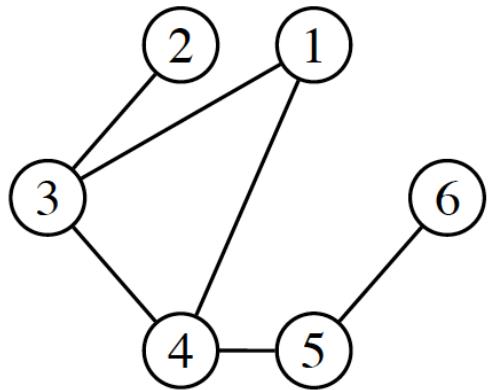


1.2 Graphe partiel et sous-graphe

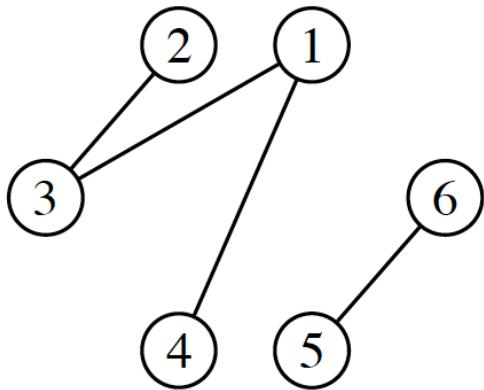
Soit $G = (V, E)$ un graphe. Le graphe $G' = (V, E')$ est un **graphe partiel** de G , si E' est inclus dans E . Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G .



Graphe G

$$V = \{1, 2, 3, 4, 5, 6\}$$

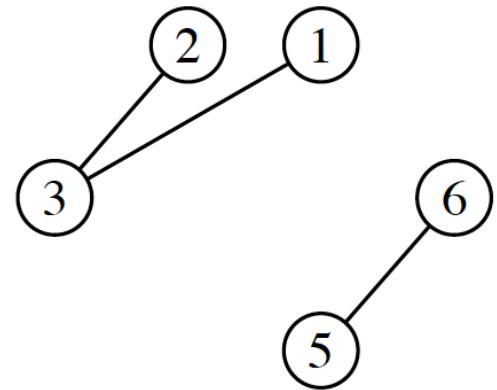
$$E = \{\{1, 3\}, \{1, 4\}, \{2, 3\}, \\ \{3, 4\}, \{4, 5\}, \{5, 6\}\}$$



Graphe partiel de G

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{\{1, 3\}, \{1, 4\}, \\ \{2, 3\}, \{5, 6\}\}$$



Sous-graphe de G

$$V = \{1, 2, 3, 5, 6\}$$

$$E = \{\{1, 3\}, \{2, 3\}, \{5, 6\}\}$$

Un graphe partiel d'un sous-graphe est un **sous-graphe partiel** de G . On appelle **clique** un sous-graphe complet de G . Dans le graphe G ci-dessus, le sous-graphe $K = (V, E)$, avec $V = \{1, 3, 4\}$ et $E = \{\{1, 3\}, \{1, 4\}, \{3, 4\}\}$ est une clique.

Exercice 5

Montrez que dans un groupe formé de six personnes, il y en a nécessairement trois qui se connaissent mutuellement ou trois qui ne se connaissent pas (on suppose que si A connaît B , B connaît également A).

Montrez que cela n'est plus nécessairement vrai dans un groupe de cinq personnes.

On appelle **degré** du sommet v , et on note $d(v)$, le nombre d'arêtes incidentes à ce sommet.

Attention ! Une boucle sur un sommet compte double.

Dans un graphe simple, on peut aussi définir le degré d'un sommet comme étant le nombre de ses voisins (la taille de son voisinage).

Dans le multigraphe ci-contre, on a les degrés :

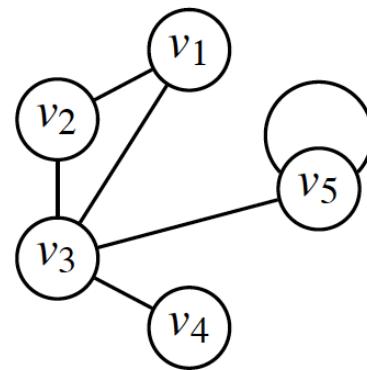
$$d(v_1) = 2$$

$$d(v_2) = 2$$

$$d(v_3) = 4$$

$$d(v_4) = 1$$

$$d(v_5) = 3$$



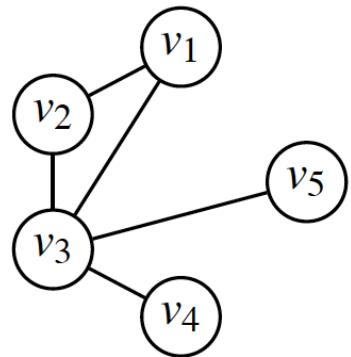
Théorème 1.1 (Lemme des poignées de mains)

La somme des degrés des sommets d'un graphe est égale à deux fois le nombre d'arêtes.

Exercice 6

Démontrez le lemme des poignées de mains.

Le **degré d'un graphe** est le degré maximum de tous ses sommets. Dans l'exemple ci-dessous, le degré du graphe est 4, à cause du sommet v_3 .



Un graphe dont tous les sommets ont le même degré est dit **régulier**. Si le degré commun est k , alors on dit que le graphe est k -régulier.

Exercice 8

Montrez que dans une assemblée de n personnes, il y a toujours au moins 2 personnes qui ont le même nombre d'amis présents.

Exercice 9

Est-il possible de relier 15 ordinateurs de sorte que chaque appareil soit relié avec exactement trois autres ?

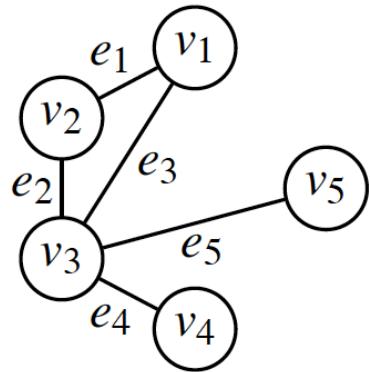
Exercice 10

On s'intéresse aux graphes 3-réguliers. Construisez de tels graphes ayant 4, 5, 6, puis 7 sommets. Qu'en déduisez-vous ? Prouvez-le !

Une **chaîne** dans G , est une suite ayant pour éléments alternativement des sommets et des arêtes, commençant et se terminant par un sommet, et telle que chaque arête est encadrée par ses extrémités.

On dira que la chaîne **relie** le premier sommet de la suite au dernier sommet. En plus, on dira que la chaîne a pour longueur le nombre d'arêtes de la chaîne.

Le graphe ci-dessous contient entre autres les chaînes $(v_1, e_1, v_2, e_2, v_3, e_5, v_5)$ et $(v_4, e_4, v_3, e_2, v_2, e_1, v_1)$.



On appelle **distance** entre deux sommets la longueur de la plus petite chaîne les reliant.

On appelle **diamètre** d'un graphe la plus longue des distances entre deux sommets.

Une chaîne est **élémentaire** si chaque sommet y apparaît au plus une fois.

Une chaîne est **simple** si chaque arête apparaît au plus une fois. Dans le graphe précédent, $(v_1, e_1, v_2, e_2, v_3)$ est une chaîne simple et élémentaire.

Une chaîne dont les sommets de départ et de fin sont les mêmes est appelée chaîne **fermée**.

Dans le graphe précédent, $(v_4, e_4, v_3, e_5, v_5, e_5, v_3, e_4, v_4)$ est une chaîne fermée.

