

Cours n° 4 : programmation dynamique

Christophe Gonzales



HUGO3 — Algorithmique

Programmation dynamique

- ▶ Idée : **diviser** un problème en sous-problèmes *non* indépendants
- ▶ **Stocker** les solutions de ces sous-problèmes
⇒ on ne calcule pas 2 fois la même chose
- ▶ Se servir des solutions stockées pour résoudre le problème d'origine

Programmation dynamique

- ▶ Idée : **diviser** un problème en sous-problèmes *non* indépendants
- ▶ **Stocker** les solutions de ces sous-problèmes
⇒ on ne calcule pas 2 fois la même chose
- ▶ Se servir des solutions stockées pour résoudre le problème d'origine

Illustrations du principe :

- ▶ Séquence commune la plus longue à 2 chaînes de caractères
- ▶ Distance de Levenshtein (entre 2 chaînes)
- ▶ Multiplications d'une séquence de matrices
- ▶ Stockage : sac à dos 0-1, rendu de monnaie
- ▶ Plus court chemin
- ▶ Apprentissage de paramètres (Baum-Welch)

- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press



- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press

- ▶ **Préface du livre** :

- ▶ « Théorie mathématique de processus de décision séquentiels »



- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press

- ▶ **Préface du livre** :

- ▶ « Théorie mathématique de processus de décision séquentiels »
- ▶ Systèmes qui évoluent dans le temps



- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press



- ▶ **Préface du livre** :

- ▶ « Théorie mathématique de processus de décision séquentiels »
- ▶ Systèmes qui évoluent dans le temps
- ▶ Décision : choix de la transformation du système
 - ⇒ on recherche la « meilleure » transformation
 - ⇒ **optimisation**

- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press



- ▶ **Préface du livre** :

- ▶ « Théorie mathématique de processus de décision séquentiels »
- ▶ Systèmes qui évoluent dans le temps
- ▶ Décision : choix de la transformation du système
 - ⇒ on recherche la « meilleure » transformation
 - ⇒ **optimisation**
- ▶ Séquentiels : plusieurs décisions à prendre **dans le temps**

- ▶ **Historique** : paradigme développé par Richard Bellman dans les années 50 :



Dynamic Programming (1957)
Princeton University Press

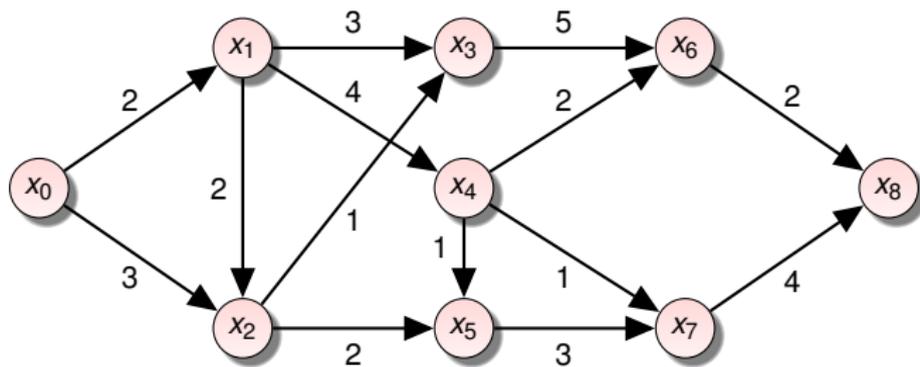


- ▶ **Préface du livre** :

- ▶ « Théorie mathématique de processus de décision séquentiels »
- ▶ Systèmes qui évoluent dans le temps
- ▶ Décision : choix de la transformation du système
 - ⇒ on recherche la « meilleure » transformation
 - ⇒ **optimisation**
- ▶ Séquentiels : plusieurs décisions à prendre **dans le temps**
- ▶ Temps ⇒ « dynamique »
- ▶ « Le temps joue un rôle important et l'ordre des opérations est cruciale »
- ▶ Programmation dynamique ⇒ utiliser cet ordre pour construire *progressivement* la solution

