L3 Informatique des Organisations

Algorithmes dans les graphes

Exercices de TD et TP

2024-2025

Virginie Gabrel (bureau P630)
email : gabrel@lamsade.dauphine.fr

TD1 : Applications et Modélisations par les graphes

Exercice 1

Comment faire pour aller le plus rapidement possible de Bordeaux à Grenoble sachant que :

Bordeaux → Nantes	4h
Bordeaux → Marseille	9h
Bordeaux → Lyon	12h
Nantes → Paris-Montparnasse	2h
Nantes → Lyon	7h
Paris-Montparnasse → Paris-Lyon	1h
Paris-Lyon → Grenoble	4h30
Marseille → Lyon	2h30
Marseille → Grenoble	4h30
Lyon → Grenoble	1h15

Proposer une modélisation à l'aide d'un graphe et formuler le problème à résoudre dans ce graphe.

Exercice 2

8 groupes d'étudiants (numérotés de G1 à G8) doivent passer des examens dans différentes disciplines, chaque examen occupant une demi-journée :

chimie	G1 et G2
Electronique	G3 et G4
Informatique	G3, G5, G6 et G7
Mathématiques	G1, G5, G6 et G8
Physique	G2, G6, G7 et G8

On cherche à organiser la session d'examen la plus courte possible. Proposer une modélisation à l'aide d'un graphe et formuler le problème à résoudre dans ce graphe.

Exercice 3

Septs amis, Rachel, Monica, Ross, Chandler, Janice, Joe, et Phoebe habitent New York et projettent de louer deux voitures pour aller passer des vacances en Floride. Toutefois, passer ensemble les 18 heures que dure le trajet peut s'avérer problématique si les passagers des deux véhicules ne sont pas soigneusement choisis.

En effet, il faut savoir que :

— Rachel et Ross viennent de rompre, il ne serait donc pas judicieux qu'ils voyagent ensemble,

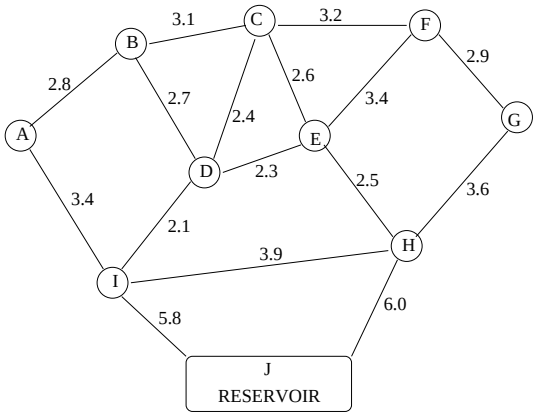
— Joe tente de séduire Rachel, il y a donc des tensions entre lui et Ross,

— Phoebe semble tenter d'attirer l'attention de Joe, ce qui a jeté un froid entre elle et Rachel,
— Chandler vient juste de piquer à Joe la place de capitaine de l'équipe de foot, il en résulte un certain ressentiment,
— Monica et Chandler sont jaloux de Janice, parce qu'elle attend un bébé et qu'ils ne réussissent pas à en faire un,
— Monica pense que Joe est un crétin.

Formulez ce problème en termes de graphe et donnez une solution pour ce cas particulier.

Exercice 4

Une communauté de commune veut desservir un ensemble de villages (indicés de A à J dans la figure ci-dessous) en eau potable à partir d'un château d'eau placé en J. Certains villages ne peuvent pas être reliés à d'autres à cause du relief. Le coût d'installation des canalisations pour transporter l'eau est estimé pour certaines paires de villages. Une arête entre deux sommets i et j dans la figure ci-dessous représente la possibilité d'installer une canalisation entre i et j et sa valeur représente le coût d'installation. On suppose que les capacités des canalisations sont illimitées. Il faut déterminer un réseau qui puisse acheminer de l'eau dans chaque village, et ce au moindre coût.



1. Formuler le problème en utilisant des concepts issus de la théorie des graphes.
2. Pouvez-vous proposer une solution de coût minimal ?

Exercice 5 : Composition d'équipes

En 1941, les escadrilles anglaises se composaient d'avions biplaces, mais certains pilotes ne pouvaient pas faire équipe dans le même avion par suite de différence de langues et de capacités. Sous ces contraintes, on cherche à déterminer le nombre maximum d'avions pouvant voler simultanément.
Considérer l'exemple numérique suivant où 8 pilotes sont disponibles pour faire voler 4 avions. Dans le tableaux ci-dessous, une croix ligne i colonne j signifie que les pilotes i et j ne peuvent pas faire équipe :

	1	2	3	4	5	6	7	8
1			×		×	×	×	
2							×	×
3	×			×	×	×	×	×
4			×			×		
5	×		×					×
6	×		×	×			×	×
7	×	×	×			×		×
8		×	×		×	×	×	

1. Modéliser ce problème sous la forme d'un graphe.
2. Pour déterminer le nombre maximum d'avions pouvant voler simutanément, quel problème doit-on résoudre ?
3. Est-il possible de faire voler les 4 avions de la flotte en même temps ? Si non, combien d'avions peuvent voler simultanément ?

TD2 : Propriétés élémentaires

Exercice 1

1. Combien d'arêtes au maximum peut comporter un graphe simple à n sommets ?
2. Combien d'arcs au maximum peut comporter un 1-graphe à n sommets ?
3. Combien existe-t-il de graphes simples différents à 4 sommets ? À n sommets ?
4. Prouver que dans tout graphe simple le nombre de sommets de degré impair est pair.

Exercice 2

1. Montrer qu'il n'existe pas de graphe simple (donc non orienté) $G = (X, U)$ d'ordre 6 avec des sommets de degré 2, 3, 3, 4, 4, 5.
2. Montrer qu'il n'existe pas de graphe simple $G = (X, U)$ d'ordre 5 avec des sommets de degré 2, 3, 4, 4, 5.
3. Montrer qu'il n'existe pas de graphe simple $G = (X, U)$ d'ordre 6 avec des sommets de degré 2, 2, 5, 5, 5, 5.
4. Un graphe simple r-régulier est un graphe $G = (X, U)$ dont tous les sommets ont le même degré r. Considérant un graphe r-régulier $G = (X, U)$ d'ordre n, calculer le nombre d'arêtes m en fonction de n et de r.

Exercice 3

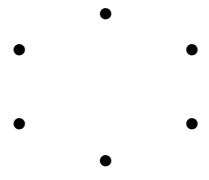
On considère un graphe simple connexe. Montrer qu'il existe toujours deux sommets ayant le même degré.

Exercice 4

Démontrer (par l'absurde) qu'une condition nécessaire pour qu'un graphe soit sans circuit est qu'il comporte au moins un sommet de demi-degré intérieur nul.

Exercice 5

On joue au jeu suivant : on donne à deux joueurs deux stylos de couleurs différentes. Puis chacun à leur tour les joueurs tracent sur la figure ci-dessous une arête reliant deux sommets entre eux.



Le gagnant est celui qui le premier aura tracé un triangle avec sa couleur. Montrer qu'il n'y a jamais de partie nulle.

Exercice 6

Montrer que si G est un graphe simple d'ordre n avec p composantes connexes, le nombre maximum d'arêtes dans G est :

$$\frac{1}{2}(n-p)(n-p+1).$$

Exercice 7

Soit $G = (X, E)$ un graphe simple. Démontrer que :

1. si G est connexe, alors $\mid E \mid \geq \mid X \mid -1$,
2. si G est sans cycle, alors $\mid E \mid \leq \mid X \mid -1$.

TD3 : Représentation de graphes

Exercice 1

Soit la matrice d'incidence sommets-arcs représentant le graphe G :

	u_1	u_2	u_3	u_4	u_5	u_6
a	$+1$		-1	$+1$		
b		-1				
c	-1				$+1$	$+1$
d			$+1$	-1	-1	
e		$+1$				-1

1. Quel est le demi-degré intérieur de chacun des sommets ?
2. Quel est le demi-degré extérieur de chacun des sommets ?
3. Quel est le degré de chacun des sommets ?
4. Quelle est la matrice d'adjacence $\mathcal{A}$ associée à G ?
5. Dessiner le graphe G.
6. G est-il connexe ? fortement connexe ?

Exercice 2

Donner la matrice d'adjacence du graphe non orienté de l'exercice 2 du TD 1.

Exercice 3

On considère la matrice d'adjacence d'un graphe simple d'ordre n.

1. Donner un algorithme permettant de déterminer si deux sommets donnés i, j ont au moins un voisin commun (autre que i et j si i et j sont voisins).
2. Donner le temps de calcul de cet algorithme.
3. Si on adopte une représentation par liste d'adjacence, donner l'algorithme permettant d'effectuer ce même test.

