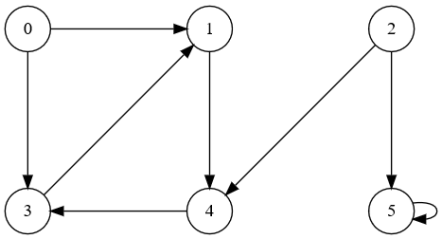
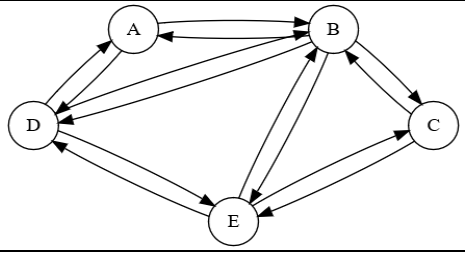
TD 01 : Graphes - 1

1 Représentation d'un graphe

Un graphe est défini par un couple $(\mathcal{S}, \mathcal{A})$ où $\mathcal{S} = \{s_0, s_1, \dots, s_{n-1}\}$ est un ensemble de sommets et $\mathcal{A}$, une partie de $\mathcal{S} \times \mathcal{S}$, dont les éléments, ayant la forme de couples de la forme (s_i, s_j), est l'ensemble des arcs reliant les sommets du graphe.
Si un arc est décrit par un couple (s_i, s_j), on dit qu'il a pour origine s_i et pour extrémité s_j.

On considère les graphes orientés suivants :

Graphe G_1	**Graphe G_2**
	

On définit dans un fichier **.txt**, chacun de ces deux graphes avec la convention suivante :

- **sur la première ligne figurent, espacés par une espace, les noms de tous les sommets du graphe** ;
- le fichier comporte ensuite autant de lignes que d'arcs dans le graphe ;
- sur chacune de ces lignes, représentant chacune un arc, figure le nom du sommet origine de l'arc, puis, espacé par une tabulation, le nom du sommet extrémité de l'arc.

On a joint au TD les deux fichiers, **graphe1.txt** et **graphe2.txt**.

On rappelle la syntaxe pour ouvrir en lecture un fichier

```
f = open(<chemin absolu ou relatif vers le fichier>, mode = 'r')
... # <- traitement du fichier
f.close()
```

ou bien

```
with open(<chemin absolu ou relatif vers le fichier>, mode = 'r') as f :
    ... # <- traitement du fichier
```

Pour la lecture dans un fichier, on peut omettre de spécifier le mode, `'r'`, car ce mode d'ouverture est le mode par défaut.

PARTIE 1 Module `os`
On souhaite écrire, dans un script **ex1.py**, une suite d'instructions permettant de construire la matrice d'adjacence du graphe G_1, à partir de la lecture du fichier le décrivant, **graphe1.txt**.

Question 1. Dans la console interactive, vérifier, à l'aide de la commande `getcwd` du module `os`, quel est le répertoire de travail courant de votre IDE (environnement de travail intégré).
Consulter l'aide de la commande `chdir` de ce même module, à l'aide de la fonction `help`, afin de pouvoir imposer que le répertoire de travail soit le dossier **TD01** du sous-répertoire **Info** que vous aurez préalablement créé dans votre dossier personnel situé à côté du dossier **_travail** dans le répertoire de votre classe sur le serveur pédagogique.
Sur un PC personnel, on imposera le chemin vers le dossier qui contient les deux fichiers, et dans lequel on écrira aussi le script **ex1.py**.

Question 2. À l'aide de la commande `listdir` du module `os`, vérifier que vous avez bien copié les fichiers du TD, **graphe1.txt** et **graphe2.txt** dans le répertoire défini à la **question 1**.

Question 3. Créer un fichier **ex1.py** dans ce même répertoire, y définir une importation du module `os` et imposer à l'aide de la commande `chdir`, et d'un chemin absolu, que le répertoire de travail courant soit celui qui contient votre fichier **.py** et les fichiers décrivant les graphes 1 et 2.

PARTIE 2 Traitement du fichier **graphe1.txt**

Question 4. Écrire une première instruction dans le fichier **ex1.py**, vérifiant, à l'aide de l'instruction `assert`, que le fichier **graphe1.txt** est bien présent dans la liste des fichiers du répertoire de travail courant.
On rappelle que l'instruction `assert expr`, où `expr` est une expression booléenne, déclenche une exception `AssertionError`, provoquant l'arrêt du programme en cours d'exécution, si l'expression `expr` n'est pas évaluée à `True` (et ne fait rien sinon).

