# Graphes et Algorithmes

## Travaux Pratiques n°5

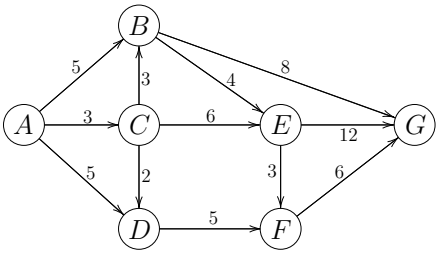
**Université de la Nouvelle-Calédonie, 2024**

## Introduction

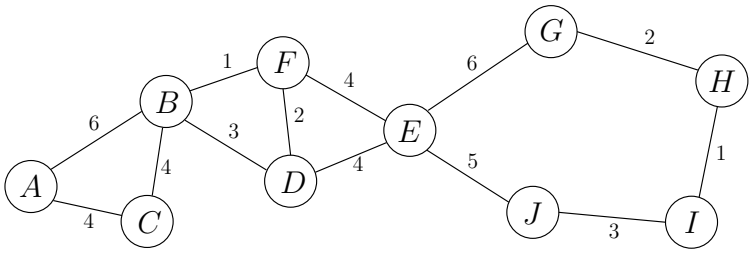
Dans ce TP, vous allez explorer et implémenter trois algorithmes fondamentaux en théorie des graphes : l'algorithme de Floyd-Warshall, la coloration de graphes, et l'algorithme de Bron-Kerbosch pour la recherche de cliques maximales.
Rappel :

- La coloration d'un graphe consiste à déterminer le nombre minimum de couleurs nécessaires pour que deux sommets adjacents soient de couleurs différentes. Si $k$ est le nombre minimum de couleurs nécessaires à la coloration d'un graphe, alors on dit que le graphe est $k$-coloriable.
- Une clique d'un graphe non orienté est un sous-graphe tel que ce sous-graphe est complet (tous les sommets sont adjacents deux à deux). Une clique d'un graphe est dite maximale si et seulement s'il n'existe pas de plus grande clique dans le graphe.

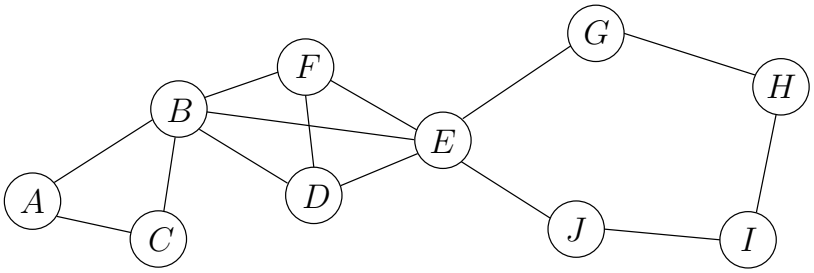
Pour cette séance, vous vous baserez notamment sur les graphes suivants :




(a) Graphe 1




(b) Graphe 2




(c) Graphe 3

Vous pouvez implémenter les graphes sous forme de dictionnaire ou sous forme d'objet (cf. TP4).

## Exercice 1 : Algorithme de Floyd-Warshall (plus court chemin)

L'algorithme de Floyd-Warshall permet de déterminer la longueur du plus court chemin entre tout couple de sommets. Implémentez la fonction `FloydWarshall` à partir du pseudo-code suivant :

**Algorithm 1** `FloydWarshall`$(G)$

```
Initialiser l'ensemble des distances D avec les poids de G, ∞ pour les absences de chemins ;
Pour chaque sommet i
    pour chaque sommet j
        Pour chaque sommet k
            Si D[j][k] > D[j][i] + D[i][k]
                D[j][k] = D[j][i] + D[i][k] ;
```

Vous testerez votre fonction avec les Graphes 1 et 2.
Adaptez votre implémentation pour trouver les plus **longs** chemins.

## Exercice 2 : Coloration d'un graphe

Implémentez la fonction `coloration`, à partir du pseudo-code suivant :

**Algorithm 2** `Coloration`$(G)$

```
Initialiser un dictionnaire color de taille |V| avec toutes les valeurs à -1 (non coloré)
Assigner le premier sommet v_0 à 0
Pour chaque sommet u de V \ {v_0}
    Initialiser un tableau available de taille |V| avec toutes les valeurs à True
    Pour chaque voisin v_i de u
        Si color[v_i] ≠ -1
            Marquer la couleur de v_i comme non disponible
    Trouver la première couleur c disponible dans available
    color[u] = c
```

Vous testerez votre fonction sur le Graphe 3.

## Exercice 3 : Recherche de cliques maximales

Implémentez la fonction récursive `BronKerbosch` pour déterminer une clique maximale, à partir du pseudo-code suivant :

**Algorithm 3** `BronKerbosch`$(R, V, X)$

```
Si V et X sont vides
    R est une clique maximale
Pour tout sommet v dans V
    BronKerbosch(R ∪ {v}, V ∩ N(v), X ∩ N(v)) avec N(v) l'ensemble des voisins de v
    V = V \ {v}
    X = X ∪ {v}
```

La détection de cliques maximales et le comptage des cliques maximales en font un problème #P-complets en $\mathcal{O}(3^{n/3})$, ce qui les rend particulièrement complexes et difficiles à résoudre pour les grands graphes.

Un problème #P-complet est un problème de comptage où l'objectif est de déterminer le nombre de solutions possibles à un problème de décision NP.

Un problème de décision NP ("nondeterministic polynomial time") est un problème pour lequel une solution proposée peut être vérifiée comme correcte ou incorrecte en temps polynomial (c'est-à-dire en un temps raisonnable par rapport à la taille de l'entrée).

Vous testerez donc votre fonction avec le graphe suivant :