Arbres Binaires de recherche

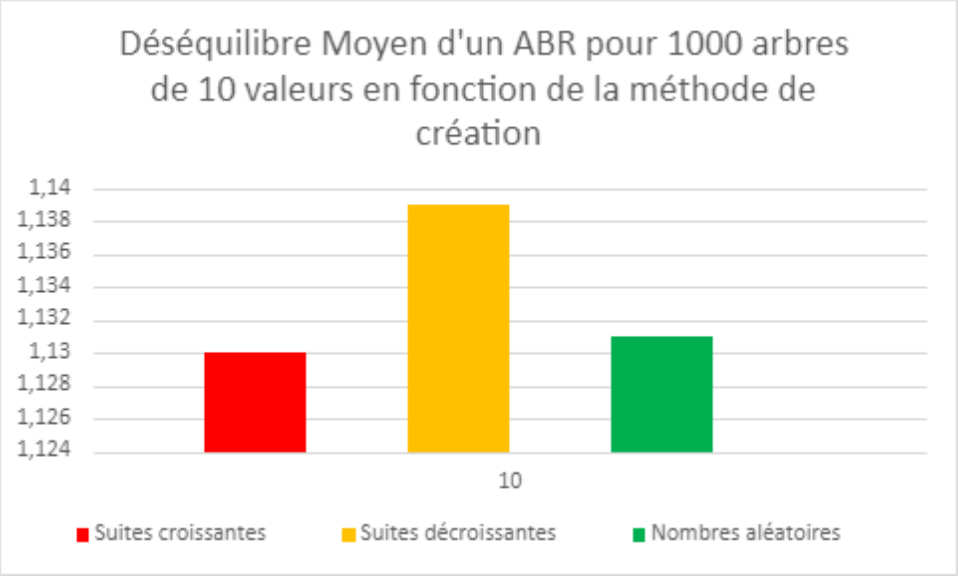
Nous avons commencé par créer la fonction bst_rnd_create qui permet de créer un arbre binaire de recherche en ajoutant des nombres aléatoires au feuilles. Pour cela nous avons utilisé la fonction Random.int du module Random d'Ocaml.

Ensuite, nous avons créé une fonction qui nous permet d'estimer le déséquilibre d'un arbre binaire de recherche. Puis nous avons adapté ce code pour créer un arbre à partir de plusieurs suites de nombres, ces suites sont créées aléatoirement et sont de type [n, n+1, n+2, n+...].

Résultats d'expérimentation :

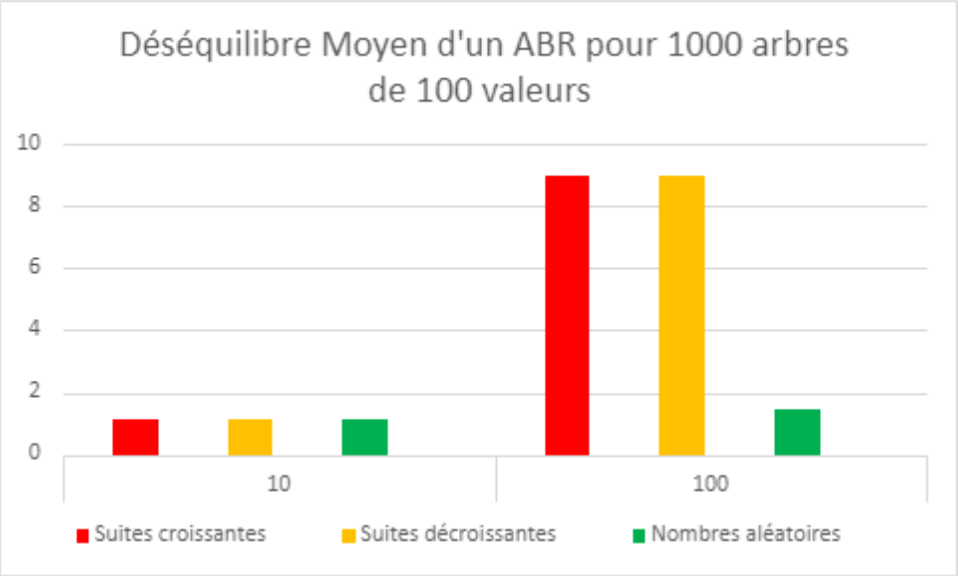
Différents tests : 10, 100, 1000 arbres

Résultats pour arbres de taille 10



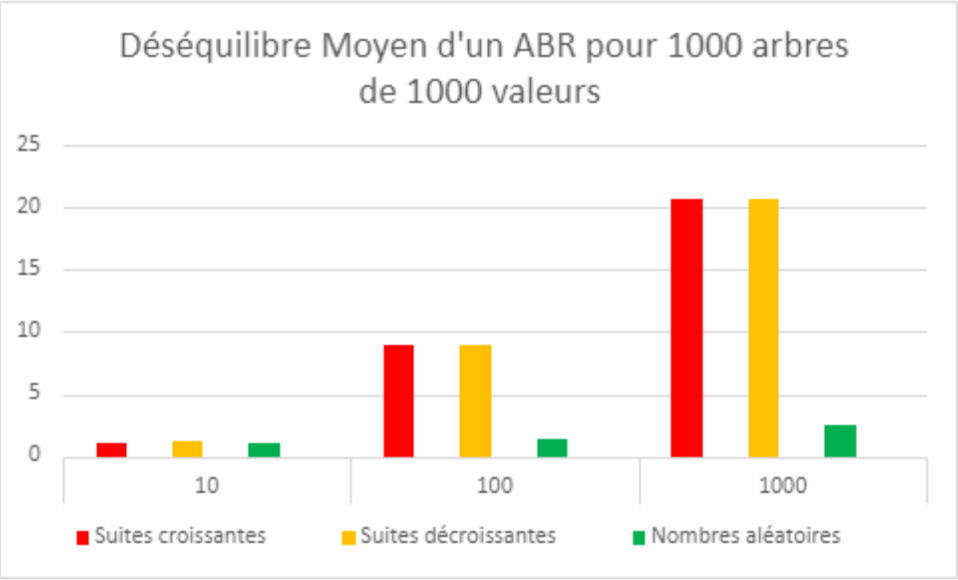
Chaque arbre a 10 valeurs. Nous constatons que le déséquilibre moyen n'est pas égal pour tous. La différence est faible entre les 3, même si les suites décroissantes ont un déséquilibre moyen plus élevé (≈1.139) et les suites croissantes en ont un moins élevé (≈1.13). Mais cela est négligeable à ce niveau-là.

