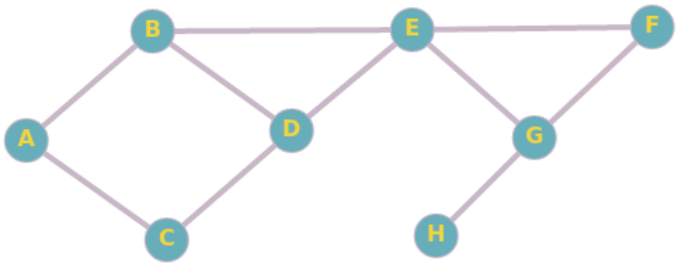
1 Présentation

Les algorithmes sur les graphes sont utilisés dans de très nombreux domaines :
- les logiciels de guidage par GPS (la cartographie routière est un graphe).
- les protocoles de routage réseau comme RIP ou OSPF.
- ...

Dans ce cours, nous prendrons en exemple le graphe suivant :



http://graphonline.ru/fr/?graph=PpgmMcOJNwoDsVMs

Sa matrice d'adjacence se définit ainsi :

```
matrice = [[0,1,1,0,0,0,0,0],
           [1,0,0,1,1,0,0,0],
           [1,0,0,1,0,0,0,0],
           [0,1,1,0,1,0,0,0],
           [0,1,0,1,0,1,1,0],
           [0,0,0,0,1,0,1,0],
           [0,0,0,0,1,1,0,1],
           [0,0,0,0,0,0,1,0]]
```

2 Parcours en profondeur d'abord (Deep First Search - DFS)

Parcourir un graphe consiste à explorer l'ensemble de ses sommets (ou arêtes). Un traitement des sommets peut être effectué au cours du parcours.
Pour le parcours en profondeur, on commence avec un sommet donné et on explore chaque branche complètement avant de passer à la suivante. Autrement dit, on commence d'abord par aller le plus profond possible. Comme pour les arbres, cet algorithme s'écrit naturellement de manière récursive.

Le principe est le suivant :

- on choisit un sommet de départ.
- on le marque comme traité (par exemple en le coloriant en noir)
- pour chaque sommet successeur non traité, on réitère récursivement cet algorithme.

La fonction récursive au cœur de l'algorithme est la suivante :

```
dfs(graphe, sommet)
   Ajouter(Traités, sommet)      # Traités est une liste des sommets traités
   Pour chaque successeur de sommet
      Si successeur nest pas dans Traités
         dfs(graphe, successeur)
```

On peut exploiter cette fonction dans l'algorithme complet suivant :

```
ParcoursEnProfondeur(graphe, depart)
   dfs(graphe, sommet)
      Ajouter(Traités, sommet)
      Pour chaque successeur de sommet
         Si successeur nest pas dans Traités
            dfs(graphe, successeur)
   Traités = ListeVide
   dfs(graphe, depart)
   Renvoyer Traités
```

Exemple : voir l'image gif sur ENT comme exemple, ou sur Graphonline.

 En utilisant le même algorithme, vous réaliserez sur la feuille jointe un parcours alternatif.

Le parcours en profondeur peut par exemple être mis en œuvre dans la recherche d'une solution à un problème de type labyrinthe : on s'enfonce dans le labyrinthe en explorant une voie jusqu'à une impasse ou la sortie. Si on tombe sur une impasse, on revient sur ses pas depuis l'embranchement de la dernière voie explorée (backtracking).

3 Parcours en largeur d'abord (Breadth First Search - BFS)

Là encore, on va s'inspirer du parcours en largeur pour les arbres. Pas de fonction récursive ici, mais on va faire appel à une **file** pour gérer la liste des sommets à découvrir et à traiter. La structure FIFO nous assure que les premiers sommets découverts seront les premiers traités, ce qui constitue la différence majeure avec le parcours en profondeur.
On explore le sommet de départ, puis tous les sommets à une distance de 1 arête de ce sommet, puis tous les sommets à une distance de 2 arêtes, etc...

Le principe est le suivant :

1. On enfile le sommet de départ et on le marque comme découvert.
2. Tant que la file nest pas vide, on réitère les points suivants :
   — on défile (c'est-à-dire on supprime la tête de file).
   — on enfile les sommets successeurs s'ils ne sont pas déjà découverts et on les ajoute à la liste des sommets découverts.
3. On renvoie la liste dans l'ordre des sommets découverts.

Exemple : voir l'image gif sur ENT comme exemple, ou sur Graphonline.

