ITI3

Python for Newbies

Durée : *2h*
Documents autorisés : **accès au site Web de Python**

Informations

Ceci est un examen Blanc pour vous entrainer à l'examen final. Pour obtenir votre note, vous pouvez executer le script `calculer_note.sh`. Cette note est fonction du nombre de tests unitaires (boite noire et boite blanche) qui passent (70%) et le respect des bonnes pratiques de codage, vérifiées par pylint (30%). Tous les paramètres formels des fonctions/méthodes devront être annotées.

Un module proposant la classe `Tas`

L'objectif de cet examen est le développement d'un module Python (`tas.py`) proposant la classe `Tas`. Une instance de cette classe a les propriétés suivantes :
— c'est une collection d'éléments que l'on va pouvoir comparer entre eux à l'aide d'une fonction `cle` qui permet d'obtenir le critère de comparaison (par défaut la fonction identité) ;
— c'est un tas : il est possible d'obtenir le plus grand élément en $O(1)$ et d'ajouter ou de retirer un élément en $O(\log n)$.
De plus cette classe est générique, paramétrée par le type `T` [1].

Ce module déclarera aussi trois classes exceptions :
— `TasErreur` qui est la classe mère des deux autres ;
— `ElementNonComparableErreur` qui est levée lorsqu'un élément n'est pas comparable ;
— `TasVideErreur` qui est levée lorsque l'on veut obtenir ou retirer le plus grand élément d'un tas vide.

Voici les caractéristiques (à vous d'identifier les méthodes à développer) de la classe `Tas` :
— on doit pouvoir créer un tas en lui passant optionnellement (dans l'ordre des paramètres formels) :
  — une succession d'éléments : paramètre formel `elements` de type `Iterable` qui a pour valeur par défaut un tuple vide ;
  — une fonction qui indique comment comparer deux éléments : paramètre formel `cle` de type `Callable[[T], Any]` qui a pour valeur par défaut la fonction `lambda x: x`.
— on doit savoir à l'aide de la propriété `est_vide` si le tas est vide ;
— on doit pouvoir utiliser la méthode `ajouter` pour ajouter un élément (paramètre formel `element`). Si l'élémént n'est pas comparable avec les éléments déjà présents, ou s'il n'est pas comparable avec d'autres instances de son type, une exception `ElementNonComparableErreur` est levée ;
— on doit pouvoir utiliser la méthode publique `retirer` pour retirer et obtenir le plus grand élément. Si le tas est vide, une exception `TasVideErreur` est levée.
— on doit pouvoir obtenir le plus grand élément sans le retirer à l'aide de la propriété `element`. Si le tas est vide, une exception `TasVideErreur` est levée ;
— on doit pouvoir obtenir le nombre d'éléments du tas à l'aide de la fonction standard `len` ;
— on doit pouvoir savoir si un élément est présent dans le tas à l'aide de l'opérateur `in` ;
— on doit pouvoir itérer le tas. Les éléments obtenus vont du plus grand au plus petit ;
— on doit pouvoir obtenir une représentation informelle tel que que son affichage (fonction `print`) est un arbre binaire complet tel que les deux fils d'un noeud de l'arbre sont sur les lignes suivantes (le fils gauche puis le fils droit) mais décalés du bord gauche d'un espace de plus. Par exemple :

1. La version de Python, la 3.10.2, installée en salle machine nécessite d'utiliser le type `Generic` du module `typing` pour cette déclaration

```
>>> tas = Tas([3, 5, 2, 8, 7, 1])
>>> print(tas)
8
 7
  3
  5
 2
  1
>>> str(tas)
'8\n 7\n 3\n  5\n 2\n 1\n'
```

— on doit pouvoir obtenir une représentation formelle, par exemple :

```
>>> tas = Tas([3, 5, 2, 8, 7, 1])
>>> repr(tas)
'<Tas object 140465347204800>'
```

140465347204800 étant la référence de l'objet.

Conception d'un tas

Comme vu dans le cours « Algorithmique avancé et programmation C », un tas est un arbre binaire complet tel que :

— l'élément se trouvant dans un nœud est plus grand que les éléments se trouvant dans le fils gauche et le fils droit ;

— le fils gauche et le fils droit sont des tas.

