Séq. 11 – Les Graphes

Objectifs

1. Identifier des situations nécessitant une structure de données relationnelles ou arbre
2. Définir un graphe en tant que structure relationnelle (sommets, arcs, arêtes)
3. Distinguer graphes orientés ou non orientés
4. Écrire les implémentations correspondantes d'un graphe (classes Python, matrice d'adjacence, liste de successeurs/de prédécesseurs)
5. Passer d'une implémentation à une autre
6. Modéliser des situations sous forme de graphes (réseau routier, réseau électrique, Internet, réseaux sociaux)

Cette séquence s'appuie sur :

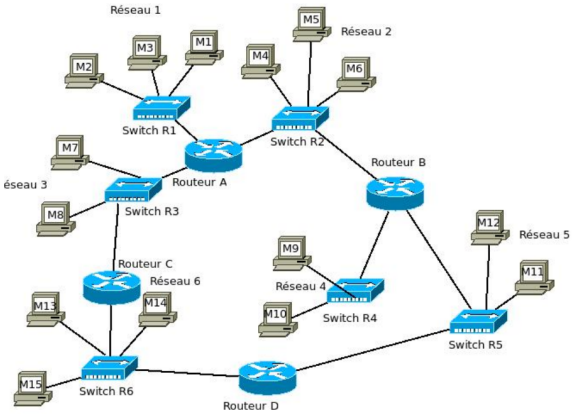
- http://www.monlyceenumerique.fr/nsi_terminale/sd/sd5_graphe.php

1 Introduction

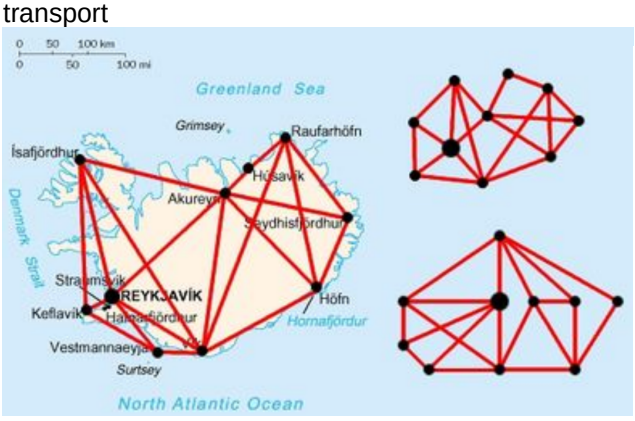
1.1 Dans quels domaines utilise-t-on les graphes ?

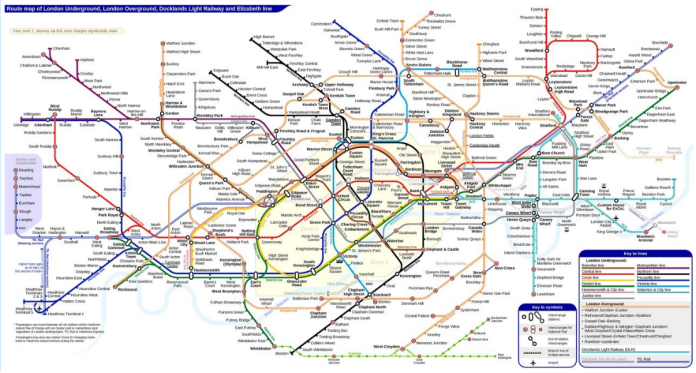
En 2nde SNT, vous avez déjà eu l' occasion de voir des graphes.

- Quand on étudie Internet et les réseaux informatiques, les graphes modélisent des réseaux informatiques entre ordinateurs



- Dans le domaine de la Géolocalisation/Cartographie, les graphes modélisent des réseaux routiers et de transport





- Quand on étudie les réseaux sociaux, les graphes modélisent les relations sociales

1.2 Activité introductive

Imaginez un réseau social ayant 6 abonnés (A, B, C, D, E et F) où :

- A est ami avec B, C et D,
- B est ami avec A et D,
- C est ami avec A, E et D,
- D est ami avec tous les autres abonnés,
- E est ami avec C, D et F,
- F est ami avec E et D,

On peut représenter ce réseau social par un graphe où :

- Chaque abonné est représenté par un cercle avec son nom
- Chaque relation "X est ami avec Y" par un segment de droite reliant X et Y ("X est ami avec Y" et "Y est ami avec X" étant représenté par le même segment de droite).

A faire vous même 1.

Proposez le schéma, représentation de ce réseau social (que l'on appelle graphe).

2 Vocabulaire de base

Les graphes sont des objets mathématiques très utilisés, notamment en informatique.

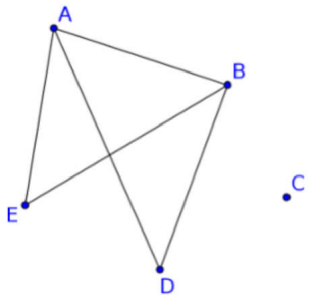
Les cercles sont appelés des sommets et les segments de droites réalisant les liaisons sont appelés des arêtes.

