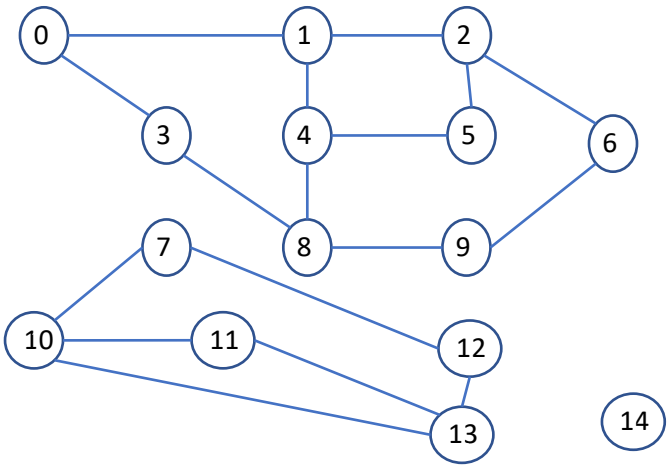
Théorie des graphes

TD/TP 2

Parcourir un graphe (BFS – DFS)
Recherche des composantes connexes dans un graphe non orienté
Chaines et cycles eulériens
Implémentation (C++)

Soit le graphe non orienté G dont la représentation sagittale est donnée ci-dessous :



- **Quel est son ordre ? quelle est sa taille ?**
- **Donner sa matrice d'adjacence.**
- **Combien possède-t-il de composantes connexes ?**
- **Calculer les degrés des sommets.**
- **Admet-il une chaine ou un cycle eulérien ? et ses composantes connexes ?**
- **Quels sommets sont les extrémités d'une chaine eulérienne si elle existe ?**

Julienne Palasi

I. Parcourir un graphe

1. Parcourir en largeur BFS

- **Effectuer un parcours en largeur à partir du sommet 1**. Dérouler l'algorithme en complétant le tableau ci-dessous comme vu en cours.
  Pour chaque sommet on indique sa couleur et son prédécesseur.

Sommets / étapes	0	1	…	Etat de la file
0	B ?	G -	…	1
1	G 1	N -	…	0 …
…				

A chaque étape, indiquer la couleur et le prédécesseur des sommets.

- **Quel est l'ordre de découverte des sommets ?**
- **Dessiner l'arborescence obtenue**

Rappel : le parcours BFS à partir d'un sommet initial s_0 donne les plus courts chemins de s_0 aux autres sommets.

2. Parcourir en profondeur DFS (avec une pile)

Rappel : avec un parcours en profondeur, le résultat obtenu dépend de l'ordre de visite des successeurs de chaque sommet.

- **Effectuer deux parcours en profondeur à partir du sommet 1** : **pour le premier parcours, les successeurs de chaque sommet seront visités suivant l'ordre de leurs numéros ; pour le deuxième, dans l'ordre inverse.**
  Dérouler l'algorithme en complétant le tableau ci-dessous comme vu en cours.

Sommets / étapes	0	1	…	Etat de la pile
0	B ?	G -	…	1
1	G 1	N -	…	…
…				

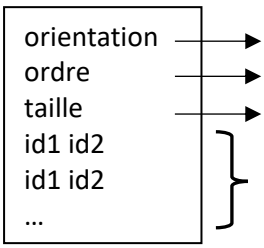
- **Pour chaque parcours, dessiner et comparer les arborescences obtenues**

Julienne Palasi

3. Implémentation des algorithmes : BFS avec une file, DFS en récursif

<u>Objectifs :</u> réaliser un programme c++ capable de :

- **Charger un graphe à partir d'un fichier texte au format suivant :**

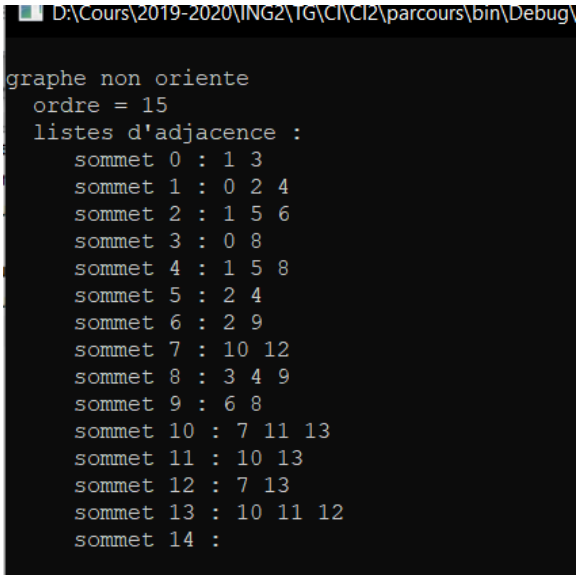


0 : non orienté / 1 : orienté
nombre de sommets. Les sommets sont numérotés à partir de 0.
nombre d'arêtes/d'arcs