On rappelle les méthodes `.strip()`, et `.split()`, qui, appelées sur des objets de type `str`, permettent, respectivement :

- méthode `.strip()` : appelée sans argument autre que l'objet auquel on l'applique, supprime tous les caractères d'espacement, `' '` (espace), `'\t'` (tabulation), ou `'\n'` (fin de ligne), situés au début ou la fin de la chaîne à laquelle on l'applique :

```
>>> " \n ab\n cd e\tfg \n"
"ab\n cd e\tfg"
```

```
>>> "abcdef\n"
"abcdef"
```

- méthode `.split(sep)` : appelée avec un séparateur, `sep`, placé en argument, en complément de l'objet auquel applique la méthode, renvoie une liste composée de toutes les sous-chaînes (éventuellement vides), situées de part et d'autre de chaque occurrence du séparateur :

```
>>> "abracadabra".split('ab')
["", "racad", "ra"]
```

```
>>> "1;234;5;;7".split(";")
["1", "234", "5", "", "7"]
```

On rappelle la fonction de conversion de type, `int`, prend en argument une chaîne de caractères correspondant à une représentation acceptable d'un entier et renvoyant l'entier représenté :

```
>>> int("251")
251
```

Question 5. Écrire, dans le fichier **ex1.py**, une séquence d'instructions permettant d'ouvrir le fichier **graphe1.txt** et construisant, la liste, `LS`, de ses sommets, sous la forme d'une liste d'entiers, et la liste, `LArcs`, de tous les arcs du graphe G_1, sous la forme d'une liste de couple d'entiers.
À l'ordre près des sommets et des couples, on devra donc obtenir :

```
>>> LS, LArcs
([0, 1, 2, 3, 4, 5], [(0, 1), (1, 4), ..., (5, 5)])
```

Question 6. Compléter le script pour construire, à partir de la liste `LArcs`, la matrice d'adjacence, `M1`, du graphe G_1. La matrice `M1` aura la forme d'une liste de liste d'entiers égaux à 0 ou 1.
Considérant le graphe G_1, cette liste de listes devra coder la matrice suivante :

$$M_1 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

et avoir donc la forme :

```
M1 = [[0, 1, 0, 1, 0, 0],
      [0, 0, 0, 0, 1, 0],
      [0, 0, 0, 0, 1, 0],
      [0, 1, 0, 0, 0, 0],
      [0, 0, 0, 1, 0, 0],
      [0, 0, 0, 0, 0, 1]]
```

Question 7. On considère encore dans cette question, uniquement des graphes dont les sommets sont étiquetés par des numéros pris dans l'intervalle $[\![0, n-1]\!]$ où n est l'ordre du graphe, et décrits dans un fichier texte d'extension **.txt**, formatté comme indiqué dans le préambule.
On évaluera la complexité de l'algorithme en fonction de l'ordre n du graphe (nombre de sommets) et de sa taille m (nombre d'arêtes).
Dans un script **fonctions_td01.py**, en reprenant les réponses aux questions **4** et **5**, définir trois fonctions :

- `lire1_graphe`, prenant en argument le chemin vers un répertoire r et un nom de fichier f (sans extension), et renvoyant sous la forme d'un tuple, l'ordre, la taille, la liste des sommets et la liste des arcs du graphe décrit par le fichier f situé dans le répertoire r ;
- `arcs_vers_mat1`, prenant en argument l'ordre d'un graphe et la liste de ses arcs, et renvoyant la matrice d'adjacence du graphe.
- `charger1_graphe`, prenant en argument le chemin vers un répertoire et le nom d'un fichier de ce répertoire décrivant un graphe et renvoyant la matrice d'adjacence du graphe. Cette fonction fera appel aux fonctions `lire1_graphe` et `arcs_vers_mat1`.

Tester la fonction `charger1_graphe` avec le graphe G_1.

PARTIE 3 Traitement du fichier **graphe2.txt**

Question 8. Ouvrir à l'aide d'un éditeur de texte le fichier **graphe2.txt**, et constater qu'un sommet est absent de la liste des sommets, et que certains arcs du graphe ne sont pas codés dans le fichier.

Question 9. Dénombrer le nombre d'arcs non représentés dans le fichier, puis, dans un fichier **ex2.py**, écrire une séquence d'instructions permettant d'ouvrir le fichier **graphe2.txt**, puis d'écrire un nouveau fichier, **graphe2c.txt**, dans lequel

- la liste des sommets est complète sur la première ligne ;
- tous les arcs du graphe sont représentés.

