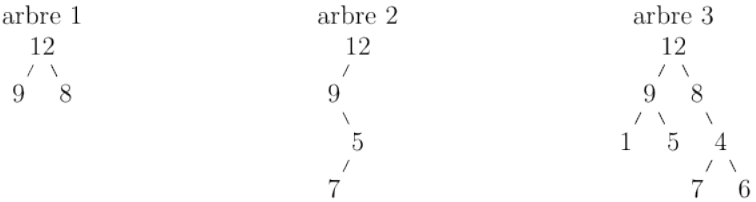
Arbres

Complexité

1 Arbres binaires

1. Ecrire le code de 3 fonctions *arbre1, arbre2, arbre3* qui crée les arbres des figures suivantes



2. Ecrire une fonction *imprimer* qui affiche les valeurs contenues dans un arbre binaire de manière infixée. Testez sr les trois arbres exemples.
3. Ecrire une fonction *nombre_feuilles* qui calcule le nombre de feuilles d'un arbre donné en paramètre.
4. Ecrire une fonction *taille* qui compte le nombre total de nœuds (nœuds internes et feuilles) d'un arbre.
5. Ecrire une fonction *hauteur* qui calcule la hauteur d'un arbre donné en paramètre. Par convention, un arbre vide est de hauteur -1.
6. Etant donné une taille n fixée, il est possible de construire des arbres binaires dont la topologie est différente. Le nombre de ces topologies est donné par la récurrence.

$$\begin{cases} c_0 = 1 \\ c_{n+1} = \sum_{k=0}^{n} c_k c_{n-k} \end{cases}$$

   Ecrire une fonction *nombre_arbres* avec un algorithme récursif s'appuyant sur ces équations de récurrence.
7. Calculer les c_n pour n compris entre 0 et 19. Que constatez-vous pour le calcul des dernières valeurs ?
8. Programmez une seconde version de la fonction *nombre_arbres* nommée *nombre_arbres_efficace* avec un algorithme itératif utilisant un tableau pour stocker les valeurs des nombres c_k avec $k \leq n$.
9. A l'aide de cette seconde version, calculez à nouveau les valeurs de c_n pour n compris entre 0 et 19. Y-a-t'il une amélioration des temps de calcul ?

2 Arbres binaires de recherche

1. Ecrire une fonction *insertion* qui ajoute une valeur à un arbre binaire de recherche
2. Ecrire une fonction *abr1* permettant la création d'un arbre binaire de recherche dans lequel les valeurs sont insérées successivement en suivant l'ordre suivant : 6, 4, 2, 7, 5, 1 ; puis une fonction *abr2* construisant l'arbre suivant l'ordre : 5, 4, 2, 7, 6, 1 ; puis une fonction *abr3* construisant l'arbre binaire de recherche selon l'ordre : 7, 1, 4, 5, 6, 2.
3. Ecrire une fonction qui renvoie 1 si l'arbre qui lui est passé est bien un arbre binaire de recherche, 0 sinon
4. Ecrire une fonction *appartient* qui teste si une valeur est présente dans l'arbre.
5. Modifier cette fonction pour compter le nombre de comparaison effectuées. Sur lequel des trois arbres donnés en exemple y-a-t'il le moins de comparaisons pour la recherche de 0 ? Pourquoi ?
6. Où se trouve respectivement l'élément minimal et maximal d'un arbre binaire de recherche ? Ecrire les deux fonctions qui retournent ces valeurs.

3 Arbres rouge-noir

Le but de cet partie est d'implémenter des arbres rouge-noir (ANR) pour améliorer les arbres binaires de recherche. Vous devez rechercher des informations sur ces arbres sur internet ou dans des livres d'algorithmique.

1. Dans votre rapport, vous expliquerez ce que vous avez compris des arbres rouge-noir. Pour les algorithmes demandés ci-dessous, vous donnerez le pseudo-code et indiquerez la complexité.
2. Implémenter les fonctions d'ajout d'un élément dans un arbre rouge-noir et de recherche dans un élément.
3. Implémentez une expérimentation consistant à construire des ABR [1] et des ANR pour des séquences de n entiers puis à rechercher ces n entiers dans les arbres obtenus.. Vous reporterez dans des graphes, pour différentes tailles de séquences, le nombre de comparaisons effectuées pour l'insertion et la recherche pour les ABR et les ANR.
4. Implémentez une fonction qui affiche par valeur croissante les éléments contenus dans une ANR (cela marche aussi pour les ABR).

1. En utilisant les fonctions de la partie précédente