Algorithme de parcours en profondeur d'abord

Version récursive

En pseudo-code

```
FONCTION explore(graphe, sommet)
    etat[sommet] <- 'exploré'
    pour tout j successeur de sommet dans graphe
        si etat[j] == 'inexploré' alors
            explore(graphe, j)

ALGORITHME DFS
    lire G, n
    pour i allant de 1 à n
        etat[i] <- 'inexploré'
    pour i allant de 1 à n
        si etat[i] == 'inexploré' alors
            explore(G, i)
```

Avec les numérotations suffixes et préfixes

```
FONCTION exploreNum(graphe, i, A, pref, suff, p, s)
    pref[i] <- p
    p <- p+1
   pour tout j successeur de i dans graphe
        si pref[j]==0 ALORS
            A <- A + (i,j)
            exploreNum(graphe, j, A, pref, suff, p, s)
    suff[i] <- s
    s <- s+1

ALGORITHME DFS_RECURSIF_PREF_SUFF
    lire G,n
    pour i allant de 1 à n
       pref[i] <- 0
       suff[i] <- 0
    A <- ensemble vide
    p <- 1
    s <- 1
    POUR i allant de 1 à n
        SI pref[i]==0 ALORS
            exploreNum(G, i, A, pref, suff, p, s)
```

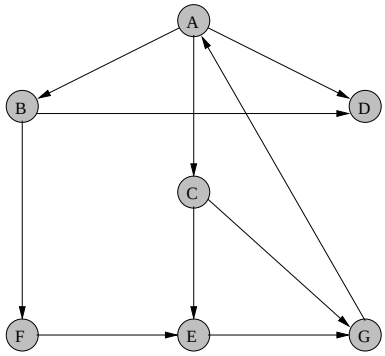
Version itérative (avec une Pile)

```
ALGORITHME DFS_itératif
lire G, n
pour i allant de 1 à n
   pref[i] <- 0
   suff[i] <- 0
P <- PileVide()
A <- ensemble vide
p <- 1
s <- 1
Pour i allant de 1 à n
    Si pref[i]==0 Alors
        empiler(P,i)
        pref[i] <- p
        p <- p+1
        Tant que P non vide faire
            v <- sommetPile(P)
            Si existe un successeur j de v t.q pref[j]==0 alors
                empiler(P,j)
                pref[j] <- p
                p <- p+1
                A <- A + (v,j)
            Sinon
                i <- depiler(P)
                suff[i] <- s
                s <- s+1
```

TD4 : Parcours de graphes

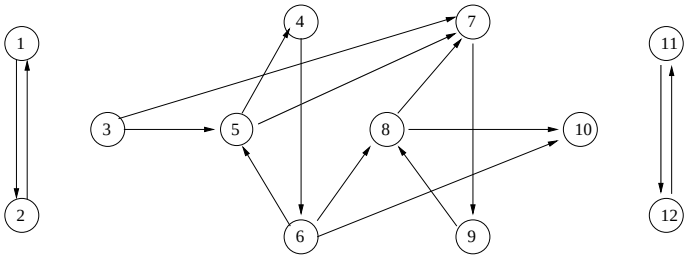
Exercice 1

Décrire le parcours en profondeur d'abord du graphe G de la figure ci-dessous à partir du sommet A, c'est-à-dire donner l'arborescence couvrante, les arcs avant, arrière, croisé. Donner la numérotation préfixe et suffixe de chacun des sommets.



Exercice 2

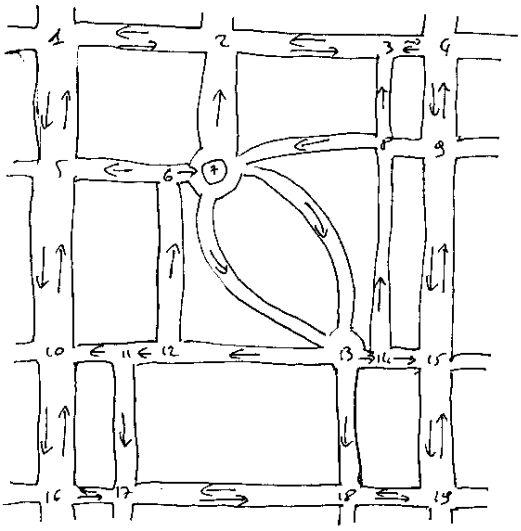
1. Faire un parcours en profondeur d'abord (DFS) à partir du sommet 1 dans le 1-graphe représenté ci-dessous.



2. Lister les arcs de types avant, arrière et croisé.
3. Donner les numérotations préfixe et suffixe de chacun des sommets.
4. Déterminer les composantes fortement connexes avec l'algorithme de Kosaraju-Sharir.
5. Donner le graphe réduit associé.
6. Déterminer les composantes connexes du graphe ci-dessus.

Exercice 3

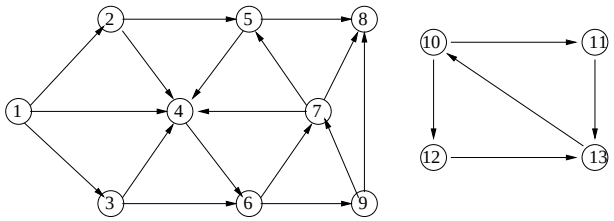
Un ingénieur du service technique d'une ville propose, pour la zone centrale, le plan de sens uniques représenté dans la figure ci-dessous.



Avant d'adopter ce plan, il convient d'examiner s'il ne rend pas certains points inaccessibles.

1. Proposez un graphe orienté $G = (X, U)$ représentant ce plan.
2. Quelle propriété ce graphe devra-t-il vérifier pour que le plan de circulation soit acceptable ?
3. Tracer le graphe réduit associé au graphe de la question 2.
4. Le plan de circulation est-il acceptable ?

Exercice 4



1. Faire un parcours en profondeur d'abord à partir du sommet 1 (en choisissant prioritairement le successeur de plus petit numéro) du graphe G=(X,U) représenté dans la figure ci-dessus.
2. G contient-il des circuits (justifier votre réponse) ?
3. Modifier l'orientation de deux arcs afin de rendre le graphe G sans circuit.
4. Déterminer l'ordre topologique de ce nouveau graphe

Exercice 5

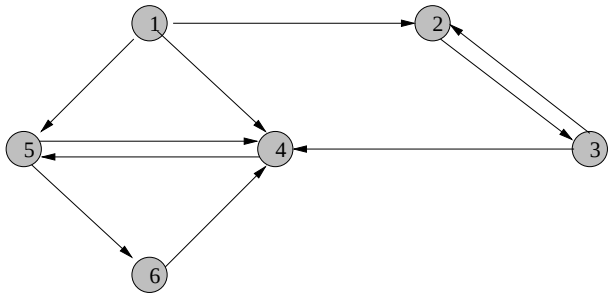
La mise en oeuvre d'une série de 13 programmes $P_1, P_2, ..., P_{13}$ sur un ordinateur doit être planifiée. Il existe des couples (P_i, P_j) de programmes qui subissent des contraintes d'antériorité : P_i doit être planifié avant P_j car P_j utilise des données calculées par P_i. Ces couples sont :
(P_1, P_6) (P_2, P_1) (P_3, P_8) (P_4, P_1) (P_4, P_2) (P_5, P_{10}) (P_6, P_2) (P_6, P_{12}) (P_6, P_{13}) (P_8, P_{10}) (P_8, P_{12}) (P_9, P_1) (P_9, P_4) (P_{10}, P_7) (P_{11}, P_4) (P_{12}, P_3) (P_{12}, P_5) (P_{13}, P_3)

L'informaticien en charge de la planification de ces programmes doit s'assurer que cette mise en oeuvre est possible et doit définir un ordre d'exécution des programmes.

1. Pour tester si l'exécution des 13 programmes est possible, l'informaticien s'appuie sur une modélisation du problème sous la forme d'un graphe. Définir le graphe associé. Formuler le problème de l'informaticien en terme de problème classique de graphe. Comment localiser un ensemble de programmes qui posent un problème ?
2. Représenter le graphe associé et appliquer un algorithme général pour effectuer le test. Existe-t-il un ensemble de programmes qui posent un problème ?
3. Pour définir un ordre d'exécution des programmes, quel problème classique de graphe doit-on résoudre ? Justifier votre réponse.

Exercice 6

Déterminer la fermeture transitive du graphe ci-dessous.



Evaluer la complexité en temps dans le pire cas de cet algorithme.