La figure 1 issue de wikipédia présente un tas. Cette même figure montre aussi comment représenter un tas à l'aide d'un tableau :

— la racine d'un tas a pour indice 0 dans le tableau ;

— le nœud d'un tas représenté par la case d'indice i du tableau a pour fils gauche le nœud se trouvant à la case $2i + 1$ et pour fils droit le nœud se trouvant à la case $2i + 2$.

L'algorithme d'ajout d'un élément dans un tas contenant n éléments consiste à positionner cet élément dans la case d'indice n et de remonter cet élément au niveau du nœud père (à l'indice $n - 1$ *div* 2) tant que la valeur contenue dans ce nœud père est plus petite que l'élément à ajouter. La méthode `_remonter` implante l'algorithme de cette dernière action.

L'algorithme de suppression du plus grand élément consiste à copier le dernier élément du tableau en première case, de supprimer ce dernier élément du tableau et descendre l'élément se trouvant à la racine à sa bonne position. La méthode `_descendre` implante l'algorithme de cette dernière action.

Les attributs privés d'un tas se nomme `_elements` et `_cle`.

Travail à réaliser

Vous devez compléter le fichier `tas.py` de façon à ce que tous les tests unitaires passent en respectant les bonnes pratiques de codage (respect de la Pep 8 et utilisation des annotations).

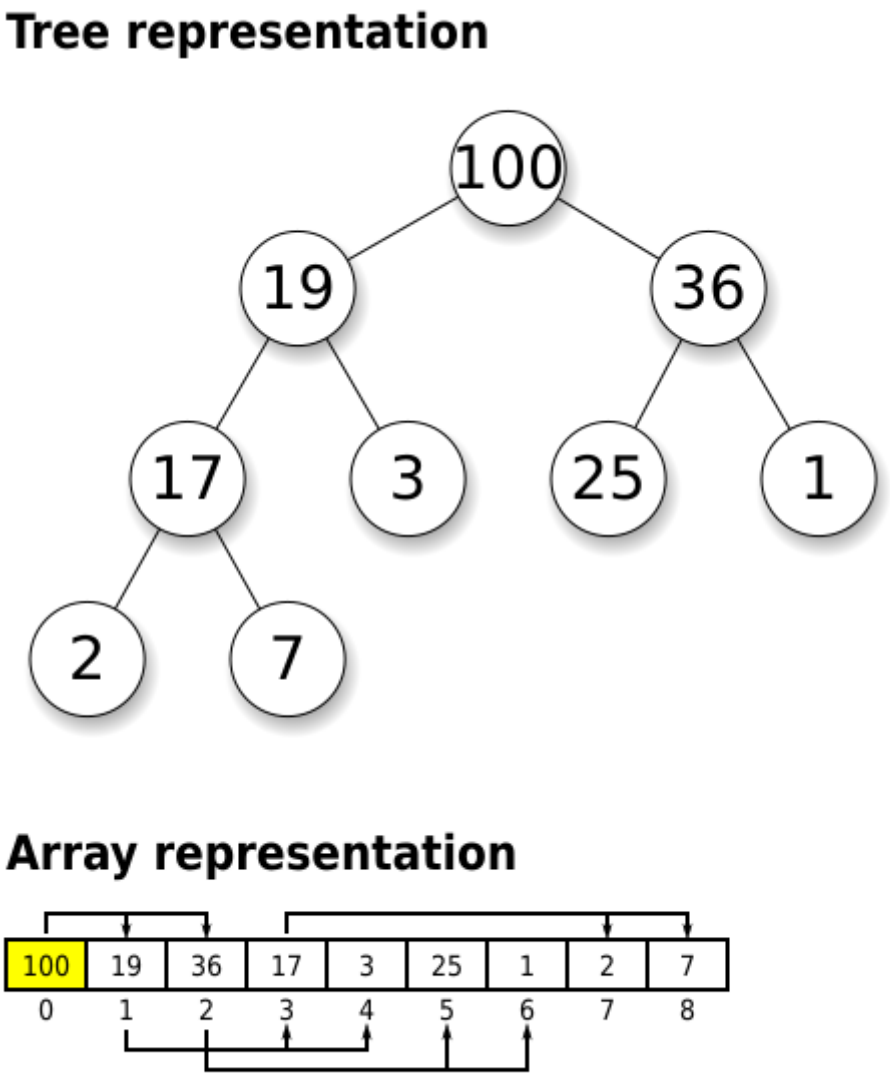

FIGURE 1 – Un tas et sa représentation à l'aide d'un tableau (source Wikipédia)