Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3
Epita

9 novembre 2021 - 9 : 30

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - **Tout code `Python` non indenté ne sera pas corrigé.**
  - Tout ce dont vous avez besoin (classes, fonctions, méthodes) est indiqué en **annexe** !
  - Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
- ☐ Durée : 2h00

Exercice 1 (Graphes et composantes... – *5 points*)

Soit le graphe `G=<S,A>` orienté avec :

```
  S={1,2,3,4,5,6,7,8,9}
et A={(1,2),(1,6),(2,3),(2,5),(3,1),(3,4),(3,5),(4,5),(4,8),(6,2),(6,5),
      (7,5),(7,6),(7,8),(8,5),(8,9),(9,4),(9,7)}
```

1. Remplir le tableau des demi-degrés intérieurs des sommets de G.
2. Donner les sommets du graphe G dans l'ordre de rencontre *préfixe* en partant du sommet 3 (choisir les sommets en ordre croissant).
3. Le graphe G est-il fortement connexe ?
4. Si NON, combien possède-t-il de composantes fortement connexes ?
5. S'ils existent, quels sont les sommets de `G` de degré égal à 0 ? S'il n'en existe pas, répondre 0.

Exercice 2 (Famille nombreuse – *4 points*)

Écrire la fonction `morechildren(`T`)` qui vérifie si chaque nœud interne de l'arbre T a strictement plus de fils que son père, avec l'implémentation *premier fils - frère droit*.

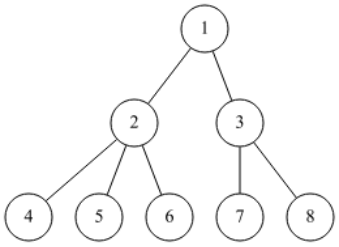

FIGURE 1 – Arbre `T1`

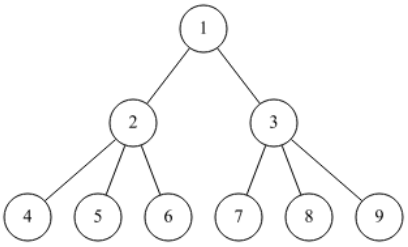

FIGURE 2 – Arbre `T2`

Exemples d'applications avec les arbres des figures 1 (`T1`) et 2 (`T2`) :

```
>>> morechildren(T1)
False
>>> morechildren(T2)
True
```

Exercice 3 (Décroissant – *4 points*)

Écrire la fonction `decrease(`B`)` qui construit la liste des clés du B-arbre B en ordre décroissant.

Exemple d'application avec le B-arbre `B1` de la figure 3 :

```
>>> decrease(B1)
[51, 40, 38, 29, 25, 23, 21, 17, 14, 10, 7, 3]
```

Exercice 4 (B-arbre : insertions et supression – *3 points*)

Pour chaque question, utiliser le principe "à la descente" vu en td (hors bonus). Dessiner uniquement l'arbre final.

1. Dessiner l'arbre après insertions successives des valeurs 11, 32, 20 dans l'arbre de la figure 3.

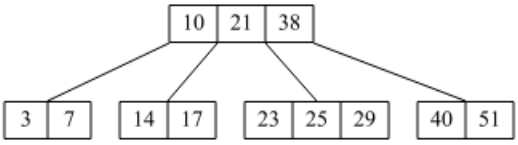

FIGURE 3 – B-arbre `B1` pour insertion, degré 2

2. Dessiner l'arbre après suppression de la valeur 15 dans l'arbre de la figure 4.

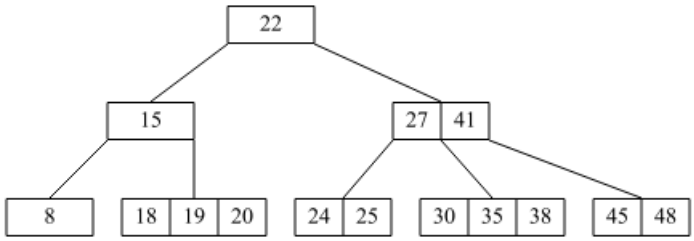

FIGURE 4 – B-arbre `B2` pour suppression, degré 2

Exercice 5 (Mystery – *4 points*)

