Sujet BAC 10 : Les arbres - Partie 1

1. 2022, Polynésie, J1, Ex. 5

BAC : Construction d'arbres binaires

On manipule ici les arbres binaires avec trois fonctions :

- `est_vide(A)` renvoie `True` si l'arbre binaire `A` est vide, `False` s'il ne l'est pas ;
- Pour un arbre binaire `A` **non vide** :
  - `sous_arbre_gauche(A)` renvoie le sous-arbre à gauche de `A` ;
  - `sous_arbre_droite(A)` renvoie le sous-arbre à droite de `A`.

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

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

- la hauteur de l'arbre vide est 0 ;
- sinon, la hauteur est égale à $1 + M$, où M est la plus grande des hauteurs de ses sous-arbres (à gauche et à droite).

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

```
graph TD
   N0( ) --> N2( )
   N0   --> N1( )
   N2  --> N5( )
   N2  --> N6( )
   linkStyle 3 stroke-width:0px;
   style N6 opacity:0;
```

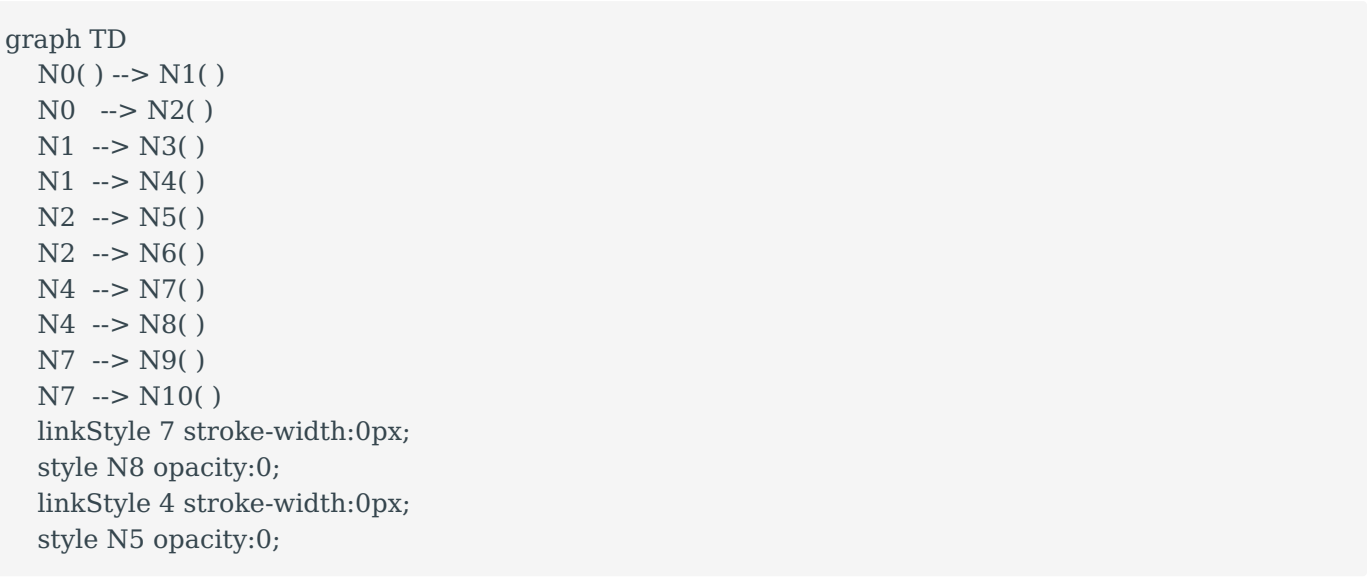
Réponse

Avec cette définition, la hauteur de cet arbre binaire est 3.

1.b. Dessiner sur la copie un arbre binaire de hauteur 5.

Réponse

Avec cette définition, voici un arbre binaire de hauteur \(5\).



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

```
1  Algorithme hauteur(A) :
2      si A vide :
3          renvoyer ...
4      sinon:
5          renvoyer 1 + max(
6              hauteur(sous_arbre_gauche(A)),
7              ...
8          )
```

2. Recopier sur la copie les lignes 3 et 7 en complétant les points de suspension.

Réponse

```
1  Algorithme hauteur(A) :
2      si A vide :
3          renvoyer 0
4      sinon:
5          renvoyer 1 + max(
6              hauteur(sous_arbre_gauche(A)),
7              hauteur(sous_arbre_droite(A)),
8          )
```

On considère un arbre binaire `R` dont on note `G` le sous-arbre à gauche et `D` le sous-arbre à droite. On suppose que `R` est de hauteur \(5\) et `G` de hauteur \(3\).

