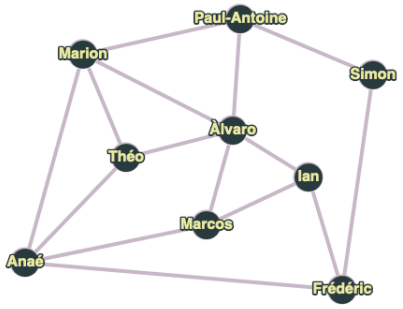
PARCOURS DE GRAPHES

Pour implémenter un graphe, nous avons utilisé un **dictionnaire de liste de successeurs.**



Par exemple, l´implémentation de ce graphe est réalisée ainsi :

```
graphe={'Marion': ['Anaé','Théo','Paul-Antoine','Àlvaro'],
        'Frédéric': ['Anaé','Simon','Ian'],
        'Anaé': ['Marion','Frédéric','Théo','Marcos'],
        'Simon': ['Frédéric','Paul-Antoine'],
        'Ian': ['Frédéric','Marcos','Àlvaro'],
        'Marcos': ['Anaé','Ian','Àlvaro'],
        'Théo': ['Marion','Anaé','Àlvaro'],
        'Paul-Antoine': ['Marion','Simon','Àlvaro'],
        'Àlvaro': ['Marion','Paul-Antoine','Théo','Marcos','Ian'],
        }
```

Pour parcourir un graphe, nous avons vu deux approches :

- L'algorithme de **parcours en largeur** (ou BFS, pour Breadth First Search en anglais) permet le parcours d'un graphe ou d'un arbre de la manière suivante : on commence par explorer un sommet source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc. Comme pour les arbres, il utilise une file.

- **Parcourir un graphe en profondeur** consiste à explorer le graphe de successeur en successeur jusqu´à tomber sur une impasse ou un sommet déjà visité. La récursivité est ici à l´œuvre. En voici un exemple, en partant du sommet *Marcos*.

```
parcours=[]
def dfs(parcours, graph, nom):  #function for dfs
    if nom not in parcours:
        parcours.append(nom)
        for voisin in graph[nom]:
            dfs(parcours, graph, voisin)

# appel de la fonction
dfs(parcours,graphe, 'Marcos')

parcours
```

Retourne : **['Marcos', 'Anaé', 'Marion', 'Théo', 'Àlvaro', 'Paul-Antoine','Simon', 'Frédéric', 'Ian']**