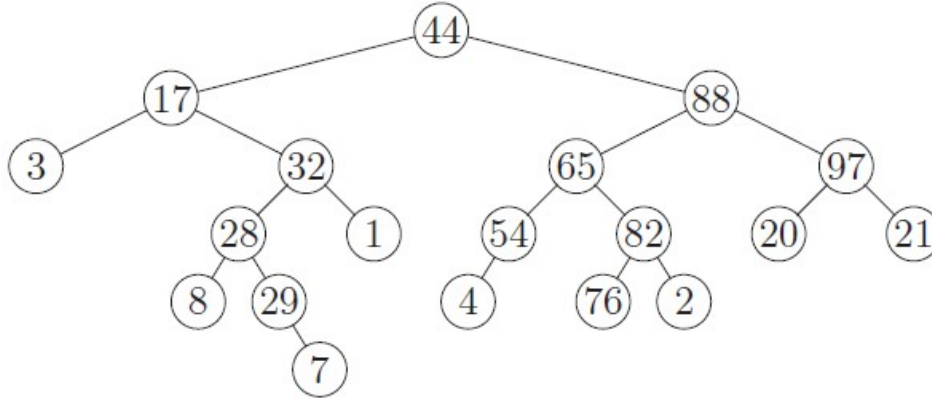


**Solutionnaire exercices sur les arbres et algorithme de Huffman**

1. Dans l'arbre suivant:



Le nœud racine est **44**.

Les nœuds internes sont **44, 17, 3, 32, 28, 29, 88, 65, 97, 54, 82, 97**.

Le nœud 32 a cinq descendants (**28, 8, 29, 7 et 1**), deux ancêtres (**17 et 44**), et un frère (**3**).

La profondeur du nœud 32 est deux (la longueur du chemin du nœud à la racine).

La hauteur du nœud 32 est  $3 + 1 = 4$  (soit le nombre total de nœuds dans le chemin plus long, ce dernier étant entre la dernière feuille qui descend du nœud et le nœud).

La suite des sommets selon l'ordre...

... préfixe est: **44,17,3,32,28,8,29,7,1,88,65,54,4,82,76,2,97,20,21.**

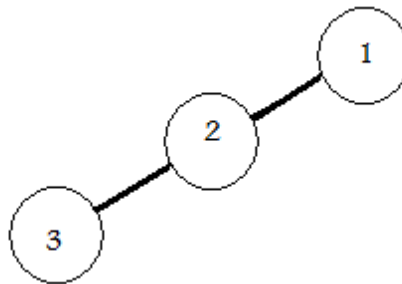
... suffixe est: **3,8,7,29,28,1,32,17,4,54,76,2,82,65,20,21,97,88,44.**

... l'ordre symétrique est: **3,17,8,28,7,29,32,1,44,4,54,65,76,82,2,88,20,97,21.**

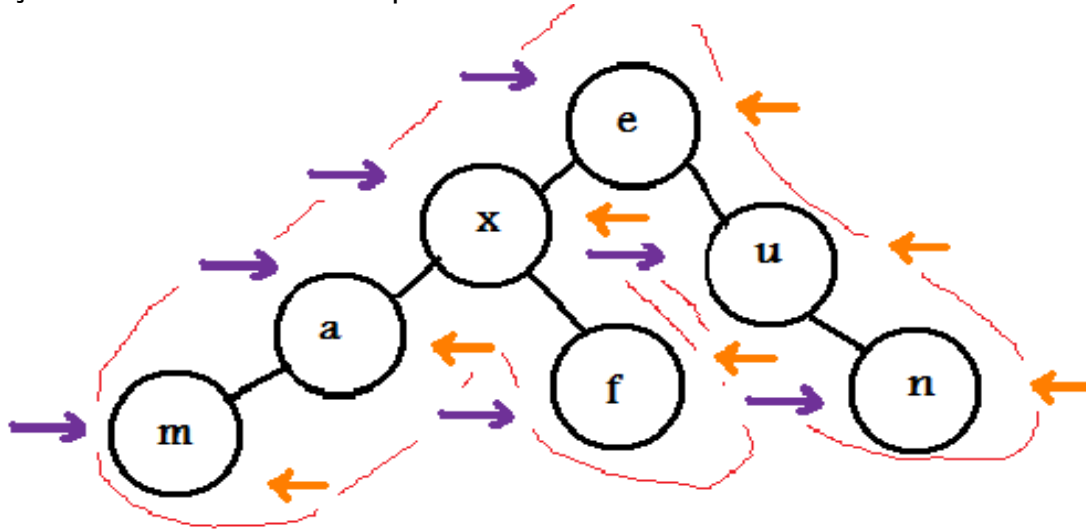
... l'ordre hiérarchique est: **44,17,88,3,32,65,97,28,1,54,82,20,21,8,29,4,76,2,7.**

