Algorithmics
Correction Midterm #3 (C3)

Undergraduate 2^{nd} year - S3 – Epita

9 November 2021 - 9 : 30

Solution 1 **(Graphs and components... – *5 points*)**

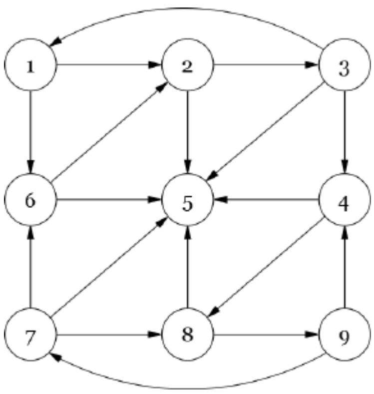

Figure 1: Digraph G

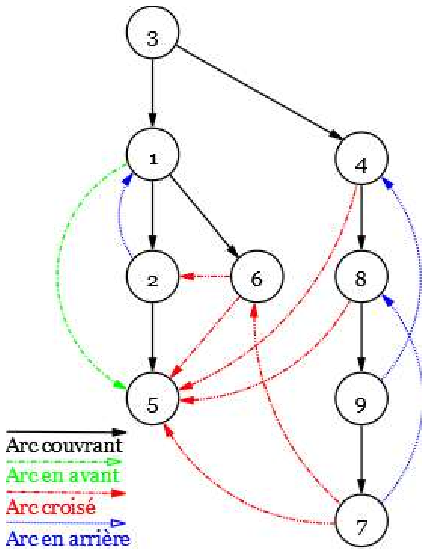

Figure 2: Spanning forest of the DFS (from 3 with vertices in increasing order)

1. The indegree array of G's vertices is as follows :

	1	2	3	4	5	6	7	8	9
indegrees	1	2	1	2	6	2	1	2	1

2. The *preorder* traversal vertices of the graph G starting from the vertex 3 are :
`3, 1, 2, 5, 6, 4, 8, 9, 7`

3. No the graph G is not strongly connected.

4. The graph has 2 strongly connected components.

5. There are no vertices of degree equal to 0.

Solution 2 (Large Family – *4 points*)

Specifications:

The function `morechildren(`T`)` checks if each internal node of the tree T (`TreeAsBin`) has strictly more children than its parent.

```python
def morechildren(B, nbc=0): # nbc = child number of B's parent
    k = 0
    C = B.child
    while C:
        k += 1
        C = C.sibling
    if B.child and k <= nbc:
        return False
    else:
        C = B.child
        while C and morechildren(C, k):
            C = C.sibling
        return C == None
```

Solution 3 (Decreasing – *4 points*)

Specifications:

`decrease(`B`)` returns the list of the keys of the B-tree B in decreasing order.

```python
def __decrease(B, L):
    if B.children == []:
        for i in range(B.nbkeys-1, -1, -1):
            L.append(B.keys[i])
    else:
        for i in range(B.nbkeys, 0, -1):
            __decrease(B.children[i], L)
            L.append(B.keys[i-1])
        __decrease(B.children[0], L)

def decrease(B):
    L = []
    if B:
        __decrease(B, L)
    return L
```

Solution 4 **(B-tree: insertion and deletion – *3 points*)**

1. Tree `B1` after the insertions of the values 11, 32, 20, using the "in going down" principle:

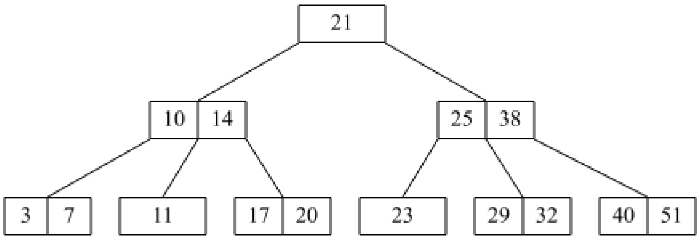

Figure 3: Après insertions

2. Tree `B2` after the deletion of the value 15, using the "in going down" principle:

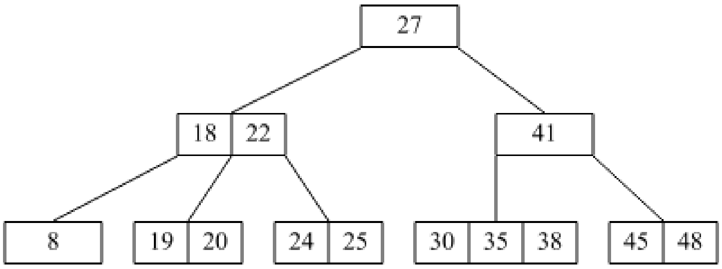

Figure 4: Après suppression

Solution 5 **(What? – *4 points*)**

1.

	Returned result	*Call number*
(a) `mystery(B2, 0, 92)`	`True`	8
(b) `mystery(B3, 0, 20)`	`False`	6
(c) `mystery(B3, 1, 99)`	`False`	8

2. The function `mystery(B, a, b)` tests whether B is "well-ordered" i.e. is a search tree, with its values in the interval $[a, b]$.