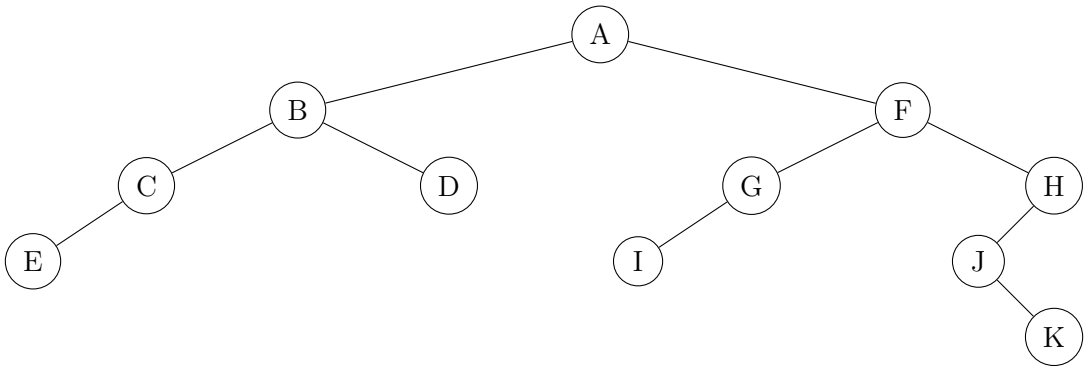
1 Aspect récursif des algorithmes sur les arbres binaires

On prendra comme exemple pour ce cours l'arbre binaire suivant :



1.1 Taille d'un arbre

Rappel : La taille d'un arbre est le nombre de nœuds dans cet arbre.

1. Quelle est la taille de l'arbre exemple ?
2. Quelle est la taille de son sous-arbre gauche ? de son sous-arbre droit ?
3. Quelle relation y a t-il entre la taille d'un arbre et la taille de ses sous-arbres gauche et droit ?
4. En déduire un algorithme récursif permettant de calculer la taille d'un arbre binaire.

```
Fonction taille(arbre)
  SI arbre EST_VIDE
     RENVOYER 0    # l'arbre vide a une taille nulle
  SINON
      RENVOYER 1 + taille(sous_arbre_gauche) + taille(sous_arbre_droit)
```

1.2 Hauteur d'un arbre

Rappel : La hauteur d'un arbre est la profondeur maximum d'un arbre.

1. Quelle est la hauteur de l'arbre exemple ?
   Arbitrage : On prendra comme référence une hauteur nulle pour un arbre vide et donc une hauteur de 1 pour un arbre réduit à sa racine.
2. Quelle est la hauteur de son sous-arbre gauche ? de son sous-arbre droit ?
3. Quelle relation y a t-il entre la hauteur d'un arbre et la hauteur de ses sous-arbres gauche et droit ?
4. En déduire un algorithme récursif permettant de calculer la hauteur d'un arbre binaire.

```
Fonction hauteur(arbre)
  SI arbre EST_VIDE
    RENVOYER 0    # l'arbre vide a une hauteur nulle
  SINON
    RENVOYER 1 + max(hauteur(sous_arbre_gauche), hauteur(sous_arbre_droit))
```

2 Parcours d'un arbre en profondeur

Un parcours **en profondeur** est un **parcours récursif** de l'arbre.

Il y a plusieurs manières de parcourir un arbre en profondeur selon l'ordre dans lequel on affiche le nœud parcouru.

2.1 Parcours dans l'ordre préfixe

Sur notre arbre exemple, le parcours préfixe est A, B, C, E, D, F, G, I, H, J, K.

1. Analysez le sens de parcours en profondeur préfixe.
2. Quel est le parcours préfixe du sous-arbre gauche ? celui du sous-arbre droit ?
3. Comment est construit le parcours de l'arbre en fonction de ses deux sous-arbres ?

2.2 Parcours dans l'ordre suffixe

Sur notre arbre exemple, le parcours suffixe est E, C, D, B, I, G, K, J, H, F, A.

1. Analysez le sens de parcours en profondeur suffixe.
2. Quel est le parcours suffixe du sous-arbre gauche ? celui du sous-arbre droit ?
3. Comment est construit le parcours de l'arbre en fonction de ses deux sous-arbres ?

2.3 Parcours dans l'ordre infixe

Sur notre arbre exemple, le parcours infixe est E, C, B, D, A, I, G, F, J, K, H.

1. Analysez le sens de parcours en profondeur infixe.
2. Quel est le parcours infixe du sous-arbre gauche ? celui du sous-arbre droit ?
3. Comment est construit le parcours de l'arbre en fonction de ses deux sous-arbres ?

2.4 Algorithme en pseudo-code

En résumé, le parcours en profondeur s'écrit ainsi :

```
Fonction Parcours_profondeur(arbre)
  SI arbre EST_VIDE
    Ne rien faire
  SINON
    Traiter racine # ligne à conserver pour parcours préfixe
    Parcours_profondeur(sous_arbre_gauche)
    Traiter racine # ligne à conserver pour parcours infixe
    Parcours_profondeur(sous_arbre_droit)
    Traiter racine # ligne à conserver pour parcours suffixe
```

3 Parcours en largeur

> Le parcours en largeur d'un arbre est le plus simple. On balaie les nœuds d'un même niveau, avant de passer au niveau suivant.