Exercice 7. Graphe d'intervalle

On considère un ensemble de n intervalles. Pour $i = \{1, \ldots, n\}$, l'intervalle i est défini par $[a_i, b_i]$ avec $a_i < b_i$. Deux intervalles $[a_i, b_i]$ et $[a_j, b_j]$ sont dits **compatibles** si et seulement si :

$$[a_i, b_i] \cap [a_j, b_j] = \emptyset$$

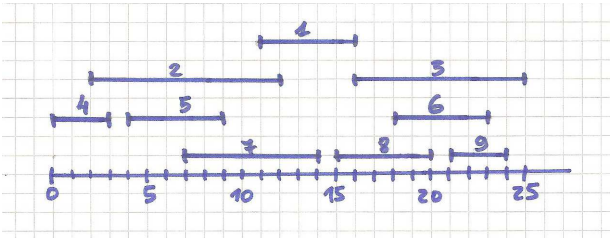
et ils sont dits **incompatibles** sinon.

Le problème à résoudre est celui de déterminer un sous-ensemble d'intervalles tous compatibles 2 à 2 de cardinalité maximale.

Par la suite, on considèrera d'application numérique suivante avec $n = 9$:

i	1	2	3	4	5	6	7	8	9
$[a_i, b_i]$	$[11, 16]$	$[2, 12]$	$[16, 25]$	$[0, 3]$	$[4, 9]$	$[18, 23]$	$[7, 14]$	$[15, 20]$	$[21, 24]$

Il est aussi très utile de représenter cette application numérique comme suit :



1. Considérant l'application numérique ci-dessus, représenter le problème sous la forme d'un graphe non orienté $G = (X, E)$ dont les sommets représentent les intervalles et les arêtes la relation binaire d'incompatibilité.
2. Formuler le problème à résoudre sur G (justifier votre réponse)
3. Proposer une solution au problème décrit dans l'application numérique (non nécessairement optimale).
4. On définit maintenant un graphe orienté $G^{\mathcal{P}} = (X, U)$ dont les sommets représentent les intervalles et les arcs la relation binaire de précédence, notée $\mathcal{P}$, suivante : $[a_i, b_i]\mathcal{P}[a_j, b_j]$ si et seulement si $b_i < a_j$. Montrer que $G^{\mathcal{P}}$ est un graphe transitif c'est-à-dire que, pour tout triplet de sommets (i, j, k), $(i, j) \in U$ et $(j, k) \in U$ implique $(i, k) \in U$.
5. Montrer que si on renumérote les sommets i de $G^{\mathcal{P}}$ par ordre croissant des a_i, on définit un ordre topologique. Quel est alors la complexité dans le pire des cas pour définir un ordre topologique dans $G^{\mathcal{P}}$? En terme de complexité dans le pire des cas, serait-il préférable d'utiliser l'algorithme vu en cours pour déterminer un ordre topologique (vous prendrez soin de bien justifier votre réponse) ?
6. A partir de $G^{\mathcal{P}} = (X, U)$, on définit un second graphe orienté, noté $G^{\mathcal{P}}_R = (X_R, U_R)$, comme suit :
   — $X_R = X \cup \{e, s\}$ avec e et s deux sommets fictifs,
   — U_R contient les arcs (e, i) pour tout $i \in X$ tel que $d^-(i) = 0$ et, les arcs (i, s) pour tout $i \in X$ tel que $d^+(i) = 0$; U_R contient également tous les arcs de $U \setminus T$ avec

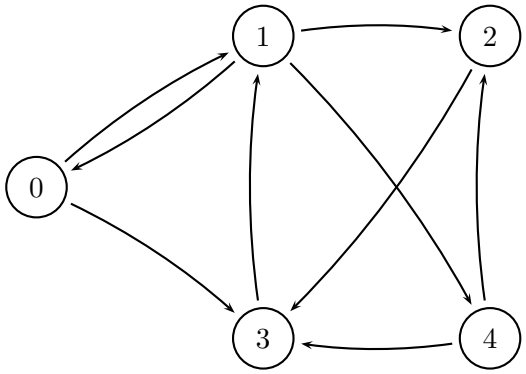
   $$T = \{(i, k) \in U : \exists j \in X \setminus \{i, k\} \text{ avec } (i, j) \in U \text{ et } (j, k) \in U\}$$

   T est appelé l'ensemble des arcs de transitivité.
   Dessiner $G^{\mathcal{P}}_R = (X_R, U_R)$ associé à l'application numérique.
7. A quoi correspond dans $G^{\mathcal{P}}_R$ la solution trouvée question 3 ?
8. Expliciter les étapes permettant de construire G à partir de $G^{\mathcal{P}}_R$.
9. Formuler le problème à résoudre sur $G^{\mathcal{P}}_R$.

TP 1 : modules numpy, networkx et matplotlib en Python

Exercice 1 : Exploration en profondeur d'abord d'un graphe orienté en Python

Soit le graphe orienté $G = (X, U)$ représenté ci-dessous :



1. Ecrire en Python une fonction `creeMatAdj()` qui prend en arguments l'ordre d'un graphe et la liste de ses arcs et qui renvoie la matrice d'adjacence (sous la forme d'une liste de listes).
2. Ecrire en Python une fonction `creeListeSucc()` qui prend en arguments l'ordre d'un graphe et la liste de ses arcs et qui renvoie une liste de listes de successeurs.
3. Ecrire en Python une fonction `degreExt()` qui prend le nom d'un sommet en argument et renvoie son demi-degré extérieur.
4. Traduire en Python la procédure `explore()` présentée en cours alors que le graphe G est représenté sous la forme d'une liste de listes de successeurs.
5. Traduire en Python l'algorithme DFS présenté en cours et afficher l'ensemble A ainsi que le numéros préfixes et suffixes de chacun des sommets.
6. Terminer votre programme en affichant les listes des arcs avants, arrières et croisés.

Exercice 2 : Utilisation du module `numpy`

Brève présentation du module `numpy`

Dédiée au calcul scientifique, le module `numpy` fournit un type tableau multidimensionnel et des fonctions très performantes pour effectuer des calculs sur ce type de données. Pour importer le module `numpy`, on exécute l'instruction `import numpy as np`.

Le type `ndarray` est le type `numpy` qui correspond à un tableau à n dimensions pour stocker des valeurs de même type. On crée une valeur de type `ndarray` avec la fonction `array` comme par exemple :

— `t1 = np.array([2, 4, 6])`

— `t2 = np.array([[2, 4, 6], [1, 2, 3]], dtype = float)`

Il existe des fonctions `numpy` pour initialiser un tableau :

— `np.zeros((3,4), dtype = int)` renvoie un tableau à 2 dimensions (3 lignes et 4 colonnes) ne contenant que des 0
— `np.ones(10)` renvoie un tableau unidimensionnel de 10 éléments, tous égaux à 1.0
— `np.full((5,3), 9)` renvoie un tableau à deux dimensions (5 lignes et 3 colonnes) dont tous les éléments valent 9.0
— `np.arange(10, 25, 5)` renvoie un tableau contenant des valeurs allant de 10 à 25 (exclue) par pas de 5 (équivalent à `range` mais renvoie un `ndarray`)
— `np.linspace(10, 25, 5)` renvoie un tableau contenant 5 valeurs équitablement répartie dans l'intervalle $[10, 25]$
— `np.eye(5, dtype = int)` renvoie une matrice identité 5×5
— `np.random.random((10,2)` renvoie un tableau de 10 lignes et 2 colonnes dont les éléments sont des nombres aléatoires compris entre 0 et 1

Les méthodes ci-dessous permettent d'afficher des informations sur `t` une variable référençant une valeur de type `ndarray`

— `t.shape` renvoie les dimensions de t
— `len(t)` renvoie la longueur du tableau (nbre de lignes)
— `t.ndim` renvoie le nombre de dimensions de t
— `t.size` renvoie le nombre d'éléments dans t
— `t.dtype` renvoie le type des éléments
— `t.astype(type)` renvoie le tableau dans lequel les éléments de t sont convertit dans le type donné en argument

L'accès aux éléments d'un tableau est similaire à celui d'une liste. Concernant l'opérateuer d'extraction appliqué à un tableau à deux dimensions, il est possible de :

— Extraire des colonnes

```
t[:,[0,3]]   => renvoie les colonnes 0 et 3
t[:,0:3]     => renvoie les colonnes indicées
                de 0 à 2
```

— Extraire des lignes

```
t[[0,3],:]   => renvoie les lignes 0 et 3
t[0:3,:]     => renvoie les lignes indicées
                de 0 à 2
```

Il est également possible de filtrer les valeurs d'un tableau de la façon suivante :

```
>>> t = np.linspace(1,50,20).round(1)
>>> t[t>40]
array([42.3, 44.8, 47.4, 50. ])
```

Avec `numpy`, les calculs sur les tableaux sont très performants :

— `arr1 * 10` multiplie tous les éléments de arr1 par 10
— `arr1 + arr2` fait une addition terme à terme
— `arr1 * arr2` fait une multiplication terme à terme

Il existe également des fonctions universelles : `max`, `min`, `sum`, `round`, `floor`... disponibles sous la forme de fonction `numpy` ou de méthode de la classe `ndarray`.

