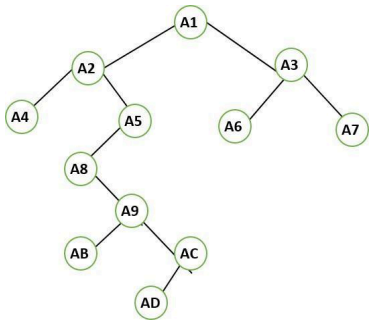
TD ALGO n°2
Arbres

Exercice 1

Un arbre binaire est un arbre binaire de recherche si:

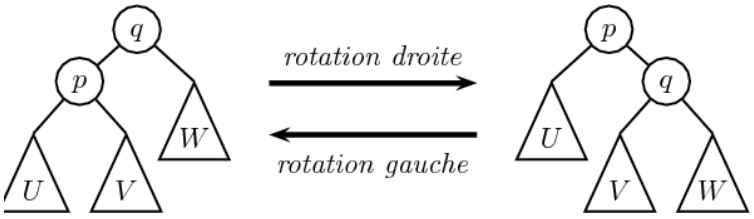
- Il existe une relation d'ordre sur les informations portées par les noeuds de l'arbre
- L'information portée par chaque noeud de l'arbre est supérieure à celle portée par tous les noeuds de son sous arbre gauche et inférieure à celle portée par tous les noeuds de son sous arbre droit

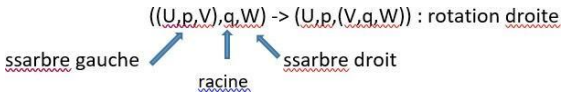
1) Exemples d'arbres :
    a) Créer un arbre binaire de recherche en y insérant les données suivantes dans l'ordre suivant : 100, 27, 250, 300, 121, 30, 29, 270, 129,308,307,340
    b) Créer un arbre binaire de recherche en y insérant les données suivantes dans l'ordre suivant :300, 307, 340, 129, 270, 250, 30, 100, 121, 27, 29, 308
    c) Quelles conclusions peut-on tirer de a) et b) ?
    d) Créer un arbre binaire de recherche en y insérant les données suivantes dans l'ordre suivant : 340, 308, 307, 300, 270, 250, 129, 121, 100, 30, 29, 27
2) Quel algorithme de parcours paraît avoir un intérêt pour un arbre binaire de recherche ? Pourquoi ?
3) Création/Insertion : écrire les algorithmes et donner leurs complexités
4) Recherche : écrire l'algorithme et donner sa complexité
5) Suppression d'un nœud : écrire l'algorithme et donner sa complexité
6) Soit l'arbre binaire de recherche suivant pour lequel les étiquettes ne sont pas les informations portées par les nœuds permettant de les ordonner (celles-ci ne sont pas visibles) mais juste un moyen de désigner les différents nœuds.



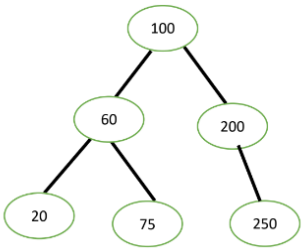
    a) Donnez le prédécesseur de A5 et expliquez
    b) Donnez le prédécesseur de AB et expliquez

7) Rotations : Un arbre est équilibré si en chaque nœud de l'arbre on a une différence de 0 ou 1 entre les hauteurs des deux sous-arbres.(explication : de nombreuses complexité dépendent de la hauteur de l'arbre). Les rotations permettent en gardant un arbre de recherche (ordonné) de modifier un arbre déséquilibré en un arbre équilibré.





a) Construire l'arbre suivant : 100, 20, 60
b) Quelles rotations faut-il appliquer pour obtenir un arbre avec 2 sous-arbres de hauteur 1
c) Ajouter 200 et 250
d) Quelle rotation appliquer pour obtenir un arbre équilibré ?
e) Ajouter 75
f) Que faire pour obtenir un arbre équilibré tel que celui ci-dessous?



Exercice 2

Un arbre AVL (Adelson-Velskii et Landis) est un arbre binaire (de recherche) tel qu'en tout sommet, la hauteur du fils droit et du fils gauche diffèrent au plus d'un.

