Algorithmique
Correction Contrôle n° 3 (C3)

Info-spé S3# – Epita

6 mars 2018 - 14 : 15

Solution 1 **(Hachages –** ***2 points*****)**

Hachage avec chaînage séparé (voir figure 1),

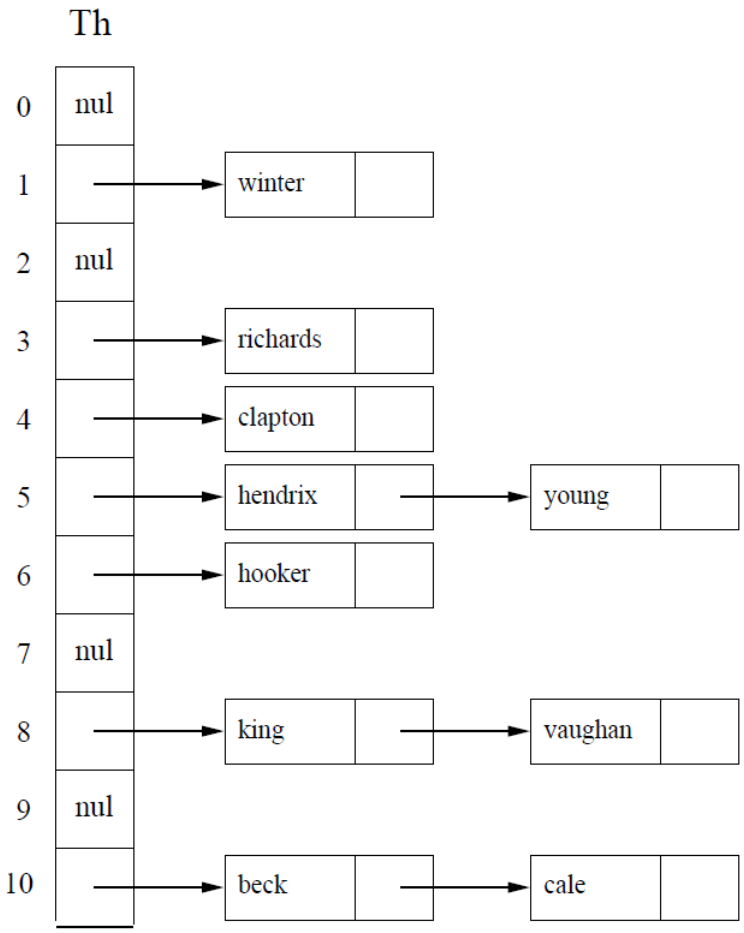

Figure 1 – Hachage avec chaînage séparé.

Solution 2 **(Hachage : Tableaux valides –** ***3 points*****)**

Les tableaux ne pouvant pas résulter d'une insertion quelconque des clés sont : A-B-D

Le seul valable est le C qui pourrait correspondre à la séquence d'ajout {B, E, A, C, D, F, G} (*il y en a d'autres*).

Solution 3 **(Arité moyenne d'un arbre général – *5 points*)**

Spécifications :

La fonction `average_arity(`T`)` retourne l'arité moyenne de l'arbre T (`TreeAsBin`).

```
"""
arity(B)  return (nb links, nb internal nodes)
"""

    def arity(B):
        '''
        with "classical" traversal
        '''
        if B.child == None:
            return (0, 0)

        else:
            (links, nodes) = (0, 1)
            child = B.child
            while child:
                (l, n) = arity(child)
                links += l + 1
                nodes += n
                child = child.sibling

            return (links, nodes)




    def arity(B):
        '''
        "binary" traversal
        '''
        if B.child == None:
            (links, nodes) = (0, 0)
        else:
            (l, n) = arity(B.child)
            (links, nodes) = (l + 1, n + 1)

        if B.sibling != None:
            (l, n) = arity(B.sibling)
            links += l + 1
            nodes += n

        return (links, nodes)
```

```

    def average_arity(B):
        (links, nodes) = arity(B)
        return links / nodes if nodes else 0
```

Solution 4 **(B-arbres : Représentation linéaire – *5 points*)**

Spécifications :

