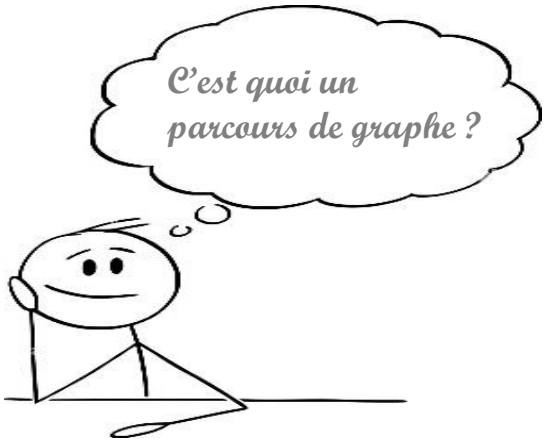
Chapitre 18 - Les graphes – Partie 2

Un parcours de graphe est un algorithme consistant à explorer **tous** les sommets d'un graphe de proche en proche à partir d'un sommet initial. Ces parcours sont notamment utilisés pour rechercher un plus court chemin (et donc dans les GPS) ou pour trouver la sortie d'un labyrinthe...



Pour réaliser un parcours avec une **méthode itérative**, on utilise plus ou moins le même algorithme de base :

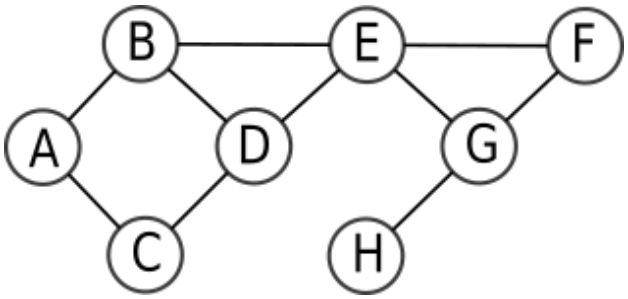
- On visite un nœud A . On crée une structure S qui contiendra au départ uniquement le nœud A .
- Tant que S n'est pas vide :
  - on choisit un nœud *nd* de S (on dit qu'on «visite» s);
  - on ajoute à S tous les voisins de *nd* **pas encore visités**.

Sur un arbre binaire les fils d'un nœud sont nécessairement visités après la visite du nœud parent. Par contre, sur un graphe, ce n'est pas vrai. Le voisin d'un nœud peut avoir **déjà** été visité en tant que voisin d'un nœud précédent...
Il est donc nécessaire de mémoriser les nœuds déjà visités ou découverts (on dira qu'un nœud est découvert lorsqu'on l'ajoute à S).
Le choix de la structure de l'ensemble S est prépondérant :

- Si on choisit une **file** (FIFO): on visitera les nœuds dans l'ordre d'arrivée, donc les plus proches du nœud précédent. On obtient donc un *parcours en largeur* **.**
- Si on choisit une **pile** (LIFO): on visitera d'abord les derniers nœud arrivés, donc on parcourt le graphe en visitant à chaque étape un voisin du précédent. On obtient donc un *parcours en profondeur* .

D'autre part, il est également possible de réaliser un *parcours en profondeur* en utilisant une **méthode récursive**.

1- Parcours iteratif avec une structure de File :

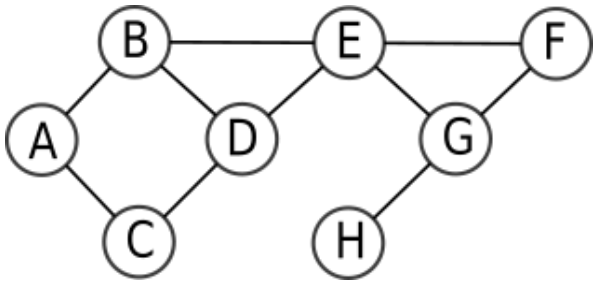


Parcours du graphe ci-contre en partant du nœud A :

Etat de la file f	*Etat de la file f*

Nœuds déjà visités :

Parcours du graphe ci-contre en partant du nœud E :

