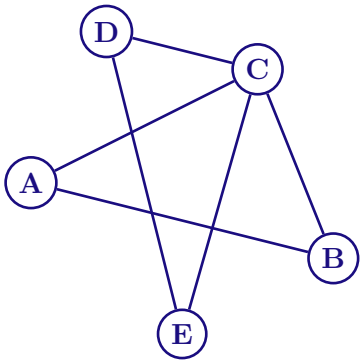
Activité : Implémenter les graphes en Python

Introduction

Un graphe est un ensemble de sommets reliés par des arcs (arêtes).
On donne ci-dessous un graphe G :



1) Combien le graphe G possède de sommets et d'arcs ?

2) On dit que 2 sommets sont adjacents s'ils sont reliés par un arc.
Donner pour chaque sommet du graphe les sommets qui lui sont adjacents.

3) On dresse un tableau qui contient autant de lignes et colonnes que le nombre de sommets du graphe. Chaque colonne et chaque ligne représente un sommet du graphe (dans l'ordre alphabétique).
Compléter le tableau sachant que lorsque les sommets sont adjacents on inscrit le nombre 1. S'ils ne sont pas adjacents, on inscrit 0.

4) On note ce tableau sous forme de matrice écrite uniquement avec les nombres 0 et 1 entre deux parenthèses englobantes. Soit M cette matrice d'adjacence.

a) Le nombre de ligne (et colonnes) donne la dimension de la matrice. Quelle est la dimension n de la matrice M ?

b) On repère chaque valeur de la matrice M par $M[i, j]$ où i est un indice de ligne et j un indice de colonne. Les indices i et j sont compris entre 0 et $n - 1$. Donner les valeurs de $M[0, 1]$ et $M[4, 2]$.

Matrice d'adjacence

Dans cette partie, on représente en Python la matrice d'adjacence par une liste. L'indice de la liste représente un sommet du graphe : le sommet A sera l'indice 0, le sommet B d'indice 1, etc.

1) Écrire la variable M définissant la matrice d'adjacence de notre graphe G sachant que chaque ligne de la matrice est représentée par une liste.

2) Écrire une fonction **est_adjacent** qui vérifie si deux sommets sont adjacents. La fonction prend en paramètre la matrice et les 2 sommets du graphe et renvoie un booléen qui vérifie si les sommets sont adjacents.

3) Écrire une fonction qui initialise une matrice d'adjacence de dimension n donnée avec des valeurs égales à 0. La dimension de la matrice sera donnée en argument de la fonction.

4) On se propose d'écrire une fonction **ajouter_arc** qui modifie la matrice d'adjacence d'un graphe donné. Cette fonction prend en paramètre 2 sommets et modifie la valeur de la matrice en remplaçant la valeur 0 par 1.
Attention : les graphes sont non orientés, ce qui implique que les sommets sont mutuellement adjacents !

5) Écrire une fonction **supprimer_arc** qui supprime un arc entre deux sommets du graphe.

6) On va créer une classe **Graphe_mat** comprenant :

— Les attributs **n** et **mat** où **n** est la dimension de la matrice (nombre de sommets du graphe) et **mat** la matrice d'adjacence ;

— Les méthodes **est_adjacent**, **ajouter_arc** et **supprimer_arc**.

Créer l'objet G, graphe étudié dans l'introduction, en y ajoutant les arcs entre les différents sommets. Pour contrôler que tout se passe bien, on pourra comparer le graphe G avec la matrice M de la première question.

Dictionnaire d'adjacence

L'implémentation précédente ne permet pas de repérer facilement les sommets du graphe. On peut implémenter le graphe G avec un dictionnaire. Chaque sommet du graphe sera une clef du dictionnaire et les valeurs associées à chacune des clefs sera l'ensemble de ces sommets adjacents.

Pour créer ce dictionnaire, on va se familiariser avec la structure de donnée **ensemble** de Python qui s'appuie sur la définition mathématique (les événements en probabilités, solutions d'équations) et qui permet de réunir différentes valeurs et réaliser quelques opérations.

En python, un ensemble de valeurs se crée :

— Soit en écrivant directement les valeurs de l'ensemble entre 2 accolades séparées par des virgules ;

— Soit avec le mot clef **set()** avec au plus une valeur de l'ensemble entre les parenthèses.

Par exemple, pour créer l'ensemble $\{"A","B"\}$, on peut :

— écrire directement $E = \{"A","B"\}$

— écrire $E = \text{set}()$

Une fois l'ensemble créé, on peut ajouter des valeurs avec la méthode **add** : E.add("B") ajoute la valeur B à l'ensemble E.

Une remarque importante, un ensemble ne duplique pas les valeurs ; chaque valeur est unique dans l'ensemble.

1) Créer le graphe G (de l'introduction) sous forme de dictionnaire.

2) Écrire une fonction **init** qui crée un dictionnaire d'adjacence vide. Cette fonction ne prend pas de paramètre.

3) Écrire une fonction **ajouter_sommet** qui ajoute un sommet S à un graphe existant sous forme de dictionnaire d'adjacence. Le sommet et le graphe seront des paramètres de la fonction. Un test préalable de présence du sommet dans le graphe sera nécessaire.

4) Écrire une fonction **ajouter_arc** qui prend en paramètre deux sommets et ajoute un arc entre les deux sommets. Si les sommets n'existent pas, ils seront créés. Le graphe et les deux sommets seront passés en paramètre de la fonction.

Attention : les graphes sont non orientés, ce qui implique que les sommets sont mutuellement adjacents !

5) Écrire une fonction **arc** qui vérifie l'existence d'un arc entre deux sommets. La fonction prend en paramètre les 2 sommets et le graphe.

6) Écrire une fonction **sommets** qui renvoie la liste des sommets du graphe. Le graphe sera passé en paramètre.

7) Écrire une fonction **voisins** qui renvoie la liste des sommets du graphe adjacents au sommet passé en argument dans la fonction. Le graphe sera aussi un paramètre de la fonction.

8) On va créer une classe **Graphe_dict** comprenant :

— L'attribut **adj** où **adj** est un dictionnaire d'adjacence initialement vide ;

— Les méthodes **ajouter_sommet**, **ajouter_arc**, **arc**, **sommets**.

Créer l'objet g, graphe étudié dans l'introduction, en créant son dictionnaire d'adjacence. Pour contrôler que tout se passe bien, on pourra comparer le graphe g avec le graphe G de la première question.

De l'un à l'autre

1) Créer une fonction qui crée un dictionnaire d'adjacence d'un graphe à partir de sa matrice d'adjacence.

2) Créer une fonction qui crée une matrice d'adjacence d'un graphe à partir de son dictionnaire d'adjacence.