Algorithmics
Correction Test #2 (C2)
(Teacher version)

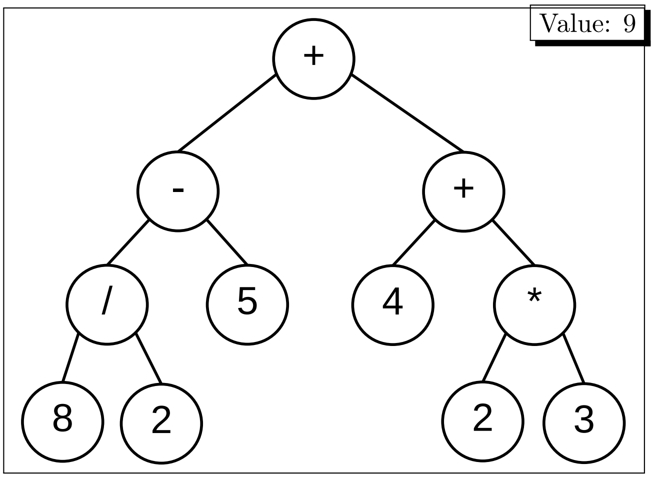
Undergraduate 1^{st} year (S2) – Epita

9 Mar. 2016 - 10:00

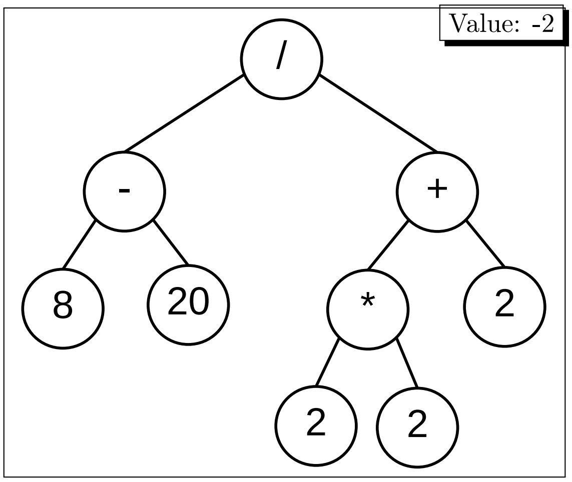
1 Expressions and trees

Solution 1.1 **(Draw me –** ***5 points*****)**

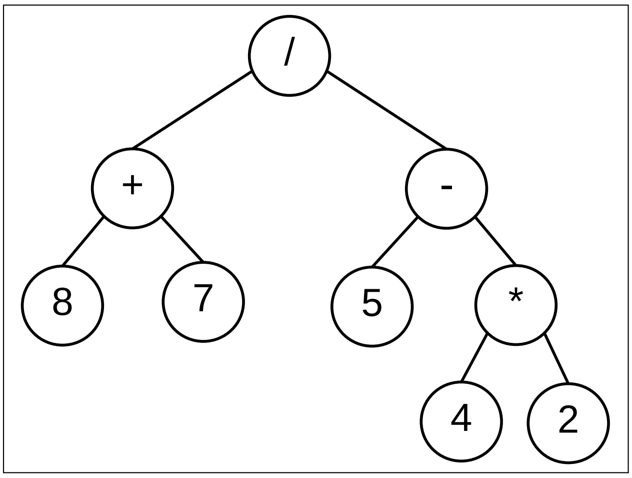
The tree B_1 :



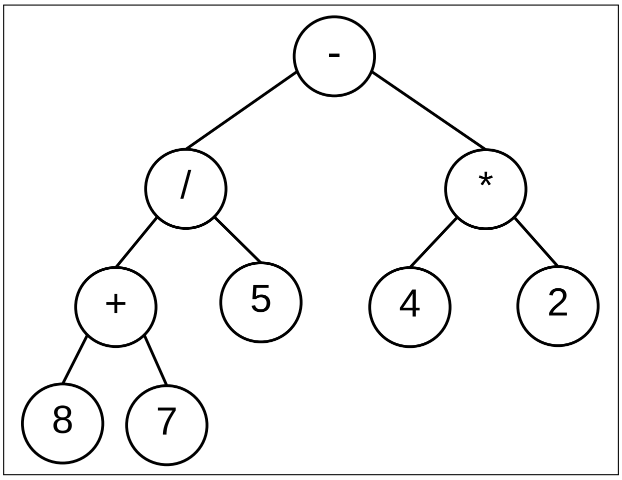
The tree B_2 :