Retour sur le GPS : plus court chemin

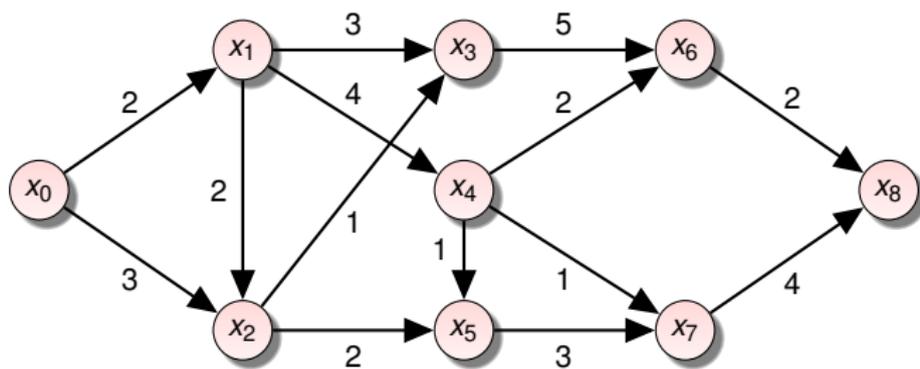
Problème : aller le plus rapidement de x_0 à x_8 :



► Nœud x_i : ville

Retour sur le GPS : plus court chemin

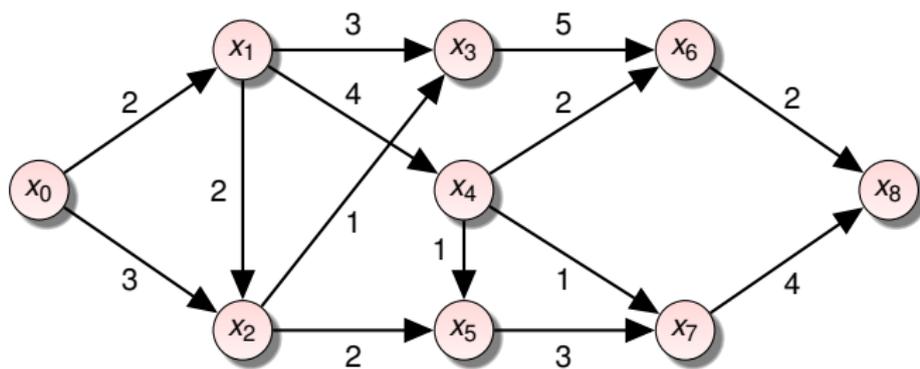
Problème : aller le plus rapidement de x_0 à x_8 :



- ▶ Nœud x_i : ville
- ▶ Arc $x_i \rightarrow x_j$: « chemin » permettant de se rendre de x_i à x_j

Retour sur le GPS : plus court chemin

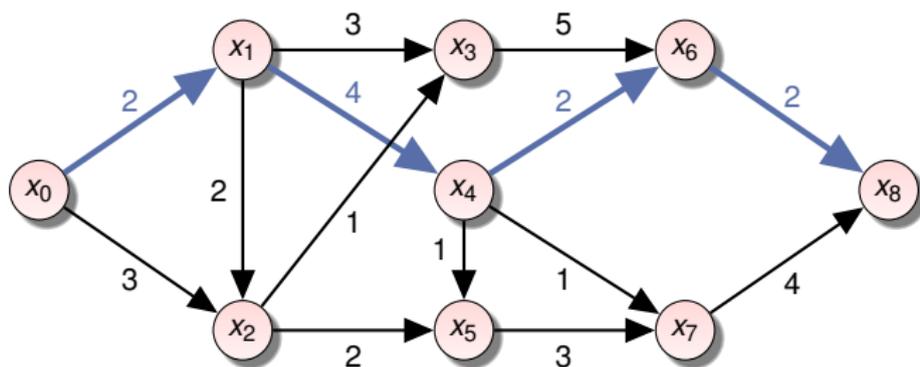
Problème : aller le plus rapidement de x_0 à x_8 :



- ▶ Nœud x_i : ville
- ▶ Arc $x_i \rightarrow x_j$: « chemin » permettant de se rendre de x_i à x_j
- ▶ Nombre à côté d'un arc $x_i \rightarrow x_j$: temps pour aller de x_i à x_j

Retour sur le GPS : plus court chemin

Problème : aller le plus rapidement de x_0 à x_8 :



- ▶ Nœud x_i : ville
- ▶ Arc $x_i \rightarrow x_j$: « chemin » permettant de se rendre de x_i à x_j
- ▶ Nombre à côté d'un arc $x_i \rightarrow x_j$: temps pour aller de x_i à x_j

Chemin le plus rapide \implies minimum somme des temps

- ▶ **Exemple** : temps $(x_0 \rightarrow x_1 \rightarrow x_4 \rightarrow x_6 \rightarrow x_8) = 2 + 4 + 2 + 2 = 10$

Graphe, orienté, non orienté, DAG...

