Parcourir un arbre

1. Pourquoi parcourir des arbres

Les arbres sont des structures de données. Les données (informations) sont contenues dans les nœuds des arbres. Donc il faut pouvoir parcourir les arbres pour lire ces données et éventuellement ajouter, supprimer ou modifier des données.

Le parcours d'un arbre peut servir à afficher une arborescence de fichiers, à réaliser des algorithmes de recherche textuelle par exemple ou à gérer des "AI" (intelligence artificielle) de jeux vidéo.

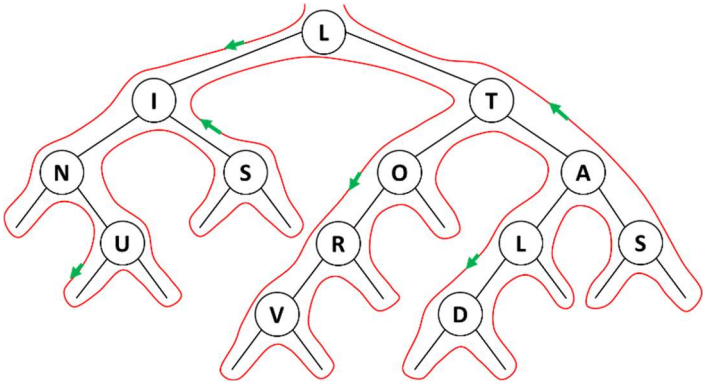
2. Le parcours en profondeur

Il existe trois manières de parcourir un arbre en profondeur. L'idée de ces parcours, c'est de descendre tout en bas de l'arbre avant de se déplacer vers la droite de l'arbre.
Ces parcours sont parfois notés DFS pour Depth First Search.
Ces parcours serviront à réaliser des algorithmes de recherche textuelle par exemple ou à gérer des "AI" (intelligence artificielle) de jeux vidéo.

Le parcours en profondeur consiste à se balader autour de l'arbre en suivant le trait rouge dans le sens des flèches vertes.

Dans le schéma ci-contre, on a rajouté des "nœuds fantômes" pour montrer que l'on peut considérer que chaque nœud est visité 3 fois :

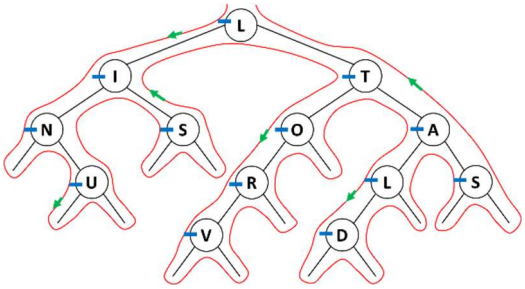
- une fois par la gauche (parcours préfixe) ;
- une fois par en dessous (parcours infixe) ;
- une fois par la droite (parcours suffixe).

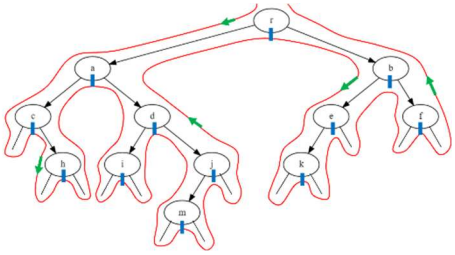


2.1. Parcours préfixe

Dans un parcours préfixe, on liste le nœud la première fois qu'on le rencontre, donc lorsque l'on parcours le nœud par la gauche.

Cela donne : **L I N U S T O R V A L D S**





2.2. Parcours infixe

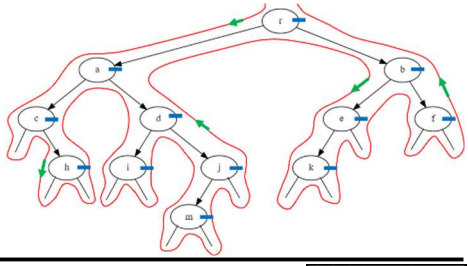
Dans un parcours infixe, on liste le nœud la deuxième fois qu'on le rencontre, donc lorsque l'on parcourt le nœud par le dessous.

✍ Donner le parcours infixe :

2.3. Parcours suffixe

Dans un parcours suffixe, on liste le nœud la dernière fois qu'on le rencontre, donc lorsque l'on parcourt le nœud par la droite.

