❑ **Exercice 1** : *Représentation d'arbres binaires*

1. Dessiner tous les arbres binaires ayant 3 noeuds.
2. Dessiner tous les arbres binaires ayant 4 noeuds.
3. Dessiner un arbre binaire ayant 8 noeuds et de hauteur maximale (resp. minimale).

❑ **Exercice 2** : *Représentation en C*

On rappelle qu'on a défini en C, un arbre binaire (avec des étiquettes entières) par :

```c
#include <stdbool.h>

struct noeud
{
    struct noeud *sag;
    int valeur;
    struct noeud *sad;
};
```

1. Rappeler la définition de la hauteur d'un arbre binaire et écrire une fonction de prototype `int hauteur(ab arbrebinaire)` qui renvoie la hauteur de l'arbre donné en argument.
2. On rappelle que dans cette implémentation, l'espace nécessaire au stockage des noeuds est alloué dynamiquement à l'aide d'instructions `malloc`. Ecrire une fonction de prototype `void libere(ab* arbrebinaire)` qui détruit l'arbre binaire donné en paramètre, en libérant l'espace alloué par ses noeuds. A la fin de l'appel `ab` vaut `NULL`.

❑ **Exercice 3** : *Représentation en OCaml*

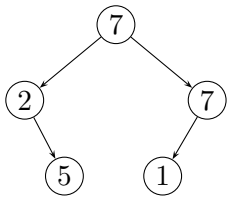
On rappelle qu'on a défini en OCaml un arbre binaire (avec des étiquettes entières) par :

```
type ab = Vide | Noeud of ab * int * ab ;;
```

1. Dessiner l'arbre représenté par :

```
let t = Noeud(
          Noeud(
            Noeud(Noeud(Vide,2,Noeud(Vide,3,Vide)),8,Vide),
            9,
            Noeud(Vide,12,Vide)),
          11,
            Noeud(Noeud(Vide,13,Vide),
            15,
            Vide))
```

2. Donner sa taille et sa hauteur
3. S'agit-il d'un arbre binaire de recherche ? Justifier
4. Donner la représentation en OCaml de l'arbre :

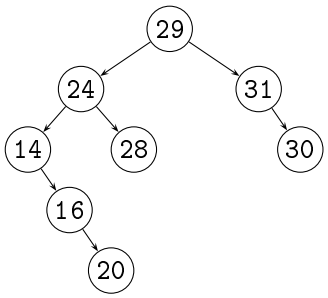


❑ **Exercice 4** : *Un peu de dénombrement*

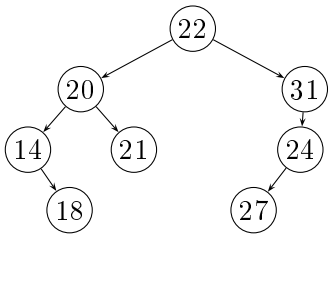
On note T_n le nombre d'arbres binaires à n noeuds.

1. Donner T_0 et déterminer une relation de récurrence liant les $(T_k)_{0\leqslant k\leqslant n}$
   🛟 Utiliser la définition par récurrence des arbres binaires.
2. Vérifier que $T_5 = 42$.
   Le nombre de Catalan d'indice n est défini par :
   $$C_n = \frac{1}{n+1}\binom{2n}{n}$$
   et on prouve que $T_n = C_n$.

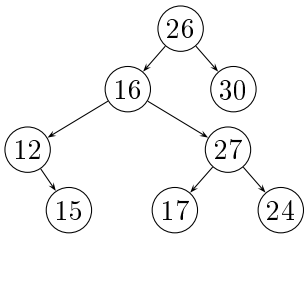
❑ **Exercice 5** : *Parcours d'un arbre binaire*



T_1



T_2



T_3

1. Pour chacun des trois arbres binaires ci-dessus, donner l'ordre des noeuds lors d'un parcours prefixe, infixe et suffixe.
2. Lequel de ces arbres binaires est un ABR ? Justifier

❑ **Exercice 6** : *Un peu de complexité*
On considère la fonction OCaml suivante qui prend en argument un arbre binaire tel que défini par le type de l'exercice 3

```
match ab with
| Vide -> []
| Noeud(g,v,d) ->   v::mystere g @ mystere d;;
```

1. Ecrire une spécification et donner un nom plus approprié à la fonction `mystère`.
2. Rappeler la complexité de l'opérateur `@` et en déduire celle de la fonction `mystere`
3. Proposer une version de cette fonction ayant une complexité linéaire en fonction du nombre de noeuds de l'arbre.
   🛟 Utiliser une fonction auxiliaire avec un accumulateur.