Une chaîne fermée simple est appelée **cycle**. Dans le graphe précédent, la chaîne $(v_1, e_1, v_2, e_2, v_3, e_3, v_1)$ est un cycle.

Exercice 13

Quels sont les graphes de diamètre 1 ?

Théorème 1.2

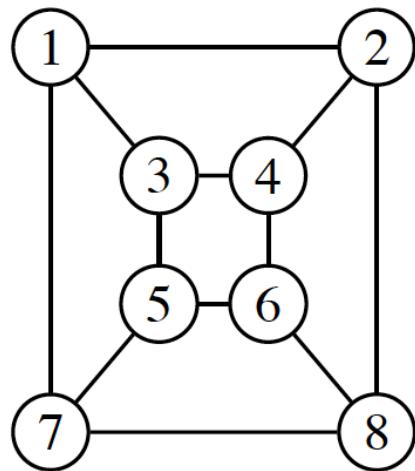
Un graphe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire.

Exercice 14

Démontrez le théorème 1.2.

Exercice 15

Montrez que ce graphe est biparti :



Ann a rencontré Betty, Charlotte, Félicia et Georgia.

Betty a rencontré Ann, Charlotte, Edith, Félicia et Helen.

Charlotte a rencontré Ann, Betty et Edith.

Edith a rencontré Betty, Charlotte et Félicia.

Félicia a rencontré Ann, Betty, Edith et Helen.

Georgia a rencontré Ann et Helen.

Helen a rencontré Betty, Félicia et Georgia.

Vous voyez, mon cher Holmes, les réponses sont concordantes !

On appelle **cycle eulérien** d'un graphe G un cycle passant une et une seule fois par chacune des arêtes de G . Un graphe est dit **eulérien** s'il possède un cycle eulérien.

On appelle **chaîne eulérienne** d'un graphe G une chaîne passant une et une seule fois par chacune des arêtes de G . Un graphe ne possédant que des chaînes eulériennes est **semi-eulérien**.

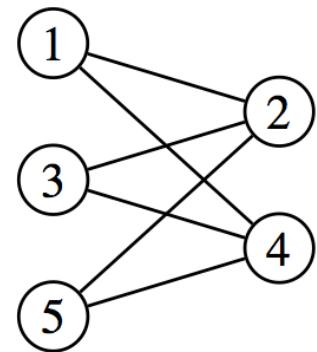
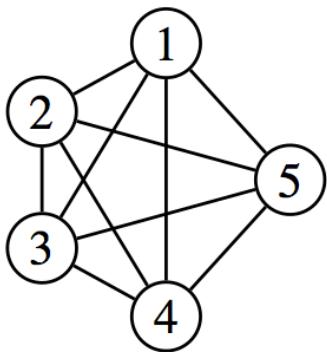
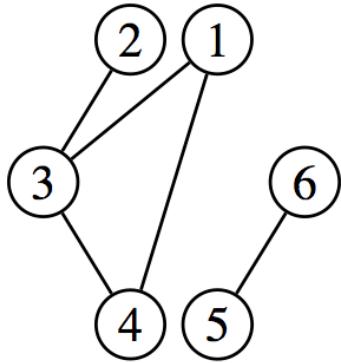
Plus simplement, on peut dire qu'un graphe est eulérien (ou semi-eulérien) s'il est possible de dessiner le graphe sans lever le crayon et sans passer deux fois sur la même arête.

Exercice 18

Donnez un critère permettant de dire à coup sûr si un graphe est eulérien.

Exercice 19

Les graphes suivants sont-ils eulériens (ou semi-eulériens) ?



Exercice 20

Soit G un graphe non eulérien. Est-il toujours possible de rendre G eulérien en lui rajoutant un sommet et quelques arêtes ?

On appelle **cycle hamiltonien** d'un graphe G un cycle passant une et une seule fois par chacun des sommets de G . Un graphe est dit **hamiltonien** s'il possède un cycle hamiltonien.

On appelle **chaîne hamiltonienne** d'un graphe G une chaîne passant une et une seule fois par chacun des sommets de G . Un graphe ne possédant que des chaînes hamiltoniennes est **semi-hamiltonien**.

Exercice 23

Dessinez un graphe d'ordre au moins 5 qui est...

- 1) hamiltonien et eulérien
- 2) hamiltonien et non eulérien
- 3) non hamiltonien et eulérien
- 4) non hamiltonien et non eulérien.

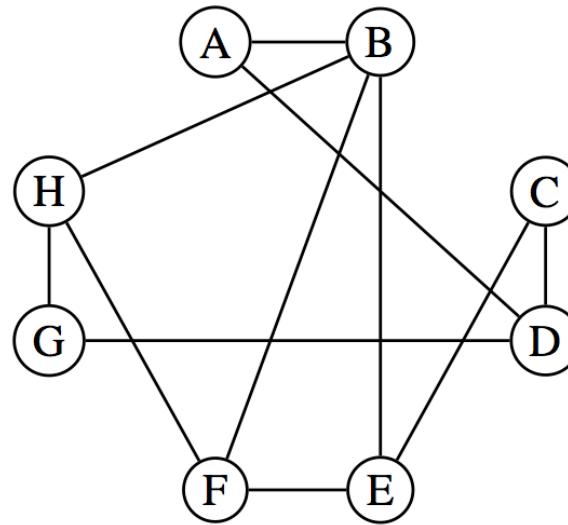
Exercice 24

Un club de 9 joueurs se réunit chaque jour autour d'une table ronde. Une règle du club interdit qu'un joueur ait deux fois la même personne à côté de lui.

- 1) Combien de jours au maximum pourront-ils se réunir en satisfaisant cette règle ?
- 2) Donnez une organisation de la table pour chacun de ces jours.
- 3) Même question que 1), mais avec 3 tables de 3 places.
- 4) Donnez une organisation des trois tables pour chacun de ces jours.

Exercice 25

Huit personnes se retrouvent pour un repas de mariage. Le graphe ci-dessous précise les incompatibilités d'humeur entre ces personnes (une arête reliant deux personnes indique qu'elles ne se supportent pas).

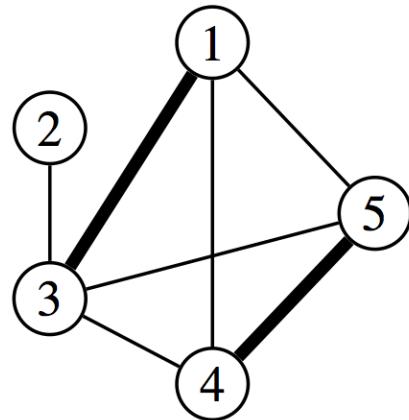


Proposez un plan de table (la table est ronde) en évitant de placer côté à côté deux personnes incompatibles.

Soit G un graphe simple. Un **couplage** C de G est un sous-graphe partiel 1-régulier de G . On peut aussi dire qu'un couplage (ou appariement) est un ensemble d'arêtes deux à deux non-adjacentes.

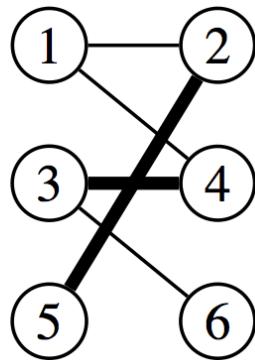
Un sommet v est **saturé** par un couplage C si v est l'extrémité d'une arête de C . Dans le cas contraire, v est **insaturé**.

Un **couplage maximum** est un couplage contenant le plus grand nombre possible d'arêtes. Un graphe peut posséder plusieurs couplages maximum.

