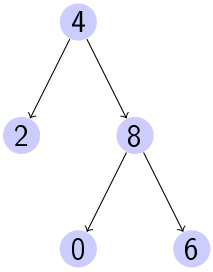
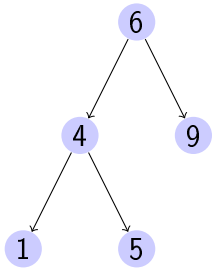
TD-TP n°7

Parcours, arbres binaires de recherche.

Télécharger sur Moodle le fichier intitulé Primitives.rkt, et copier/coller les fonctions de construction sur les arbres en haut de votre éditeur Racket.

Définir en Racket les arbres suivants:

$a_1 =$ 

$a_2 =$ 

Exercice 1. Arbres binaires quelconques

1. Définir une fonction *ajoute1* qui prend en paramètres un arbre de nombres, et ajoute 1 à chaque nœud.

   `(ajoute1 a1)` renvoie '(5 (3 () ()) (9 (1 () ()) (7 () ())))

2. Définir une fonction *postfixe* qui prend en paramètre un arbre de nombre, et renvoie la liste correspondant au parcours postfixé de l'arbre.

   `(postfixe a1)` renvoie `'(2 0 6 8 4)`.

3. Définir une fonction *somme-fils* qui prend en paramètre un arbre de nombres, et remplace la valeur de chaque nœud par la somme des valeurs de tous ses fils.

   `(somme-fils a1)` renvoie '(20 (2 () ()) (14 (0 () ()) (6 () ())))

4. Définir une fonction récursive terminale *somme-pere* qui prend en paramètre un arbre de nombres, et remplace la valeur de chaque nœud par la somme des valeurs de tous ses parents.

   `(somme-pere a1)` renvoie '(4 (6 () ()) (12 (12 () ()) (18 () ())))

Exercice 2. Arbres binaires de recherche

1. Définir une fonction *minimum-abr* qui renvoie le minimum d'un arbre binaire de recherche, en prenant bien en compte qu'il s'agit d'un arbre binaire de recherche.

   `(minimum-abr a2)` renvoie `1`.

2. Définir une fonction *insere-abr* qui insère un élément x à la bonne place dans un ABR a.

   `(insere-abr a2 8)` renvoie `'(6 (4 (1 () ()) (5 () ())) (9 (8 () ()) ()))`.

3. Définir une fonction *est-abr?* qui prend en paramètre un arbre a, et teste si a est un ABR, en renvoyant $\#t$ si oui, et $\#f$ sinon.

   `(est-abr? a1)` renvoie $\#f$. `(est-abr? a2)` renvoie $\#t$.

4. Définir une fonction *supprime* qui prend en paramètres un ABR a et un nombre x, et supprime x de a en le remplaçant quand nécessaire par le minimum du fils gauche.

   `(supprime a2 4)` renvoie `'(6 (1 () (5 () ()) (9 () ()))`.

5. Tester la fonction *infixe* vue en cours sur l'arbre a_2. Que remarquez-vous ? À partir de cette observation, définir une fonction *trie-abr* qui prend en paramètre une liste L, et renvoie la liste triée en passant par la construction d'un arbre binaire de recherche dans lequel on insère les éléments un par un.

   `(trie-abr '(2 5 1 7 6))` renvoie `'(1 2 5 6 7)`.

6. Définir une fonction *filtre* qui prend en paramètres un ABR a, et deux nombres x et y, et renvoie le sous-arbre de a ne contenant que les valeurs comprises entre x et y inclus.

   `(filtre a2 4 7)` renvoie `'(6 (4 () (5 () ())))`.

Exercice 3. Parcours selon un chemin. On définit un chemin comme une liste composée de symboles `g` (pour gauche) ou `d` (pour droite), correspondant aux directions à suivre en partant de la racine de l'arbre.

1. Définir une fonction *chemin-sort?* qui prend en paramètres un arbre a et un chemin C, et qui teste si le chemin spécifié sort de l'arbre ou non.

   `(chemin-sort? a1 '(d d))` renvoie $\#f$ `(chemin-sort? a1 '(g d))` renvoie $\#t$.

2. Définir une fonction *somme-chemin* qui prend en paramètres un arbre a et un chemin C (dont on supposera qu'il ne sort pas de l'arbre), et calcule la somme des valeurs des nœuds qui se trouvent le long du chemin.

   `(somme-chemin a1 '(d d))` renvoie `18`.