❑ **Exercice 7** : *Reconstruction d'un arbre à partir de parcours*

1. Est-il possible de reconstruire de façon unique un arbre binaire à partir de son parcours préfixe et de son parcours postfixe ?
2. Quel est l'arbre binaire dont le parcours préfixe est `16; 6; 2; 8; 9; 12; 24; 19; 26;` et le parcours infixe `2; 6; 8; 9; 12; 16; 19; 24; 26;`

❑ **Exercice 8** : *Arbre binaire de recherche*

1. Rappeler la caractérisation d'un arbre binaire de recherche par le parcours infixe
2. En utilisant le parcours infixe, écrire une fonction `est_abr` de signature `ab -> bool` qui indique si l'arbre donné en paramètre est un ABR ou non. Donner sa complexité, on supposera qu'on dispose déjà de la fonction qui renvoie le parcours infixe de l'arbre et que cette fonction a une complexité en $O(n)$.
3. On propose la solution suivante pour déterminer si un arbre est un ABR :

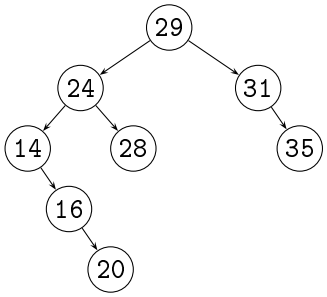
```
let rec est_abr tabr =
  match tabr with
  | Vide -> true
  | Noeud (g, v, d) -> est_abr g && est_abr d && plus_petit g v && plus_grand d v
;;
```

Selon vous, quel est le rôle des fonctions `plus_petit` et `plus_grand` ? Ecrire ces fonctions.

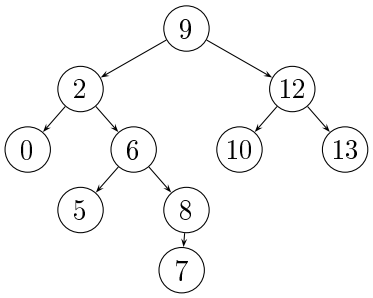
4. La solution précédente est-elle correcte ? Quelle est sa complexité ?
5. Pour tester si un arbre est un ABR, on propose de parcourir l'arbre en donnant l'intervalle de valeurs dans lequel doit se trouver les éléments. Initialement l'intervalle est celui des entiers représentables (qu'on peut obtenir avec `Int.min_int` et `Int.max_int`), puis à chaque fois qu'on descend à gauche (resp. à droite) on met à jour la borne droite (resp gauche) de l'intervalle. Ecrire cette nouvelle méthode pour tester si un arbre est un ABR.
6. Quelle est la complexité de cette fonction ?

❑ Exercice 9 : *Opérations sur un* ABR

1. Rappeler l'algorithme d'insertion d'un élément dans un arbre binaire de recherche
2. Détailler l'insertion de la valeur 17 dans l'arbre ci-dessous (en ayant vérifié qu'il s'agit bien d'un ABR)



