INFORMATIQUE

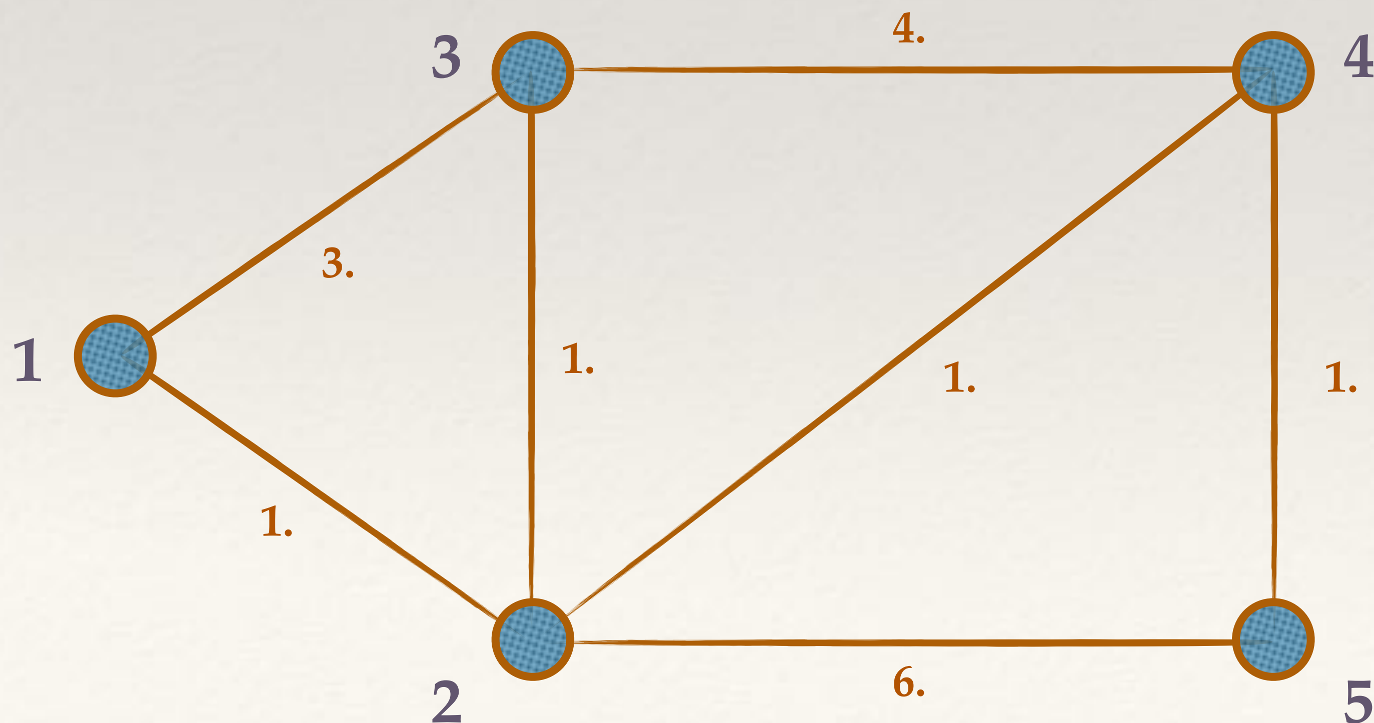
Introduction
aux
graphes

Présentation des graphes

On appelle graphe un ensemble de N - **Noeuds** reliés par M - **Cotés** :

—> une idée simple est de se figurer des villes reliées par des routes.
—> des terminaux informatiques dans un réseau, des routeurs.
—> des opérations d'achat/vente de titres sur des places boursières.
—> Le graphe orienté peut représenter un enchaînement de processus (automate)
Les noeuds sont des états, les cotés des transitions possibles vers un autre état.

Graphe non orienté :



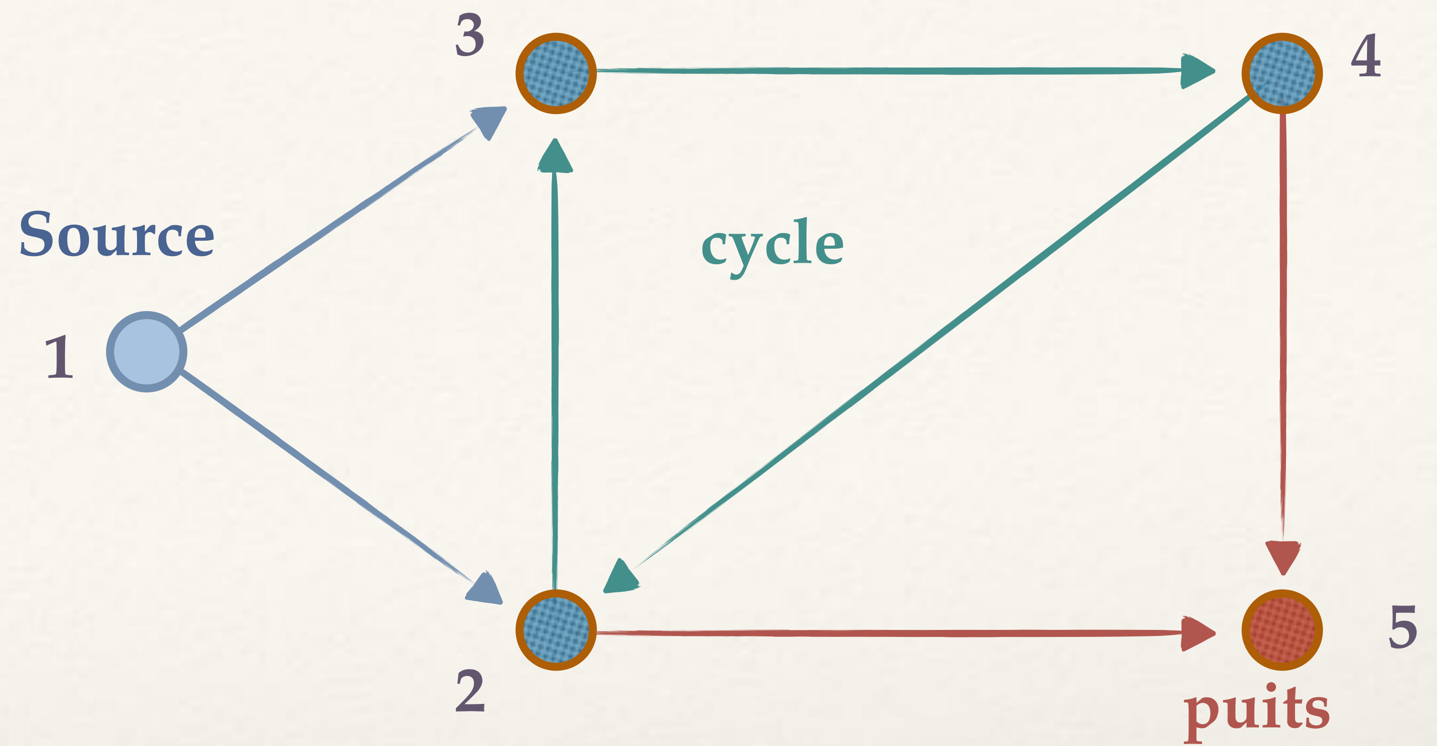
On peut vouloir (ou non) ajouter des distances :

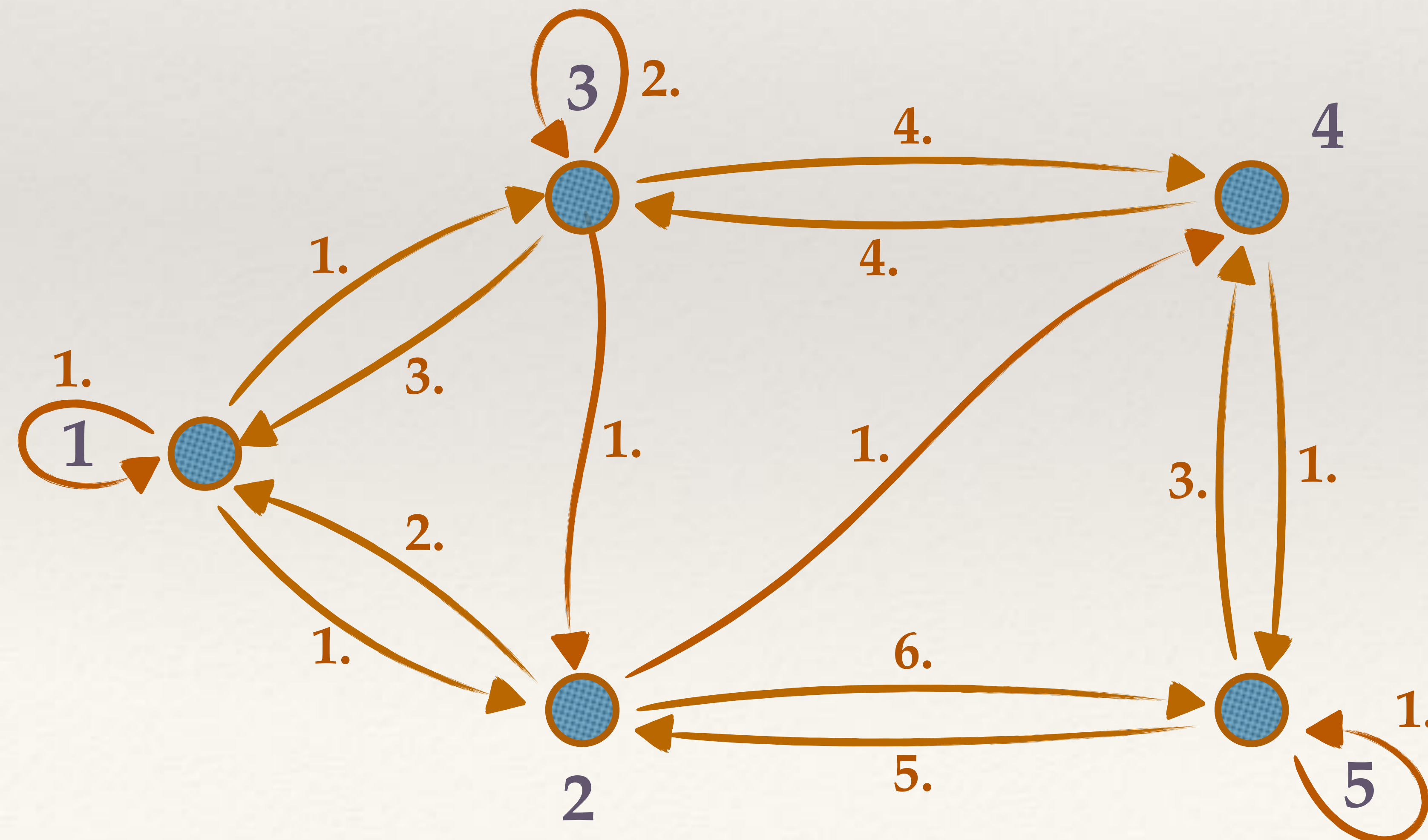
=> des villes reliées par des routes à double sens.

Graphe orienté :

On tient compte ici du sens de circulation dans la ville :

- on ne peut pas revenir à la source.
- on ne peut pas sortir du puits.





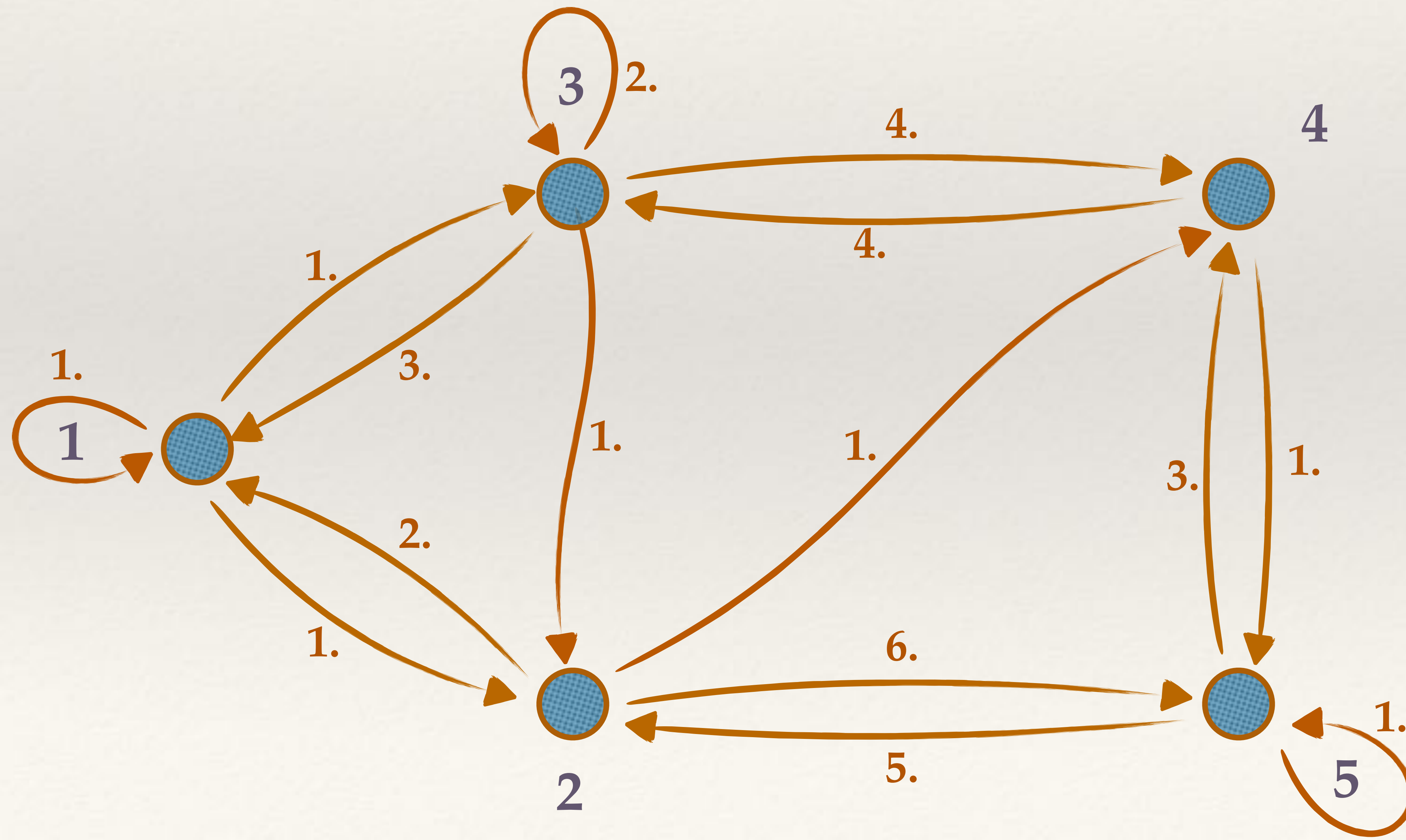
Graphe complexe !

- Automate
- parcours citadin complexe (axes rapides ou lents cf google maps)

Un **automate** représente un ensemble d'**états** (noeuds) on peut réaliser une **transition** (coté) d'un noeud vers un autre selon des critères d'évolution du système :

- **Jeu :** (ex : morpion, Go) [**états** : types équivalents de remplissage de la grille de jeu]
  (jeu de la vie : automate cellulaire) [**transitions** : selon le coup joué (boucle => passe son tour)]

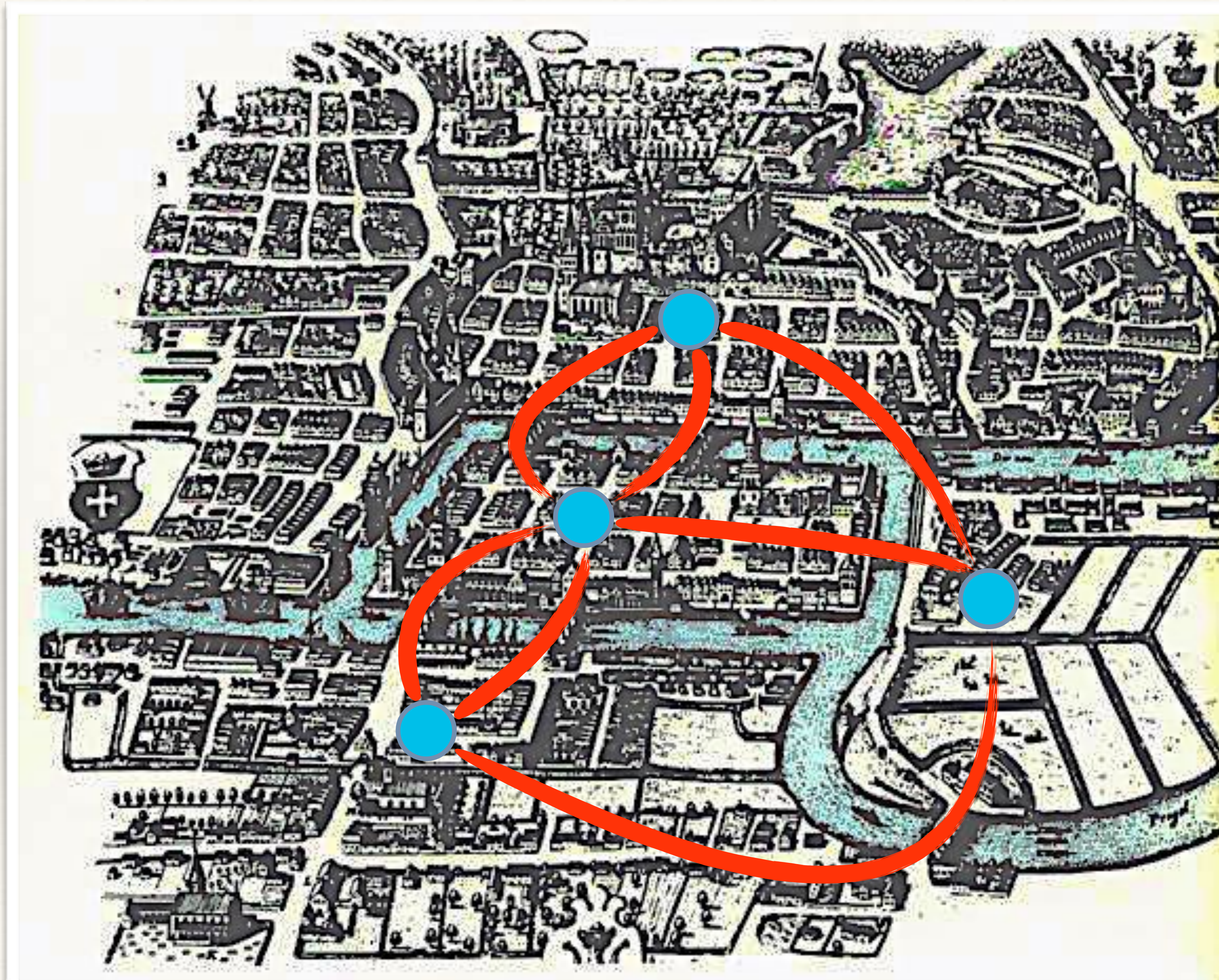
- **Protocole de communication réseau :** (ex : routeur / carte réseau)
  [**états** : envoie une requête /en attente (boucle) / reçoit confirmation etc …]
  [**transitions** : réagit à un message reçu/ un délai expiré]



A l'origine de la théorie des graphes : le Problème d'Euler

Peut-on emprunter les 7 ponts de la ville de Koenigsberg une fois et une seule ?