- ▶ Graphe : couple $G = (V, E)$
- ▶ V : ensemble de nœuds (ou *sommets*) = $\{x_0, \dots, x_8\}$
- ▶ $E \subseteq V \times V$: extrémités des arcs ou arêtes

Grphe, orienté, non orienté, DAG...

- ▶ Graphe : couple $G = (V, E)$
- ▶ V : ensemble de nœuds (ou *sommets*) = $\{x_0, \dots, x_8\}$
- ▶ $E \subseteq V \times V$: extrémités des arcs ou arêtes

- ▶ Graphe orienté : $G_1 =$ 
 $V = \{A, B, C\}, E = \{(A, B), (B, C)\} = \{\text{arcs}\}$

Graphe, orienté, non orienté, DAG...

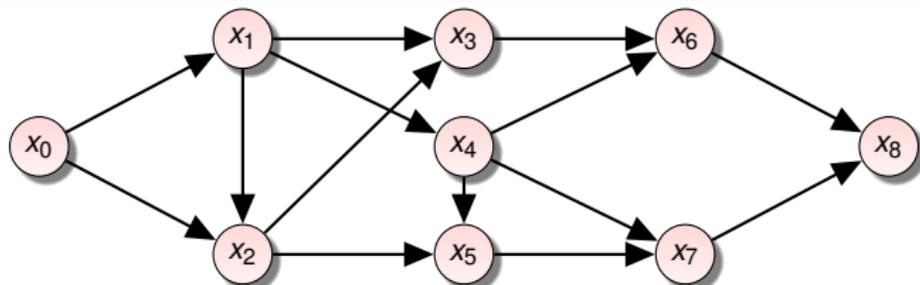
- ▶ Graphe : couple $G = (V, E)$
- ▶ V : ensemble de nœuds (ou *sommets*) = $\{x_0, \dots, x_8\}$
- ▶ $E \subseteq V \times V$: extrémités des arcs ou arêtes
- ▶ Graphe orienté : $G_1 =$ 
 $V = \{A, B, C\}, E = \{(A, B), (B, C)\} = \{\text{arcs}\}$
- ▶ Graphe non orienté : $G_2 =$ 
 $V = \{A, B, C\}, E = \{(A, B), (B, A), (B, C), (C, B)\} = \{\text{arêtes}\}$

Graphe, orienté, non orienté, DAG...

- ▶ Graphe : couple $G = (V, E)$
- ▶ V : ensemble de nœuds (ou *sommets*) = $\{x_0, \dots, x_8\}$
- ▶ $E \subseteq V \times V$: extrémités des arcs ou arêtes
- ▶ Graphe orienté : $G_1 =$ 
 $V = \{A, B, C\}$, $E = \{(A, B), (B, C)\} = \{\text{arcs}\}$
- ▶ Graphe non orienté : $G_2 =$ 
 $V = \{A, B, C\}$, $E = \{(A, B), (B, A), (B, C), (C, B)\} = \{\text{arêtes}\}$
- ▶ Dans un graphe orienté, pour tout nœud $v \in V$:
 - ▶ v' : $\text{parent}(v) \iff (v', v) \in E$: dans G_1 , $B = \text{parent}(C)$
 - ▶ v' : $\text{enfant}(v) \iff (v, v') \in E$: dans G_1 , $C = \text{enfant}(B)$

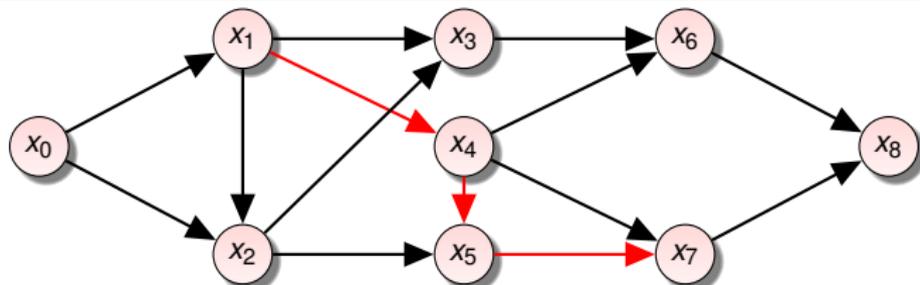
Graphe, orienté, non orienté, DAG...

