TP n°27 - Arbres généraux

Toutes les fonctions de cette section seront codées dans un fichier `NOM_tree.ml`

Les arbres binaires ont une structure très rigide, avec exactement deux sous-arbres à chaque nœud. Il arrive que l'on ait besoin de plus de souplesse, c'est-à-dire de structures arborescentes où chaque nœud peut avoir un nombre variable de sous-arbres. Dans ce cas, on utilise des **arbres généraux**, ou **arbre** sans plus de précision.
L'objet de cette section est la manipulation de structures de données représentant de tels arbres.

I. Définition et construction d'arbres généraux

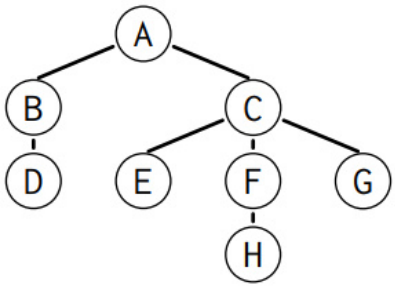
Définition 1 (Arbres généraux)

Un **arbre** général est un ensemble de $n \geq 1$ nœuds structurés de la manière suivante :

- un nœud particulier r est appelé la **racine** de l'arbre ;
- les $n-1$ nœuds restants sont partitionnés en $k \geq 0$ sous-ensembles disjoints qui forment autant d'arbres, appelés **sous-arbres** de r ;
- la racine r est liée à la racine de chacun des k sous-arbres.

Exemple 1

Voici un exemple d'arbre contenant 8 nœuds, ici étiquetés avec les lettres `A`,...,`H`, et dont la racine est `A`.



Comme pour les arbres binaires, la racine est toujours dessinée en haut et les sous-arbres en dessous. Ici, la racine possède deux sous-arbres, contenant respectivement les nœuds $\{\texttt{B}, \texttt{D}\}$ et $\{\texttt{C}, \texttt{E}, \texttt{F}, \texttt{G}, \texttt{H}\}$. Le premier sous-arbre a pour racine `B`, et ainsi de suite.

Définition 2 (Forêt)

Les sous-arbres d'un nœud forment une séquence finie et ordonnée d'arbres que l'on appelle une **forêt**. Une forêt peut être vide.

De façon équivalente, on peut donc définir un arbre comme la donnée d'un nœud, la racine, et d'une forêt, ses sous-arbres.

Définition 3 (Feuille)

Un arbre réduit à un unique nœud est appelé une **feuille**.

Exemple 2

Les nœuds `D`, `E`, `H` et `G` constituent les quatre feuilles de l'arbre ci-dessus

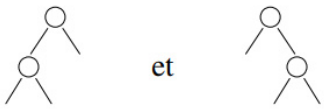
Question 1. Proposez un type OCaml `tree` permettant d'implémenter une structure de données d'arbre général étiqueté polymorphe

Question 2. Créer un objet de type `arbre` correspondant au dessin ci-dessus.

Question 3. Implémenter une fonction `tree_number_of_nodes: 'a tree -> int` permettant de calculer le nombre total de noeuds d'un arbre général. Tester sur de petits arbres et sur l'arbre de la question précédente. On pourra écrire deux **fonctions récursives croisées** définies conjointement avec le mot clé `and`. On proposera dans une premier temps une version non récursive terminale.

Attention (Les arbres binaires... ne sont pas des arbres !).
Aussi surprenant que cela puisse paraître, un arbre binaire **n'est pas un arbre** et ceci pour 2 raisons :

1. Un arbre binaire peut être vide, c'est-à-dire ne contenir aucun nœud, là où un arbre contient toujours au moins un nœud.
   On note également que le dessin d'un arbre binaire inclut de « petites pattes », illustrant la présence de sous-arbres vides, et qu'il n'y a pas lieu de dessiner de telles petites pattes dans le cas d'un arbre.
2. Les arbres binaires font la distinction entre le sous-arbre gauche et le sous-arbre droit. Ainsi, les deux arbres binaires



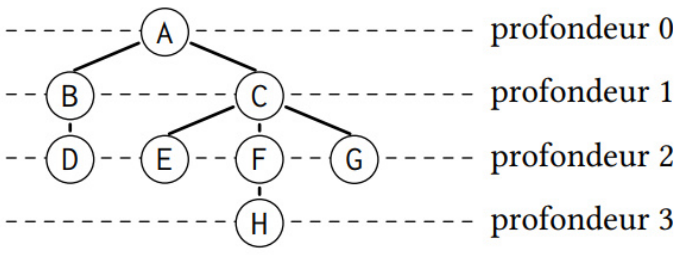
   sont distincts. On parle d'arbres **positionnels** pour les arbres binaires. Pour les arbres généraux, il n'y a qu'un seul arbre contenant deux nœuds :

II. Hauteur d'un arbre général

Définition 4 (Hauteur d'un arbre général)

La **hauteur** d'un arbre est définie comme la plus grande distance entre la racine et un nœud de l'arbre.
En particulier, un arbre réduit à un seul nœud a pour hauteur 0.

