

Algorithmique

Correction Contrôle n° 2 (C2)

INFO-SUP (S2) – EPITA

22 février 2017 - 9 : 30

Solution 1 (Il faut oser ... - 3 points)

1. Représenter graphiquement l'arbre correspondant.

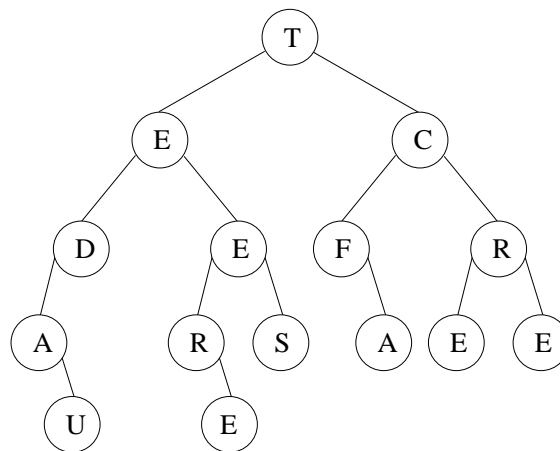


FIGURE 1 – Arbre représentant le tableau hiérarchique

2. En considérant un parcours profondeur main gauche de cet arbre, donner l'ordre infixé de rencontre des noeuds de cet arbre

Le parcours infixé de l'arbre de la figure 1 est :

A U D E R E E S T F A C E R E (Audere est facere¹) "Oser c'est faire"

3. Donner sous forme d'occurrences ($B = \{\varepsilon, 0, 1, 01, 10, \dots\}$) la représentation de cet arbre.

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

1. Devise du club de foot de Tottenham

Solution 2 (Maximum Gap – 5 points)

Spécifications :

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

```
1     def gapList(L):
2         valMin = L[0]
3         valMax = L[0]
4         for i in range(1, len(L)):
5             valMin = min(valMin, L[i])
6             valMax = max(valMax, L[i])
7         return valMax - valMin
8
9     def maxGapMatrix(M):
10        mgap = gapList(M[0])
11        for i in range(1, len(M)):
12            mgap = max(mgap, gapList(M[i]))
13        return mgap
```

In one function (gapList inlined) :

```
1     def maxGapMatrix2(M):
2         mgap = 0
3         (l, c) = (len(M), len(M[0]))
4         for i in range(l):
5             valMin = M[i][0]
6             valMax = M[i][0]
7             for j in range(1, c):
8                 valMin = min(valMin, M[i][j])
9                 valMax = max(valMax, M[i][j])
10            mgap = max(mgap, valMax - valMin)
11        return mgap
```

Solution 3 (Synergistic Dungeon – 4 points)

Spécifications :

La fonction `dungeon(M)` retourne le nombre minimum de points de vie que doit avoir la princesse pour sauver le chevalier dans le donjon représenté par la matrice non vide M .

```
1     def dungeon(M):
2         (l, c) = (len(M), len(M[0]))
3                                     #first row
4         for j in range(1, c):
5             M[0][j] += M[0][j-1]
6                                     #first column
7         for i in range(1, l):
8             M[i][0] += M[i-1][0]
9                                     # rest of the grid
10        for i in range(1, l):
11            for j in range(1, c):
12                M[i][j] += min(M[i-1][j], M[i][j-1])
13
14        return M[l-1][c-1] + 1
```

Solution 4 (Tests – 8 points)

1. **Spécifications :** La fonction `equal(B1, B2)` vérifie si les arbres `B1` et `B2` sont indentiques.

```
1     def equal(B1, B2):
2         if B1 == None:
3             return B2 == None
4         elif B2 == None:
5             return False
6         elif B1.key == B2.key:
7             return equal(B1.left, B2.left) and equal(B1.right, B2.right)
8         else:
9             return False
10
11 #-----
12
13     def equal2(B1, B2):
14         if B1 == None or B2 == None:
15             return B1 == B2
16         else:
17             return (B1.key == B2.key) \
18                 and equal2(B1.left, B2.left) \
19                 and equal2(B1.right, B2.right)
```

2. **Spécifications :** La fonction `isSubTree(S, B)` vérifie si l'arbre `S` est un sous-arbre de l'arbre `B`.

```
1     def isSubTree(sB, B):
2         if sB == None:
3             return True
4         elif B == None:
5             return False
6         else:
7             if sB.key == B.key:
8                 return equal(sB, B)
9             else:
10                return isSubTree(sB, B.left) or isSubTree(sB, B.right)
11
12 #-----
13
14     def search(x, B):
15         if B == None:
16             return None
17         elif x == B.key:
18             return B
19         else:
20             R = search(x, B.left)
21             if R == None:
22                 R = search(x, B.right)
23             return R
24
25     def isSubTree2(sB, B):
26         if sB == None:
27             return True
28         else:
29             return equal(sB, search(sB.key, B))
```