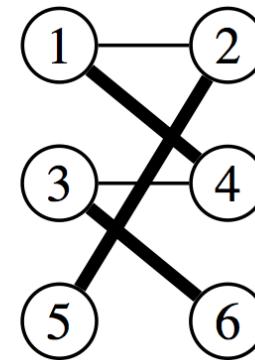


En gras, un couplage maximum de G . Les sommets 1, 3, 4 et 5 sont saturés.

Un **couplage parfait** est un couplage où chaque sommet du graphe est saturé.



Un couplage



Un couplage maximum et parfait

Exercice 26

Une assemblée est formée de personnes parlant plusieurs langues différentes (voir tableau ci-après). On veut former des binômes de personnes qui pourront dialoguer entre elles. Comment maximiser le nombre de binômes ?

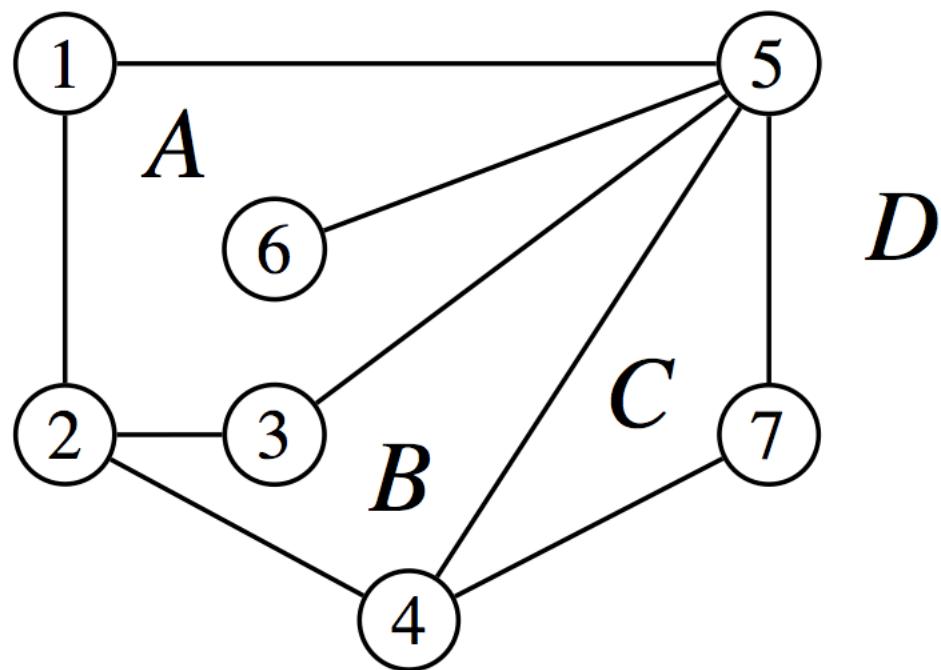
	Allemand	Anglais	Arabe	Chinois	Français	Espagnol	Russe
Alfred				X	X		
Bernard	X					X	
Claude					X	X	
Denis	X						
Ernest		X					
Fabien		X			X		X
Georges		X			X		
Henri			X	X		X	
Isidore					X		X
Joseph		X	X				
Kurt	X	X					
Louis							X

Exercice 27

Une entremetteuse essaie de former le plus de couples possible avec 6 filles et 6 garçons en fonction de critères de compatibilité d'humeur. Elle a dressé le tableau d'incompatibilités ci-après, où une croix indique que deux personnes sont incompatibles. Combien de couples pourra-t-elle former au maximum ?

	Anne	Béatrice	Carine	Drew	Eléonore	Florie
Alfred	x	x		x	x	
Bernard						x
Claude	x			x	x	x
Denis	x				x	x
Ernest		x	x			
Fabien	x		x	x	x	

On dit qu'un graphe est **planaire** si on peut le dessiner dans le plan de sorte que ses arêtes ne se croisent pas. Rappelons que les arêtes ne sont pas forcément rectilignes.

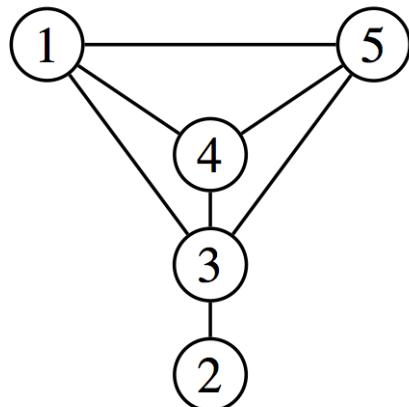


1.9 Représentations non graphiques d'un graphe

1.9.1 Matrice d'adjacences

On peut représenter un graphe simple par une **matrice d'adjacences**. Une matrice $(n \times m)$ est un tableau de n lignes et m colonnes. (i, j) désigne l'intersection de la ligne i et de

la colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie que le sommet i est adjacent au sommet j .



$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

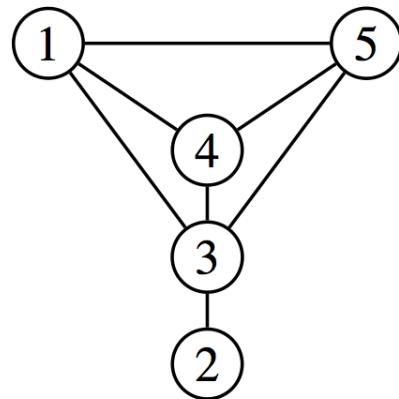
Exercice 30

On a calculé ci-dessous les matrices M^2 et M^3 (M est la matrice ci-dessus). Pour chacune de ces matrices, à quoi correspondent les nombres obtenus ?

$$M^2 = \begin{pmatrix} 3 & 1 & 2 & 2 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 0 & 4 & 2 & 2 \\ 2 & 1 & 2 & 3 & 2 \\ 2 & 1 & 2 & 2 & 3 \end{pmatrix} \quad M^3 = \begin{pmatrix} 6 & 2 & 8 & 7 & 7 \\ 2 & 0 & 4 & 2 & 2 \\ 8 & 4 & 6 & 8 & 8 \\ 7 & 2 & 8 & 6 & 7 \\ 7 & 2 & 8 & 7 & 6 \end{pmatrix}$$

1.9.2 Listes d'adjacences

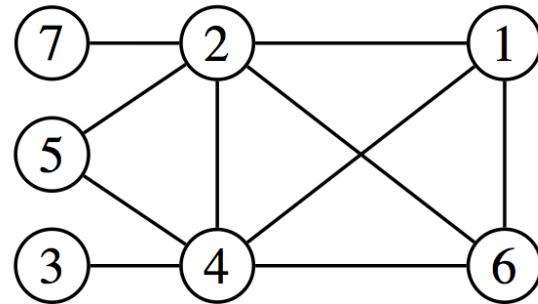
On peut aussi représenter un graphe simple en donnant pour chacun de ses sommets la liste des sommets auxquels il est adjacent. Ce sont les **listes d'adjacences**.



1 : 3, 4, 5
2 : 3
3 : 1, 2, 4, 5
4 : 1, 3, 5
5 : 1, 3, 4

Exercice 31

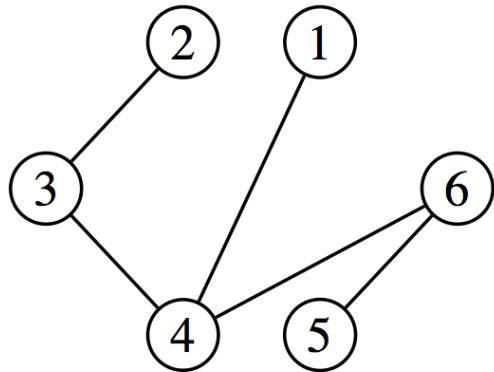
Décrivez le graphe G ci-dessous par une matrice d'adjacences et des listes d'adjacences.