Question 10. Dans le fichier **ex2.py**, écrire une séquence d'instructions permettant, connaissant la structure du fichier **graphe2c.txt**, de lire le contenu de ce fichier et de construire :

- une liste `LS` de ses sommets et de définir une variable `n` égale à l'ordre, n, du graphe ;
- un dictionnaire `DS` établissant une numérotation de ses sommets, de 0 à $n-1$;
- une liste `LArcs` de tous les arcs du graphe et une variable `m` égale à m, la taille du graphe.

L'ordre dans les listes et dans la numérotation n'étant pas imposés, on aura par exemple :

```
>>> LS
['A', 'B', 'C', 'D', 'E']
>>> DS
['A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4]
>>> LArcs
[('A', 'B'), ('A', 'D'), ('B', 'C'), ('B', 'E'), ('B', 'D'), ('B', 'A'),
 ('C', 'E'), ('C', 'B'), ('E', 'D'), ('E', 'B'), ('E', 'C'), ('D', 'A'),
 ('D', 'B'), ('D', 'E')]
```

Question 11. Compléter le fichier par une séquence d'instructions permettant, à partir de la liste `LArcs`, de définir, sous la forme d'une liste de listes, une matrice d'adjacence, `Madj`, pour le graphe, cohérente avec la numérotation choisie pour les sommets, de sorte que `Madj[i][j]` soit égal à 1 si un arc existe du sommet numéroté i vers le sommet numéroté j et à zéro sinon.

Question 12. Compléter le fichier par une séquence d'instructions permettant, à partir de la liste `LArcs`, de définir, sous la forme d'une liste de listes, une liste des listes d'adjacence, `LLadj`, pour le graphe, cohérente avec la numérotation choisie pour les sommets, de sorte que `LLadj[i]` soit une liste des noms des sommets directement accessibles depuis le sommet numéroté i.

Question 13. Compléter le fichier par une séquence d'instructions permettant, à partir de la liste `Larcs`, de définir, un dictionnaire, `DLadj`, dont les clés sont les noms des sommets et les valeurs des listes de noms de sommets, la liste associée à chaque clé étant la liste des sommets directement accessibles depuis le sommet-clé.

Question 14. On considère dans cette question, des graphes décrits dans un fichier formattés comme l'est le fichier **graphe2c.txt**. Dans le script **fonctions_td01.py**, en reprenant les réponses aux questions **10** à **13**, définir cinq fonctions :

- `lire2_graphe`, prenant en argument un chemin vers un fichier f (incluant l'extension du fichier), et renvoyant sous la forme d'un tuple, l'ordre, la taille, une liste des noms des sommets, et une liste des arcs sous la forme d'une liste de couples de noms de sommets, pour le graphe décrit par le fichier f ;
- `charger2_graphe`, prenant en argument un chemin vers un fichier f décrivant un graphe et renvoyant un dictionnaire des listes d'adjacence des sommets du graphe, sous la forme vue en question **13**.
- `numerotation_sommets`, prenant en argument un dictionnaire de listes d'adjacences et renvoyant un dictionnaire définissant une numérotation des sommets comme en question **10** ;
- `DLadj_to_Madj`, prenant en argument un dictionnaire définissant une numérotation des sommets d'un graphe et un dictionnaire des listes d'adjacences de ses sommets, et renvoyant la matrice d'adjacence du graphe cohérente avec la numérotation des sommets.
- `Madj_to_DLadj`, prenant en argument un dictionnaire définissant une numérotation des sommets d'un graphe et une matrice d'adjacence du graphe cohérente avec la numérotation des sommets et renvoyant un dictionnaire des listes d'adjacences de ses sommets.

Tester ces fonctions avec le graphe G_2.

PARTIE 4 Exploitation d'une matrice d'adjacence ou des listes d'adjacence

On répondra aux questions dans un fichier **ex3.py**. On effectuera une importation des fonctions depuis le fichier **fonctions_td01.py**, elles serviront à charger les deux graphes-exemples.

Chacune des fonctions demandées sera testée sur le graphe G_1 ou le graphe G_2.

Question 15. On dit qu'un arc (u, v) est incident à un sommet s s'il a pour extrémité s, *i.e.* si $v = s$.
Écrire une fonction `incidents(Madj, i)`, prenant en argument une matrice d'adjacence et un numéro de sommet, i, et renvoyant une liste des numéros des sommets origine d'un arc incident au sommet numéroté i.