2.1 Définitions :

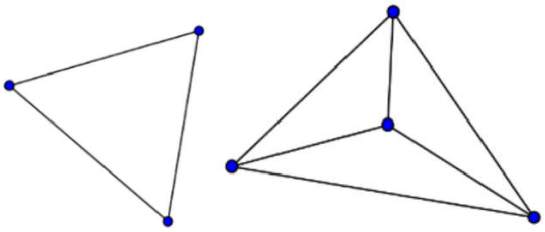
- On appelle graphe .....
  .....
- Le nombre de sommets d'un graphe.....
- Deux sommets reliés entre eux par une arête sont dits .....
- Le degré d'un sommet est .....
- Un sommet qui n'est adjacent à aucun autre sommet du graphe est dit .....
- Un graphe est dit complet si.....
  .....

2.2 Exemples :

- Le graphe ci-contre est d'ordre 5 car il possède 5 sommets.
- Les sommets A et B sont adjacents.
- Les sommets D et E ne sont pas adjacents.
- Le sommet C est isolé.



- Ci-contre deux graphes complets d'ordre 3 et 4.



A faire vous même 2.

En reprenant le vocabulaire précédent, caractérisez le graphe de l' activité introductive.

.....

.....

Sommet *(ici, ami)*	sommets adjacents *(ici, … est amis avec…)*	Degré du sommet
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....

.....

3 Différents types de graphes

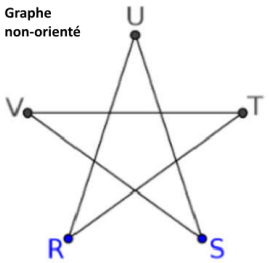
Un graphe peut être orienté ou non-orienté :

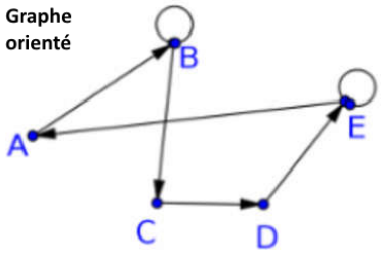
- Dans un graphe non-orienté, ...
- Dans un graphe orienté, ...

.....

Un graphe (orienté ou non-orienté) peut contenir des boucles c'est-à-dire.....

.....(on a par exemple une boucle B sur la représentation ci-dessous).





Propriété :

Le nombre d'arêtes est égal à la moitié de la somme des degrés des sommets

Ce résultat s'explique assez facilement: en ajoutant les degrés de chaque sommet (c'est à dire le nombre d'arêtes issues de ce sommet), on comptabilise deux fois chaque arête (une fois avec le sommet d'une extrémité et une seconde fois avec le sommet de l'autre extrémité de l'arête). Une boucle est donc comptabilisée deux fois (exemple1 : dans un sens R-T-V-S-U-R et l'autre sens R-U-S-V-T-R, soit 5 arêtes et une somme des degrés égale à 10) .

Il découle de cette propriété que :

- la somme des degrés des sommets est nécessairement paire (exemple1 : la somme des degrés est 10)
- le nombre de sommets de degré impair dans le graphe est un nombre pair (exemple1 : 0 sommets de degré impair, 0 est pair)

A faire vous même 3.

Vérifiez cette propriété pour le graphe de l' activité introductive.

4 Notions de chaîne et de cycle

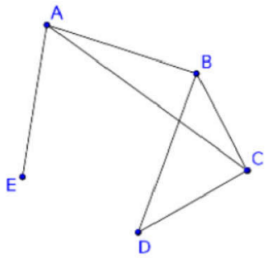
Dans un graphe orienté ou non,

- .....

  .....
- Le nombre d'arêtes qui composent une chaîne est appelé .....
- On appelle chaîne fermée toute chaîne dont.....
- On appelle cycle.....

Exemple, dans le graphe ci-contre :

- E-A-C-B est une chaîne de longueur 3.
- E-A-C-B-A-E est une chaîne fermée de longueur 5.
- D-B-A-C-D est un cycle de longueur 4.
- E-A-C-B-A-E n'est pas un cycle car l'arête A-E est parcourue deux fois.



A faire vous même 4.

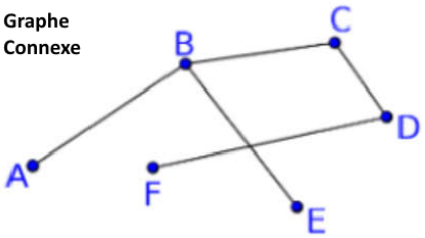
Pour le graphe de l' activité introductive,

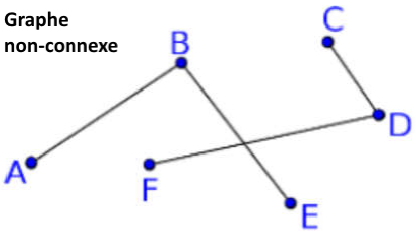
1. Identifier une chaîne non fermée et donner sa longueur.
2. Identifier une chaîne fermée et donner sa longueur.
3. Identifiez un cycle et donner sa longueur.

