TP5 — La classe Graphe

Le contexte

L'objectif de ce TP est d'implémenter une classe `Graphe` permettant de gérer des graphes, éventuellement orientés, éventuellement valués (au niveau des arêtes). Un graphe peut être représenté par une matrice d'adjacence ou par une **liste d'adjacence**, associant à chaque sommet son ensemble de voisins.

Si l'implémentation via matrice d'adjacence présente de nombreux avantages de conception, elle est trop gourmande en mémoire : chaque graphe à n sommets est représenté par une matrice de taille $n \times n$, et ce quel que soit son nombre d'arêtes m.

Implémenter une librairie de gestion de graphes via liste d'adjacence, avec -au minimum- les méthodes suivantes :

- ajouter un sommet dans le graphe
- ajouter une arête dans le graphe
- supprimer un sommet du graphe
- supprimer une arête du graphe
- afficher le graphe

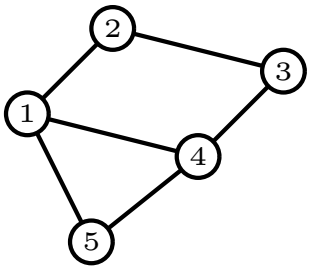
Avant de commencer à coder, il est important de définir les structures de données de la STL qui seront utiles à l'implémentation. Pour un sommet donné, il est nécessaire de stocker tous ses voisins **ainsi que** le poids de ses connexions.

Surcharge d'opérateurs

Visualiser un graphe peut s'avérer utile dans de nombreuses applications, mais il n'est pas toujours évident de savoir comment disposer les sommets et arêtes. De nombreux logiciels permettent d'obtenir une visualisation à partir d'un format de fichier particulier.

Entre autres, `Graphviz` propose de nombreux algorithmes pour dessiner un graphe. La version basique consiste à dessiner des graphes non-orientés sans attributs particuliers :

```
graph graphname {
    1 -- 2;
    1 -- 5;
    3 -- 2;
    4 -- 1;
    3 -- 4 -- 5;
}
```



Utilisation. Le logiciel `XDot` est installé sur les machines, et permet d'afficher des graphes (format `.dot`). Pour une installation sur vos machines personnelles, voir www.graphviz.org.

En utilisant la notion de *surcharge d'opérateurs*, redéfinir l'opérateur `«` afin d'afficher un graphe au format `Graphviz`.

Algorithmes de parcours

La classe `Graphe` maintenant créée, il est nécessaire de l'enrichir avec certains algorithmes afin de la rendre fonctionnelle. La notion d'**algorithme de parcours** constitue une notion de base dans la manipulation des graphes.

En partant d'un sommet donné, un algorithme de parcours **visite** tous les sommets du graphe (si ce dernier est connexe), et permet ainsi d'obtenir un ordre sur les sommets. Il existe deux principales versions d'algorithme de parcours :

— en **profondeur** : les voisins d'un sommet sont visités récursivement, jusqu'à ne plus trouver de sommets à visiter.
— en **largeur** : les voisins d'un sommet sont explorés les uns à la suite des autres.

Implémenter des algorithmes de parcours en largeur et en profondeur dans la classe `Graphe`.

Certaines structures de données de la STL peuvent à nouveau s'avérer utiles.