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

Réponse

Si `D` est égal à l'arbre vide, alors la hauteur de `R` est égale à `1 + hauteur(G)` qui est égal à \(1+3=4\), or `R` est de hauteur \(5\). Contradiction.

Ainsi `D` n'est pas l'arbre vide.

Dans ce cas `1 + max(hauteur(G), hauteur(D))` est égal à \(4\). D'où

- `1 + max(3, hauteur(D))` est égal à \(5\).
- `max(3, hauteur(D))` est égal à \(4\).
- `hauteur(D)` est égal à \(4\).

3.b. Illustrer cette situation par un dessin.

Réponse

- Cet arbre est de hauteur \(5\),
- son sous arbre à gauche est de hauteur \(3\),
- son sous arbre à droite est de hauteur \(4\).

```
graph TD
   N0( ) --> N1( )
   N0   --> N2( )
   N1  --> N3( )
   N1  --> N4( )
   N2  --> N5( )
   N2  --> N6( )
   N4  --> N7( )
   N4  --> N8( )
   N6  --> N9( )
   N6  --> N10( )
   N9  --> N11( )
   N9  --> N12( )
   linkStyle 7 stroke-width:0px;
   style N8 opacity:0;
   linkStyle 4 stroke-width:0px;
   style N5 opacity:0;
```

Soit un arbre binaire non vide de hauteur `h`. On note `n` le nombre de nœuds de cet arbre. On admet que \(h \leqslant n \leqslant 2^h - 1\).

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

Réponse

Dans la question **1.a.**, l'arbre binaire possède $n = 4$ nœuds et a une hauteur $h = 3$.

On a bien $3 \leqslant 4 \leqslant 2^3 - 1$ qui s'écrit aussi $3 \leqslant 4 \leqslant 7$

4.b. Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant `h` nœuds.

Réponse

Il **suffit**, par exemple, de construire un arbre binaire où pour chaque nœud, soit le sous arbre à gauche est vide, soit celui à droite.

- Cela peut être toujours celui à gauche qui est vide, on parle alors d'arbre peigne à droite.
- Cela peut être toujours celui à droite qui est vide, on parle alors d'arbre peigne à gauche.

4.c. Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant $2^h - 1$ nœuds.

Indication : $2^h - 1 = 1+2+4+...+2^{h-1}$.

Réponse

Il **faut**, dans ce cas, construire un arbre binaire complet ; les sous-arbres vides sont tous à la même profondeur.

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 $h < n < 2^h - 1$, et qui renvoie un arbre binaire de hauteur `h` à `n` nœuds.

Pour cela, on utilise les deux fonctions suivantes :

- `arbre_vide()`, qui renvoie un arbre vide ;
- `arbre(gauche, droite)` qui renvoie l'arbre fils à `gauche` et le fils à `droite`.

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.

```
graph TD
   N0( ) --> N2( )
   N0   --> N1( )
   N1 --> N3( )
   N1 --> N4( )
   N2  --> N5( )
   N2  --> N6( )
   linkStyle 5 stroke-width:0px;
   style N6 opacity:0;
   N3 --> N7( )
   N3 --> N8( )
   N4 --> N9( )
   N4 --> N10( )
```

Réponse

```
graph TD
   N0(1) --> N2(9)
   N0   --> N1(2)
   N1 --> N3(3)
   N1 --> N4(6)
   N2  --> N5(10)
   N2  --> N6( )
   linkStyle 5 stroke-width:0px;
   style N6 opacity:0;
   N3 --> N7(4)
   N3 --> N8(5)
   N4 --> N9(7)
   N4 --> N10(8)
```

La fonction `fabrique` ci-dessous a pour but de répondre au problème posé.