5 Notion de connexité

Un graphe est dit connexe si .....

.....





Un graphe est complet lorsque.....

.....

.....

A faire vous même 5.

Que dire de la connexité du graphe de l' activité introductive ? Expliquer.

6 Chaîne et cycle eulérien

On appelle chaîne eulérienne d'un graphe.....

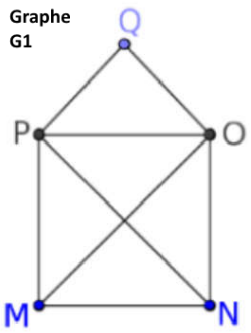
.....

On appelle cycle eulérien.....

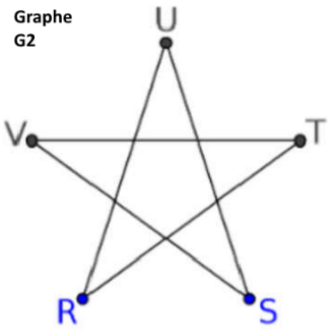
Dans le graphe G1, la chaîne M-P-Q-O-M-N-P-O-N est une chaîne eulérienne (elle contient bien 8 arêtes toutes distinctes).

Dans le graphe G2, Le cycle R-U-S-V-T-R est un cycle eulérien.

Graphe G1



Graphe G2



Théorème d'Euler :

Soit G un graphe connexe.

- G admet un cycle eulérien si et seulement si tous les sommets de G sont de degré pair.
- G admet une chaîne eulérienne (non fermée) si et seulement si le nombre de sommets de degré impair dans G est 2.
  Dans ce cas, les extrémités de la chaîne eulérienne sont les deux sommets de degré impair.

A faire vous même 6.

Pour le graphe de l' activité introductive, si possible, donner une chaîne eulérienne et un cycle eulérien.

A faire vous même 7.

- Lire livre P. 152-153
- Consulter la vidéo https://youtu.be/YYv2R1cCTa0

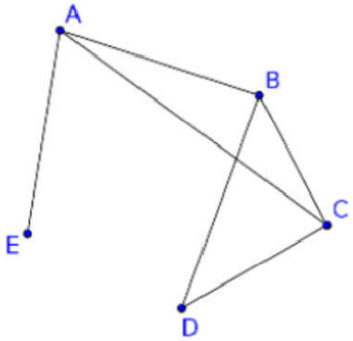
7 Représenter un graphe par une matrice d'adjacence

Considérons un graphe d'ordre n (entier naturel non nul). On numérote ses sommets de 1 à n .

On appelle matrice d'adjacence associée à ce graphe la matrice A dont le terme a_{ij} vaut 1 si les sommets et sont reliés par une arête et 0 sinon.

Exemple :

La matrice d'adjacence est obtenue avec la mise en forme dans un tableau "carré".



	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	0
E	1	0	0	0	0

Dans le cas d'un graphe non orienté, les coefficients a_{ij} et a_{ji} sont égaux pour tout i et tout j compris entre 1 et n .

Autrement dit, la matrice d'adjacence est symétrique suivant une diagonale.

Rq : Dans le cas d'un graphe orienté, la matrice d'adjacence n'est pas a priori symétrique.

A faire vous même 8.

Écrire la matrice d'adjacence représentant le graphe de l' activité introductive.

	A	B	C	D	E	F
A	.....	.....	.....	.....	.....	.....
B	.....	.....	.....	.....	.....	.....
C	.....	.....	.....	.....	.....	.....
D	.....	.....	.....	.....	.....	.....
E	.....	.....	.....	.....	.....	.....
F	.....	.....	.....	.....	.....	.....

8 Graphe non orienté et implémentation en Python

8.1 Utilisation d' objets dédiés et du module pygraphviz

A faire vous même 9.

Pour créer nos arbres à l' aide de python, nous allons installer :

- Graphviz : https://graphviz.org/
- Pygraphviz : https://pygraphviz.github.io/

Voir le tutoriel pygraphviz : https://pygraphviz.github.io/documentation/stable/tutorial.html

- Graphs

  To make an empty pygraphviz graph use the AGraph class:

```
>>> import pygraphviz as pgv
>>> G = pgv.AGraph()
```

  You can use the strict and directed keywords to control what type of graph you want. The default is to create a strict graph (no parallel edges or self-loops). To create a digraph with possible parallel edges and self-loops use

```
>>> G = pgv.AGraph(strict=False, directed=True)
```

  You may specify a dot format file to be read on initialization:

```
>>> G = pgv.AGraph("Petersen.dot")
```

  Other options for initializing a graph are using a string,

```
>>> G = pgv.AGraph("graph {1 - 2}")
```

  using a dict of dicts,

```
>>> d = {"1": {"2": None}, "2": {"1": None, "3": None}, "3": {"2": None}}
>>> A = pgv.AGraph(d)
```

  or using a SWIG pointer to the AGraph datastructure,

```
>>> h = A.handle
>>> C = pgv.AGraph(h)
```

