Algorithmique
Correction Contrôle n° 3 (C3)

Info-spé (S3) – Epita

7 novembre 2017 - 14 : 45

Solution 1 **(Quelques résultats différents – *6 points*)**

Représentations des tables de hachages en cas de :

1. hachage coalescent :

0	5	-1
1	20	-1
2	16	0
3	39	-1
4	11	2
5	44	10
6	94	3
7	12	8
8	23	-1
9	13	-1
10	88	4

2. hachage linéaire :

0	11
1	39
2	20
3	5
4	16
5	44
6	88
7	12
8	23
9	13
10	94

3. double hachage :

0	11
1	23
2	20
3	16
4	39
5	44
6	94
7	12
8	88
9	13
10	5

Solution 2 **(Préfixe - Suffixe –** ***3 points*****)**

Spécifications :

La fonction `preffsuff(`T`)` construit la liste des clés de l'arbre T en préfixe puis en suffixe.

```
def prefsuff(B, L=[]):

    L.append(B.key)

    child = B.child
    while child:
        prefsuff(child)
        child = child.sibling

    L.append(B.key)
    return L


def prefsuff2(B, L=[]):

    if B != None:
        L.append(B.key)
        prefsuff2(B.child)
        L.append(B.key)
        prefsuff2(B.sibling)
    return L
```

Solution 3 **(B-tree or not B-tree... –** ***5 points*****)**

Spécifications :

La fonction `test_Btree(`B`,` inf`,` sup`)` vérifie si l'arbre B est bien "ordonné" avec ses valeurs dans l'intervalle $]inf, sup[$.

```
def test_Btree(B, inf, sup):

    if B.keys[0] < inf or B.keys[B.nbkeys-1] > sup:
        return False

    else:

        for i in range(B.nbkeys-1):
            if B.keys[i] >= B.keys[i+1]:
                return False

        if B.children:
            for i in range(B.nbkeys):
                if not test_Btree(B.children[i], inf, B.keys[i]):
                    return False
                inf = B.keys[i]
            return test_Btree(B.children[B.nbkeys], inf, sup)

        else:
            return True
```

Solution 4 **(B-Arbres : insertions –** ***4 points*****)**

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

    if i >= B.nbkeys or B.keys[i] != x:

        if B.children != []:

            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)

        else:
            B.keys.insert(i, x)
            return True

    else:
        return False
```

Solution 5 **(B-arbres et mystère –** ***2 points*****)**

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

degree = 2
```