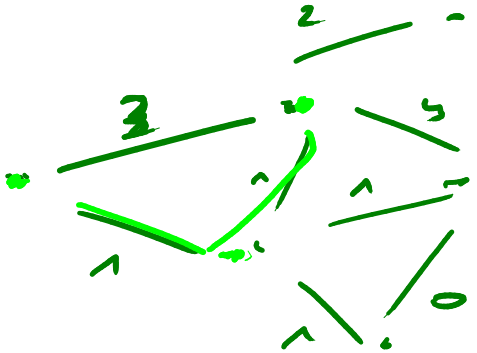
Application I

# Problème du plus court chemin

# Problème du plus court chemin (1)



Soit $G = (X, A, w)$, un **graphe valué** (ou pondéré) tel que :

- $G = (X, A)$ est un graphe orienté
- $w : A \to \mathbb{R}$ est une application qui associe à tout arc de $G$ un poids réel (coût, distance...)

## Définition

La **longueur d'un chemin** $\Gamma$ dans un graphe $G$ est égale à la somme des longueurs des arcs qui constituent ce chemin.

$$l(\Gamma) = \sum_{a \in \Gamma} w(a)$$

Cette définition généralise la notion de longueur d'un chemin que l'on avait utilisée jusqu'à présent et qui correspondait *au nombre d'arcs*. Ce qui était équivalent à un poids $w$ telle que $\forall a \in A$, $w(a) = 1$.

$w : A \longrightarrow \{1\}$

# Problème du plus court chemin (2)

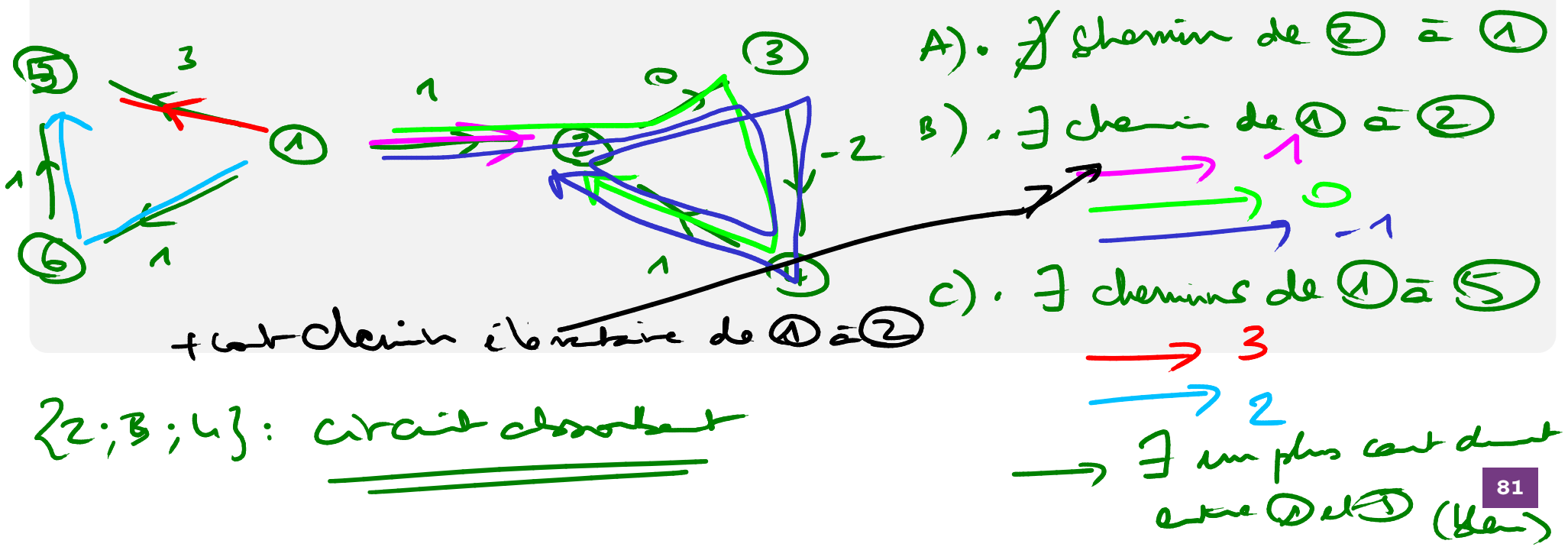
Etant donné deux sommets $x$ et $y$ d'un graphe $G$, 3 cas peuvent se présenter :

A) • Il n'existe pas de chemin entre $x$ et $y$.