La fonction `btree2list(`B`)` retourne la représentation linéaire (de type `str`) de B s'il est non vide, la chaîne vide sinon.

```
1         def __BtreeToList(B):
2             s = '(<'
3             for i in range(B.nbKeys-1):
4                 s += str(B.keys[i]) + ','
5             s += str(B.keys[B.nbKeys-1]) + '>'
6             for child in B.children:
7                 s += __BtreeToList(child)
8             s += ')'
9             return s
10
11        def BtreeToList(B):
12            if B:
13                return __BtreeToList(B)
```

Solution 5 **(B-Arbres : insertions – *2 points*)**

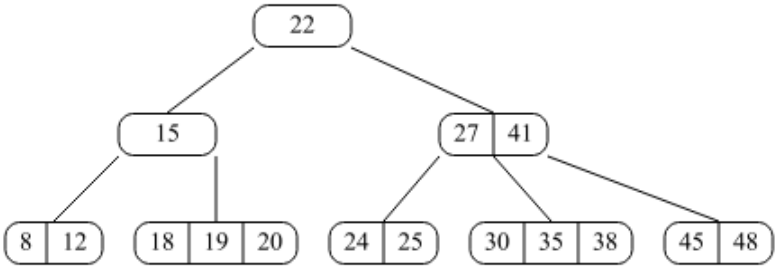

FIGURE 2 – B-arbre d'ordre 2

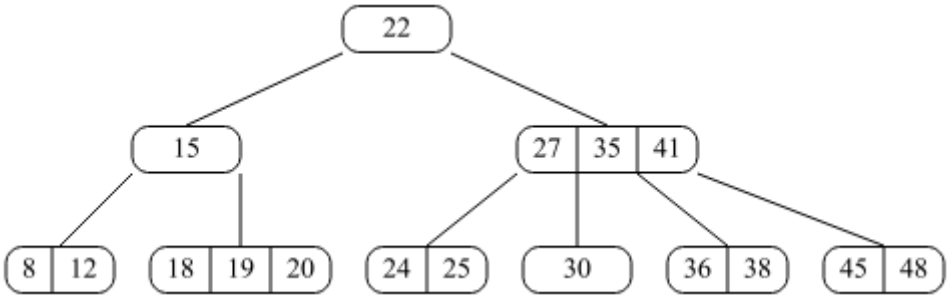

FIGURE 3 – Après insertion de 36

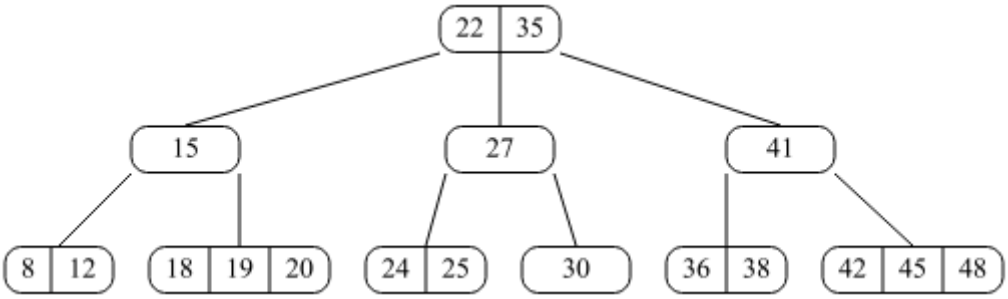

FIGURE 4 – Après insertion de 42

Solution 6 **(What ? –** ***3 points*****)**

1. Résultats :
   (a) `what(`B_3`, 2)` retourne `5`
   (b) `what(`B_3`, 7)` retourne `13`
   (c) `what(`B_3`, 18)` retourne `20`
   (d) `what(`B_3`, 39)` retourne `40`
   (e) `what(`B_3`, 41)` retourne `42`
   (f) `what(`B_3`, 99)` retourne `None`
2. La fonction `what(B, x)` retourne la clé de `B` immédiatement supérieure à `x`. La fonction renvoie `None` si une telle valeur n'existe pas.