Sans jamais repasser sur le même pont ?

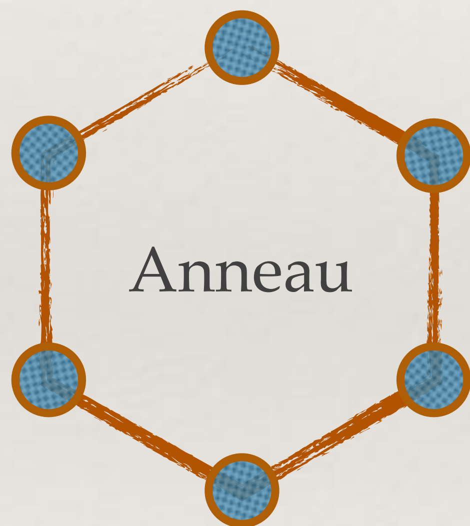


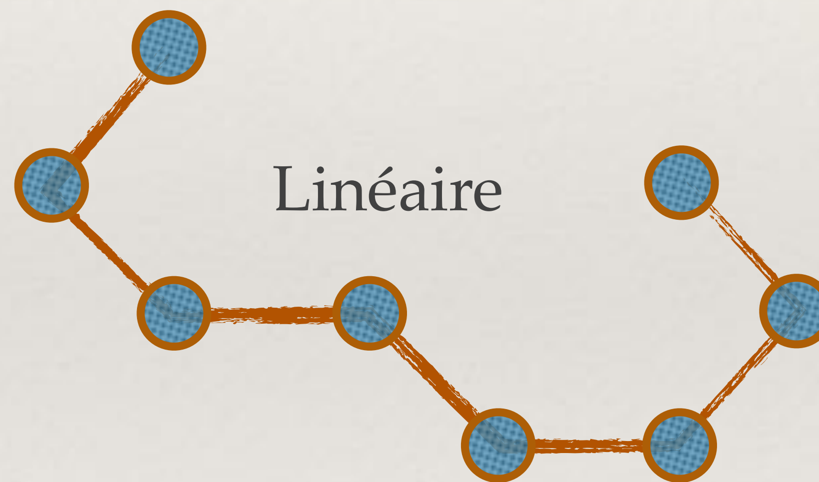
Quantification des graphes

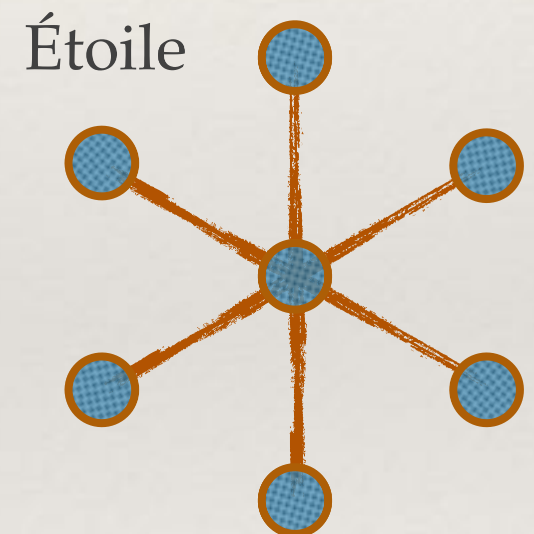
Soit **N le nombre de noeuds** et **M le nombre de cotés** :
La complexité des algorithmes concernant les graphes s'exprimeront en fonction de ces deux quantités N et M. Comment peut-on relier N et M ?

Rq : On ne considère ici que des graphes dits « connexes », c-à-d qu'il n'y a qu'une seule partition : Deux noeuds quelconques peuvent toujours être reliés par un chemin.

Graphes peu denses :







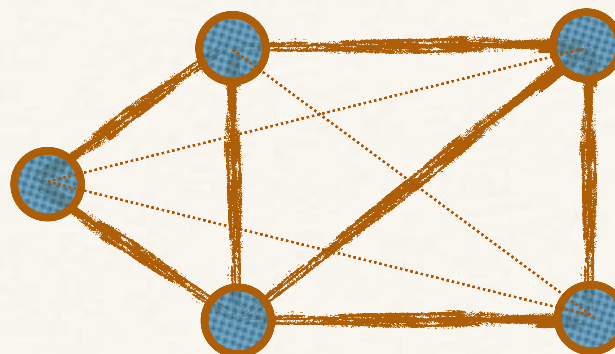
N = 6
M = 6

N = 8
M = 7

N = 7
M = 6

Si le graphe est connexe : on ne peut pas avoir **M < N-1** sans couper le graphe en deux !

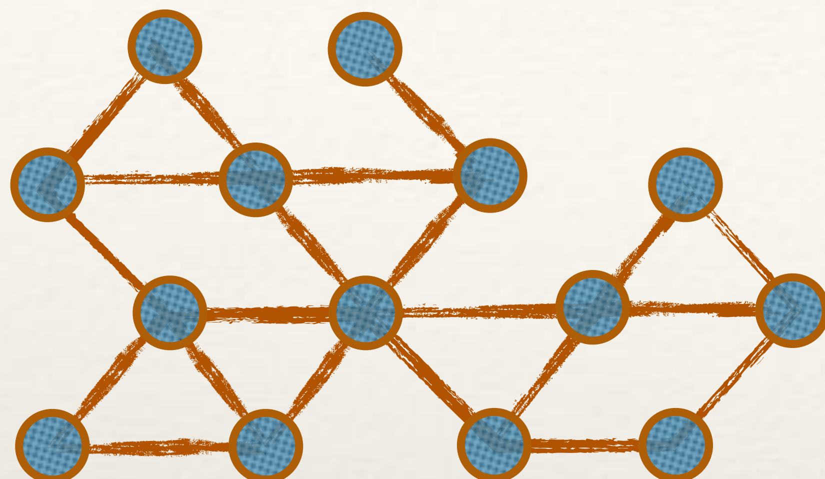
Graphes denses :

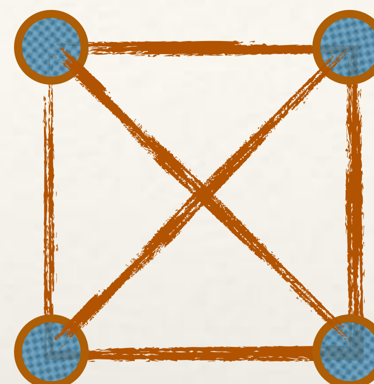


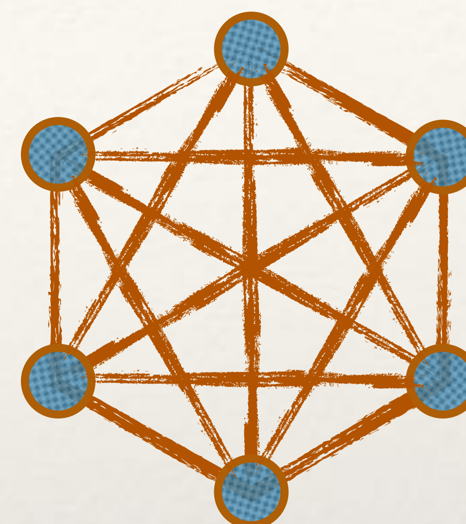
N = 5

M = 7

M < 11







N = 14

M = 21

M < 92

N = 4

M = 6

saturé !

N = 6

M = 15

saturé !

Exercice : montrer que le nombre M de cotés vaut au plus $\mathbf{M_{max} = N.(N-1)/2}$

On montre donc que selon la densité du graphe, le nombre de cotés vérifie :

$$N-1 \;\leq\; M \;\leq\; \frac{N(N-1)}{2}$$

Calcul de densité :

Soit M le nombre de cotés et N le nombre de noeuds :
On a **M_{max} = N.(N-1)/2**

On peut définir la densité comme :

$$\rho \equiv \frac{M}{M_{\max}} = \frac{2M}{N(N-1)}$$

On aura donc toujours :

$$\frac{2}{N} \leq \rho \leq 1$$

Dans le cas où N devient grand on imagine mal que chaque noeud soit connecté à une large fraction du réseau :

- Twitter / Facebook : on a rarement 1 milliard de followers !
- Un routeur n'est connecté qu'à ses voisins géographiques immédiats, même chose pour des villes ou un réseau de neurones.
- Pour un automate (type jeu de la vie) on ne prend en compte que les voisins

Dans bien des cas lorsque : $N \to \infty \quad \rho \to 0$

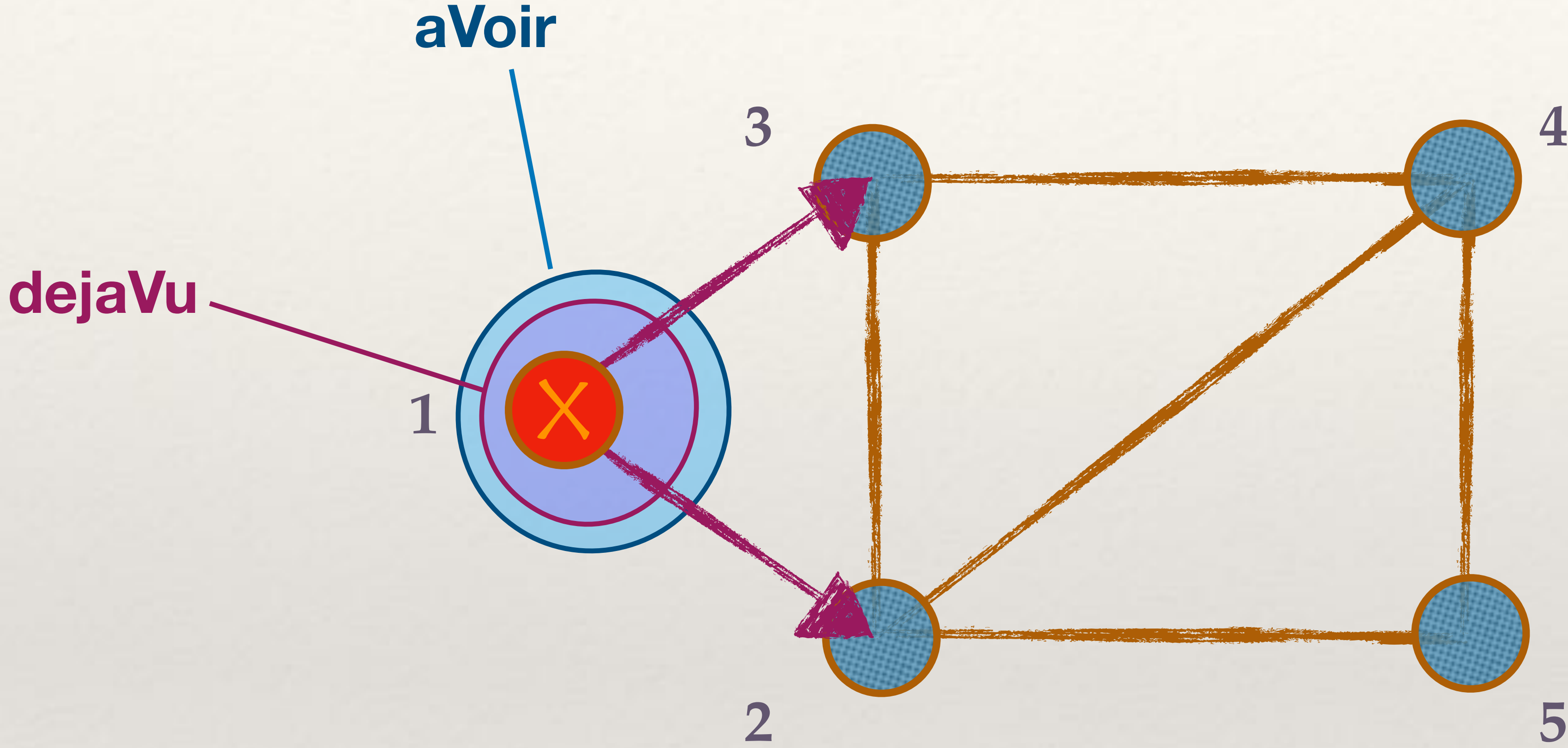
EXPLORATION DU GRAPHE

- **Parcours de graphe en largeur :** **Breadth First Search (BFS)**

- **Parcours de graphe en profondeur :** **Deep First Search (DFS)**

Parcours de graphe Breadth First Search (BFS) :

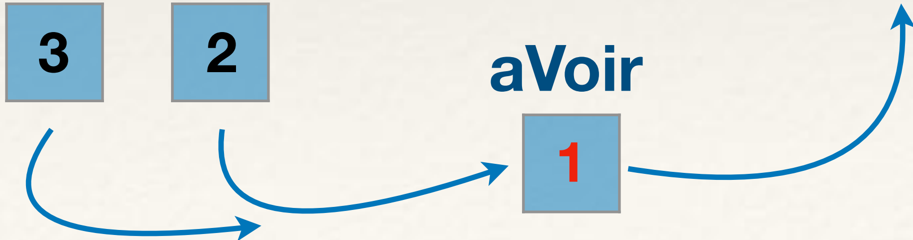
Conditions initiales : Noeud de départ et ses connections



Affiche :

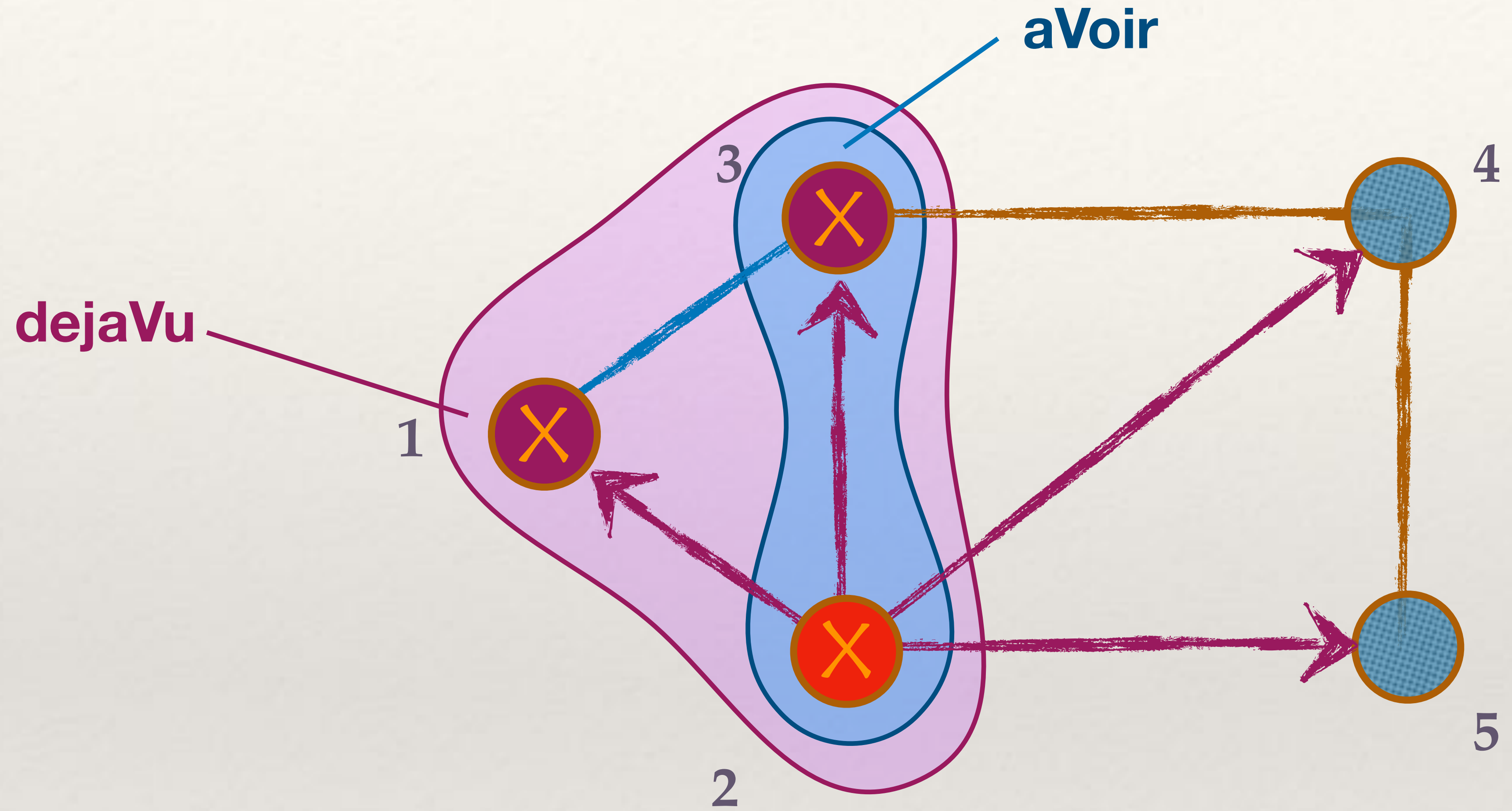
dejaVu

Noeud	1	2	3	4	5
Marqué	1	0	0	0	0

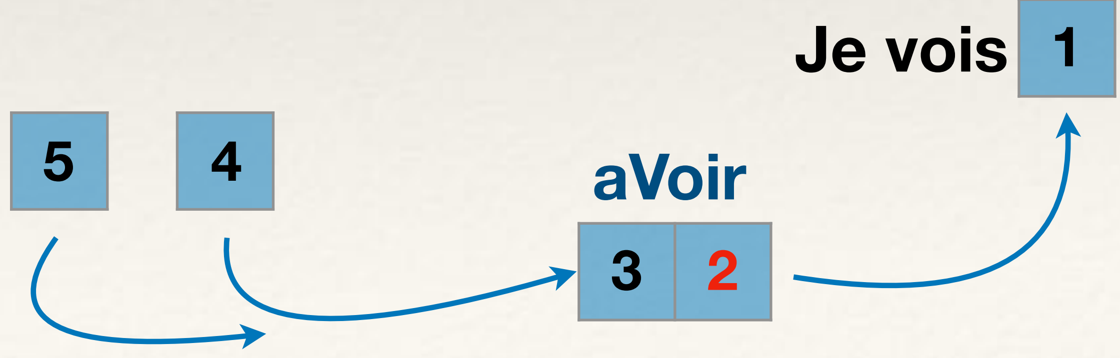


Parcours de graphe Breadth First Search (BFS) :

Propagation : - Le dernier noeud sort de la queue. On prend le suivant :
- Les noeuds voisins rentrent dans la queue.