Concernant le calcul matriciel, il existe les fonctions suivantes :

— `dot(a,b)` : produit matriciel entre `a` et `b`
— `transpose(a)` : renvoie la matrice transposée de `a`

— `linalg.matrix_power(a,n)` qui calcule la puissance nième de la matrice `a`,
— `linalg.inv(a)` qui calcule l'inverse de la matrice carrée `a`,
— `linalg.det(a)` qui calcule le déterminant de la matrice carrée `a`,
— `linalg.solve(a,b)` qui résoud le système $ax = b$,
— `linalg.norm(x,opt)` qui calcule la norme du vecteur `x` : norme euclidienne par défaut, norme 1 si `opt=1`, norme infinie si `opt=np.inf`,
— `linalg.eigvals(a)` renvoie le vecteur des valeurs propres de `a`.

Tester la connexité forte avec `numpy`

Soit $\mathcal{A}$ la matrice d'adjacence d'un graphe G d'ordre n.

1. Définissons $\mathcal{A}^k = \mathcal{A} * \mathcal{A}^{k-1}$ pour $k = 2, 3, \ldots$.
   (a) Montrer que l'élément a_{ij}^2 de la matrice $\mathcal{A}^2$ correspond au nombre de chemins allant de i à j de longueur 2. En déduire que l'élément a_{ij}^k de la matrice $\mathcal{A}^k$ correspond au nombre de chemins allant de i à j de longueur k.
   (b) Montrer que G est fortement connexe si et seulement si la matrice R définie par $R = \mathcal{A} + \mathcal{A}^2 + \mathcal{A}^3 + \cdots + \mathcal{A}^n$ ne possède aucune entrée nulle.
2. Ecrire en Python une fonction qui prend en paramètre k et A et renvoie A^k. Faire appel au module numpy (`https://numpy.org/`) pour effectuer les produits de matrice. Appeler cette fonction pour tester s'il existe un chemin de longueur 4 entre le sommet 1 et 2 dans le graphe de l'exercice 1.
3. On a démontré précédemment que G est fortement connexe si et seulement si la matrice R définie par $R = A + A^2 + A^3 + \cdots + A^n$ ne possède aucune entrée nulle. Ecrire en Python une fonction qui effectue ce test.
4. Appeler cette fonction sur le graphe de l'exercice 1 pour savoir si ce graphe est fortement connexe.

Exercice 3 : Utilisation du module `networkx`

Brève présentation du module `networkx`

Le module `networkx` est très complet (cf. `https://networkx.org/` avec une documentation complète disponible) : il permet de créer, représenter des graphes (orienté ou non, simple ou multiple, valué ou non) et d'appliquer des algorithmes classiques. Pour l'utiliser, il suffit d'exécuter : `import networkx` ou, `import networkx as nx` si on souhaite utiliser l'alias `nx`. Un graphe G est représenté par sa liste de voisins (ou de successeurs si le graphe est orienté), implémentée sous la forme d'un dictionnaire de dictionnaires. Pour déclarer une variable `g` de type graphe non orienté et, une variable `go` de type graphe orienté, on exécute les instructions :

```
g=nx.Graph()
go=nx.DiGraph()
```

`g` (resp. `go`) est alors un dictionnaire dont chaque clé représente un sommet i de G et, chaque valeur associée à i est un dictionnaire représentant la liste des voisins (resp. successeurs), avec comme clé le sommet j, voisin (resp. successeur) de i dans G, et comme valeur un dictionnaire représentant entre autre la longueur de l'arc (i, j) si elle existe (plus d'autres paramètres éventuels de l'arc). Pour initialiser une variable `g` ou `go` de type Graph ou DiGraph, on a plusieurs possibilités et méthodes à notre disposition :
— ajouter un sommet de nom `'s'` : `g.add_node('s')`
— ajouter une arête (i, j) : `g.add_edge(i,j)`

— ajouter une liste de sommets : `g.add_nodes_from(lst)` où `lst` est de type `list`
— ajouter une liste `lst` d'arêtes (ou d'arcs valués ou non) : `g.add_edges_from(lst)` où `lst` est de type `list` de tuples `(i,j[,val])` avec `val` la valeur de l'arête (ou l'arc) (i, j).

Suite à l'initialisation : `for n in g` permet de parcourir la liste des sommets, `for d in G[i]` permet de parcourir la liste des voisins du sommet i, `g.adj` retourne le dictionnaire de dictionnaires `g`, `g.nodes()` retourne la liste des sommets, `g.edges()` retourne la liste des arêtes.

Pour obtenir une représentation graphique de G à l'aide de `matplotlib.pyplot`, il suffit alors d'exécuter les instructions suivantes :

```
nx.draw(g)
plt.show()
```

Tester la connexité et la connexité forte avec `networkx`

1. Stocker le graphe de l'exercice 1 sous la forme d'une variable de type `DiGraph`.
2. Représenter graphiquement le graphe de l'exercice 1 avec `matplotlib`.
3. Après avoir appliqué `DFS`, représenter graphiquement le graphe de l'exercice 1 avec `matplotlib` en mettant en noir les arcs d'arbre, en bleu les arcs arrières, en rouge les arcs avants, et en vert les arcs croisés.
4. A l'aide du module `networkx`, tester si le graphe de l'exercice 1 est connexe. On trouve deux fonctions : `is_connected(G)` retourne vrai si `G` est connexe et faux sinon ; `number_connected_components(G)` retourne le nombre de composantes connexes de G.
5. Ecrire un programme qui génére aléatoirement un graphe orienté d'ordre `n` à l'aide de `networkx`, avec par exemple la fonction

   ```
   nx.gnp_random_graph(n, p, seed=None, directed=True)
   ```

6. A l'aide du programme de l'exercice 2, tester la connexité forte de graphes d'ordre 100, 200, 300 et 400 générés aléatoirement. Comparer les temps de calcul pour faire ce même test mais avec la fonction du module `networkx` `is_strongly_connected(G)`.

Exercice 4 : Résolution de problèmes avec `networkx`

Dans le TD1 de modélisation par les graphes, nous avons introduit et modélisé différents problèmes. En utilisant le module `networkx`, proposer une solution à ces différents problèmes (sauf le problème d'ordonnancement) sachant que :

— le choix d'un itinéraire est un 'shortest path problem',
— la détermination d'une session d'examen la plus courte possible est un 'coloring problem',
— la distribution d'eau potable est un 'minimum spanning tree',
— la composition d'équipes est un 'matching problem'.

Algorithme de parcours en largeur d'abord

Dans l'algorithme de parcours en largeur (ou BFS, pour Breadth First Search en anglais), l'ordre d'exploration des sommets est géré par une file.

```
FONCTION exploreLargeur(graphe, sommet, etat, F, A)
    etat[sommet] <- 'exploré'
    F.enfiler(sommet)
    tant que F non vide faire
        v <- F.defile()
        pour tout j successeur de v
            si etat[j]=='inexploré' alors
                etat[j] <- 'exploré'
                A <- A + (v,j)
                F.enfiler(j)

Algorithme BFS
    lire G, n
    A <- ensemble vide
    F <- FileVide()
    pour i allant de 1 à n
        etat[i] <- 'inexploré'
    pour i allant de 1 à n
        si etat[i] == 'inexploré' alors
            exploreLargeur(G, i, etat, F, A)
```

Traduction en Python

```
from collections import deque
def exploreLargeur(graphe, sommet, etat, F, A):
    etat[sommet] = 'exploré'
    F.append(sommet)
    while len(F) > 0:
        v = F.popleft()
        for j in G[v]:
            if etat[j]=='inexploré':
                etat[j] = 'exploré'
                A.append((v,j))
                F.append(j)
#G est représenté par une listes de listes de successeurs
etat = ['inexploré' for _ in range(n)]
F = deque([]), A = []
for i in range(n):
    if etat[i] == 'inexploré':
        exploreLargeur(G, i, etat, F, A)
```

Algorithme de Dijkstra

$\lambda(i_0) \leftarrow 0$
  $\lambda(i) \leftarrow +\infty \; \forall i \in X \setminus \{i_0\}$
  $p(i) \leftarrow i \; \forall i \in X$
  $S \leftarrow \{i_0\}$
  $i \leftarrow i_0$
  TantQue $S \neq X$ Faire
    Pour tout $j \in \Gamma^+(i) \setminus S$ Faire
      Si $\lambda(j) > \lambda(i) + l_{ij}$ alors
        $\lambda(j) \leftarrow \lambda(i) + l_{ij}$
        $p(j) \leftarrow i$
      FinSi
    FinPour
    Sélectionner $i \in X \setminus S$ tel que $\lambda(i) = \min_{j \in X \setminus S} \lambda(j)$
    $S \leftarrow S \cup \{i\}$
  FinTantQue

