

# Graphes et Algorithmes

## Travaux Pratiques n°4 (noté)

Université de la Nouvelle-Calédonie, 2024

- Il s'agit d'un TP **noté**, à faire **seul**.
- Vous avez jusqu'au **Mercredi 16 Octobre 2024 à 23h59 (UTC+11)**.
- Un dépôt moodle sera créé pour que vous soumettiez votre TP.
- Passé ce délai → un point de malus ; Passé 01h00 du matin → dépôt impossible → 0 direct.

## Introduction

Dans ce TP, nous allons définir un graphe par une implémentation en Python 3. Toutes les fonctions doivent être implémentées en Programmation Orienté Objet.

Les sommets de ces graphes sont sans boucle.

Nous supposons que chaque sommet est associé à un identifiant unique qui est le moyen par lequel l'utilisateur l'identifie et le manipule. Deux sommets différents ont ainsi des numéros différents.

Grâce aux TP précédents, vous disposez des structures de données en Python de manière à pouvoir représenter un graphe.

Dans ce TP, on considère le graphe orienté suivant :

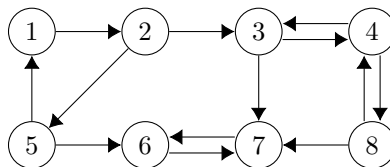
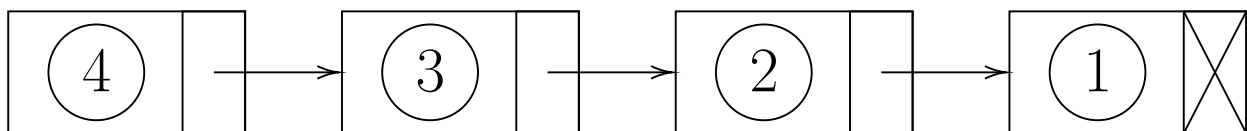

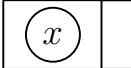



Schéma de la structure de données à utiliser pour représenter une File et une Pile :



  
Sommet

  
Maillon

  
Pointeur vers le  
maillon suivant

# Travail à faire

## Classes à implémenter

Méthodes à implémenter pour la classe **File** :

Méthodes	Entrée	Sortie	Rôle
enfiler	- Un maillon $s$		Enfile un maillon $s$ : Ajoute un maillon à la fin d'une liste chaînée de maillon.
defiler		- Un maillon $s$	Défile un maillon : Retire le premier maillon d'une liste chaînée de maillon.

Méthodes à implémenter pour la classe **Pile** :

Méthodes	Entrée	Sortie	Rôle
empiler	- Un maillon $s$		Empile un maillon $s$ : Ajoute un maillon au début d'une liste chaînée de maillon.
depiler		- Un maillon $s$	Dépille un maillon : Retire le premier maillon d'une liste chaînée de maillon.

Rien ne vous empêche d'implémenter d'autres méthodes dans les classes **File** et **Pile** si vous le souhaitez.

Méthodes à implémenter pour la classe **Graph** :

Méthodes	Entrée	Sortie	Rôle
ajouterSommet	- Un sommet $v_i$	- Une liste de sommets $V$ contenant le sommet $v_i$ ajouté	Ajoute le sommet $v_i$ en paramètre à la liste des sommets $V$ (attribut de la classe).
chercherSommet	- L'indice $j$ d'un sommet	- Le sommet $v_j$	Rend le sommet dont l'indice est $j$ dans la liste de sommets $V$ .
afficherListeSommet			Affiche la liste des sommets présents dans le graphe $G$ .
ajouterArcValue	- Un sommet $v_i$ - Un sommet $v_j$ - Un entier $e_{i,j}$		Ajoute un arc étiqueté $e_{i,j}$ entre les deux sommets $v_i$ et $v_j$ dans la liste de sommets $V$ .
ajouterArc	- Un sommet $v_i$ - Un sommet $v_j$		Fait appel à la procédure <b>ajouterArcValue</b> pour ajouter un arc étiqueté 1.
initMarqueSommet	- Un entier $i$		Initialise la marque de chaque sommet, de la liste de sommets $V$ , à une même valeur $i$ .
initMarqueArc	- Un entier $i$		Initialise la marque de chaque arc de chaque sommet, dans la liste de sommets $V$ , à une même valeur $i$ .

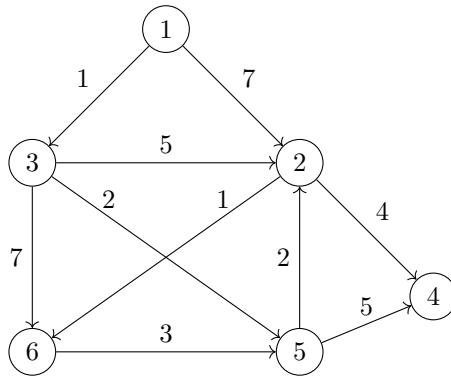
La marque d'un sommet/noeud et d'un arc/arête est une valeur permettant de savoir si l'on est passé sur ce noeud/arc lors d'un parcours. Il peut donc être de type `bool` ou `int`. Ici, nous l'avons défini comme un entier afin de savoir le nombre de fois où l'on est passé sur ce sommet/arc mais rien ne vous empêche de le définir comme un booléen.