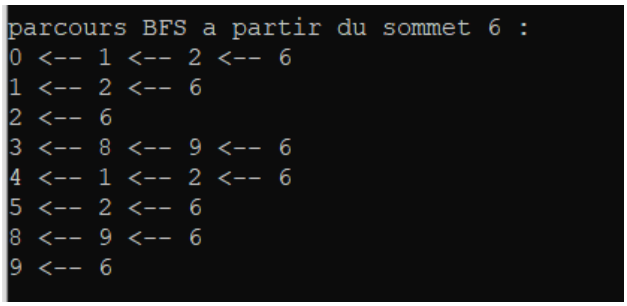
liste des arêtes/arcs. Dans le cas d'un graphe non orienté, chaque arête n'apparait qu'une fois. Il faut y penser en construisant les listes d'adjacences.

graphe.txt

- **Afficher le graphe en console :**
  - L'ordre du graphe
  - Pour chaque sommet :
    - la liste de ses successeurs

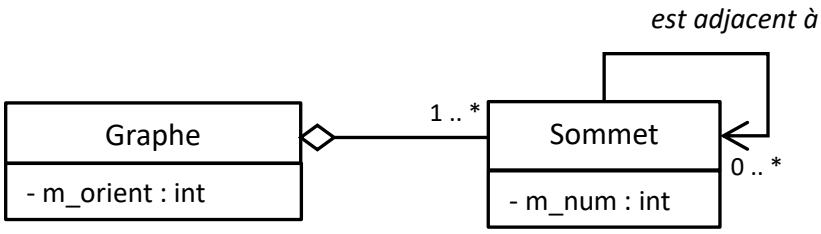


- Demander le numéro d'un sommet
- **Lancer des parcours BFS et DFS sur le sommet choisi et afficher le résultat des parcours** sous la forme id <-- id <-- id



Aide pour la conception/réalisation du programme

- Une représentation mémoire par liste de successeurs est bien adaptée.
  Modèle proposé :



- La classe graphe devrait disposer au moins des méthodes suivantes :
  - ✓ Un constructeur qui charge un graphe en mémoire à partir d'un fichier texte
    Paramètre : le nom du fichier
  - ✓ Une méthode d'affichage *void afficher() const ;*
  - ✓ Une méthode qui réalise un parcours en largeur → cours 2 (slide 28)
    Paramètre : le sommet initial (ou son numéro)
    Retour : la liste des prédécesseurs. Elle peut être stockée dans un vecteur *l_preds* (*l_preds[i]* donne le prédécesseur du sommet numéro i).
    Pour coder un parcours en largeur, on utilise une file : classe *queue*. Pour marquer les sommets découverts, on peut utiliser un vecteur.
  - ✓ Les méthodes qui réalisent le parcours en profondeur, dont la méthode récursive.
    → cours 2 (slide 35)

II. Recherche des composantes connexes (graphe non orienté)

- Ajouter une méthode qui **recherche et affiche les composantes connexes d'un graphe non orienté.**
  (si le graphe est orienté, il faut oublier l'orientation donc modifier les listes d'adjacence …)

```
composante connexe 1 : 0 1 2 3 4 5 6 8 9
composante connexe 2 : 7 10 11 12 13
composante connexe 3 : 14
```

Rappel : un parcours BFS ou DFS à partir d'un sommet initial s_0 marque tous les sommets appartenant à la composante connexe de s_0. Utiliser judicieusement l'une des méthodes codées précédemment !

Pour trouver toutes les composantes connexes, il suffit de relancer un parcours sur un sommet non-marqué … tant qu'il reste des sommets non marqués.

III. Recherche de chaines ou de cycles eulériens

- **Compléter le programme précédent pour qu'il détermine et affiche si un graphe non orienté admet un cycle ou une chaine eulérienne.**

- Facultatif : chercher et coder un algorithme qui trouve et affiche une chaine ou un cycle eulérien s'ils existent.