Affiche :



dejaVu

Noeud	1	2	3	4	5
Marqué	1	1	1	0	0

Parcours de graphe Breadth First Search (BFS) :

Propagation : - Le dernier noeud sort de la queue. On prend le suivant :
- Les noeuds voisins rentrent dans la queue.

aVoir

dejaVu

3

4

1

2

5

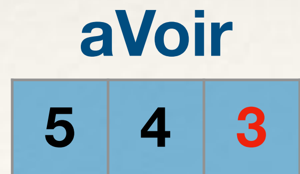
Affiche :

Je vois 2

dejaVu

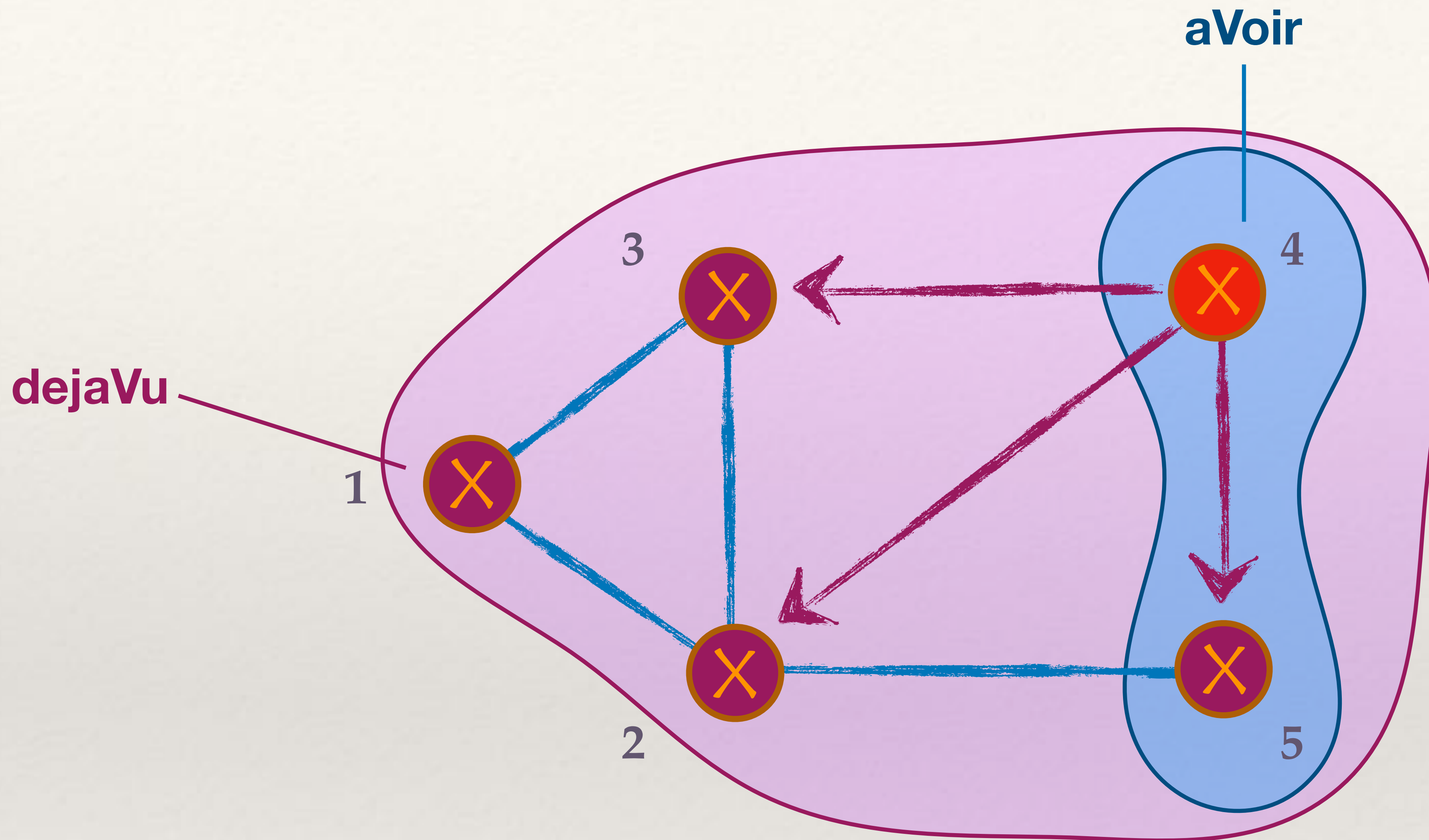
Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

aVoir



Parcours de graphe Breadth First Search (BFS) :

Propagation : - Le dernier noeud sort de la queue. On prend le suivant :
- Les noeuds voisins rentrent dans la queue.



Affiche :

Je vois 3

aVoir

5	4

dejaVu

Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

Parcours de graphe Breadth First Search (BFS) :

Propagation : - Le dernier noeud sort de la queue. On prend le suivant :
- Les noeuds voisins rentrent dans la queue.

dejaVu

3 4 1 2 5

aVoir

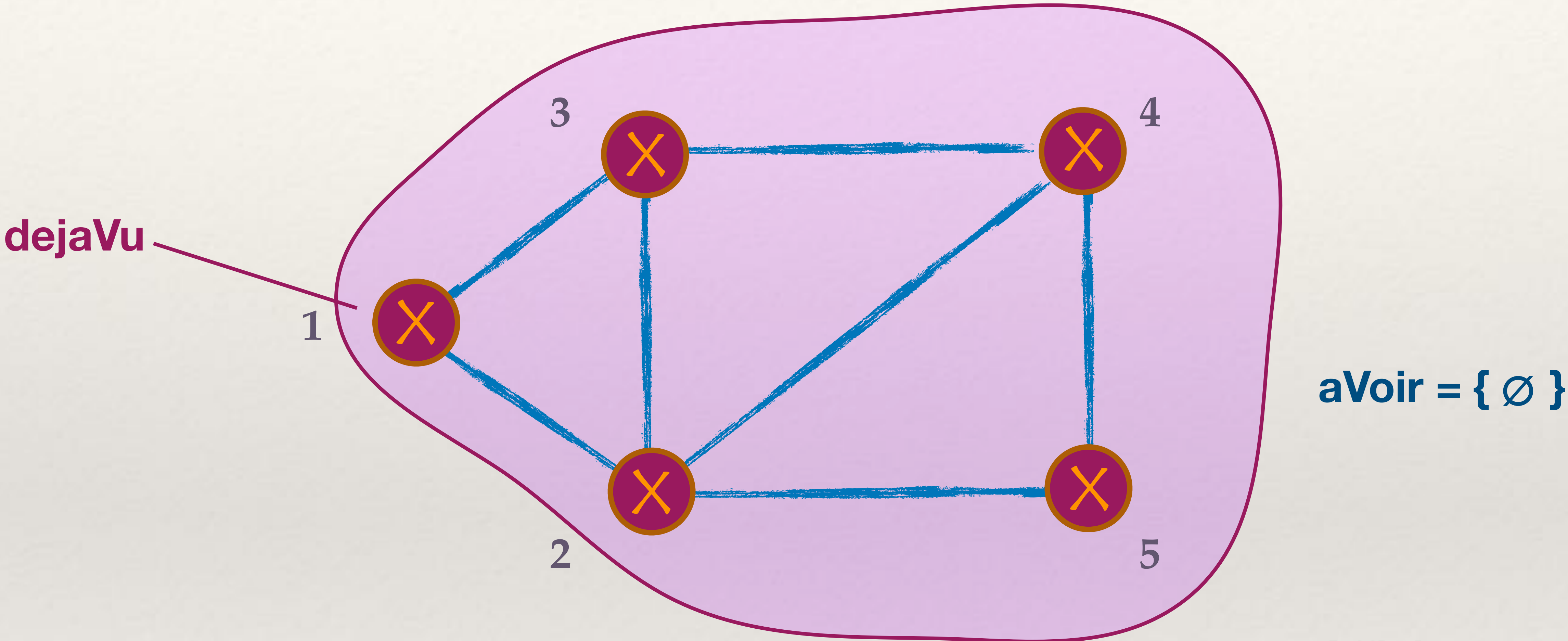
Affiche :

dejaVu

Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

Parcours de graphe Breadth First Search (BFS) :

Terminaison : **Quand la queue est vide, l'algorithme prend fin :**
- tous les noeuds ont été visités.
- tous les cotés aussi.



aVoir = { ∅ }

Affiche :

Je vois 5

aVoir

∅

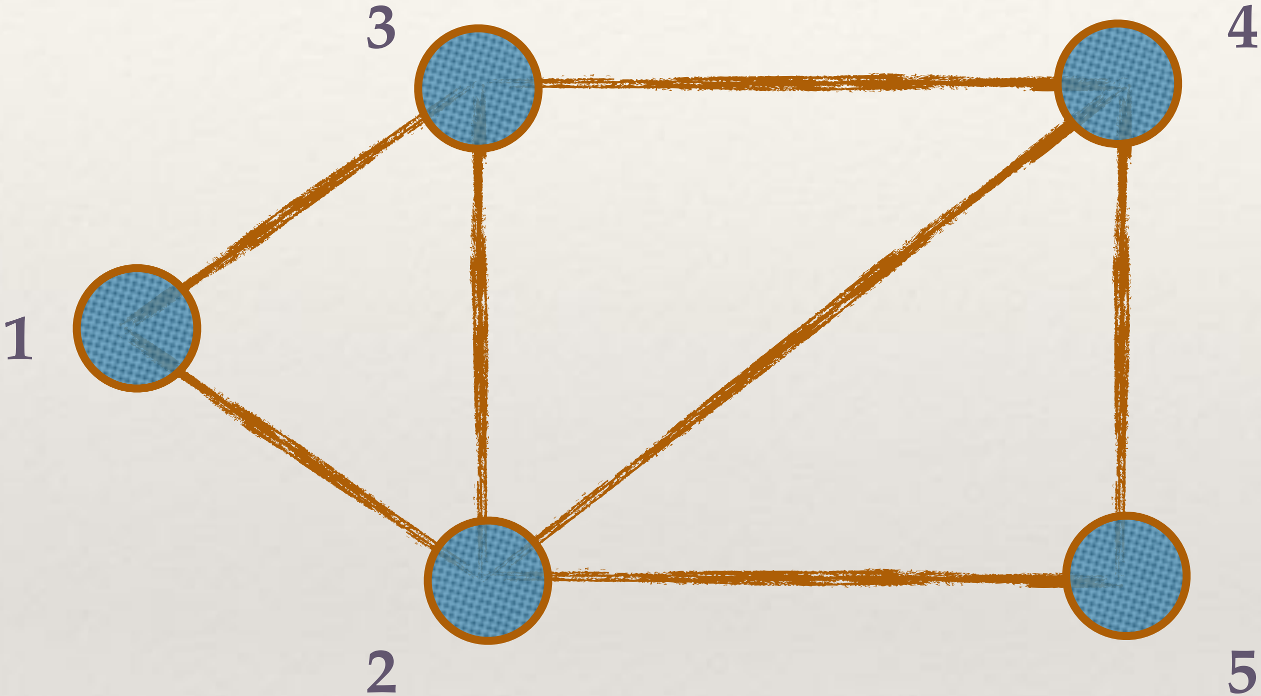
dejaVu

Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

Parcours de graphe Deep First Search (DFS) :

Conditions initiales : **Noeud de départ**

dejaVu = { ∅ }



Affiche :

Je vois

dejaVu

Noeud	1	2	3	4	5
Marqué	0	0	0	0	0

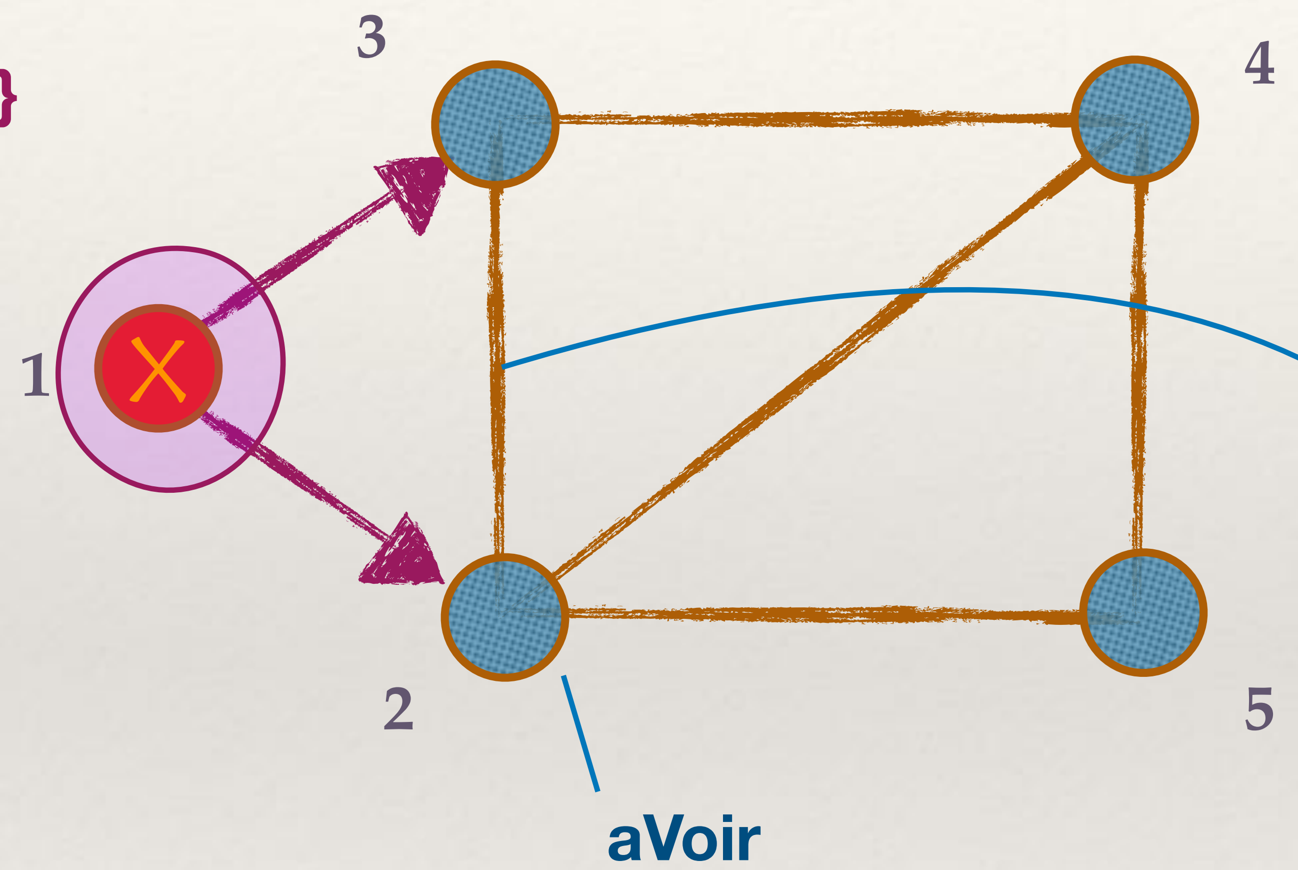
aVoir

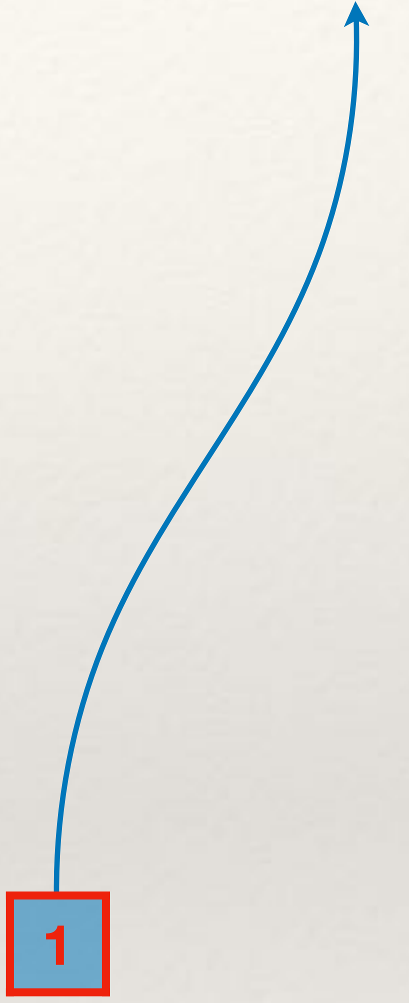
Parcours de graphe Deep First Search (DFS) :

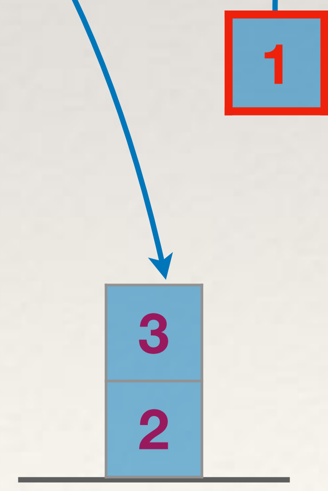
Affiche :