 En utilisant le même algorithme, vous réaliserez sur la feuille jointe un parcours alternatif. Vous noterez à chaque étape l'état de la file.

Voici une première ébauche de l'algorithme :

```
ParcoursEnLargeur(graphe, depart)
   ATraiter = File(depart)        # sommets à traiter
   Découverts = Liste(depart)     # sommets déjà découverts
   Traités = ListeVide            # sommets déjà traités
   Tant que ATraiter.NonVide
      EnTraitement = Defiler(ATraiter)
      Ajouter(Traités, EnTraitement)
      Pour chaque successeur de EnTraitement
         Si successeur nest pas dans Découverts
            Ajouter(Découverts, successeur)
            Enfiler(ATraiter, successeur)
   Renvoyer Traités
```

Pour un parcours en largeur, l'ordre dans lequel on découvre les sommets est aussi l'ordre dans lequel on les traite. On peut donc simplifier l'algorithme précédent en conservant une seule liste Découverts (qui est donc aussi la liste Traités) :

```
ParcoursEnLargeur(graphe, depart)
   ATraiter = File(depart)
   Découverts = Liste(depart)
   Tant que ATraiter.NonVide
      EnTraitement = Defiler(ATraiter)
      Pour chaque successeur de EnTraitement
         Si successeur nest pas dans Découverts
            Ajouter(Découverts, successeur)
            Enfiler(ATraiter, successeur)
   Renvoyer Découverts
```

Un parcours en largeur pourrait par exemple être utilisé pour déterminer à quelle distance se trouve un sommet d'un autre sommet. Ça peut être la base de la recherche d'un plus court chemin entre 2 sommets comme dans les algorithmes de routage.

4 Repérer la présence de cycles

Un cycle est une partie d'un graphe dont les sommets peuvent être reliés en une séquence fermée.

 Quels sont les cycles qui apparaissent dans notre graphe exemple ?

Comment détecter un cycle ?
Sans le savoir nous avons déjà un peu répondu à la question lors du parcours du graphe.

On reprend l'algorithme de parcours en largeur (*remarque* : on pourrait de manière identique utiliser la base du parcours en profondeur) mais on n'exploite pas la liste Decouverts. Tous les successeurs non traités d'un sommet sont enfilés dans la liste ATraiter (même s'ils avaient déjà été découverts depuis un autre sommet). Ceci a pour conséquence de pouvoir enfiler plusieurs fois le même sommet dans la liste ATraiter (à partir de plusieurs prédécesseurs).
Si on se trouve dans un cas où on traite un même sommet une deuxième fois, c'est qu'un autre chemin permettait d'atteindre ce sommet et donc qu'il existe un cycle dans le graphe !

```
Cycle(graphe)
   depart = SommetAuHasard(graphe)
   ATraiter = File(depart)
   Traités = ListeVide
   Tant que ATraiter.NonVide
      EnTraitement = Defiler(ATraiter)
      Si EnTraitement est dans Traités:
         Renvoyer VRAI
      Sinon
         Ajouter(Traités, EnTraitement)
         Pour chaque successeur de EnTraitement
            Si successeur nest pas dans Traités
               Enfiler(ATraiter, successeur)
   Renvoyer FAUX
```

5 Chercher un chemin

La recherche de chemin dans un graphe est quelque chose de très utile : les logiciels de guidage par GPS utilisent de tels algorithmes pour préparer un itinéraire optimisé en distance ou en temps pour aller d'une ville A à une ville B.

L'algorithme le plus connu dans le domaine est l'**algorithme de Dijkstra**.

Exemple en vidéo :  `https://www.youtube.com/watch?v=MybdP4kice4`
Remarque : Cet algorithme a été mis en œuvre à la main dans le contexte du routage sur un réseau (chap. P3-III).

Si on ne s'intéresse pas spécialement à optimiser le chemin, pour simplement répondre à la question « *existe-t-il un chemin entre un sommet A et un sommet B* » il suffit de réaliser un parcours du graphe depuis le sommet A et arrêter le parcours si on tombe à un moment donné sur le sommet B. Si le parcours s'arrête sans rencontrer le sommet B, c'est qu'il n'était tout simplement pas accessible depuis A (il n'y a donc pas de chemin de A à B).

6 Arbre et graphe

On notera pour finir qu'un **arbre** n'est en fait qu'un graphe particulier. En effet, c'est un **graphe non orienté**, **connexe** (il existe un chemin entre toute paire de sommets), et **acyclique**.