

Algorithmique

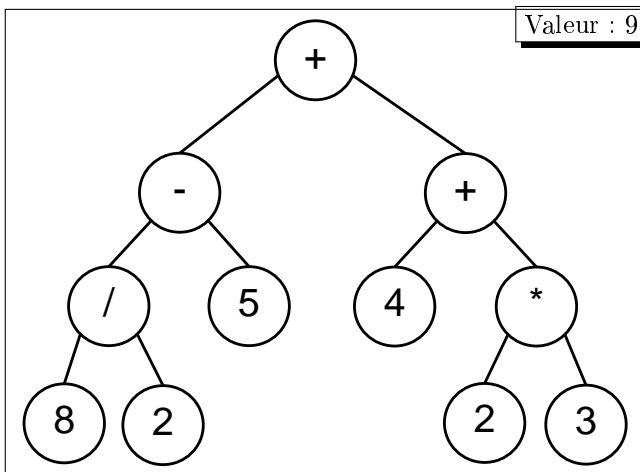
Correction Contrôle n° 2 (C2)

INFO-SUP S2 – EPITA

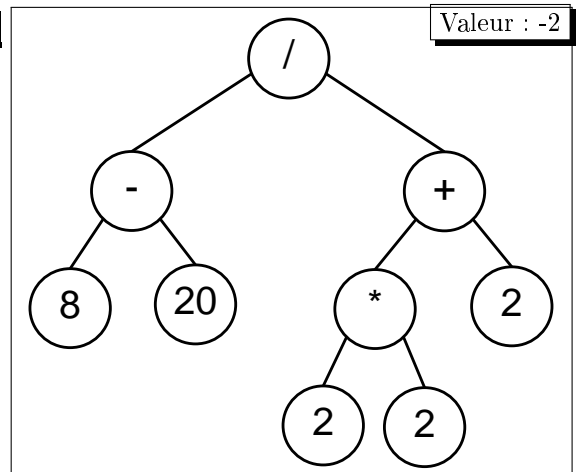
5 mars 2018 - 8 : 30

Solution 1 (Dessine moi – 4 points)

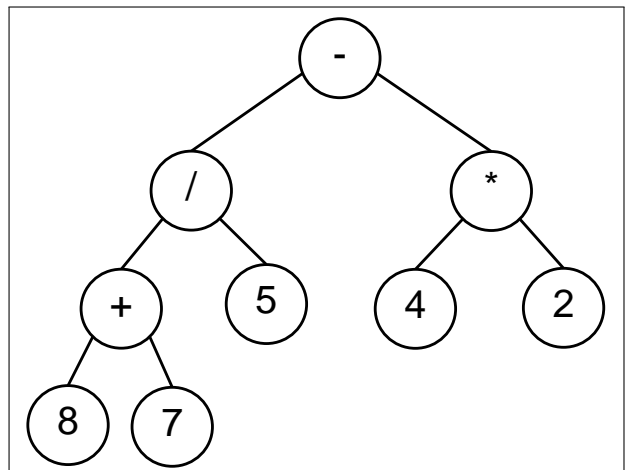
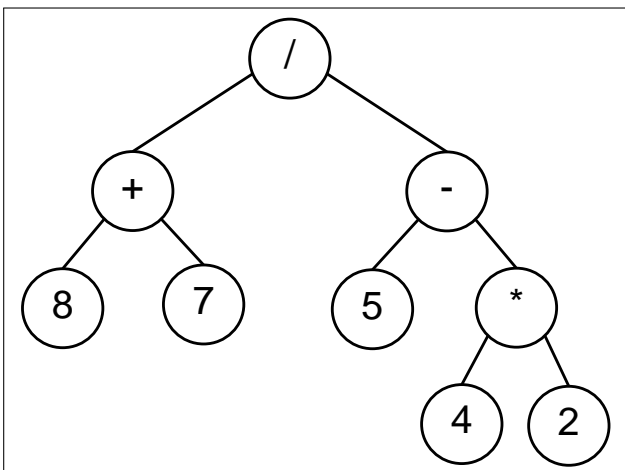
L'arbre B_1 :



L'arbre B_2 :



L'arbre B_3 :



Solution 2 (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` et retourne le couple `(op, val)`.

```

1  def nbLeaves(B):
2      if B.left == None:
3          return 1
4      else:
5          return nbLeaves(B.left) + nbLeaves(B.right)
6
7  def nodes(B):
8      if B == None:
9          return (0, 0)
10     else:
11         n = nbLeaves(B)
12         return (n-1, n)
13
14 # -----
15 def nodes_rec(B):
16     if B.left == None:
17         return (0, 1)
18     else:
19         (int_left, ext_left) = nodes_rec(B.left)
20         (int_right, ext_right) = nodes_rec(B.right)
21         return (int_left + int_right + 1, ext_left + ext_right)
22
23 def nodes2(B): # useless here as B is not None !
24     if B == None:
25         return (0, 0)
26     else:
27         return nodes_rec(B)

```

Solution 3 (Affiche moi – 2 points)

Spécifications :

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

```

1  def tree2expr(T):
2      if T.left == None:
3          return str(T.key)
4      else:
5          s = '('
6          s = s + tree2expr(T.left)
7          s = s + str(T.key)
8          s = s + tree2expr(T.right)
9          s = s + ')'
10         return s
11 # v2
12 def tree2expr2(T):
13     if T.left == None:
14         return str(T.key)
15     else:
16         return '(' + tree2expr(T.left) + str(T.key) + tree2expr(T.right) + ')'
17
18 # call
19 def exp2str(T): # useless here as B is not None !
20     if T == None:
21         return ""
22     else:
23         return tree2expr(T)

```

Solution 4 (Matrices : Symétrie - 4 points)

Spécifications :

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

```

1     def isSymmetric(A):
2         (i, n) = (0, len(A))
3         sym = True
4         while i < n and sym:
5             j = 0
6             while j < i and A[i][j] == A[j][i]:
7                 j += 1
8             sym = j == i
9             i += 1
10        return sym

```

Solution 5 (Minimax – 4 points)

La fonction `minimax(M)` retourne la **valeur minimale** parmi les maximums de chaque ligne de la matrice d'entiers M .

```

1     def maxList(L):
2         ''' maximum of list L, not empty '''
3         m = L[0]
4         for i in range(1, len(L)):
5             m = max(m, L[i])
6         return m
7
8     def minimax(M):
9         m = maxList(M[0])
10        for i in range(1, len(M)):
11            m = min(m, maxList(M[i]))
12        return m
13
14    #-----
15
16    def minimax2(M):
17        (n, p) = (len(M), len(M[0]))
18        mini = M[0][0]
19        for j in range(1, p):
20            mini = max(mini, M[0][j])
21        for i in range(1, n):
22            maxi = M[i][0]
23            for j in range(p):
24                maxi = max(maxi, M[i][j])
25            mini = min(mini, maxi)
26        return mini

```

Solution 6 (Mystery – 4 points)

1. Résultats :

`what(A)` retourne 6

Matrice A après application :

	0	1	2
0	2	5	5
1	4	14	5
2	7	7	6

`what(B)` retourne 11

Matrice B après application :

	0	1	2
0	1	5	10
1	3	6	14
2	8	8	11

2. Nombre d'additions effectuées par `what(M)` avec M non vide de taille $n \times n$: $n^2 - 1$