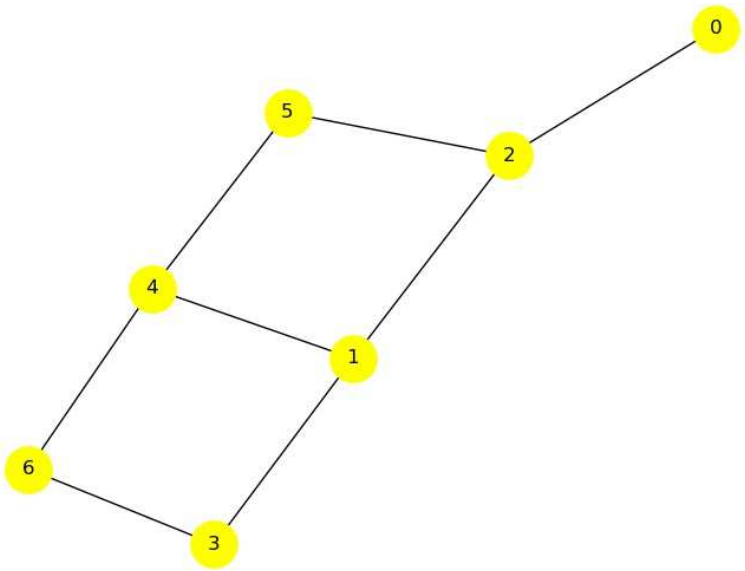
Algorithme de Dijkstra avec une structure de tas

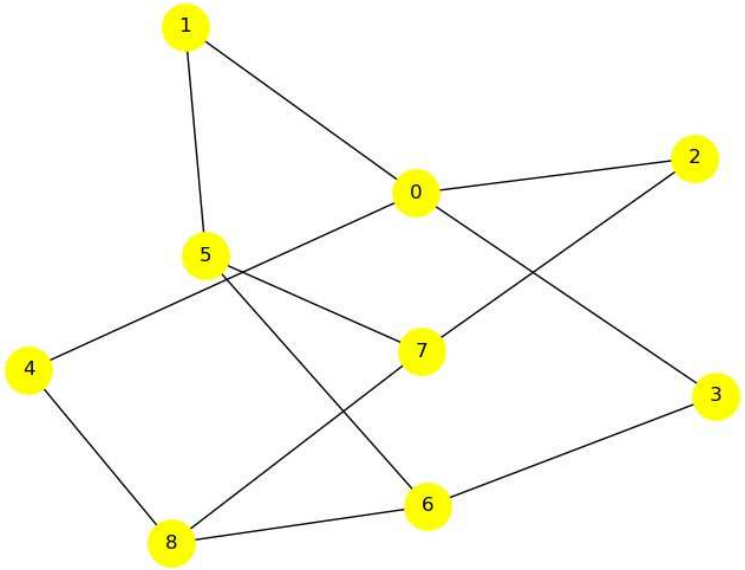
$\lambda(i_0) \leftarrow 0$
  $\lambda(i) \leftarrow +\infty \; \forall i \in X \setminus \{i_0\}$
  $p(i) \leftarrow i \; \forall i \in X$
  $S \leftarrow \{i_0\}$
  $i \leftarrow i_0$
  TantQue $S \neq X$ Faire
    Pour tout $j \in \Gamma^+(i) \setminus S$ Faire
      Si $\lambda(j) > \lambda(i) + l_{ij}$ alors
        $\lambda(j) \leftarrow \lambda(i) + l_{ij}$
        $p(j) \leftarrow i$
      FinSi
    FinPour
    Sélectionner $i \in X \setminus S$ tel que $\lambda(i) = \min_{j \in X \setminus S} \lambda(j)$
    $S \leftarrow S \cup \{i\}$
  FinTantQue

TD5 : Recherche de plus courts chemins avec le parcours en largeur d'abord et l'algorithme de Dijkstra

Exercice 1

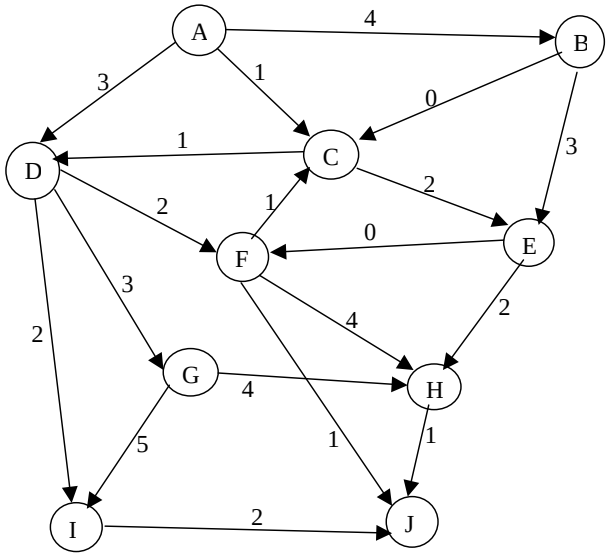
En appliquant l'algorithme de parcours en largeur d'abord, tester si les deux graphes ci-dessous sont bipartis ou non.



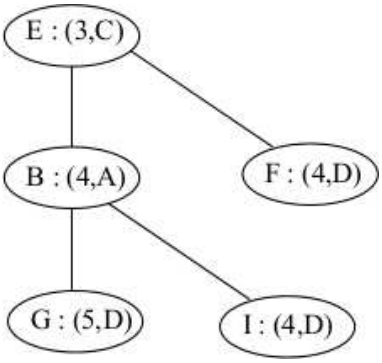


Exercice 2

Soit le graphe ci-dessous :



1. Appliquer l'algorithme de Dijkstra pour déterminer le plus court chemin allant de A à J.
2. On utilise la version de Dijkstra avec une structure de Tas pour déterminer le plus court chemin allant de A à J. A une itération de l'algorithme, alors que l'on a marqué définitivement les sommets A, C et D au cours des itérations précédentes, le minimier est :

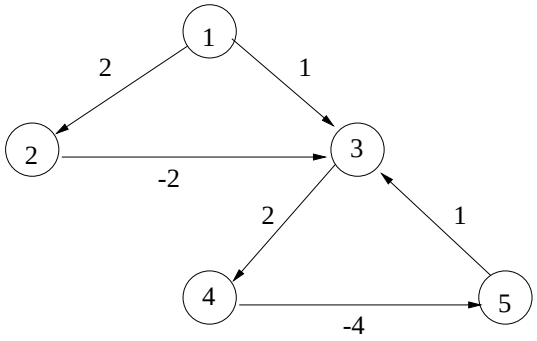


   (a) Décrivez en détail l'itération suivante de l'algorithme de Dijkstra avec Tas : quel sommet marque-t-on définitivement ? Comment le minimier va-t-il être modifié ?

   (b) Donner la complexité théorique dans le pire cas de l'algorithme de Dijkstra en utilisant une structure de Tas.

Exercice 3

Appliquer l'algorithme de Dijkstra pour trouver les distances minimales des chemins allant du sommet 1 à tous les autres sommets du graphe représenté ci-dessous.


1
2
1
2
3
-2
2
1
4
5
-4

TD6 : Recherche de plus courts chemins dans les graphes sans circuit avec l'algorithme de Bellman

Exercice 1 (Un problème de stocks)

L'entreprise bretonne QuandLaBriseFutVenue fabrique des voiliers. Pendant le premier trimestre 2024, elle doit livrer deux voiliers par mois : en janvier, février et mars, les deux voiliers doivent être livrés à la fin de chaque mois. Le fabricant souhaite établir le plan de production de ces voiliers.

Le stock ne peut dépasser deux unités en février et mars et, est nul en janvier et en avril. La production maximale pour un mois donné est de quatre unités.

Pour un stock de j voiliers et une production i, le coût mensuel vaut

$$C(i, j) = f(i) + 6j$$

avec

$$f(0) = 0, \qquad f(1) = 15, \qquad f(2) = 17, \qquad f(3) = 19, \qquad f(4) = 21.$$

Modéliser le problème du fabricant (qui souhaite définir un plan de production en janvier, février et mars de façon à satisfaire la demande tout en minimisant ses coûts) sous la forme de la recherche d'un plus court chemin dans un graphe orienté sans circuit. Déterminer la solution optimale à l'aide de l'algorithme de Bellman.

Exercice 2 : Gestion de projet et ordonnancement de tâches

On considère un projet de rénovation de maison décomposé en 10 tâches indicées de A à J. Dans le tableau ci-dessous sont présentées les durées des tâches (en semaines) et les contraintes d'antériorité entre les tâches. Par exemple, pour pouvoir commencer les travaux d'électricité, il faut nécessairement avoir terminé le gros oeuvre.

