RÉSUMÉ 19 - ABR DES POKEMONS

Lien vers l'activité : ABR des Pokemons
www.infoforall.fr - Dernière modif. : 30 01 2021

1 - FONCTIONS D'INTERFACE UTILISÉES

Les fonctions liées aux Noeuds :

1. `nvND(cle:Cle, data:Data) -> Noeud` : on crée un nouveau noeud et son élément attaché. Ce n'est pas une fonction d'interface de l'arbre mais on a besoin au moins de pouvoir créer un Noeud.
2. `cle(noeud:Noeud) -> Cle` : renvoie la clé ou l'étiquette du noeud.
3. `data(noeud:Noeud) -> Data` : renvoie les données associées au noeud.

Les fonctions liées à l'Arbre Binaire de Recherche lui-même :

1. `nvAV() -> Arbre` : on le note ainsi pour dire **nouvelArbreVide** : on crée un nouvel ARBRE BINAIRE vide.
2. `nvAB(r:Noeud, g:Arbre, d:Arbre) -> Arbre` : on crée un nouvel ARBRE BINAIRE dont la racine est **r** et dont les sous-arbres sont **g** et **d** fournis.
3. `estArbreVide(arbre:Arbre) -> bool` : True si l'arbre est un arbre vide.
4. `racine(arbre:Arbre) -> Noeud` : renvoie le noeud jouant le rôle de la racine pour cet arbre.
5. `gauche(arbre:Arbre) -> Arbre` : renvoie le sous-arbre gauche de **arbre**. On obtient bien un Arbre. Si vous voulez le noeud gauche, il faudra appliquer en plus la fonction **racine**.
6. `droite(arbre:Arbre) -> Arbre` : renvoie le sous-arbre droit de **arbre**.
7. `inserer_gauche(arbre:Arbre, noeud:Noeud) -> None` : modifie l'arbre en plaçant le noeud en tant que fils-gauche de la racine de l'arbre.
8. `inserer_droite(arbre:Arbre, noeud:Noeud) -> None` : modifie l'arbre en plaçant le noeud en tant que fils-droite de la racine de l'arbre.

2 - RECHERCHE DANS UN ABR

Idée générale : On part à gauche ou à droite en fonction de la comparaison de la clé cherchée et de la clé de la racine, jusqu'à trouver la bonne clé ou un arbre vide.

Version itérative utilisant les fonctions d'interface :

```
1  def recherche(arbre:ABR, cr:Cle) -> Noeud:
2      '''Recherche un noeud portant la bonne clé et renvoie sa référence'''
3      while not estArbreVide(arbre):
4          if cr == cle(racine(arbre)):
5              return racine(arbre)
6          elif cr < cle(racine(arbre)):
7              arbre = gauche(arbre)
8          elif cr > cle(racine(arbre)):
9              arbre = droite(arbre)
```

Version récursive de la recherche :

```
1  def recherche(arbre:ABR, cr:Cle) -> Noeud:
2      '''Recherche un noeud portant la bonne clé et renvoie sa référence'''
3      if estArbreVide(arbre):  # Condition d'arrêt et cas de base
4          return None
5      elif cr == cle(racine(arbre)):  # Condition d'arrêt et cas de base
6          return racine(arbre)
7      elif cr < cle(racine(arbre)):  # Appel récursif
8          return recherche(gauche(arbre))
9      elif cr > cle(racine(arbre)):  # Appel récursif
10         return recherche(droite(arbre))
```

3 - INSERTION D'UN NOEUD DANS UN ABR

Idée générale : On part à gauche ou à droite en fonction de la comparaison entre la clé du noeud à insérer et la clé de la racine, jusqu'à trouver un arbre vide. **A chaque déplacement, on mémorise également le parent** de la position actuelle pour savoir aufinal où insérer notre noeud.

```python
def inserer_noeud_dans_ABR(arbre:ABR, nd:Noeud) -> None:
    '''Insére le noeud dans l'ABR'''

    arbre_parent = None
    while not estArbreVide(arbre):
        arbre_parent = arbre
        if cle(nd) < cle(racine(arbre)):
            arbre = gauche(arbre)
        else:
            arbre = droite(arbre)
    if cle(nd) < cle(racine(arbre_parent)):
        inserer_gauche(arbre_parent, nvABR(nd))
    else:
        inserer_droite(arbre_parent, nvABR(nd))
```

On notera que cette insertion n'est pas optimisée et peut mener à la création d'un arbre pas très équilibré. La recherche de l'équilibre sera vue dans le supérieur.

4 - LECTURE DANS L'ORDRE CROISSANT DES CLÉS

Un ABR permet d'obtenir assez facilement une liste triée par valeur des clés, la valeur maximale et la valeur minimale.

- Pour lire les clés d'un ABR **dans l'ordre croissant**, il suffit de faire une lecture **infixe** (GRD).

```python
def parcours_infixe(arbre:Arbre) -> None:
    '''Exploration (et affichage) en profondeur en infixe GRD'''
    if not estArbreVide(arbre):
        parcours_infixe(gauche(arbre))
        print(cle(racine(arbre)))
        parcours_infixe(droite(arbre))
```

- Pour trouver la **valeur minimale**, il suffit d'aller à gauche jusqu'à trouver une feuille. On mémorise et transfère le minimum trouvé jusqu'à présent.

```python
def cle_min(arbre, minimum=None):
    if estArbreVide(arbre):
        return minimum
    else:
        minimum = cle(racine(arbre))
        return cle_min(gauche(arbre), minimum)
```

- Pour trouver la **valeur maximale**, il suffit d'aller à ... droite.

```python
def cle_max(arbre, maximum=None):
    if estArbreVide(arbre):
        return maximum
    else:
        maximum = cle(racine(arbre))
        return cle_max(droite(arbre), maximum)
```

5 - POO

Si on sait que l'implémentation de l'arbre et des noeuds se fait via des objets dont les attributs se nomment **gauche**, **droite**..., On peut alors utiliser un code plus "clair" mais obsolète si on change l'implémentation.

```python
def inserer_noeud_dans_ABR(arbre:ABR, nd:Noeud) -> None:
    '''Insére le noeud dans l'ABR'''

    arbre_parent = None
    while not arbre == None :
        arbre_parent = arbre
        if nd.cle < arbre.racine.cle:
            arbre = arbre.gauche
        else:
            arbre = arbre.droite
    if nd.cle < arbre_parent.racine.cle:
        inserer_gauche(arbre_parent, ABR(nd))
    else:
        inserer_droite(arbre_parent, ABR(nd))
```