Algorithmique
Correction Contrôle n° 3 (C3)

Info-spé (S3) – Epita

24 octobre 2016 - 14 : 45

Solution 1 **(Hachage linéaire –** ***2 points*****)**

Représentation des structures de données dans le cas du hachage linéaire *(Hachage linéaire avec un coefficient de décalage $d = 4$) voir tableau 1 :*

Table 1 – Hachage linéaire

0	sisko
1	odo
2	quark
3	neelix
4	data
5	kirk
6	q
7	picard
8	worf
9	tuvok
10	

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 **Hachage : Questions...** ***(3 points)***

1. Les trois propriétés essentielles que doit posséder une fonction de hachage sont :
   (a) Uniforme
   (b) Déterministe
   (c) Facile et rapide à calculer
2. Une collision secondaire est due au fait que deux éléments se retrouvent en collision sur une case du tableau de hachage alors que leur valeur de hachage primaire est différente (cf. Hachage coalescent).
3. Le phénomène provoqué par le hachage linéaire est le regroupement ou accumulation d'éléments (clustering) que l'on peut résoudre en envisageant un double hachage.

Solution 4 **(Arité moyenne d'un arbre général – *4 points*)**

```
"""
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 averageArity(B):
        (links, nodes) = arity(B)
        return links / nodes if nodes else 0
```

Solution 5 **(Égalité –** ***5 points*****)**

```
# T is the one traversed / with return statement in loop
    def equal(T, B):

        if T.key != B.key:
            return False

        else:
            Bchild = B.child
            for Tchild in T.children:
                if Bchild == None or not(equal(Tchild, Bchild)):
                    return False
                Bchild = Bchild.sibling

            return Bchild == None

# without return in the loop
 def equal2(T, B):

     if T.key != B.key:
         return False

     else:
         Bchild = B.child
         i = 0
         while i < T.nbChildren and (Bchild and equal2(T.children[i], Bchild)):
             i += 1
             Bchild = Bchild.sibling

         return i == T.nbChildren and Bchild == None
```

Solution 6 **(B-arbres et mystère –** ***3 points*****)**

		Résultat retourné	*Nombre d'appels*
1.	(a) `mystery(`B_1`, 1, 77)`	29	10
	(b) `mystery(`B_1`, 10, 30)`	11	7

2. `mystery(B, a, b)` calcule le nombre de valeurs de B dans $[a, b[$.