B) • Il existe un chemin entre $x$ et $y$, mais pas de plus court chemin.

C) • Il existe un plus court chemin entre $x$ et $y$.

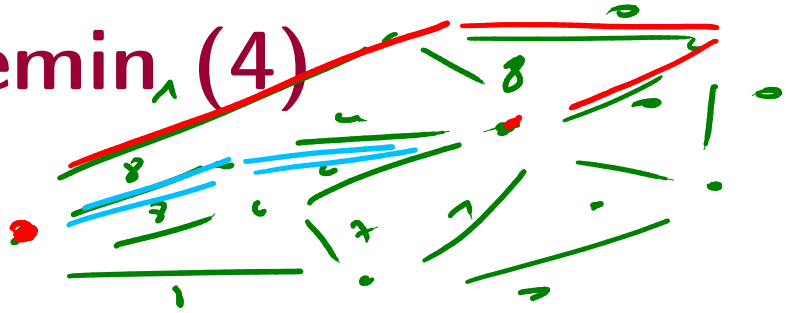
## Exemple



# Problème du plus court chemin (3)

- L'existence d'un chemin entre deux sommets $x$ et $y$ peut être vérifiée en appliquant un algorithme d'exploration du graphe à partir de $x$ (jusqu'à trouver $y$).
- L'existence d'un chemin ne garantit pas qu'il existe un plus court chemin : cas des circuits absorbants. Ce phénomène est dû à l'existence d'un circuit $C$ pour lequel sa longueur est négative ($l(C) < 0$). Ainsi, au plus on parcourt ce circuit, au plus la longueur totale du chemin diminue.
- La recherche du plus court chemin élémentaire permet de s'affranchir des circuits absorbants.

# Problème du plus court chemin (4)



Remarque : Combinatoire explosive...

Il existe dans un graphe fini un nombre fini de chemins élémentaires. La recherche de chemins élémentaires entre 2 sommets fixés sont des problèmes d'optimisation combinatoire non difficiles en principe car il "suffit" d'énumérer tous les chemins élémentaires possibles et de choisir celui de longueur minimale. La difficulté pratique provient de l'augmentation très rapide du nombre de chemins avec le nombre de sommets et d'arcs, ce qui rend l'énumération impossible.

Dans un graphe complet à $n$ sommets, il existe $n!$ chemins élémentaires.
Si $n = 5$, il suffit d'énumérer 120 chemins.
Si $n = 10$, un peu moins de 4 millions de chemins existent.
Si $n = 25$, un peu plus de $10^{25}$ chemins sont possibles. Avec un ordinateur qui évaluerait $10^9$ chemins par secondes, il faudrait l'âge de la Terre pour pouvoir tous les évaluer ! ! !

Nécessité de connaître des algorithmes qui donnent le plus court chemin (quand il existe) entre deux sommets sans énumérer toutes les possibilités.

# Problème du plus court chemin (5)

Le problème du plus court chemin peut être formuler de 3 façons différentes :

A/ Etant donné deux sommets $r$ et $t$, trouver un plus court chemin de $r$ à $t$.

B/ Etant donné un sommet de départ $r$, trouver un plus court chemin de $r$ vers tout autre sommet.

C/ Trouver un plus court chemin entre tout couple de sommets, ie. calculer une matrice appelée *distancier*.

Le problème B est celui qui nous intéresse car c'est le plus général. En effet, la résolution du problème B permet de traiter directement le problème A : l'algorithme est stoppé dès que le sommet de destination $t$ a été définitivement traité. De même, le problème C peut être traité en appliquant le problème B pour chaque sommet du graphe comme sommet de départ. Pour les problèmes A et C, il peut exister des algorithmes *très* spécifiques nécessitant certaines contraintes sur le graphe qui s'avèrent meilleurs en complexité que ceux proposés dans le cas du problème B. Ceux-ci ne seront pas étudiés dans ce cours.

# Problème du plus court chemin (6)

Problème : étant donné un sommet de départ $r$, trouver un plus court chemin de $r$ vers tout autre sommet du graphe $G = (X, A, w)$ ($n$ sommets et $m$ arcs).

