

Algorithmique

Correction Contrôle n° 3 (C3)

INFO-SPÉ - S3 – EPITA

29 octobre 2018 - 13 : 30

Solution 1 (Hachage fortement connecté – 4 points)

1. Le hachage linéaire ou le double hachage.
2. Le hachage avec chaînage séparé. En effet les éléments sont chaînés entre eux à l'extérieur du tableau.
3. La recherche par intervalle est incompatible avec la dispersion des éléments effectuée par le hachage.
4. Les collisions secondaires apparaissent avec le hachage coalescent.
5. Le graphe $G=\langle S, A \rangle$ non orienté correspondant à :

$S=\{1,2,3,4,5,6,7,8,9,10\}$
 et $A=\{(1,2), (1,6), (1,7), (2,3), (2,6), (3,1), (3,5), (4,3), (4,8), (4,9), (4,10),$
 $(5,1), (7,6), (8,5), (8,10), (10,9)\}$

est celui de la figure 1

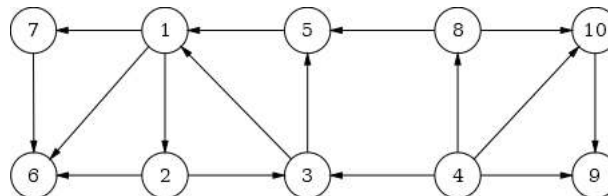


FIGURE 1 – Graphe orienté.

6. 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 2 (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
```

Solution 3 (Levels – 4 points)

Spécifications :

La fonction `levels(T)` construit la liste des clés de l'arbre T niveaux par niveaux.

```
1 def levels(T):
2     q = queue.Queue()
3     q.enqueue(T)
4     q2 = queue.Queue()
5     Levels = []
6     L = []
7     while not q.isempty():
8         T = q.dequeue()
9         L.append(T.key)
10        C = T.child
11        while C:
12            q2.enqueue(C)
13            C = C.sibling
14        if q.isempty():
15            (q, q2) = (q2, q)
16            Levels.append(L)
17            L = []
18
19    return Levels
```

Solution 4 (Gap maximum – 4 points)

Spécifications :

La fonction `maxgap(B)` calcule le gap maximum du B-arbre B .

```

1  # optimised version: searching in all children is useless,
2  # first and last child are sufficient!
3
4  def __maxgap(B):
5      gap = 0
6      for i in range(B.nbkeys-1):
7          gap = max(gap, B.keys[i+1] - B.keys[i])
8      if B.children:
9          gap = max(gap, __maxgap(B.children[0]))
10         gap = max(gap, __maxgap(B.children[-1]))
11     return gap
12
13 # less optimized...
14
15 def __maxgap2(B):
16     gap = 0
17     for i in range(B.nbkeys-1):
18         gap = max(gap, B.keys[i+1] - B.keys[i])
19
20     for child in B.children:
21         gap = max(gap, __maxgap2(child))
22     return gap
23
24 def maxgap(B):
25     return 0 if B is None else __maxgap(B)

```

Solution 5 (B-arbres et mystère – 3 points)

1. Résultats des applications :

| | Résultat retourné | Nombre d'appels |
|--|-------------------|-----------------|
| (a) <code>mystery(B_1, 1, 77)</code> | 29 | 10 |
| (b) <code>mystery(B_1, 10, 30)</code> | 11 | 7 |

2. `mystery(B, a, b)` ($a < b$) calcule le nombre de valeurs de B dans $[a, b[$.