

Algorithmique

Correction Contrôle n° 2 (C2)

INFO-SUP S2 – EPITA

mars 2021

Solution 1 (Arbre Binaire : Ordres – 2 points)

1. L'arbre B correspondant à ces deux séquences est le suivant :

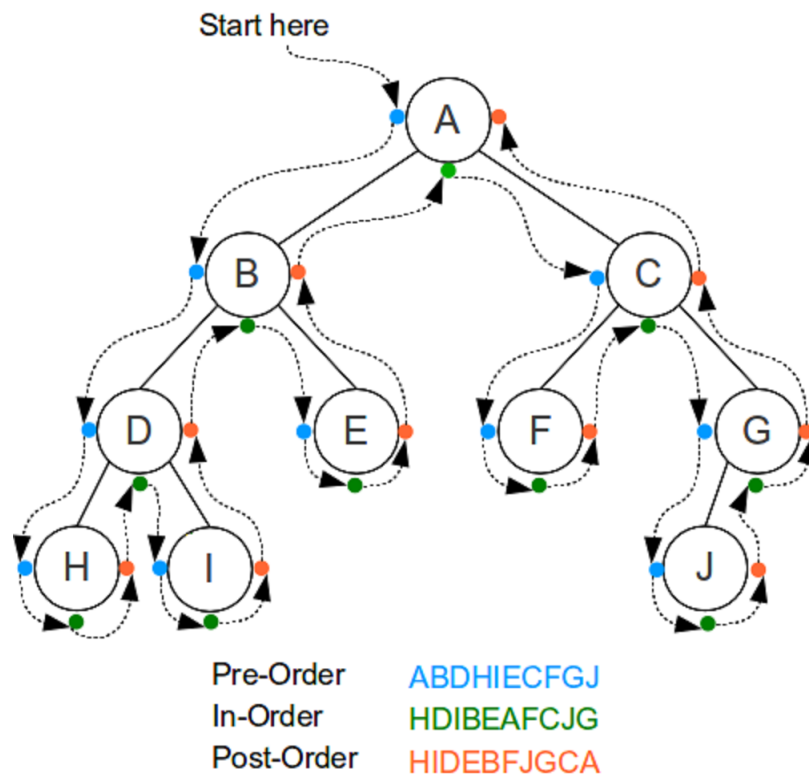


FIGURE 1 – Arbre binaire B

2. Les valeurs *suffixe* de rencontre de l'arbre B sont : H I D E B F J G C A

Solution 2 (ABR : insertions – 3 points)

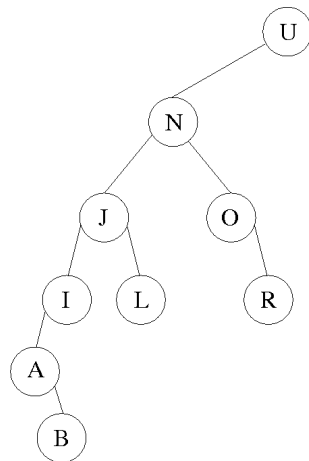


FIGURE 2 – Un joli ABR, construit en feuille

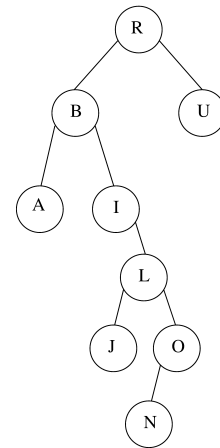


FIGURE 3 – Un joli ABR, construit en racine

Solution 3 (Matrice triée – 5 points)

Spécifications :

La fonction `list_sorted(L, n)` vérifie si la liste non vide `L` de longueur `n` est triée en ordre croissant.

```

1  def list_sorted(L, n):
2      i = 1
3      while i < n and L[i-1] <= L[i]:
4          i = i+1
5      return i == n
6

```

Spécifications :

La fonction `matrix_sorted(M)` vérifie si la matrice `M` est *triée* en ordre croissant. La matrice `M` est supposée non vide.

```

1  def matrix_sorted(M):
2      c = len(M[0])
3      if not list_sorted(M[0], c):
4          return False
5      else:
6          l = len(M)
7          i = 1
8          while i < l and M[i-1][c-1] < M[i][0] and list_sorted(M[i], c):
9              i = i+1
10         return i == l
11

```

Solution 4 (Largeur moyenne pondérée – 6 points)

Spécifications :

La fonction `get_average(B)` construit la liste `L`.

— `len(L)` = nombre de niveaux de l'arbre `B`

— `L[i]` = somme des clés des noeuds du niveau `i` divisée par le nombre de noeuds du niveau `i`

```

1  def get_average(B):
2      if not B:
3          return []
4      else:
5          q = queue.Queue()
6          q.enqueue(B)

```

```
7     q.enqueue(None)
8     s = 0
9     cpt = 0
10    res = []
11    while not q.isempty():
12        B = q.dequeue()
13        if B == None:
14            res.append(s/cpt)
15            if not q.isempty():
16                q.enqueue(None)
17                s = 0
18                cpt = 0
19        else:
20            s = s+B.key
21            cpt = cpt+1
22            if B.left != None:
23                q.enqueue(B.left)
24            if B.right != None:
25                q.enqueue(B.right)
26    return res
```

Solution 5 (Recherche du numéro hiérarchique – 4 points)

Spécifications :

La fonction `search_hier(B, x)` retourne le numéro en ordre hiérarchique de `x` s'il est présent dans `B`, `None` sinon.

```
1 def search_hier(B, x, num=1):
2     if B == None:
3         return None
4     else:
5         if B.key == x:
6             return num
7         else:
8             v = search_hier(B.left, x, 2*num)
9             if v != None:
10                 return v
11             else:
12                 return search_hier(B.right, x, 2*num+1)
```