- Nodes, and edges¶

  Nodes and edges can be added one at a time

```
>>> G.add_node("a")  # adds node 'a'
>>> G.add_edge("b", "c")  # adds edge 'b'-'c' (and also nodes 'b', 'c')
>>> nodelist = ["f", "g", "h"]
```

  or from lists or containers.

```
>>> nodelist = ["f", "g", "h"]
>>> G.add_nodes_from(nodelist)
```

  If the node is not a string an attempt will be made to convert it to a string

```
>>> G.add_node(1)  # adds node '1'
```

- Attributes¶

  To set the default attributes for graphs, nodes, and edges use the graph_attr, node_attr, and edge_attr dictionaries

```
>>> G.graph_attr["label"] = "Name of graph"
>>> G.node_attr["shape"] = "circle"
```

```
>>> G.edge_attr["color"] = "red"
```

Graph attributes can be set when initializing the graph

```
>>> G = pgv.AGraph(ranksep="0.1")
```

Attributes can be added when adding nodes or edges,

```
>>> G.add_node(1, color="red")
>>> G.add_edge("b", "c", color="blue")
```

or through the node or edge attr dictionaries,

```
>>> n = G.get_node(1)
>>> n.attr["shape"] = "box"
>>> e = G.get_edge("b", "c")
>>> e.attr["color"] = "green"
```

- Layout and Drawing¶

  Pygraphviz provides several methods for layout and drawing of graphs.
  To store and print the graph in dot format as a Python string use

  ```
  >>> s = G.string()
  ```

  To write to a file use

  ```
  >>> G.write("file.dot")
  ```

  To add positions to the nodes with a Graphviz layout algorithm

  ```
  >>> G.layout()  # default to neato
  >>> G.layout(prog="dot")  # use dot
  ```

  To render the graph to an image

  ```
  >>> G.draw("file.png")  # write previously positioned graph to PNG file
  >>> G.draw("file.ps", prog="circo")  # use circo to position, write PS file
  ```

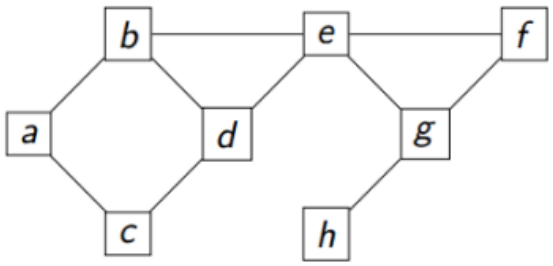
A faire vous même 10.

Afficher le graphe de l' activité introductive avec la méthode ci-dessus.

8.2 Modélisation par dictionnaire d'adjacence

8.2.1 Création de graphe

Un graphe peut être représenté par une matrice d'adjacence composée de 1 et de 0 selon que deux sommets sont ou ne sont pas reliés par une arête.



	a	b	c	d	e	f	g	h
a	0	1	1	0	0	0	0	0
b	1	0	0	1	1	0	0	0
c	1	0	0	1	0	0	0	0
d	0	1	1	0	1	0	0	0
e	0	1	0	1	0	1	1	0
f	0	0	0	0	1	0	1	0
g	0	0	0	0	1	1	0	1
h	0	0	0	0	0	0	1	0

Une façon d'encoder un graphe en Python est d'utiliser un dictionnaire. Les clés seront les sommets du graphe et leur valeur sera la liste des sommets adjacents. Prenons par exemple le graphe ci-dessus.

```
G = dict()
G['a'] = ['b','c']
G['b'] = ['a','d','e']
G['c'] = ['a','d']
G['d'] = ['b','c','e']
G['e'] = ['b','d','f','g']
G['f'] = ['e','g']
G['g'] = ['e','f','h']
G['h']= ['g']
print(G)
```

A faire vous même 11.

Implémenter le graphe de l' activité introductive avec une liste d'adjacence.

8.2.2 Exploitation

Pour exploiter un graphe implémenté sous forme de liste d'adjacence, voici quelques rappels sur les dictionnaires :

```
print(G.keys()) # affiche les clés
print(G.values()) # affiche les valeurs
print(len(G)) # affiche le nombre de clés
print(G['e'])# affiche la valeur de la clé 'e'
```

Remarque :

`G.keys()` et `G.values()` sont itérables.

```
# affiche les valeurs du dictionnaire
for el in G.values():
    print(el)
#affiche les clés et les valeurs des clés
for key in G.keys():
    print(key,G[key])
```

A faire vous même 12.

Tester les commandes précédentes pour le graphe de l' activité introductive.

A faire vous même 13.

Écrire des fonctions permettant d'obtenir les informations suivantes sur le graphe G:

1. `ordre` qui prend en paramètre un dictionnaire et renvoie un entier, l'ordre de ce graphe.
2. `degre` qui prend en paramètre un dictionnaire ainsi qu'un sommet à explorer sous forme d'une chaîne de caractères et qui renvoie un entier, le degré de ce sommet avec une précondition pour l'éventuelle absence du sommet dans le graphe.
3. `sommets_adjacents` qui prend en paramètre un dictionnaire ainsi qu'un sommet à explorer sous forme d'une chaîne de caractères et qui renvoie la liste des sommets adjacents à ce sommet, avec une précondition pour l'éventuelle absence du sommet dans le graphe.
4. `lister_aretes` qui prend en paramètre un dictionnaire et renvoie la liste des arêtes d'un graphe. Une arête sera représentée par un tuple à deux éléments (attention aux doublons).

