Algorithmique
Correction Contrôle n° 2 (C2)
(Version profs)

Info-sup S2# – Epita

novembre 2020

Solution 1 **(Maximum Gap –** ***4 points*****)**

Spécifications :

La fonction `maxgap(`M`)` retourne le gap maximum des lignes de la matrice non vide M.

```
def gaplist(L):
    """
    returns the gap of the list L (not empty)
    """
    valMin = L[0]
    valMax = L[0]
    for i in range(1, len(L)):
        valMin = min(valMin, L[i])
        valMax = max(valMax, L[i])
    return valMax - valMin

def maxgap(M):
    mgap = gaplist(M[0])
    for i in range(1, len(M)):
        mgap = max(mgap, gaplist(M[i]))
    return mgap
```

In one function (gaplist inlined) :

```
            def maxgap2(M):
                mgap = 0
                (l, c) = (len(M), len(M[0]))
                for i in range(l):
                    valMin = M[i][0]
                    valMax = M[i][0]
                    for j in range(1, c):
                        valMin = min(valMin, M[i][j])
                        valMax = max(valMax, M[i][j])
                    mgap = max(mgap, valMax - valMin)
                return mgap
```

***Solution 2* (Symétrique - *4 points*)**

Spécifications :

La fonction `isSymmetric(`A`)` teste si la matrice carrée A non vide est symétrique.

```
def issymmetric(A):
    (i, n) = (0, len(A))
    sym = True
    while i < n and sym:
        j = 0
        while j < i and A[i][j] == A[j][i]:
            j += 1
        sym = j == i
        i += 1
    return sym
```

***Solution 3* (Représentations et questions... – *3 points*)**

1. Elle s'appelle la *bijection premier fils-frère droit.*
2. C'est une bijection, il y a autant de noeuds dans l'arbre binaire qu'il y en a dans l'arbre général qu'il représente. Il suffit donc de faire un parcours de l'arbre binaire en comptant les noeuds rencontrés.
3. Seuls les fils sont plus haut que leur père. Les frères eux sont à la même hauteur. Il suffit donc de faire un parcours de l'arbre binaire en initialisant la hauteur et la hauteur maximum à 0. On augmente cette hauteur de 1 uniquement lorsqu'on suit un lien premier fils. A chaque fois qu'elle augmente, on la compare à la hauteur maximum qui prend sa valeur si celle-ci est supérieure. A la fin du parcours, hauteur maximum est égal à la hauteur de l'arbre général.
4. La figure 1 donne la représentation "binaire" (premier fils - frère droit) :

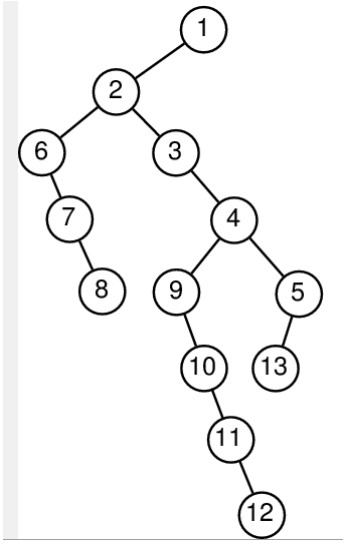
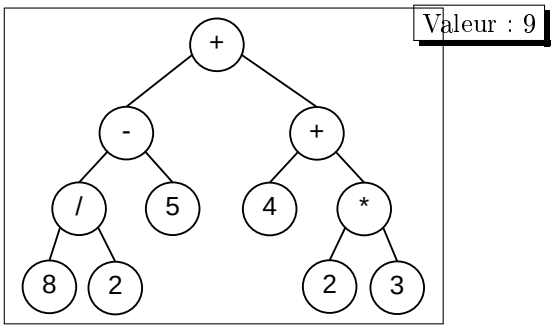


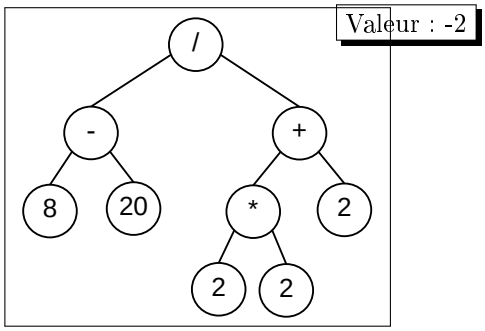
FIGURE 1 – Arbre binaire représentant l'arbre général

Solution 4 **(Dessine moi – *4 points*)**

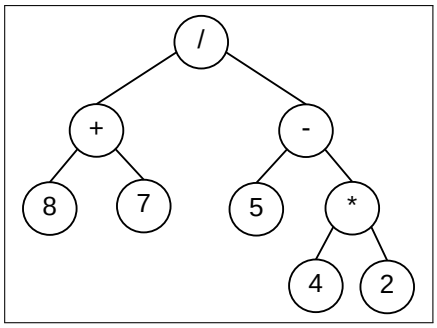
L'arbre B_1 :

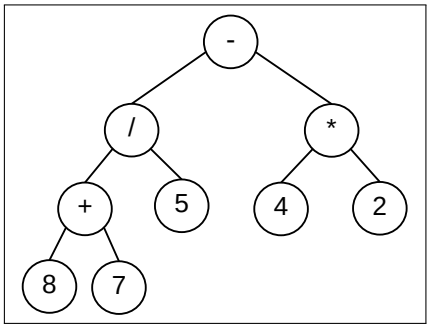


L'arbre B_2 :



B_3 peut être un des deux arbres suivants :





Solution 5 **(Compte moi – *3 points*)**

Spécifications :

La fonction `nodes(B)` calcule le nombre d'opérateurs *op* et le nombre d'opérandes *val* de l'arbre B non vide et retourne le couple (*op*, *val*).

```python
def nodes(B):
    """
    B is not empty and full
    """
    if B.left == None:  # no need to test B.right as the tree is full!
        return (0, 1)
    else:
        (int_left, ext_left) = nodes(B.left)
        (int_right, ext_right) = nodes(B.right)
        return (int_left + int_right + 1, ext_left + ext_right)

# ------------------------------------------------------------------------
# tips: as B is full, intern node number is leaf number + 1!

def __nbleaves(B):
    if B.left == None:
        return 1
    else:
        return __nbleaves(B.left) + __nbleaves(B.right)

def nodes2(B):
    """
    B is not empty and full
    """
    n = __nbleaves(B)
    return (n-1, n)
```

Solution 6 **(Affiche moi –** ***2 points*****)**

Spécifications :

La fonction `exp2str(`B`)` retourne une chaîne contenant l'expression, complètement parenthésée, représentée par l'arbre binaire non vide B.

```
1  def exp2str(T):
2      """
3      B is not empty and full
4      """
5      if T.left == None:
6          return str(T.key)
7      else:
8          s = '('
9          s = s + exp2str(T.left)
10         s = s + str(T.key)
11         s = s + exp2str(T.right)
12         s  = s + ')'
13         return s
14 # v2
15 def exp2str(T):
16     if T.left == None:
17        return str(T.key)
18     else:
19         return '(' + exp2str(T.left) + str(T.key) + exp2str(T.right) + ')'
```