- **$w$ constante** : revient à trouver les plus courts chemins en nombre d'arcs. Résolution avec une exploration de graphe en largeur, implémentable en $O(m)$.
- **$w \geq 0$** : algorithme de Dijkstra avec une complexité en $O(n^2)$.
- **$G$ sans circuit** : algorithme de Bellman avec une complexité en $O(m)$ (si les sommets sont triés).
- sinon : algorithme de Bellman-Ford (programmation dynamique)

Résultats :

- $\pi$ est un tableau contenant pour chaque sommet $x$, le coût du plus court chemin $\pi[x]$.
  Initialisation : $\pi[x] = +\infty$ pour tout $x \neq r$ et $\pi[r] = 0$.
- `Père` est un tableau stockant le père de chaque sommet.
  `Père`$[x] = y$ signifie que $y$ précède $x$ dans le chemin optimal reliant $r$ à $x$.
  `Père`$[x] = 0$ signifie que $x$ n'a pas été atteint par un chemin d'origine $r$.
  Initialisation : `Père`$[x] = 0$ pour tout $x \neq r$ et `Père`$[r]=r$.

# Algorithme de Dijkstra

Conditions d'application de l'algorithme de Dijkstra :

- Poids positifs ou nuls
- Graphe avec ou sans circuit(s)

## Principe

Calcule les plus courtes longueurs de proche en proche par ajustements successifs.

- Initialiser $\pi$, `Père` et `Mark`

Tant qu'il existe des sommets non marqués :

- Choisir parmi les sommets non marqués, le sommet $x$ avec $\pi[x]$ minimum.
- Marquer $x$
- Pour tous les successeurs $y$ de $x$,
  Si $\pi[y] > \pi[x] + w(xy)$ Alors $\pi[y] = \pi[x] + w(xy)$ et $\texttt{Père}[y] = x$.

↳ propagation des potentiels

# Algorithme de Dijkstra

## Pseudo-code

```
Initialiser Mark à Faux, π à +∞, Père à 0 et Fini := False
π[r] := 0, Père[r] := r

Répéter
    Fini := True
    Choisir x tel que Mark[x]=Faux et π[x] minimum
    Si x existe Alors
       Fini := False
       Mark[x] := Vrai
       Pour k := Head[x] à Head[x+1]-1
          y := Succ[k]
          Si π[y] > π[x]+w(xy) Alors
             π[y] := π[x]+w(xy)
             Père[y] := x
          FinSi
       FinPour
    FinSi
Jusqu'à (non Fini)
```

# Algorithme de Dijkstra

Remarques :

- L'algorithme de Dijkstra fournit l'arborescence des plus courts chemins enracinés en $r$ ainsi que toutes les longueurs $\pi(x)$ de $r$ à $x$.
- Preuve de l'optimalité : par récurrence
- Complexité :

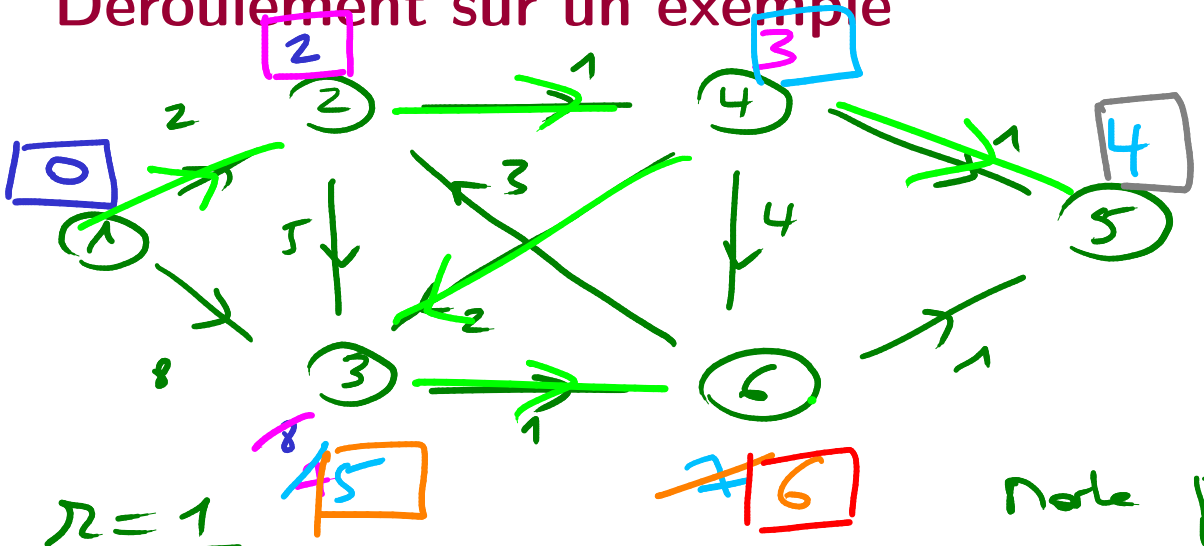
$$O(n + \sum_{i=1}^{n}(n + d^+(x))) = O(n^2)$$

Recherche du potentiel $\pi$ minimum : $O(n)$

Implémentation avec un tas : $O(\log n)$

# Algorithme de Dijkstra

## Déroulement sur un exemple




Head

| 1 | 3 | 5 | 6 | 9 | 9 |
|---|---|---|---|---|---|

Succ

| 2 \| 3 | 3 \| 4 | 6 | 3 \| 5 | 4 \| 2 | 5 |
|---|---|---|---|---|---|

w

| 2 \| 8 | 5 \| 1 | 1 \| 2 | 1 \| 4 | 3 \| 1 |
|---|---|---|---|---|---|

$R = 1$

Mark

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| V | V | F | V | F | V |

potentiel $\pi$

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 0 | 2 | ~~8~~ 7 | 3 | 4 | ~~7~~ 6 |

père

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 1 | ~~1~~ 2 | 2 | 4 | ~~4~~ 3 |

→ arborescence des plus courts chemins

enracinée en 1

ordre d'exploration : $x$ : 1 2 4 5 3 6

# Graphe sans circuit : ordre topologique

## Théorème

Un graphe sans circuit contient au moins une source, ie. un sommet $x$ tel que $d^-(x) = 0$.

## Conséquence

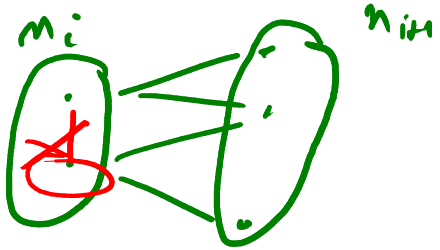
Un graphe orienté $G = (X, A)$ est sans circuit si et seulement si $X$ admet une partition en niveaux $X = N_0 \bigcup N_1 \bigcup \ldots \bigcup N_p$ telle que pour tout $x$ et pour tout $i$ :

$x \in N_i \leftrightarrow i$ est la longueur (taille) d'un plus **long** chemin se terminant en $x$.

Remarque : Les niveaux $N_i$ forment des stables.

Ainsi, les sommets sont triés par un tri topologique tel que :

$$\forall x \in X, \forall y \in X \text{ tels que } (xy) \in A \text{ alors } n(x) < n(y)$$



Ici, on numérote d'abord les sommets appartenant à $N_0$ puis $N_1$, ...

# Algorithme de Bellman

Conditions d'application de l'algorithme de Bellman :

- Graphe sans circuit
- Toute valeur réelle de poids
- Les sommets doivent être triés dans l'ordre topologique (tableau `Sorted`)

## Principe

Calcule les plus courtes longueurs en suivant la numérotation des sommets.

- Initialiser $\pi$, `Père`

Pour $k$ de 2 à $n$ :

- $y =$`Sorted`$[k]$
- Choisir le sommet $x$ parmi les prédécesseurs de $y$ tel que
$$\pi[y] = \min(\pi[x] + w(xy))$$

# Algorithme de Bellman

## Pseudo-code

```
Initialiser π à +∞, Père à 0
π[r] := 0, Père[r] := r

Pour i := 2 à n
    y := Sorted[i]
    Pour tout prédécesseur x de y
       Si (π[x] ≠ ∞) et (π[x]+w(xy)<π[y]) Alors
          π[y] := π[x]+w(xy)
          Père[y] := x
       FinSi
    FinPour
FinPour
```

# Algorithme de Bellman

Remarques :

- Complexité : $O(m)$
- Nécessite de trier les sommets par niveaux (en $O(n+m)$ avec un algorithme de parcours en profondeur)
- Variante : utiliser les niveaux pour éviter les prédécesseurs.

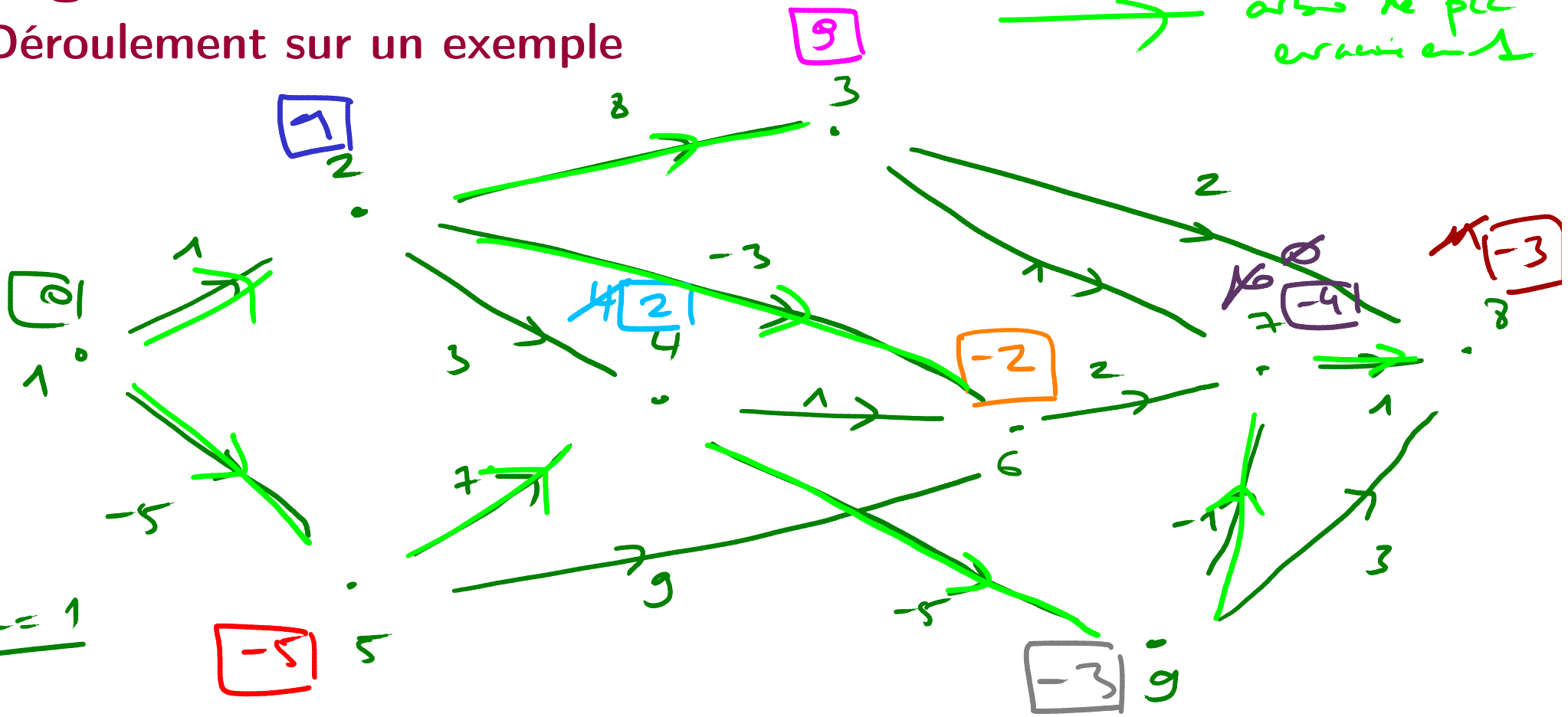
```
Initialiser π à +∞, Père à 0
π[r] := 0, Père[r] := r
Pour tout x successeur de r (ie. x ∈ N[1])
    π[x] := w(rx)
    Père[x] := r
FinPour
Pour i := 1 à L /* L est le nombre total de niveau */
    Pour tout x dans N[i]
       Pour tout successeur y de x tel que π[x]+w(xy)<π[y]
          π[y] := π[x]+w(xy)
          Père[y] := x
       FinPour
    FinPour
FinPour
```

# Algorithme de Bellman

## Déroulement sur un exemple

$N_2 = ?$ (3) (4)

→ arbre de pcc




Etape 1 Tri topologique des sommets

$N_0 = \{1\}$
$N_1 = \{2;5\}$
$N_2 = \{3;4\}$
$N_3 = \{6,9\}$
$N_4 = \{7\}$
$N_5 = \{8\}$

Etape 2 :

Sorted (=x) { $N_0$ 1, $N_1$ 2, $N_1$ 5, $N_2$ 3, $N_2$ 4, $N_3$ 6, $N_3$ 9, $N_4$ 7, $N_5$ 8 }

Père

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 1 | 2 | 9 | 7 | 4 |

$\pi$

| 0 | 1 | 9 | 2 | -5 | -2 | -4 | -3 | -3 |
|---|---|---|---|---|---|---|---|---|