# TP5 - AVL : Arbres binaires de recherche bien équilibrés

## Ce qu'on voit lors de ce TP

- Un rappel des fonctions classiques des Arbres Binaires de Recherches tel que c'était enseigné en L2
- La notion d'AVL
- Comment implémenter une insertion dans un AVL

**Téléchargez l'archive liée au TP puis décompressez-la dans votre répertoire de travail. Dans le fichier `main.c`, il vous est fourni un exemple d'AVL pour tester vos fonctions.**

Notamment, une fonction qui transforme un AVL en fichier `.dot` y est appliquée.

Pour transformer un fichier `.dot` en une image visible, on pourra par exemple taper depuis un terminal

```
dot -Tsvg mon_fichier.dot -o mon_image.svg
```

## I) Fonctions basiques

**Vous commencerez par écrire les premières fonctions dont les prototypes sont dans `avl.h` et qui portent en réalité plus sur les arbres binaires de recherche tout court que sur les AVL.**

Le but avant tout c'est de se rappeler les algos sur les ABR que vous aviez fait en L2.

La recherche dans un AVL ça restera la même chose que la recherche d'un arbre binaire ! (sauf qu'on a en plus une garantie sur la hauteur d'un AVL et donc sur la complexité temporelle d'une recherche).

## II) C'est quoi un AVL ?

**Votre deuxième objectif sera de coder une fonction qui détermine si un arbre est bien un AVL (toujours en suivant les prototypes écrits dans `avl.h`.**

Plus complexe que ça n'en a l'air, il faudra vérifier que :

- notre arbre est bien un arbre binaire de recherche (i.e pour tout noeud `n`, tous

les noeuds présents dans le sous-arbre gauche de `n` ont une valeur plus petite que ou égale à la valeur de `n` et tous les noeuds présents dans le sous-arbre gauche de `n` ont une valeur plus que petite que la valeur de `n`)

- les facteurs d'équilibrage correspondent bien à la différence de la hauteur du sous-arbre droit avec le sous-arbre gauche
- les facteurs d'équilibrage sont tous compris entre $-1$ et $1$.

# III) Codage de l'insertion

**En tant qu'étape préliminaire, on codera les rotations simples qui sont très présentes chez les arbres binaires de recherche auto-équilibrant.**

Bien qu'on ne les utilisera que pour l'insertion, ces fonctions de rotation sont aussi importantes pour la suppression, l'union, etc. (qu'on ne codera pas ici)

**Après cela, on s'attaquera à la fonction d'insertion elle-même.**

Il sera très conseillé de vous munir du cours avec vous pour vous aider à programmer l'insertion.

# IV) Expérimentation

**Vous pouvez maintenant tester vos fonctions d'insertion sur les exemples à décommenter dans `main.c`.**

Vous pouvez également voir la différence de la hauteur entre l'insertion classique sans rééquilibrage (écrite dans la section I) et l'insertion d'un avl sur les différents exemples : permutation aléatoire, tableau trié, tableau mal mélangé (je n'ai pas encore programmé ce dernier point...).