Je vois 1

dejaVu = { ∅ }



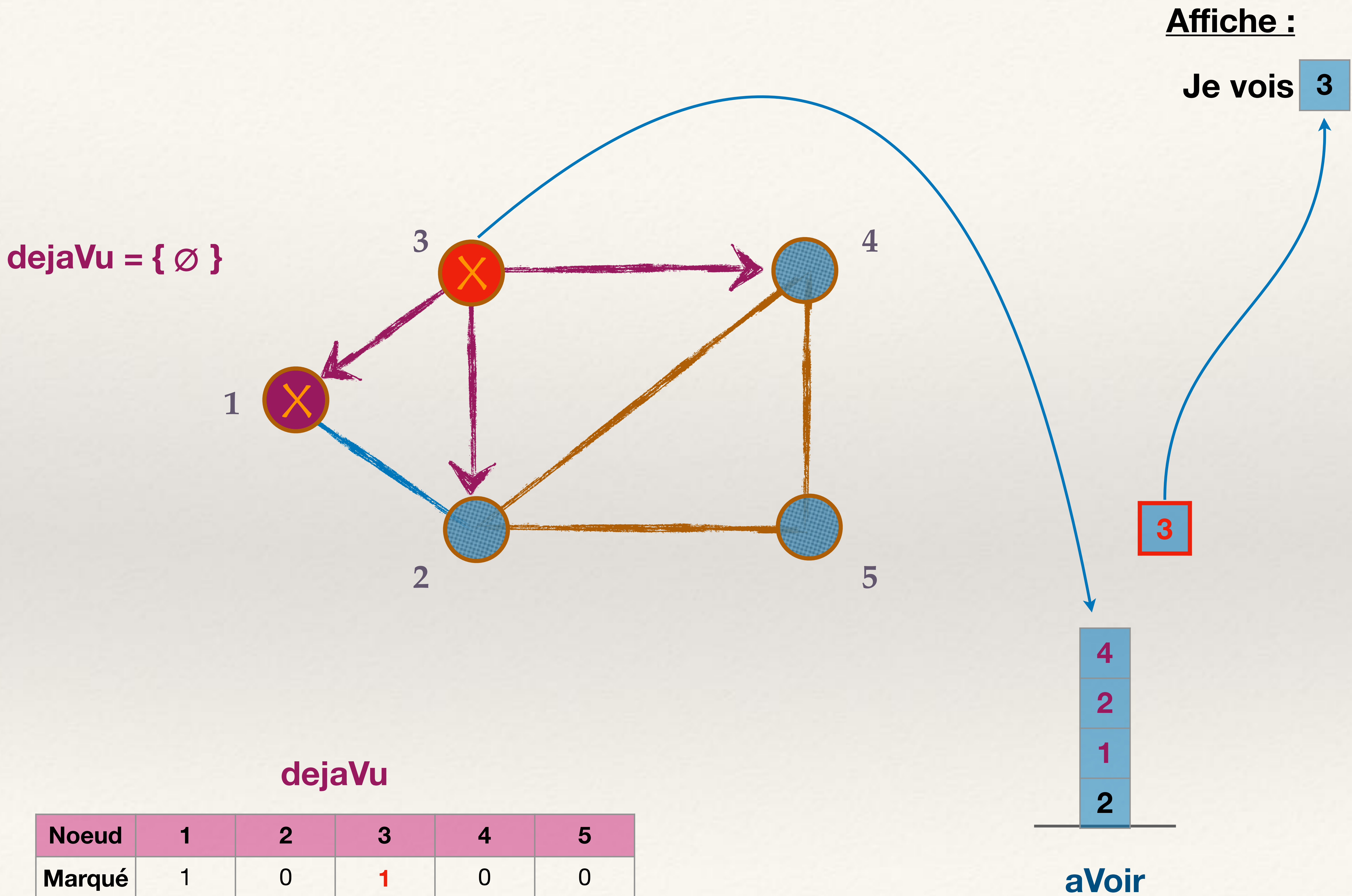




dejaVu

Noeud	1	2	3	4	5
Marqué	1	0	0	0	0

Parcours de graphe Deep First Search (DFS) :



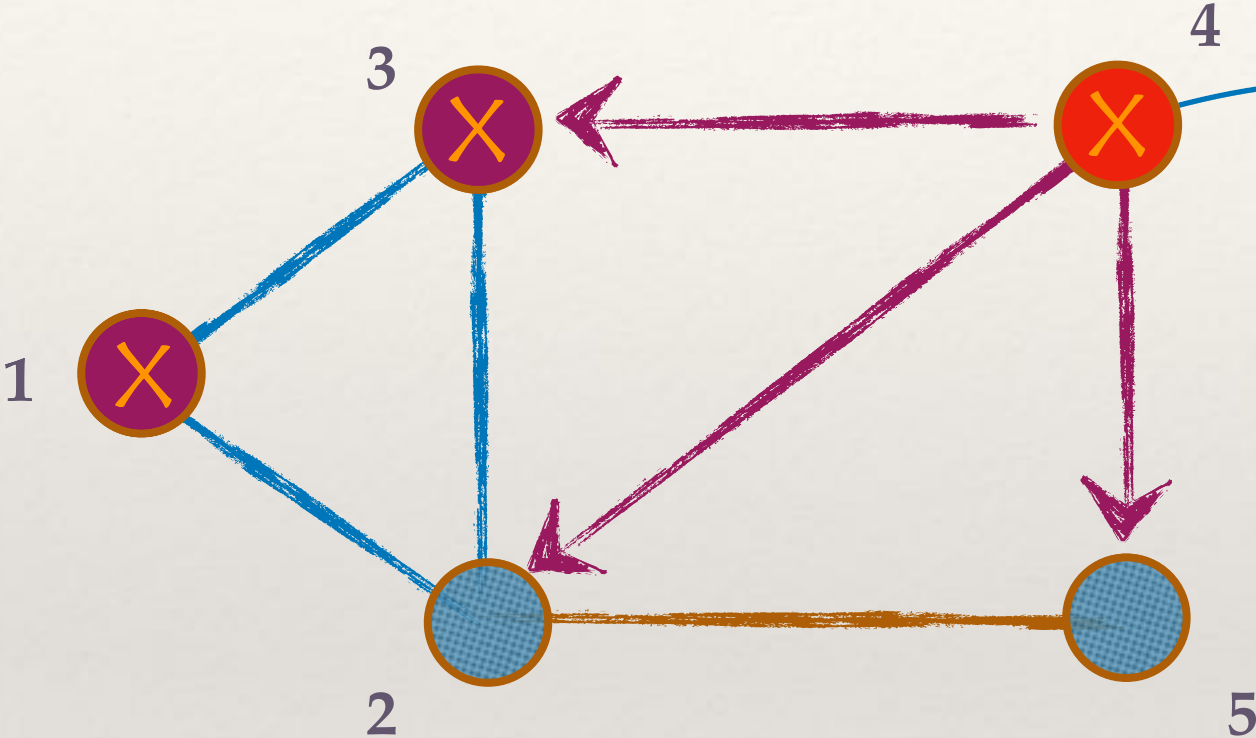
Noeud	1	2	3	4	5
Marqué	1	0	1	0	0

Parcours de graphe Deep First Search (DFS) :

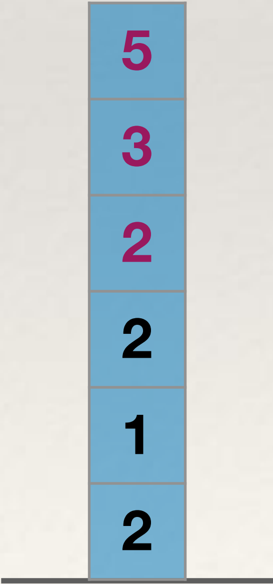
Affiche :

Je vois 4

dejaVu = { ∅ }



4



aVoir

dejaVu

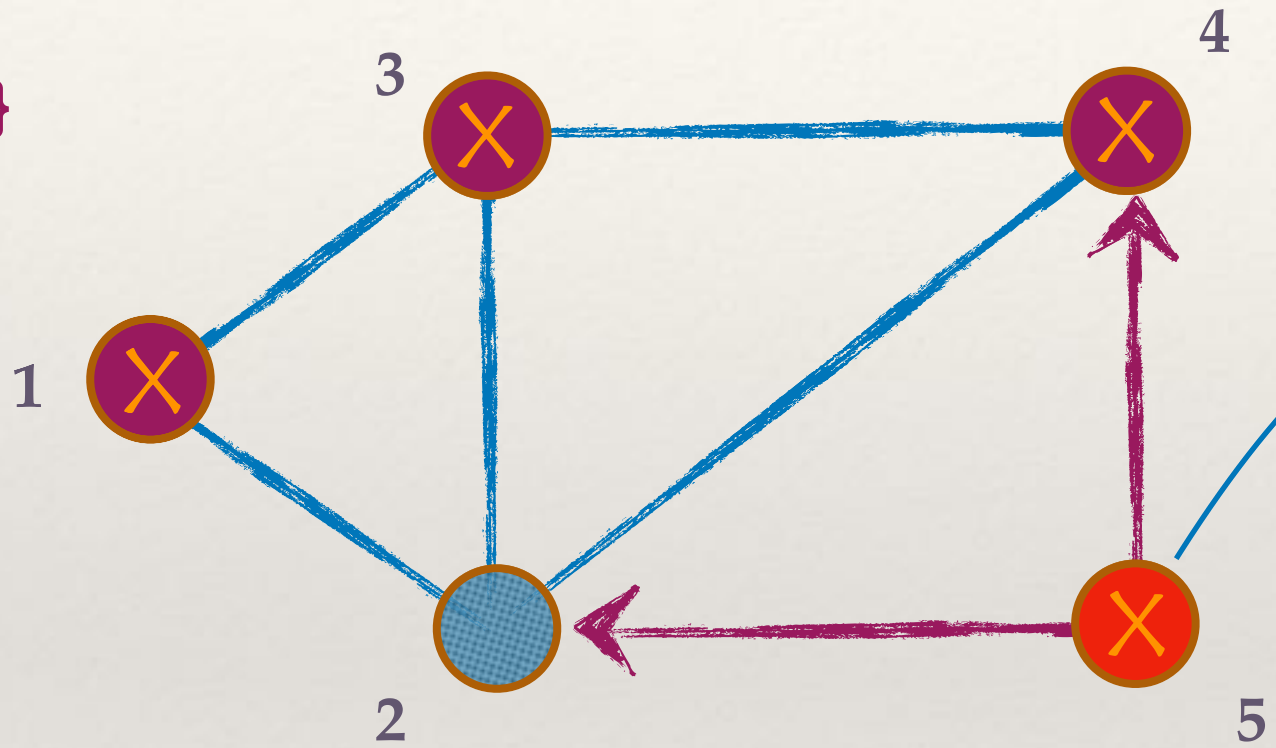
Noeud	1	2	3	4	5
Marqué	1	0	1	1	0

Parcours de graphe Deep First Search (DFS) :

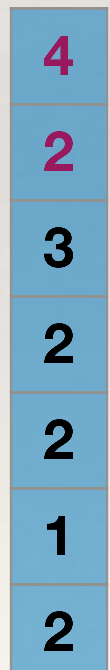
Affiche :

Je vois 5

dejaVu = { ∅ }



5

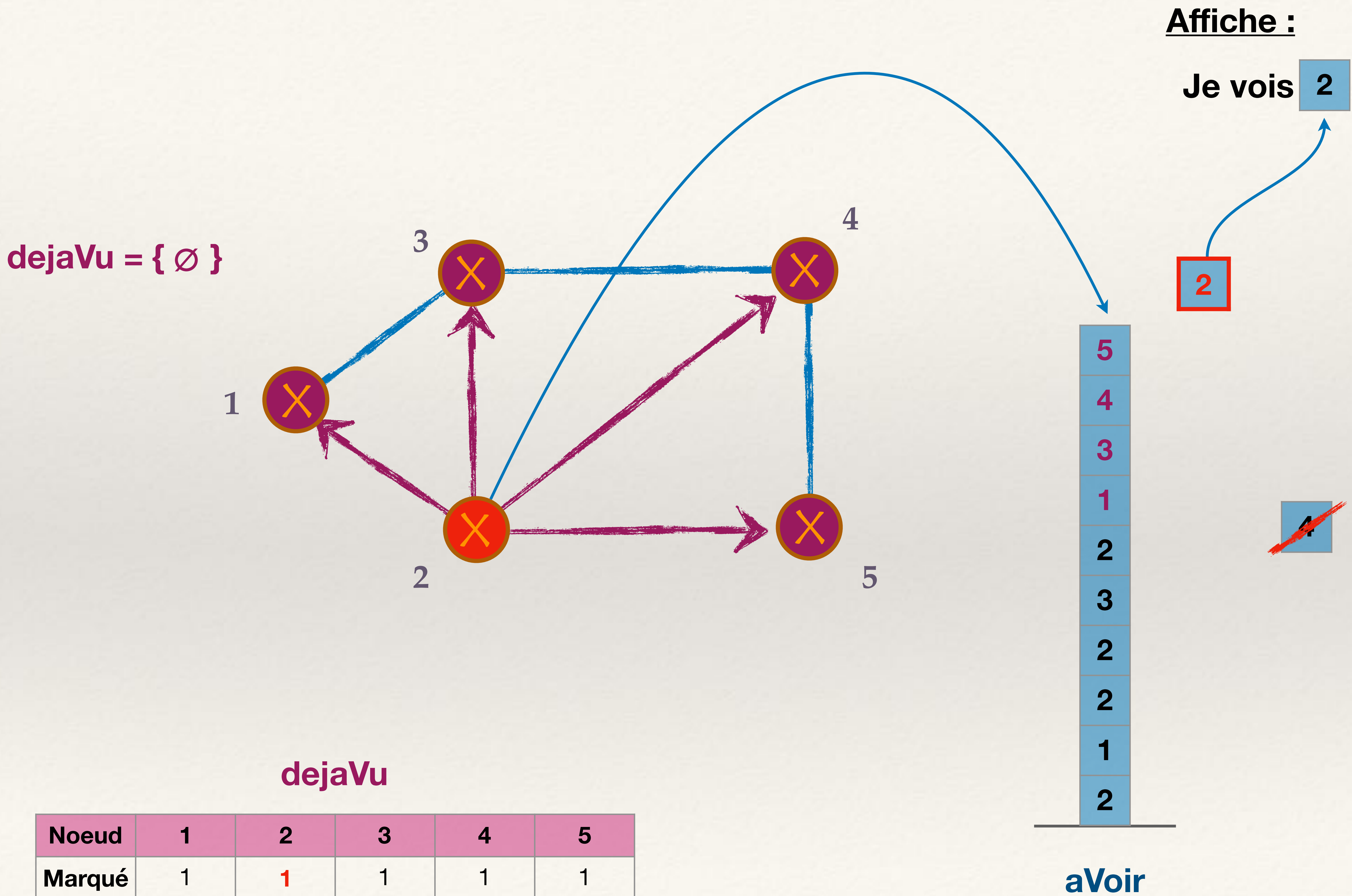


aVoir

dejaVu

Noeud	1	2	3	4	5
Marqué	1	0	1	1	1

Parcours de graphe Deep First Search (DFS) :



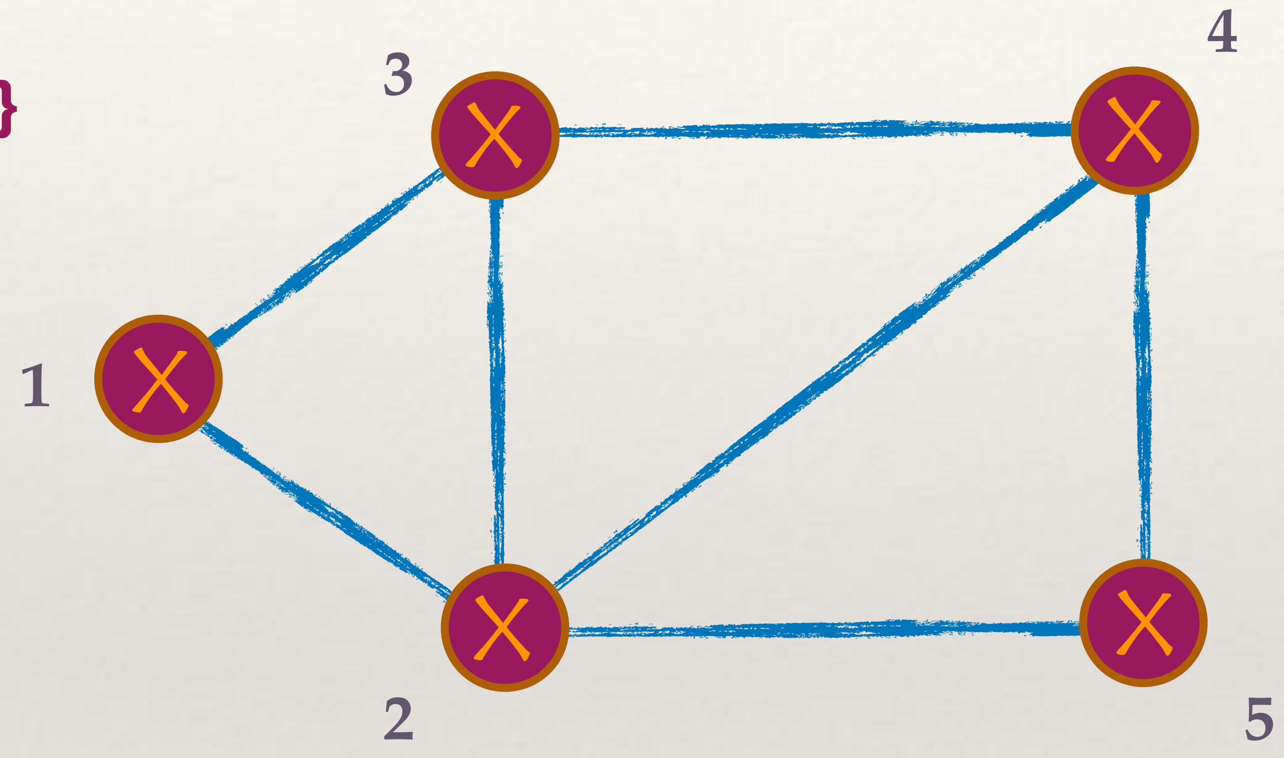
Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