- ▶ Graphe : couple $G = (V, E)$
- ▶ V : ensemble de nœuds (ou *sommets*) = $\{x_0, \dots, x_8\}$
- ▶ $E \subseteq V \times V$: extrémités des arcs ou arêtes
- ▶ Graphe orienté : $G_1 =$ 
 $V = \{A, B, C\}, E = \{(A, B), (B, C)\} = \{\text{arcs}\}$
- ▶ Graphe non orienté : $G_2 =$ 
 $V = \{A, B, C\}, E = \{(A, B), (B, A), (B, C), (C, B)\} = \{\text{arêtes}\}$
- ▶ Dans un graphe orienté, pour tout nœud $v \in V$:
 - ▶ v' : $\text{parent}(v) \iff (v', v) \in E$: dans $G_1, B = \text{parent}(C)$
 - ▶ v' : $\text{enfant}(v) \iff (v, v') \in E$: dans $G_1, C = \text{enfant}(B)$
- ▶ Dans un graphe non orienté, pour tout nœud $v \in V$:
 - ▶ v' : $\text{voisin}(v) \iff (v', v) \in E$: dans $G_2, B = \text{voisin}(C)$



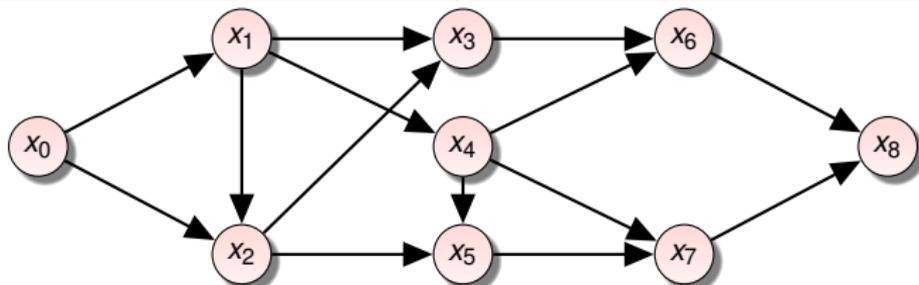
Propriétés des graphes orientés

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs.



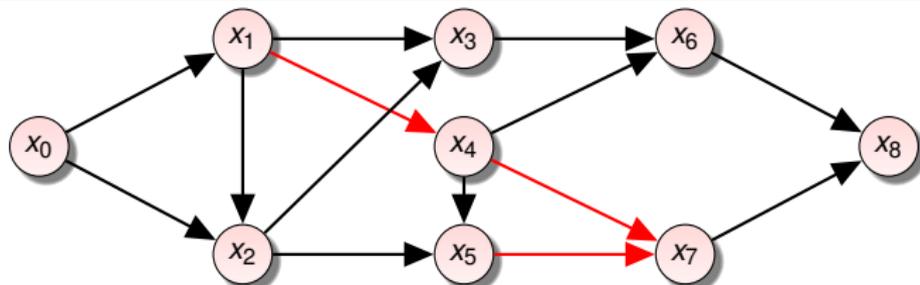
Propriétés des graphes orientés

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs. **Exemple** : $\{x_1, x_4, x_5, x_7\}$



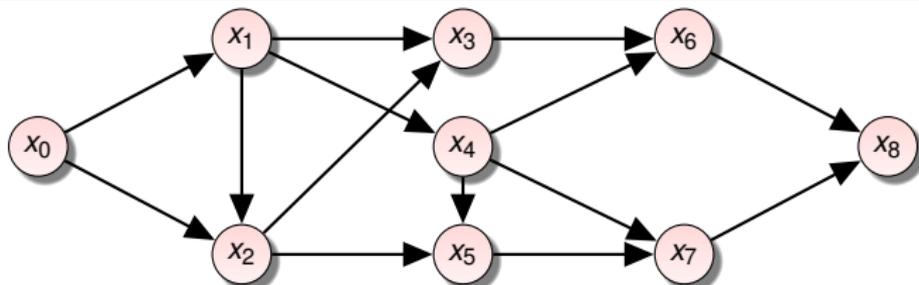
Propriétés des graphes orientés

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs. **Exemple** : $\{x_1, x_4, x_5, x_7\}$
- ▶ **Chaîne** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$ **ou** $(v_{i+1}, v_i) \in E$
On peut ne pas suivre le sens des arcs.