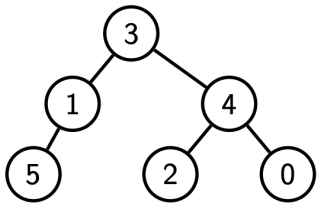
On veut démontrer que la hauteur d'un tel arbre est en $O(log_2 n)$ si n est le nombre de noeuds de l'arbre, plus précisément que $log_2(1 + n) \leq 1 + h \leq 1.44\, log_2(2 + n)$

1) Déterminez le nombre maximal de sommets dans un arbre de profondeur *h*. En déduire que le nombre de sommets *n* d'un arbre de haute *h* vérifie $log_2(1 + n) \leq 1 + h$.

2) On note *N(h)* le nombre minimal de sommets dans un arbre AVL. Justifiez que $N(h) = 1 + N(h-1) + N(h-2)$.
3) On pose $K(h) = 1 + N(h)$. Montrez que la suite $K(h)$ vérifie une formule de récurrence.
4) En utilisant que le n[ième] terme de la suite de Fibonacci *F(n)* peut être calculé par $F(n) = 1/\sqrt{5}(\Phi^n - \Phi^{-n})$ où $\Phi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or[1], et que $\frac{1}{log_2 \Phi} \leq 1.44$, montrez que $1 + h \leq 1.44 \times log_2(2 + n)$.

Exercice 3

Soit l'arbre suivant où chaque nœud contient une valeur.



On vous redonne ci-dessous l'algorithme d'exploration en profondeur d'abord. On considère que la fonction t.sons retourne les fils du nœud passés en paramètre dans l'ordre dans lequel ils apparaissent de gauche à droite dans la figure. Par exemple les fils de 3 sont la liste [1,4]

```
1: Function DepthFirstExplore (t:arbre)
2: if t is null then
3:     return
4: else
5:     for v ∈ t.sons(t.root) do
6:         Process(v)
7:         DepthFirstExplore(v, t)
8:     end for
9: end if
```

1) Introduisez les tests nécessaires dans l'algorithme précédent pour éviter les boucles infinies.
2) Exécutez l'algorithme modifié sur cet arbre, où la fonction Process est la fonction d'affichage de la valeur contenue dans le nœud. Détaillez bien les appels récursifs, les paramètres, et le contenu des variables au fur et à mesure de l'exécution. Dans quel ordre les valeurs des nœuds sont-elles affichées ?

[1] https://fr.wikipedia.org/wiki/Suite_de_Fibonacci

Exercice 4 *

On s'intéresse à des algorithmes de recherche d'une valeur dans un arbre, qui renvoient VRAI si une valeur est présente dans l'arbre et FAUX sinon.

1) En vous inspirant des algorithmes itératifs d'exploration en largeur et en profondeur d'un arbre vus en cours, écrivez une version itérative des algorithmes de recherche en largeur et en profondeur dans un arbre quelconque.

```
1: Function BreadthFirstExplore (n: node ; g:graph)
2: fringe = FIFO({n})
3: while Fringe!=Null do
4:     v ← Head(Fringe)
5:     if Not(Marked(v)) then
6:         Process(v)
7:         fringe.add(g.Neighbors(v))
8:     end if
9: end while
```

```
1: Function DepththFirstExplore (n: node ; g:graph)
2: fringe = LIFO({n})
3: while Fringe!=Null do
4:     v ← Head(Fringe)
5:     if Not(Marked(v)) then
6:         Process(v)
7:         fringe.add(Neighbors(v))
8:     end if
9: end while
```

On s'intéresse dans la suite à calculer la complexité de ces algorithmes en fonction de:

- d la profondeur dans l'arbre à laquelle se trouve la valeur cherchée
- b le facteur de branchement, égal au nombre (moyen) de fils d'un noeud
- m la profondeur de l'arbre (supposée finie)

2) Quelle est la complexité en temps et espace de la recherche en Largeur d'abord
3) Quelle est la complexité en temps et espace de la recherche en Profondeur d'abord
4) Qu'en déduisez vous sur l'algorithme à utiliser en fonction de la taille de l'arbre.

Largeur d'abord est impracticable pour des problèmes de recherche un peu complexes (b, d et m grands)

5) Ecrire la fonction *DepthFirstSearchPlimitée*
6) Quelle est la complexité en temps et en espace de *DepthFirstSearchPlimitée* ? de *DepthFirstSearchProfondeurItérative* ? Quelle conclusion en tirez vous ?