# NSI – Graphes –>
# Découverte du parcours en profondeur

Rappel : le parcours en largeur d'un graphe consiste, à partir d'un sommet, de lister tous les sommets de proche en proche, c'est à dire commencer à placer ceux qui sont à une distance de 1, puis ceux qui sont à une distance de 2 et ainsi de suite du sommet de départ.

**Philosophie du parcours en profondeur (en anglais Depth First Search) :**

→ Exploration du graphe à partir d'un sommet

→ Détermination de tous les sommets accessibles à partir de ce sommet

→ Pratiquement, pour programmer un parcours en profondeur, on utilise une PILE (dernier arrivé, premier sorti) , dans laquelle on empile les nouveaux sommets découverts, et dans laquelle on pioche les sommets à partir desquels on poursuit l'exploration…

→ le parcours en profondeur combine deux idées : emprunter les embranchements de façon arbitraire et marquer les endroits par lesquels on est déjà passé.

→ le parcours en profondeur dans un graphe est un peu l'équivalent des parcours infixes, préfixe et suffixe pour les arbres binaires.

→ Contrairement au cas des arbres binaires, il est nécessaire de retenir, à l'instar du parcours en largeur, une liste de sommets à visiter.

→ le parcours en profondeur est celui qu'instinctivement on prendrai pour sortir d'un labyrinthe : on essaie un chemin pas déjà emprunté, et dès qu'on ne peut plus (ou qu'on arrive sur un endroit déjà visité), on revient sur nos et on repart dès qu'on trouve un chemin non emprunté.

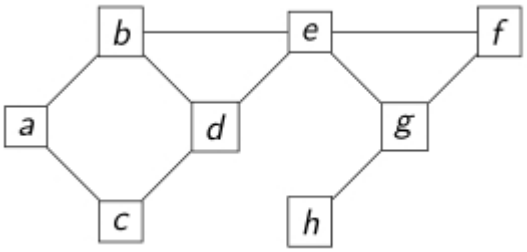
→ comme à chaque sommet on choisit une option (parmi d'autres) il n'existe pas un mais des parcours en profondeur.

→ les applications possibles d'un parcours en largeur sont :

- rechercher tous les sommets atteignables depuis un sommet de départ donné
- la recherche de cycles, mais on verra cela plus tard…

**Exercice 1** : avec un graphe non orienté…

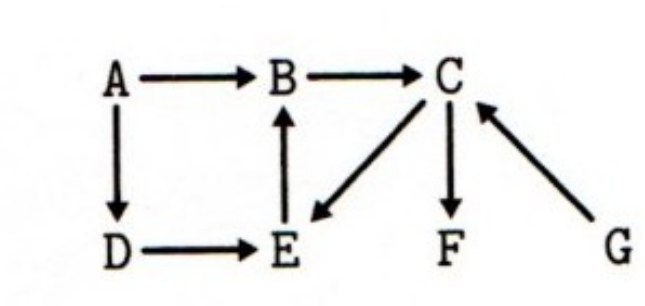
Donner, à partir de chaque sommet, un parcours en profondeur.




```
g1 = {
      'A': ['B', 'C'],
      'B': ['A', 'D', 'E'],
      'C': ['A', 'D'],
      'D': ['B', 'C', 'E'],
      'E': ['B', 'D', 'F', 'G'],
      'F': ['E', 'G'],
      'G': ['E', 'F', 'H'],
      'H': ['G']}
```

**Exercice 2** : avec un graphe orienté…

Donner, à partir de chaque sommet, un parcours en profondeur.




```
g2 = {




}
```

ALGORITHMIQUE  POUR LE PROFONDEUR EN D'UN GRAPHE

PARCOURS EN PROFONDEUR SIMPLE D'UN GRAPHE

```
données :

graphe : un graphe, sous forme de liste d'adjacence
s : le sommet du graphe à partir on commence le parcours

initialisation :

parcours : variable de type list , contient les sommets vus
pile : variable de type list , contient les sommets à visiter, qui sont
les suivants des sommets visités, qu'on empile comme futur sommets
visitables...

parcours est vide
on place s dans la pile

tant qu'il y a des éléments dans la pile :
     on prend un sommet en dépilant la pile
     si ce sommet n'est pas déjà dans le parcours :
          on l'ajoute dans le parcours
          on ajoute chacun de ses suivants dans la pile

Quand cette boucle est achevée, la variable parcours contient le parcours
en profondeur du graphe à partir de la source s
```

1) Programmer l'algorithme de parcours en profondeur décrit ci-dessus.

On pourra appeler cette fonction ***parcours_profondeur_simple(graphe, s)*** .

2) Tester cette fonction avec les graphes g1 et g2 définis précédemment.

3) Programmer la fonction **existe_chemin(g, u,v)**, qui prend comme paramètre un graphe g, un sommet u puis un autre v, et qui renvoie True s'il existe un chemin entre u et v, et False sinon. Tester cette fonction avec le graphe g2, pour lequel on voit qu'il n'y pas de chemin entre A et G.

4) Programmer l'algorithme de parcours en profondeur de manière récursive.