Codes	Tâches	Durées	Tâches antérieures
A	Etude	5	-
B	Gros oeuvre	8	A
C	Electricité	3	B
D	Plomberie	4	B
E	Chauffage	2	C
F	Isolation	2	D,E
G	Plafonnage	4	F
H	Carrelage	6	F
I	Menuiserie	5	G
J	Finition	6	I, H

On veut déterminer la durée minimale d'exécution du projet. Proposer une modélisation à l'aide d'un graphe et formuler le problème à résoudre dans ce graphe. Résoudre ce problème en utilisant l'algorithme de Bellman

Exercice 3

La veille de partir en randonnée se pose le problème de remplir son sac-à-dos de façon à marcher avec un sac de poids inférieur ou égal à une certaine valeur v (imposée par vos capacités physiques). Parmi n objets possibles indicés de 1 à n, vous devez donc choisir les objets à emporter connaissant leur poids mais surtout leur utilité. Pour cela, vous attribuez une note, comprise entre 0 et 20, à chacun des objets : plus la note est élevée, plus l'objet est utile. Considérons que $v = 5$ et que vous devez choisir parmi les 5 objets suivants :

Objet i	1	2	3	4	5
Poids p_i	2	2	5	4	2
Utilité u_i	5	6	12	8	7

TABLE 1 – Poids et utilité des objets

Votre problème est donc de sélectionner les objets à emporter de façon à maximiser la somme de leur utilité et à constituer un sac-à-dos de poids total inférieur ou égal à v.

1. On peut représenter ce problème sous la forme d'un graphe orienté valué défini comme suit :
   — Les sommets représentent les états possibles (i, d) du sac-à-dos. $(0, 0)$ est un sommet racine qui représente le sac-à-dos vide. $(n + 1, v)$ est un sommet anti-racine qui represente le sac-à-dos rempli des objets sélectionnés. Pour tout i allant de 1 à n, et pour tout d allant de 0 à v, il existe un sommet (i, d) si et seulement si il existe un ensemble d'objets $O \subseteq \{1, \ldots, i\}$ tel que $d = \sum_{i \in O} p_i \leq v$.
   — Pour tout i allant de 0 à $n - 1$, et pour tout sommet (i, d), il existe un arc allant du sommet (i, d) au sommet $(i + 1, d)$ de valeur 0 : cet arc représente le fait de ne pas ajouter l'objet $i + 1$.
   — Pour tout i allant de 0 à $n - 1$, et pour tout sommet (i, d), il existe un arc allant du sommet (i, d) au sommet $(i + 1, d')$ de valeur u_{i+1} si et seulement si

   $$d' = d + p_{i+1} \leq v$$

   Cet arc représente le fait d'ajouter l'objet $i + 1$.
   — Pour tout sommet (n, d), il existe un arc allant de (n, d) à $(n + 1, v)$ de valeur O.
   Représenter le graphe associé au problème décrit dans la table 1 (indication : on procède par niveau en commençant par $(0, 0)$ puis on ajoute les sommets de la forme $(1, d)$ en evisageant les deux cas : ajouter l'objet 1 ou pas, puis on passe aux sommets de la forme $(2, d)$...).
2. Peut-on borner supérieurement le nombre de sommets du graphe ?
3. Expliquer pourquoi ce graphe est nécessairement sans circuit.
4. Dans le cas général, que faut-il déterminer dans ce graphe pour trouver le sac-à-dos d'utilité maximale ?
5. Quel algorithme peut-on utiliser pour résoudre le problème ? Justifier votre réponse et donner la complexité de cet algorithme.

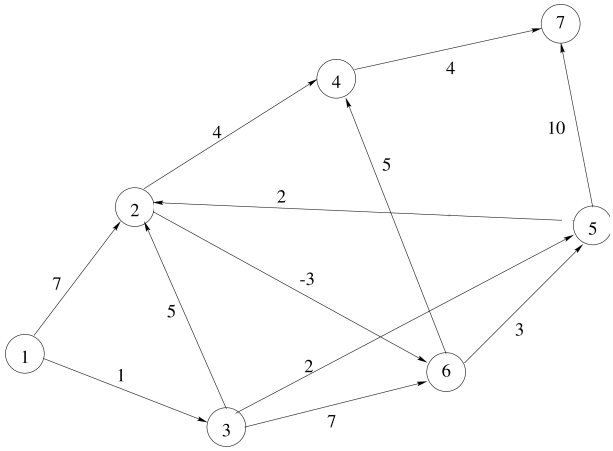
Algorithme général de Ford-Bellman

$k \leftarrow 0$
$\lambda^0(i_0) \leftarrow 0$
$\lambda^0(i) \leftarrow +\infty \; \forall i \in X \setminus \{i_0\}$
$p(i) \leftarrow i \; \forall i \in X \setminus \{i_0\}$
$M \leftarrow \{i_0\}$
TantQue $k \leq n$ et $M \neq \emptyset$ faire
    $k \leftarrow k + 1$
    $M' \leftarrow \emptyset$
    Pour tout $j \in \Gamma^+(M)$ faire
        $\lambda^k(j) \leftarrow \min\{\lambda^{k-1}(j); \lambda^{k-1}(i) + l_{ij},\ i \in \Gamma^-(j) \cap M\}$
        Si $\lambda^k(j) < \lambda^{k-1}(j)$ Alors
            $M' \leftarrow M' \cup \{j\}$
            $p(j) \leftarrow i^*$ avec $i^* \in \Gamma^-(j) \cap M$ tel que $\lambda^k(j) = \lambda^{k-1}(i^*) + l_{i^*j}$
        FinSi
    FinPour
    $M \leftarrow M'$
FinTantQue
Si $M \neq \emptyset$ Alors $\exists$ un circuit de valeur négative FinSi

TD7 : Recherche de plus courts chemins avec l'algorithme général de Ford-Bellman et avec l'algorithme de Floyd

Exercice 1

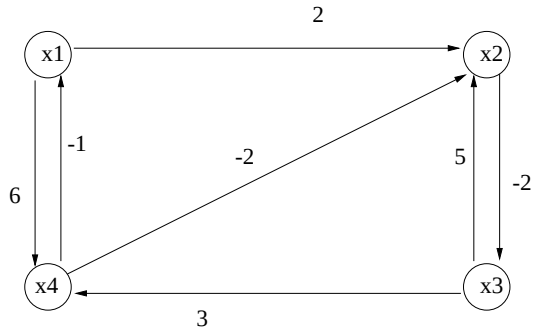
1. Déterminer les distances minimales à partir du sommet 1 sur le graphe suivant :



2. Comment obtenir par le même algorithme les plus courts chemins à partir du sommet 2 ?

Exercice 2

Appliquer l'algorithme de Floyd pour déterminer les valeurs des plus courts chemins entre tous les couples de sommets du graphe ci-dessous.



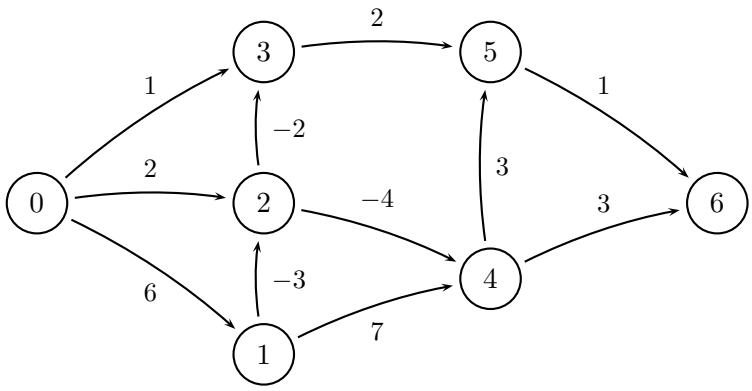
T.P. 2

Exercice 1 : Plus court chemins

1. Etant donné un graphe G d'ordre n sans circuit, dont on supposera que les sommets sont indicés de 0 à $n-1$ dans l'ordre topologique, écrire en Python une fonction `pcch` qui prend comme argument le graphe G. Cette fonction retourne la valeurs du plus court chemin allant de 0 à $n-1$. Cette fonction utilise l'algorithme de Bellman rappelé ci-dessous :

   $\lambda(0) \leftarrow 0$
   $p(0) \leftarrow 0$
   Pour $j \leftarrow 1$ à $n-1$
   $\lambda(j) = \min_{i \in \Gamma^-(j)} \lambda(i) + l_{ij}$
   $p(j) = \text{argmin}_{i \in \Gamma^-(j)} \lambda(i) + l_{ij}$
   FinPour

   Vous testerez votre fonction sur le graphe ci-dessous en affichant la valeur d'un plus court chemin allant de 0 à 6 :



