

Soit un arbre binaire non vide de hauteur h. On note n le nombre de nœuds de cet arbre. On admet que $h+1 \leqslant n \leqslant 2^{h+1}-1$.

4. a. Vérifier ces inégalités sur l'arbre binaire de la question 1.a.

   b. Expliquer comment construire un arbre binaire de hauteur h quelconque ayant h+1 nœuds.

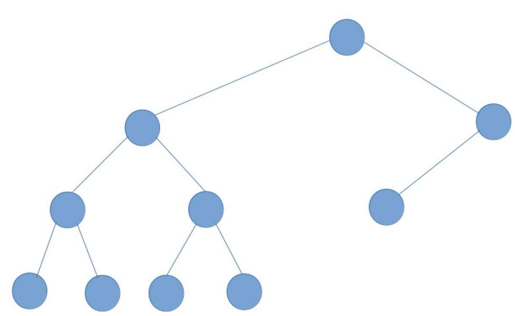
   c. Expliquer comment construire un arbre binaire de hauteur h quelconque ayant $2^{h+1}-1$ nœuds.
   Indication : $2^{h+1}-1 = 1+2+4+...+2^h$.

L'objectif de la fin de l'exercice est d'écrire le code d'une fonction `fabrique(h, n)` qui prend comme paramètres deux nombres entiers positifs `h` et `n` tels que $h+1 < n < 2^{h+1}-1$, et qui renvoie un arbre binaire de hauteur `h` à `n` nœuds.

Pour cela, on a besoin des deux fonctions suivantes:

- `arbre_vide()`, qui renvoie un arbre vide ;
- `arbre(gauche, droit)` qui renvoie l'arbre de fils gauche `gauche` et de fils droit `droit`.

5. Recopier sur la copie l'arbre binaire ci-dessous et numéroter ses nœuds de 1 en 1 en commençant à 1, en effectuant un parcours en profondeur préfixe.



La fonction `fabrique` ci-dessous a pour but de répondre au problème posé. Pour cela, la fonction `annexe` utilise la valeur de `n`, qu'elle peut modifier, et renvoie un arbre binaire de hauteur `hauteur_max` dont le nombre de nœuds est égal à la valeur de `n` au moment de son appel.

```
1.  def fabrique(h, n):
2.    def annexe(hauteur_max):
3.      if n == 0 :
4.        return arbre_vide()
5.      elif hauteur_max == 0:
6.        n = n - 1
7.        return ...
8.      else:
9.        n = n - 1
10.       gauche = annexe(hauteur_max - 1)
11.       droite = ...
12.       return arbre(gauche, droite)
13.   return annexe(h)
```

6. Recopier sur la copie les lignes 7 et 11 en complétant les points de suspension.