Propriétés des graphes orientés

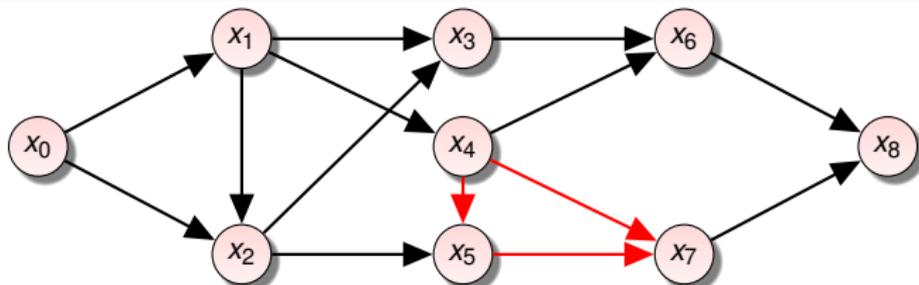
- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs. **Exemple** : $\{x_1, x_4, x_5, x_7\}$
- ▶ **Chaîne** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$ **ou** $(v_{i+1}, v_i) \in E$
On peut ne pas suivre le sens des arcs. **Exemple** : $\{x_1, x_4, x_7, x_5\}$



Propriétés des graphes orientés

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs. **Exemple** : $\{x_1, x_4, x_5, x_7\}$
- ▶ **Chaîne** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$ **ou** $(v_{i+1}, v_i) \in E$
On peut ne pas suivre le sens des arcs. **Exemple** : $\{x_1, x_4, x_7, x_5\}$
- ▶ **Circuit** : chemin $\{v_1, v_2, \dots, v_k, v_1\}$
 \implies chemin où on revient au point de départ

Terminologie : théorie des graphes (suite)



Propriétés des graphes orientés

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ **Chemin** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$
On suit le sens des arcs. **Exemple** : $\{x_1, x_4, x_5, x_7\}$
- ▶ **Chaîne** : suite de nœuds $\{v_1, v_2, \dots, v_k\} \subseteq V$ tel que :
pour tout $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$ **ou** $(v_{i+1}, v_i) \in E$
On peut ne pas suivre le sens des arcs. **Exemple** : $\{x_1, x_4, x_7, x_5\}$
- ▶ **Circuit** : chemin $\{v_1, v_2, \dots, v_k, v_1\}$
 \implies chemin où on revient au point de départ
- ▶ **Cycle** : chaîne $\{v_1, v_2, \dots, v_k, v_1\}$. **Exemple** : $\{x_4, x_5, x_7, x_4\}$

Directed Acyclic Graph (DAG)

DAG = graphe orienté sans circuit.

Directed Acyclic Graph (DAG)

DAG = graphe orienté sans circuit.

Ordre topologique

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ Ordre \prec sur les nœuds tel que :
pour tout arc $(x_i, x_j) \in E$, on a $x_i \prec x_j$

Directed Acyclic Graph (DAG)

DAG = graphe orienté sans circuit.

Ordre topologique

- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ Ordre \prec sur les nœuds tel que :
pour tout arc $(x_i, x_j) \in E$, on a $x_i \prec x_j$

Théorème : dans tout DAG, il existe un ordre topologique.

Démonstration : par récurrence.

Directed Acyclic Graph (DAG)

DAG = graphe orienté sans circuit.

Ordre topologique

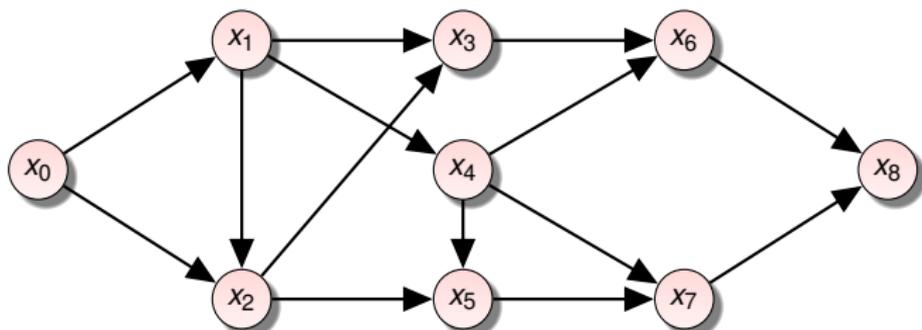
- ▶ Soit $G = (V, E)$ un graphe orienté
- ▶ Ordre \prec sur les nœuds tel que :
pour tout arc $(x_i, x_j) \in E$, on a $x_i \prec x_j$

