Algorithmics
Correction Midterm #3 (C3)

Undergraduate 2^{nd} year - S3 – Epita

5 November 2019 - 9 : 30

Solution 1 (Axes and graphs... – *5 points*)

1. The hashing with separate chaining and the coalesced hashing.
2. The collision resolution method with which secondary collisions appear is the **coalesced hashing**.
3. A secondary collision is a collision without a coincidence of hash values between an x and a y, with x different from y.
4. The order of a digraph is its number of vertices.
5. A zero degree vertex is called **isolated vertex**.
6. The vertices of `G` which have an outdegree equal to 0 are: **{6,9}**
7. The vertices of `G` which have an indegree equal to 1 are: **{2,7,8}**

Solution 2 (Average Arity of a General Tree – *5 points*)

Specifications:

The function `averageArity(`T`)` returns the average arity of the a general tree T(`TreeAsBin`).

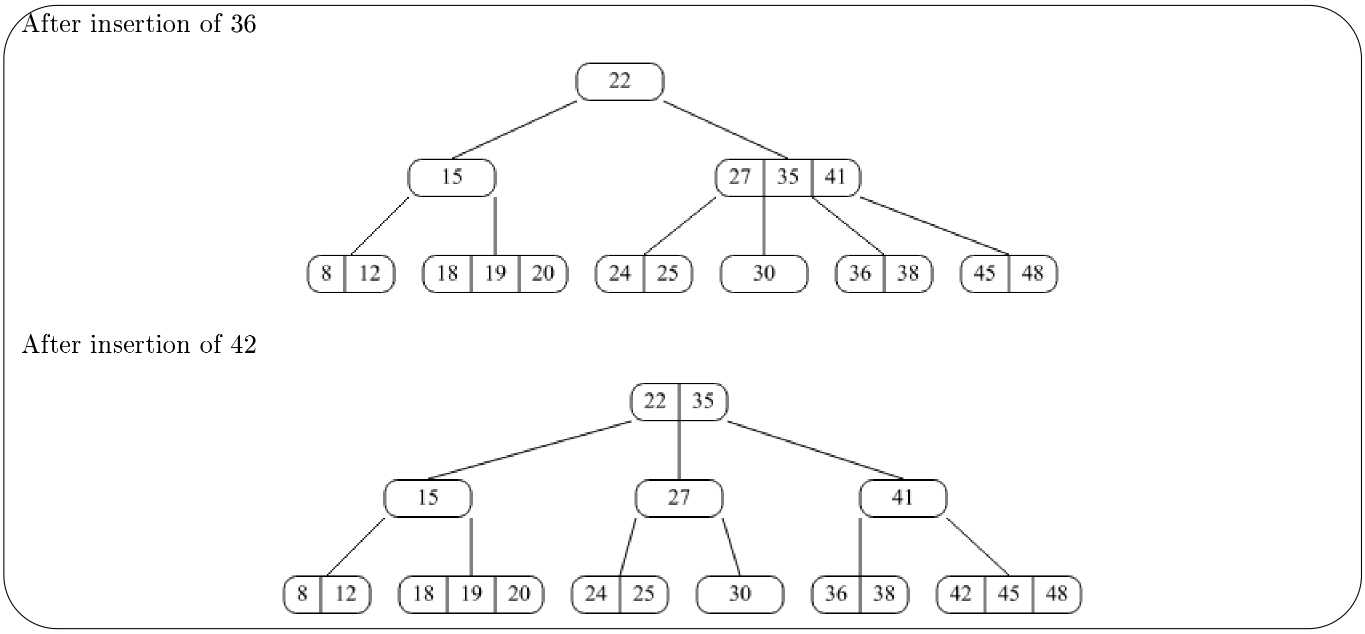
```
"""arity(B)  return (nb links, nb internal nodes)
"""
def arity(B):  # with "classical" traversal
    if B.child == None:
        return (0, 0)
    else:
        (links, nodes) = (0, 1)
        child = B.child
        while child:
            (lc, nc) = arity(child)
            links += lc + 1
            nodes += nc
            child = child.sibling
        return (links, nodes)

def arity(B):  # "binary" traversal
    if B.child == None:
        (links, nodes) = (0, 0)
    else:
        (lc, nc) = arity(B.child)
        (links, nodes) = (lc + 1, nc + 1)
    if B.sibling != None:
        (ls, ns) = arity(B.sibling)
        links += ls + 1
        nodes += ns
    return (links, nodes)

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

Solution 3 (B-trees: Insertions – *8 points*)

1. *Insertion of keys 36 and 42:*



2. **Specifications:**
   The function `__insert(`B`, `x`)` inserts the key x in the B-tree B, unless x is already in the tree. B is nonempty, and its root is not a full node (not a $2t$-node). It returns a boolean that tells if the insertion occurred.

```python
def __insert(B, x):
    i = search_pos(B.keys, x)

    if i < B.nbkeys and B.keys[i] == x:
        return False
    elif B.children == []:
        B.keys.insert(i, x)
        return True
    else:
        if B.children[i].nbkeys == 2 * B.degree - 1:
            if B.children[i].keys[B.degree-1] == x:
                return False
            split(B, i)
            if x > B.keys[i]:
                i += 1
        return __insert(B.children[i], x)
```

Solution 4 (B-Trees and Mystery – *2 points*)

```
nodes = [[22], [15], [27, 41], [8, 12], [18, 19, 20], [24, 25], [30, 35, 38], [45, 48]]

degree = 2
```