On utilisera le type suivant d'arbre binaire en OCaml : `type 'a tree = V | N of 'a * 'a tree * 'a tree;;`
Et en C :

```c
typedef struct tree {
    tree *g;
    tree *d;
    int val;
} tree;
```

```c
tree* make_node(int r) {
    tree *t = malloc(sizeof(tree));
    t->g = t->d = NULL;
    t->val = r;
    return t;
}
```

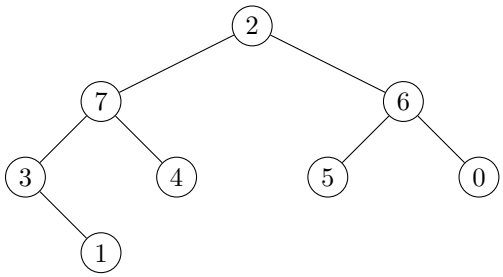
Les prototypes de fonctions sont écrites en OCaml, mais on pourra en écrire en C aussi.

I Définitions

Étant donné un arbre binaire `a` d'étiquette r, de sous-arbre gauche g et sous-arbre droit d, on rappelle que :

- le parcours (en profondeur) **préfixe** consiste à d'abord visiter r puis g (récursivement) et enfin d (récursivement)
- le parcours (en profondeur) **infixe** consiste à d'abord visiter g (récursivement), puis r et enfin d (récursivement)
- le parcours (en profondeur) **suffixe** ou **postfixe** consiste à d'abord visiter g, puis d et enfin r.
- le parcours **en largeur** consiste à visiter les sommets par profondeur croissante : d'abord la racine (profondeur 0), puis les sommets à profondeur 1, puis 2, ...

1. Donner l'ordre de chacun des parcours pour l'arbre suivant :



2. Est-ce que le parcours postfixe est l'inverse (en lisant de droite à gauche) du parcours préfixe?
3. Écrire une fonction `suffixe : 'a tree -> 'a list` renvoyant la liste des sommets d'un arbre dans l'ordre suffixe, si possible en complexité linéaire en le nombre de sommets.

II Reconstruction d'un arbre à partir du parcours préfixe et infixe

1. Quel est l'arbre binaire qui possède les tableaux de parcours préfixe `[|2; 8; 4; 3; 1; 7; 5; 6|]` et de parcours infixe `[|4; 8; 3; 1; 2; 5; 7; 6|]`?
2. Écrire une fonction qui prend un tableau des éléments d'un arbre binaire parcouru dans l'ordre préfixe et un tableau des éléments du même arbre parcouru dans l'ordre infixe, et qui renvoie l'arbre correspondant. On supposera que les étiquettes de l'arbre sont des entiers tous différents.
3. Comment savoir si deux tableaux correspondent aux parcours préfixe et infixe d'un arbre binaire?
4. Est-il possible de reconstruire de façon unique un arbre binaire à partir de son parcours préfixe et postfixe?

III Parcours infixe d'un arbre binaire de recherche

1. Écrire une fonction `est_abr` permettant de déterminer si un arbre a est un ABR.
   On pourra renvoyer 3 informations (utiles pour les appels récursifs) : un booléen déterminant si a est un ABR, l'étiquette minimum de a et l'étiquette maximum de a.
2. Quelle est la complexité de `est_abr`?

Dans la suite, on étudie une autre méthode de déterminer si un arbre est un ABR.

3. Notons $i(a)$ le parcours infixe d'un arbre a. Montrer qu'un arbre a est un ABR si et seulement si $i(a)$ est croissant.
4. Écrire une fonction `croissant : int list -> bool` déterminant si une liste est triée par ordre croissant.
5. En déduire une fonction pour déterminer si un arbre est un ABR. Quelle est sa complexité?
6. Autre application: écrire une fonction pour trier une liste, en utilisant un ABR. Quelle est sa complexité dans le pire des cas?

IV Fusion d'ABR

On veut fusionner deux ABR de tailles n_1 et n_2, i.e obtenir un ABR constitué des éléments des deux ABR.

1. (Méthode naïve) Quelle est la méthode la plus simple qui vous vient à l'esprit pour fusionner deux ABR? L'implémenter.
2. (Cas simple) Écrire une fonction en $O(h)$ pour fusionner deux ABR g et d de hauteurs $\leq h$, en supposant que tous les éléments de g sont inférieurs à ceux de d. En déduire une fonction pour supprimer une valeur dans un ABR.
3. Écrire une fonction renvoyant la liste infixe des sommets d'un arbre à n sommets.
4. Écrire une fonction prenant deux listes d'entiers triées et renvoyant leur fusion triée de taille n, en $O(n)$.
5. Écrire une fonction `array_to_abr` ayant un tableau **trié** `t` de taille n en argument et renvoyant un ABR dont les sommets sont étiquetés par les éléments de `t`, en $O(n)$.
6. En déduire une fonction pour fusionner deux ABR de tailles n_1 et n_2 en complexité $O(n_1 + n_2)$. On pourra utiliser `Array.of_list : list -> array` pour convertir une liste en tableau, en temps linéaire.

V Calcul de rang (order statistics)

Étant donné un tableau `t` de taille n, on souhaite sélectionner le $k$ème plus petit (que l'on appelle élément de rang k).

Méthode simple

1. Comment effectuer un prétraitement en $O(n \log(n))$ sur le tableau, pour ensuite être capable d'obtenir l'élément de rang k en $O(1)$?

Avec un ABR

2. Comment obtenir l'élément de rang 1 d'un ABR? En quelle complexité?

Pour récupérer l'élément de rang k quelconque dans un ABR, on ajoute une information à chaque sommet s: le nombre de sommets du sous-arbre enraciné en s.
On utilise donc le type : `type 'a arb_rang = V | N of 'a * 'a arb_rang * 'a arb_rang * int;;`

3. Écrire une fonction pour ajouter un élément dans un `arb_rang`. Complexité?
4. Écrire une fonction pour supprimer un élément dans un `arb_rang`. Complexité?
5. Écrire une fonction pour récupérer l'élément de rang k dans un `arb_rang` en temps linéaire en sa hauteur (et indépendant de k).