A faire vous même 14.

Afficher l'abonné du réseau social du graphe de l' activité introductive qui a le plus grand nombre d'amis ainsi que leurs noms.

8.3 Modélisation par matrice d'adjacence

La matrice d'adjacence est une liste de listes.

On peut l'écrire à la main.

Ou bien utiliser le code suivant pour la fabriquer à partir de la liste des sommets et du dictionnaire G précédent.

```
[[0, 1, 1, 0, 0, 0, 0, 0],
 [1, 0, 0, 1, 1, 0, 0, 0],
 [1, 0, 0, 1, 0, 0, 0, 0],
 [0, 1, 1, 0, 1, 0, 0, 0],
 [0, 1, 0, 1, 0, 1, 1, 0],
 [0, 0, 0, 0, 1, 0, 1, 0],
 [0, 0, 0, 0, 1, 1, 0, 1],
 [0, 0, 0, 0, 0, 0, 1, 0]]
>>>
```

```
liste_sommets=['a','b','c','d','e','f','g','h']
n=len(liste_sommets)
matrice=[[0]*n for i in range(n)]
for i in range(n):
    for j in range(n):
        if liste_sommets [j] in G[liste_sommets [i]]:
            matrice[i][j]=1
print(matrice)
```

A faire vous même 15.

Afficher la matrice d'adjacence du graphe de l' activité introductive avec la méthode ci-dessus.

A faire vous même 16.

- Programmez une fonction `dico_to_matrix` qui transforme un dictionnaire d' adjacence en une matrice d' adjacence.
- Programmez une fonction `matrix_to_dico` qui transforme une matrice d' adjacence en un dictionnaire d' adjacence.

A faire vous même 17. Pour les plus rapides

- Programmez une fonction `dico_to_graph` qui transforme un dictionnaire d' adjacence en un graphe pygraphviz.
- Programmez une fonction `matrix_to_graph` qui transforme une matrice d' adjacence en graphe pygraphviz.

8.4 choix d'implémentation

Nous avons vu précédemment qu'il est possible d'implémenter un graphe soit à partir d'un dictionnaire d'adjacence, soit à partir d'une matrice d'adjacence.

Sur quel(s) critère(s) en effectuer le choix ? Cela dépend :

- de la structure du graphe :
  Plus un graphe est "dense", c'est-à-dire plus il y a d'arêtes pour un nombre donnés de sommets, plus le choix de la matrice d'adjacence est pertinente.
  En effet, supposons un graphe avec n sommets et m arêtes :
  - la complexité en mémoire d'un dictionnaire d'adjacence est en $O(n^2)$
  - la complexité en mémoire d'une liste de listes d'adjacence est en $O(n+m)$
- du travail à faire sur ce graphe :
  Vous verrez ultérieurement des algorithmes pour étudier et travailler avec des graphes. Certains algorithmes nécessiteront d'implémenter le graphe d'une manière plutôt qu'une autre.
  Deux situations pour comprendre. Pour un graphe à n sommets, si on veut: (voir 1 et 2 ci-dessous)
  1. tester que deux sommets sont adjacents ou non :
     On a une complexité en temps en $O(1)$ pour un dictionnaire d'adjacence (lecture directe d'un terme de la matrice) mais en $O(d)$ pour une liste d'adjacence où d correspond au degré d'un sommet testé (on doit, dans le cas où il n'y a pas d'arête reliant les sommets, balayer toute la liste des sommets voisins)
     L'implémentation sous forme de matrice d'adjacence est alors préférable.
  2. itérer sur les voisins d'un sommet :
     On a une complexité en temps en $O(n)$ pour une matrice d'adjacence (lecture de tous les termes d'une même ligne (ou colonne) pour connaître quels sont les sommets voisins) mais en $O(d)$ pour un dictionnaire d'adjacence où d correspond au degré d'un sommet testé (on doit balayer toute la liste des sommets voisins).

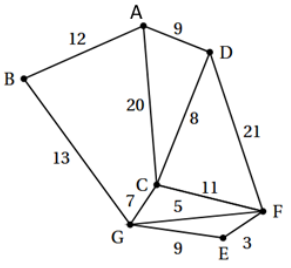
  L'implémentation sous forme de dictionnaire d'adjacence est alors préférable.

9 Graphe non orienté pondéré

On dit qu'un graphe est pondéré si .....

..... Suivant la situation, ce nombre peut être un temps, une distance ou toute autre grandeur relative à la situation représenté par ce graphe.

Le graphe pondéré de l'exemple ci-dessous peut être représenté par la matrice d'adjacence qui se trouve à côté.



0	12	20	9	0	0	0
12	0	0	0	0	0	13
20	0	0	8	0	11	7
9	0	8	0	0	21	0
0	0	0	0	0	3	9
0	0	11	21	3	0	5
0	13	7	0	9	5	0

A faire vous même 18.