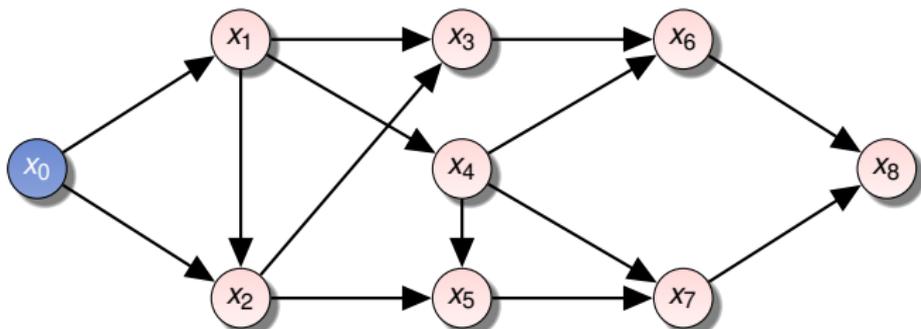
Théorème : dans tout DAG, il existe un ordre topologique.

Démonstration : par récurrence.

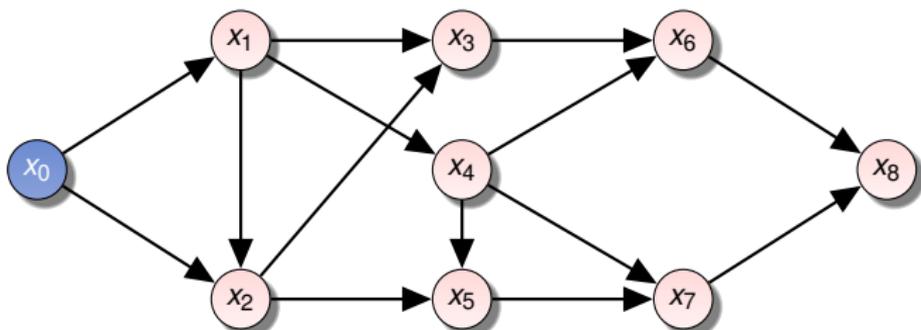


Plus court chemin : examiner les nœuds dans cet ordre !



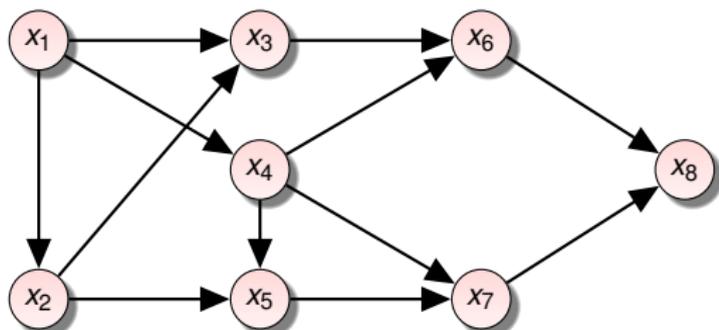


- DAG \implies il existe au moins un nœud sans parent !