✍ Donner le parcours suffixe :



2.4. Algorithmes

✍ Voici trois algorithmes récursifs, dire pour chacun d'eux à quel parcours ils correspondent.
Chaque algorithme affiche le résultat du parcours de l'arbre

Algorithme : parcours . . . . . .

```
fonction parcours_1(arbre):
    si l'arbre n'est pas vide alors
        parcours (sous-arbre gauche)
        parcours(sous-arbre droit)
        afficher la racine de l'arbre
```

Algorithme : parcours . . . . . .

```
fonction parcours_2(arbre):
    si l'arbre n'est pas vide alors
        parcours (sous-arbre gauche)
        afficher la racine de l'arbre
        parcours(sous-arbre droit)
```

Algorithme : parcours . . . . . .

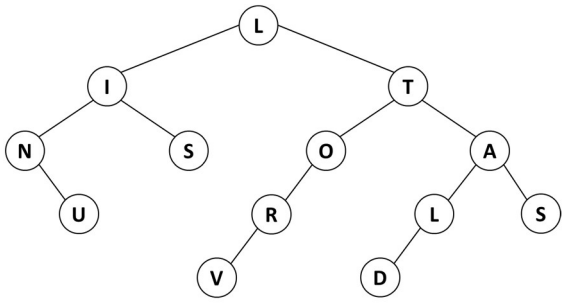
```
fonction parcours_3(arbre):
    si l'arbre n'est pas vide alors
        afficher la racine de l'arbre
        parcours (sous-arbre gauche)
        parcours(sous-arbre droit)
```

2.5. Parcours en profondeur pour des arbres de tuples

✍ Implémenter les trois fonctions ci-dessous afin de parcourir les arbres écrit à partir de tuples et donc compatibles avec la fonction **noeud** ci-dessous :

```
def noeud(nom, fils_gauche, fils_droit):
    """crée un noeud pour constituer un arbre binaire"""
    return (nom, fils_gauche, fils_droit)
```

✍ Vérifier les résultats des parcours préfixe, infixe et suffixe en recréant l'arbre de l'exemple utilisé (ci-dessous).



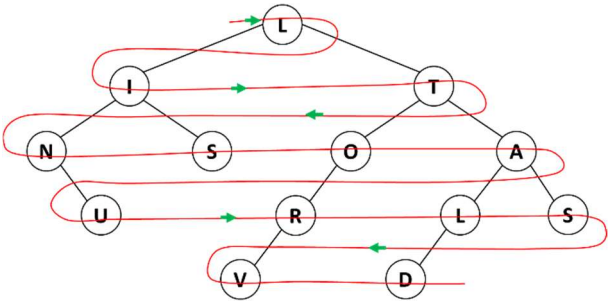
2.6. Parcours en profondeur en programmation objet

✍ Implémenter toutes les fonctions de parcours en profondeur d'un arbre pour pouvoir lire parcours un arbre créer à partir de la classe **Noeud**.

```
class Noeud:
    def __init__(self, nom, fils_gauche, fils_droit):
        self.nom = nom
        self.fg = fils_gauche
        self.fd = fils_droit
```

3. Le parcours en largeur

Le parcours en largeur consiste à explorer, en partant de la racine, chaque nœud d'un même niveau de la gauche vers la droite, puis de passer au niveau inférieur.
Ce parcours est parfois appelé BFS pour Breadth First Search.



Le parcours en largeur donne : **L I T N S O A U R L S V D**

Le parcours peut se faire suivant l'idée qui suit :

- On crée une file.
- On met le nœud racine dans la file.
- Puis tant que la file n'est pas vide :
    - On défile la file en récupérant le nœud.
    - On affiche la valeur du nœud.
    - On enfile le fils gauche du nœud s'il existe.
    - On enfile le fils droit du nœud s'il existe.

3.1. Parcours en largeur en programmation objet

✍ Implémenter le parcours en profondeur en utilisant la programmation objet (utiliser la classe **Noeud**) et le module **`file`** utilisé précédemment dans le chapitre correspondant (si besoin, créer le module avec les programmes vus dans le chapitre).

3.2. Parcours en largeur pour des arbres de tuples

✍ Implémenter le parcours en largeur pour un arbre créé avec des tuples.

Source : https://isn-icn-ljm.pagesperso-orange.fr/NSI-TLE/co/section_chapitre3.html