Parcours de graphe Deep First Search (DFS) :

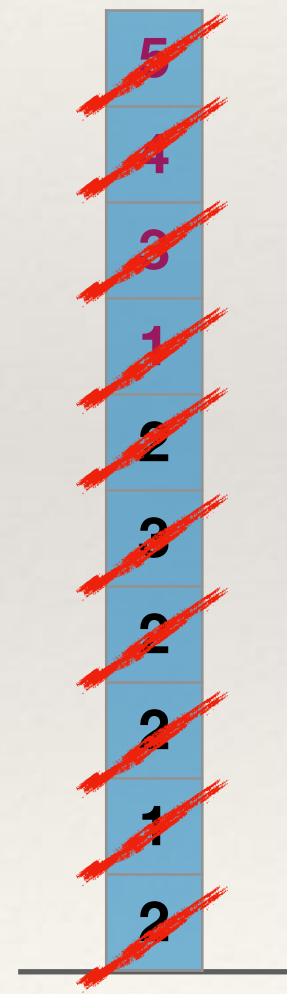
Affiche :

Je vois 2

dejaVu = { ∅ }



2



STOP

aVoir

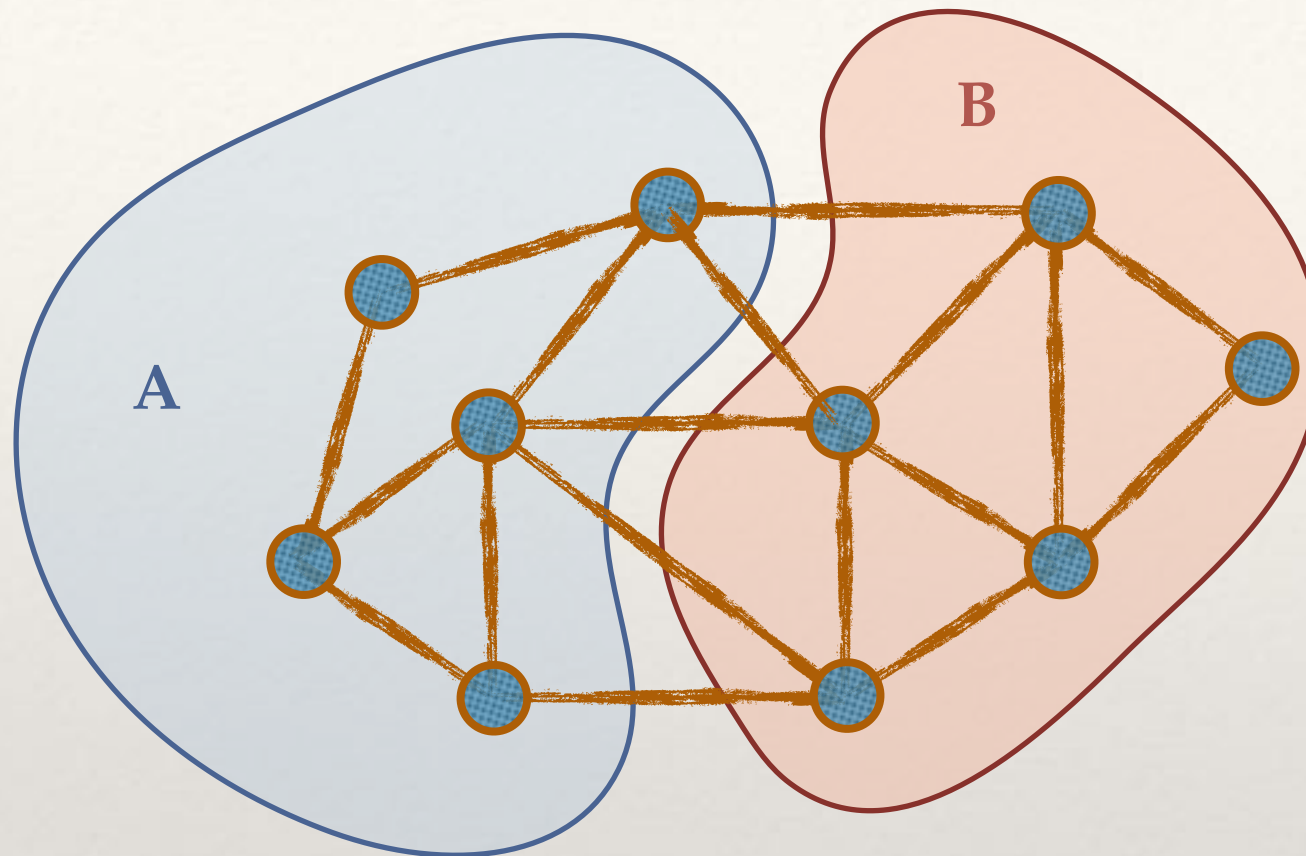
dejaVu

Noeud	1	2	3	4	5
Marqué	1	1	1	1	1

Notion de coupe :

On appelle coupe une séparation des noeuds du graphe en deux catégories :

A - Noeuds à l'intérieur de la coupe **B - Noeuds à l'extérieur de la coupe**



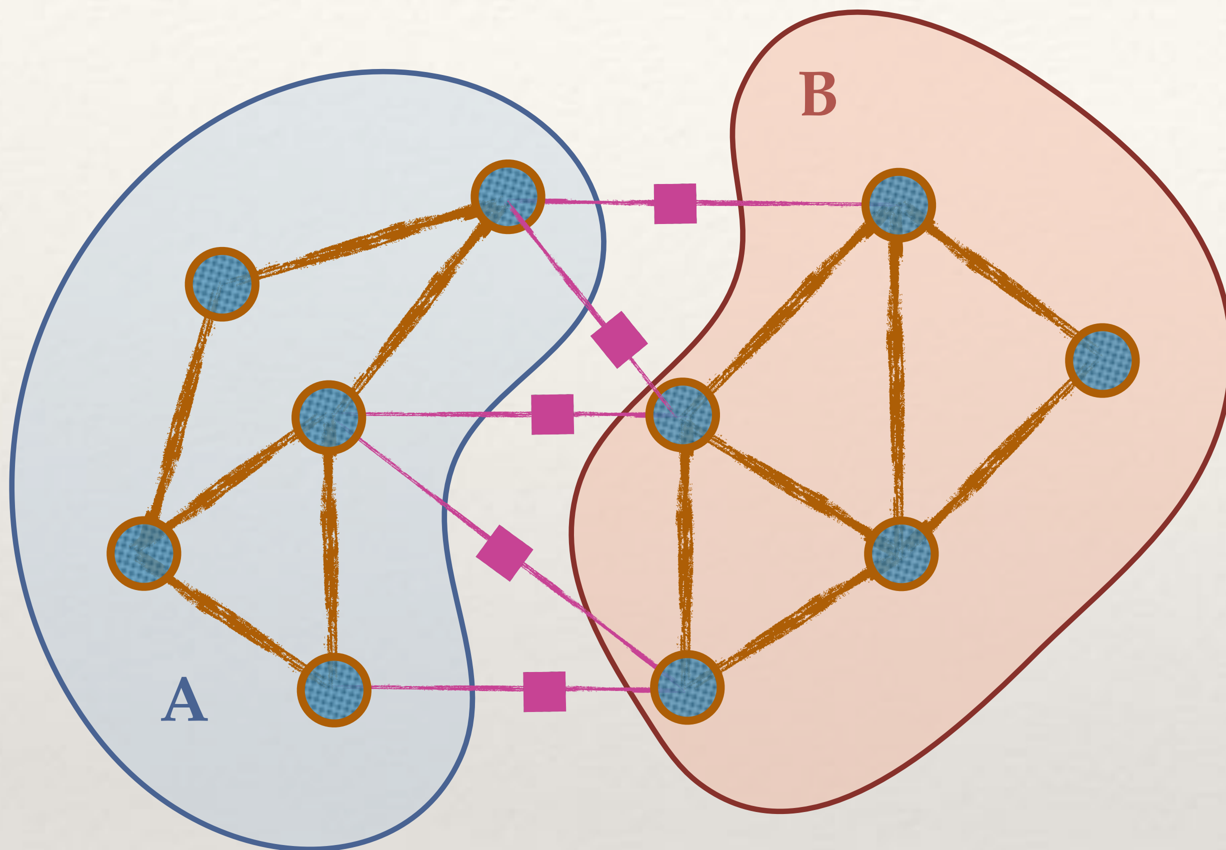
Exercice :

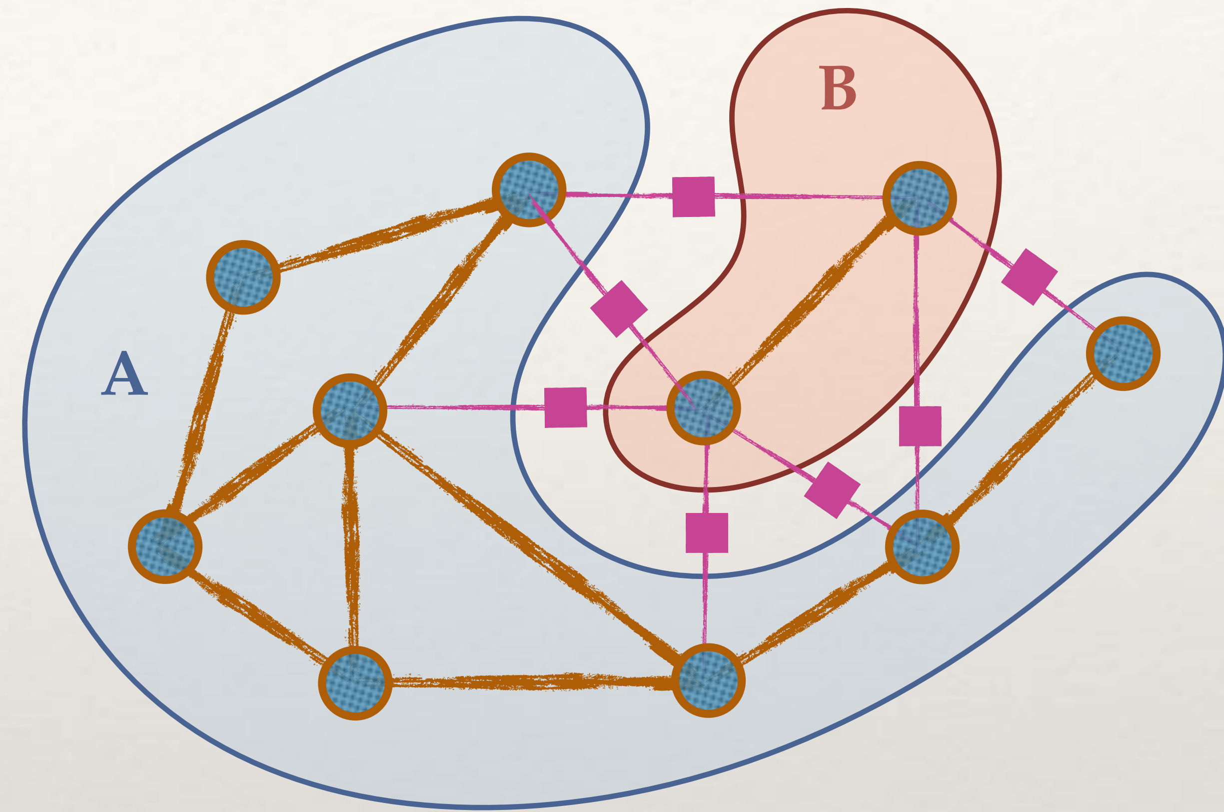
- Soit un graphe de N noeuds et M cotés, combien de coupes différentes sont possibles ?
- Que conclure quant à la complexité d'un algorithme naïf concernant les coupes ?

Notion de coupe :

On appelle coupe une séparation des noeuds du graphe en deux catégories :

A - Noeuds à l'intérieur de la coupe **B - Noeuds à l'extérieur de la coupe**



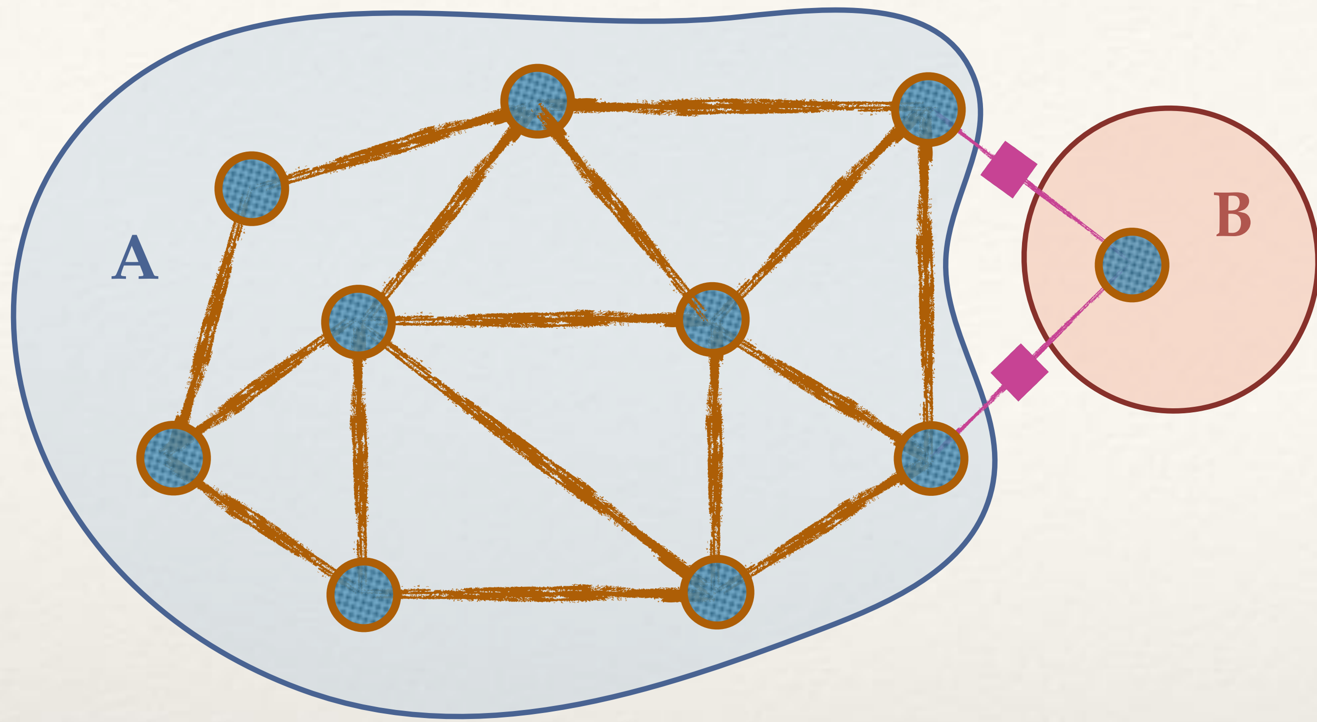


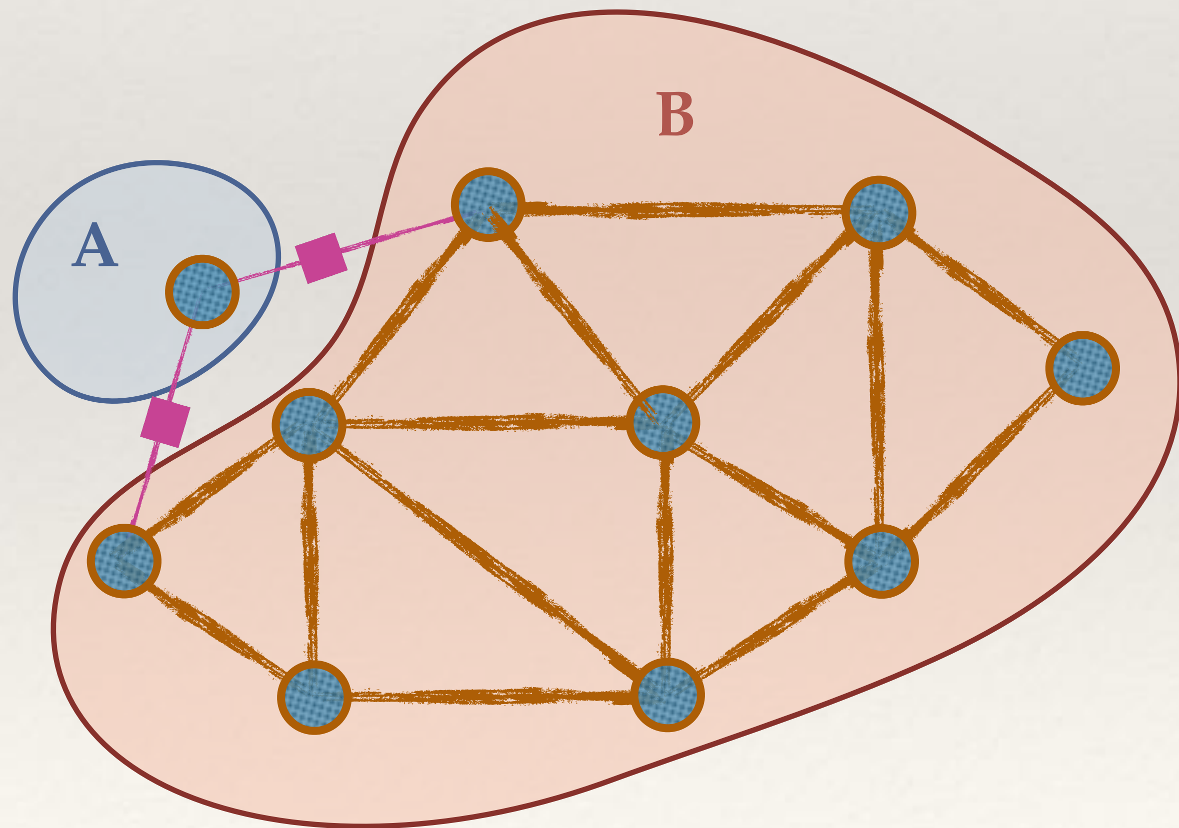
Exercice : **Guerre froide !**

On veut couper les communications terrestres entre l'est et l'ouest :
soit rompre les connections avec un minimum de bombardements.