2. Soit un arbre ordonné ayant plus d'un nœud. Il n'est pas possible que l'ordre préfixe visite les nœuds dans le même ordre que l'ordre suffixe, puisque rien qu'avec deux nœuds dans l'arbre l'ordre de visite n'est plus le même.

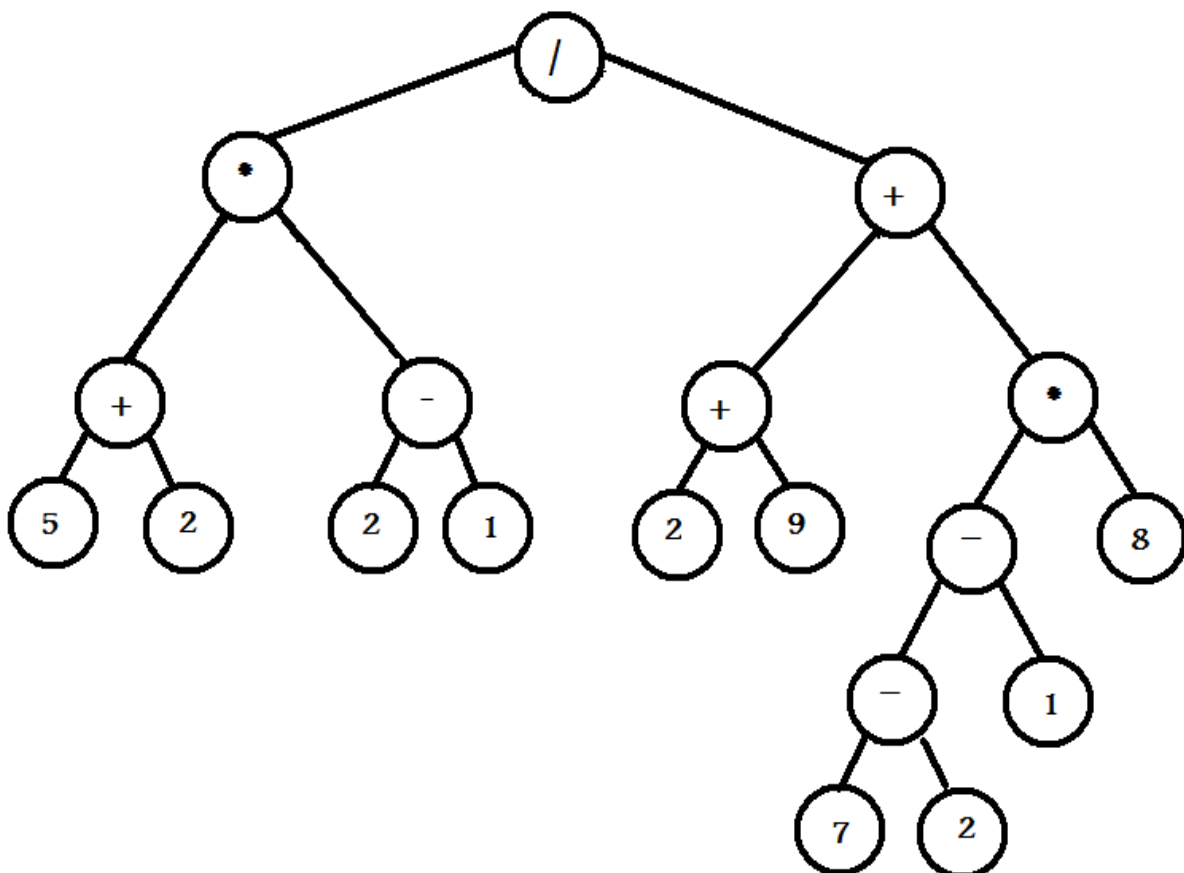
3. Soit un arbre ordonné ayant plus d'un nœud. Il est possible que l'ordre préfixe visite les nœuds dans l'ordre inverse que l'ordre suffixe. Il suffit d'imaginer l'arbre suivant, où les sommets visités selon l'ordre préfixe serait 1, 2, 3 et, selon l'ordre suffixe, 3, 2, 1.