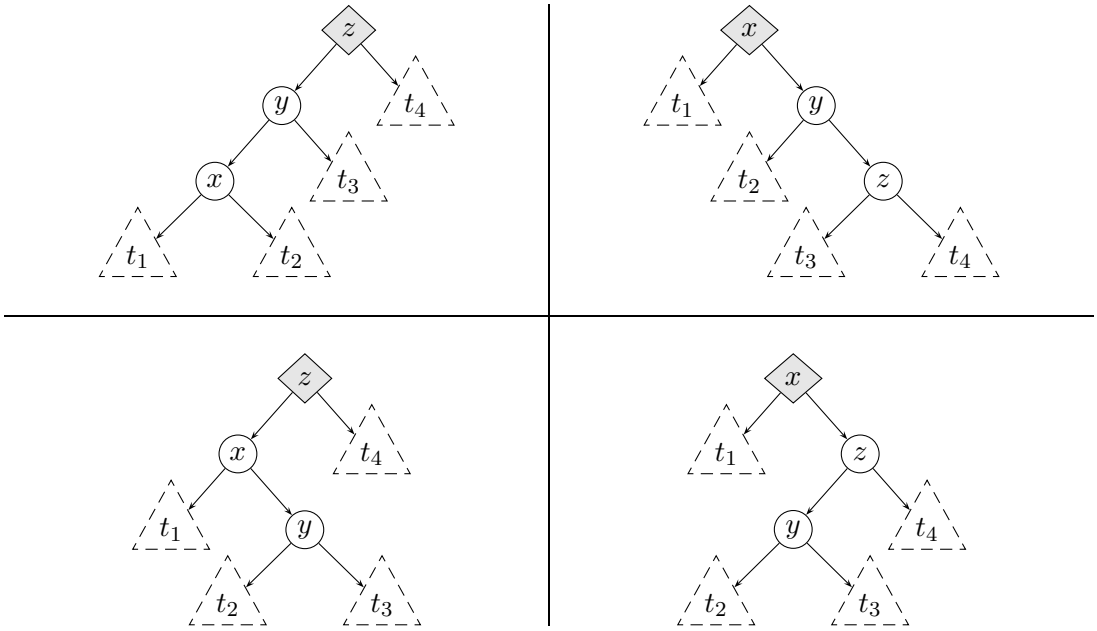
3. Ecrire une fonction `extraire_min` en OCaml de signaure `abr -> int * abr`, qui renvoie un couple composé du minimum d'un arbre binaire de recherche et de cet arbre privé du noeud de valeur minimale. On gère le cas de l'arbre vide avec `failwith`
4. Afin de supprimer une valeur dans un ABR, on propose de procéder de la façon suivante : si l'arbre est vide la suppression est impossible, sinon on descend dans le sous arbre gauche (resp. droit) si la racine est strictement plus grande (resp. petite) que la valeur a supprimé. Si la racine est égale à la valeur à supprimer alors on la remplace par le minimum du sous arbre droit que l'on supprime. Détailler le fonctionnement de cette méthode sur l'arbre suivant d'où on veut supprimer la valeur 2 :



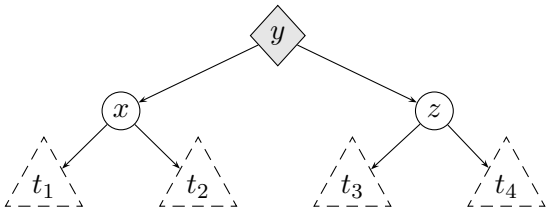
5. Donner une implémentation en OCaml sous la forme d'une fonction `supprime` de signature `abr -> int -> abr`

❑ Exercice 10 : *Equilibrage d'un arbre rouge-noir*

1. Pour insérer un noeud dans un arbre rouge-noir, on commence par utiliser l'algorithme d'insertion usuel dans un ABR et on attribut au nouveau noeud la couleur *rouge*. Quel est alors le seul conflit possible ? (on appellera un tel conflit un *conflit rouge-rouge*).
2. Si le conflit rouge-rouge se situe à la racine, donner une méthode simple pour le résoudre.
3. Si le conflit n'est pas situé à la racine, justifier qu'on se trouve dans l'un des quatres cas suivants où les noeuds rouges sont représentés dans un cercle et les noeuds noirs dans un losange grisé :



4. Montrer qu'en effectuant une ou plusieurs rotations, ces arbres se ramènent à



❑ **Exercice 11** : *Tri par tas*

1. Rappeler la définiton d'un tas binaire
2. Rappeler les relations liant l'indice d'un parent à ceux de ses fils (lorsqu'ils existent) lorsqu'on représente un tas binaire par un tableau
3. Rappeler le principe de l'insertion d'un nouvel élément dans un tas binaire
4. Rappeler le principe de la suppression de l'élément minimal d'un un tas binaire
5. Détailler les étapes et le fonctionnement de l'algorithme du tri par tas pour trier le tableau `[7; 15; 3; 4; 19; 11]`
6. On rappelle ci-dessous la définition du type représentant un tas binaire en OCaml :
   ```
   type 'a heap = {mutable size : int; data : 'a array};;
   ```
   Ecrire une fonction `insere` de signature `'a -> 'a heap -> unit` qui permet d'insérer une nouvelle valeur dans un tas binaire.