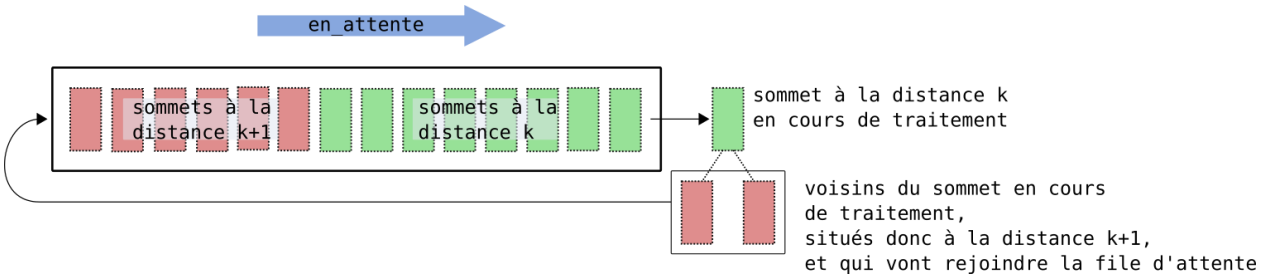


Etat de la file f	*Etat de la file f*

Nœuds déjà visités :

2- PARCOURS ITERATIF AVEC UNE STRUCTURE DE FILE POUR TROUVER LE PLUS COURT CHEMIN :

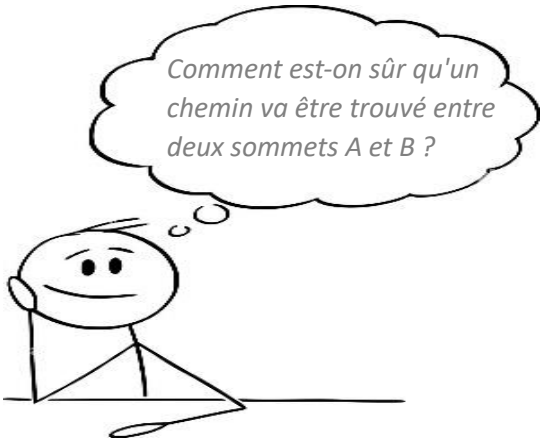
À chaque instant, la file en attente contient des sommets à la distance k+1 et à la distance k du point de départ :



Le parcours avec file découvre les sommets « par cercles concentriques » autour du point de départ (ainsi que le montre la structure de la file d'attente). On découvre d'abord tous les sommets à la distance 1 du point de départ, puis à la distance 2, puis 3, etc.

Un sommet situé à la distance 5 sera découvert en tant que voisin d'un sommet à la distance 4, qui lui-même aura été découvert grâce à un sommet à la distance 3, qui lui-même...

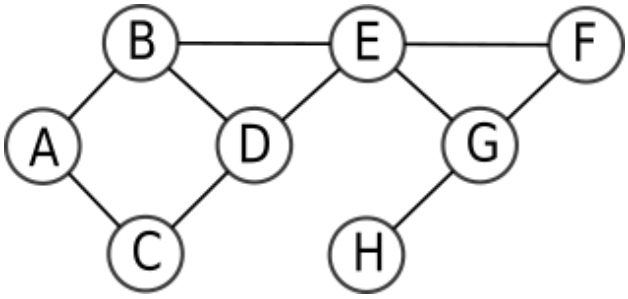
On comprend donc que si on arrive à se souvenir du sommet « parent » de chaque sommet (celui qui lui a permis d'être découvert), on pourra alors reconstituer un chemin permettant de remonter au point de départ. Nous allons pour cela nous servir d'une structure de dictionnaire pour associer à chaque sommet son sommet-parent.



