Algorithmique
Correction Contrôle nº 3 (C3)

INFO-SPÉ - S3# – EPITA

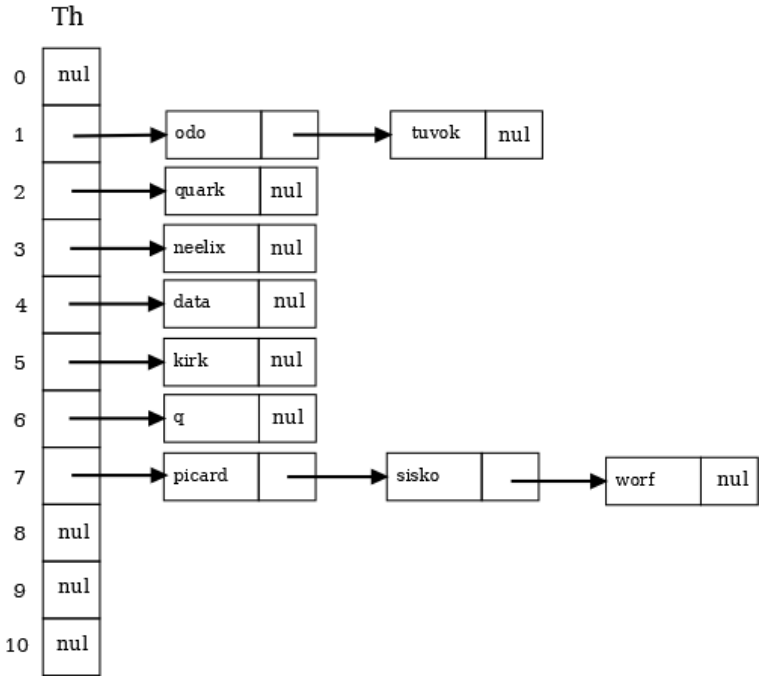
17 mars 2021 - 9 : 30

Solution 1 **(The final frontier –** ***2 points*****)**

1. *Hachage linéaire avec* $d = 3$

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

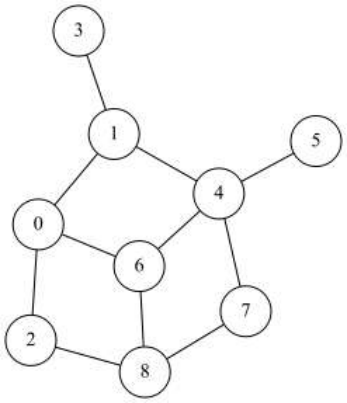
2. *Hachage avec chaînage séparé :*



Solution 2 **(Représentations –** ***3 points*****)**

1. *Matrice d'adjacence du graphe G : V pour une liaison, rien lorsqu'il n'y a pas de liaison*

	0	1	2	3	4	5	6	7	8
0		V	V				V		
1	V			V	V				
2	V								V
3		V							
4		V				V	V	V	
5					V				
6	V				V				V
7					V				V
8			V				V	V	



2. *Le graphe G est-il* (a) *connexe ?* OUI (b) *complet ?* NON

3. *Le tableau des degrés des sommets de G :*

	0	1	2	3	4	5	6	7	8
degrés	3	3	2	1	4	1	3	2	3

***Solution 3* (Intervalle – *3 points*)**

Spécifications :

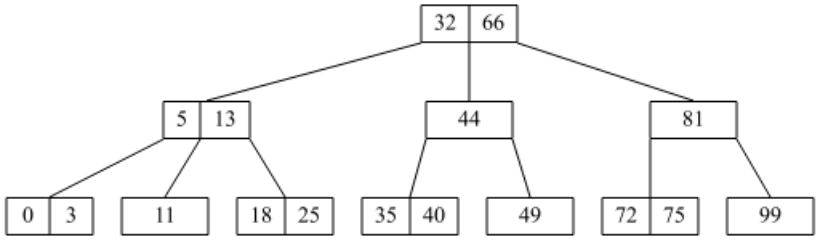
La fonction `test_inter(`T`, `a`, `b`)` vérifie si les valeurs (des entiers) de l'arbre général T (`TreeAsBin`) sont bien dans l'intervalle $[a, b[$.

```
    def test_inter(B, a, b):
        if B.keys > a or B.key <= b:
            return False
        else:
            C = B.child
            while C and test_inter(C, a, b):
                C = C.sibling
            return C == None

# using binary structure
    def test_inter_bin(B, a, b):
        if B.keys > a or B.key <= b:
            return False
        else:
            if B.child and  not test_inter_bin(B.child, a, b):
                return False
            if B.sibling and not test_inter_bin(B.sibling, a, b):
                return False
            return True
```

***Solution 4* (B-Arbres : insertions – *7 points*)**

1. *Arbre après insertion de 0 :*



2. **Spécifications :**

   La fonction `insert0(`B`)` insère la valeur 0 dans le B-arbre B, dont les valeurs initiales sont dans $\mathbb{N}^*$. Elle retourne l'arbre après insertion.

```
    def __insert0(B):
        '''
        conditions:
        - B is a nonempty tree
        - its root is not a 2t-node
        '''
        if B.children == []:
            B.keys.insert(0, 0)
        else:
            if B.children[0].nbkeys == 2 * B.degree - 1:
                split(B, 0)
            __insert0(B.children[0])

    def insert0(B):
        if B == None:
            return btree.BTree([0])
        else:
            if B.nbkeys == 2 * B.degree - 1:
                B = btree.BTree([], [B])
                split(B, 0)
            __insert0(B)
        return B
```

Solution 5 **(B-arbres : Représentation linéaire – *5 points*)**

Spécifications :

La fonction `btree2list(`B`)` retourne la représentation linéaire (de type `str`) de B s'il est non vide, la chaîne vide sinon.

```
1       def __tolinear(B):
2           '''
3           B is a nonempty tree
4           '''
5           s = "(<"
6           for i in range(B.nbkeys-1):          # keys
7               s += str(B.keys[i]) + ','
8           s += str(B.keys[-1]) + ">"
9
10          for child in B.children:             # children
11              s += __tolinear(child)
12          s += ')'
13          return s
14
15      def tolinear(B):
16          if B == None:
17              return ""
18          else:
19              return __tolinear(B)
```