Implémenter le graphe pondéré ci-dessus associé à sa matrice d'adjacence en python et avec le module pygraphviz. Pour cela :

Adaptez la fonction `matrix_to_dico` afin qu' elle :

- prenne comme 1er paramètre `matrice` une matrice d'adjacence de type liste de liste, matrice qui contient les poids du graphe pondéré,
- prenne comme 2nd paramètre `noms` une liste de chaînes de caractères qui correspond aux noms des sommets, sommets apparaissant dans l'ordre de la matrice,
- renvoie un dictionnaire dont les clés sont les sommets et les valeurs sont une liste de tuples au format `[('sommet1', poids1), ('sommet2', poids2),…]`

```
noms=["A","B","C","D","E","F","G"]
matrice=[ [0,12,20,9,0,0,0],
          [12,0,0,0,0,0,13],
          [20,0,0,8,0,11,7],
          [9,0,8,0,0,21,0],
          [0,0,0,0,0,3,9],
          [0,0,11,21,3,0,5],
          [0,13,7,0,9,5,0]]
```

A faire vous même 19. Pour les plus rapides

Utiliser cette fonction `matrix_to_dico` et adaptez `dico_to_graph` pour créer le graphe.

10 Graphe orienté et implémentation en Python

Lorsque les arêtes sont marquées d'une flèche, elles sont orientées. On les appelle alors des arcs.

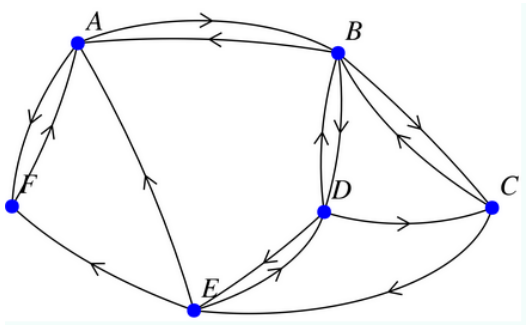
Un arc ne se parcourt que dans le sens de la flèche. Dans ce cas on note généralement les arcs avec des parenthèses pour désigner des couples. Par exemple l'arc (A, B) part de A et arrive en B.

Dans un graphe orienté, on appelle chemin de longueur n une succession de n arcs tels que l'extrémité de chacun est l'origine du suivant (sauf pour le dernier arc).

On dit que c'est un chemin fermé lorsque l'origine du premier arc coïncide avec l'extrémité du dernier.

Un chemin fermé composé d'arcs tous distincts est appelé circuit.

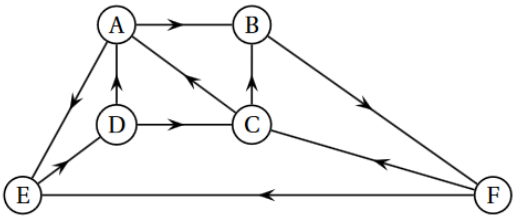
Exemple :



- On peut se rendre du sommet C vers le sommet E mais pas du sommet E vers le sommet C : c'est donc un graphe orienté.
- On peut se rendre du sommet B vers le sommet A et du sommet A vers le sommet B : deux arcs sont nécessaires.
- À partir du sommet C, on peut se rendre au sommet C grâce à un circuit : C−B−D−C
- Le chemin A−B−C−E−D−B est un chemin de longueur 5.
- Le chemin A−B−D−E−F−A est un circuit de ce graphe.

A faire vous même 20.

Le graphe suivant représente le plan d'une ville. Les arcs du graphe représentent ses avenues commerçantes, en prenant en compte le sens de circulation, et les sommets du graphe les carrefours de ces avenues.



1. Est-ce un graphe orienté ou non-orienté ? Pourquoi ?
2. Quel est l'ordre du graphe ?
3. Proposer un chemin de longueur 4 d'origine F et d'extrémité B.
4. Proposer un circuit passant par tous les sommets.

A faire vous même 21.

Implémenter le graphe ci-dessus en python et avec le module pygraphviz. Pour cela quelques commandes adaptées aux graphes orientés :

```
graphe = nx.DiGraph()   # nx.DiGraph() pour un graphe orienté
graphe.add_edges_from([("s1","s2"),("s1","s3")]) # 2 arcs issues du sommet s1, la fleche part de s1
```

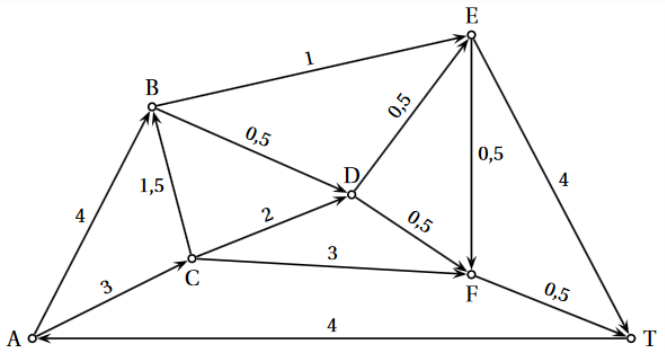
A faire vous même 22.