Résultats pour arbres de taille 100



Chaque arbre a 100 valeurs. Cette fois nous constatons que les suites croissantes (≈9) et décroissantes (≈9) sont très proches l'une de l'autre en termes de déséquilibre. Cependant pour les nombres aléatoires il y a un déséquilibre moyen (≈1.5) vraiment plus faible que les deux autres.

Résultats pour arbres de taille 1000



Chaque arbre a 1000 valeurs. Ici nous observons que le constat fait pour arbres de 100 valeurs s'accentue davantage pour les arbres de 1000 valeurs.

Conclusion :

Pour des arbres binaires de recherche nous pouvons en déduire que le déséquilibre moyen est significativement plus petit pour une suite de nombres aléatoires en comparaison aux déséquilibres moyens des suites croissantes et décroissantes pour un arbre avec beaucoup de valeurs. De plus le parcours de ces arbres est forcément différent. Le déséquilibre étant plus petit chez les arbres à valeurs aléatoires, leurs tailles sont par conséquent plus petites. Donc nous pouvons

en conclure que la complexité pour obtenir une valeur N est plus petite pour les arbres à valeurs aléatoires que pour les arbres à suites croissantes et décroissantes.

Arbres AVL

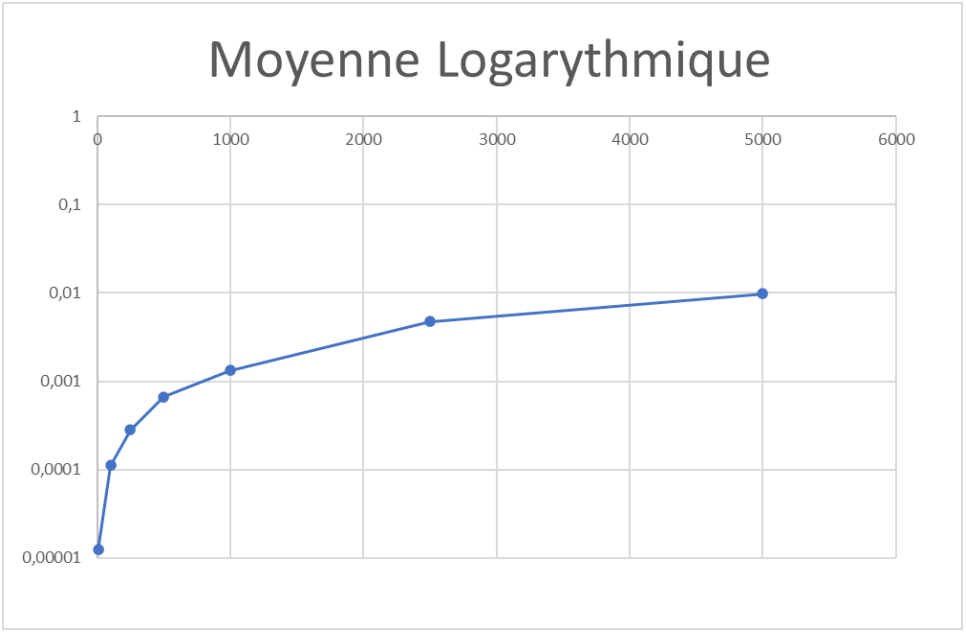
Nous avons fait la fonction avl_rnd_create qui nous permet de générer aléatoirement un arbre AVL. Cette fonction utilise avl_insert pour ajouter une valeur.

Pour tester la complexité moyenne de nos arbres AVL, nous utilisons les fonction avl_complexite (seek , insert, delete en fonction du cas) qui calcule la complexité des fonctions de recherche, d'insertion et de suppression.

Résultats d'expérimentation :

Différents tests de complexité :

Résultats pour la fonction d'insertion (avl_insert)



Cette courbe est de la forme logarithmique ce qui confirme que la fonction d'insertion à une complexité en O(log n) ou n est la taille de l'AVL.

Pour la fonction de recherche et de suppression dans un AVL, nous savons que ces deux fonctions se basent sur le principe de la fonction d'insertion, nous avons donc dans un premier temps supposé que ces courbes allaient être similaires, puis nous avons ensuite vérifié ces suppositions qui se sont révélées concluante. Nous avons estimé que l'ajout de ces deux graphes similaires à celui de l'insertion serait peut utile ici.

Nombre de rotation pour des suites de nombres :

Nous avons essayé de traiter cette question, mais faute de temps nous n'avons pas trouvé de réponse. Nous savons cependant qu'il faut ajouter 1 à chaque rotation gauche ou droite et 2 à chaque rotation droite_gauche ou gauche_droite et que cette fonction doit être récursive jusqu'à obtenir un arbre AVL équilibré.