1.10 Arbres

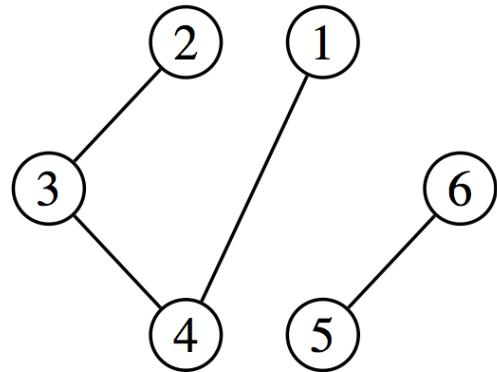
On appelle **arbre** tout graphe connexe sans cycle. Un graphe sans cycle mais non connexe est appelé une **forêt**.

Une **feuille** ou **sommet pendant** est un sommet de degré 1.



Arbre

Les sommets 1, 2 et 5 sont les feuilles



Forêt

Les sommets 1, 2, 5 et 6 sont les feuilles

Théorème 1.10

Les affirmations suivantes sont équivalentes pour tout graphe G à n sommets.

1. G est un arbre,
2. G est sans cycle et connexe,
3. G est sans cycle et comporte $n - 1$ arêtes,
4. G est connexe et comporte $n - 1$ arêtes,
5. chaque paire u, v de sommets distincts est reliée par une seule chaîne simple (et le graphe est sans boucle).

Exercice 33

Démontrez le théorème 1.11.

Exercice 34

Combien d'arbres différents existe-t-il avec 5 sommets ? avec 6 sommets ? avec 7 sommets ?

1.10.1 Codage de Prüfer

Le codage de Prüfer (1918) est une manière très compacte de décrire un arbre. Il a été proposé par le mathématicien allemand Ernst Paul Heinz Prüfer (1896-1934).

Codage

Soit l'arbre $T = (V, E)$ et supposons $V = \{1, 2, \dots, n\}$.

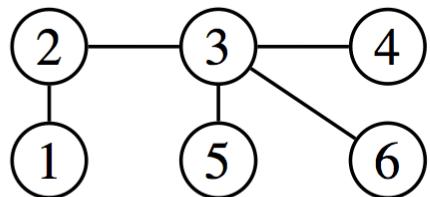
L'algorithme ci-dessous fournira le code de T , c'est-à-dire une suite S de $n - 2$ termes employant (éventuellement plusieurs fois) des nombres choisis parmi $1, \dots, n$.

Pas général de l'algorithme de codage

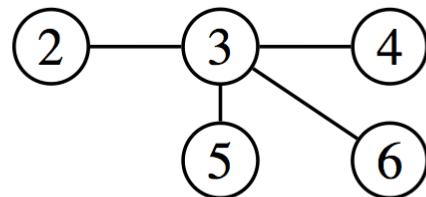
(à répéter tant qu'il reste plus de deux sommets dans l'arbre T)

1. identifier la feuille v de l'arbre ayant le numéro minimum ;
2. ajouter à la suite S le seul sommet s adjacent à v dans l'arbre T ;
3. enlever de l'arbre T le sommet v et l'arête incidente à v .

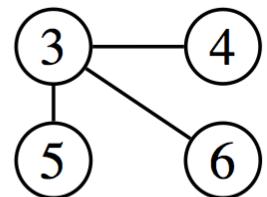
Exemple de codage



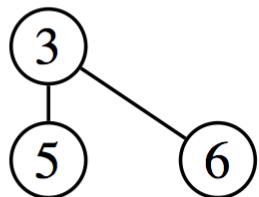
$$S = \{\}$$



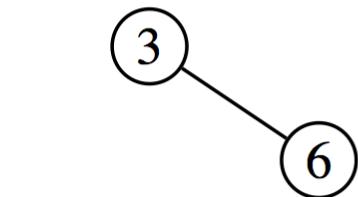
$$S = \{2\}$$



$$S = \{2, 3\}$$



$$S = \{2, 3, 3\}$$



$$S = \{2, 3, 3, 3\}$$

Il reste 2 sommets :
fin du codage

Décodage

Donnée : suite S de $n - 2$ nombres, chacun provenant de $\{1, \dots, n\}$.

Posons $I = \{1, \dots, n\}$.

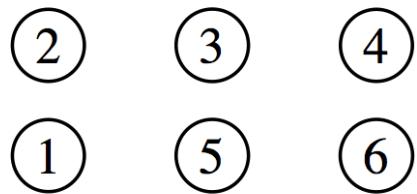
Pas général de l'algorithme de décodage

(à répéter tant qu'il reste des éléments dans S et plus de deux éléments dans I)

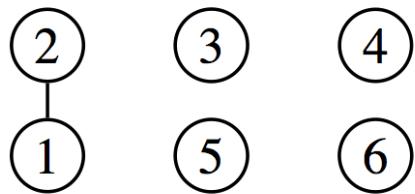
1. identifier le plus petit élément i de I n'apparaissant pas dans la suite S ;
2. relier par une arête de T le sommet i avec le sommet s correspondant au premier élément de la suite S ;
3. enlever i de I et s de S .

Les deux éléments qui restent dans I à la fin de l'algorithme constituent les extrémités de la dernière arête à ajouter à T .

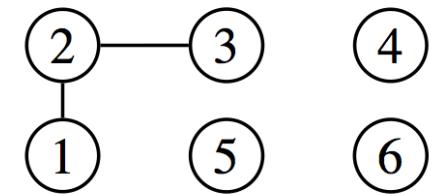
Exemple de décodage



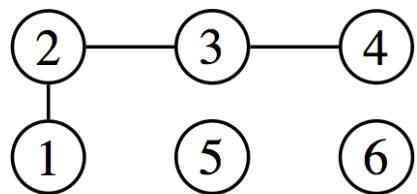
$$S = \{2, 3, 3, 3\}$$
$$I = \{1, 2, 3, 4, 5, 6\}$$



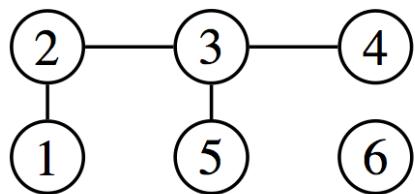
$$S = \{3, 3, 3\}$$
$$I = \{2, 3, 4, 5, 6\}$$



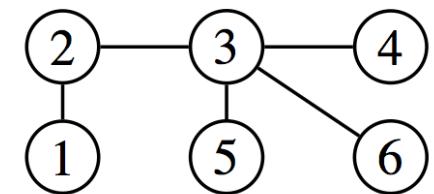
$$S = \{3, 3\}$$
$$I = \{3, 4, 5, 6\}$$



$$S = \{3\}$$
$$I = \{3, 5, 6\}$$



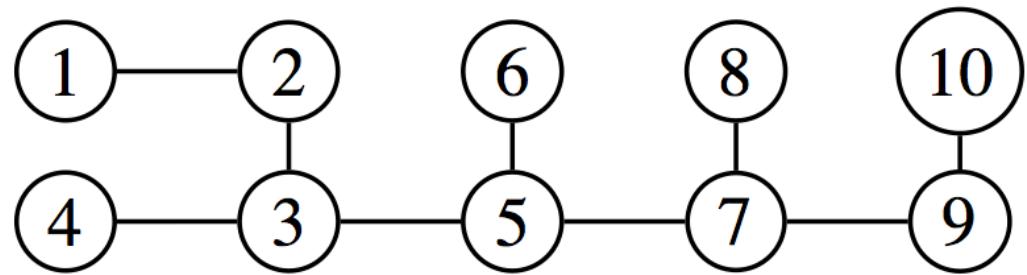
$$S = \{\}$$
$$I = \{3, 6\}$$



$$S = \{\}$$
$$I = \{\}$$

Exercice 36

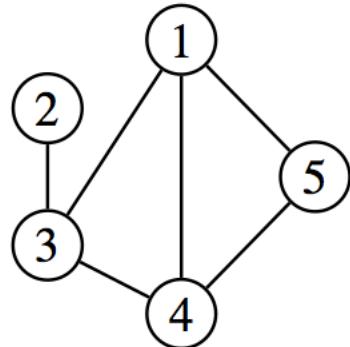
Trouvez le codage de Prüfer de l'arbre ci-dessous.