🐍 **Script Python**

```python
def fabrique(h, n):
    if n == 0:
        return ...
    else:
        reste = n - 1                  # 1 pour la racine du sous-arbre
        n_gauche = min(reste, ...)       # le plus possible à gauche
        n_droite = reste - n_gauche      # la suite à droite
        return arbre(
            fabrique(..., n_gauche),
            ...
        )
```

6. Recopier sur la copie les lignes 3, 6 et 9 en complétant les points de suspension.

Réponse

```python
def arbre_vide():
    return []

def arbre(gauche, droite):
    return [gauche, droite]

def fabrique(h, n):
    if n == 0:
        return arbre_vide()
    else:
        reste = n - 1                    # 1 pour la racine du sous-arbre
        n_gauche = min(reste, 2**(h-1) - 1)  # le plus possible à gauche
        n_droite = reste - n_gauche          # la suite à droite
        return arbre(
            fabrique(h-1, n_gauche),
            fabrique(h-1, n_droite)
        )

def taille(arbre):
    if arbre == []:
        return 0
    else:
        gauche, droite = arbre
        return 1 + taille(gauche) + taille(droite)

def hauteur(arbre):
    if arbre == []:
        return 0
    else:
        gauche, droite = arbre
        return 1 + max(hauteur(gauche), hauteur(droite))


for h in range(4):
    print("Hauteur", h)
    for n in range(h, 2**h):
        n_sav = n
        arbre_hn = fabrique(h, n)
        print(arbre_hn)
        assert (h, n_sav) == (hauteur(arbre_hn), taille(arbre_hn))
    print()
```

2. 2022, Métropole, J1, Ex. 4

BAC : Somme des valeurs d'un arbre binaire

Cet exercice traite du calcul de la somme d'un arbre binaire. Cette somme consiste à additionner toutes les valeurs numériques contenues dans les nœuds de l'arbre.

L'arbre utilisé dans les parties A et B est le suivant :

```
graph TD
    N0(3) --> N1(6)
    N0    --> N2(2)
    N1    --> N4(7)
    N1    --> N5(4)
    N2    --> N6(9)
    N2    --> N7(1)
```

2.1. Partie A : Parcours d'un arbre

1. Donner la somme de l'arbre précédent. Justifier la réponse en explicitant le calcul qui a permis de l'obtenir.

Réponse

La somme de l'arbre est $3 + 6 + 2 + 7 + 4 + 9 + 1 = 32$

2. Indiquer la lettre correspondante aux noms « racine », « feuille », « nœud », « SAG » (Sous Arbre à Gauche) et « SAD » (Sous Arbre à Droite). Chaque lettre A, B, C, D et E ne devra être utilisée qu'une seule fois.

```
flowchart TD
    subgraph G0 ["Arbre avec des lettres à associer
                 &nbs

        N0(3) --> N1(6)
        N0    --> N2(2)
        subgraph G2 [ ]
            N2    --> N6(9)
            N2    --> N7(1)
        end
        subgraph G3 [ ]
            N1    --> N4(7)
            N1    --> N5(4)
        end

        A>A] -.- N0
        B>B] -.- N1
        C>C] -.- N4
        G3 -.- D>D]
        G2 -.- E>E]
    end
```

Réponse

Lettre	Nom
A	racine
B	nœud
C	feuille
D	SAG
E	SAD

3. Parmi les quatre propositions A, B, C et D ci-dessous, donnant un parcours en largeur de l'arbre, une seule est correcte. Indiquer laquelle.

- **Proposition A** : 7 - 6 - 4 - 3 - 9 - 2 - 1
- **Proposition B** : 3 - 6 - 7 - 4 - 2 - 9 - 1
- **Proposition C** : 3 - 6 - 2 - 7 - 4 - 9 - 1
- **Proposition D** : 7 - 4 - 6 - 9 - 1 - 2 - 3

Réponse

On lit chaque niveau, de la gauche vers la droite.

- 3, puis
- 6, 2, puis
- 7, 4, 9, 1.

La proposition **C** est la bonne.

4. Écrire en langage Python la fonction `somme` qui prend en paramètre une liste de nombres et qui renvoie la somme de ses éléments.

Exemple : `somme([1, 2, 3, 4])` est égale à 10.

Réponse

Plusieurs réponses possibles

Itération sans indice **Itération avec indice**

🐍 **Script Python**

```
def
somme(une_liste):
    resultat = 0
    for un_entier in
une_liste:
        resultat +=
un_entier
    return resultat
```

Recommandée.