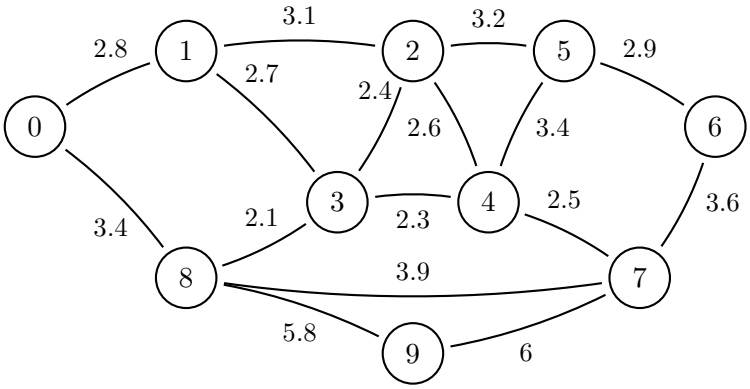
2. Considérer maintenant le graphe d'ordre 1000 décrit dans le fichier `bigDAG.txt` disponible sous moodle. Les sommets sont indicées de 0 à 999 dans l'ordre topologique. Comparer le temps de calcul nécessaire à votre fonction `pcch()` pour trouver la valeur d'un plus court chemin allant de 0 à 999 et celui nécessaire à la fonction `shortest_path_length` de `networkx` pour faire le même calcul.

Exercice 2 : Arbre couvrant

Soit $G = (X, E)$ le graphe simple valué d'ordre $n = 10$ ci-dessous. $\forall e = (i, j) \in E$, l_e ou l_{ij} est le poids de l'arête.

Un **arbre couvrant** $T = (X, E_T)$ est un graphe partiel de G connexe et sans cycle. En conséquence, $|E_T| = n - 1$. Le poids d'un arbre couvrant est la somme des longueurs des arêtes lui appartenant : $v(T) = \sum_{(i,j) \in E_T} l_{ij}$. On cherche à déterminer l'arbre couvrant de poids minimal, noté T^*.

Le principe de l'algorithme de Kruskal est de sélectionner les arêtes de plus petit poids de façon à former un arbre. Après avoir ordonné les arêtes par ordre de poids croissant, on construit pas à pas l'arbre



couvrant en sélectionnent $(n-1)$ arêtes parmi les arêtes de plus petit poids sans jamais créer de cycle.

Trier par ordre de poids croissant les arêtes : $E = \{e_1, e_2, \ldots, e_m\}$ avec $l_{e_1} \leq l_{e_2} \leq \ldots \leq l_{e_m}$.
$E_{T^*} \leftarrow \emptyset$
$v \leftarrow 0$, $j \leftarrow 0$, $cpt \leftarrow 0$
$c[i] \leftarrow i \; \forall i \in 1, \ldots, n$
TantQue $cpt < n-1$ et $j < m$ faire
    Considérer $e_j = (k, l)$
    Si $c[k] \neq c[l]$ alors
        $E_{T^*} \leftarrow E_{T^*} \cup \{e_j\}$
        $v \leftarrow v + l_{e_j}$
        $cpt \leftarrow cpt + 1$
        $tmp \leftarrow c[k]$
        Pour tout i allant de 1 à n faire
            Si $c[i] == tmp$ alors
                $c[i] \leftarrow c[l]$
            Fin Si
        Fin Pour
    Fin si
    $j \leftarrow j + 1$
FinTantQue

Le graphe est décrit comme suit dans le fichier arbreCouvrantTP2.txt, disponible sous moodle, contenant : ligne 1 : n, ligne 2 : m, sur chacune des m lignes suivantes : extrémités d'une arête et son poids.

1. Traduire l'algorithme de Kruskal en python et déterminer la solution optimale.
2. A l'aide de Networkx, effectuer une représentation graphique de la solution optimale.
3. A l'aide de Networkx, générer aléatoirement un graphe non orienté valué de densité $d = 0.7$ avec des poids choisis aléatoirement entre 2 et 50. Donner le temps d'exécution de votre programme python pour un graphe d'ordre 100, 1000, 2000.
4. Comment pouvez-vous détecter que G généré aléatoirement n'est pas connexe et que E_T^* n'existe pas ?

Algorithme de marquage de Ford-Fulkerson

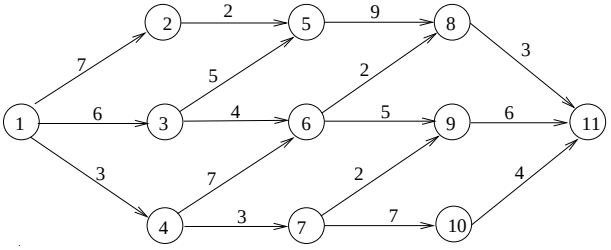
$\phi_u \leftarrow 0 \; \forall u$
Faire
    marque(i) $\leftarrow 0 \; \forall i \in N$
    pred(i) $\leftarrow i \; \forall i \in N$
    marque(s) $\leftarrow +\infty$ et LISTE $\leftarrow \{s\}$
    Tant que LISTE $\neq \emptyset$ et marque(t) $= 0$ faire
        Sélectionner $i \in$ LISTE
        LISTE $\leftarrow$ LISTE $\setminus \{i\}$
        Pour chaque arc $u = (i, j) \in \omega^+(i)$ faire
            Si marque(j)$= 0$ et $c_u > \phi_u$ alors
                marque(j) $\leftarrow c_u - \phi_u$ (marquage de type +)
                pred(j) $\leftarrow i$
                LISTE $\leftarrow$ LISTE $\cup \{j\}$
            FinSi
        FinPour
        Pour chaque arc $u = (j, i) \in \omega^-(i)$ faire
            Si marque(j)$= 0$ et $\phi_u > 0$ alors
                marque(j) $\leftarrow -\phi_u$ (marquage de type -)
                pred(j) $\leftarrow i$
                LISTE $\leftarrow$ LISTE $\cup \{j\}$
            FinSi
        FinPour
    FinTantque
    Si marque(t)$\neq 0$ alors **augmenter**
Tant que marque(t) $\neq 0$

Procédure augmenter
Identifier la chaîne μ à l'aide de pred(t) : $\mu = (s, i_0, i_1, \ldots, i_k, t)$
$\delta \leftarrow \min_{i \in \mu} |$ marque(i) $|$
$\phi_{(s,i_0)} \leftarrow \phi_{(s,i_0)} + \delta$
Pour $j \leftarrow 1$ à k faire
    Si marque(i_j) < 0 alors
        $\phi_{(i_{j-1},i_j)} \leftarrow \phi_{(i_{j-1},i_j)} - \delta$
    Sinon
        $\phi_{(i_{j-1},i_j)} \leftarrow \phi_{(i_{j-1},i_j)} + \delta$
    FinSi
FinPour
$\phi_{(i_k,t)} \leftarrow \phi_{(i_k,t)} + \delta$

TD8 : Flot de valeur maximale et algorithme de marquage de Ford-Fulkerson

Exercice 1 : Application de l'algorithme de Ford et Fulkerson

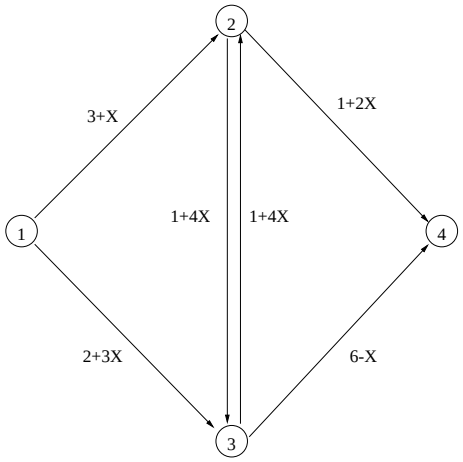
Soit le réseau de transport R représenté dans la figure ci-dessous (les valeurs sur les arêtes représentent des valeurs de capacités) : le sommet 1 est la source, le sommet 11 le puits.



1. Faire circuler dans G le flot suivant :
   chemin(1,2,5,8,11) : 2 unités
   chemin(1,3,6,9,11) : 3 unités
   chemin(1,3,6,8,11) : 1 unité
   chemin(1,4,7,10,11) :3 unités
   Quelle est la valeur de ce flot ? Ce flot est-il complet ? Est-il de valeur maximale ?
2. Appliquer à partir de ce flot l'algorithme de Ford et Fulkerson pour déterminer le flot de valeur maximale.
3. Spécifier la coupe séparant 1 et 11 de capacité minimale.

Exercice 2 : Flot maximum / coupe minimum

Déterminer selon les valeurs de x, x étant un réel compris entre 0 et 2 (bornes comprises), la valeur maximale du flot entre la source 1 et le puits 4.



