ExerciceType Bac
NSI *Terminale*

Épreuve écrite
Année .....

Exercice 8

Arbres binaires - Parcours - récursivité

On considère l'arbre binaire ci-contre

- Le noeud long a pour valeur `'LONG'`
- Son fils gauche est le noeud algo.
- Son fils droit est le noeud golf.
- Son père est le noeud loin.

? Question 1:

1. On effectue un parcours en largeur de l'arbre de l'exemple ci-dessus depuis la racine. Donner la liste des sommets obtenus par ce parcours.
   INFO LOIN PION POIL LONG PAIN LION PILE POLE ALGO GOLF
2. Donner la liste des sommets obtenus dans un parcours en profondeur préfixe.
   INFO LOIN POIL PILE POLE LONG ALGO GOLF PION PAIN LION
3. Donner la liste des sommets obtenus dans un parcours en profondeur infixe.
   PILE POIL POLE LOIN ALGO LONG GOLF INFO PAIN PION LION
4. Donner la liste des sommets obtenus dans un parcours en profondeur postfixe (ou suffixe).
   PILE POLE POIL ALGO GOLF LONG LOIN PAIN LION PION INFO

On choisit de représenter cet arbre à l'aide d'une liste de listes.

Chaque noeud de l'arbre est représenté par une liste.

- Un tuple qui contient l'étiquette du noeud et l'étiquette du père du noeud.
- Deux listes pour les enfants gauche et droit du noeud.
- Notre arbre est donc la liste `info`

```
pile=[("PILE","POIL"),[],[]]
pole=[("POLE","POIL"),[],[]]
algo=[("ALGO","LONG"),[],[]]
golf=[("GOLF","LONG"),[],[]]
pain=[("PAIN","PION"),[],[]]
lion=[("LION","PION"),[],[]]
poil=[("POIL","LOIN"),pile,pole]
long=[("LONG","LOIN"),algo,golf]
loin=[("LOIN","INFO"),poil,long]
pion=[("PION","INFO"),pain,lion]
info=[("INFO",None),loin,pion]
```

Voici une fonction parcours.

```
def parcours(arbre):
    f = File()
    f.enfiler(arbre)
    while f.est_vide() == False:
        arbre=f.defiler()
        print(arbre[0][0],end = " ")
        if arbre[1] != []:
            f.enfiler(arbre[1])
        if arbre[2] != []:
            f.enfiler(arbre[2])
```

On dispose d'une classe `File` et de ses primitives usuelles:

- `f= File()` crée une file `f`
- `f.est_vide()` renvoie `True` si la file est vide
- `f.enfiler(val)` met `val` dans la file
- `f.defiler()` supprime et renvoie la tête de la file

Question 2:

1. De quel type de parcours s'agit-il ? en largeur
2. Quel affichage produit l'appel suivant : `parcours(info)` ?
   INFO LOIN PION POIL LONG PAIN LION PILE POLE ALGO GOLF
3. Recopier la fonction `parcours` et modifier là pour qu'elle renvoie une liste contenant les tuples `(étiquette,père)` de chaque noeud de l'arbre.

Question 3:

Dans cette question on suppose que la fonction `parcours` a été modifié comme dans la question 2.
Son appel a renvoyé la `liste_noeud`:
`liste_noeud` = [('INFO', None), ('LOIN', 'INFO'), ('PION', 'INFO'), ('POIL', 'LOIN'), ('LONG', 'LOIN'), ('PAIN', 'PION'), ('LION', 'PION'), ('PILE', 'POIL'), ('POLE', 'POIL'), ('ALGO', 'LONG'), ('GOLF', 'LONG')]
Écrire une fonction `pere` prenant en paramètre la liste ci-dessus et l'étiquette d'un noeud et qui renvoie l'étiquette du père du noeud.

Question 4:

Voici une fonction qui est sensée donner le chemin depuis la racine jusqu'au noeud passé en paramètre:

```
def chemin(cible):
    while cible!=None:
        print(cible,end=" ")
        cible=pere(liste\_noeud,cible)
```

1. Quel affichage produit cet appel : `chemin("ALGO")`
   ALGO LONG LOIN INFO
2. Écrire une version récursive de cette fonction.

```
def parcours(arbre):
    f=File()
    lst=[]
    f.enfiler(arbre)
    while f.est_vide()==False:
        arbre=f.defiler()
        lst.append(arbre[0])
        if arbre[1] !=[]:
            f.enfiler(arbre[1])
        if arbre[2] !=[]:
            f.enfiler(arbre[2])
    return lst
def pere(liste_noeud,enfant):
    for el in liste:
        if el[0]==enfant:
            return el[1]

def cheminr(cible):
```

```
    print(cible,end=" ")
    if pere(lst2,cible)==None:
        return
    else:
        p=pere(lst2,cible)
        cheminr(p)
```