Quelques remarques sur les fichiers Python contenu dans ce dossier :

- Implementation_ABR.py et Implementation_AVL :

  Implémentations classiques des ABR et AVL. Un jeu de test est lancé automatiquement pour chacunes de ces classes. Les fonctions principales disposent d'une description accessible avec la fonction usuelle help.

- Test_statistique.py

  Il y a deux fonctions principales, chacune déclinée en deux versions, une pour les AVL et une pour les ABR.

  graphe_distribution_hauteur(taille,nom_fichier = None) prend un entier et une chaine de caractère optionnelle en paramètre.
  La fonction génère un graphe représentant la répartition de 1000 ABR, chacun de la taille passée en paramètre, générés aléatoirement, en fonction de leur hauteur. La hauteur est en abscisse et l'ordonnée représente le nombre d'arbre d'une telle hauteur. La chaine de caractère permet d'enregistrer un png du graphique obtenu. La fonction graphe_distribution_hauteur_AVL est analogue.

  Pour la fonction
  graphe_complexite_ABR(taille_max,nb_tests,fonction,nom_fichier=False) il faut indiquer la taille maximale à atteindre pour les tests, en sachant qu'on commence par des arbres de taille 100, et le nombre de test à faire pour une taille fixée. On recommande respectivement 30000 et 100 au minimum pour ces paramètres. Il faut de plus indiquer la fonction à tester : "suppression", "insertion" ou "recherche"
  Le paramètre nom de fichier est un booleen, un nom de fichier avec les paramètres du test est généré automatiquement si nom_fichier est True.

  graphe_complexite_AVL fonctionne de la même manière

  Les autres fonctions sont en général des routines nécessaires aux fonctions ci-dessus, ou alors des fonctions pour faire des analyses supplémentaires que nous n'avons pas ajouté dans le rapport, comme la mesure du temps de création d'ABR ou d'AVL à partir de liste d'entiers croissants.

- Implementation_quadtree.py

  Contient notre implémentation des quadtree point. Les fonctions principales ont aussi des descriptions, le fonctionnement est similaire aux ABR. Nous n'avons pas pu faire de

fonctions de suppression et de recherche de voisins ou recherche dans une zone, faute de temps.

- Nouvelle_implementation_quadtree.py

  Contient notre implémentation des quadtree région. Certaines méthodes ne sont peut être pas fonctionnelles, nous n'avons pas eu le temps de toutes les tester/réparer.

- Compression
  Attention: Le programme utilise la librairie PIL (Python Image Library), il faut avant toute Installer cette dernière, sinon le programme ne peut pas fonctionner.
  Si vous avez Windows/MacOSX, il suffit de taper **"pip install pillow"** sur la console.

  Si vous avez Linux, essayez **"sudo apt-get install python3-pip"** puis **"sudo pip3 install pillow"**

  Utilisation: Une fois la librairie installée, Il faut se placer dans le répertoire de l'image que l'on souhaite compresser.
  Il faut ensuite modifier la *première* ligne de la fonction main: La structure est "**I=Image.open(<*nom de votre image*>)**"

  Il faut ensuite modifier la *troisième* ligne du main pour choisir le taux de compression voulu: La structure est **"Tree.compression(<*Taux de compression voulu*>)"**

  Remarques: Un taux de compression trop faible donne une image de très basse qualité. Un taux de compression trop grand est inutile car l'image renvoyée est exactement l'image de départ (un message d'erreur vous indique lorsque le taux de compression est trop grand).
  Les taux de compression intéressants se situent généralement entre 5 et 9. Cela peut changer en fonction de la taille de l'image.
  Le programme est relativement long sur les grosses images.

  On peut ensuite exécuter la fonction en faisant un appel à la fonction **main()**