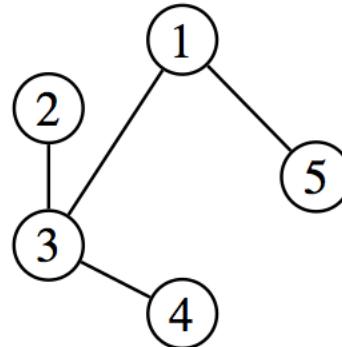
Exercice 37

Dessinez l'arbre correspondant à la suite $S = \{1, 1, 1, 1, 1, 1, 1, 1\}$.

1.11 Arbres couvrants



Graphe G



Un arbre couvrant

Un **arbre couvrant** (aussi appelé arbre maximal) est un graphe partiel qui est aussi un arbre.

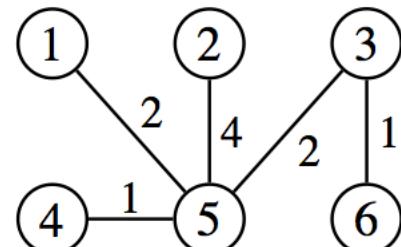
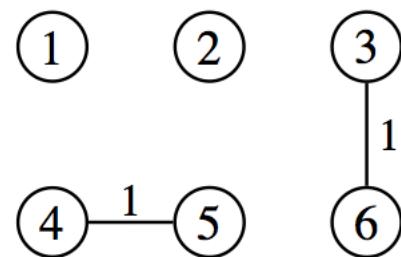
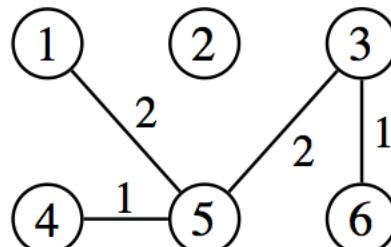
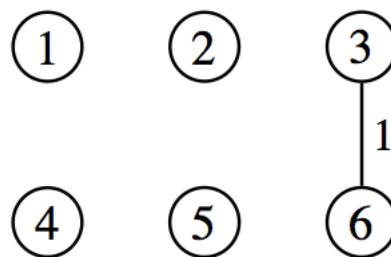
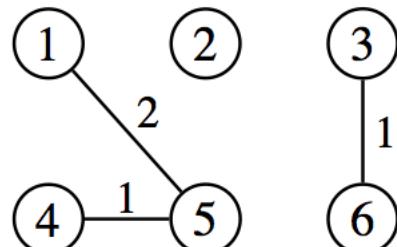
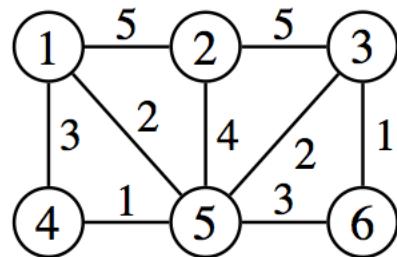
Exercice 39

Combien d'arbres couvrants différents le graphe G ci-dessus possède-t-il ?

1.11.1 Arbre couvrant de poids minimum

Soit le graphe $G = (V, E)$ avec un poids associé à chacune de ses arêtes. On veut trouver, dans G , un arbre maximal $A = (V, F)$ de poids total minimum.

Exemple



Algorithme de Kruskal (1956)

Données :

- Graphe $G = (V, E)$ ($|V| = n$, $|E| = m$)
- Pour chaque arête e de E , son poids $c(e)$.

Résultat : Arbre ou forêt maximale $A = (V, F)$ de poids minimum.

Trier et renuméroter les arêtes de G dans l'ordre croissant de leur poids :

$$c(e_1) \leq c(e_2) \leq \dots \leq c(e_m).$$

Poser $F := \emptyset$, $k := 0$

Tant que $k < m$ et $|F| < n - 1$ faire

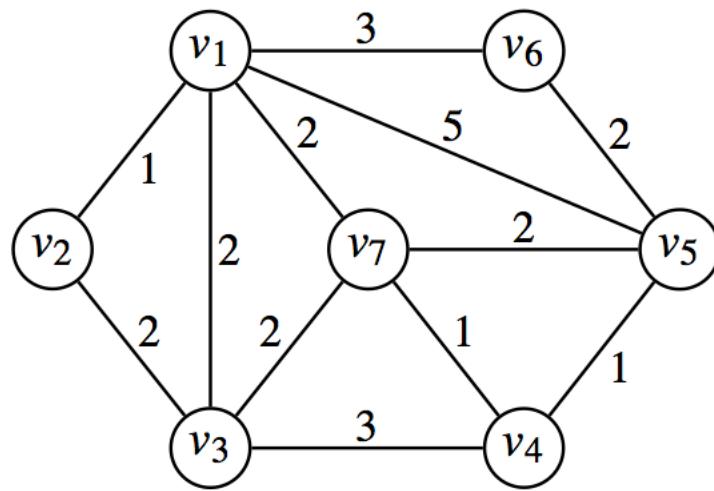
 Début

 si e_{k+1} ne forme pas de cycle avec F alors $F := F \cup \{e_{k+1}\}$
 $k := k + 1$

 Fin

Exercice 40

Trouvez tous les arbres couvrants de poids minimum du graphe ci-après (les chiffres sur les arêtes représentent leur poids).



PYTHON

- Langage de programmation
 - Interprété
 - Très flexible au niveau du typage
 - Portable par rapport à l'OS (attention aux versions)
-
- Un programme python est un fichier texte typiquement avec l'extension **.py**
 - Un programme python se lance en tapant

python nom_fichier.py

PYTHON

script.py IPython Shell

```
1  ''' Python program to find the
2  multiplication table (from 1 to 10)'''
3
4  num = 12
5
6  # To take input from the user
7  # num = int(input("Display multiplication table of? "))
8
9  # use for loop to iterate 10 times
10 for i in range(1, 11):
11     print(num, 'x', i, '=', num*i)
```

```
12 x 9 = 108
12 x 10 = 120
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
```

PYTHON

```
1 # define punctuation
2 punctuations = '''!()-[]{};:'"\,;<.>/?@#$%^&*_~'''
3
4 my_str = "Hello!!!, he said ---and went."
5
6 # To take input from the user
7 # my_str = input("Enter a string: ")
8
9 # remove punctuation from the string
10 no_punct = ""
11 for char in my_str:
12     if char not in punctuations:
13         no_punct = no_punct + char
14
15 # display the unpunctuated string
16 print(no_punct)
```