De manière équivalente, on peut définir la **profondeur** de chaque nœud, en considérant que la racine est à la profondeur 0, les racines de ses sous-arbres à la profondeur 1, etc., puis définir la hauteur comme la profondeur maximale d'un nœud.



Ainsi, cet arbre a pour hauteur 3, cette distance étant atteinte entre la racine `A` à la profondeur 0 et le nœud `H` à la profondeur 3.

Question 4. Implémenter une fonction `tree_height: 'a tree -> int` qui calcule la hauteur d'un arbre général

III. Parcours d'arbres généraux

Question 5. Implémenter une fonction d'ordre supérieur

```
tree_postorder: 'a tree -> ('a -> unit) -> unit
```

qui applique une fonction à valeur retour vide à tous les nœuds d'un arbre général, en suivant un parcours post-fixé. Validez sur votre exemple.

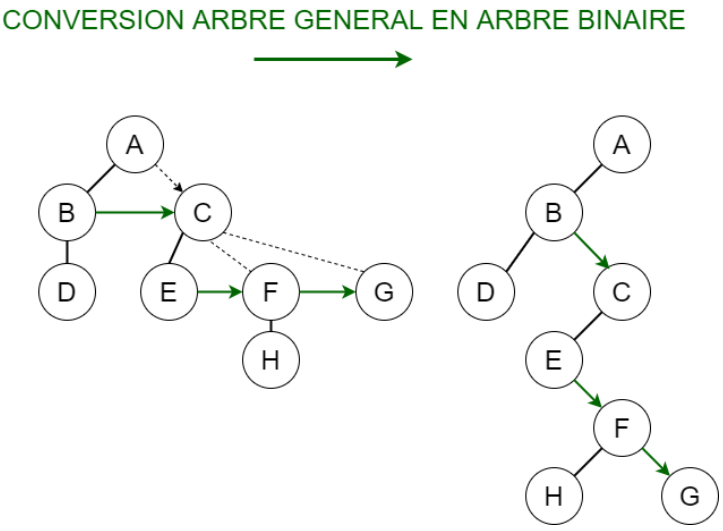
Question 7. Même question pour `tree_preorder` avec l'ordre préfixe. Validez sur votre exemple.

Question 8. Expliquez pourquoi nous n'avons pas implémenté de parcours infixe.

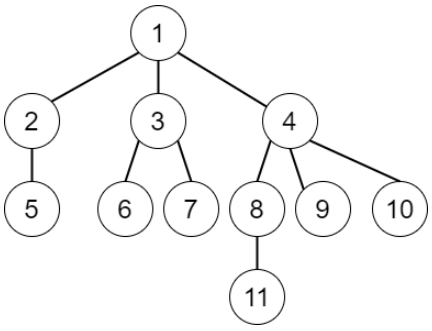
IV. Conversion depuis et vers les arbres binaires

Question 9. Définir un type OCaml implémentant une structure d'arbre binaire étiqueté polymorphe (question de cours)

Convertir un arbre général en arbre binaire est possible : on part de la racine de l'arbre et on redéfinit les relations père fils, comme indiqué par les flèches ci-dessous. Pour faire cela, on considère à chaque étape que seul le premier enfant de la forêt est un fils gauche, le premier frère de la liste étant alors décalé au niveau suivant et devenant le fils droit de ce premier enfant.



Question 10. Développer l'algorithme sur l'arbre général ci-dessous, et dessiner l'arbre binaire obtenu après conversion.



Question 11. Que peut-on dire de manière générale sur les arbres binaires obtenus par cette méthode ?

Question 12. Écrire une fonction `bintree_of_tree: 'a tree -> 'a bintree` qui convertit un arbre général en arbre binaire. Vérifier sur l'exemple de l'énoncé et sur le test déroulé à la main à la question précédente.

Question 13. Quelle précondition doit-on vérifier sur l'arbre binaire donné en entrée de la fonction réciproque `tree_of_bintree` pour coder une conversion parfaitement réciproque (inverse à gauche et à droite) ? Quels ensembles d'arbres sont alors isomorphes ?
Question 13. Écrire une fonction `tree_of_bintree: 'a bintree -> 'a tree` qui convertit un arbre binaire en un arbre général. Votre fonction doit être la réciproque de votre fonction `bintree_of_tree`, tester que c'est bien le cas avec votre fonction !

A la maison : Dans un fichier `NOM_tree.c`, implémentez une structure de donnée d'arbre général en C, avec toutes les primitives ci-dessus. On privilégiera les approches itératives dans ce contexte. On veillera à la lisibilité et la structuration du code et des différents tests. On commentera chaque fonction, et on appliquera rigoureusement les principes de la programmation modulaire et de la programmation défensive.