

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)

```
1
2 """
3 arity(B) return (nb links , nb internal nodes)
4 """
5
6 def arity(B):
7     '''
8     with "classical" traversal
9     '''
10    if B.child == None:
11        return (0, 0)
12
13    else:
14        (links, nodes) = (0, 1)
15        child = B.child
16        while child:
17            (l, n) = arity(child)
18            links += l + 1
19            nodes += n
20            child = child.sibling
21
22        return (links, nodes)
23
24
25
26
27 def arity(B):
28     '''
29     "binary" traversal
30     '''
31    if B.child == None:
32        (links, nodes) = (0, 0)
33    else:
34        (l, n) = arity(B.child)
35        (links, nodes) = (l + 1, n + 1)
36
37    if B.sibling != None:
38        (l, n) = arity(B.sibling)
39        links += l + 1
40        nodes += n
41
42    return (links, nodes)
```

```
1
2 def averageArity(B):
3     (links, nodes) = arity(B)
4     return links / nodes if nodes else 0
```

Solution 5 (Égalité – 5 points)

```
1 # T is the one traversed / with return statement in loop
2 def equal(T, B):
3
4     if T.key != B.key:
5         return False
6
7     else:
8         Bchild = B.child
9         for Tchild in T.children:
10            if Bchild == None or not(equal(Tchild, Bchild)):
11                return False
12            Bchild = Bchild.sibling
13
14        return Bchild == None
15
16 # without return in the loop
17 def equal2(T, B):
18
19     if T.key != B.key:
20         return False
21
22     else:
23         Bchild = B.child
24         i = 0
25         while i < T.nbChildren and (Bchild and equal2(T.children[i], Bchild)):
26             i += 1
27             Bchild = Bchild.sibling
28
29     return i == T.nbChildren and Bchild == None
```

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

	Résultat retourné	Nombre d'appels
1. (a) $\text{mystery}(B_1, 1, 77)$	29	10
(b) $\text{mystery}(B_1, 10, 30)$	11	7

2. $\text{mystery}(B, a, b)$ calcule le nombre de valeurs de B dans $[a, b[$.