Hello he said and went

PYTHON

script.py IPython Shell

```
1 # Program to perform different set operations like in mathematics
2
3 # define three sets
4 E = {0, 2, 4, 6, 8};
5 N = {1, 2, 3, 4, 5};
6
7 # set union
8 print("Union of E and N is",E | N)
9
10 # set intersection
11 print("Intersection of E and N is",E & N)
12
13 # set difference
14 print("Difference of E and N is",E - N)
15
16 # set symmetric difference
17 print("Symmetric difference of E and N is",E ^ N)
```

Union of E and N is {0, 1, 2, 3, 4, 5, 6, 8}

Intersection of E and N is {2, 4}

Difference of E and N is {8, 0, 6}

Symmetric difference of E and N is {0, 1, 3, 5, 6, 8}

PYTHON

script.py IPython Shell

```
1 # Program to count the number of each vowel in a string
2
3 # string of vowels
4 vowels = 'aeiou'
5
6 # change this value for a different result
7 ip_str = 'Hello, have you tried our tutorial section yet?'
8
9 # uncomment to take input from the user
10 #ip_str = input("Enter a string: ")
11
12 # make it suitable for caseless comparisions
13 ip_str = ip_str.casifold()
14
15 # make a dictionary with each vowel a key and value 0
16 count = {}.fromkeys(vowels,0)
17
18 # count the vowels
19 for char in ip_str:
20     if char in count:
21         count[char] += 1
22
23 print(count)
```

```
{'o': 5, 'e': 5, 'i': 3, 'a': 2, 'u': 3}
```

PYTHON

```
1 # Program to multiply two matrices using nested loops
2
3 # 3x3 matrix
4 X = [[12,7,3],
5     [4 ,5,6],
6     [7 ,8,9]]
7 # 3x4 matrix
8 Y = [[5,8,1,2],
9     [6,7,3,0],
10    [4,5,9,1]]
11 # result is 3x4
12 result = [[0,0,0,0],
13             [0,0,0,0],
14             [0,0,0,0]]
15
16 # iterate through rows of X
17 for i in range(len(X)):
18     # iterate through columns of Y
19     for j in range(len(Y[0])):
20         # iterate through rows of Y
21         for k in range(len(Y)):
22             result[i][j] += X[i][k] * Y[k][j]
23
24 for r in result:
25     print(r)
```

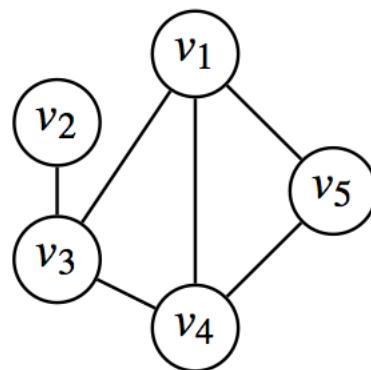
```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
```

PYTHON

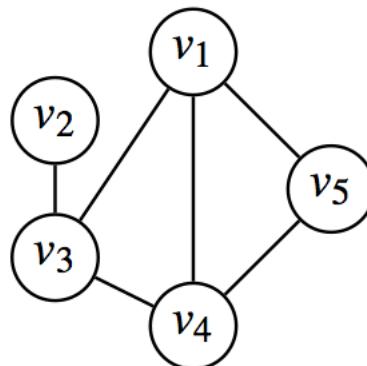
```
import numpy
import math
n = input("Taille du graphe: ");
m = numpy.zeros((n,n))
for i in range(n):
    for j in range(n):
        v = 2
        while((v!=0)and(v!=1)):
#            print("m[{0},{1}] = ".format(i,j), end=" ")
#            v = int(input())
            v = int(input("m[{0},{1}] = ".format(i,j)))
            if((v!=0)and(v!=1)):
                print("La valeur doit etre 0 ou 1")
            else:
                m[i,j] = int(v)
print('\n'.join([''.join(['{:2d}'.format(int(item)) for item in row]) for row in m]))
# degré
deg_in = numpy.zeros((n))
deg_out = numpy.zeros((n))
for i in range(n):
    for j in range(n):
        deg_in[i] += m[i,j]
        deg_out[j] += m[i,j]
for i in range(n):
    print("Sommet {0}: degré = {1}  degré< = {2}".format(i+1,int(deg_in[i]),int(deg_out[i])))
```

STABILITE

Soit $G = (V, E)$ un graphe. Un sous-ensemble S de V est un **stable** s'il ne comprend que des sommets non adjacents deux à deux. Dans le graphe ci-dessous, $\{v_1, v_2\}$ forment un stable ; $\{v_2, v_4\}$ aussi, ainsi que $\{v_2, v_5\}$ et $\{v_3, v_5\}$.



Le cardinal du plus grand stable est le **nombre de stabilité** de G ; on le note $\alpha(G)$. Dans le graphe ci-dessous, on a $\alpha(G)=2$.

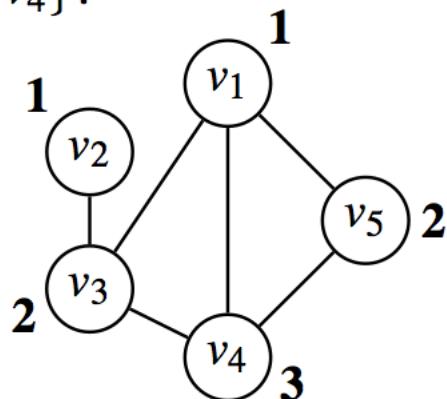


COLORATION

La **coloration** des sommets d'un graphe consiste à affecter à tous les sommets de ce graphe une couleur de telle sorte que deux sommets adjacents ne portent pas la même couleur. Une coloration avec k couleurs est donc une partition de l'ensemble des sommets en k stables.

Le **nombre chromatique** du graphe G , noté $\gamma(G)$, est le plus petit entier k pour lequel il existe une partition de V en k sous-ensembles stables.

Sur le graphe ci-dessous, on a eu besoin de trois couleurs (notées 1, 2 et 3) pour colorer les sommets de sorte que deux sommets adjacents aient des couleurs différentes. On a donc trois stables : $\{v_1, v_2\}$, $\{v_3, v_5\}$ et $\{v_4\}$. On ne peut pas utiliser moins de couleurs, à cause des cliques $\{v_1, v_4, v_5\}$ et $\{v_1, v_3, v_4\}$.



COLORATION

Majoration

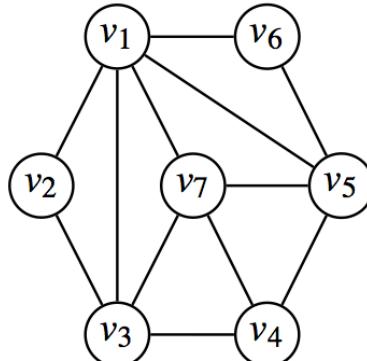
- $\gamma(G) \leq r + 1$, où r est le plus grand degré des sommets de G .
- $\gamma(G) \leq n + 1 - \alpha(G)$

Minoration

- Le nombre chromatique d'un graphe est supérieur ou égal à celui de chacun de ses sous-graphes.
- Le nombre chromatique du graphe sera supérieur ou égal à l'ordre de sa plus grande clique, que l'on note $\omega(G)$ (prononcer oméga de G). Autrement dit, $\gamma(G) \geq \omega(G)$

Exercice 41

Majorez et minorez le nombre chromatique de ce graphe.