=> On cherche la « **coupe minimale** » !

Coupes minimales



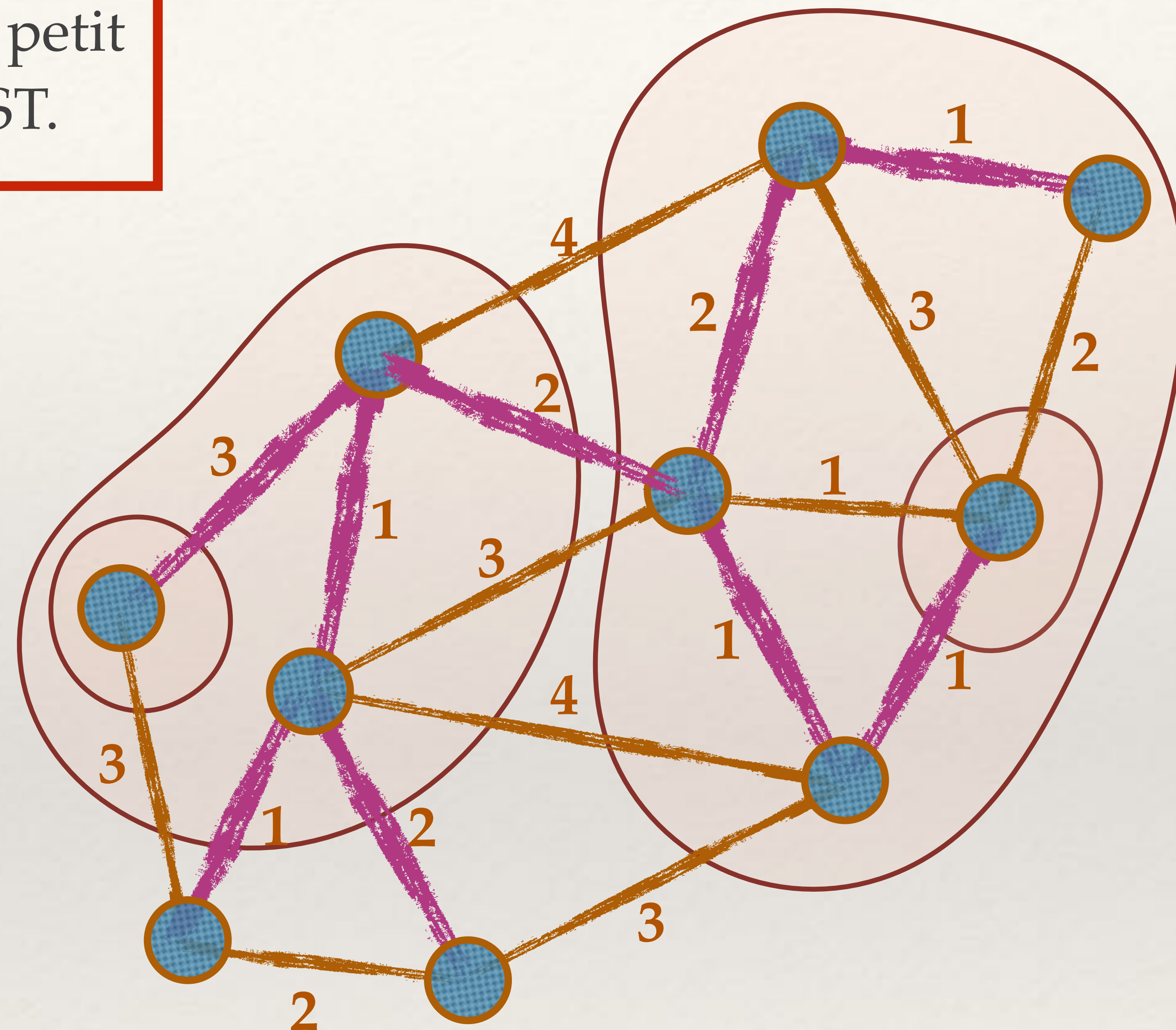


Application :   Minimum Spanning Tree

« Arbre couvrant de poids minimal »

Propriété de « coupe » :

Quelle que soit une coupe d'un graphe, le plus petit des cotés sortants de la coupe appartient au MST.



Algorithme de Prim :

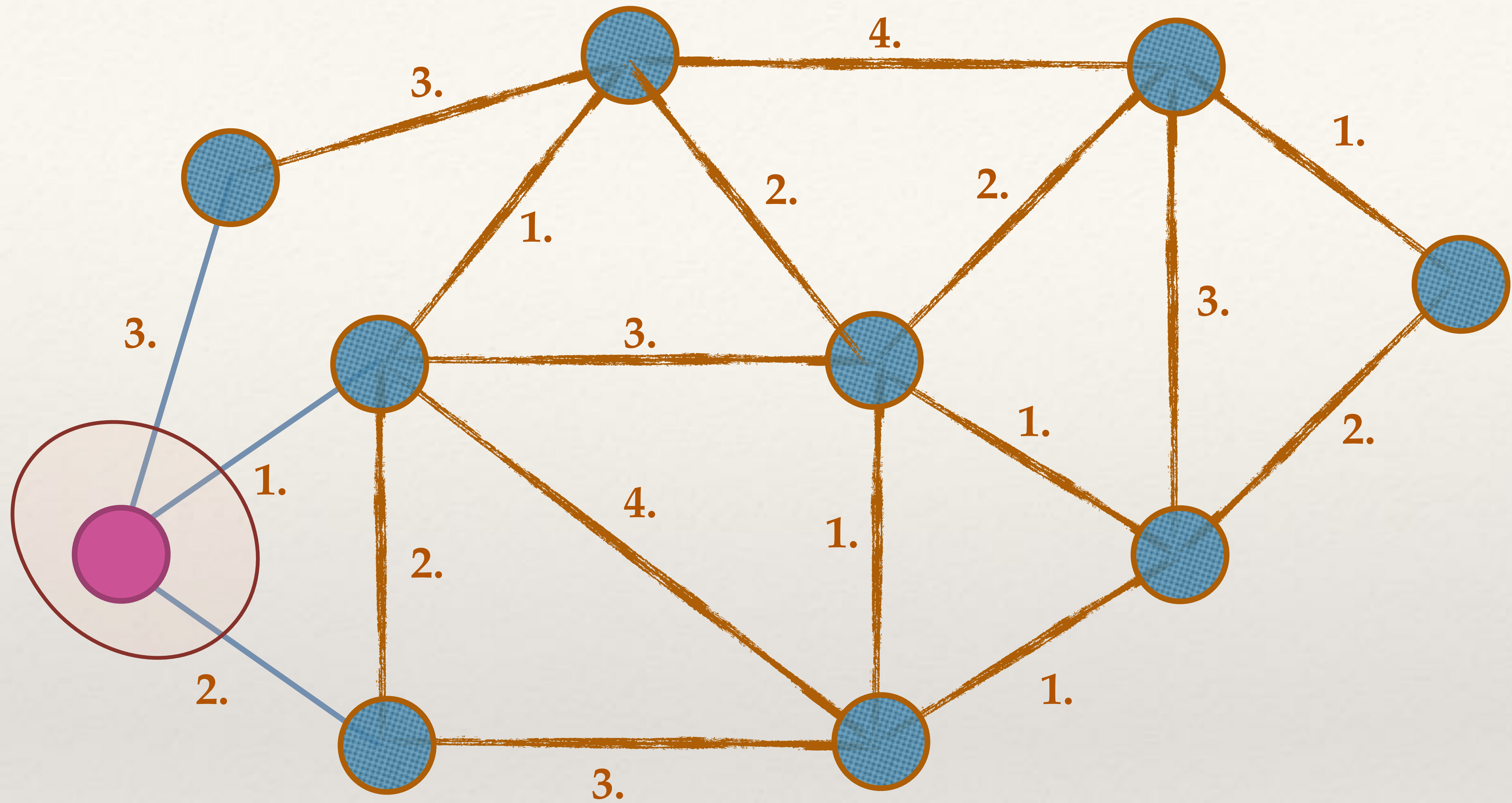
1 - On part d'une coupe à 1 noeud (qcq.)
  et ses connections sortantes stockées.

2 - Pour toute coupe [itération] on choisit
  le noeud extérieur à la coupe dont le
  coté sortant est le plus petit. [glouton]

3 - On met à jour les cotés sortants
=> supprime les cotés internes à la coupe
=> ajoute les connexions du nouveau noeud

Rq : Plusieurs solutions sont possibles lorsqu'il y a des longueurs de cotés sont identiques.

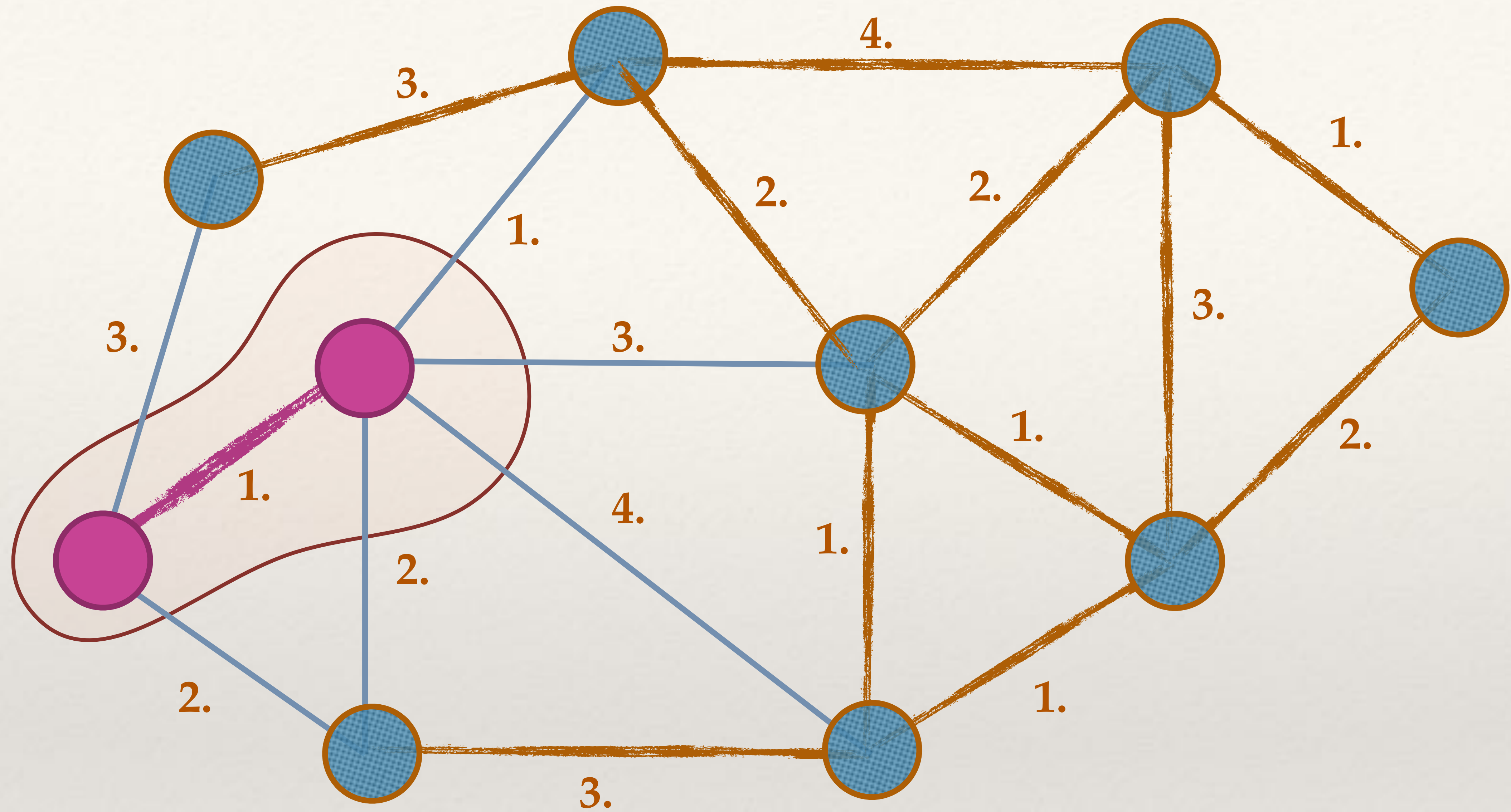
Application : Minimum Spanning Tree



Idée : Le 1er noeud devra de toute façon être relié au reste :
par l'un des cotés sortant de la coupe => quel coté choisir ?

L'ensemble des cotés « frontières » sont stockés dans une liste ou mieux une queue prioritaire [classée à partir du coté le plus petit].

Démo : Minimum Spanning Tree

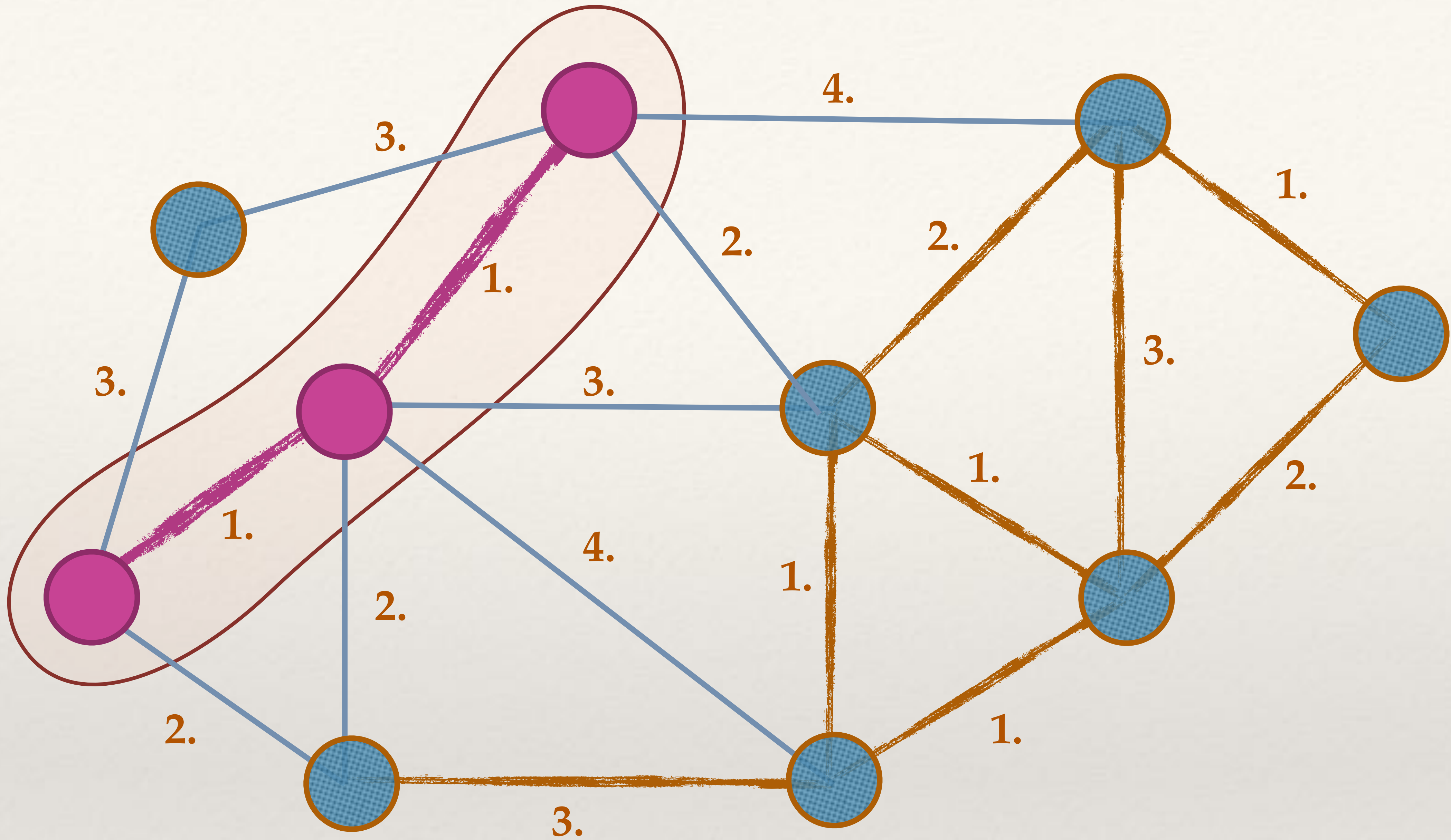


On choisit le noeud qui est le plus proche du MST actuel !