Testez les méthodes successors et predecessors sur le graphe précédent.

A faire vous même 23. Pour les plus rapides

Le graphe ci-contre représente, dans un aéroport donné, toutes les voies empruntées par les avions au roulage. Ces voies, sur lesquelles circulent les avions avant ou après atterrissage, sont appelées taxiways. Les sommets du graphe sont les intersections. Les arcs du graphe représentent les voies de circulation (les « taxiways ») en prenant en compte le sens de circulation pour les avions dans les différentes voies ainsi que le temps de parcours pour chacune en minute(s).

1. En rangeant les sommets par ordre alphabétique, déterminer la matrice associée à ce graphe orienté pondéré.
2. Implémenter cette matrice en Python en tant que tableau de tableau.



A faire vous même 24. Pour les plus curieux

Voir cette vidéo d' Arté : http://ninoo.fr/LC/Term_NSI/seq11_graphes/arte_tv_les_graphes.mp4

11 Exercice type bac

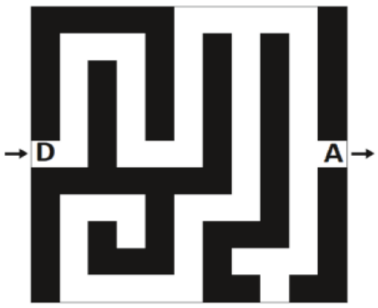
11.1 Représentation d'un labyrinthe à l'aide d'un tableau et parcours (adaptation d'un énoncé de bac 2021)

On modélise un labyrinthe par un tableau à deux dimensions à n lignes et m colonnes avec m et n des entiers strictement positifs.

La case en haut à gauche est repérée par $(0,0)$ et la case en bas à droite par $(n-1,m-1)$.

Dans ce tableau :

- 0 représente une case vide, hors case de départ et arrivée,
- 1 représente un mur,
- 2 représente le départ du labyrinthe,
- 3 représente l'arrivée du labyrinthe.




```
lab1 = [[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [2, 0, 1, 0, 0, 0, 1, 0, 1, 0, 3],
        [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1]]
```

11.1.1 Recherche d'une solution dans un labyrinthe

On suppose dans cette partie que les labyrinthes possèdent un unique chemin allant du départ à l'arrivée sans repasser par la même case.

Dans la suite, c'est ce chemin que l'on appellera solution du labyrinthe.

Pour déterminer la solution d'un labyrinthe, on parcourt les cases vides de proche en proche.

Lors d'un tel parcours, afin d'éviter de tourner en rond, on choisit de marquer les cases visitées.

Pour cela, on remplace la valeur d'une case visitée dans le tableau représentant le labyrinthe par la valeur 4.

L'algorithme est le suivant (c'est du backtracking).

On initialise une variable chemin, de type list, par le couple (iD,jD) les coordonnées du départ

i par la valeur iD

j par la valeur jD

Tant que l'arrivée n'a pas été atteinte :

- on marque la case visitée avec la valeur 4 ;
- si la case visitée possède une case voisine libre,
  - la première case de la liste renvoyée par la fonction voisines devient la prochaine case à visiter et on ajoute à la liste chemin ;
- sinon, il s'agit d'une impasse. On supprime alors la dernière case dans la liste chemin. La prochaine case à visiter est celle qui est désormais en dernière position de la liste chemin.

Le but des questions suivantes est de compléter le programme `labyrintheTableauEleve.py`

1. Écrire une fonction `est_valide ( i , j , n , m )` qui renvoie `True` si le couple `(i,j)` correspond à des coordonnées valides pour un labyrinthe de taille `(m,n)` et `False` sinon.

   On donne ci-dessous des exemples d'appels.

   ```
   >>> est_valide(5, 2, 10, 10)
   True
   >>> est_valide(-3, 4, 10, 10)
   False
   ```

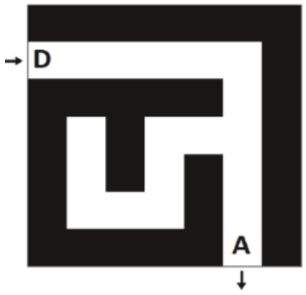
2. On dit que deux cases d'un labyrinthe sont voisines si elles ont un côté commun.

   Ecrire une fonction `voisines(i, j, lab)` qui prend en arguments le couple `(i,j)` représentant les coordonnées d'une case et un tableau lab qui représente un labyrinthe. Cette fonction renvoie la liste (l'ordre ne compte pas) des coordonnées des cases voisines de la case de coordonnées `(i,j)` qui sont valides, non visitées et qui ne sont pas des murs.

   Par exemple l'appel `voisines(1, 1, [[1, 1, 1], [4, 0, 0], [1, 0, 1]])` renvoie la liste `[(2, 1), (1, 2)]`.