🐍 **Script Python**

```
def
somme(une_liste):
    resultat = 0
    for i in
range(len(une_liste)):
        resultat +=
une_liste[i]
    return resultat
```

5. `re)` pourrait se traduire en langage naturel par :

📋 **Pseudo-code**

```
parcourir(A):
    L = liste_vide
    F = file_vide
    enfiler A dans F
    Tant que F n'est pas vide
        défiler S de F
        ajouter la valeur de la racine de S dans L
        Pour chaque sous arbre SA non vide de S
            enfiler SA dans F
    renvoyer L
```

Donner le type de parcours obtenu grâce à la fonction `parcourir`.

Réponse

Si, à un moment du traitement, la file ne contient que des éléments d'un certain niveau, **puis** (éventuellement) du niveau suivant, alors on enfile pendant le traitement des éléments du niveau suivant, ce qui fait que cette propriété est conservée.

Au départ, la propriété est de mise avec un seul élément. Elle le restera pendant tout le parcours, ainsi on traite les éléments niveau par niveau.

Il s'agit d'un **parcours en largeur**.

2.2. Partie B : Méthode « diviser pour régner »

6. Parmi les quatre propositions A,B, C et D ci-dessous, indiquer la seule proposition correcte. En informatique, le principe diviser pour régner est associé à :

- **Proposition A** : diviser une fonction en deux fonctions de plus petit code.
- **Proposition B** : utiliser plusieurs modules
- **Proposition C** : séparer les informations en fonction de leur type
- **Proposition D** : découper un problème initial en sous-problèmes, à résoudre, puis combiner leurs solutions

Réponse

Proposition D

En informatique, diviser pour régner (du latin « *Divide ut imperes* », *divide and conquer* en anglais) est une technique algorithmique consistant à :

- Diviser : découper un problème initial en sous-problèmes ;
- Régner : résoudre les sous-problèmes (récursivement ou directement s'ils sont assez petits) ;
- Combiner : calculer une solution au problème initial à partir des solutions des sous-problèmes.

7. L'arbre présenté dans le problème peut être décomposé en racine et sous arbres :

```
graph TD
    N0(3) --> N1(6)
    N0    --> N2(2)
    subgraph "SAD
                &nbs
```

```
    N2    --> N6(9)
    N2    --> N7(1)
    end
    subgraph "SAG
                 &nbs
    N1    --> N4(7)
    N1    --> N5(4)
    end
```

Indiquer dans l'esprit de « diviser pour régner » l'égalité donnant la somme d'un arbre en fonction de la somme des sous arbres et de la valeur numérique de la racine.

Réponse

`somme(arbre)` est égal à `valeur_racine(arbre) + somme(SAG(arbre)) + somme(SAD(arbre))`

8. Écrire en langage Python une fonction récursive `somme(arbre)`. Cette fonction renvoie la somme de l'arbre passé en paramètre.

Les fonctions suivantes sont disponibles :

- `est_vide(arbre)` : détermine si `arbre` est vide et renvoie un booléen `True` ou `False`.
- `valeur_racine(arbre)` : renvoie la valeur numérique de la racine de `arbre` ;
- `arbre_gauche(arbre)` : renvoie le sous arbre à gauche de `arbre` ;
- `arbre_droite(arbre)` : renvoie le sous arbre à droite de `arbre`.

Réponse

🐍 **Script Python**

```python
def somme(arbre):
    if est_vide(arbre):
        return 0
    else:
        return (
            valeur_racine(arbre)
            + somme(arbre_gauche(arbre))
            + somme(arbre_droite(arbre))
        )
```

3. 2022 Asie J1 - Ex3

BAC : Arbres binaires

Les premiers travaux concernant l'aide à la décision médicale se sont développés pendant les années soixante-dix parallèlement à l'avènement de l'informatique dans le secteur médical. L'arbre de décision est une technique décisionnelle fréquemment employée pour rechercher la meilleure stratégie thérapeutique. L'arbre de décision de cet exercice, présenté ci-dessous, est un arbre binaire que l'on nommera `arb_decision`.

```
graph TD
   N0("Conjonctivie jaune") -->|non| N2("hypercaroténémie")
   N0 --> |oui| N1("bilirubine")
   N1 --> |non| N3("spiénomégalie")
   N1 --> |oui| N4("hépatomégalie")
   N3  -->|non| N5("maladie de Gilbert <br> par défaut <br> de glycuro-conjugaison")
   N3  --> |oui| N6("anémie <br> hémolytique")
   N4 --> |non| N7("hépatie")
   N4 --> |oui| N8("douleur <br> et fièvre")
   N8 --> |non| N9("cancer de la tête <br> du pancréas")
   N8 --> |oui| N10("lithiase <br> du cholédoque")
```

Arbre de décision en présence d'une jaunisse (peau anormalement jaune) chez un patient.