Question 16. On appelle *degré entrant*, respectivement *degré sortant*, d'un sommet, le nombre d'arcs ayant pour extrémité, respectivement pour origine, ce sommet.
Écrire deux fonctions `degre_entrant(Madj, i)`, et `degre_sortant(Madj, i)`, prenant en argument une matrice d'adjacence et un numéro de sommet, i, et renvoyant, respectivement, les degrés entrant et sortant du sommet numéroté i. Évaluer leur complexité.

Question 17. Un chemin de longueur k ($k \geq 1$) dans un graphe dont les sommets sont $s_1, s_2, \dots, s_{n-1}$, est une suite de k sommets $s_{i_1}, s_{i_2}, \dots, s_{i_k}$ dont les sommets consécutifs sont reliés par un arc.

a. Démontrer par récurrence que si M une matrice d'adjacence décrivant le graphe, alors M^k, $k \geq 1$, est telle que $M^k_{i,j}$ dénombre les chemins de longueur k reliant le sommet numéroté i au sommet numéroté j.

b. Écrire une fonction `puissance`, prenant en argument une matrice d'adjacence, M, et un entier $k \geq 1$ et renvoyant la puissance $k^{ème}$ de M.
Quelle est la complexité de l'algorithme utilisé en fonction de k et de n, l'ordre du graphe ?

Question 18. Un graphe possède une boucle s'il existe dans le graphe un arc d'un sommet vers lui-même.

a. Écrire une fonction `existe_boucleM`, prenant en argument la matrice d'adjacence d'un graphe et renvoyant un booléen indiquant si le graphe présente ou non une boucle. Évaluer sa complexité en fonction de l'ordre, n, ou de la taille, m, du graphe.

b. Écrire une fonction `existe_boucleLA`, prenant en argument un dictionnaire des listes d'adjacence des sommets d'un graphe et renvoyant un booléen indiquant si le graphe présente ou non une boucle. Évaluer sa complexité en fonction de l'ordre, n, ou de la taille, m, du graphe.

Question 19. Un graphe est dit non orienté si pour tout arc $s_i \to s_j$ dans le graphe, l'arc $s_j \to s_i$ est aussi dans le graphe.

a. Écrire une fonction `est_non_orienteM`, prenant en argument la matrice d'adjacence d'un graphe et renvoyant un booléen indiquant si le graphe est orienté ou non. Évaluer sa complexité en fonction de l'ordre, n, ou de la taille, m, du graphe.

b. Écrire une fonction `est_non_orienteLA`, prenant en argument un dictionnaire des listes d'adjacence des sommets d'un graphe et renvoyant un booléen indiquant si le graphe est orienté ou non. Évaluer sa complexité en fonction de l'ordre, n, ou de la taille, m, du graphe.

Question 20. On définit ici la distance entre deux sommets comme la longueur du plus court chemin reliant ces deux sommets dans le graphe. On considèrera que la distance entre deux sommets du graphe est infinie s'il n'existe aucun chemin entre ces deux sommets.

a. Décrire un algorithme permettant déterminer tous les sommets à la distance k d'un sommet donné.

b. Évaluer la complexité de cet algorithme, en fonction de l'ordre, n, ou de la taille, m, du graphe.

c. Proposer une implémentation de cet algorithme sous la forme d'une fonction, prenant en argument, soit une matrice d'adjacence, soit un dictionnaire de liste d'adjacence décrivant le graphe, un numéro de sommet s (sommet-source) et un entier k, et renvoyant la liste des sommets à distance k de s.

Question 21. On dit qu'un graphe présente un circuit s'il existe un chemin reliant un sommet à lui-même.

a. Décrire un algorithme permettant de vérifier qu'un graphe est sans circuit. Le graphe peut alors être qualifié de « DAG » (*Directed Acyclic Graph*).

b. Évaluer la complexité de cet algorithme, en fonction de l'ordre, n, ou de la taille, m, du graphe.

c. Proposer une implémentation de cet algorithme sous la forme d'une fonction, prenant en argument, soit une matrice d'adjacence, soit un dictionnaire de liste d'adjacence décrivant le graphe, et renvoyant un booléen indiquant que le graphe est ou non sans circuit.

d. Proposer une modification de la fonction pour qu'elle renvoie un circuit présent dans le graphe, sous la forme d'une liste des sommets de ce circuit, ou une liste vide s'il n'en existe pas.