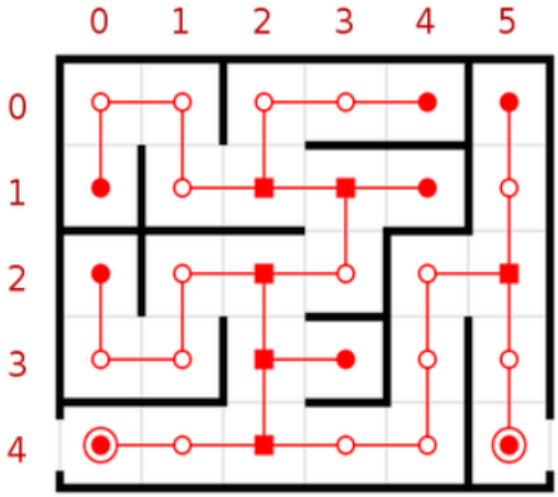
3. La fonction `resolution` dont les données sont le labyrinthe, les ligne et colonne du départ et qui renvoie une solution sous la forme d'un tableau des coordonnées des cases.

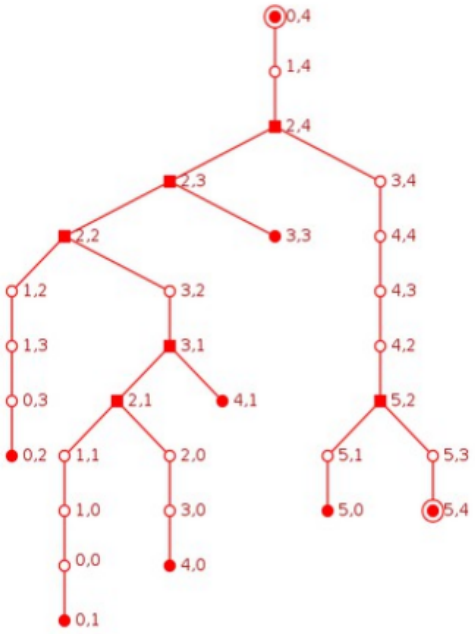
   L'appel `resolution(lab2)` doit renvoyer `[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5)]`.



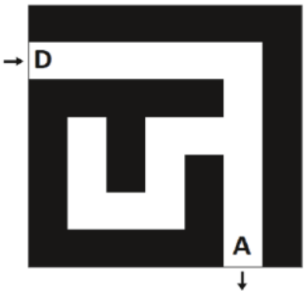

```
lab2 = [[1, 1, 1, 1, 1, 1, 1],
        [2, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0, 1],
        [1, 0, 1, 0, 0, 0, 1],
        [1, 0, 1, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 0, 1],
        [1, 1, 1, 1, 1, 3, 1]]
```

11.1.2 Labyrinthe représenté à l'aide d'un graphe

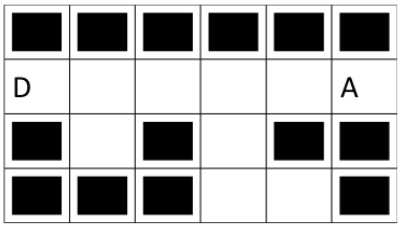




1. Dessiner le graphe associé à lab2 pour un départ en (1,0)



2. Afin de pouvoir utiliser un dictionnaire d'adjacence, on numérote chacune des cases :
   Ainsi dans un tableau à n lignes et m colonnes la case de coordonnées `(i,j)` a pour numéro : `i×m+ j`
3. Le but des questions suivantes est de compléter le programme labyrintheTableauEleve.py



Pour lab3, la case D est la case numérotée : 1×6+0=6

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

4. Le but des questions suivantes est de compléter le programme `labyrintheTableauEleve.py`
   Le programme gère la transformation tableau → graphe (les cases de départ et d'arrivée étant mise à 0).
   Aussi les numéros de nœuds prennent des valeurs de 0 à n*m
   a. Compléter la fonction `numero_noeud(i,j,m)` où `(i, j)` sont les cordonnées de la case et `m` la largeur du tableau.

```
>>> numero_noeud(1,0,6)
6
>>> numero_noeud(3,4,6)
22
```

   b. Compléter le fonction `determination_adjacent(tab,i,j)` où `tab` est le tableau représentant le labyrinthe et `(i, j)` sont les cordonnées d'une case.

```
>>> determination_adjacent(lab3,1,3)
[15, 8, 10]
```

   c. Compléter `dict_adjacent(tab)` où `tab` est le tableau représentant le labyrinthe et qui renvoie le graphe sous forme d'un dictionnaire d'adjacence.

```
>>> dict_adjacent(lab3)
{6: [7], 7: [13, 6, 8], 8: [7, 9], 9: [15, 8, 10], 10:[9, 11],
 11: [10], 13: [7], 15: [9, 21], 21:[15, 22], 22: [21]}
```

   d. Compléter la fonction `parcours_largeur(graphe,s, a)` où `graphe` est un dictionnaire d'adjacence, `s` le sommet de départ et `a` le sommet d'arrivée.
   Le parcours en largeur a été adapté :
   on s'arrête si on rencontre le sommet d'arrivé
   on stocke dans un dictionnaire de clé : futur sommet sommet empilé, valeur : l'antécédent.