Rappels :

- Un arbre binaire est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément, appelé nœud, porte une étiquette.
- Le nœud initial est appelé racine.
- Chaque nœud d'un arbre binaire possède au plus deux sous-arbres.
- Chacun de ces sous-arbres est un arbre binaire, appelés sous-arbre gauche et sous-arbre droit.
- Un nœud dont les sous-arbres sont vides est appelé une feuille.
- Dans cet exercice, on utilisera la convention suivante : la hauteur d'un arbre binaire ne comportant qu'un nœud est égale à 1.

Dans l'arbre de décision en présence d'une jaunisse chez un patient,

- un nœud représente un symptôme dont le médecin doit étudier la présence ou l'absence ; la réponse ne peut être que oui ou non ;
- le sous-arbre gauche d'un nœud donné décrit la démarche à adopter si le symptôme est absent ;
- le sous-arbre droit d'un nœud donné décrit la démarche à adopter si le symptôme est présent ;
- l'étiquette d'une feuille est la maladie induite par le chemin parcouru.

Questions :

1. Déterminer la taille et la hauteur de l'arbre donné en exemple en introduction (arbre de décision en présence d'une jaunisse).

Réponse

- Taille = 11
- hauteur = 5

2. On choisit d'implémenter un arbre binaire à l'aide d'un dictionnaire.

🐍 **Script Python**

```
arbre_vide = {}
arbre = {'etiquette': 'valeur' ,
        'sag': sous_arbre_gauche ,
        'sad': sous_arbre_droit }
```

Le code ci-dessous représente un arbre selon le modèle précédent.

🐍 **Script Python**

```
{'etiquette' :'a',
   'sag':{'etiquette' :'b',
      'sag':{},
      'sad' : {'etiquette' : 'd',
         'sag' : {},
         'sad' : {}}},
   'sad': {'etiquette' :'f',
      'sag' : {'etiquette' : 'g',
         'sag' : {},
         'sad' : {}},
      'sad' : {} }}
```

a. À quelle représentation graphique correspond la structure implémentée ci-dessus ?

- arbre 1 :

```
graph TD
    A("a") --> B("b")
    B --> d("d")
    B --> d1(" ")
    A --> F("f")
    F --> G("g")
    F --> G1(" ")
    linkStyle 2 stroke-width:0px;
    style d1 opacity:0;
    linkStyle 5 stroke-width:0px;
    style G1 opacity:0;
```

- arbre 2 :

```
graph TD
    A("a") --> B("b")
    B --> d(" ")
    B --> d1("d")
    A --> F("f")
```

```
    F --> G("g")
    F --> G1(" ")
    linkStyle 1 stroke-width:0px;
    style d opacity:0;
    linkStyle 5 stroke-width:0px;
    style G1 opacity:0;
```

- arbre 3 :

```
graph TD
    A("a") --> B("b")
    B --> d(" ")
    B --> d1("d")
    A --> F("f")
    F --> G(" ")
    F --> G1("g")
    linkStyle 1 stroke-width:0px;
    style d opacity:0;
    linkStyle 4 stroke-width:0px;
    style G opacity:0;
```

Réponse

Arbre 2

b. Représenter graphiquement l'arbre correspondant au code ci-dessous.