The tree B_3 :



The tree B_4 :



Solution 1.2 (Count me – *3 points*)

Specifications:

The function `nodes`(B) computes the operator number *op* and the operand number *val* of the tree B. It returns the pair (*op*, *val*).

```python
    def nbLeaves(B):
        if B.left == None:
            return 1
        else:
            return nbLeaves(B.left) + nbLeaves(B.right)

    def nodes(B):
        if B == None:
            return (0, 0)
        else:
            n = nbLeaves(B)
            return (n-1, n)

# ---------------------------------------------------------------------
    def nodes_rec(B):
        if B.left == None:
            return (0, 1)
        else:
            (int_left, ext_left) = nodes_rec(B.left)
            (int_right, ext_right) = nodes_rec(B.right)
            return (int_left + int_right + 1, ext_left + ext_right)

    def nodes2(B):
        if B == None:
            return (0, 0)
        else:
            return nodes_rec(B)
```

Solution 1.3 (Display me – *3 points*)

Specifications:

The function `exp2str`(B) returns a string of the expression, fully parenthesized, represented by the tree B.

```python
def tree2expr(T):
    if T.left == None:
        return str(T.key)
    else:
        s = '('
        s = s + tree2expr(T.left)
        s = s + str(T.key)
        s = s + tree2expr(T.right)
        s  = s + ')'
        return s
# v2
def tree2expr2(T):
    if T.left == None:
       return str(T.key)
    else:
        return '(' + tree2expr(T.left) + str(T.key) + tree2expr(T.right) + ')'

# call
def exp2str(T):
    if T == None:
        return ""
    else:
        return tree2expr(T)
```

2 Some matrices

Solution 2.1 **(Minimax –** ***5 points*****)**

□ The function `minimax(`M`)` returns the **minimum value** of the maximums of each line in the integer matrice M.

```
    def maxList(L):
        ''' maximum of list L, not empty '''
        m = L[0]
        for i in range(1, len(L)):
            m = max(m, L[i])
        return m

    def minimax(M):
        m = maxList(M[0])
        for i in range(len(M)):
            m = min(m, maxList(M[i]))
        return m

#-----------------------------------------------------------------

    def minimax2(M):
        mini = maxint
        for L in M:
            maxi = L[0]
            for e in L:
                maxi = max(e, maxi)
            mini = min(mini, maxi)
        return mini
```

□ The function `posMinimax(`M`)` returns the **position of the minimum value** of the maximums of each line in the integer matrice M.

```
    def posMaxList(L):
        ''' maximum position of list L, not empty '''
        p = 0
        for i in range(1, len(L)):
            if L[i] > L[p]:
                p = i
        return p

    def posMinimax(M):
        (min_i, min_j) = (0, posMaxList(M[0]))
        for i in range(1, len(M)):
            max_j = posMaxList(M[i])
            if M[i][max_j] < M[min_i][min_j]:
                (min_i, min_j) = (i, max_j)
        return (min_i, min_j)

#-----------------------------------------------------------------

    def posMinimax2(M):
        mini = maxint
        (min_i, min_j) = (0,0)
        (l, c) = (len(M), len(M[0]))
        for i in range(l):
            max_j = 0
            for j in range(1, c):
                if M[i][j] > M[i][max_j]:
                    max_j = j
            if M[i][max_j] < mini:
                mini = M[i][max_j]
                (min_i, min_j) = (i, max_j)
        return (min_i, min_j)
```

Solution 2.2 **(Symmetry –** ***5 points*****)**

Specifications:

The function `symetric(`M`)` tests whether the matrix M has a vertical axis of symmetry (horizontal symmetry).

```python
def symetric(M):
    (l, c) = (len(M), len(M[0]))
    cdiv2 = c // 2
    i = 0
    stop = False
    while i < l and not stop:
        j = 0
        while j < cdiv2 and M[i][j] == M[i][c-j-1]:
            j += 1
        stop = (j < cdiv2)
        i += 1
    return not stop
```