On met à jour les frontières de la coupe : ce sont les liens en bleus sur le schéma

- en rose un lien interne disparait
- en bleu apparaissent de nouveaux cotés sortant

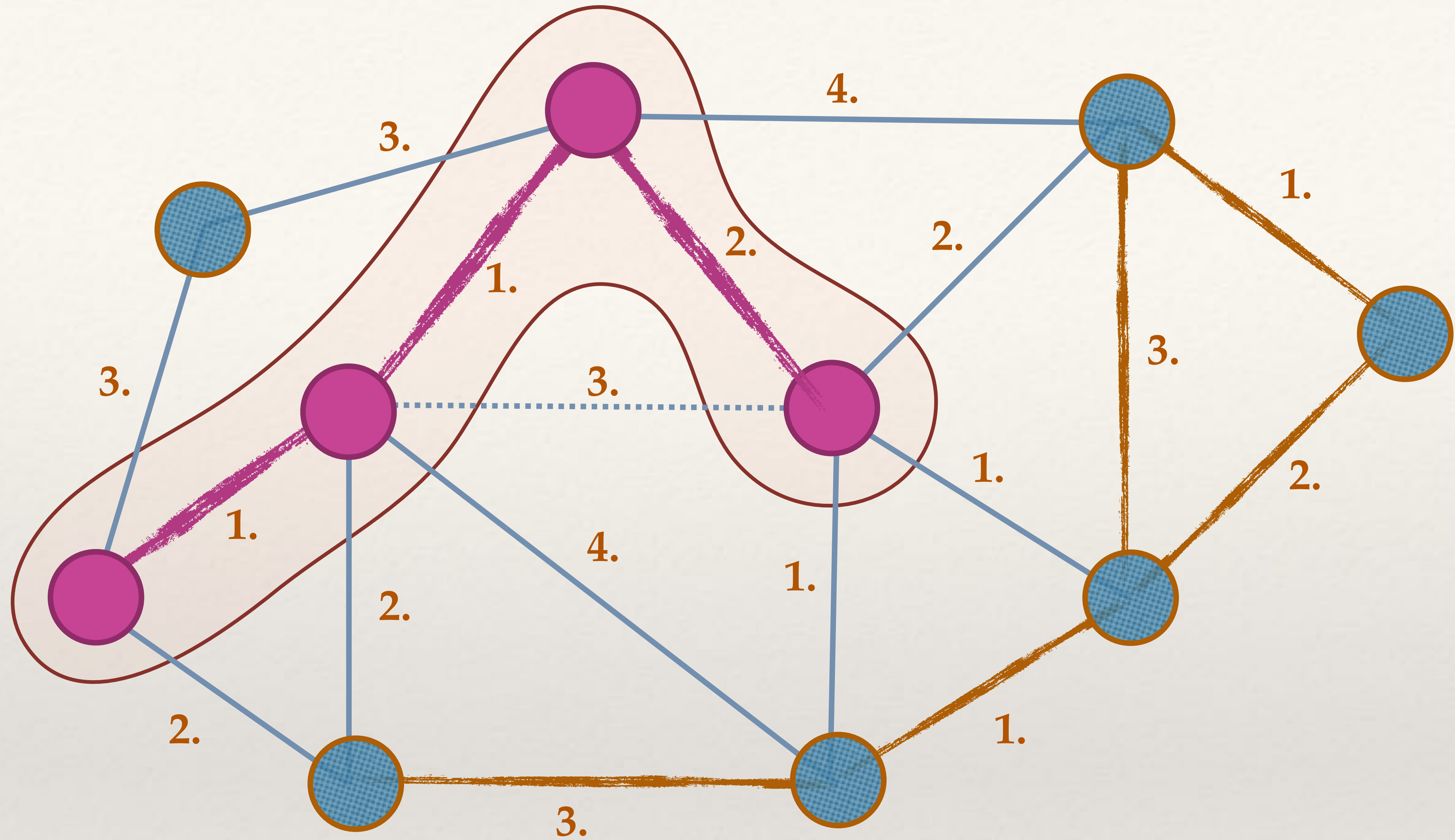
Démo : Minimum Spanning Tree



Important :
Seuls les liens internes à la coupe disparaissent.
Il ne faut pas supprimer les liens bleus sortants, car il serviront peut-être plus tard.

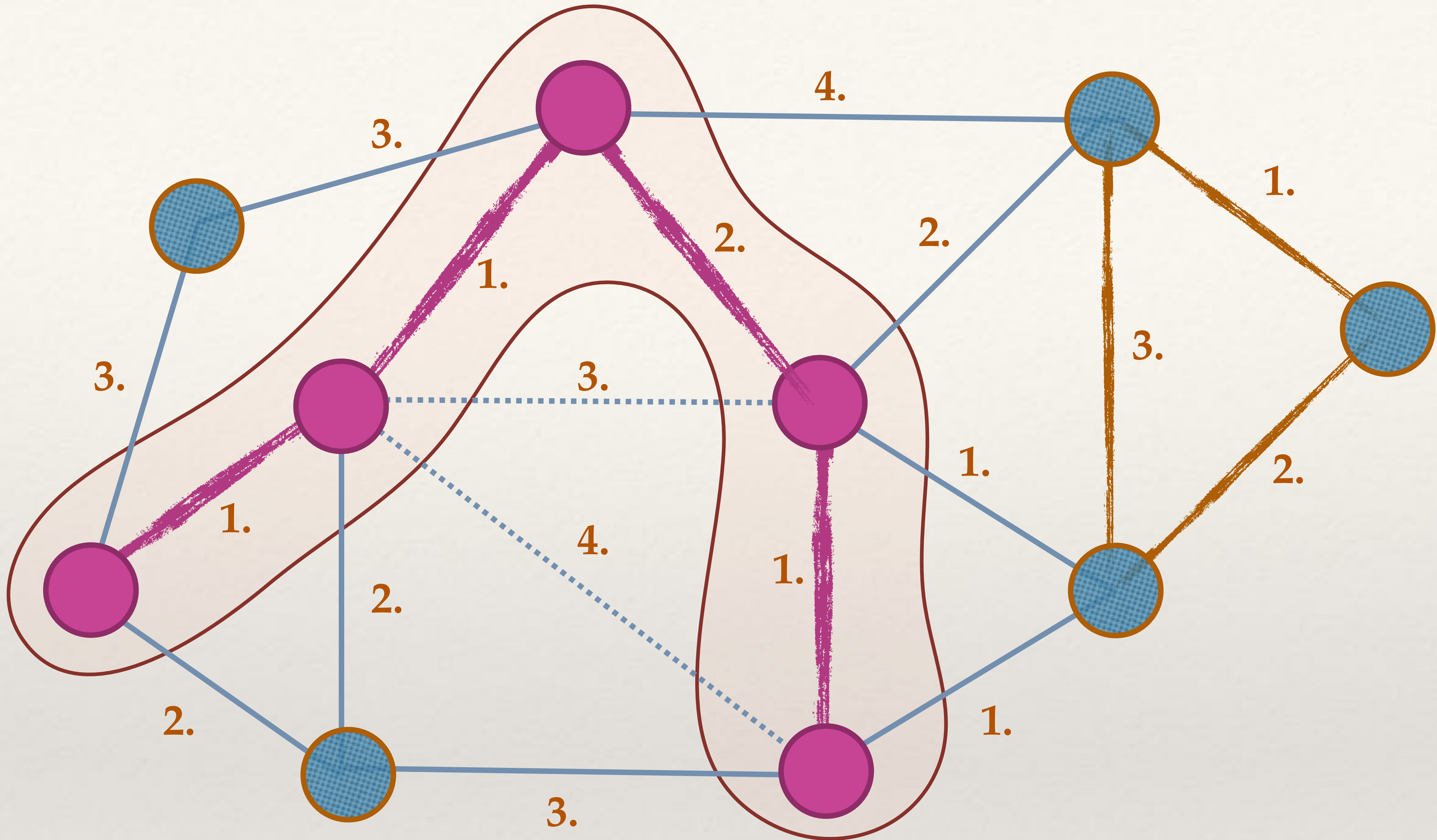
Les cotés marrons n’ont pas encore été visités.

Démo : Minimum Spanning Tree



On voit qu'un lien bleu (pointillés) ne pourra jamais faire parti du MST.
En effet les 2 noeuds à ses extrémités sont déjà connectés par un autre chemin !
On le supprime de la liste car c'est un lien interne à la coupe.

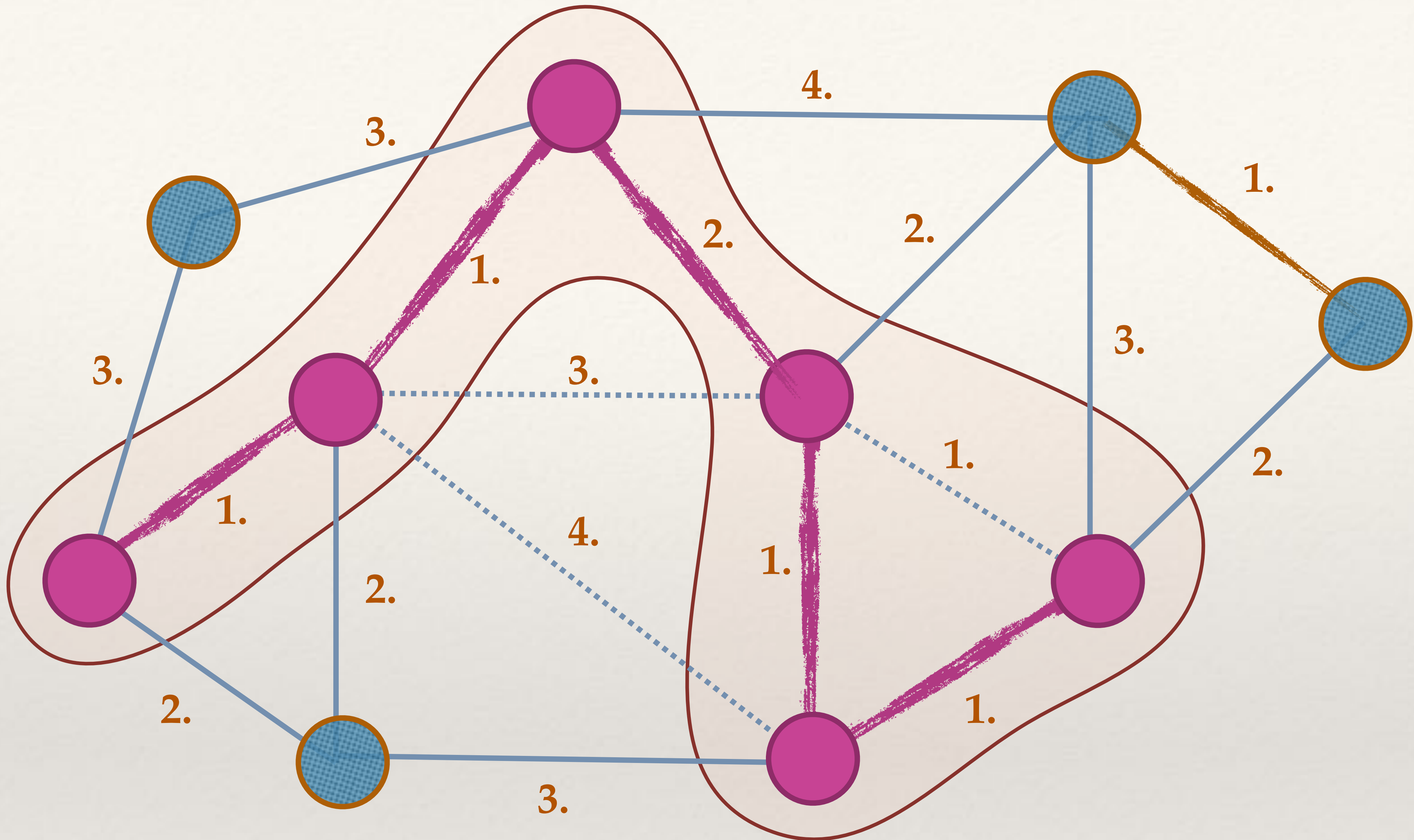
Démo : Minimum Spanning Tree



On poursuit : - choix du noeud le plus proche de la coupe MST.
- mise à jour des cotés « frontières ».

Démo :

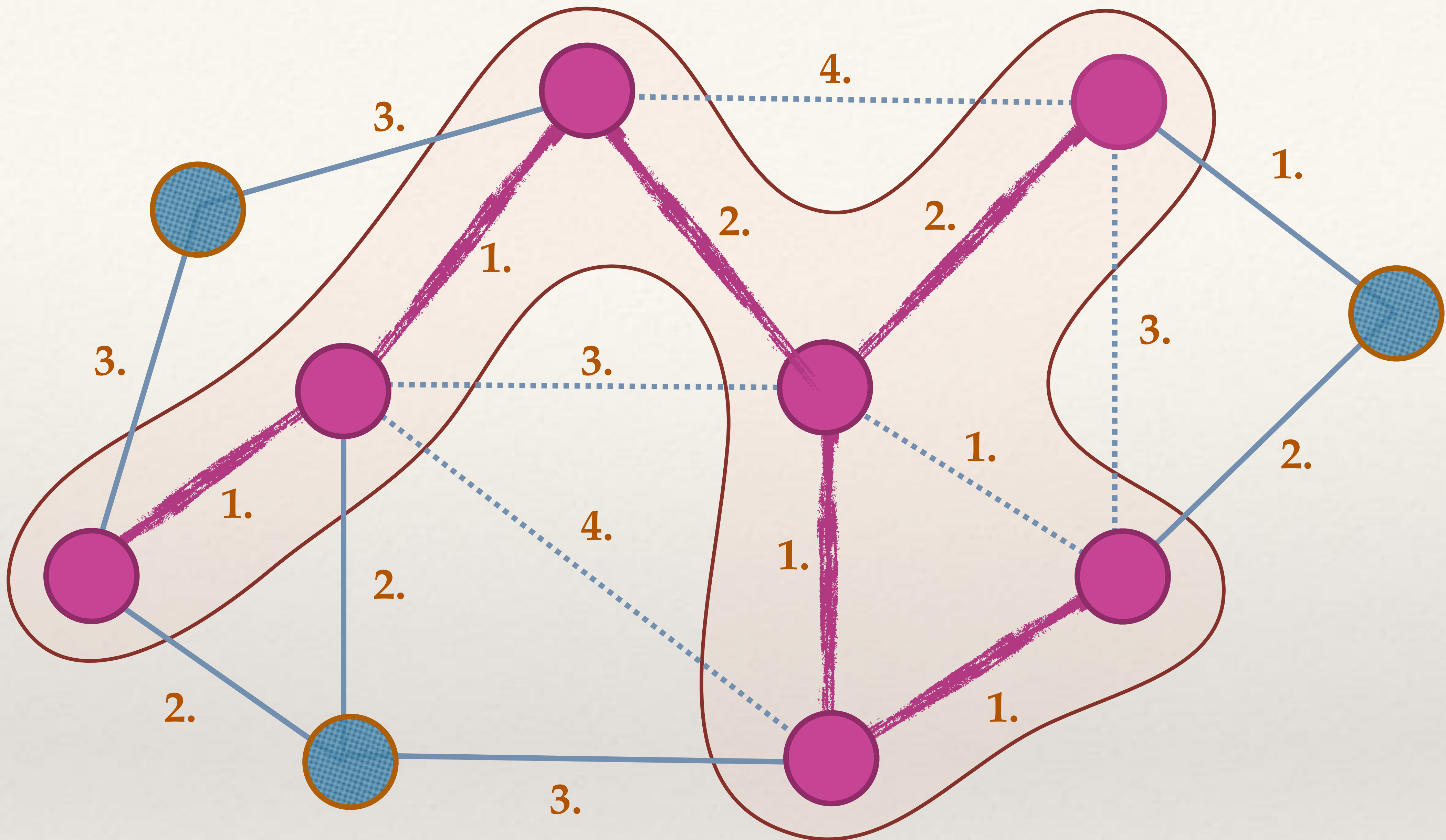
Minimum Spanning Tree



On poursuit :

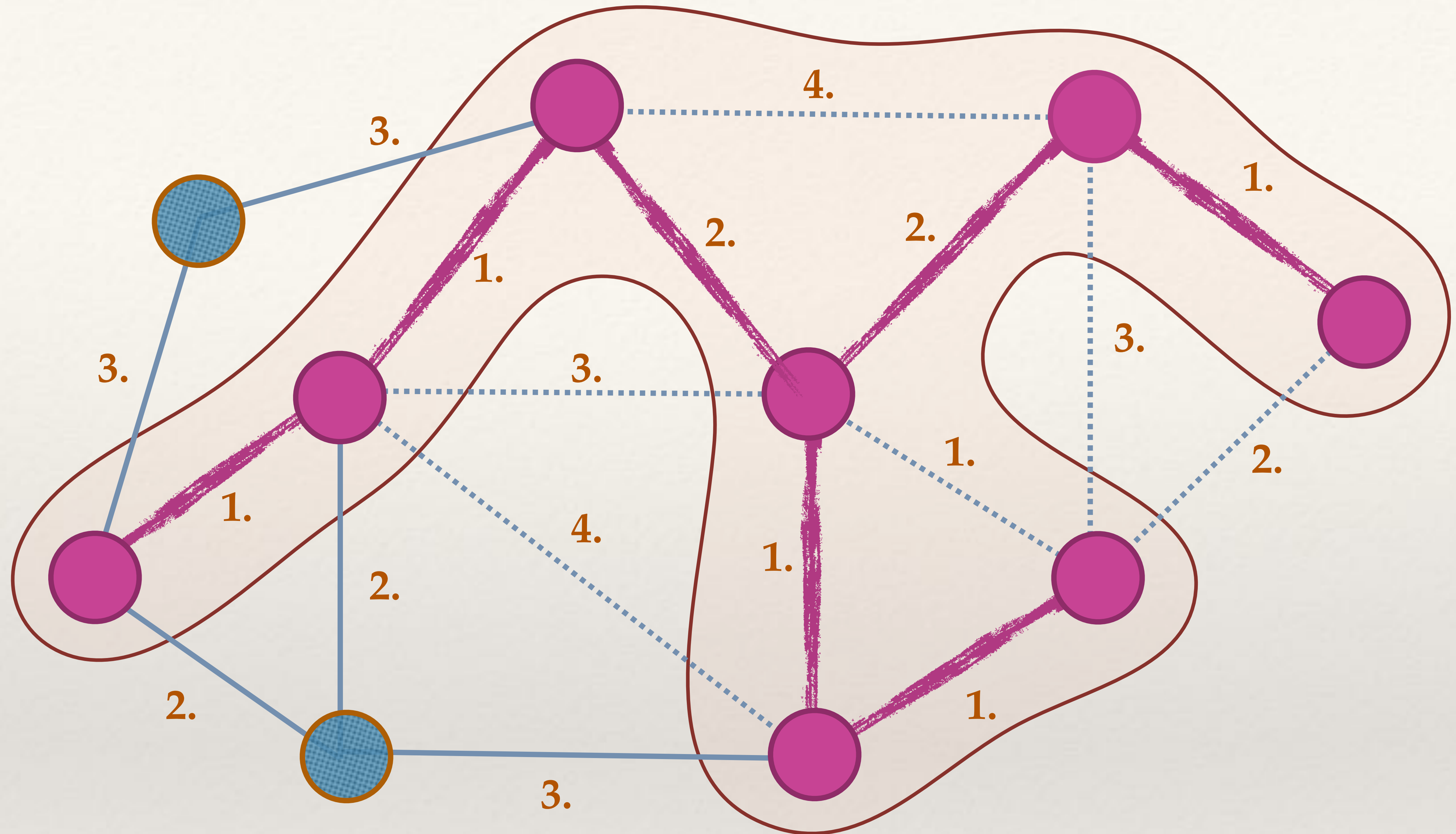
- choix du noeud le plus proche de la coupe MST.
- mise à jour des cotés « frontières ».

