Algorithmics
Correction Midterm #3 (C3)

UNDERGRADUATE 2^{nd} YEAR (S3) – EPITA

24 October 2016 - 14 : 45

Solution 1 (Linear probing – _2 points_)

Showing of the data structure in the case of linear probing (*Linear probing with an offset coefficient* $d = 4$*) see table 1:*

Table 1: Linear probing

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

Solution 2 (Hashing: Valid tables – _3 points_)

The tables which can not be the result of any insertion of the keys are : A-B-D
The only available is the table C which can be the result of the insertion sequence {B, E, A, C, D, F, G} (*there are others*).

Solution 3 Hashing: Questions... _(3 points)_

1. The three essential properties required of a hash function are:
   (a) Uniform
   (b) Consistent
   (c) Easy and fast to compute
2. A secondary collision is due to the fact that two elements are colliding on a box of hash table although their primary hash value are differents (see Coalesced hashing).
3. The phenomenon caused by linear probing is the primary clustering (the buildup of long runs of elements) that can be solved by considering a double hashing.

Solution 4 **(Average Arity of a General Tree –** ***4 points*****)**

```python

"""
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)
```

```python

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

Solution 5 (Equality – *5 points*)

```python
# 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-Trees and Mystery – *3 points*)

1.

	Returned result	*Call number*
(a) `mystery(`B_1`, 1, 77)`	29	10
(b) `mystery(`B_1`, 10, 30)`	11	7

2. `mystery(B, a, b)` calculates the number of values of B in $[a, b[$.