Algorithmique
Correction Contrôle n^{o} 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 chainage 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=<S,A>` 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

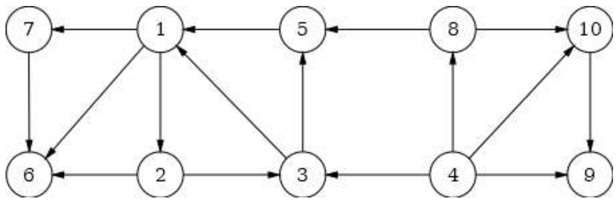

Figure 1 – Graphe orienté.

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.

```python
# 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
```

***Solution 3* (Levels – *4 points*)**

Spécifications :

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

```python
     def levels(T):
         q = queue.Queue()
         q.enqueue(T)
         q2 = queue.Queue()
         Levels = []
         L = []
         while not q.isempty():
             T = q.dequeue()
             L.append(T.key)
             C = T.child
             while C:
                 q2.enqueue(C)
                 C = C.sibling
             if q.isempty():
                 (q, q2) = (q2, q)
                 Levels.append(L)
                 L = []

         return Levels
```

Solution 4 **(Gap maximum –** ***4 points*****)**

Spécifications :

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

```
# optimised version: searching in all children is useless,
# first and last child are sufficient!

    def __maxgap(B):
        gap = 0
        for i in range(B.nbkeys-1):
            gap = max(gap, B.keys[i+1] - B.keys[i])
        if B.children:
            gap = max(gap, __maxgap(B.children[0]))
            gap = max(gap, __maxgap(B.children[-1]))
        return gap

# less optimized...

    def __maxgap2(B):
        gap = 0
        for i in range(B.nbkeys-1):
            gap = max(gap, B.keys[i+1] - B.keys[i])

        for child in B.children:
            gap = max(gap, __maxgap2(child))
        return gap

    def maxgap(B):
        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) `mystery(`B_1`, 1, 77)`	29	10
(b) `mystery(`B_1`, 10, 30)`	11	7

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