4. Pour que ce numéro fonctionne, il faudrait corriger l'énoncé: Dessinez un arbre binaire tel que" i) chaque nœud contient une seule lettre, ii) traverser les nœuds de façon préfixe donne le séquence: "EXAMFUN" et iii) traverser les nœuds de façon suffixe donne la séquence "MAFXUEN" "MAFXNUE"



5. Voici l'arbre binaire représentant l'expression arithmétique suivante:  
 $((5+2)*(2-1))/[(2+9)+((7-2)-1)*8]$

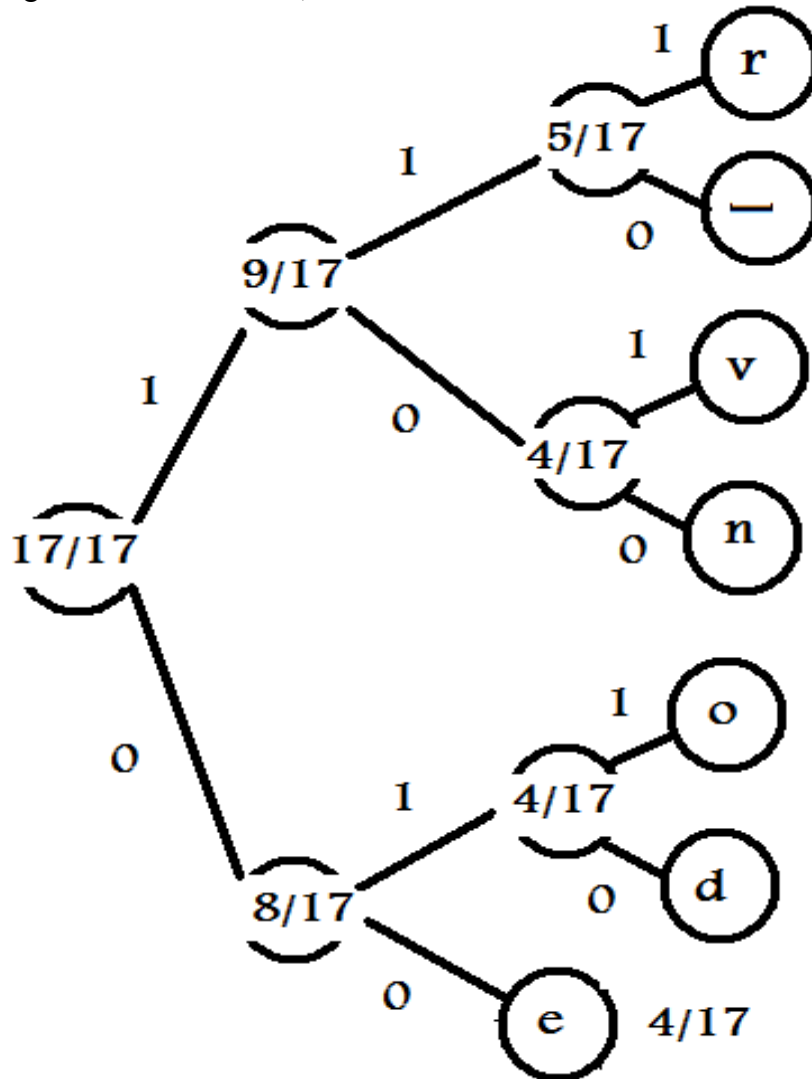


6. Soit le message suivante: never odd or even

a) Le tableau de la fréquence d'apparitions de chacune des lettres est:

Lettre	n	e	v	r	o	d	espace
Fréquence	2/17	4/17	2/17	2/17	2/17	2/17	3/17

b) Avec l'algorithme de Huffman, voici un arbre binaire donnant le meilleur codage.



**ATTENTION:** D'autres encodages sont possibles, les lettres r, v, o, n et d pouvant être échangées entre elles dans l'arbre. De même les sous-arbres ayant pour racine 4/17 pourraient aussi être échangés.

**ATTENTION 2:** L'ordre dans lequel on place les 1 et les 0 quand on construit l'arbre n'a aucune importance. Ce qui est important c'est que deux fils d'un même nœud n'ait pas le même bit.

Voici le tableau associant chaque lettre à son code binaire.