Testez le code ci-dessous

```
from turtle import *
def carre () :
    color ("red")
    begin_fill()
    for i in range (4) :
        down()
        forward (50)
        left(90)
    end_fill()

def ligne():
    for i in range (3):
        carre()
        up()
        forward (60)

x=0
y=0
for i in range (3) :
    goto(x,y)
    ligne()
    x=0
    y=y-60
```

PYTHON Dessin

Testez le code ci-dessous

```
setpos(50,100)
circle(10)
left(90)
forward(2)
write("15",False,align="center",font=("Arial",12,"normal"))
```

En déduire une fonction **sommet(c,x,y)**
qui dessine le sommet en position (x,y)
avec comme label c

Ecrire une fonction **arc(x,y,z,t)**
qui trace une ligne de (x,y) à (z,t)

Ecrire une fonction **graph(M)**
qui utilise la matrice d'adjacence M pour dessiner le
Graph correspondant

DIJKTRA

Entrées : $G = (S, A)$ un graphe avec une pondération positive *poids* des arcs, s_{deb} un sommet de S

$P := \emptyset$

$d[a] := +\infty$ pour chaque sommet a

$d(s_{deb}) = 0$

Tant qu'il existe un sommet hors de P

 Choisir un sommet a hors de P de plus petite distance $d[a]$

 Mettre a dans P

 Pour chaque sommet b hors de P voisin de a

$d[b] = \min(d[b], d[a] + \text{poids}(a, b))$

 Fin Pour

Fin Tant Que

DIJKTRA

Initialisation de l'algorithme [modifier | mod

```
Initialisation(G,sdeb)
1 pour chaque point s de G faire
2   d[s] := infini
3 fin pour
4 d[sdeb] := 0
```

Recherche d'un nœud de distance minimale

- On recherche un nœud de distance minimale implémenté pour cela une fonction Trouve_min

```
Trouve_min(Q)
1 mini := infini
2 sommet := -1
3 pour chaque sommet s de Q
4   si d[s] < mini
5     alors
6       mini := d[s]
7       sommet := s
8 renvoyer sommet
```

DIJKTRA

Mise à jour des distances [modifier | modifier le code]

- On met à jour les distances entre s_{deb} et s_2 en se posant la question : vaut-il mieux passer par s_1 ou pas ?

```
maj_distances(s1,s2)
1 si d[s2] > d[s1] + Poids(s1,s2)      /* Si la distance de sdeb à s2 est plus grande que */
2                                /* celle de sdeb à S1 plus celle de S1 à S2 */
3 alors
4     d[s2] := d[s1] + Poids(s1,s2) /* On prend ce nouveau chemin qui est plus court */
5     prédecesseur[s2] := s1        /* En notant par où on passe */
```

Fonction principale [modifier | modifier le code]

Voici la fonction principale utilisant les précédentes fonctions annexes :

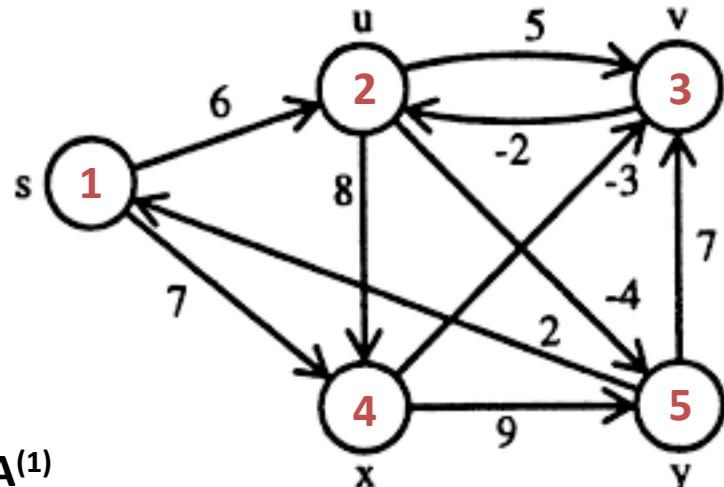
```
Dijkstra(G,Poids,sdeb)
1 Initialisation(G,sdeb)
2 Q := ensemble de tous les nœuds
3 tant que Q n'est pas un ensemble vide faire
4     s1 := Trouve_min(Q)
5     Q := Q privé de s1
6     pour chaque nœud s2 voisin de s1 faire
7         maj_distances(s1,s2)
8     fin pour
9 fin tant que
```

DIJKTRA

Le plus court chemin de s_{deb} à s_{fin} peut ensuite se calculer itérativement selon l'algorithme suivant,

```
1 A = suite vide
2 s := sfin
3 tant que s != sdeb faire
4   A = A + s                      /* on ajoute s à la suite A */
5   s = prédecesseur[s]            /* on continue de suivre le chemin */
6 fin tant que
```

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	0	7	0	0

$A^{(2)}$

0	6	1	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-4
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-4
2	0	7	9	0

$A^{(4)}$

0	4	7	-2
Inf	0	5	-4
Inf	-2	0	-6
Inf	-5	-3	-9
2	4	6	0

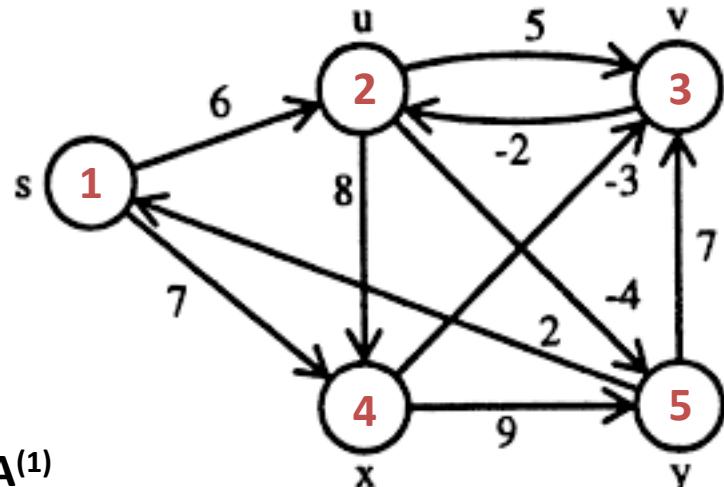
$A^{(5)}$

0	4	7	-2
-4	0	-2	-6
-5	-3	0	-9
2	4	6	0

$A =$

0	0	Inf	8	Inf
Inf	0	0	Inf	-4
Inf	-2	0	0	Inf
Inf	Inf	-3	0	0
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	0	7	0	0

$A^{(2)}$

0	6	1	7	2
Inf	0	5	8	-4
Inf	-2	0	6	0
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	0
2	0	7	9	0

$A^{(4)}$

0	4	7	-2
Inf	0	5	-4
Inf	-2	0	6
Inf	-5	-3	0
2	4	6	9

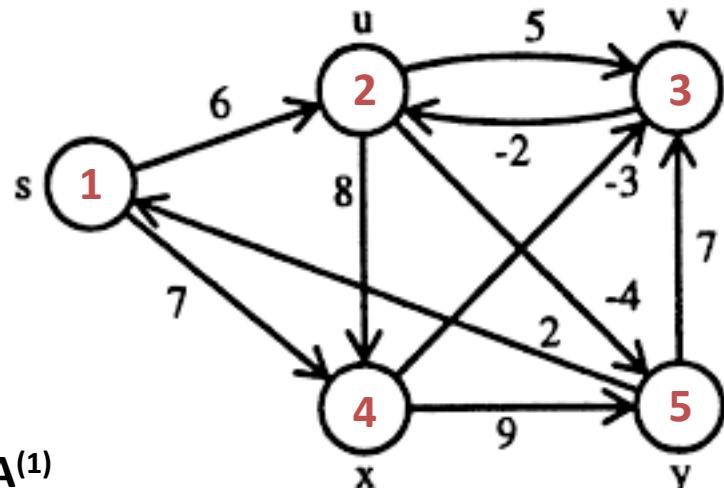
$A^{(5)}$

0	4	7	-2
-4	0	-2	0
-5	-3	0	0
2	4	6	9

$A =$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(2)}$

0	6	1	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-4
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-4
2	7	9	0	0