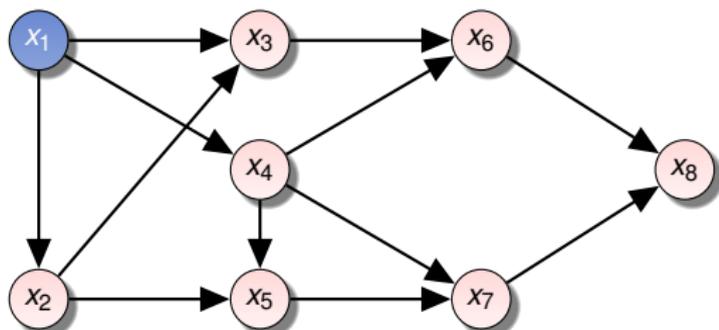
x_0

- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique



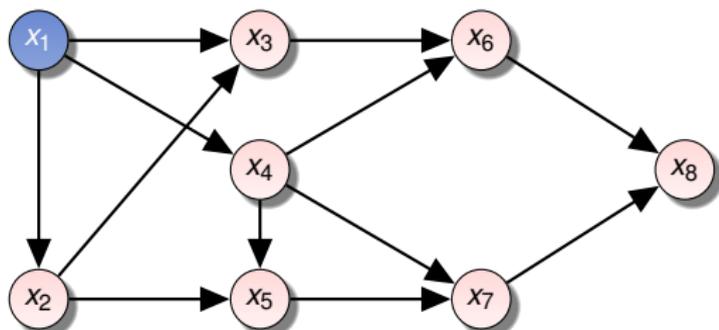
x_0

- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer

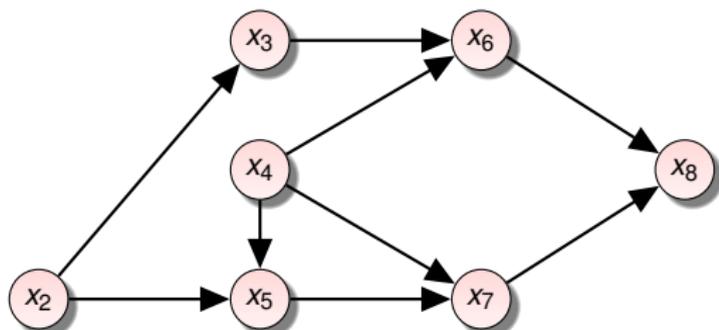


x_0

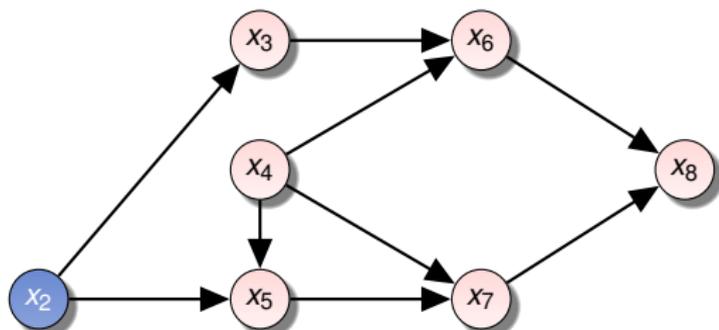
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



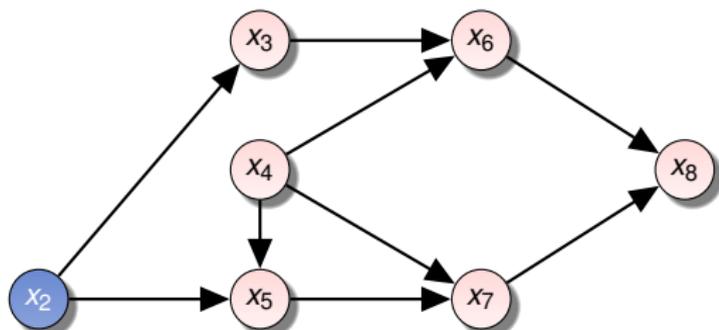
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



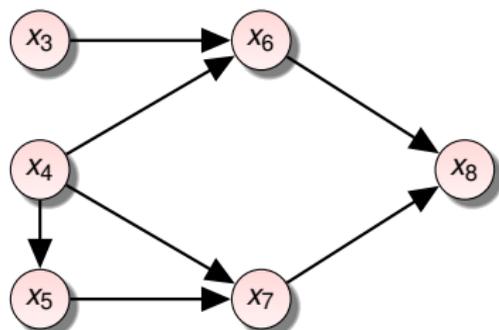
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



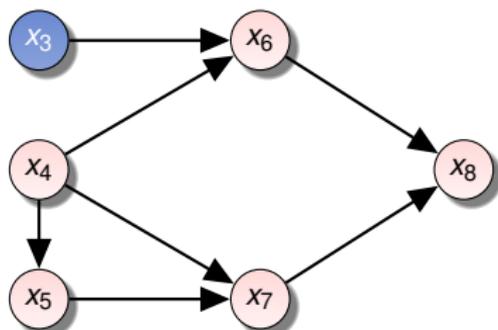
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



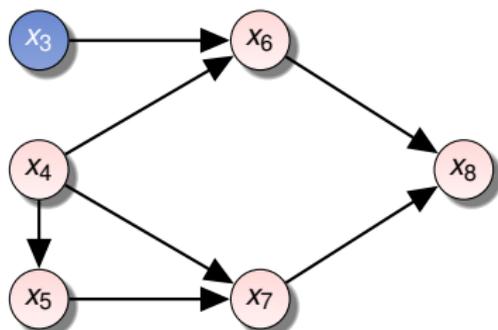
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



