Algorithmique
Correction Contrôle n° 3 (C3)

INFO-SPÉ - S3# – EPITA

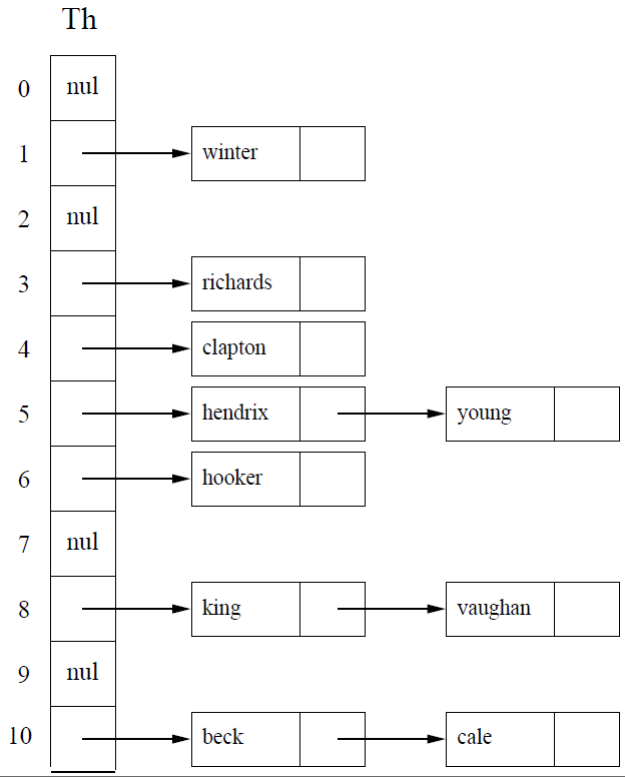
10 mars 2020 - 14h45

Solution 1 **(Hachages –** ***2 points*****)**

1. *Hachage linéaire avec* $d = 3$

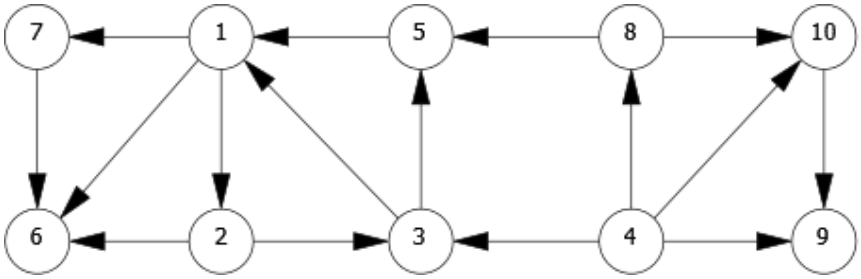
0	vaughan
1	winter
2	cale
3	richards
4	clapton
5	hendrix
6	hooker
7	
8	king
9	young
10	beck

2. *Hachage avec chaînage séparé :*



Solution 2 **(Dessiner c'est gagner –** ***2 points*****)**

1. Le graphe orienté :



2. Le tableau des degrés est le suivant :

	1	2	3	4	5	6	7	8	9	10
DemiDegréIntérieur	2	1	2	0	2	3	1	1	2	2

Solution 3 **(Égalité – *5 points*)**

Spécifications :

La fonction `same(`T`, `B`)` vérifie si T, un arbre général en représentation "classique" et B, un arbre général en représentation *premier fils - frère droit*, sont identiques.

```
# 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
```

Il y a plein d'autres manières de faire...
Dans tous les cas, bien faire attention à ce que la fonction gère bien les listes de fils de longueurs différentes !

Solution 4 **(Mesure sur les B-arbres – *4 points*)**

Spécifications :

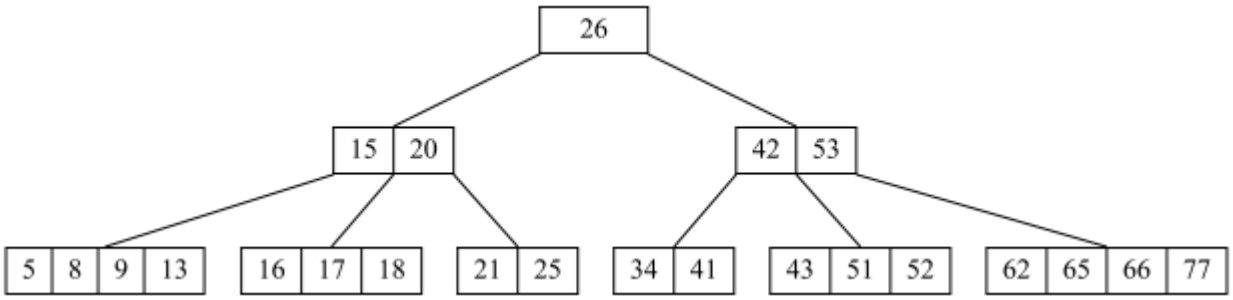
`occupation(`B`)` retourne le taux de remplissage des nœuds du B-arbre B.

```
    def __occupation(B):        # returns the pair (nb nodes, nb keys)
        (k, n) = (B.nbkeys, 1)
        for C in B.children:
            (kc, nc) = __occupation(C)
            k += kc
            n += nc
        return (k, n)

    def occupation(B):
        if not B:
            return 0
        else:
            (k, n) = __occupation(B)
            return (k/n)
```

Solution 5 **(B-Arbres : insertions –** ***8 points*****)**

1. Degré = 3
2. *Arbre après suppression de 3 :*



3. **Spécifications :**
   La fonction `__delmin(`B`)` supprime et retourne la valeur minimum du B-arbre non vide B.

```
def __delmin(B):
    if B.children == []:
        return B.keys.pop(0)
    else:
        if B.children[0].nbkeys == B.degree - 1:
            if B.children[1].nbkeys > B.degree - 1:
                print("left rotation")
                leftRotation(B, 0)
            else:
                print("merge")
                merge(B, 0)
        return __delmin(B.children[0])
```