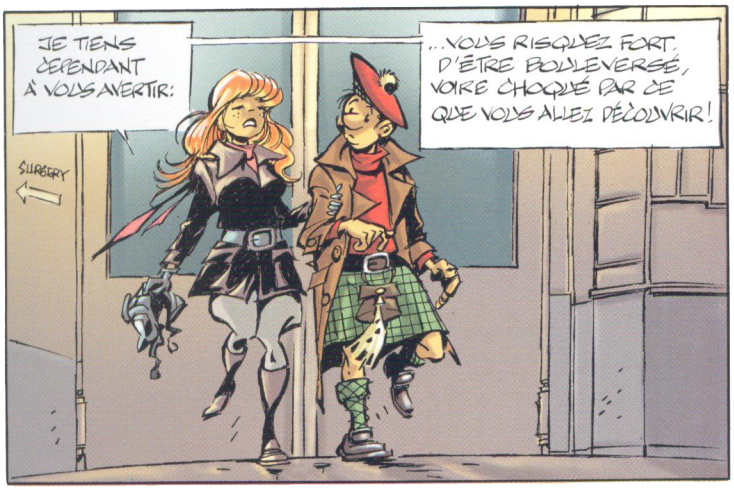
Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3#
Epita

10 mars 2020 - 14h45

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 (Hachages – *2 points*)

Supposons l'ensemble de clés suivant E={beck, cale, clapton, hendrix, hooker, king, richards, vaughan, winter, young} ainsi que le tableau 1 des valeurs de hachage associées à chaque clé de cet ensemble E. Ces valeurs sont comprises entre 0 et 10 ($m = 11$).

beck	10
cale	10
clapton	4
hendrix	5
hooker	6
king	8
richards	3
vaughan	8
winter	1
young	5

TABLE 1 – Valeurs de hachage

Représenter la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans l'ordre de la table 1 (de `beck` jusqu'à `young`) :

1. dans le cas du hachage linéaire avec un coefficient de décalage $d = 3$.
2. dans le cas du hachage avec chaînage séparé.

Exercice 2 (Dessiner c'est gagner – *2 points*)

Soit le graphe `G = <S, A>` orienté avec :
- `S = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`
- `A = {(1,2), (1,6), (1,7), (2,3), (2,6), (3,1), (3,5), (4,3), (4,8), (4,9), (4,10), (5,1), (7,6), (8,5), (8,10), (10,9)}`

1. Représenter graphiquement le graphe `G`.
2. Donner le tableau `DemiDegréIntérieur` tel que $\forall\ i \in [1, Card(S)]$, `DemiDegréIntérieur[i]` soit égal au demi-degré intérieur de `i` dans `G`.

Exercice 3 (Égalité – *5 points*)

Écrire la fonction `same(`T`,` B`)` qui vérifie si T, un arbre général en représentation "classique" et B, un arbre général en représentation *premier fils - frère droit*, sont identiques : ils contiennent les mêmes valeurs dans les mêmes nœuds.

Exercice 4 (Mesure sur les B-arbres – *4 points*)

Dans cet exercice on se propose de mesurer la *qualité* d'un B-arbre.

Écrire la fonction `occupation(B)` qui renvoie un réel correspondant au **nombre moyen de clés par nœud** (nombre de clés / nombre de nœuds) dans le B-arbre `B`. La fonction retourne 0 si l'arbre est vide.

Exercice 5 (B-Arbres : suppression du minimum – *6 points*)

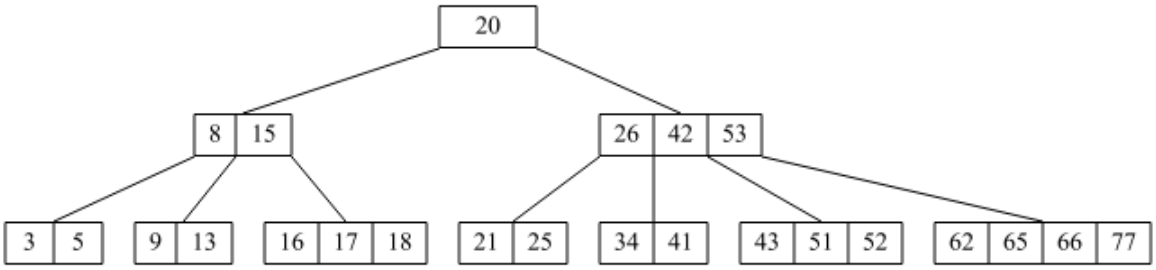


FIGURE 1 – B-tree

1. L'arbre de la figure 1 est un B-arbre. Quel est son degré ?
2. En utilisant le principe "à la descente", dessiner l'arbre après suppression de la valeur 3.
3. Écrire la fonction récursive `__delmin(`B`)` qui supprime la valeur minimum du B-arbre non vide B. La fonction retourne la valeur supprimée.
   - ▷ Utiliser le principe de précaution (à la descente).
   - ▷ Utiliser les fonctions données en annexe !
   - ▷ La fonction `__delmin` sera appelée par la fonction suivante, qui retourne l'arbre ainsi que la valeur supprimée :

```
def del_min(B):
    """
    delete the minimum value of a BTree
    return (the tree and the value deleted, None if no minimum)
    """
    x = None
    if B:
        x = __delmin(B)
        if B.nbkeys == 0:
            if B.children:
                B = B.children[0]
            else:
                B = None
    return (B, x)
```

Annexes

Les arbres généraux

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

Implémentation classique

- `T` : classe `Tree`
- `T.key`
- `T.children` : listes des fils (`[]` pour les feuilles)
- `T.nbchildren` = `len(T.children)`

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.

Rappel :

- 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)

Fonctions données

t est le degré des B-arbres.

- La fonction `leftRotation (`B`, `i`)` effectue une rotation du fils $i+1$ vers le fils i de B :
  - L'arbre B existe, son fils i existe et sa racine n'est pas un $2t$-nœud,
  - le fils $i+1$ existe et sa racine n'est pas un t-nœud.
- La fonction `rightRotation (`B`, `i`)` effectue une rotation du fils $i-1$ vers le fils i de B :
  - l'arbre B existe, son fils i existe et sa racine n'est pas un $2t$-nœud,
  - le fils $i-1$ existe et sa racine n'est pas un t-nœud.
- La fonction `merge (`B`, `i`)` fusionne les fils i et $i+1$ de l'arbre B :
  - l'arbre B existe et sa racine n'est pas un t-nœud,
  - ses fils i et $i+1$ existent et leurs racines sont des t-nœuds.

Autres fonctions et méthodes autorisées

Comme d'habitude : `len`, `range`.

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, 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.