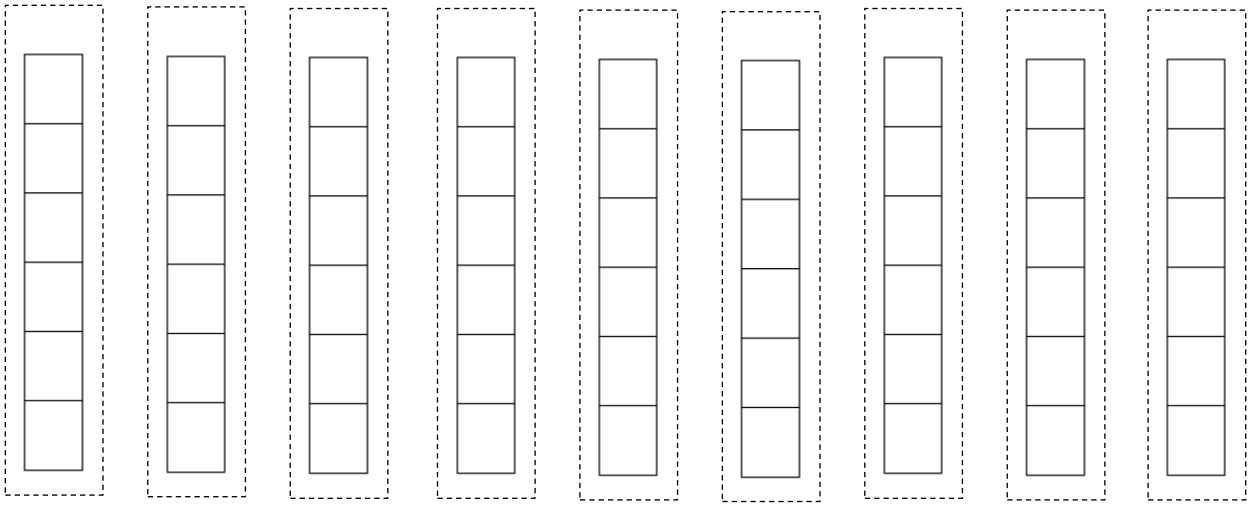
Si le graphe est connexe, tout parcours au départ de A va parcourir l'intégralité du graphe, et donc passera par B à un moment. Un chemin sera donc forcément trouvé entre A et B.

La découverte des sommets par cercles concentriques entre A et B nous assure qu'on ne peut pas rater le point B : s'il est à la distance k de A, il sera forcément visité puisque tous les sommets à la distance k vont passer par la liste d'attente, après les sommets de distance k-1 et avant les sommets de distance k+1. Lorsqu'on remontera de B vers A en passant par les sommets parents successifs, il ne peut y avoir qu'un seul sommet par « couche » : le chemin sera donc exactement de longueur k, il sera donc minimal.

3- PARCOURS ITERATIF AVEC UNE STRUCTURE DE PILE :



Parcours du graphe ci-contre en partant du nœud A :



Nœuds déjà visités :

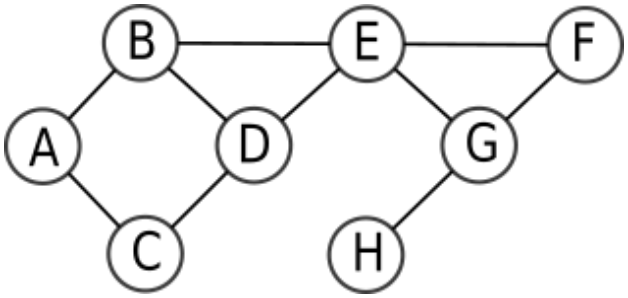
En utilisant une structure de Pile, on parcourt le graphe en profondeur. Le parcours en profondeur est un parcours où on va aller « le plus loin possible» sans se préoccuper des autres voisins non visités : on va visiter le premier de ses voisins non traités, qui va faire de même, etc. Lorsqu'il n'y a plus de voisin, on revient en arrière pour aller voir le dernier voisin non visité.

4- PARCOURS RECURSIF :

Le parcours en profondeur s'écrit naturellement de manière récursive :

```
def parcoursRecursif(noeud, listeDejaVisite) :
    listeDejaVisite.append(noeud)
    for voisin in ......
        if voisin not in listeDejaVisite :
            parcoursRecursif(..... , listeDejaVisite)
    return ....
```

Parcours du graphe ci-contre en partant du nœud A :



Parcours du graphe ci-contre en partant du nœud E :

5- CONCLUSION :

On a vu dans ce chapitre différentes méthodes pour parcourir les nœuds d'un graphe. Un de ces parcours nous permet d'obtenir le « plus court chemin » **pour un graphe non pondéré**.

Pour des graphes pondérés, il existe d'autres méthodes. Certaines explorent tous les chemins possibles, d'autres sont basées sur une **heuristique**.

« Une **heuristique** est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile ».