**U.F.R SCIENCES ET TECHNIQUES**
*Département d'Informatique*
B.P. 1155
64013 PAU CEDEX
Téléphone secrétariat : 05.59.40.79.64
Télécopie : 05.59.40.76.54

# TYPE DE STRUCTURES DE GRAPHE

I- NOTIONS DE GRAPHE
II- TYPE GRAPHE ORIENTE
III- REPRESENTATION DE GRAPHE

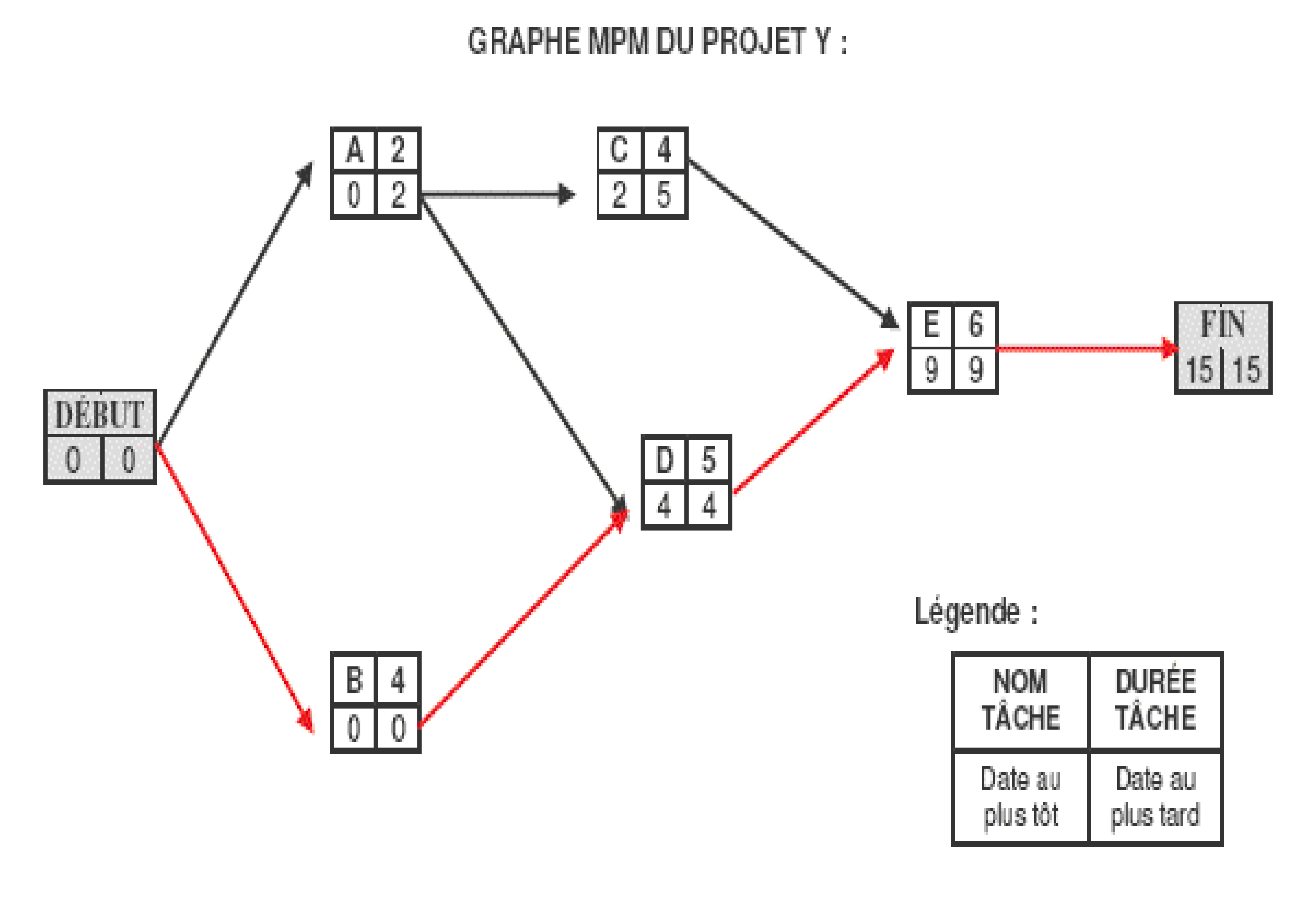
# I- Notion de graphe

Beaucoup de données traitées dans la vie courantes sont des **structures relationnelles**.

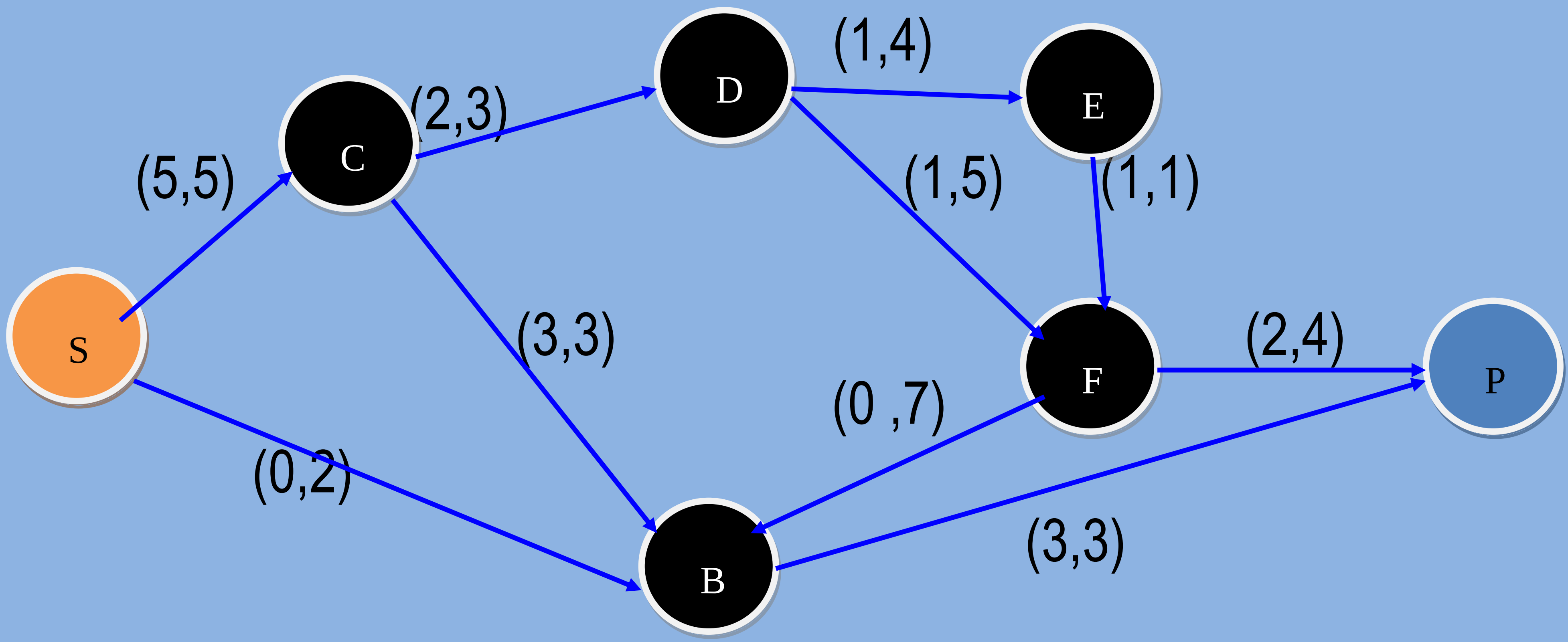
## 1- Pourquoi les graphes ?

Dans un programme, les structures relationnelles sont modélisées à l'aide des **graphes**.

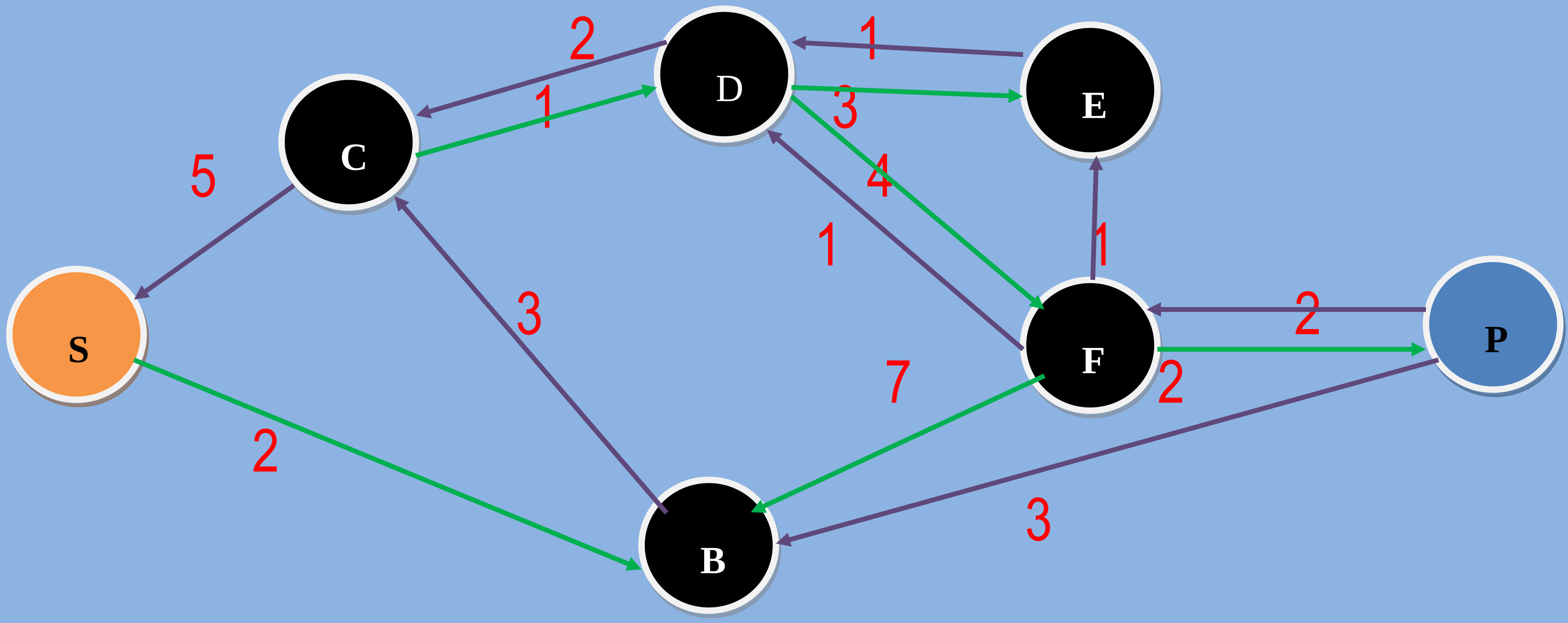
# Modèle de planification d'un projet



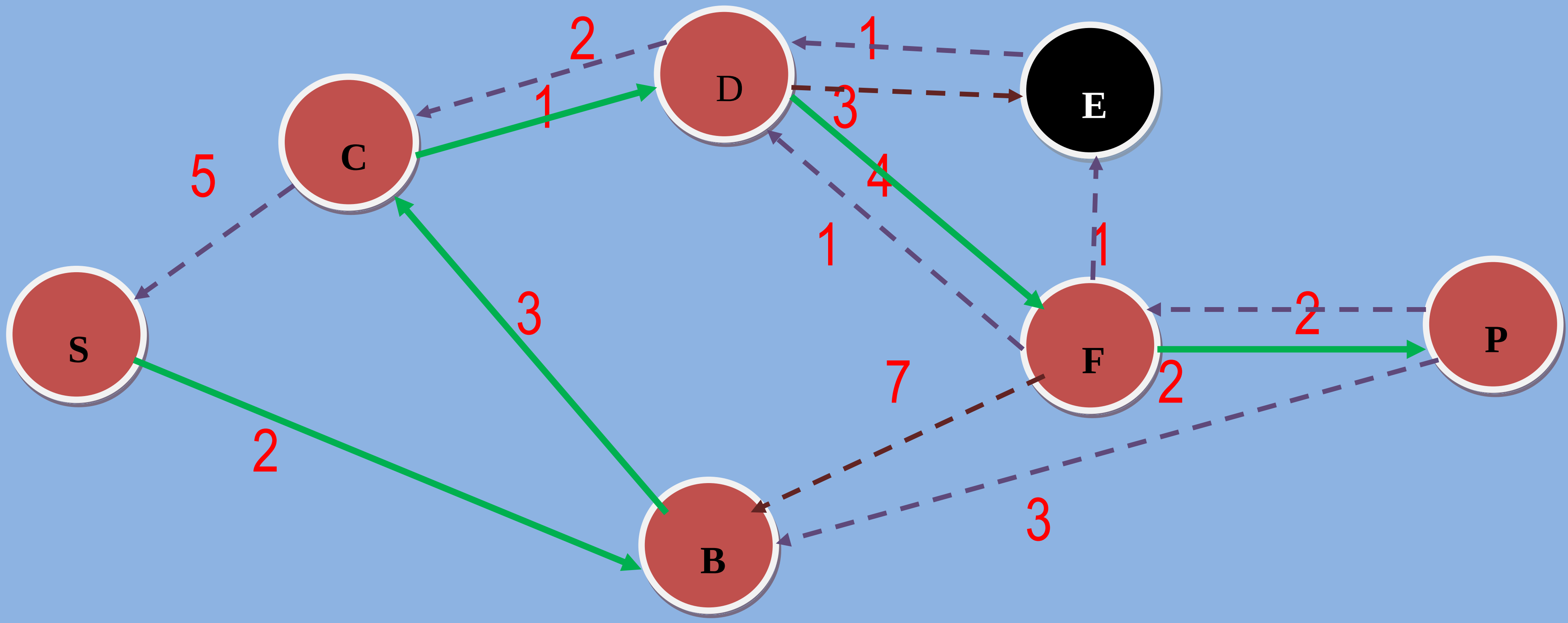
# Modèle de circulation de flot dans un «réseau transport» .



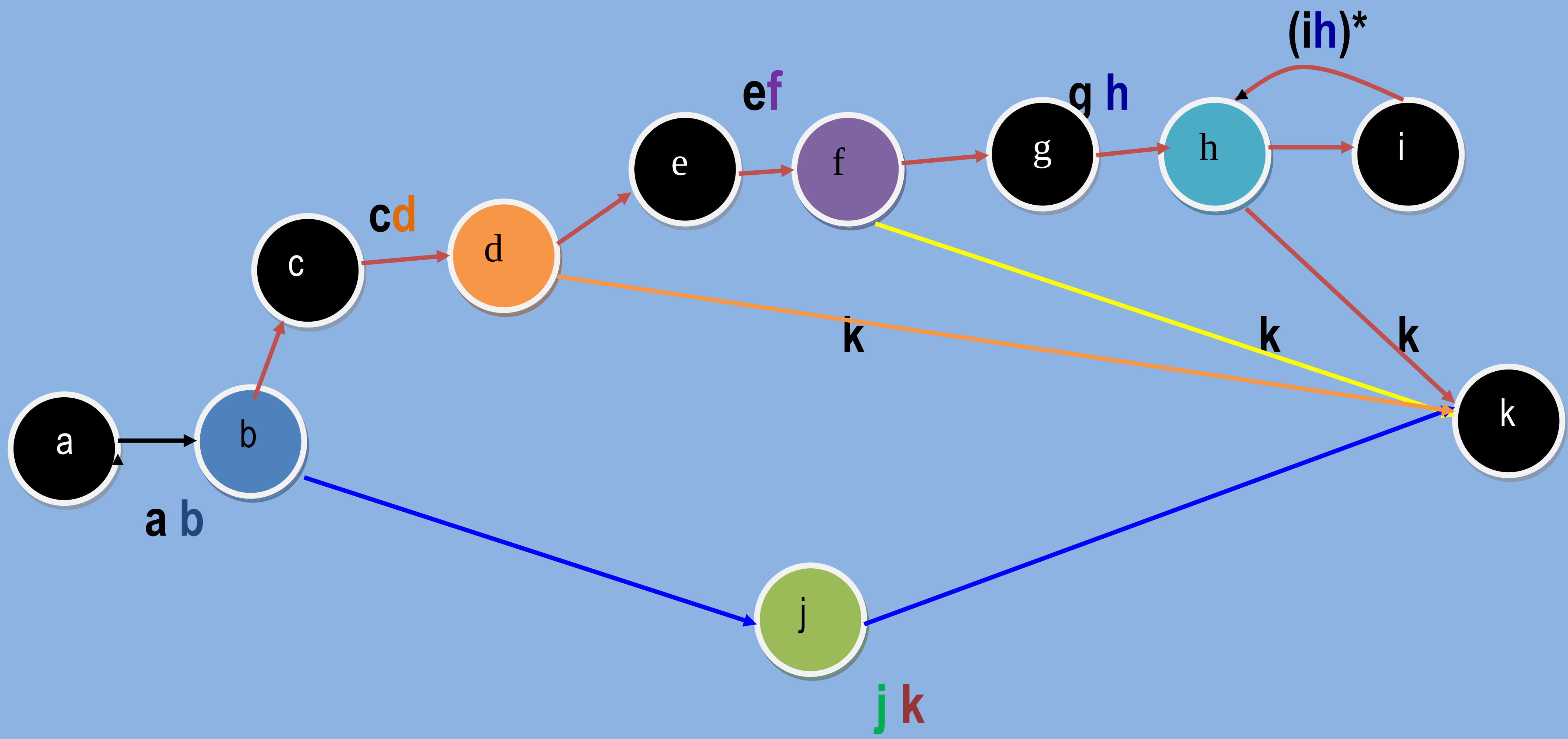
# Le graphe d'écart correspondant



# Graphe d'augmentation du flot

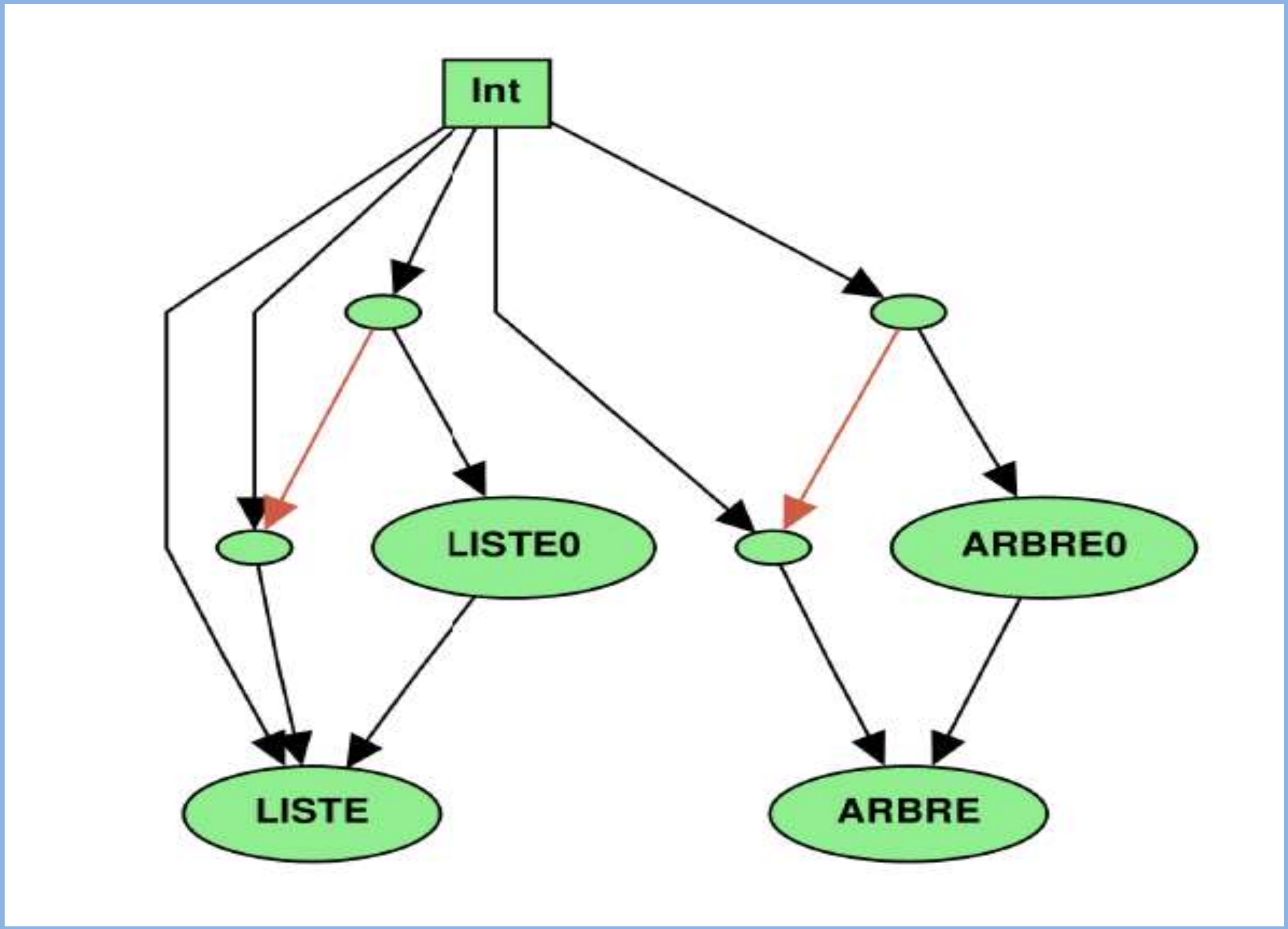


# Modèle d'un programme : graphe de contrôle

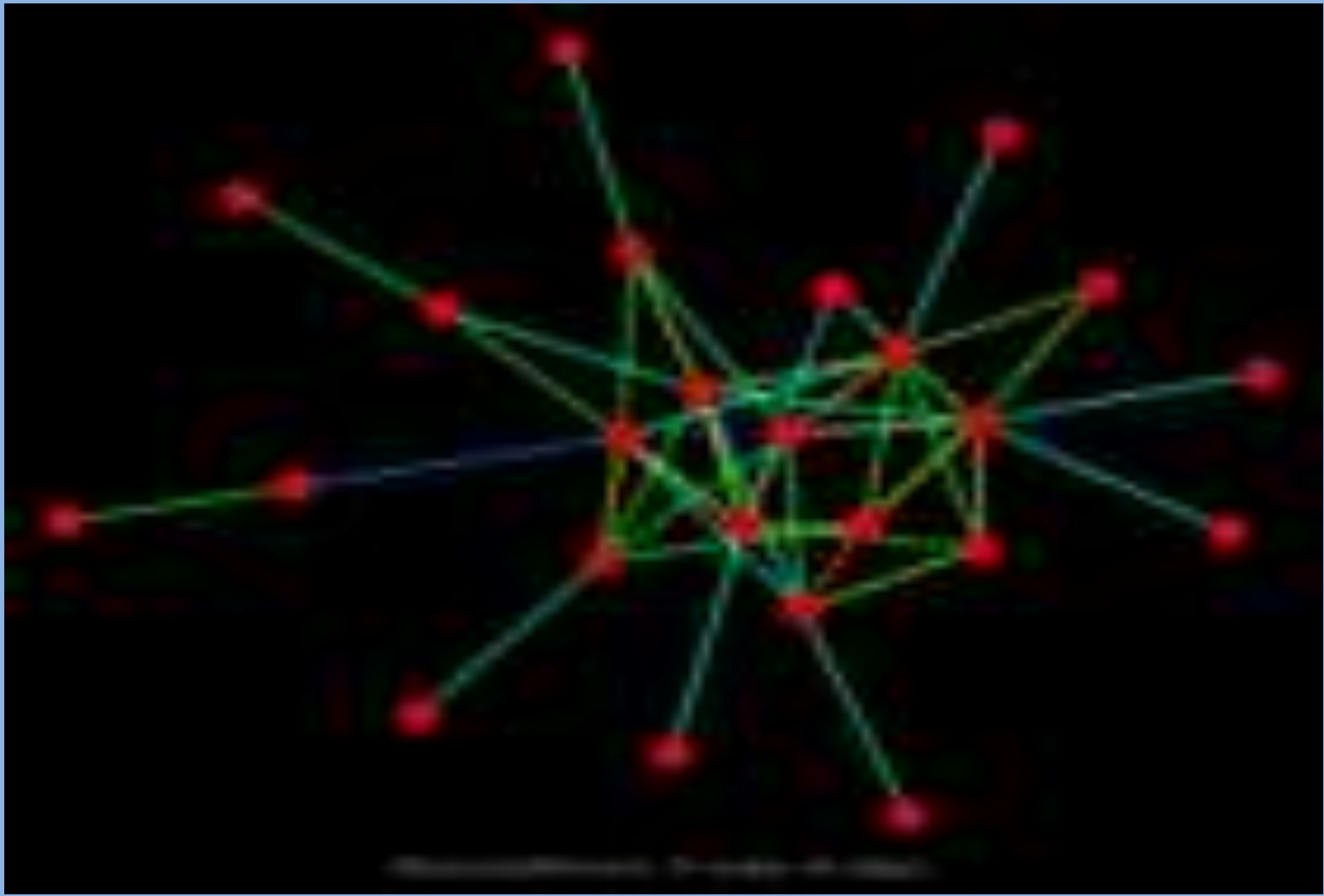


```
Unknown()
    begin
    read(b,c,x)                                          a
    if b < c    then begin                               b
                d:=2*b ; f:=3*c                          c
                if x>= 0    then begin                   d
                            y:= x; e:= c                 e
                            if(y=0)    then begin        f
                                       a:=f-e            g
                                       while d<a  begin  h
                                                  d:=d+2 i
                                                  end    k
                                       end               k
                            end                          k
        else begin
                b:=b-1                                   j
                end                                      k
    end                                                  k
    end
```

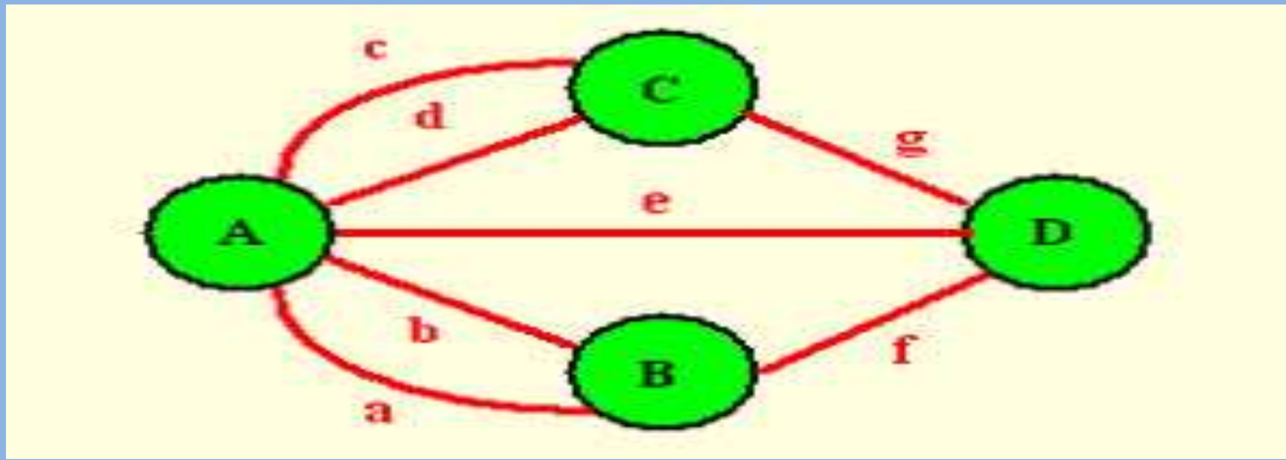
# Graphe de preuves de Vinci (Casl)



# Graphe de trafic aérien



# Modèle d’Euler  des 7 ponts de  Königsberg




# 2- Qu'est-ce qu'un graphe ?

De façon formelle, on appelle graphe G :

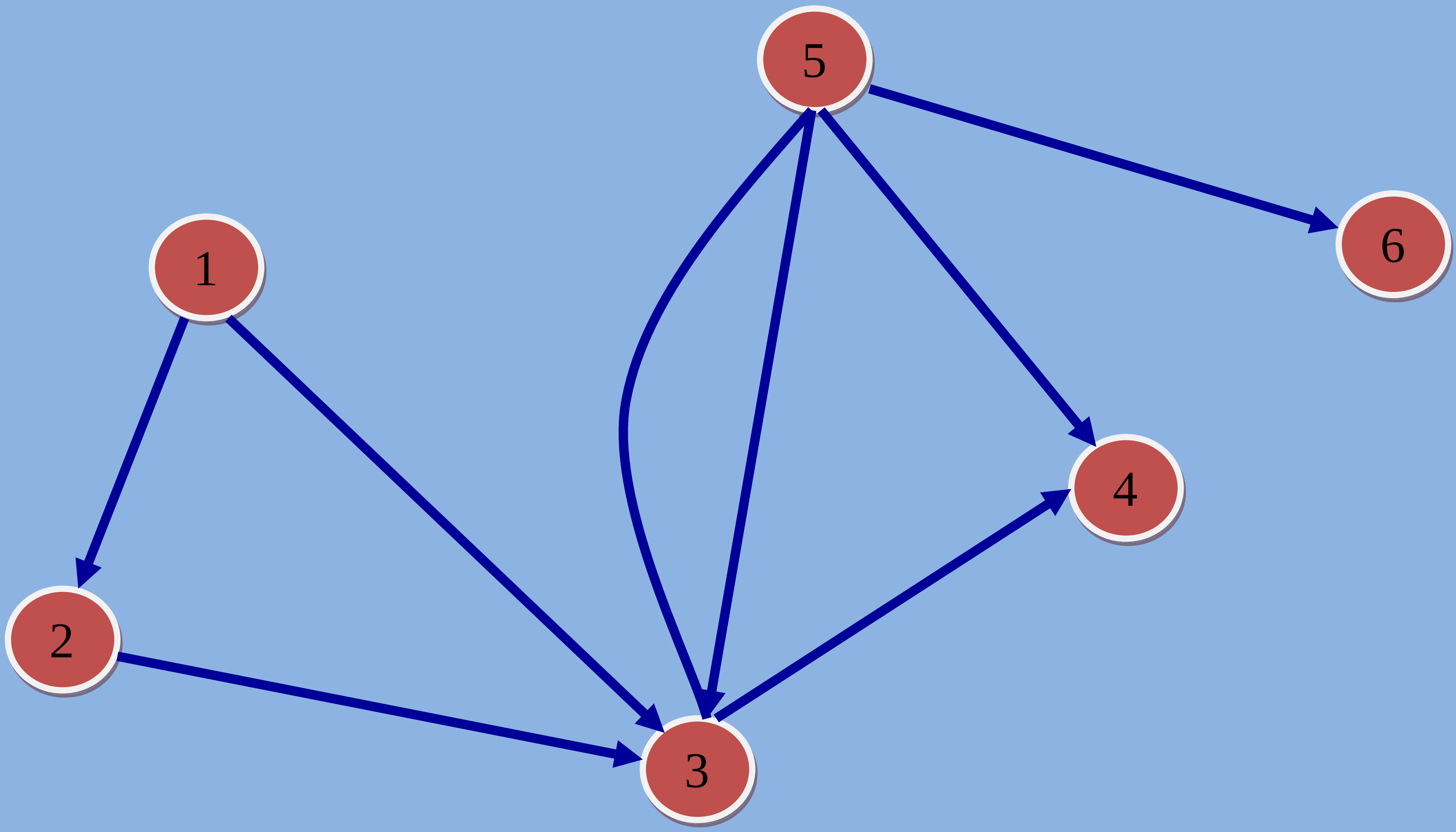
-un ensemble **S** d'objets appelés **noeuds**,

-un ensemble **A** de **relations** entre ces nœuds.

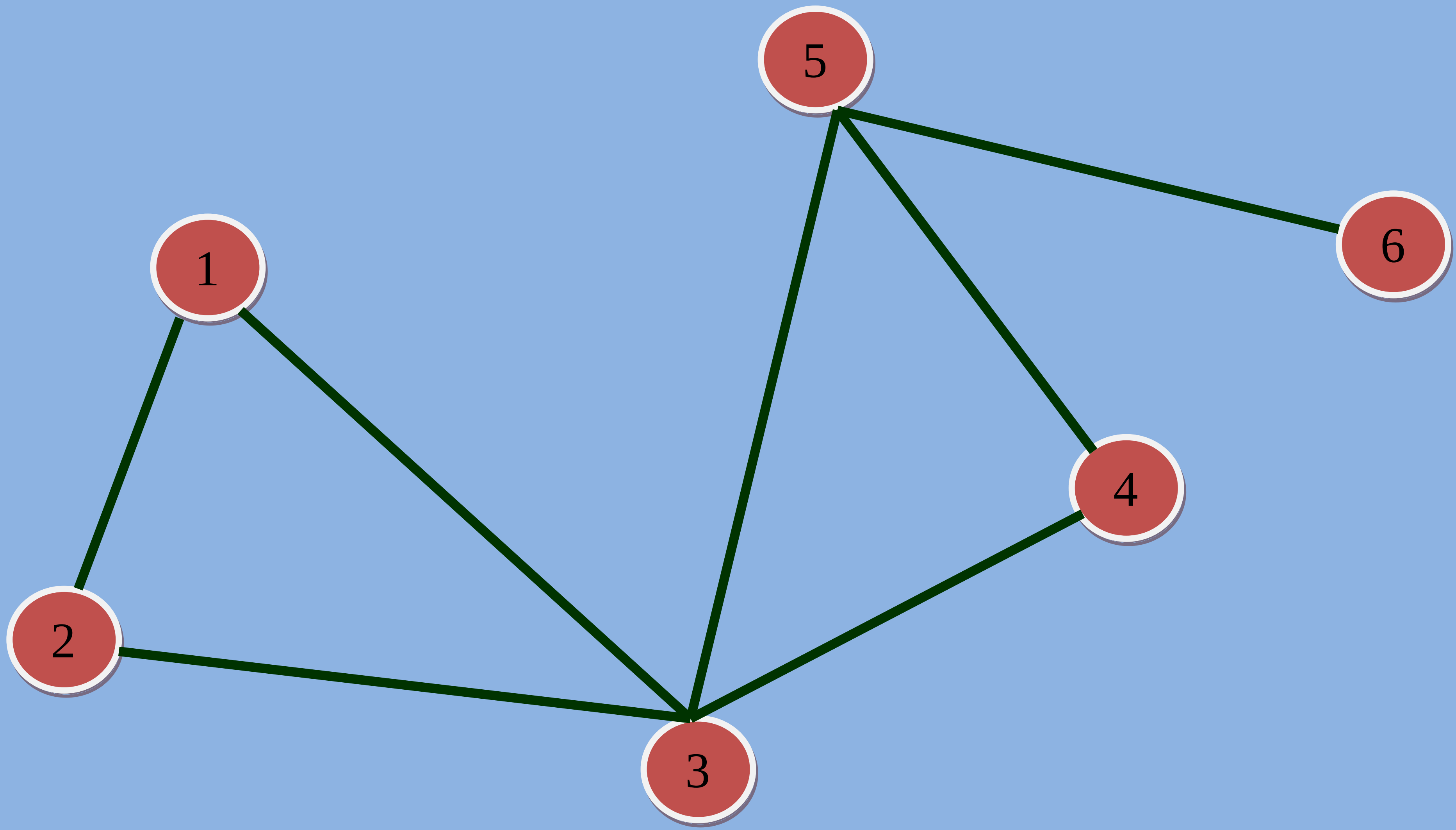
On note habituellement:

**G= (S, A)**

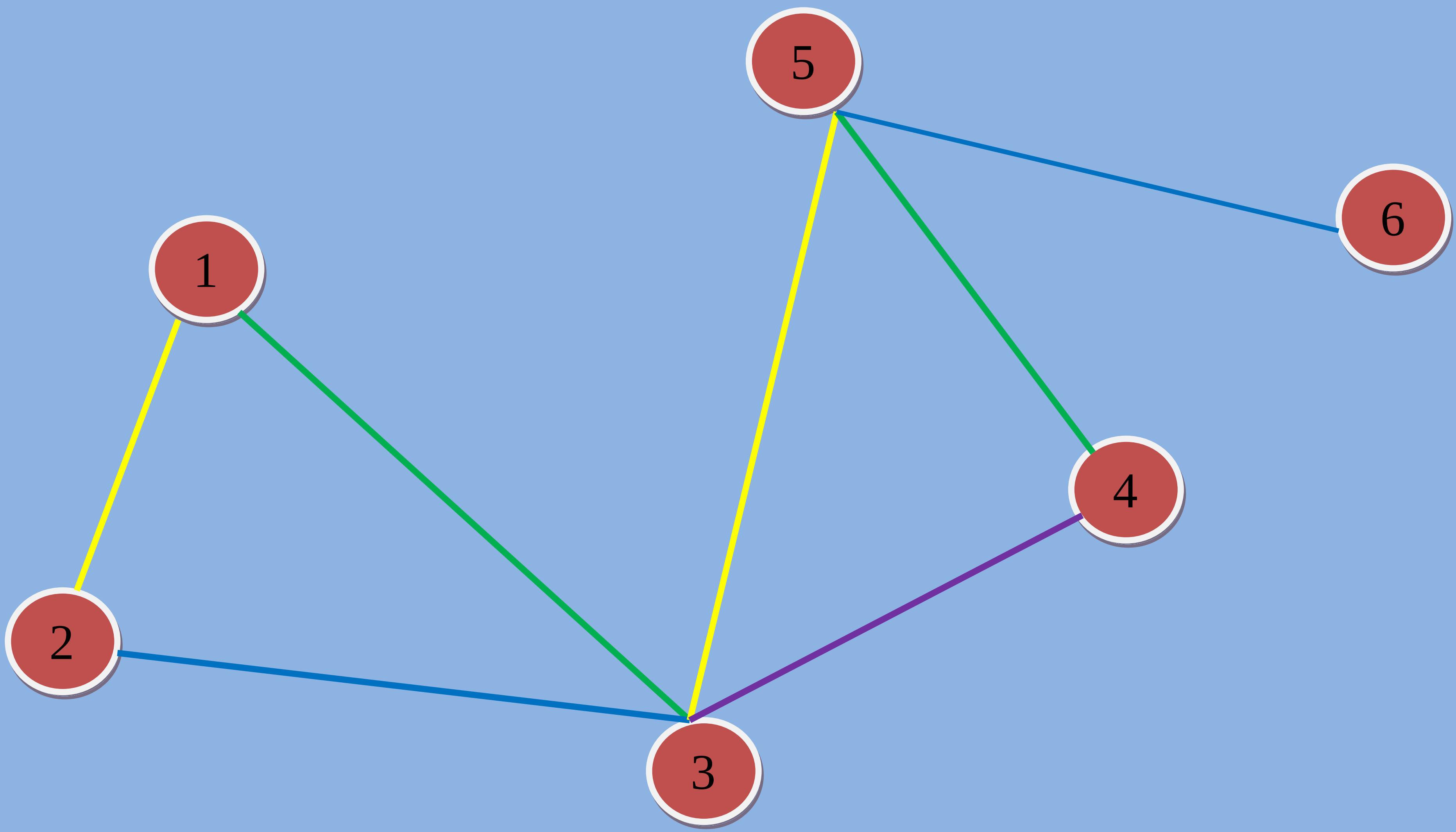
# Graphe orienté



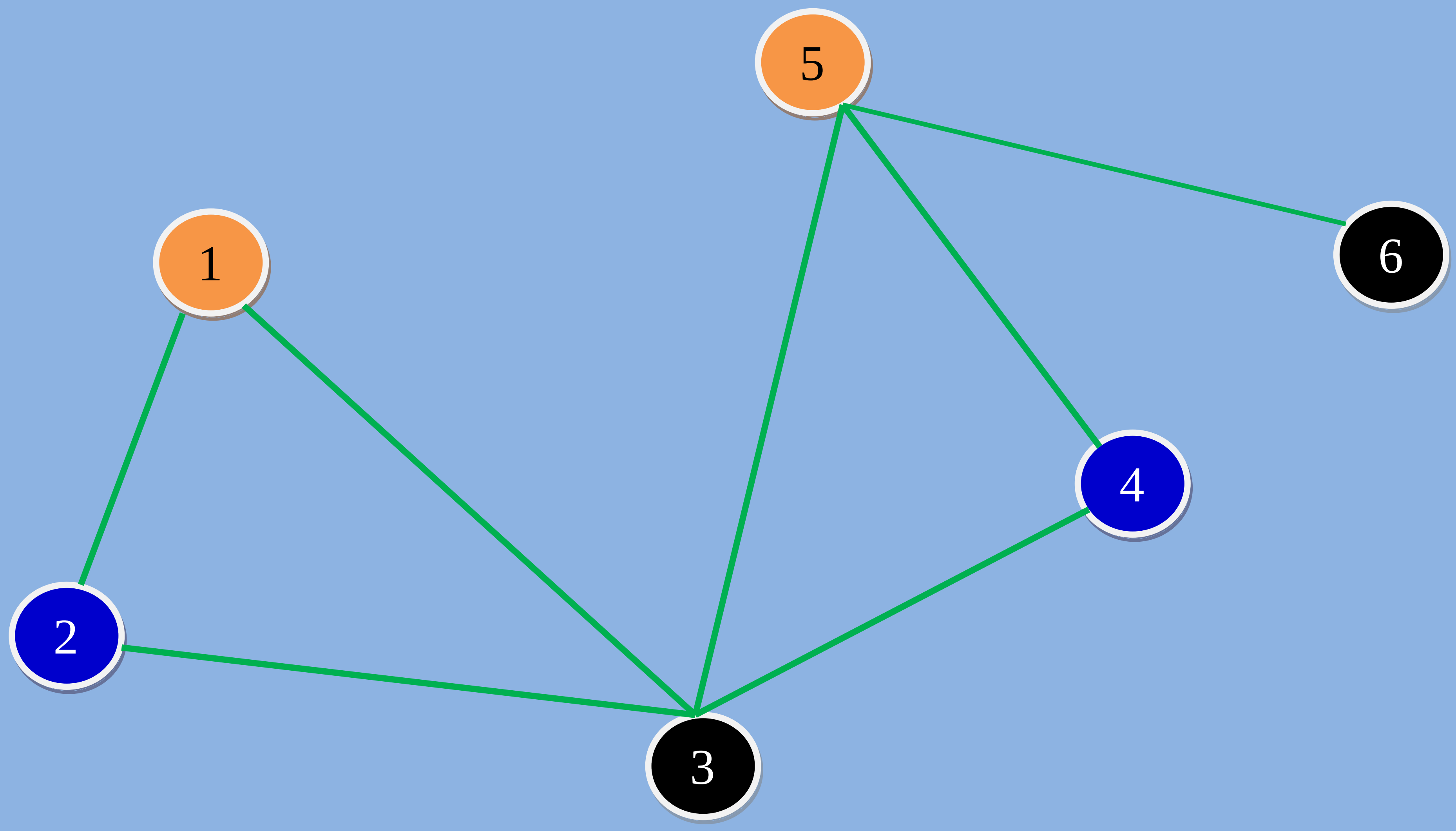
# Graphe non orienté



# Coloration des arêtes



# Coloration des nœuds



Deux hypothèses se présentent:

1- les relations sont **symétriques** : on parle alors de **graphe non orienté**,

2- les relations ne sont **pas symétriques** : on parle alors de **graphe orienté**.

# Graphe orienté

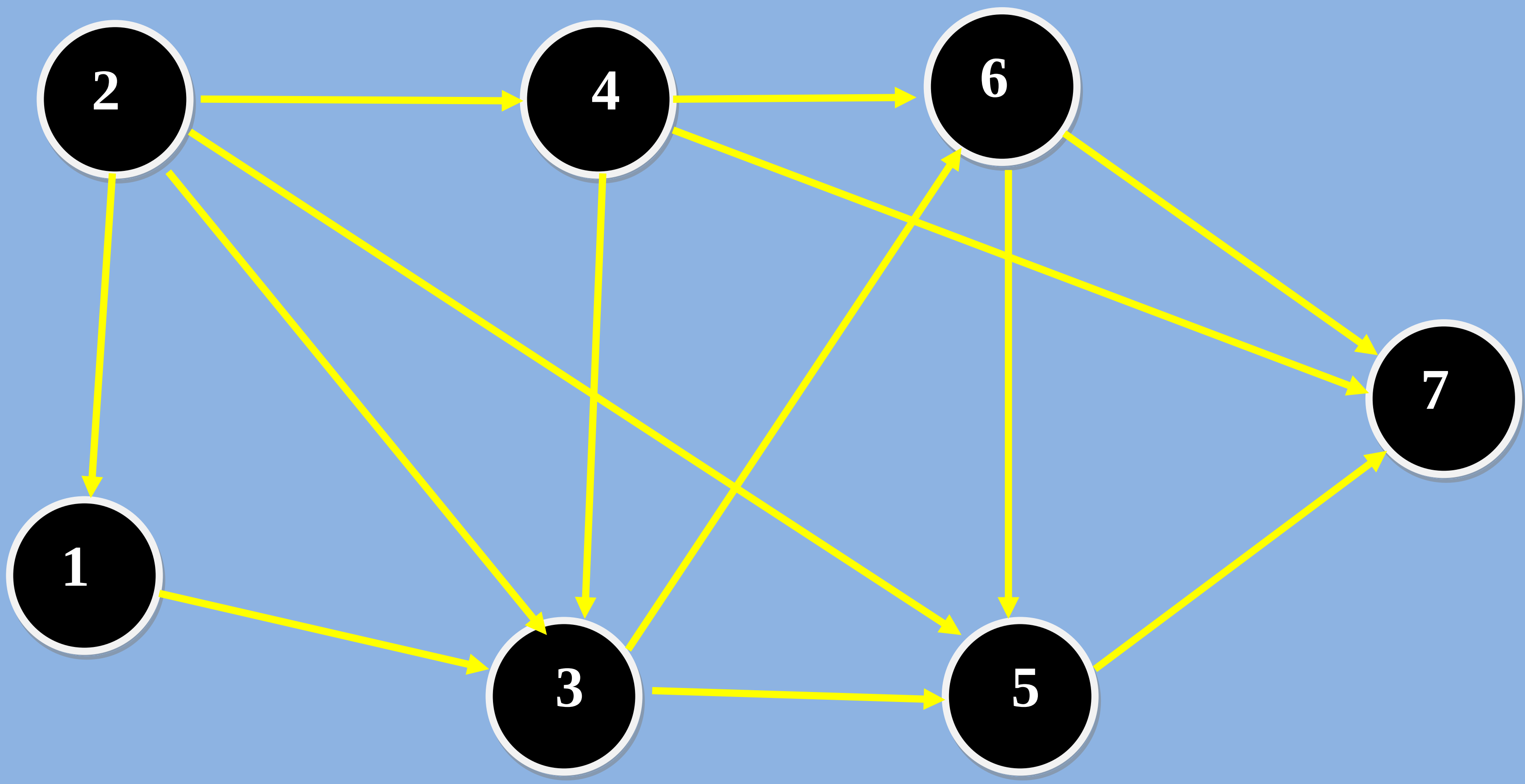
Un graphe orienté G est un couple :

$$G= (S,A)$$

Où :

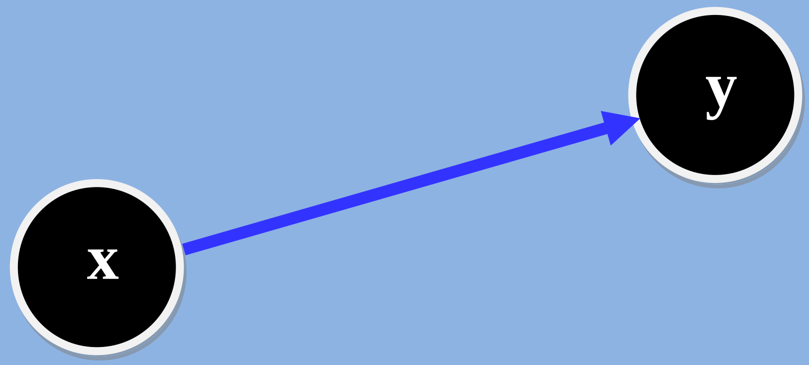
- S ensemble fini de **nœuds**,
- A, un ensemble fini de **paires ordonnées** de nœuds, appelées **arcs**.

# Graphe orienté



On note x$\longrightarrow$ y l'arc (x,y):

- **x** désigne l'**extrémité initiale**,
- **y** désigne l'**extrémité terminale**.

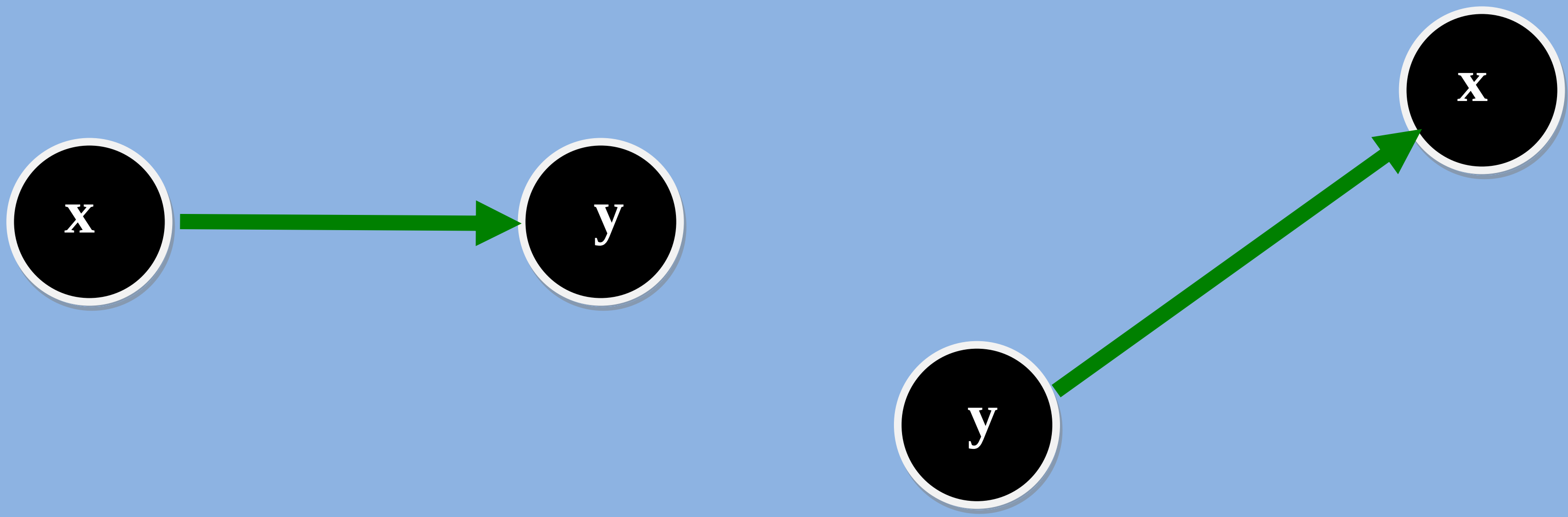



On dit que :

- **y** est le **successeur de** x,
- **x** est le **prédécesseur de** y.

Le nœud **y** est dit **adjacent** à **x** s'il existe un arc

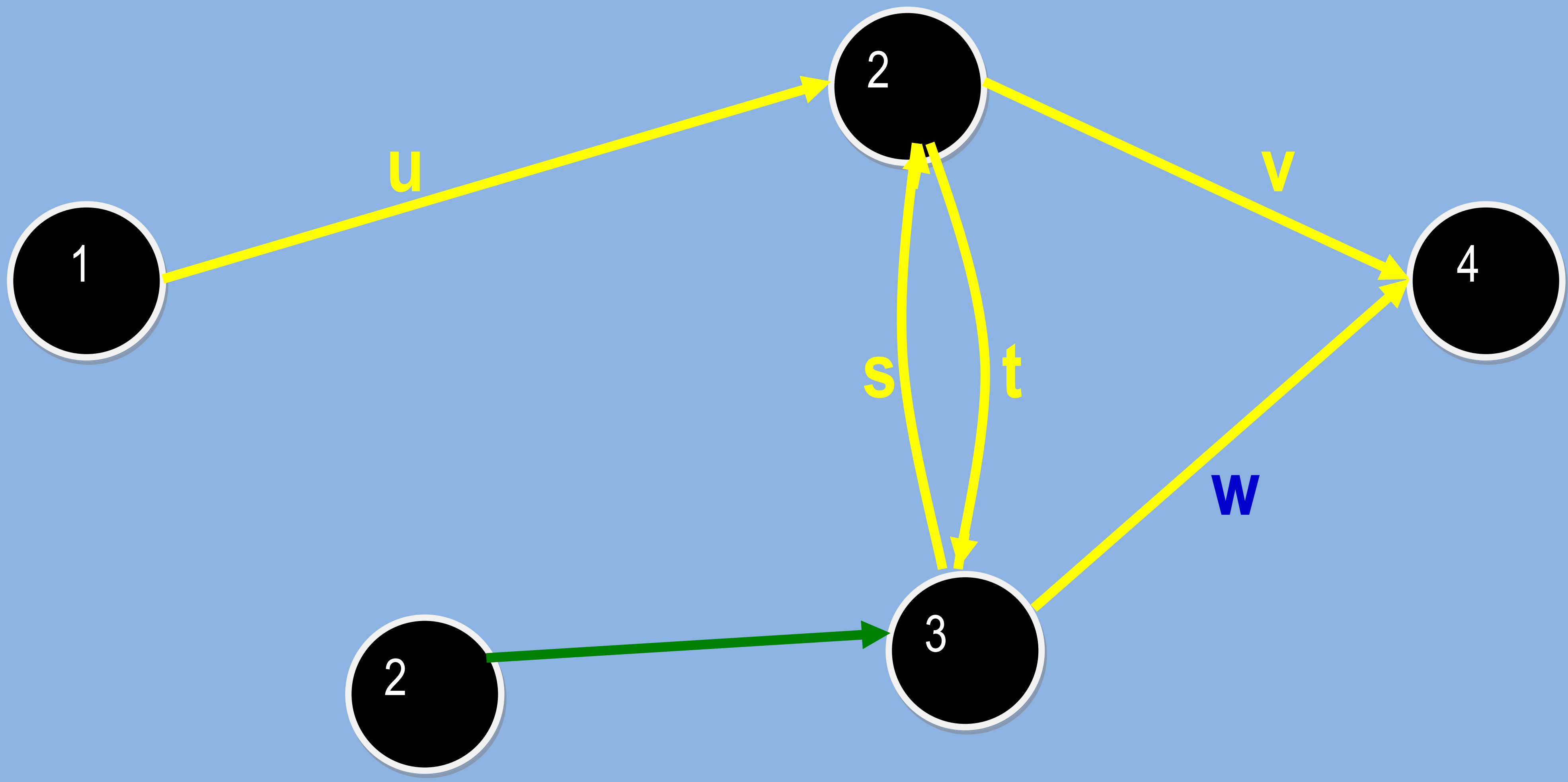
$x \longrightarrow y$ ou $y \longrightarrow x$



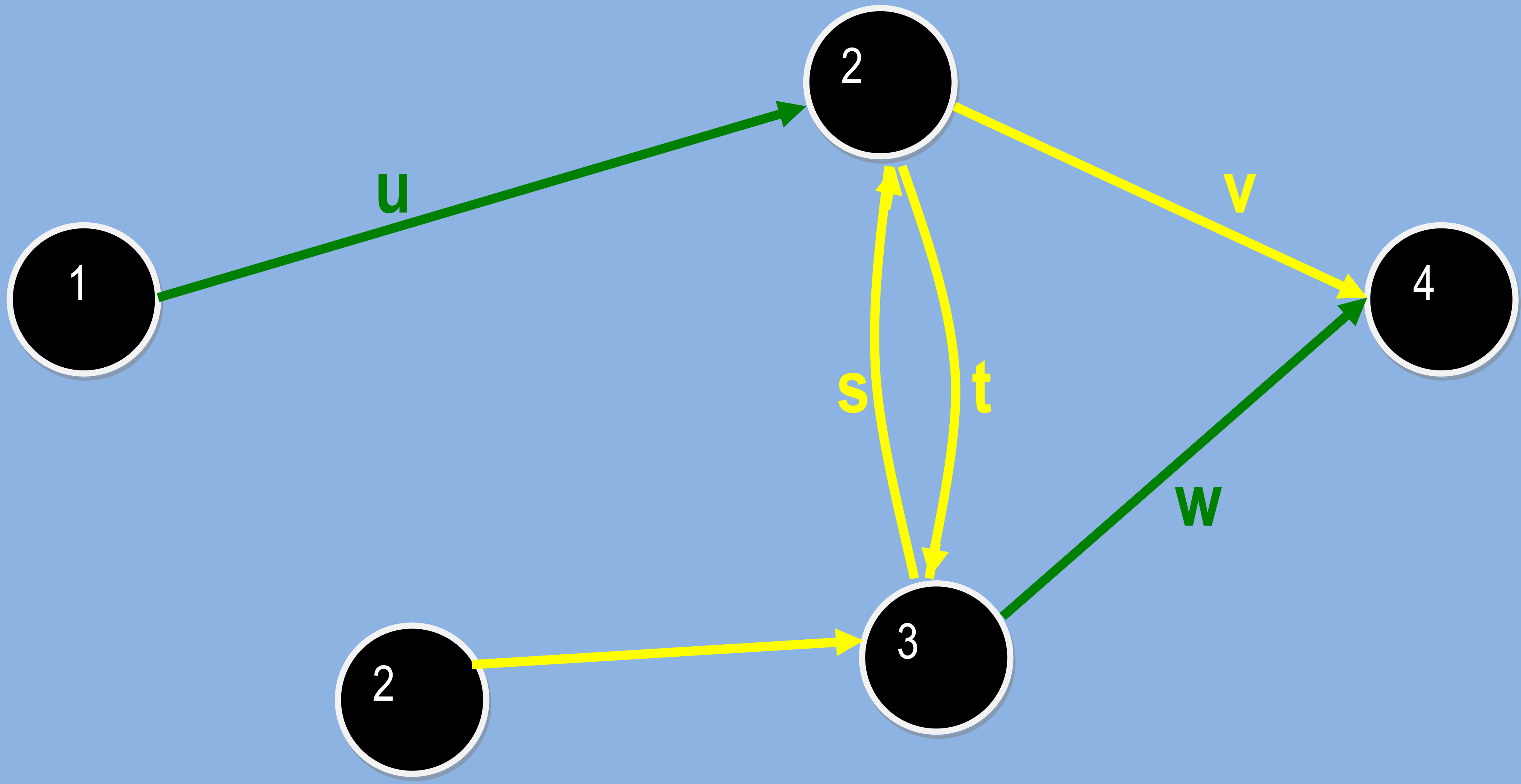
## 4- Relation entre arcs/arêtes et nœuds

### Arcs/arêtes adjacents

Deux arcs (arêtes) sont **adjacents** s'ils ont au moins une **extrémité commune**.




u,s,t,v sont des arcs **adjacents**



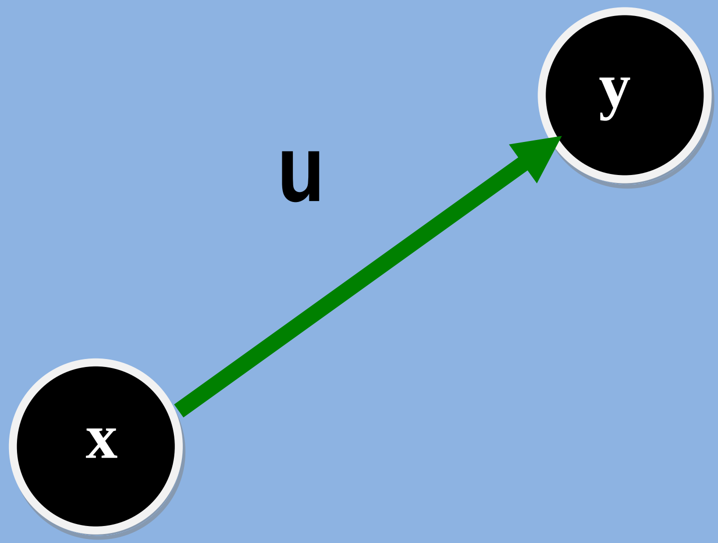

u et w sont des arcs **non adjacents**

# Incidence Arc/nœud

Si le nœud **x** est l'extrémité initiale d'un arc u :

$$\mathbf{u} = \ \mathbf{x} \longrightarrow y$$

on dit que **u** est **incident** à x **vers l'extérieur**.

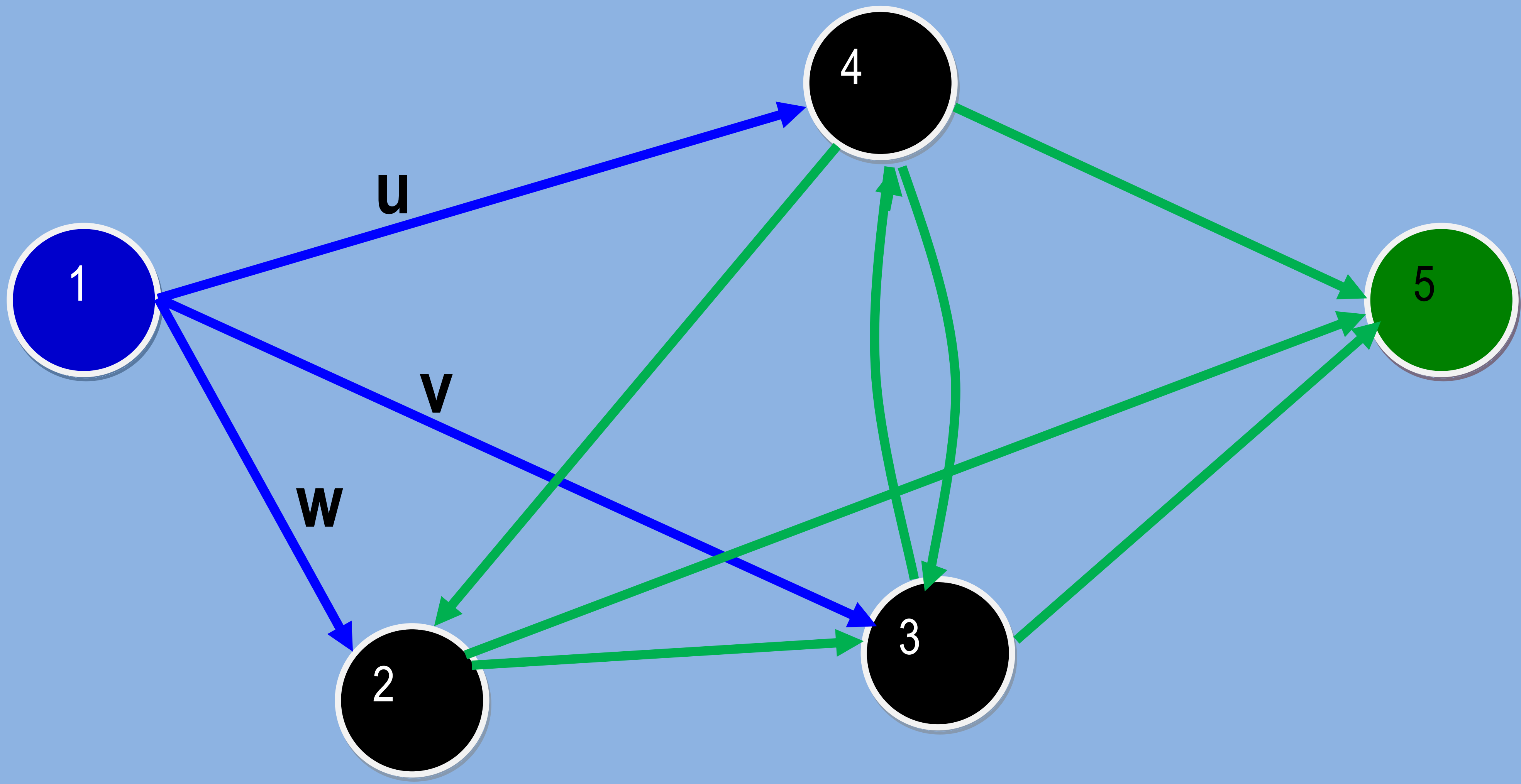


# Degré d'un nœud

Le **nombre d'arcs** incident à x vers l'extérieur est appelé **demi-degré positif**.

On note **$d°^{+}(x)$** le demi-degré positif de x.

Les arcs **incidents** au nœud **1** vers l'**extérieur** sont: **u, v** et **w**.



On écrira :

$$\mathbf{d°^{+}(1)} = 3$$

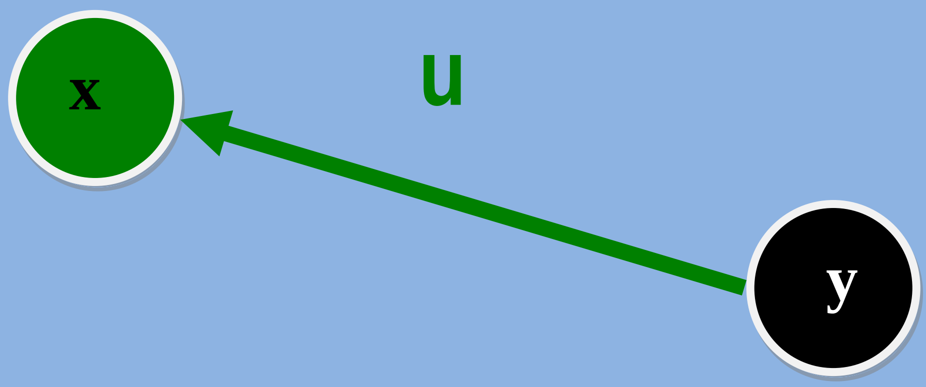
**Aucun** arc n'est incident au nœud **5 vers l'extérieur**.

On écrira :

$$\mathbf{d°^{+}(5)} = 0$$

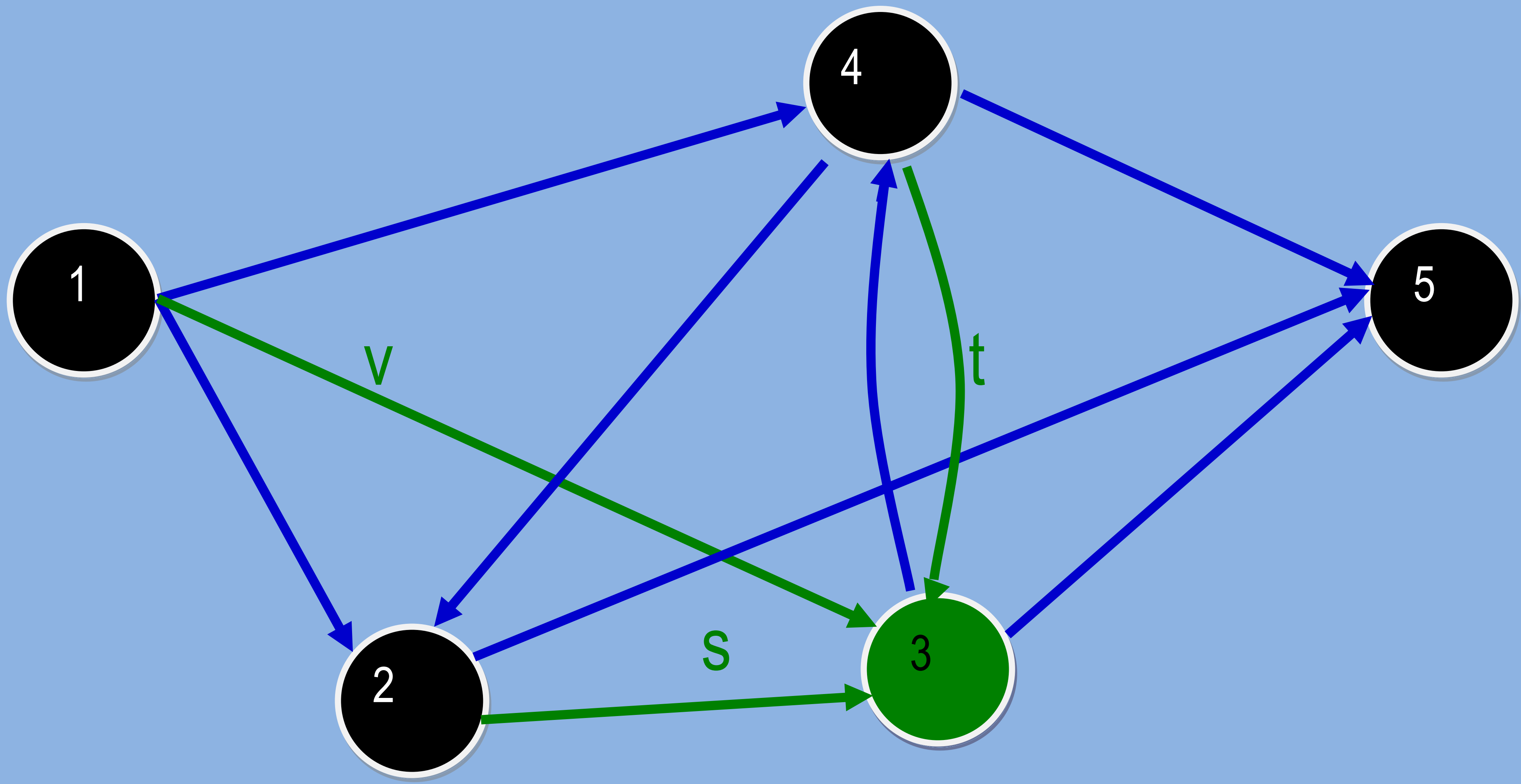
On définit symétriquement les notions :

- d'arc **incident vers l'intérieur**,
- de **demi-degré négatif**, noté **d°⁻(x).**



Les arcs incidents à **3** vers l'**intérieur** sont: **s,t,v**

$$d°^{-}(3) = 3$$



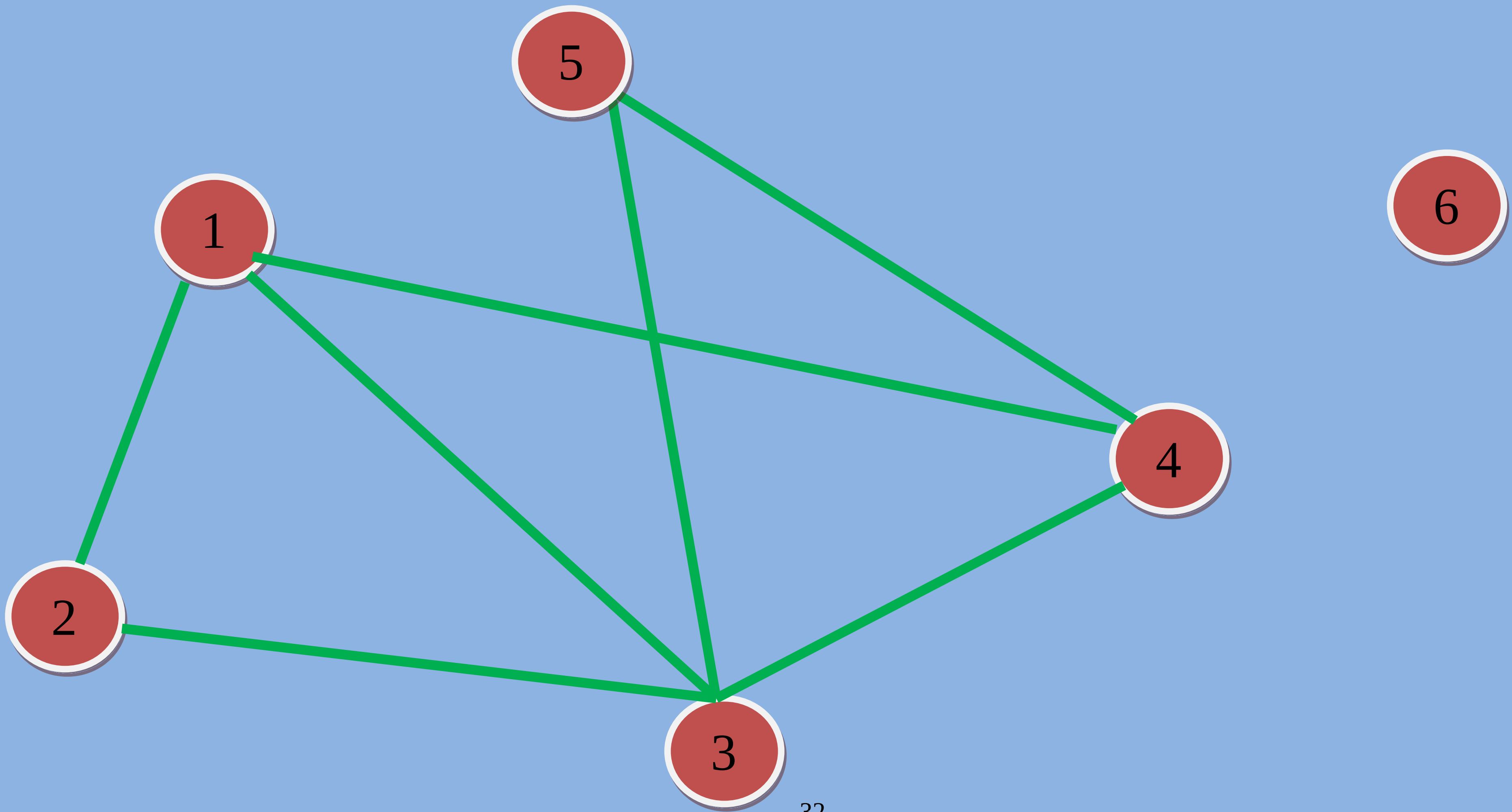
Dans le cas d'un **graphe orienté**, on :

$$\forall x \in S \quad \mathbf{d°(x) = d°^{+}(x) + d°^{-}(x)}$$

**d°(x)** est appelé **degré** du nœud x.

# Cas de graphe non orienté

Dans un graphe non orienté, le **degré** d'un nœud **x** est égal au **nombre d'arêtes** ayant pour extrémité **x**.



# Calcul des degrés des nœuds du graphe

| x | **d°(x)** |
|---|---|
| 1 | 4 |
| 2 | 2 |
| 3 | 4 |
| 4 | 4 |
| 5 | 2 |
| 6 | 0 |

# Cycle d'un graphe

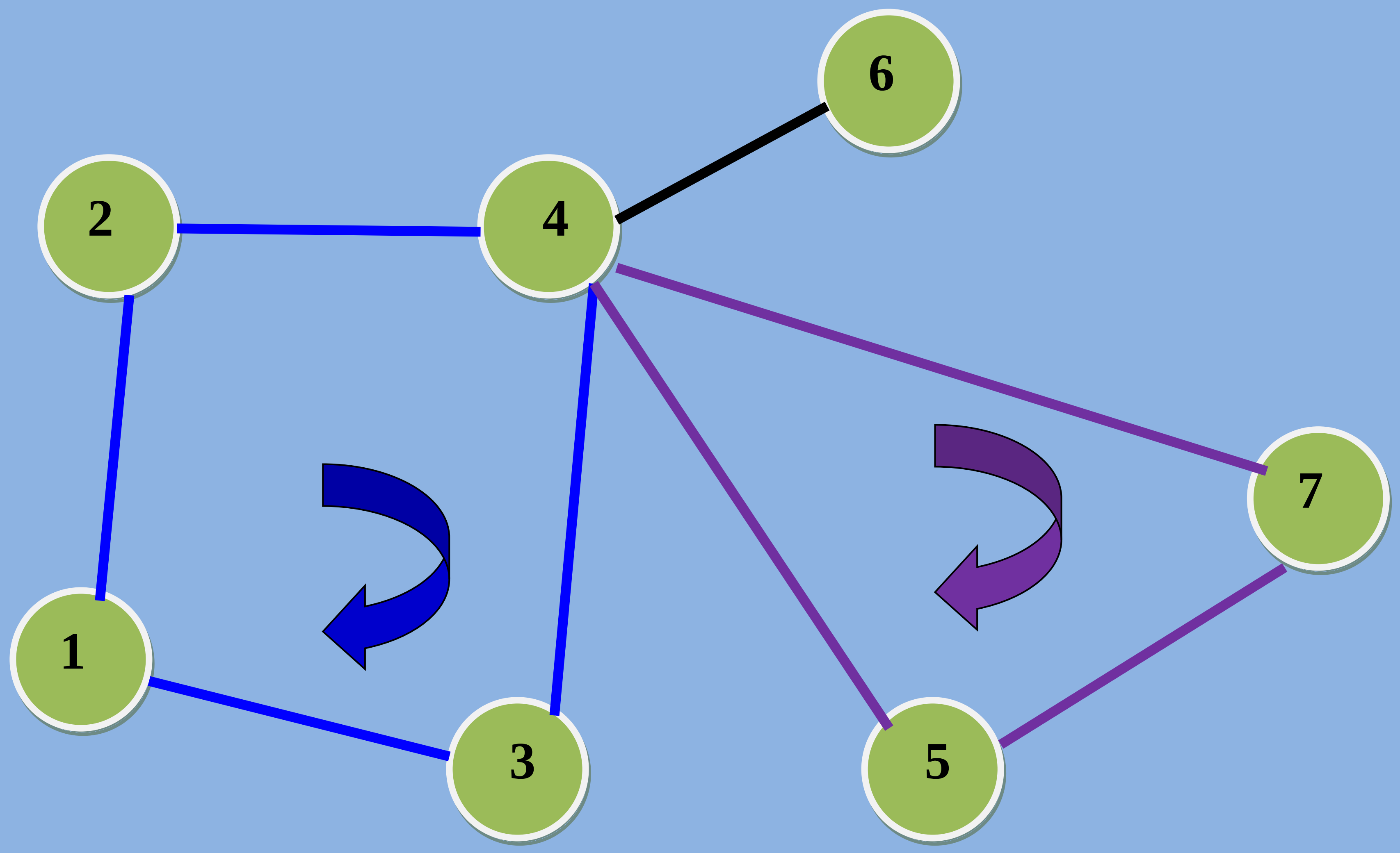
Dans un graphe non orienté G, une chaîne

$( S_0, S_1, \dots, S_{\mathbf{n}})$

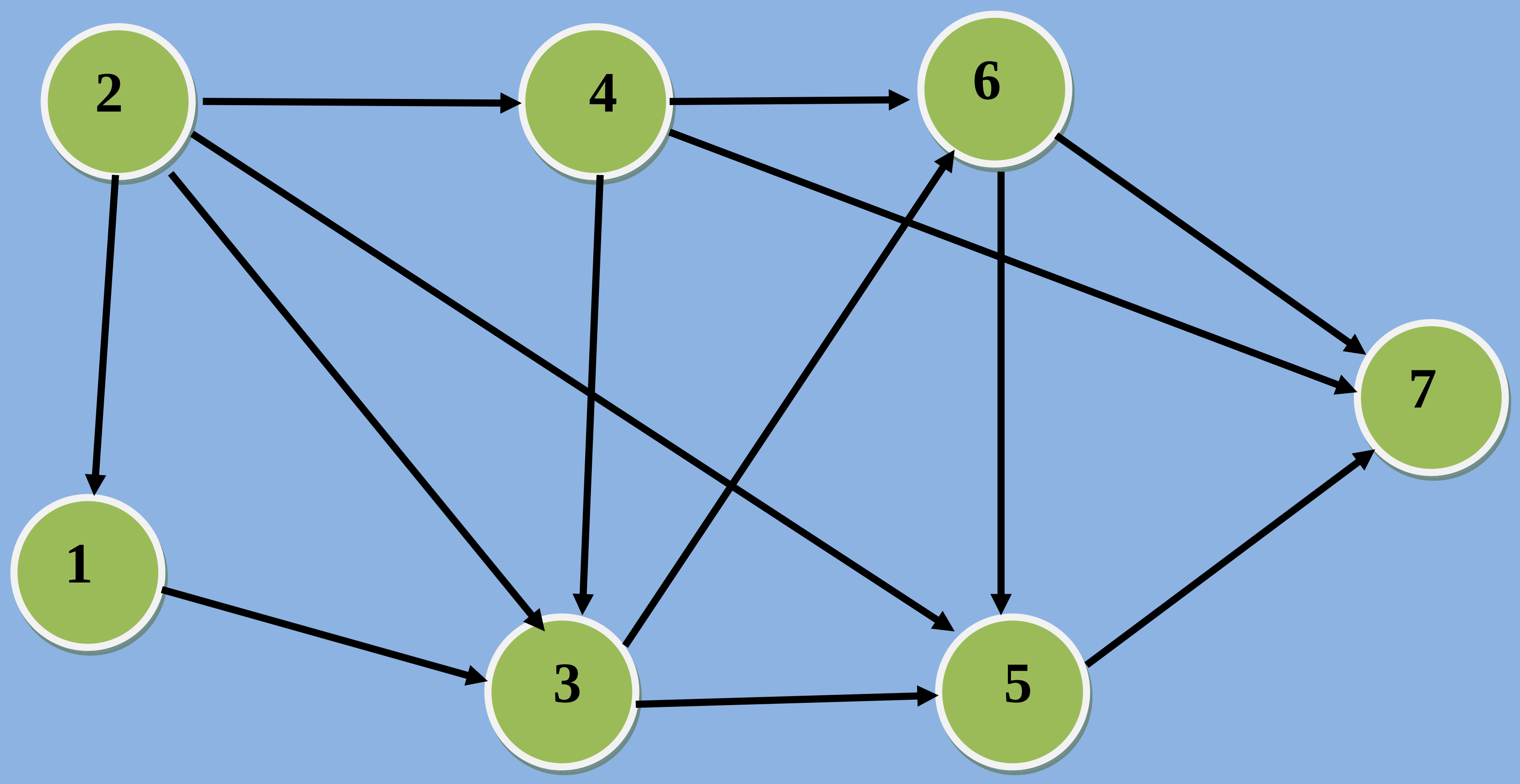
dont:

- les n arêtes sont distinctes deux à deux,
- les deux noeuds $S_0$ et $S_\lambda$ coïncident,

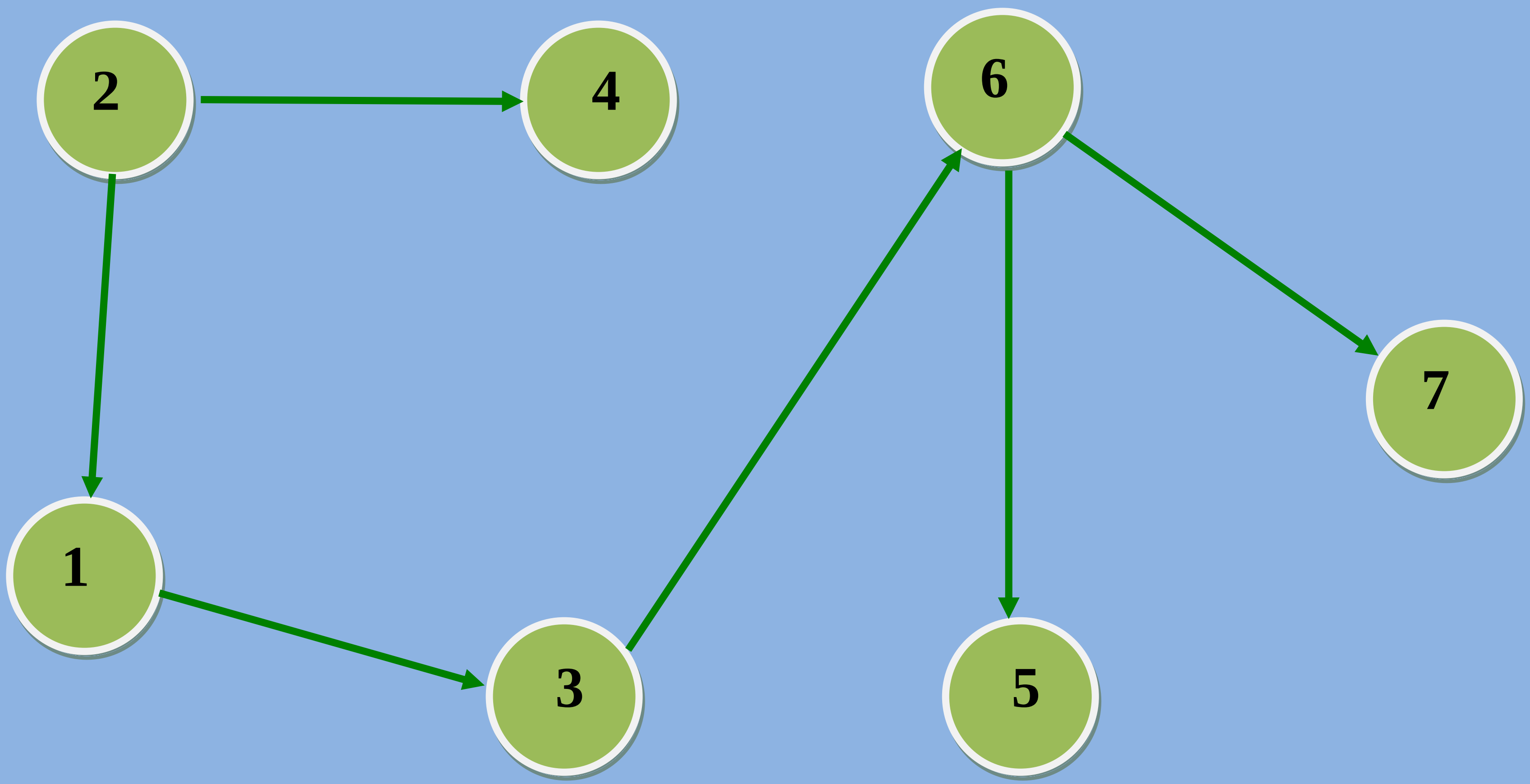
est un **cycle**.


6
2
4
7
1
3
5

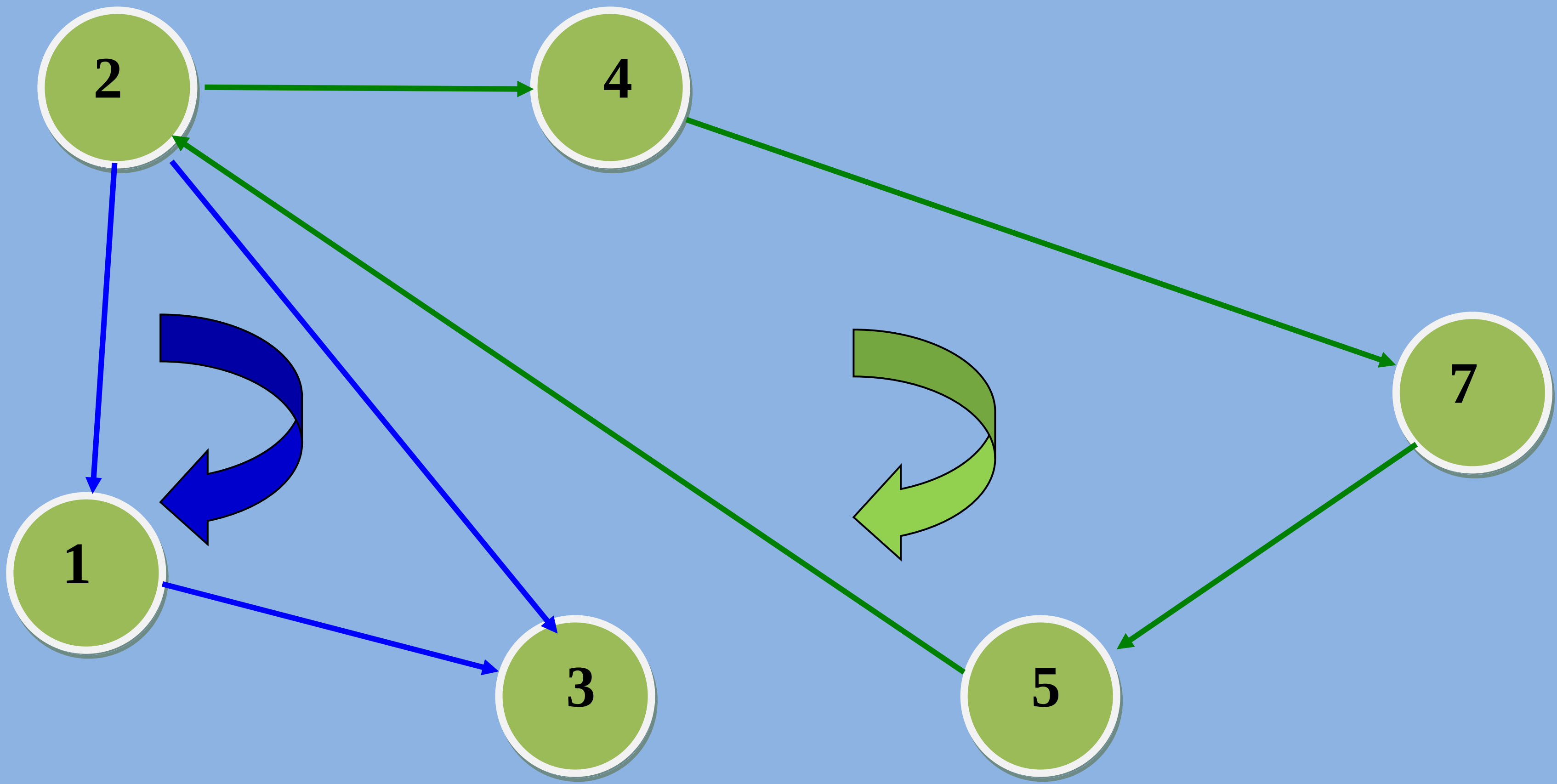
Soit le graphe orienté **visualisé** comme suit :



On peut en extraire les 2 graphes suivants :




Graphe orienté sans cycle : c'est un **Arbre**




Graphe orienté comportant 2 cycles

# Opérations de base sur le graphe

- Toujours commencer par construire le **graphe vide**

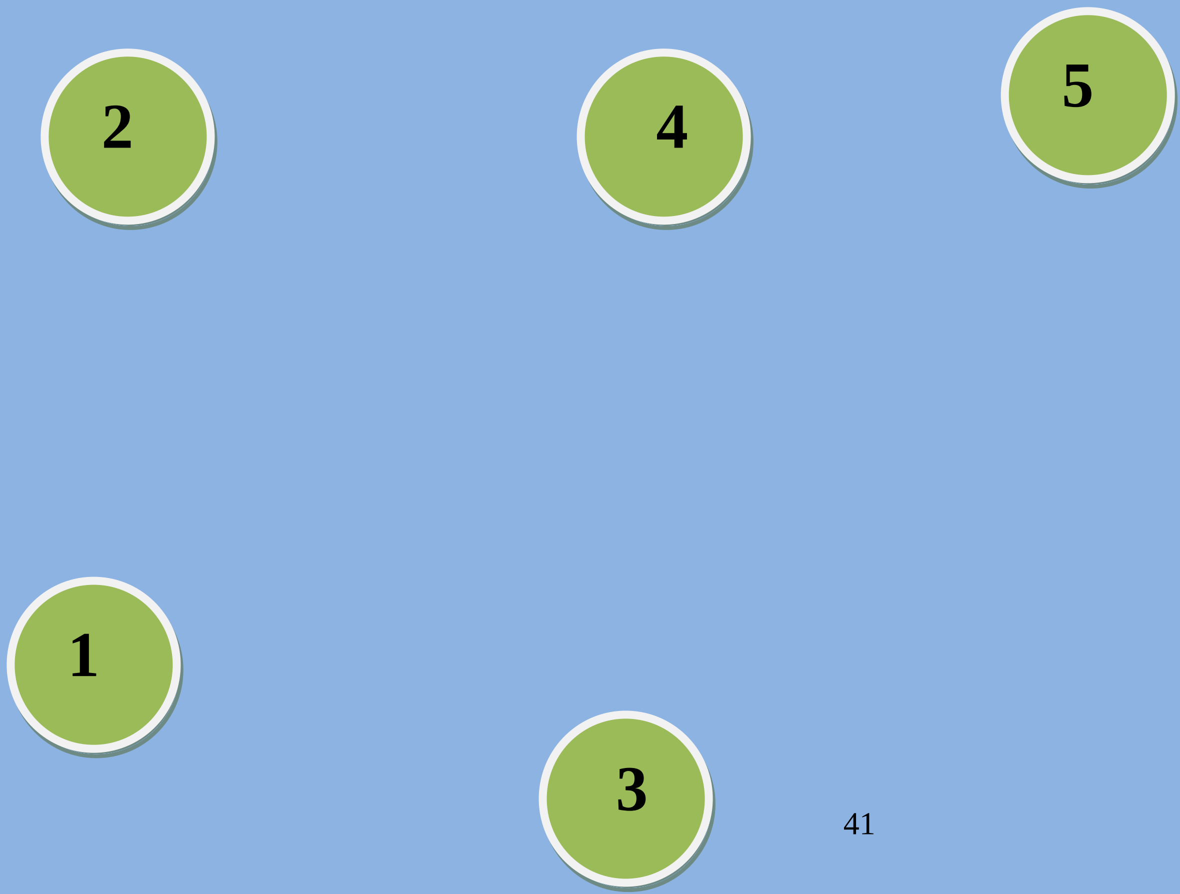
- Ensuite **ajouter** les **noeuds** et les **arcs** nécessaires

- Pour la mise à jour, on peut **supprimer** des **nœuds** ou des **arcs**

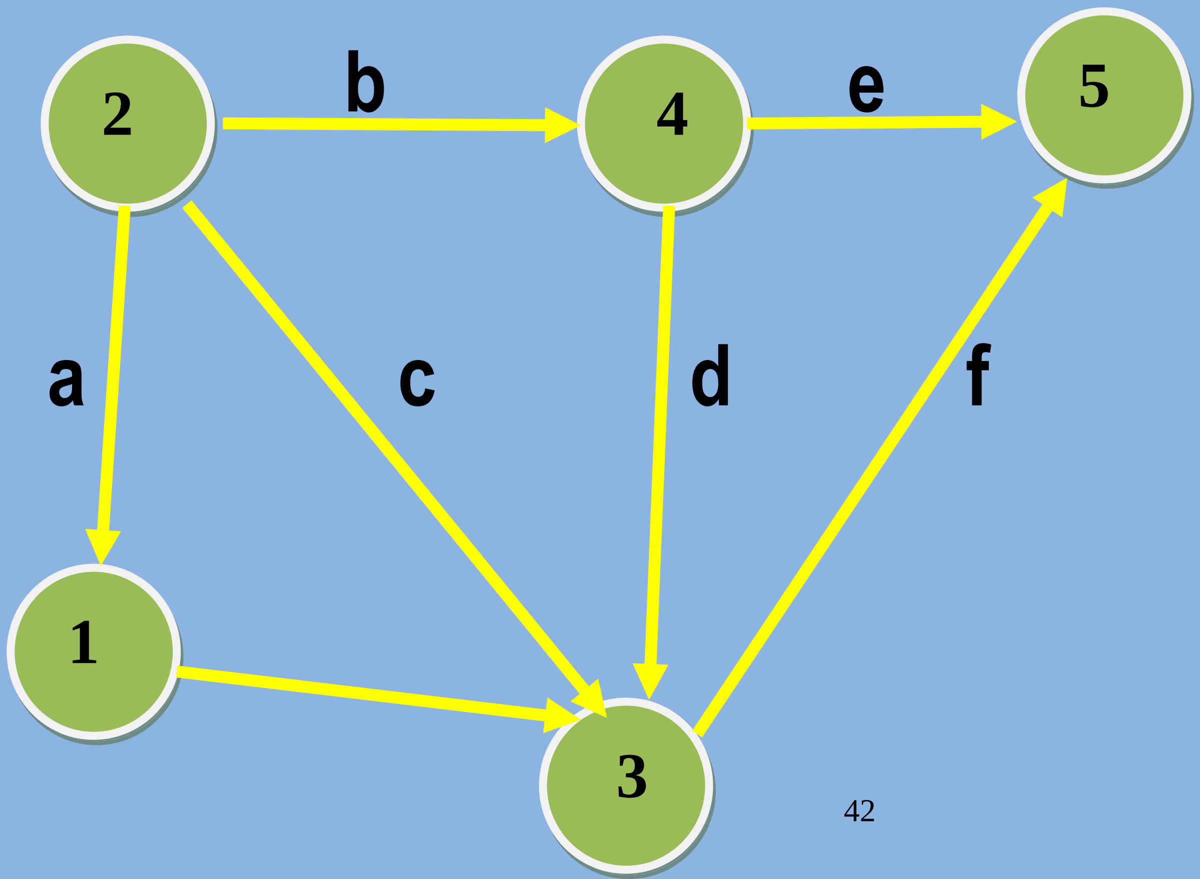
# Graphe de départ

**Graphe vide !**

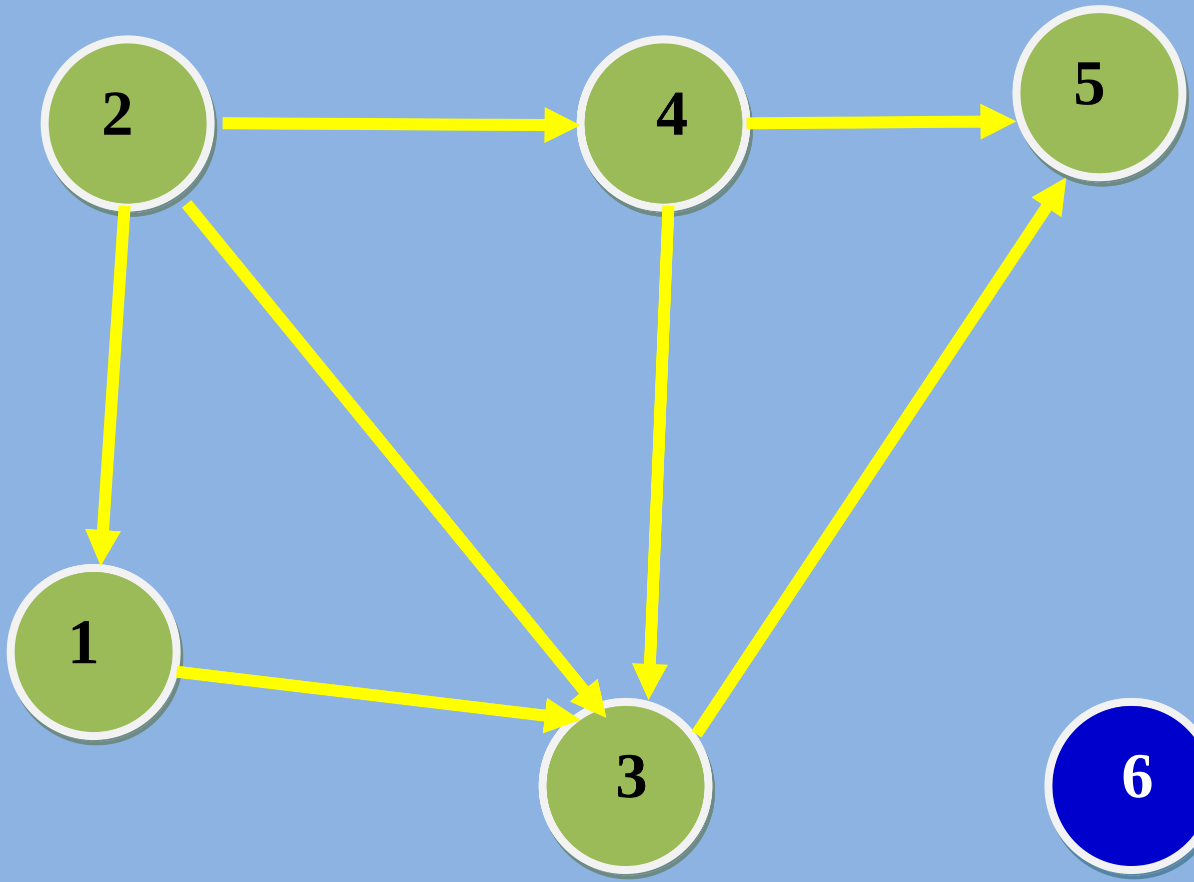
```
addNoeud(1, G)
addNoeud(2, G)
addNoeud(3, G)
addNoeud(4, G)
addNoeud(5, G)
```



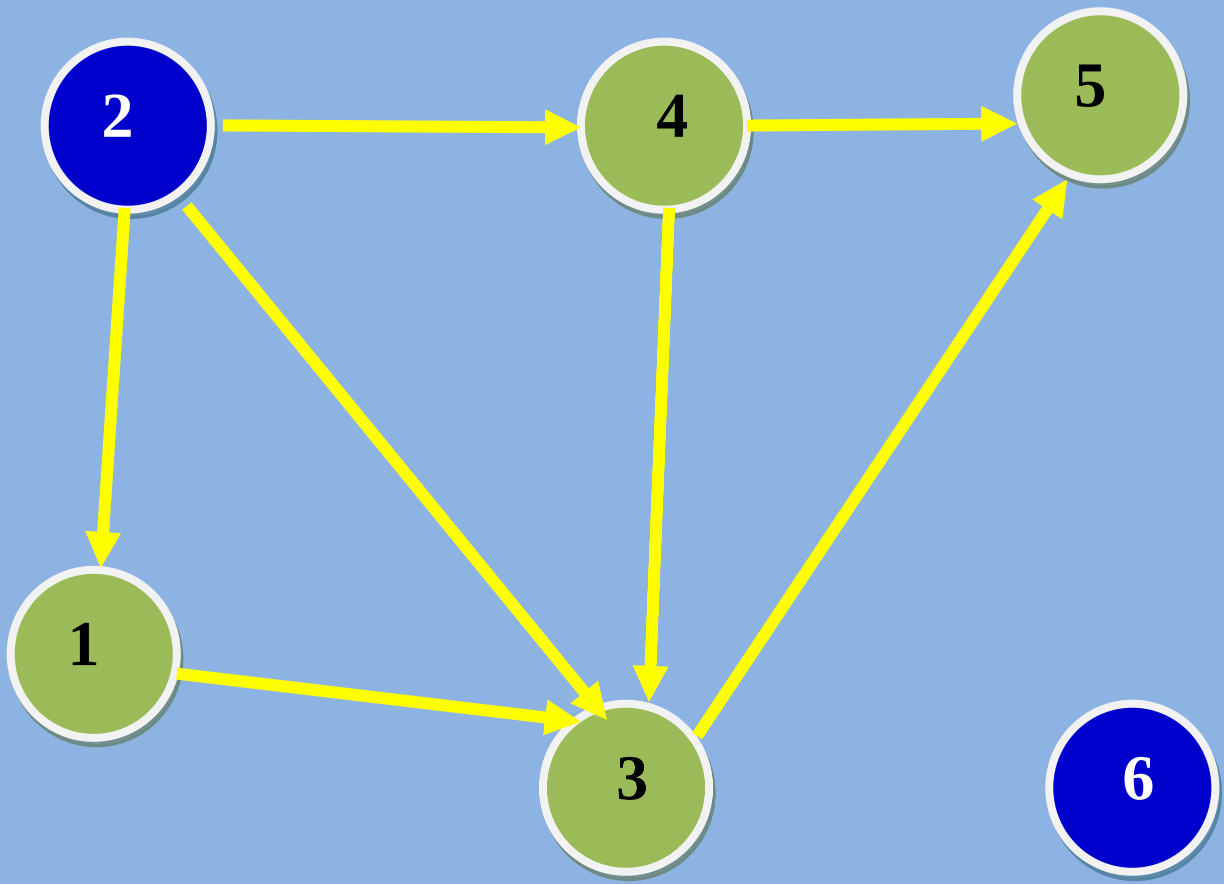
addArc(**2,1,a,G**)
addArc(**2,4,b, G**)
addArc(**2,3,c, G**)
addArc(**4,3,d,G**)
addArc(**4,5,e,G**)
addArc(**3,5,f,G**)



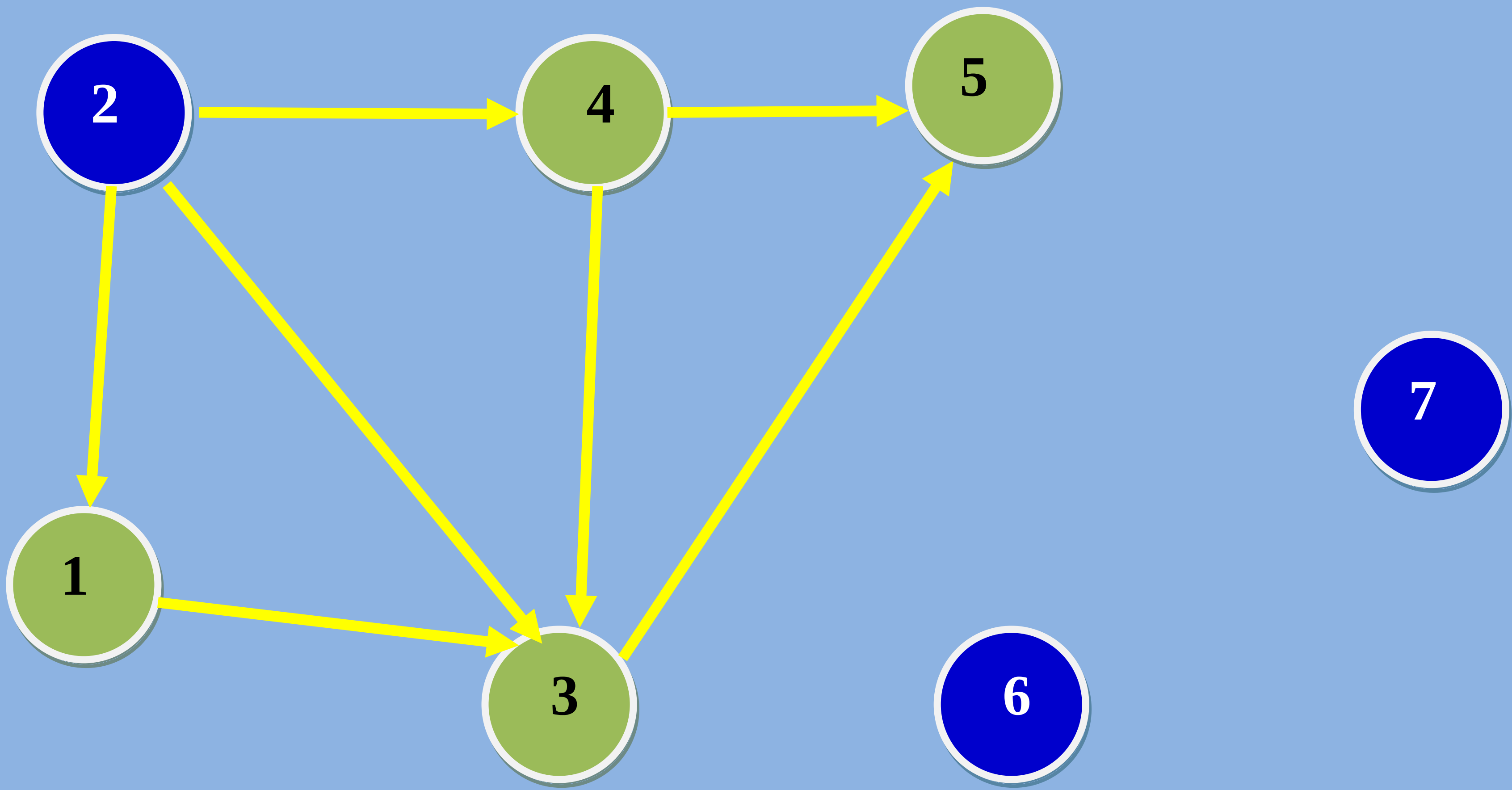
# addNoeud(6, G)



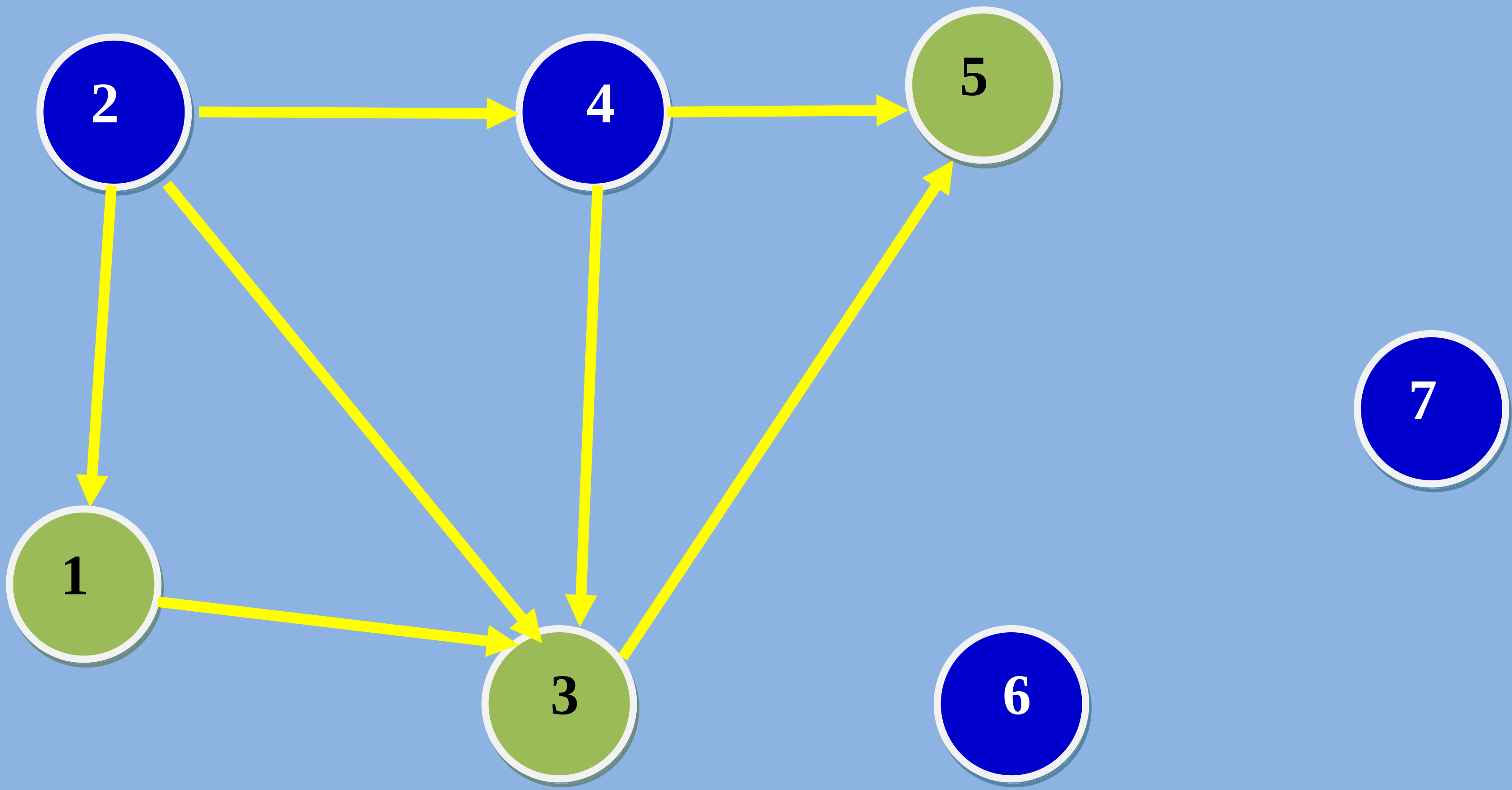
# addNoeuds(**2**, G)



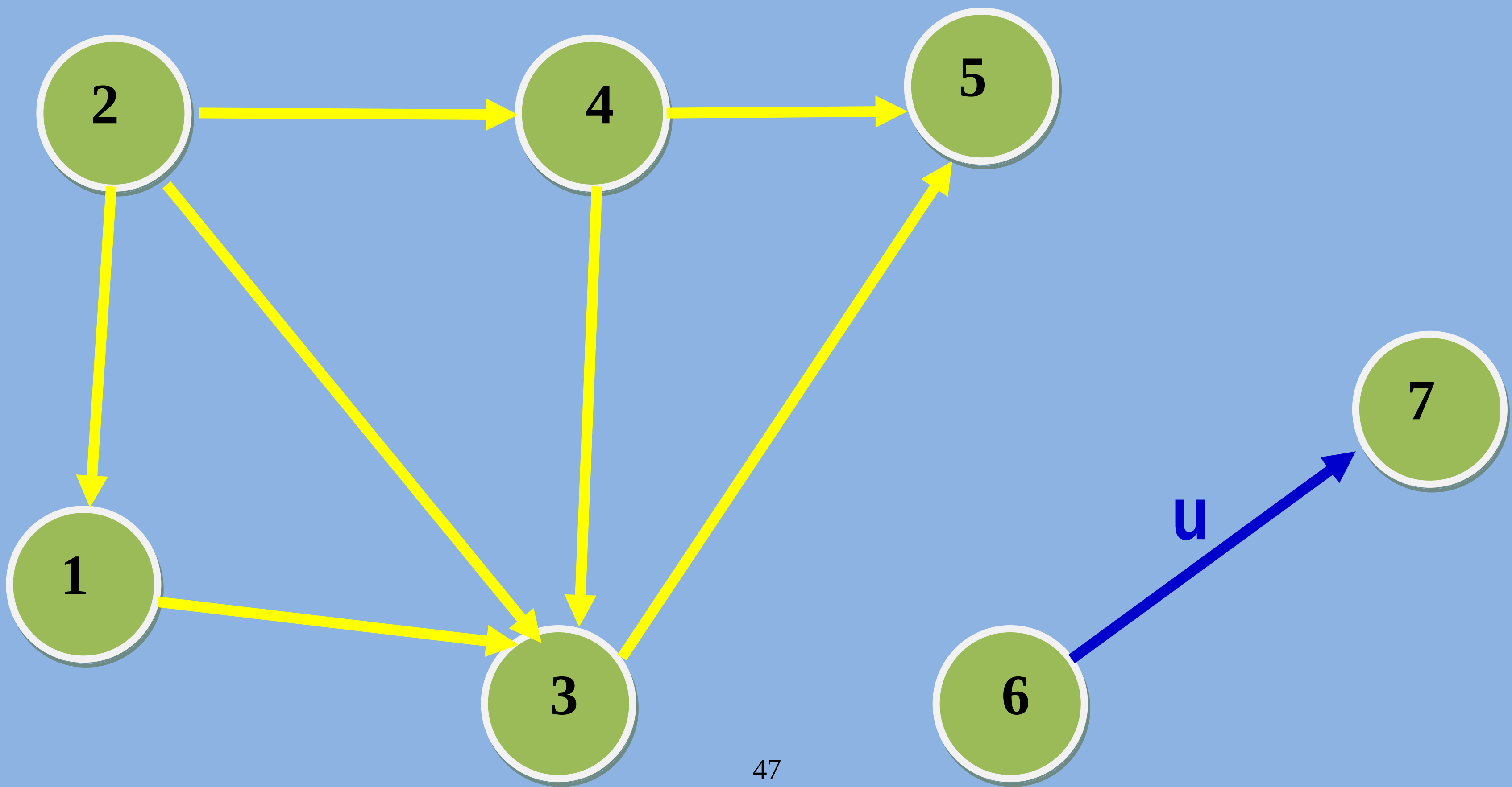
# addNoeuds(7, G)



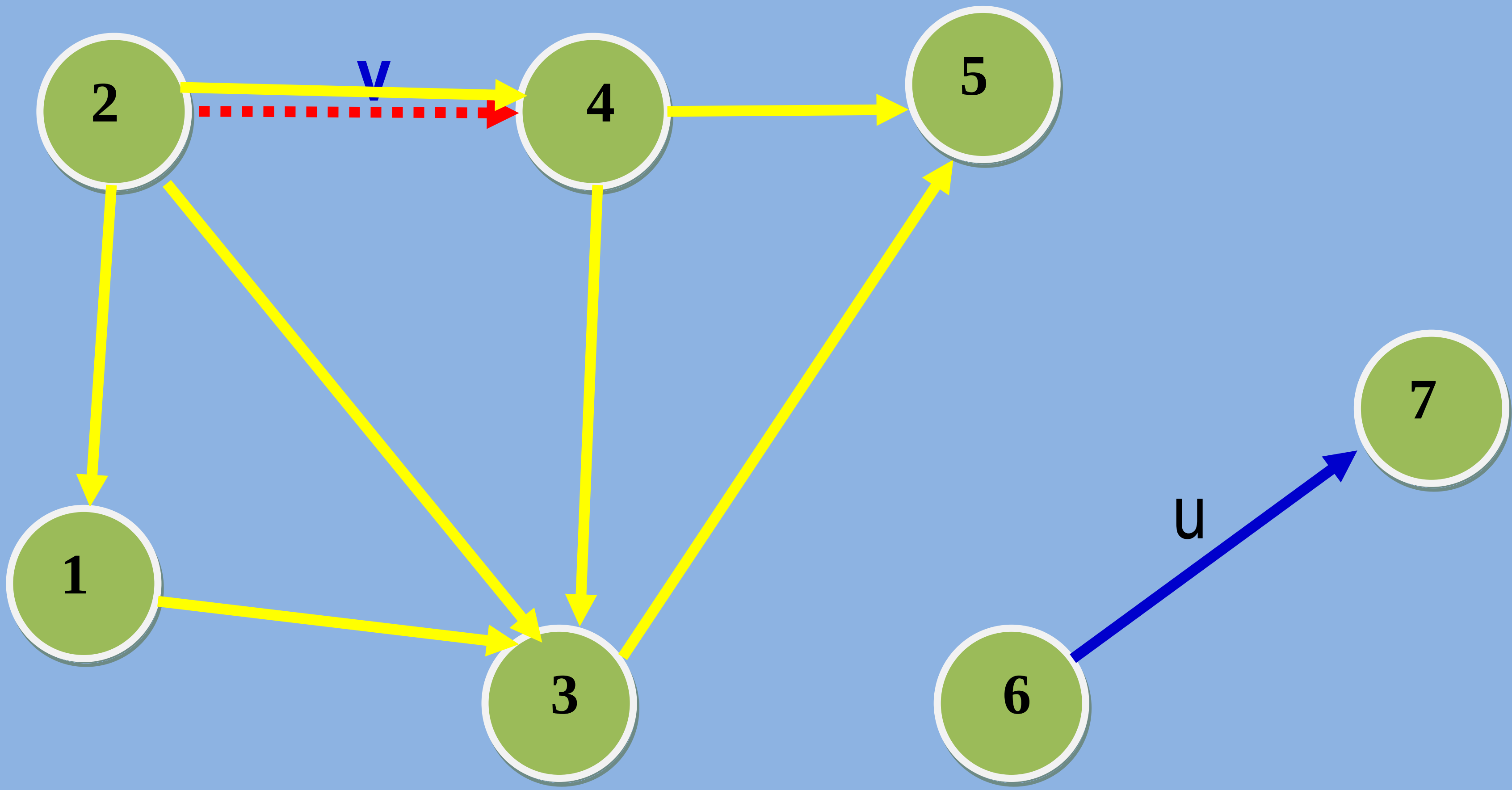
# addNoeuds(4, G)



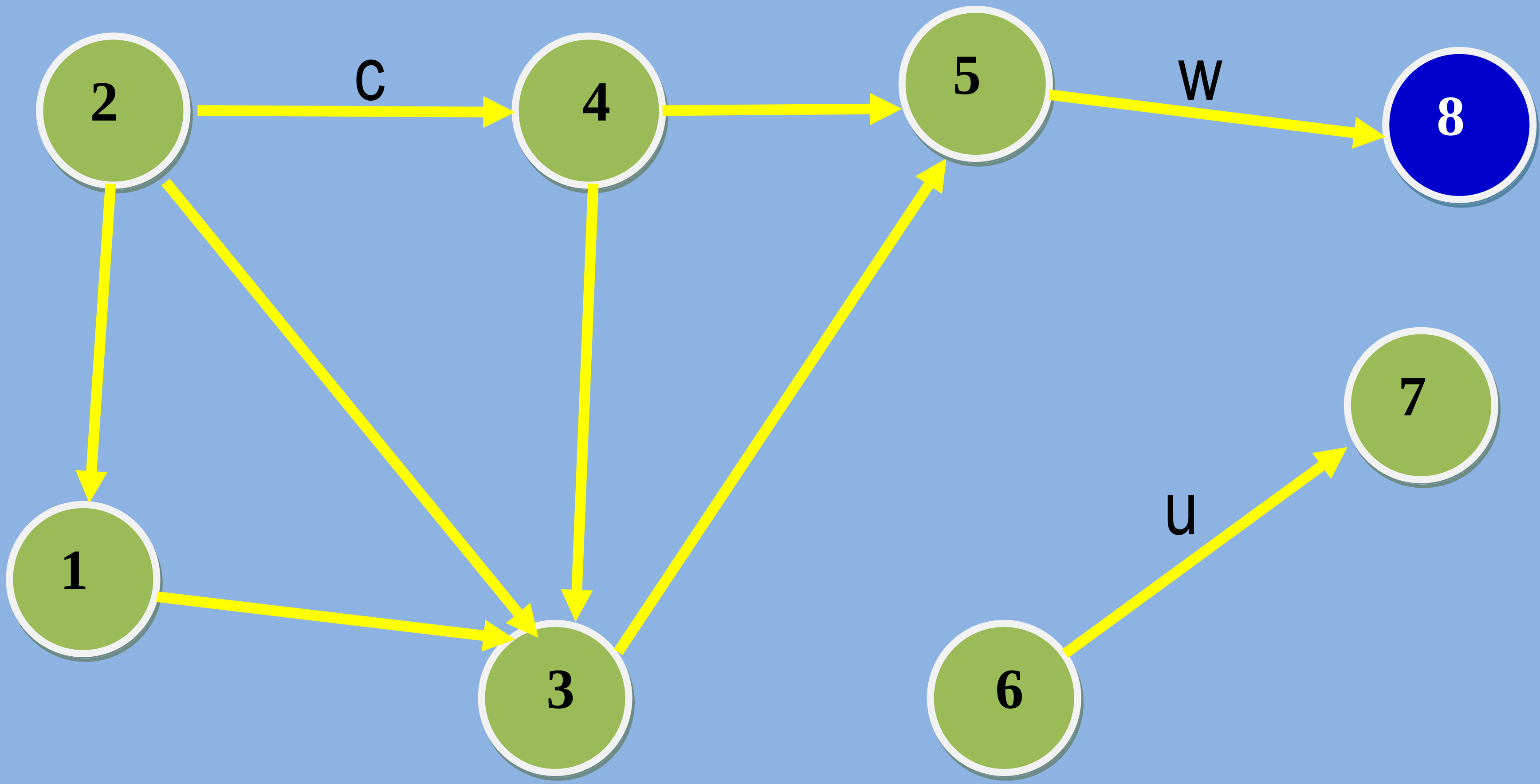
addArc(6,7,**u**, G)



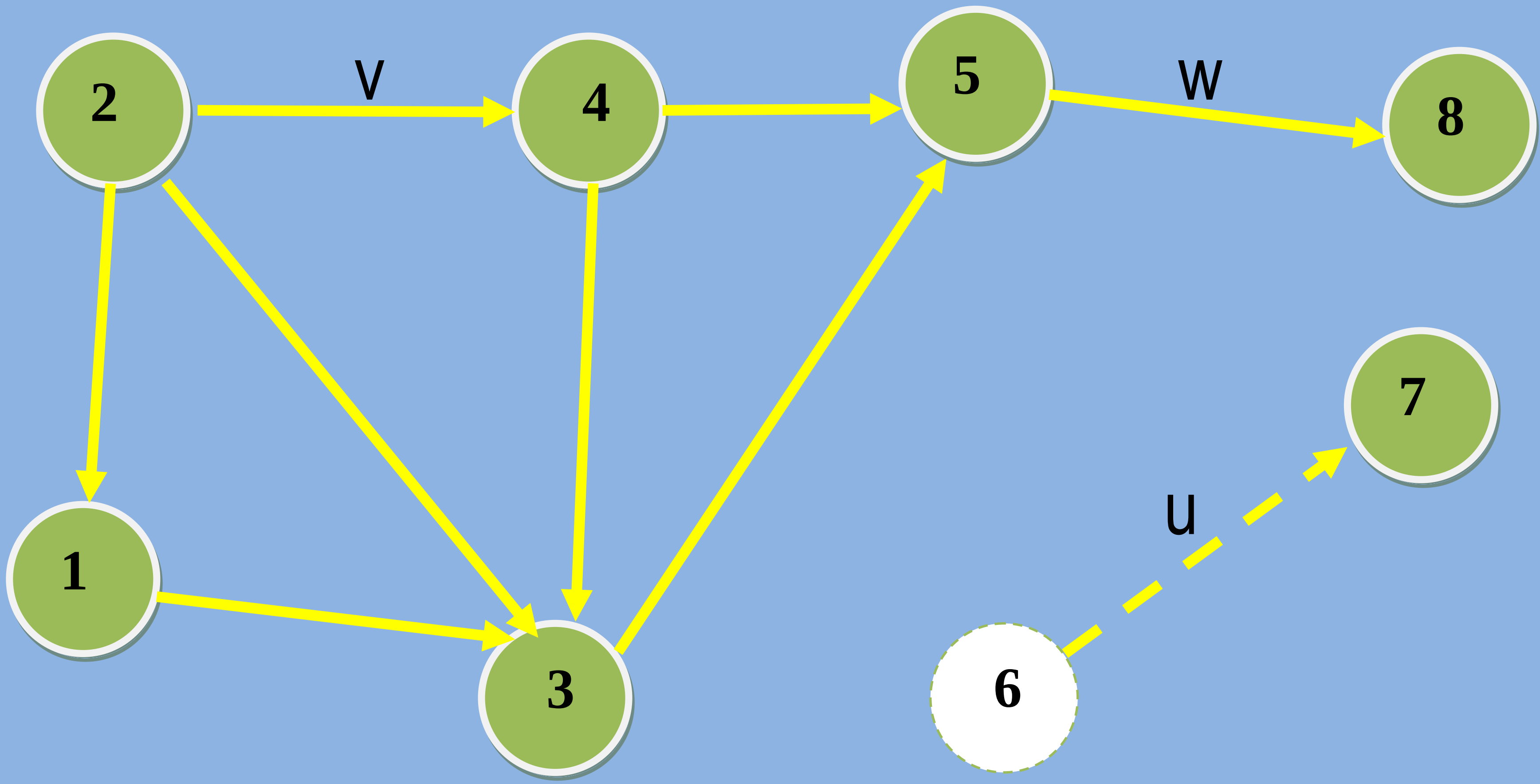
# addArc(2,4,**v**, G):opération non autorisée



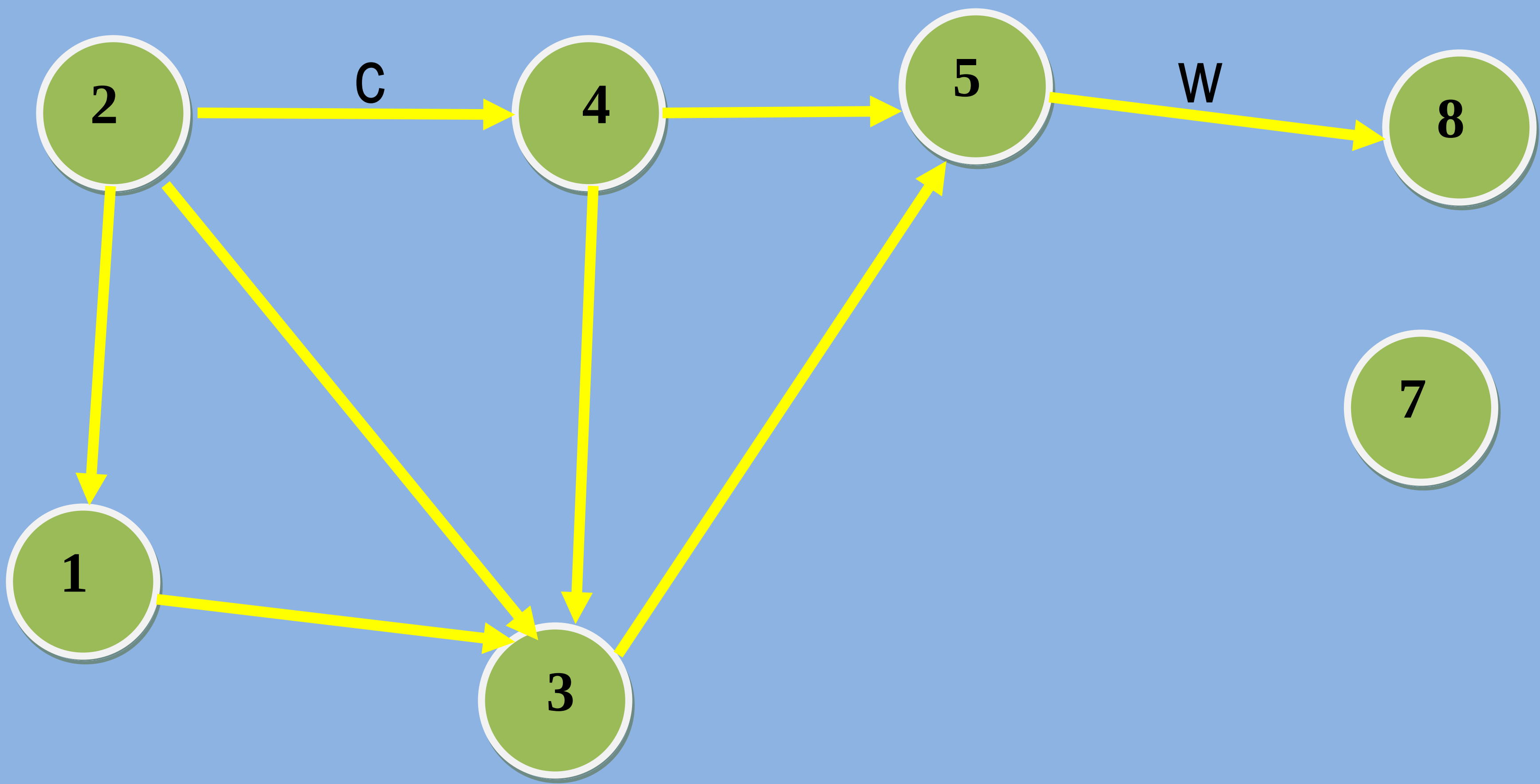
# addArc(5,8,w, G)



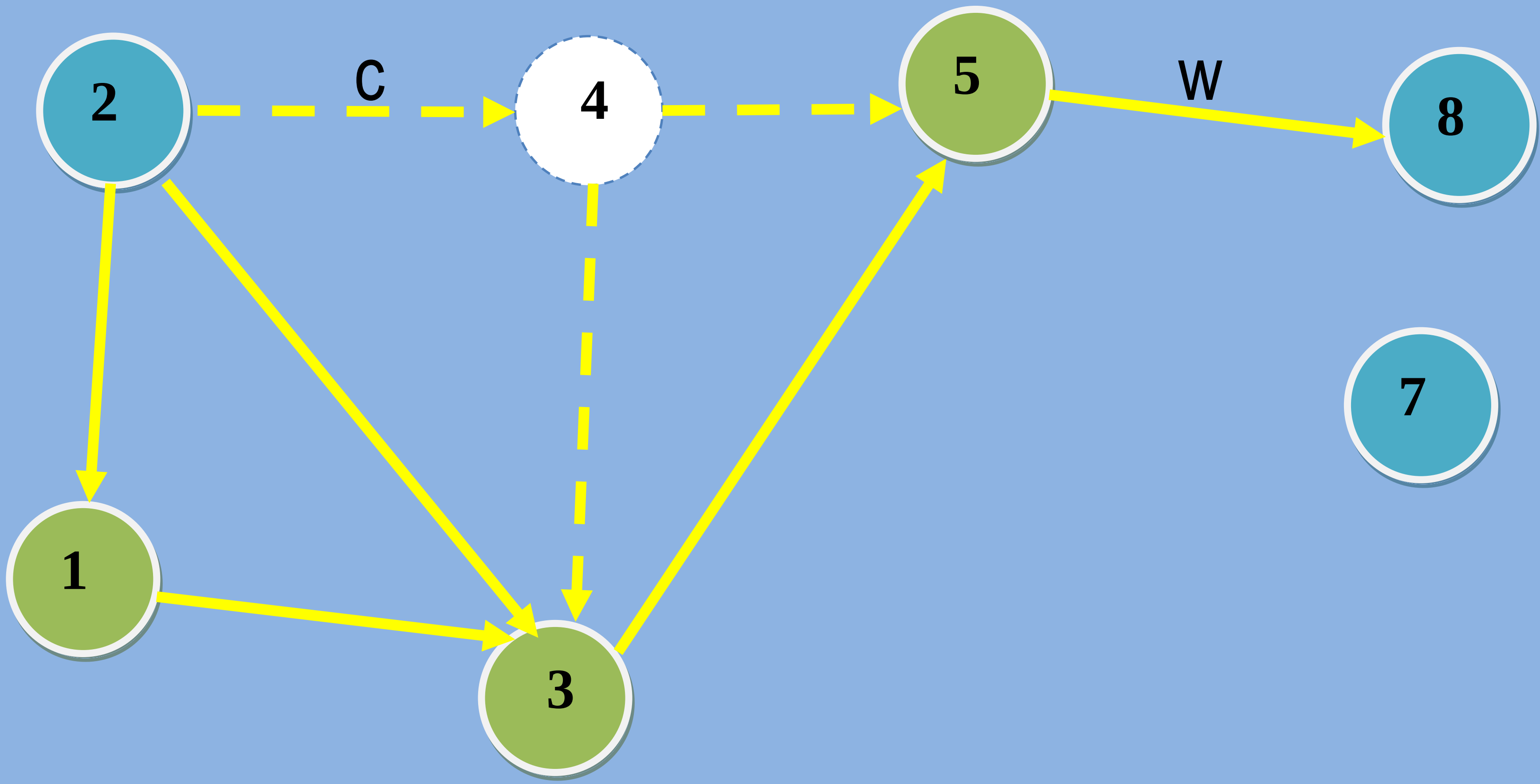
# suppNoeud(6, G)



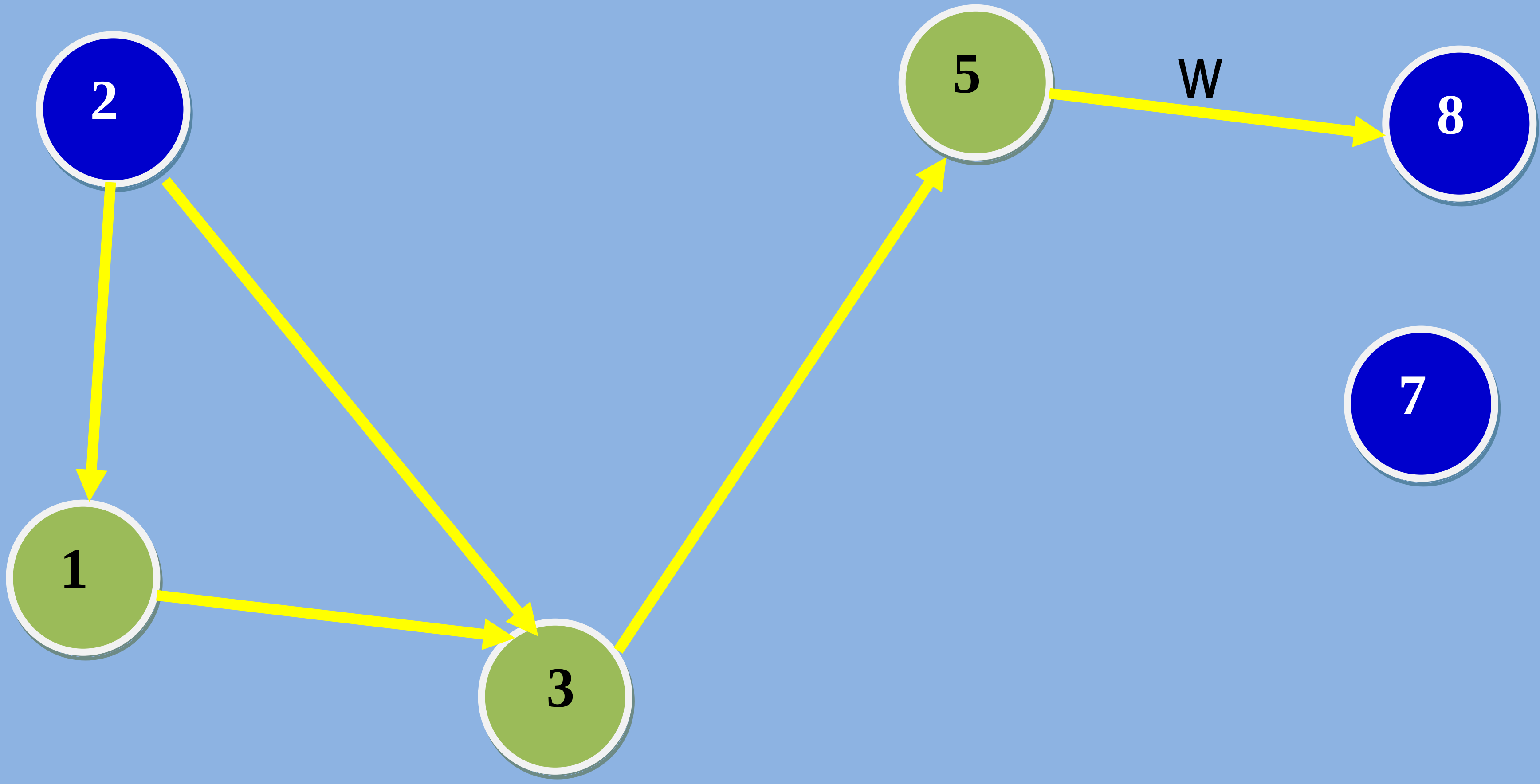
# suppNoeud(**9**, G)



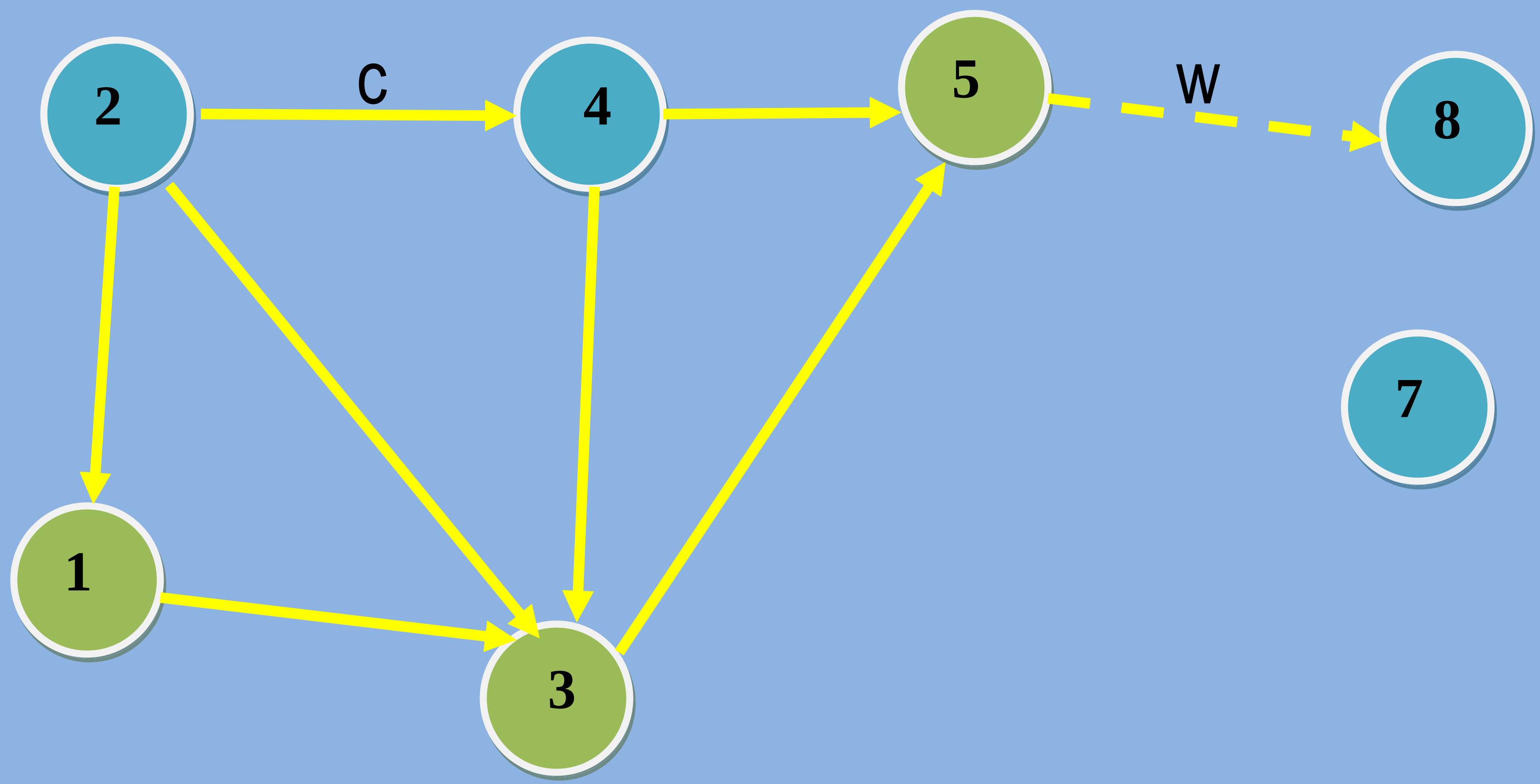
# suppNoeud(4, G)



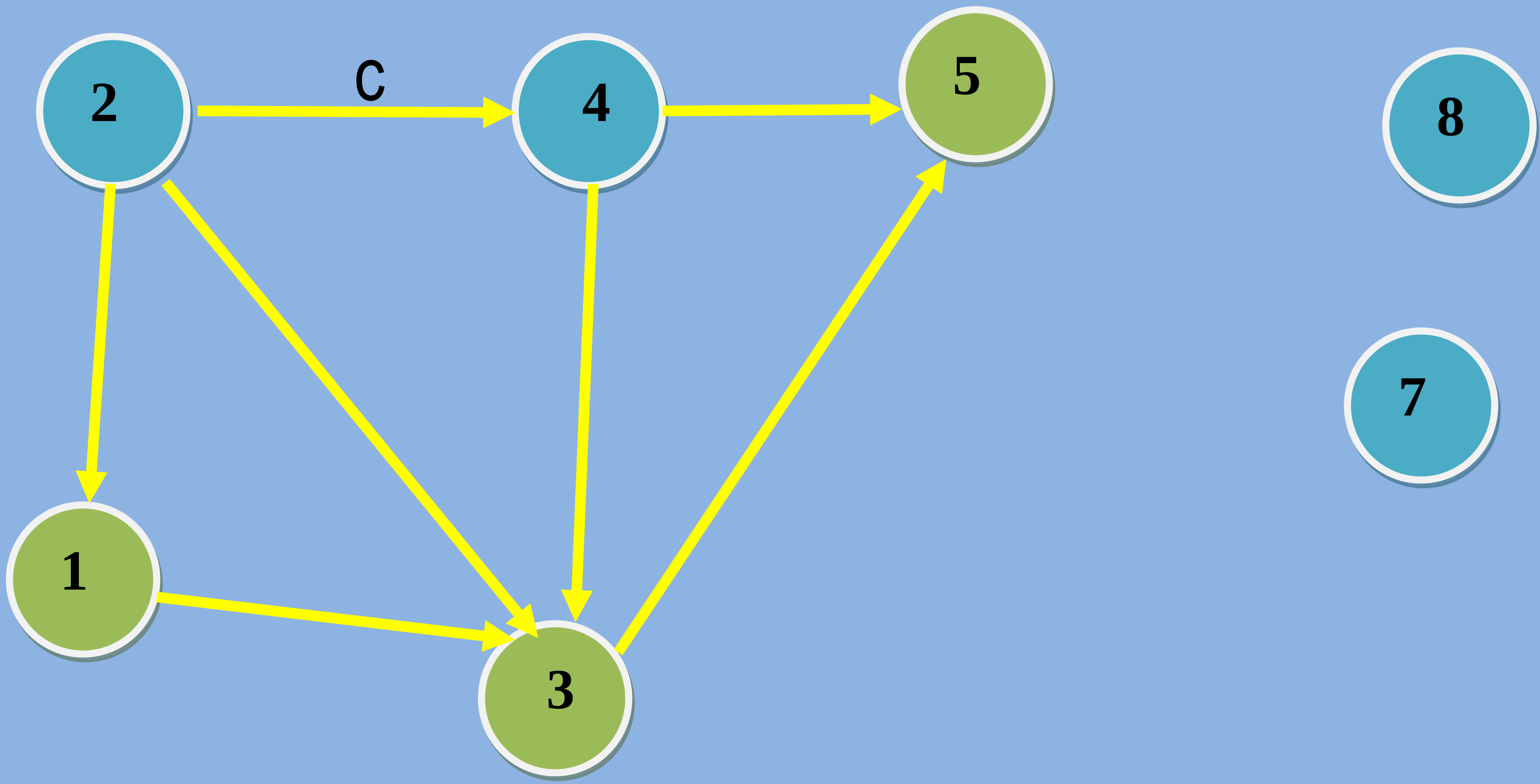
# Graphe résultant



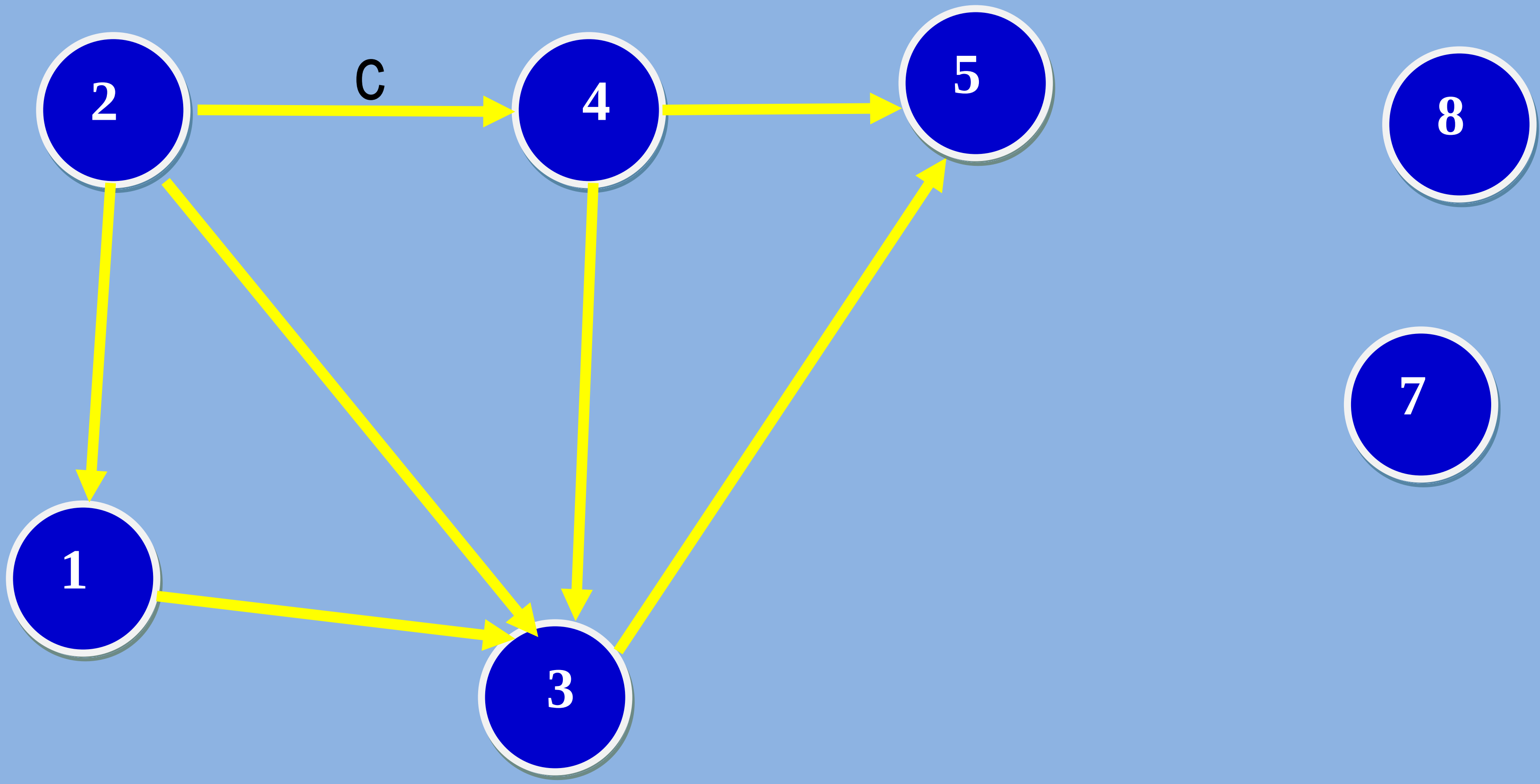
# suppArc(5,8,w, G)

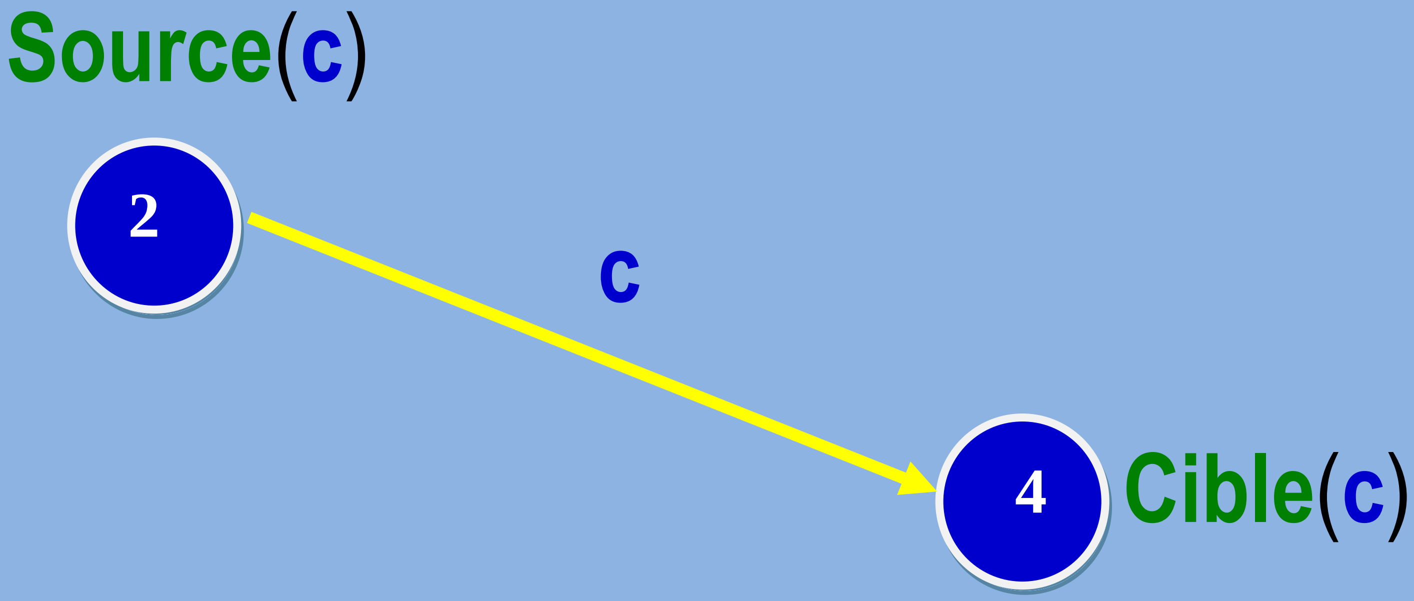


# suppArc(7,8, x, G)



# II- TYPE GRAPHE ORIENTE




Source(c)
2
c
4
Cible(c)

Une spécification minimale peut être établie comme suit :

```
library libraryGrapheOriente

spec GRAPHE[sort Noeud] [sort Arc] =
generated type Graphe ::=  grapheVide  |
                           addNoeud(Noeud; Graphe) |
                           addArc(Noeud; Noeud; Arc; Graphe) ?
```

**preds**

%% prédicats exprimant les **propriétés fondamentales** d'un graphe %

**estNoeudDe** : Noeud × Graphe;

**estArcDe** : Arc * Graphe

%% Opérations très utiles

**ops**

**supNoeud**: Noeud * Graphe -> Graphe;

**supArc**: Arc * Graphe -> Graphe;

**source**: Arc * Graphe ->? Noeud;

**cible**: Arc * Graphe ->? Noeud

```
forall  n0, n1, s0,t0,s1,t1,s2,t2:Noeud;
     e0, e1, e2 : Arc;
     g0, g1: Graphe


%% domaines des opérations partielles

 . def addArc(s0,t0, e0, g0)  <=>    not estArcDe(e0,g0)
 . def source(e0, g0)         <=>    estArcDe(e0,g0)
 . def cible(e0, g0)          <=>    estArcDe(e0,g0)
```

%% un noeud **n0** appartient-il  à un graphe  **g** ?

.**not** **estNoeudDe**(n0, **grapheVide**)
.**estNoeudDe**(n0, **addNoeud**(n1,g0))  <=> n0=n1 \/  **estNoeudDe** (n0,g0)
.**estNoeudDe** (n0, **addArc**(s0, t0, e0, g0))     <=>  n0=s0 \/ n0=t0 \/
**estNoeudDe**(n0,g0)
.**estNoeudDe**(n0, **supNoeud**(n1,g0))          <=>  **estNoeudDe**(n0,g0)  ∧
not(n0=n1)
.**estNoeudDe** (n0, **supArc**(s0, t0, e0, g0))     <=>  **estNoeudDe**(n0,g0)

%% un arc **e0** appartient-il à un graphe **g** ?

. **not** **estArcDe**(e0,**grapheVide**)
. **estArcDe**(e0,**addNoeud**(n0,g0)) <=> **estArcDe**(e0,g0)
. **estArcDe**(e0,**addArc**(s1,t1,e1,g0)) <=> e0=e1 \/ **estArcDe**(e1,g0)
. **estArcDe**(e0,**supNoeud**(n0,g0)) <=> **estArcDe**(e0,g0) $\wedge$
**not**( n0=**source**(e0,g0) ) $\wedge$
**not**( n0=**cible**(e0,g0) )

. **estArcDe**(e0,**supArc**(s1,t1,e1,g0)) <=> **estArcDe**(e0,g0) $\wedge$
**not**( e0=e1)

%% **source** d’un arc **e0** d’un graphe **g** ?

. source(e0, addNoeud(n0,g0)) = source(e0,g0)
. source(e1, addArc(s0, t0, e2, g0)) = s0 **when** e1=e2 **else** source(e1,g0)

%% **cible** d’un arc **e0** d’un graphe **g** ?

. cible(e0, addNoeud(n0,g0)) = cible(e0,g0)
. cible(e1, addArc(s0,t0,e2,g0)) = t0 **when** e1=e2 **else** cible(e1,g0)

**end**

# III- REPRESENTATION D'UN GRAPHE

Il existe deux classes de représentations pour les objets de type GRAPHE:

-la **matrice d'adjacence**,
-les **listes d'adjacence**.

# 1- Représentation par matrice d'adjacence

Soit un graphe orienté

$$G = (S,A) \quad \text{tel que} \quad |S| = n$$

La technique est centrée sur la **représentation des arcs** du graphe.

Elle permet de représenter G l'aide d'une matrice **M** appelée **matrice d'adjacence**.

# Calcul de la matrice d’adjacence

Les **lignes** et les **colonnes** de la matrice **M** représentent les **nœuds** du graphe G.

Notons $\mathbf{M_{ij}}$ l'élément appartement :

- à la ligne **i**
- et à la colonne **j** de la matrice M.

M est définie par :

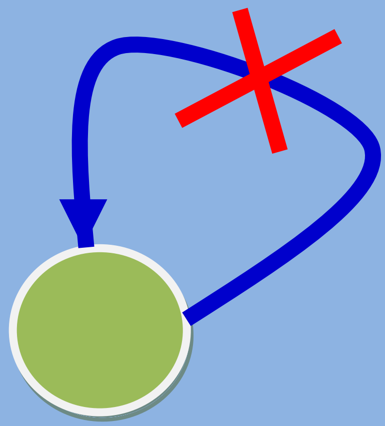
$$M_{ij} = \underline{\textbf{si}}\ \ i \rightarrow j \in A\ \ \underline{\textbf{alors}}\ \mathbf{1}\ \text{sinon}\ \ \mathbf{0}$$

La matrice d'adjacences est une matrice **binaire carrée** d'ordre **n**.

Il n'y a que des zéros sur la diagonale :

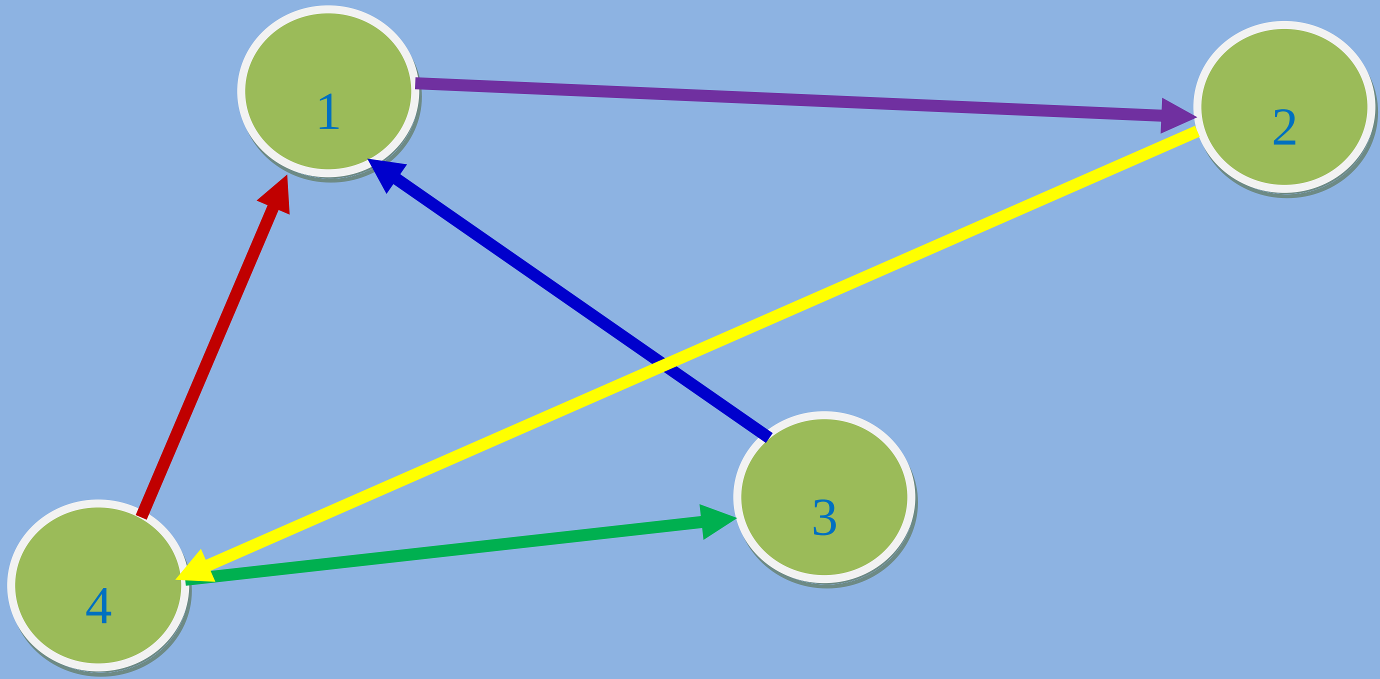
$$M_{ij} = \underline{\textbf{si}}\ i = j\ \ \underline{\textbf{alors}}\ \mathbf{1}$$

La présence d’un 1 sur la diagonale indiquerait une **boucle** : ce qui est interdit par convention.



$M_{ii}=0$

# Le graphe suivant :



est représenté par la matrice :

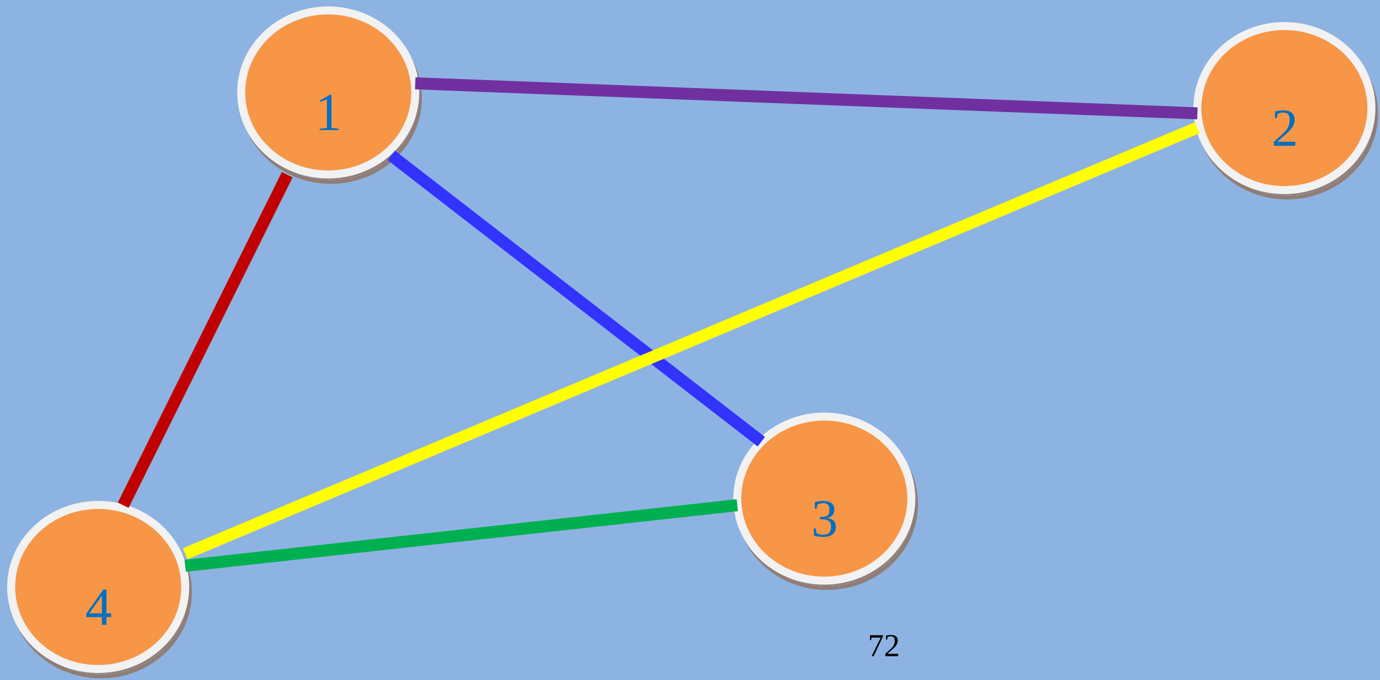
$$M = \begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{array}$$

Remarque :

La matrice d'adjacence d'un graphe **non orienté** est symétrique:

$$M_{ij} = M_{ji}$$

Le graphe suivant:



est représenté par la matrice :

$$M = \begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}$$

## 1.1 Avantages

1- La représentation matricielle est pratique pour **tester l’existence** d’un arc (arête).

2- Il est plus facile d’**ajouter** ou **retirer** un arc (arête).

3- Il est facile de **parcourir** les successeurs ou prédécesseurs d’un nœud.

## 1.2 Inconvénient

1- Il demande **n tests** pour détecter les successeurs ou prédécesseurs d'un nœud **s** quel que soit leur nombre.

2- Il en est de même du **calcul de $d°^+$ et $d°^-$** de s.

3- La **consultation complète** de la matrice de dimension **n** requiert un **temps** d'ordre **$n^2$**.

4- La représentation matricielle exige un **espace mémoire** d'ordre $\mathbf{n^2}$.

5- Cela interdit d'avoir des algorithmes d'ordre inférieur à $\mathbf{n^2}$ pour des graphes à **n** nœuds n'ayant que **peu d'arcs** (arêtes).

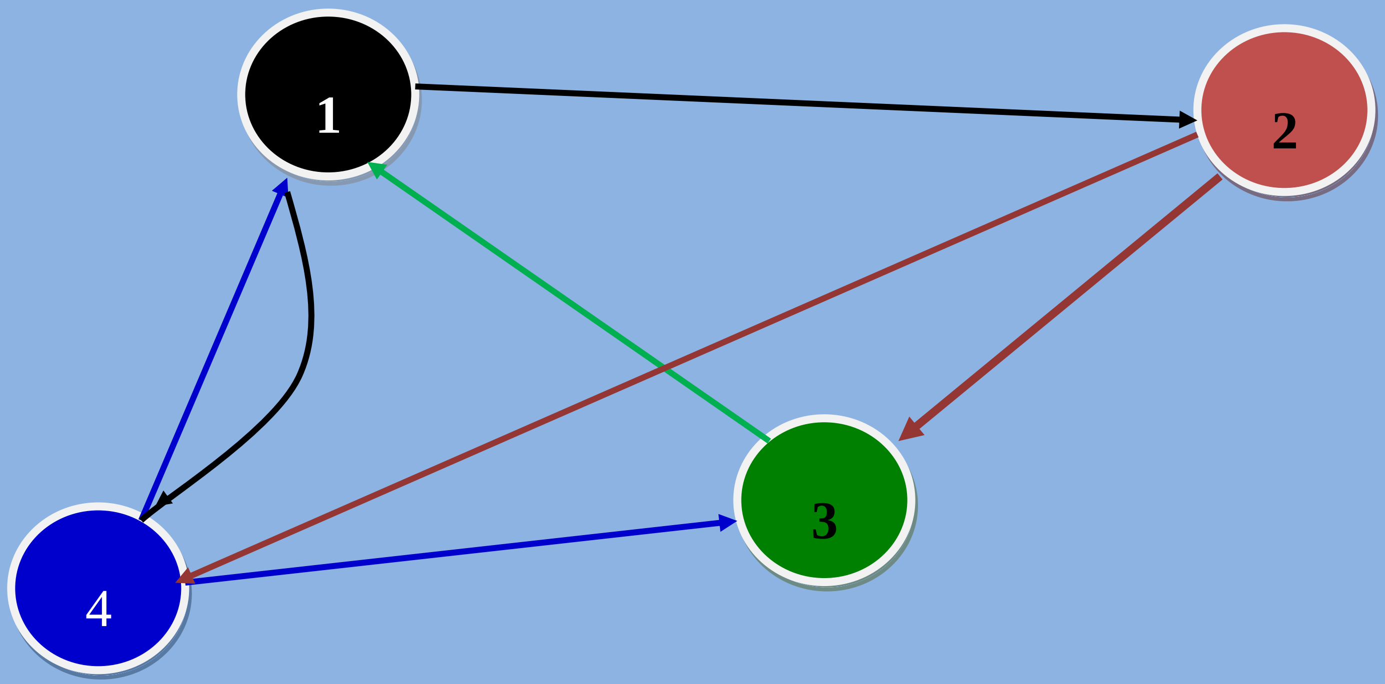
## 2- Représentation par liste d'adjacence

Cette technique est centrée sur la représentation des **nœuds**.

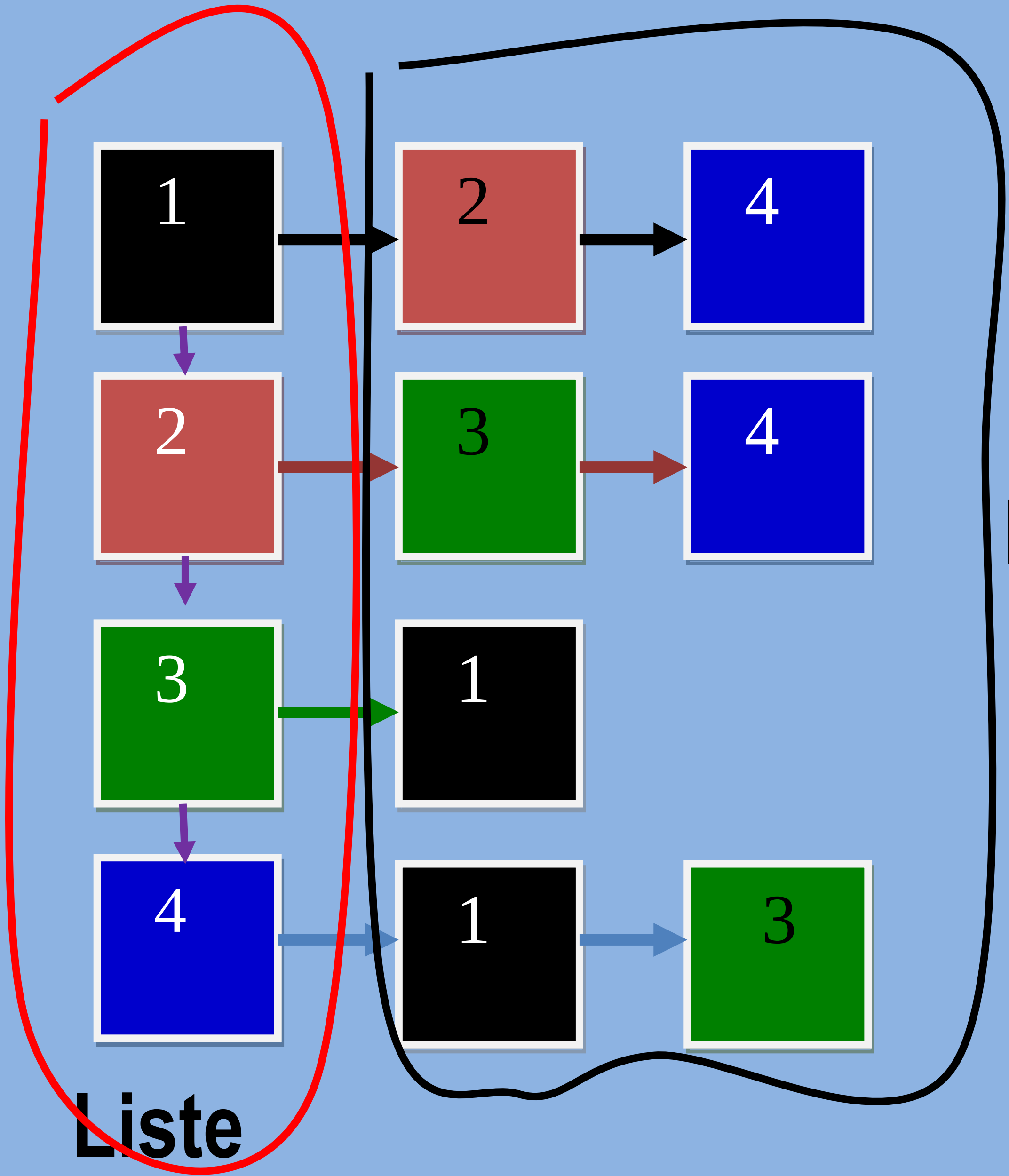
Elle consiste à :

- représenter l’ensemble des nœuds,
- associer à chaque nœud la liste de ses **successeurs** rangés dans un **ordre arbitraire**.

Le graphe suivant



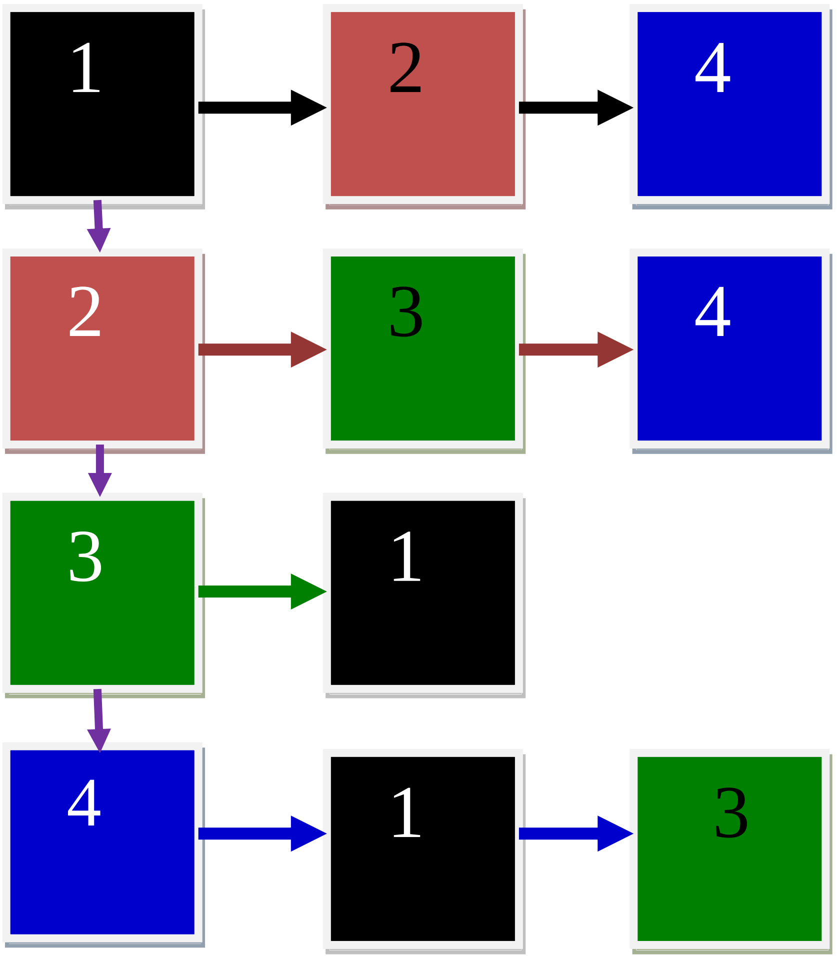

est représenté par les listes suivantes :




**Listes des successeurs**

**Liste des nœuds**

# Représentation d'un graphe par **liste d'adjacence**



Les listes générées par cette représentation sont appelées **listes d'adjacence**.

Si le nombre de nœuds  n'évolue pas, les **listes des successeurs** sont accessibles :

- à partir d'un **tableau**,
- tableau qui contient, pour chaque nœud, un pointeur vers la **tête** de liste de ses successeurs.

## 2.1-Avantages

1- L’espace mémoire utilisé pour un graphe **orienté** avec **n** nœuds et **p** arcs est d’ordre (**n+p**).

2- Dans le cas d’un graphe **non orienté** avec **n** nœuds et **p** arêtes, l’espace mémoire utilisé est d’ordre (**n+2p**).

3-Pour un traitement sur les **successeurs** d'un nœud **s**:

nombre de nœuds visités = $\mathbf{d°^+(s)}$

3- Un algorithme qui traite **tous** les arcs d'un graphe de **p** arcs peut donc être d'ordre **p**.

## 2.2- Inconvenients

1-Pour **tester** s'il existe un arc **x** $\rightarrow$**y** (arête **x-y**), la représentation exige :

- un temps d'ordre **n**
- dans le pire des cas.

Le pire des cas :

- la liste d'adjacence est de longueur **n**-1,
- **y** est en fin de liste

2- Il en va de même pour **ajouter** un arc ou une arête (avec test de non répétition).

3- Elle ne permet pas de calculer facilement les opérations relatives aux **prédécesseurs**:

**d°**$^{-}$**(**s **)** **,** **ième_pred(**i,s,g**)**