Exercice 3

Une société de vente par correspondance gère un réseau commercial et logistique composé de trois centres de prises de commandes (C_1, C_2 et C_3), deux centres de préparation de commandes (P_1 et P_2) et deux centres de distribution (D_1 et D_2). Pour répondre à la demande, le directeur désire évaluer la capacité mensuelle du réseau, c'est-à-dire le nombre maximal de commandes pouvant être prises, préparées et livrées en un mois.

Le réseau possède les caractéristiques suivantes (exprimées en milliers de commandes par mois) :
(1) les capacités des centres de prises de commande de C_1, C_2 et C_3 sont respectivement de 30, 30 et 20,
(2) les capacités des centres de préparation de P_1 et P_2 sont respectivement de 10 et 60,
(3) les capacités des centres de distribution de D_1 et D_2 sont respectivement de 30 et 50.

Chaque centre de prise de commande peut alimenter les deux centres de préparation. Toutefois, les capacités des liaisons informatiques limitent à 20000 commandes par mois le flux entre chaque centre C_i et chaque centre P_j. Le centre de distribution D_1 distribue uniquement les commandes préparées par le centre P_1. De même, D_2 distribue uniquement les commandes préparées par P_2. Le centre D_2 a la possibilité de transférer une partie de son activité au centre D_1. Ce transfert ne peut dépasser les 20000 commandes par mois. Il ne réduit pas la capacité de distribution de D_2.

1. Construire un réseau de transport modélisant ce problème.
2. Le directeur envisage la répartition suivante :
   — C_1 envoie 20000 commandes à P_2 et rien à P_1,
   — C_2 envoie 20000 commandes à P_2 et rien à P_1,
   — C_3 envoie 10000 commandes à P_1 et 10000 commandes à P_2,
   — D_2 ne transfère aucune de ces activités à D_1.
   Quel est le flot, sur le réseau défini en a), qui représente cette répartition de la charge de travail ? Quel est la valeur de ce flot ? Est-il complet ? Est-il maximum ?
3. Déterminer le flot maximum. Donner la coupe de valeur minimale. Traduisez ce flot en termes d'activité de chaque centre.
4. On s'attend pour le mois prochain à une demande accrue de 80000 commandes. Pour augmenter la capacité du réseau de distribution, plusieurs solutions sont proposées
   — Le directeur commercial propose d'augmenter les capacités de l'un des centres de C_1 ou C_2,
   — Le directeur informatique propose d'augmenter les capacités de transmission des lignes au départ de C_1 et C_2,
   — Le DRH propose de transférer temporairement (durant le mois pendant lequel la demande est accrue) du personnel de C_1 à C_3, ce qui permettrai un glissement d'une capacité de 10000 commandes par mois au profit de C_3.
   Après une analyse de la coupe de capacité minimale, commentez la pertinence de ces propositions et évaluez les actions à entreprendre.

Exercice 4

Chaque année, un directeur d'école élémentaire doit faire face au problème suivant : il doit inscrire un nombre maximal de ses n élèves à au plus un des m ateliers organisés le soir dans son école. Pour cela, chaque élève choisit au plus 3 ateliers qui l'intéressent. Chaque atelier j dispose d'un nombre limité de places, noté p_j. Il s'agit d'aider le directeur à résoudre au mieux ce problème d'affectation des élèves aux ateliers.

1. Expliquer comment il est possible de représenter les données du problème sous la forme d'un réseau de transport. Pour cela, préciser ce que représentent les sommets et les arcs ainsi que les capacités.
2. Quel problème doit-on résoudre pour déterminer une affectation des élèves aux ateliers qui maximisent le nombre total d'inscriptions ?
3. On suppose que l'affectation des élèves aux ateliers a déjà été effectuée en juillet. Mais, en septembre, 5 nouveaux élèves arrivent et seuls 4 ateliers peuvent encore accepter de nouvelles inscriptions. Il reste 2 places dans l'atelier Roller, 1 place dans l'atelier Théatre, 2 places dans l'atelier Echec et 1 place dans l'atelier Magie. Les 5 nouveaux élèves ont exprimés les choix suivants :
   — Justine souhaite suivre Roller ou Echec,
   — Adèle, Roller ou Théatre,
   — Martin, Théatre ou échec,
   — Ferdinand, Théatre, Echec ou Magie,
   — Clara, Théatre ou Magie.

   (a) Représenter le problème de l'affectation de ces 5 nouveaux élèves aux 4 ateliers dans un réseau de transport noté R=(N,A).

   (b) Le directeur décide d'inscrire Adèle à l'atelier Théatre, Martin à l'atelier Echec et Ferdinand à l'atelier Magie. Il ne propose rien à Clara et Justine.

      i. Spécifier le flot dans R qui représente cette affectation.

      ii. Ce flot est-il complet ? Si non, transformer ce flot de façon à le rendre complet.

      iii. Ce flot est-il de valeur maximale ? Si non, déterminer le flot de valeur maximale.

      iv. Déterminer la coupe de valeur minimale.

Exercice 5. Le théorème de Konig-Egervary

Soit $G = (X, E)$ un graphe non orienté. Rappelons quelques définitions :
— G est biparti si X peut être partitionné en deux ensembles stables X_1 et X_2.
— On appelle couverture de sommets dans G un ensemble C de sommets tel que toute arête de E ait au moins une extrémité dans C. Une couverture minimum est une couverture de cardinalité minimale.
— Un couplage de G est un ensemble $K \subseteq E$ d'arêtes tel que deux arêtes quelconques de K n'aient pas d'extrémités en commun. Un couplage maximum est un couplage de cardinalité maximale.

On se propose de montrer le **théorème de Konig-Egervary** : dans un graphe biparti, la cardinalité d'un couplage maximum est égale à la cardinalité d'une couverture minimum de sommets.

Par la suite, on considère le graphe $G = (X, E)$ biparti représenté en figure 1.

1. Dans le graphe de la figure 1, déterminer un ensemble de sommets C_1 qui constitue une couverture de sommets de cardinalité 3 et déterminer un ensemble d'arêtes K_1 qui constitue un couplage de cardinalité 3.
2. Transformer (et représenter sur cette feuille) le graphe de la figure 1 de la façon suivante : ajouter une source s et un puits t, ajouter des arcs allant de s vers chaque sommet de X_1 (avec une capacité de 1), ajouter des arcs allant de chaque sommet de X_2 vers t (avec une capacité de 1), orienter chaque arête de E en donnant le sommet de X_1 comme extrémité initiale et le sommet de X_2 comme extrémité terminale (les arcs obtenus ont une capacité de $+\infty$).

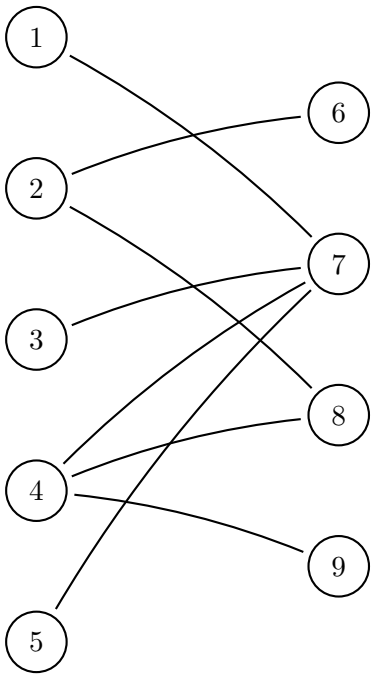


FIGURE 1 – Graphe biparti avec $X_1 = \{1, 2, 3, 4, 5\}$ et $X_2 = \{6, 7, 8, 9\}$

3. L'ensemble $K_2 = \{(2, 6), (4, 7)\}$ constitue un couplage, décrire comment le transformer en un flot de valeur $|K_2| = 2$ de s à t.
4. Montrer alors que la recherche d'un couplage de cardinalité maximum peut se ramener à la recherche d'un flot de valeur maximale de s à t.
5. Partant du flot représentant K_2, déterminer à l'aide de l'algorithme de Ford et Fulkerson, le flot de valeur maximale ainsi que la st-coupe de capacité minimale.
6. Justifier le fait que la st-coupe de capacité minimale ne puisse pas inclure des arcs allant d'un sommet de X_1 vers un sommet de X_2.
7. A partir de la st-coupe de capacité minimale donnée par l'algorithme de Ford-Fulkerson, définir C_2 l'ensemble des sommets i du graphe biparti tels que : soit (s, i), soit (i, t) appartient à la coupe. Montrer que C_2 forme nécessairement une couverture de sommets dans G.
8. Montrez alors, en appliquant le théorème de Ford-Fulkerson (la valeur d'un flot maximum est égale à la capacité minimale d'une coupe), le théorème de Konig-Egervary.