

Algorithmique

Correction Contrôle n° 3 (C3)

INFO-SPÉ - S3# – EPITA

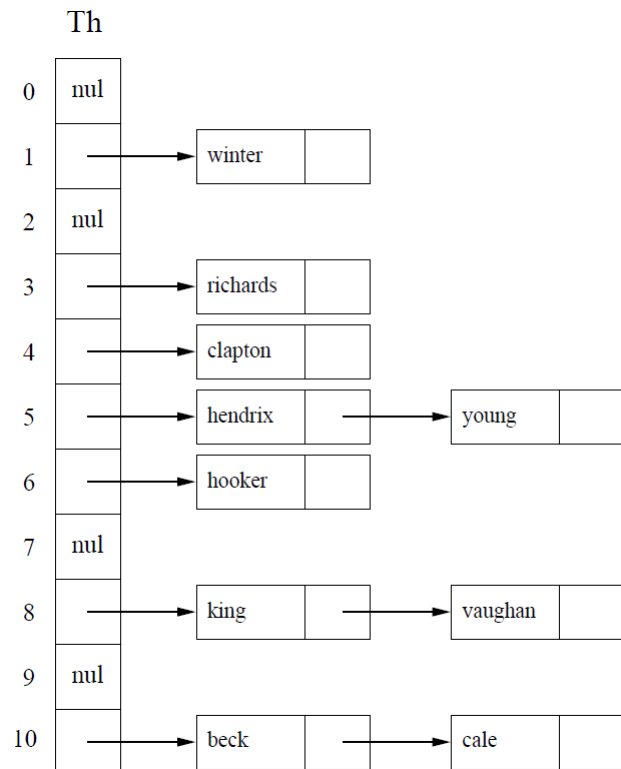
10 mars 2020 - 14h45

Solution 1 (Hachages – 2 points)

2. Hachage avec chaînage séparé :

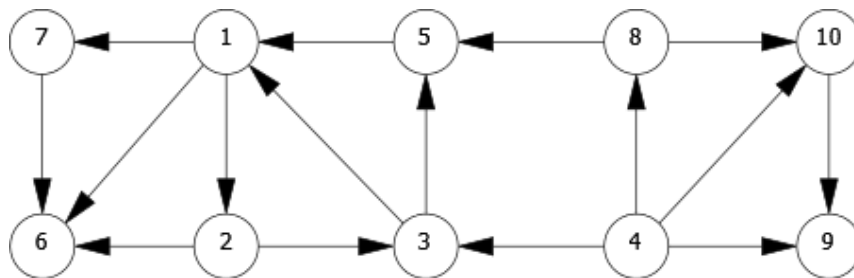
1. Hachage linéaire avec $d = 3$

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



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.

```

1  # with return statement in loop
2  def equal(T, B):
3      if T.key != B.key:
4          return False
5      else:
6          Bchild = B.child
7          for Tchild in T.children:
8              if Bchild == None or not(equal(Tchild, Bchild)):
9                  return False
10             Bchild = Bchild.sibling
11         return Bchild == None
12
13 # without return in the loop
14 def equal2(T, B):
15     if T.key != B.key:
16         return False
17     else:
18         Bchild = B.child
19         i = 0
20         while i < T.nbChildren and (Bchild and equal2(T.children[i], Bchild)):
21             i += 1
22             Bchild = Bchild.sibling
23         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 .

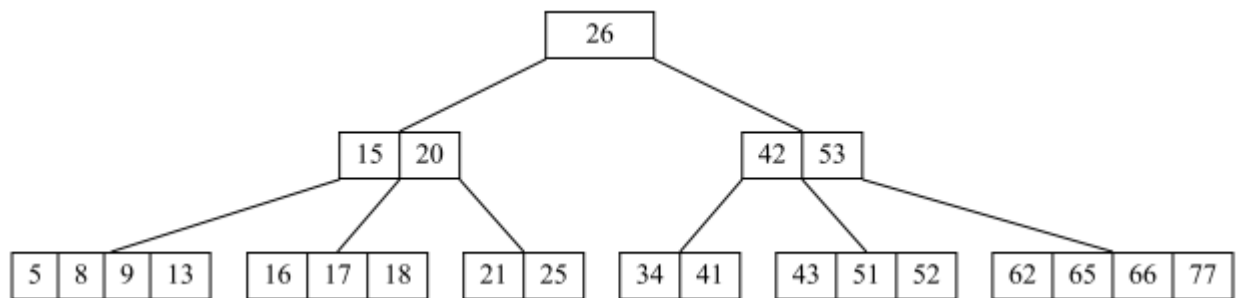
```

1  def __occupation(B):          # returns the pair (nb nodes, nb keys)
2      (k, n) = (B.nbkeys, 1)
3      for C in B.children:
4          (kc, nc) = __occupation(C)
5          k += kc
6          n += nc
7      return (k, n)
8
9  def occupation(B):
10     if not B:
11         return 0
12     else:
13         (k, n) = __occupation(B)
14         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 .

```
1 def __delmin(B):
2     if B.children == []:
3         return B.keys.pop(0)
4     else:
5         if B.children[0].nbkeys == B.degree - 1:
6             if B.children[1].nbkeys > B.degree - 1:
7                 print("left rotation")
8                 leftRotation(B, 0)
9             else:
10                print("merge")
11                merge(B, 0)
12    return __delmin(B.children[0])
```