Fonctions à implémenter pour le graphe :

Fonctions	Entrée	Sortie	Rôle
parcoursLargeur	- Un sommet $v_i$ .	- Le parcours en largeur.	Effectue le parcours en largeur à partir du sommet $v_i$ .
parcoursProfondeurIteratif	- Un sommet $v_i$ .	- Le parcours en profondeur.	Algorithme itératif qui parcourt en profondeur un graphe à partir du sommet $v_i$ .
parcoursProfondeurRecurusif	- Un pointeur sur un sommet $v_i$ .	- Le parcours en profondeur.	Algorithme récursif qui parcourt en profondeur un graphe à partir du sommet $v_i$ .
ComposanteFortementConnexe	- Un graphe.	- Un dictionnaire des composantes fortement connexes.	Retourne les composantes fortement connexes d'un graphe orienté.

Rappel : En théorie des graphes, une composante fortement connexe d'un graphe orienté  $G$  est un sous-graphe  $g$  tel qu'il existe un chemin reliant tous les sommets les uns aux autres.

On considère le graphe orienté ci-dessous :

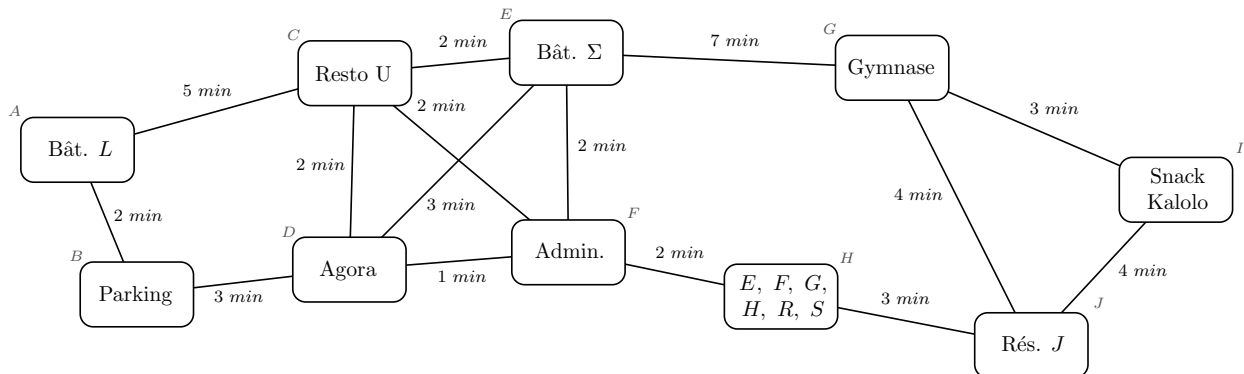


Implémenter l'algorithme de Dijkstra (prononcé [deɪkstra]) :

Fonctions	Entrée	Sortie	Rôle
dijkstra	- Un sommet $v_i$ . - Un sommet $v_j$ .	- La longueur du chemin - Les arêtes/arcs qui composent ce chemin	Affiche le plus court chemin entre deux sommets ( $v_i, v_j$ ).

Testez votre implémentation avec le graphe et différents couples de sommets ci-dessus.

**Un cas un peu plus concret** Un groupe d'étudiants souhaite se rendre au Snack Kalolo depuis le Bâtiment L. Aidez-les en déterminant le chemin le plus court pour s'y rendre grâce au graphe suivant :



## Rapport à rendre

En plus des fichiers `.py`, un rapport `.pdf` sera à rendre contenant un résultat (une capture d'écran de l'exécution) de chaque algorithme à faire s'il fonctionne, sinon un pseudo-code et une présentation des problèmes rencontrés (par algorithme). Une attention particulière sera portée sur la mise en forme du rapport ainsi qu'à la présence de fautes d'orthographe. Bien évidemment, vous pouvez implémenter les fonctions requises sous différents fichiers `.py`.

Dans le cas où vous implémentez des fonctions auxiliaires, il vous est aussi demandé une explication des fonctions ajoutées (rôle, paramètre(s) d'entrée(s) et type de données en sortie) de la même manière que les tableaux précédents. Concernant les classes `File` et `Pile` à créer, vous préciserez si vous les avez implémentées pour un `deque` ou une `list` (ou pour les deux).

**Tous** les algorithmes doivent être commentés pour la compréhension du lecteur. Le tout devra être envoyé dans un dossier compressé nommé : `TP3_GrAl_NOM_PRÉNOM.zip`