$A^{(4)}$

0	4	7	-2
Inf	0	5	-4
Inf	-2	0	-6
Inf	-5	-3	-9
2	4	6	9

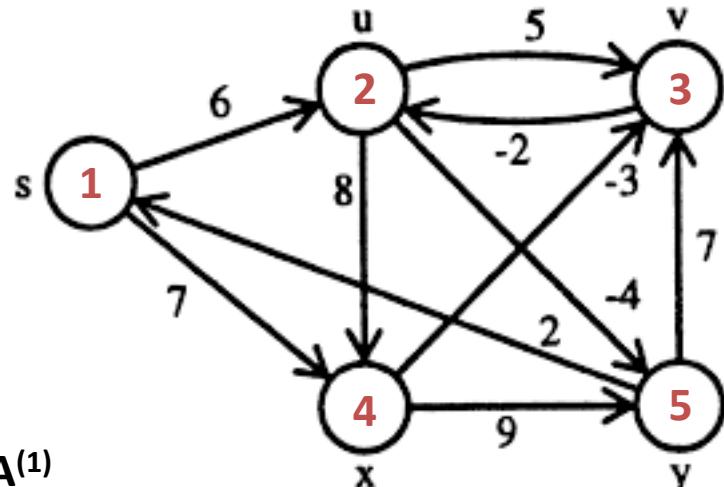
$A^{(5)}$

0	4	7	-2
-4	0	-2	-4
-5	-3	0	-6
2	4	6	9

$A =$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(2)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	7	9	0	0

$A^{(4)}$

0	4	7	-2
Inf	0	5	-4
Inf	-2	0	6
Inf	-5	-3	0
2	4	6	9

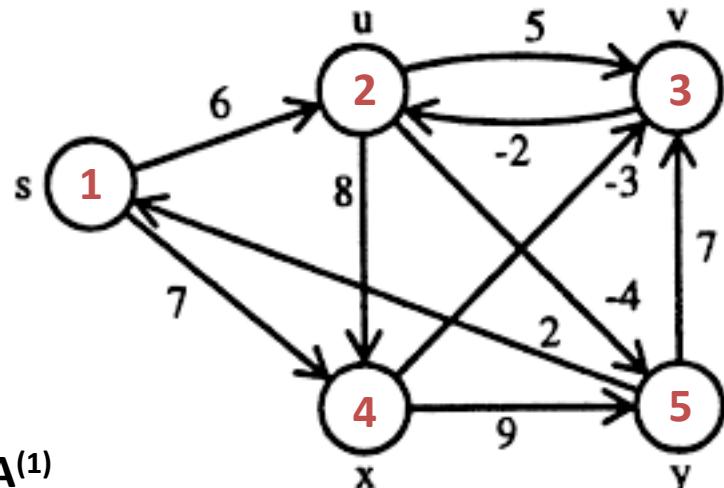
$A^{(5)}$

0	4	7	-2
-4	0	-2	0
-5	-3	0	-9
2	4	6	9

$A =$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(2)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	5	7	9	0

$A^{(4)}$

0	4	7	-2
Inf	0	5	-4
Inf	-2	0	6
Inf	-5	-3	0
2	4	6	9

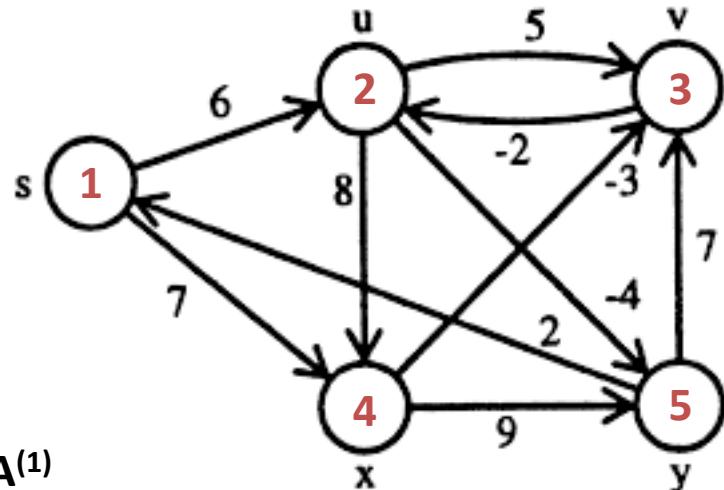
$A^{(5)}$

0	4	7	-2
-4	0	-2	-4
-5	-3	0	-6
2	4	6	9

$A =$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(2)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	5	7	9	0

$A^{(4)}$

0	2	4	7	-2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	4	6	9	0

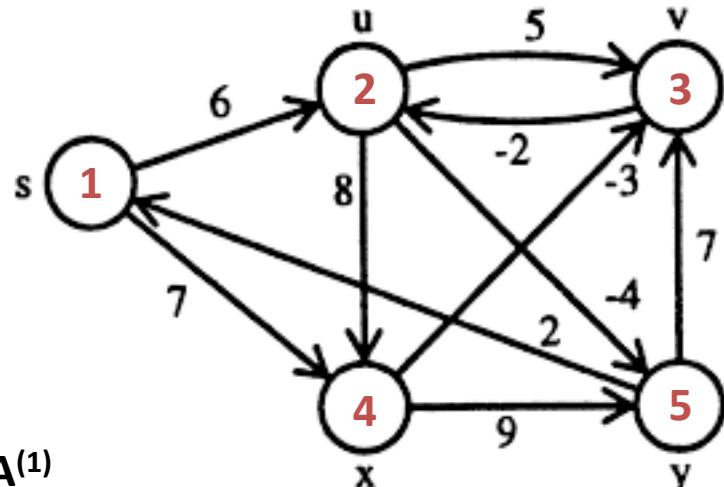
$A^{(5)}$

0				
-4	0			
-2	-2	0		
-5	-5	-3	0	
2	4	6	9	0

$A =$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	Inf	7	Inf	0

WARSHALL



$A^{(1)}$

0	6	Inf	7	Inf
Inf	0	5	8	-4
Inf	-2	0	Inf	Inf
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(2)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	Inf	-3	0	9
2	8	7	9	0

$A^{(3)}$

0	6	11	7	2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	5	7	9	0

$A^{(4)}$

0	2	4	7	-2
Inf	0	5	8	-4
Inf	-2	0	6	-6
Inf	-5	-3	0	-9
2	4	6	9	0

$A^{(5)}$

0	2	4	7	-2
-2	0	2	5	-4
-4	-2	0	3	-6
-7	-5	-3	0	-9
2	4	6	9	0

ORDONNANCEMENT

On désire planifier un ensemble de N tâches pour lesquels on dispose des contraintes de précédences.

Définir le graphe associé à ce problème

$G = (T, P)$ où

T est l'ensemble des tâches, et on a un arc (t_1, t_2) si t_1 doit être exécuté avant t_2

Quel est le critère principal que doit vérifier un tel graphe ?

Le graphe ne doit pas contenir de cycle, il doit donc être acyclique

La planification est-elle unique ? Pourquoi ?

Non. Une tâche qui peut être exécutée à un instant t peut l'être à tout instant $t+k$

Proposez un algorithme qui, étant donné un graphe de tâches, construit un planning possible des travaux.

Tri topologique du graphe de tâches (s'il y a un arc entre t_1 et t_2 , alors t_1 vient avant t_2).