🐍 **Script Python**

```
{'etiquette' :'H',
'sag':{ 'etiquette' :'G',
'sag': {'etiquette' : 'E',
'sag' : {},
'sad' : {}},
'sad' : {'etiquette' : 'D',
'sag' : {},
'sad' : {'etiquette' : 'B',
'sag' : {},
'sad' : {} }}},
'sad': {'etiquette' :'F',
'sag' : {'etiquette' : 'C',
'sag' : {},
'sad' : {'etiquette' : 'A',
'sag' : {} ,
'sad' : {}}},
'sad' : {} }}
```

Réponse

```
graph TD
    H("H") --> G("G")
    G --> E("E")
    G --> D("D")
    H --> F("F")
    F --> C("C")
    F --> F1(" ")
    D --> D1(" ")
    D --> B("B")
    C --> C1(" ")
    C --> A("A")
    linkStyle 5 stroke-width:0px;
    style F1 opacity:0;
    linkStyle 6 stroke-width:0px;
    style D1 opacity:0;
    linkStyle 8 stroke-width:0px;
    style C1 opacity:0;
```

3. La fonction parcours(arb) ci-dessous permet de réaliser le parcours des nœuds d'un arbre binaire arb donné en argument.

🐍 **Script Python**

```
def parcours(arb):
    if arb == {}:
        return None
    parcours(arb['sag'])
    parcours(arb['sad'])
    print(arb['etiquette'])
```

a. Donner l'affichage après l'appel de la fonction parcours avec l'arbre dont une représentation graphique est ci-dessous.

```
graph TD
    A("a") --> B("b")
    B --> d("d")
    B --> d1(" ")
    A --> F("f")
    F --> G("g")
    F --> G1(" ")
    linkStyle 2 stroke-width:0px;
    style d1 opacity:0;
    linkStyle 5 stroke-width:0px;
    style G1 opacity:0;
```

Réponse

Parcours suffixe : d-b-g-f-a

b. Écrire une fonction `parcours_maladies(arb)` qui n'affiche que les feuilles de l'arbre binaire non vide arb passé en argument, ce qui correspond aux maladies possiblement induites par l'arbre de décision.

Réponse

🐍 **Script Python**

```
def parcours_maladies(arb):
    if arb=={}:
        return None
    parcours_maladies (arb['sag'])
    parcours_maladies (arb['sad'])
    if arb['sag'] == {} and arb['sad'] == {}:
        print(arb['etiquette'])
```

4. On souhaite maintenant afficher l'ensemble des symptômes relatifs à une maladie. On considère la fonction `symptomes(arbre, mal)` avec comme argument arbre un arbre de décision binaire et mal le nom d'une maladie.

L'appel de cette fonction sur l'arbre de décision `arb_decision` de l'introduction fournit les affichages suivants.

🐍 **Script Python**

```
>>> symptomes(arb_decision, "anémie hémolytique")
symptômes de anémie hémolytique
splénomégalie
pas de bilirubine
conjonctive jaune
```

Pour cela, on modifie la structure précédente en ajoutant une clé `surChemin` qui sera un booléen indiquant si le nœud est sur le chemin de la maladie.
La clé `surChemin` est initialisée à False pour tous les nœuds.

🐍 **Script Python**

```
arbre = {'etiquette': 'valeur' ,
       'surChemin': False ,
       'sag': 'sous-arbre gauche' ,
       'sad': 'sous-arbre droit' }
```

Recopier et compléter les lignes 6, 8, 14 et 18 du code suivant sur votre copie.

```
1  def symptomes(arb, mal):
2      if arb['sag'] != {} :
3          symptomes(arb['sag'],mal)
4
5      if arb['sad'] != {} :
6          symptomes(...)
7
8      if ... ... ... ... :
9          arb['surChemin'] = True
10         print('symptômes de', arb['etiquette'],':')
11
12     else :
13         if arb['sad'] != {} and arb['sad']['surChemin'] :
14             print(...)
15             arb['surChemin'] = True
16
17         if arb['sag'] != {} and arb['sag']['surChemin'] :
18             print(...)
19             arb['surChemin'] = True
```

Réponse

```
1  def symptomes(arb, mal):
2      if arb['sag'] != {}:
3          symptomes(arb['sag'], mal)
4
5      if arb['sad'] != {}:
6          symptomes(arb['sad'], mal)
7
8      if arb['etiquette'] == mal:
9          arb['surChemin'] = True
10         print('symptômes de', arb['etiquette'],':')
11
12     else :
13         if arb['sad'] != {} and arb['sad']['surChemin']:
14             print(arb['etiquette'])
15             arb['surChemin'] = True
16
17         if arb['sag'] != {} and arb['sag']['surChemin']:
18             print('pas de ',arb['etiquette'])
19             arb['surChemin'] = True
```