Démo :   Minimum Spanning Tree



On poursuit :
- choix du noeud le plus proche de la coupe MST.
- mise à jour des cotés « frontières ».

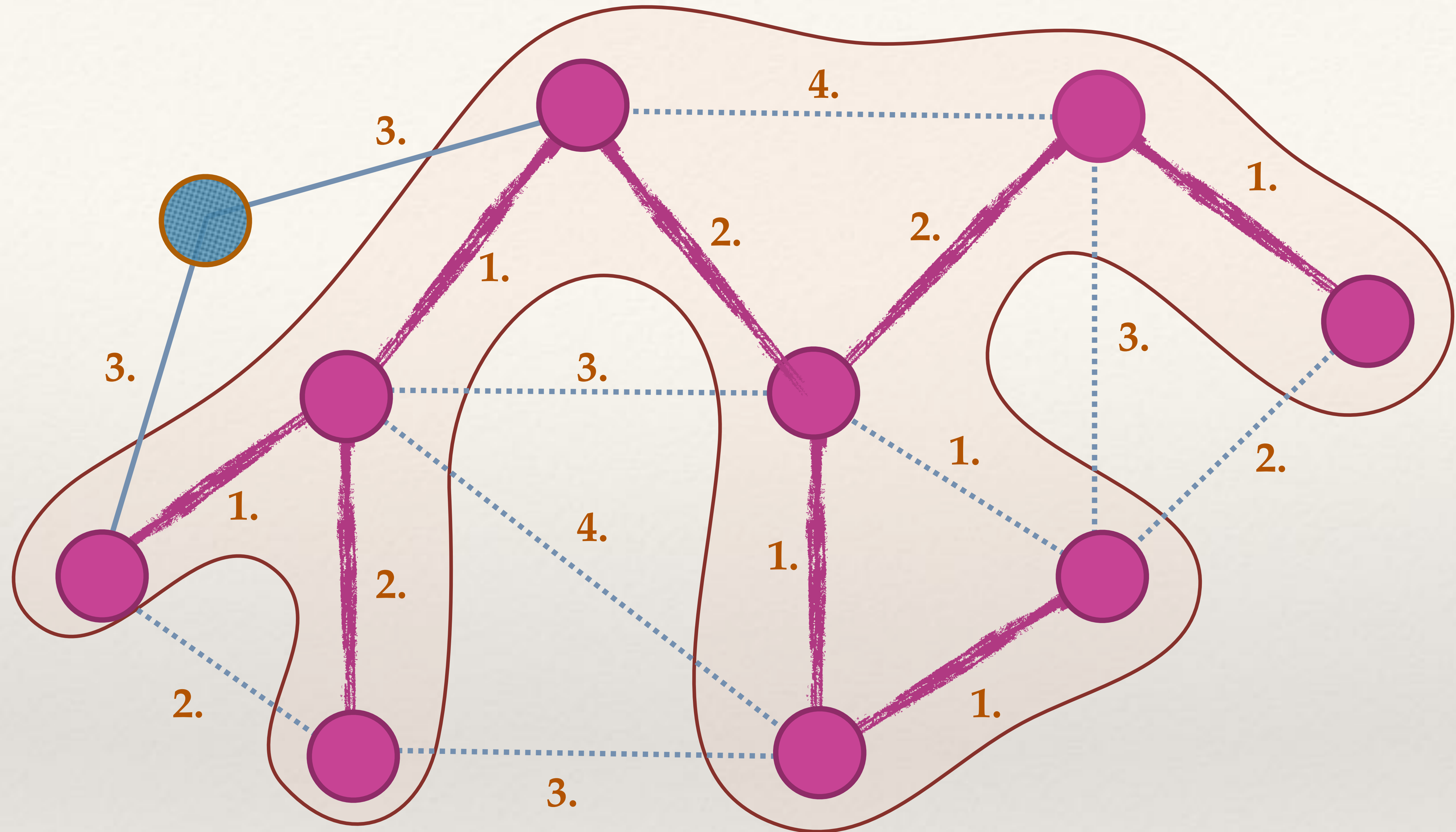
Démo : Minimum Spanning Tree



On poursuit :

- choix du noeud le plus proche de la coupe MST.
- mise à jour des cotés « frontières ».

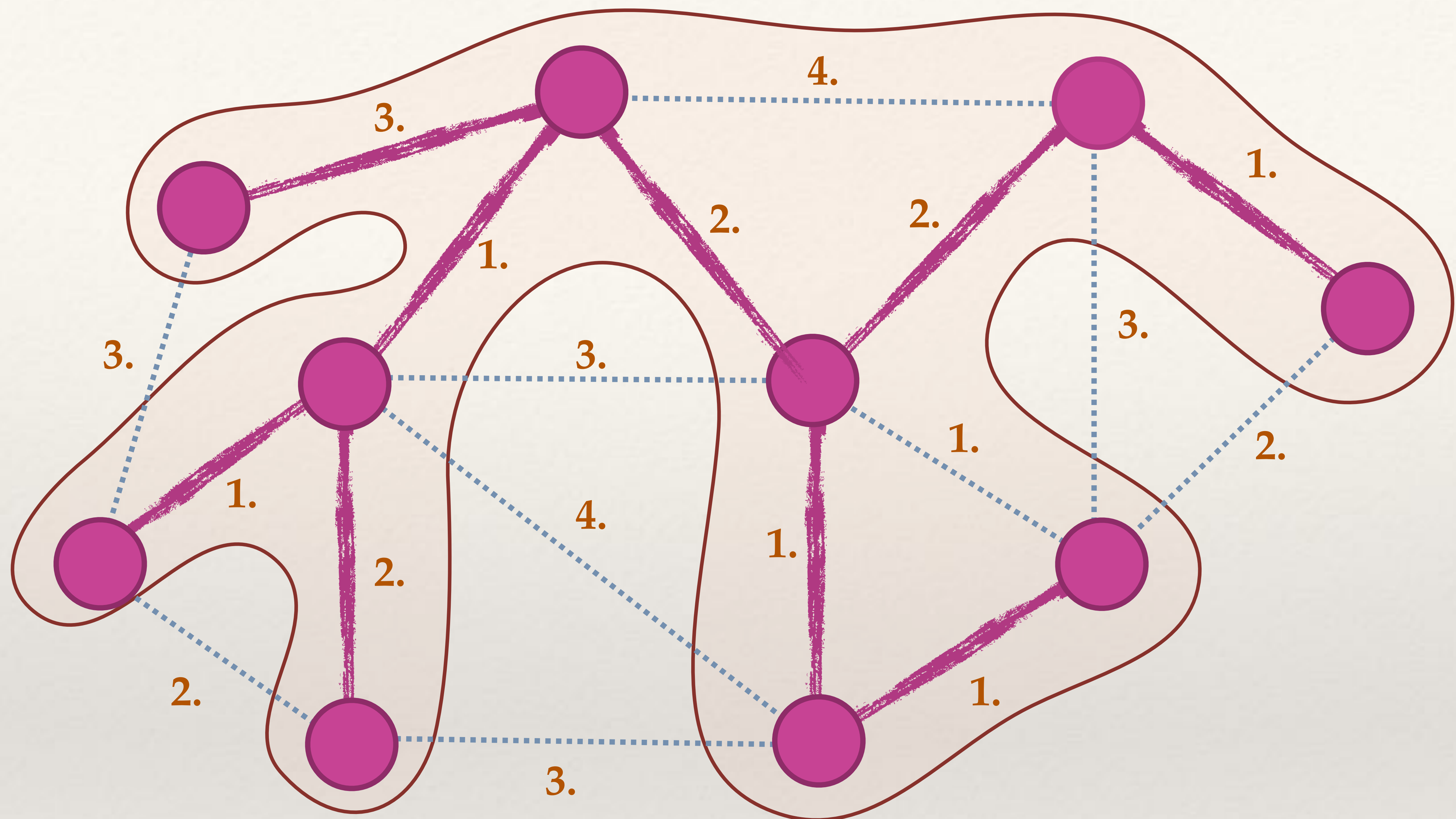
<u>Démo :</u> **Minimum Spanning Tree**



<u>On poursuit :</u>

- choix du noeud le plus proche de la coupe MST.
- mise à jour des cotés « frontières ».

Démo : Minimum Spanning Tree



Terminaison : L'algorithme de Prim termine dès lors que la coupe contient tous les noeuds. Autrement dit : lorsque la liste des cotés « frontières » est vide.

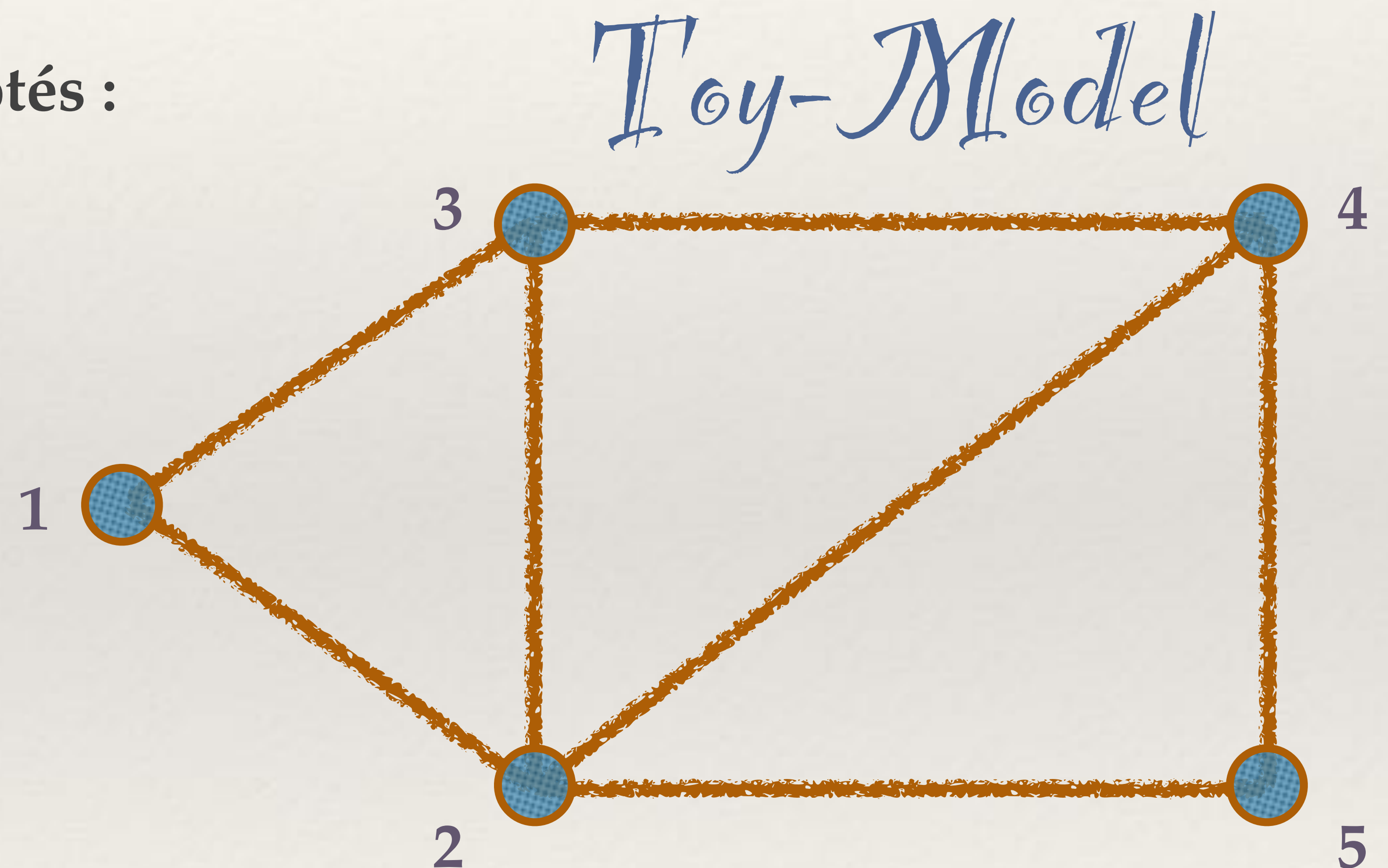
Rq : La taille de la coupe augmente strictement d'une unité à chaque itération et est bornée par la taille N du graphe => l'algorithme termine !

Structures de données

Il existe plusieurs façons standards de représenter les graphes en mémoire, chacune présentant des avantages et des inconvénients.

On se place ici dans le cas le plus simple ou le graphe est connexe, non orienté et sans distance entre les noeuds. Des ajustements mineurs permettent de généraliser la structure de données.

Soit un graphe de N-noeuds et M-cotés :



Matrice d'adjacence :

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

On voit que la matrice est symétrique, ce qui provient du fait que le graphe est non orienté.

Les éléments diagonaux sont nuls car on considère qu'un noeud n'est pas lié à lui même. Mais ce serait le cas de certains automates.

Avantages de la matrice d'adjacence : **- Savoir si deux noeuds sont connectés O(1)**

- Elle possède aussi des propriétés intéressantes (ci-dessous) mais coûteuses….

On introduit le vecteur des noeuds de départ :

Noeud 1

$$N_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Noeud 2

$$N_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$X = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Noeuds N1 ∪ N5

1 - **Calculer :** G.N1, G.N2 et Y = G. X
et interpréter ces résultats.

2 - **Calculer :** La matrice G^2

A partir de l'interprétation du calcul 1 :

- Interpréter les valeurs des coefficients G[i][j] de cette matrice.
- Interpréter en particulier la valeurs des éléments diagonaux qui ne sont plus nuls.
- Le résultat se généralise pour G puissance n.

Solutions :

$$GN_1 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Noeuds accessibles en 1 coup depuis le noeud 1 O(N²)

$$GN_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Noeuds accessibles en 1 coup depuis le noeud 2 O(N²)

$$Y = GX = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

O(N²)

Interprétation : en partant du noeud 1 **ou** du noeud 5,
il y a 2 façons d'atteindre N2,
1 façon d'atteindre N3,
et 1 façon d'atteindre N4.

$$G^2 = \begin{pmatrix} 2 & 1 & 1 & 2 & 1 \\ 1 & 4 & 2 & 2 & 1 \\ 1 & 2 & 3 & 1 & 2 \\ 2 & 2 & 1 & 3 & 1 \\ 1 & 1 & 2 & 1 & 2 \end{pmatrix}$$

O(N³)

$G^2[i][j]$: nombres de façons d'aller de i vers j en 2 coups

$G^2[i][i]$: nombres de façons de faire un aller-retour sur le noeud Ni.
Autrement dit, c'est sa connectivité !

Inconvénients de la matrice d'adjacence :

Le gros point faible de la matrice d'adjacence est que **si le graphe est peu dense, il n'y aura que des zéros dans la matrice** :

Le coût spatial de la matrice d'adjacence vaut O(N²)

Il y a N^2 coefficients pour seulement $N-1$ liens au pire soit $2(N-1)$ coefficients 1.

Le rapport entre l'information stockée et l'information nécessaire vaut : $\frac{1}{N} < \frac{M}{N^2} < 1$

Dans le meilleur des cas, matrice dense, le rendement tend vers 1, mais il tend vers 0 pour des graphes peu denses et de grande dimension. Or un graphe de grande dimension est rarement dense.

Autres points faibles en pratique : **La structure n'est pas vraiment dynamique.**

- **PB indexation :** que se passe-t-il si les indices des noeuds connectés sont [1, 17, 42] ? Ce type de situation se présente très souvent si le graphe est dynamique.

- **PB Graphe dynamique :** comment **ajouter/enlever des noeuds** à la structure de données ? => Il faut recopier toute la matrice **O(N²)**.

Liste d’adjacence :

La liste d’adjacence est simplement un **dictionnaire** où
le numéro du noeud est la clef et la valeur associée
la liste des noeuds vers lesquels il dirige.

G =
1 : [2, 3]
2 : [1, 3, 4, 5]
3 : [1, 2, 4]
4 : [2, 3, 5]
5 : [2, 4]

Chaque liaison y est mentionnée 2 fois (graphe non orienté) donc le contenu en information est de ~2M pour M liaisons. Ce format est donc très compact ! **Coût spatial : O(N.Cmax)**

- On peut ajouter la distance en remplaçant le noeud connecté par le tuple (noeud, distance)
- On peut représenter un graphe orienté par la même structure.

En dehors du caractère compact de la structure de données, l’intérêt du dictionnaire est que l’on a immédiatement accès à la liste des connections d’un noeud. La structure est souple !

Exemple : Les **connexions d’un noeud s’obtient en O(Cmax)** où Cmax est la connectivité maximale (Cmax < N-1) car on accède à la liste en O(1) puis on balaye la liste obtenue.
Par comparaison la matrice d’adjacence demande systématiquement du O(N) car on passe en revue tous les zéros inutiles. Or bien souvent Cmax << N

Exemple : Il est très rapide d’**ajouter ou de supprimer une connexion** sans altérer l’ensemble de la structure de donnée. On ne change que le strict nécessaire : **O(Cmax)**

Liste des cotés :

G = [(1,2), (1,3), (2,3), (2,4), (2,5), (3,4), (4,5)]

La liste des cotés consiste simplement à lister de façon unique tous les cotés du graphe !
On pourrait rajouter la distance sous la forme (p, q, dist) ainsi que l'orientation en tenant compte du sens de lecture du tuple.

Avantage :

- La structure est simple : on ne peut pas être plus compact …
  Pour M cotés on a M entrées: coût spatial O(M)
- Il est facile d'**ajouter un noeud :**
  Il suffit d'ajouter les connexions à la fin de la liste **O(Cmax)**
- pas de difficulté quant à l'indexation des noeuds.

Inconvénients : Gros Problème : **On a pas un accès rapide et dynamique aux données**

- Comment **connaitre les connexions** d'un noeud :
  => Il faut passer en revue les M connexions du graphe **O(M)**
- Comment **savoir si deux noeuds sont connectés O(M)**
- Comment **enlever un noeud** il faut à nouveau passer en revue toutes les connexions du graphe : **O(M)**