Algorithmics
Correction Midterm #2 (C2)

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

22 February 2017 - 9 : 30

Solution 1 (**Be daring... - *4 points***)

1. Draw the corresponding tree.

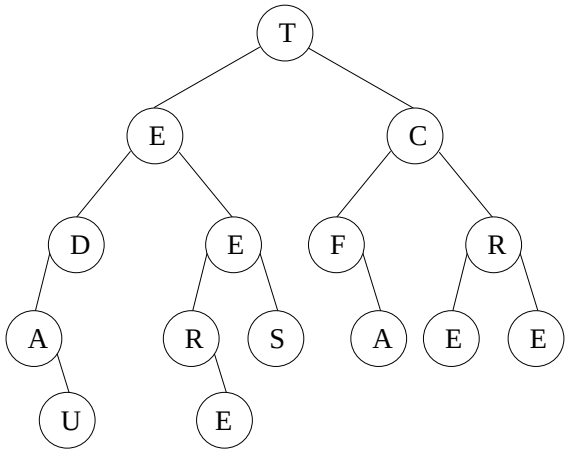

Figure 1: Graphical representation of the hierarchical array

2. Considering a depth-first traversal of this tree, give the inorder node list

   The inorder traversal of the tree of figure 1 is :

   `A U D E R E E S T F A C E R E` *(Audere est facere[1]) "Dare is to do"*

3. Give the occurrence representation ($B = \{\varepsilon, 0, 1, 01, 10, \ldots\}$) of this tree.

   $B = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 010, 011, 101, 110, 111, 0001, 0101\}$

[1] Tottenham football club motto

Solution 2 **(Maximum Gap – *5 points*)**

Specifications:

The function `maxGapMatrix(`M`)` returns the maximum gap of lines of the not empty matrice M.

```
def gapList(L):
    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 maxGapMatrix(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 maxGapMatrix2(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 3 **(Synergistic Dungeon – *4 points*)**

Specifications:

The function `dungeon(`M`)` returns the minimum initial number of health points the princess must have to rescue the knight in the dungeon represented by the not empty matrice M.

```
def dungeon(M):
    (l, c) = (len(M), len(M[0]))
                                #first row
    for j in range(1, c):
        M[0][j] += M[0][j-1]
                                #first column
    for i in range(1, l):
        M[i][0] += M[i-1][0]
                                # rest of the grid
    for i in range(1, l):
        for j in range(1, c):
            M[i][j] += min(M[i-1][j], M[i][j-1])

    return M[l-1][c-1] + 1
```

Solution 4 **(Tests – *8 points*)**

1. **Specifications:** The function `equal(`$B1$`, `$B2$`)` tests whether the trees $B1$ and $B2$ are identical.

```
        def equal(B1, B2):
            if B1 == None:
                return B2 == None
            elif B2 == None:
                return False
            elif B1.key == B2.key:
                return equal(B1.left, B2.left) and equal(B1.right, B2.right)
            else:
                return False

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

      def equal2(B1, B2):
          if B1 == None or B2 == None:
              return B1 == B2
          else:
              return (B1.key == B2.key) \
                     and equal2(B1.left, B2.left) \
                     and equal2(B1.right, B2.right)
```

2. **Specifications:** The function `isSubTree(`S`, `B`)` tests whether the tree S is a subtree of the tree B.

```
        def isSubTree(sB, B):
            if sB == None:
                return True
            elif B == None:
                return False
            else:
                if sB.key == B.key:
                    return equal(sB, B)
                else:
                    return isSubTree(sB, B.left) or isSubTree(sB, B.right)

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

        def search(x, B):
            if B == None:
                return None
            elif x == B.key:
                return B
            else:
                R = search(x, B.left)
                if R == None:
                    R = search(x, B.right)
                return R

        def isSubTree2(sB, B):
        if sB == None:
            return True
        else:
            return equal(sB, search(sB.key, B))
```