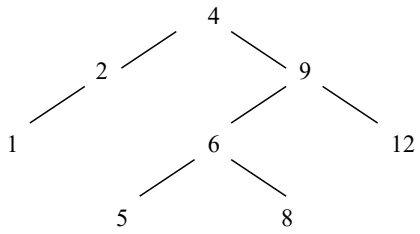
TDTP 8 : arbres binaires de recherche

Soit a l'arbre suivant :



À préparer avant la séance

• Définir une fonction qui, étant donné un arbre de nombres, retourne un arbre dans lequel on a ajouté aux valeurs des feuilles la valeur de la racine.

```
(ajoute_f a) → (4 (2 (5 () ()) ()) (9 (6 (9 () ()) (12 () ())) (16 () ())))
```

On considère maintenant des arbres binaires de recherche, c'est-à-dire des arbres dont les valeurs sont des nombres, et pour lesquels les nœuds sont ordonnés : en tout nœud, le sous-arbre gauche contient des nombres plus petits que celui du nœud, et le sous-arbre droit des nombres plus grands.

• Définir une fonction booléenne qui teste si une valeur appartient à un arbre binaire de recherche.

L'algorithme a été présenté en cours.

```
(appart-abr? 6 a) → #true
(appart-abr? 7 a) → #false
```

À faire pendant la séance

TP • Définir une fonction qui insère un nombre dans un arbre binaire de recherche.

L'algorithme a été présenté en cours.

```
(inserer 3 a) → (4(2(1()())(3()()))(9(6(5()())(8()()))(12()())))
```

TD • Définir une fonction booléenne qui teste qu'un arbre binaire est un arbre binaire de recherche.

```
(abr? a) → #true
(abr? '(4(2(1()())())(9(6(5()())(10()()))(12()())))) → #false
```

TD • Définir une fonction qui trie une liste de nombres sans répétition en passant par la construction d'un arbre binaire de recherche.

TD • Donner la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (a x) ; a ABR, x nb
    (cond ((arbre-vide? a)'())
          ((= (valeur a) x) (list (valeur a)))
          ((< x (valeur a)) (cons (valeur a) (mystere (fils-g a) x)))
          (else (cons (valeur a) (mystere (fils-d a) x))))))
```

TP • Définir une fonction qui calcule le maximum d'un arbre binaire de recherche non vide. Attention à bien tenir compte du fait qu'il s'agit d'un arbre ordonné.

```
(max-abr a) → 12
```

Parcours selon un chemin

On définit un chemin comme une liste de symboles g ou d (pour gauche et droite). Il s'agit d'une liste des directions à suivre à partir de la racine de l'arbre.

TP • Définir une fonction booléenne qui, à partir d'un arbre et d'un chemin, dit si le chemin sort de l'arbre.

```
(chemin-sort? a '(d g g)) → #false
(chemin-sort? a '(g d)) → #true
```

TP • Définir une fonction qui calcule la somme des valeurs des nœuds qui se trouvent le long d'un chemin donné. On supposera que le chemin ne sort pas de l'arbre.

```
(somme-chemin a '(d g g)) → 24
```

Pour s'entraîner (exercices supplémentaires facultatifs)

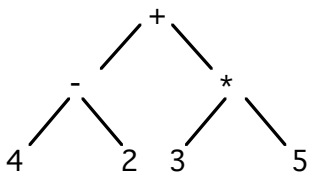
• Définir une fonction qui, étant donné un arbre de nombres A, ajoute des feuilles à A. La valeur de chacune de ces nouvelles feuilles contient la somme des valeurs sur le chemin de la racine aux (anciennes) feuilles.

```
(ajoute-feuilles a)
→ (4(2(1(7()())(7()()))(6()()))(9(6(5(24()())(24()()))(8(27()())(27()()))))
          (12(25()())(25()()))))
```

Arbres binaires pour représenter des expressions arithmétiques

On considère des arbres qui représentent des expressions arithmétiques. Les valeurs des nœuds peuvent donc être soit des nombres, soit les symboles +, -, * et /.

L'expression arithmétique (+ (- 4 2) (* 3 5)) sera par exemple représentée par l'arbre binaire suivant :



• Définir une fonction qui, étant donné un arbre représentant une expression arithmétique, retourne la valeur numérique de l'expression correspondante (par exemple 17 pour l'arbre ci-dessus). On supposera que chaque nœud possède soit 0 fils soit 2 fils.
Attention si vous avez besoin de la fonction eval il faut passer en niveau de langage "Assez gros scheme".

Arbres binaires de recherche

• Définir une fonction qui construit la liste ordonnée des valeurs d'un ABR comprises entre deux valeurs données. Attention à ne pas parcourir inutilement certaines branches de l'arbre.

```
(extrait a 2 6) → (2 4 5 6)
```

• Définir une fonction qui extrait d'un ABR un ABR dont les valeurs sont comprises entre deux valeurs données.

```
((extrait-arbre a 3 8) → (4 () (6 (5 () ()) (8 () ())))
```

Parcours selon un chemin

• Définir une nouvelle version de la fonction `somme-chemin` en considérant que le chemin peut sortir de l'arbre. Dans ce cas, une fois arrivé à une feuille, l'algorithme repart de la racine de l'arbre pour continuer son parcours.

```
(somme-chemin-bis a '(d g g g)) → 28
```