Lettre	n	e	v	r	o	d	espace
Code binaire	100	00	101	111	011	010	110

Longueur moyenne des codes associés aux lettres:

$$5 * 3 \text{ bits / lettre} * 2/17 + 1 * 3 \text{ bits / lettre} * 3/17 + 1 * 2 \text{ bits / lettre} * 4/17 = 2,67 \text{ bits/lettre.}$$

---

Tous les algorithmes qui suivent sont écrits sachant qu'on peut définir un arbre binaire récursivement:

```
Arbre
{
    Contenu
    Arbre FilsGauche
    Arbre FilsDroit
}
```

7. Voici un algorithme pour calculer le nombre de descendants de chaque nœud d'un arbre binaire.

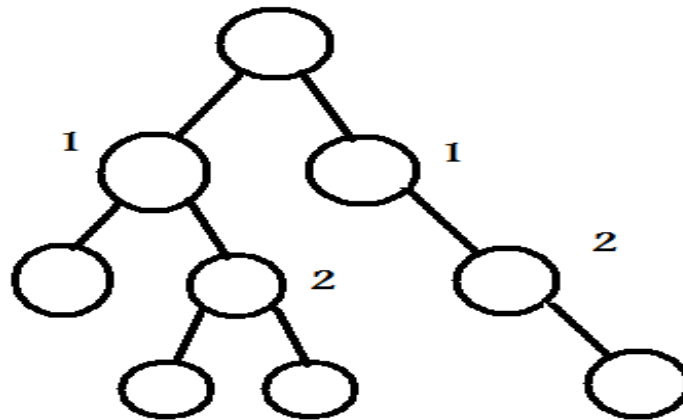
On parcourt l'arbre en profondeur par un algorithme récursif, jusqu'aux feuilles. La somme est nulle au premier appel de l'algorithme (Arbre = racine). La somme reste nulle à chaque appel. Elle augmente au fur et à mesure qu'on remonte dans l'arbre (que les appels récursifs se terminent.) On affiche la somme à la fin de l'algorithme (à la fin de chaque appel, on a le nombre de descendants de chaque nœud).

```
SommeDescendants( Arbre, Somme)
    Si Arbre. FilsG != NULL faire
        Arbre = Arbre.FilsG
        SommeDescendants(Arbre, Somme)
        Incréments Somme.
    Fin si
    Si Arbre. FilsD != NULL faire
        Arbre = Arbre.FilsD
        SommeDescendants(Arbre, Somme)
        Incréments Somme.
    Fin si
    Afficher Somme.
```



10. Voici un programme récursif qui calcule la somme des profondeurs **de tous les nœuds internes** d'un arbre binaire.

On pourrait parcourir l'arbre en largeur, et pour chaque niveau, on calculerait la somme des profondeurs. Pour parcourir l'arbre en largeur, il faudrait, au départ, mettre la racine dans le niveau courant. On remplit pour la première fois le niveau des enfants. Ensuite, on met le niveau courant à celui des enfants, et on parcourt tous les nœuds du niveau courant, en ajoutant leurs enfants dans le niveau des enfants. À chaque fois qu'on termine le parcours d'un niveau, on change le niveau : on prend le niveau des enfants comme niveau courant, et on recommence le parcours. Évidemment, à chaque niveau, on incrémente la profondeur et à chaque niveau, on calcule une somme égale au nombre de nœuds par niveau fois la profondeur. La somme des profondeurs sera égale à la somme de ces sommes. **Exemple où la somme est 6:**



```

SommeProfondeurs(Niveau courant, Somme, Profondeur)
  Si niveau courant est vide faire
    Si racine.FilsG n'est pas nul et n'est pas une feuille faire
      ajouter racine.FilsG dans niveau courant
    fin si
    Si racine.FilsD n'est pas nul et n'est pas une feuille faire
      ajouter racine.FilsD dans niveau courant
    fin si
    Incréments la profondeur
  Pour chaque arbre (nœud racine) du niveau courant faire
    Somme += nombre de nœuds dans le niveau courant * la profondeur
    Si arbre.FilsG n'est pas nul et n'est pas une feuille faire
      ajouter arbre.FilsG dans niveau enfant
    fin si
    Si arbre.FilsD n'est pas nul et n'est pas une feuille faire
      ajouter arbre.FilsD dans niveau enfant
    fin si
  Fin Pour
  Si niveau enfant n'est pas vide faire
    Niveau courant = niveau enfant
    SommeProfondeurs(Niveau courant, Somme, Profondeur)
  fin si

```