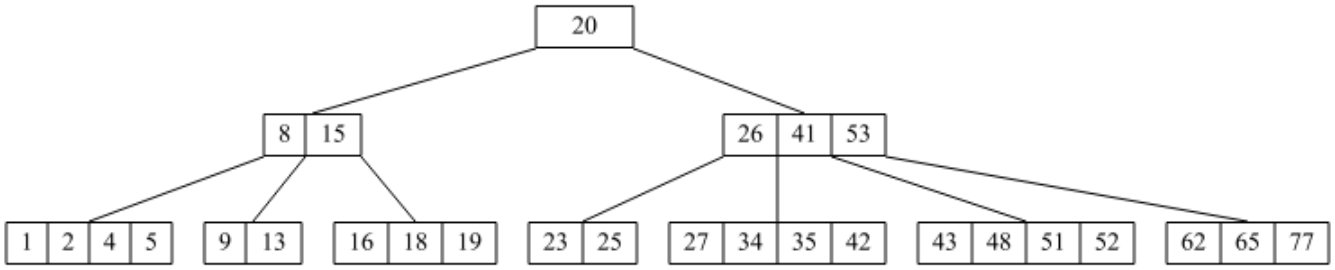

FIGURE 5 – Arbre `B3`

Soit la fonction `mystery` définie ci-dessous :

```
1  def mystery(B, a, b):
2      if B.keys[0] < a or B.keys[B.nbkeys-1] > b:
3          return False
4      else:
5          for i in range(B.nbkeys-1):
6              if B.keys[i] >= B.keys[i+1]:
7                  return False
8          if B.children == []:
9              return True
10         else:
11             i = 0
12             while i < B.nbkeys and mystery(B.children[i], a, B.keys[i]):
13                 a = B.keys[i]
14                 i += 1
15             return i == B.nbkeys and mystery(B.children[B.nbkeys], a, b)
```

1. Pour chacun des appels suivants :
   — quel est le résultat retourné ?
   — combien d'appels à `mystery` ont été effectués ?

   (a) `mystery(B2, 0, 92)` avec `B2` l'arbre de la figure 4

   (b) `mystery(B3, 0, 20)` avec `B3` l'arbre de la figure 5

   (c) `mystery(B3, 1, 99)` avec `B3` l'arbre de la figure 5

2. Soient `B` un arbre (class `BTree`) non vide contenant des entiers, et `a` et `b` deux valeurs entières telles que `a` $<$ `b`.
   Que fait la fonction `mystery(B, a, b)` ?

Annexes

Les arbres généraux

Les arbres (généraux) manipulés ici sont les mêmes qu'en td.

Implémentation *premier fils - frère droit*

- `B` : classe `TreeAsBin`
- `B.key`
- `B.child` : le premier fils
- `B.sibling` : le frère droit

B-Trees

Les B-arbres manipulés ici sont les mêmes qu'en td.
- L'arbre vide est `None`
- L'arbre non vide est un objet de la class `BTree`, que l'on suppose importée
  - `B.degree` est le degré (l'ordre) des B-arbres que l'on manipule : c'est une constante donnée !
  - `B.keys` : liste des clés
  - `B.nbkeys` = `len(B.keys)`
  - `B.children` : liste des fils (`[]` pour les feuilles)
  - `B = BTree(key_list, child_list)` construit un nouvel arbre

Autres fonctions et méthodes autorisées

Comme d'habitude : `len`, `range`, `min`, `max`, `abs`.

En plus, sur les listes :

```
>>> help(list.insert)
...    L.insert(index, object) -- insert object before index

>>> help(list.pop)
...    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.

>>> help(list.append)
...    L.append(object) -> None -- append object to end
```

Vos fonctions

Vous pouvez également écrire vos propres fonctions à condition qu'elles soient documentées : donnez leurs spécifications (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction doit être celle qui répond à la question.