Dans cet algorithme, contrairement au parcours en profondeur, on n'utilisera pas de fonction récursive. Afin de mémoriser les nœuds déjà visités, on utilisera une **file**.

Algorithme en pseudo-code :

```
file_a_traiter = Creer_file_vide()

Fonction Parcours_largeur(arbre)
  Enfiler(file_a_traiter, arbre)
  TANT QUE file_a_traiter NON_VIDE
    arbre = Defiler(file_a_traiter)
    Traiter racine
    SI sous_arbre_gauche NON_VIDE
      Enfiler(file_a_traiter, sous_arbre_gauche)
    SI sous_arbre_droit NON_VIDE
      Enfiler(file_a_traiter, sous_arbre_droit)
```

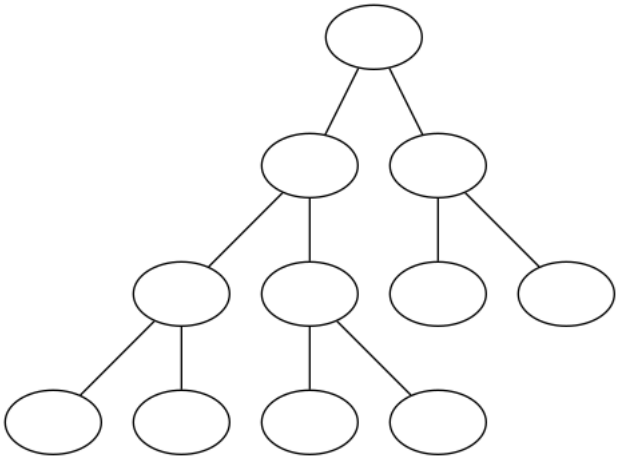
 Exécutez cet algorithme à la main sur l'arbre de démo.
Vous écrirez à chaque étape l'état de la file.

4 Arbre Binaire de Recherche (ABR)

Un **Arbre Binaire de Recherche** est un arbre binaire dont :

1. les nœuds possèdent des valeurs que l'on peut ordonner (par exemple des nombres ou des lettres).
2. pour un nœud donné :
   - tous les nœuds du sous-arbre gauche ont une valeur inférieure à celle de ce nœud.
   - tous les nœuds du sous-arbre droit ont une valeur supérieure à celle de ce nœud.

1. L'arbre exemple est-il un ABR ?
2. Si ce n'est pas le cas, essayer d'échanger les valeurs des nœuds afin de le transformer en ABR avec la structure tassée suivante :



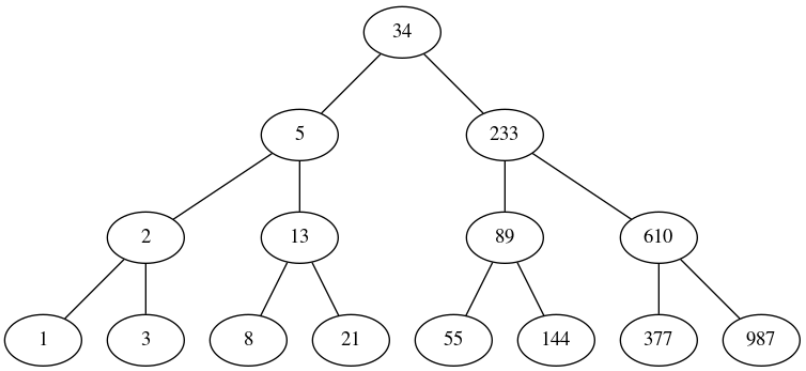
3. De tous les parcours que nous avons étudié (préfixe, infixe, suffixe, largeur), lequel permet d'obtenir les lettres classées par ordre alphabétique ?

4.1 Insérer une valeur dans un ABR

L'insertion se fait par l'algorithme suivant :

- Si l'arbre est vide, on crée un nœud racine avec la valeur à insérer.
- Si la valeur à insérer est plus petite que la valeur de la racine, on l'insère dans le sous-arbre gauche.
- Si elle est plus grande que la valeur de la racine, on l'insère dans dans le sous-arbre droit.

1. Vérifiez que l'arbre suivant est un ABR.



2. Appliquez l'algorithme d'insertion à cet arbre pour insérer la valeur 42.

4.2 Rechercher une clé dans un ABR

Du fait que l'Arbre Binaire de Recherche est une structure ordonnée, si l'arbre est **équilibré**, on peut aller plus vite qu'un parcours arbitraire pour retrouver un nœud , c'est tout son intérêt !

> Le nombre de comparaisons à effectuer ne dépasse pas la profondeur de l'arbre qui est en logarithme de sa taille. (cf début de cours)
> La recherche dans un arbre binaire de recherche équilibré est donc en coût logarithmique.

Rappel : le logarithme (en base 2) traduit le nombre de bits du nombre considéré. Donc si la taille des données double, il ne faut qu'une étape de plus en terme de coût.

Voici l'idée d'un algorithme récursif de recherche efficace :

```
Fonction Rechercher(arbre, valeur)
  SI arbre EST_VIDE
    RENVOYER Faux
  SINON SI valeur < valeur_racine
    RENVOYER Rechercher(sous_arbre_gauche, valeur)
  SINON SI valeur > valeur_racine
    RENVOYER Rechercher(sous_arbre_droit, valeur)
  SINON
    RENVOYER Vrai
```

1. Dans l'ABR ci-dessus, combien d'étapes sont nécessaires pour trouver la valeur "23".