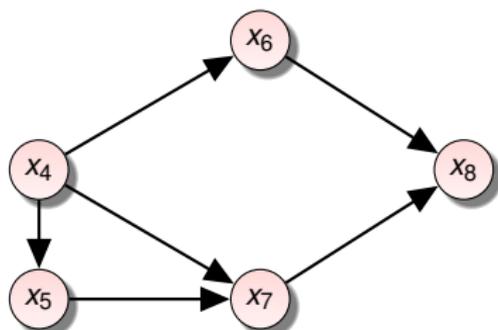
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



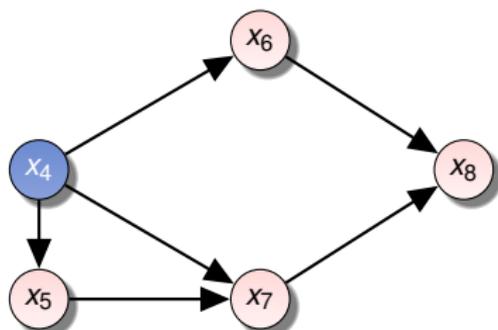
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



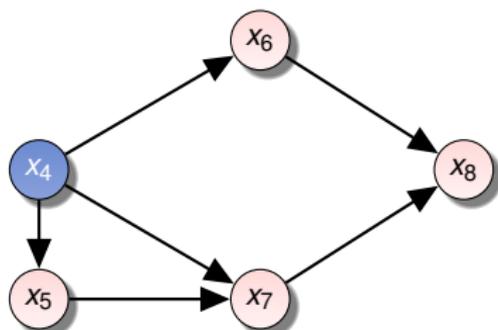
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



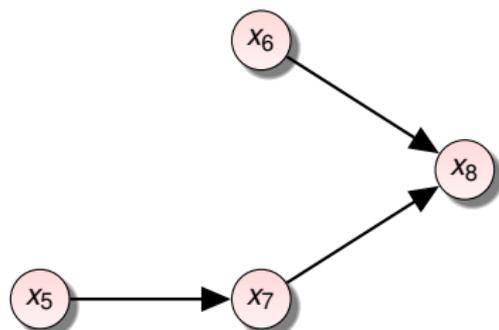
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



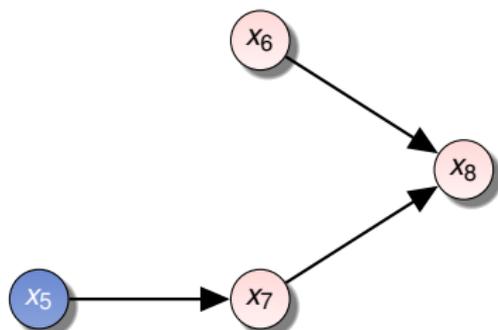
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



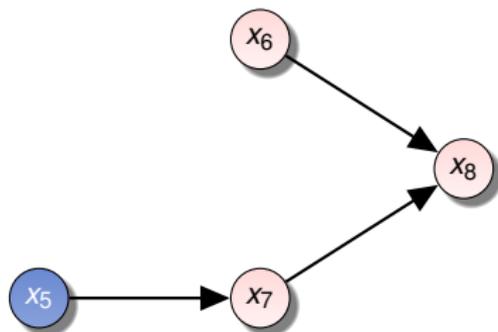
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



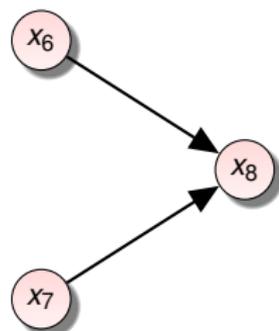
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



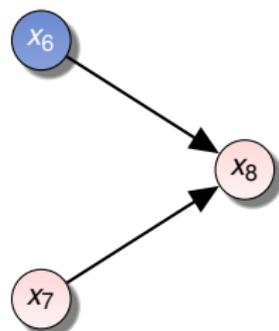
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



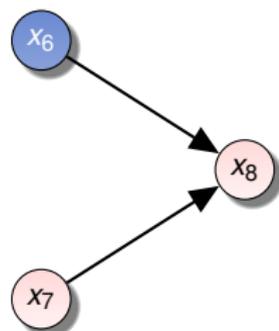
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



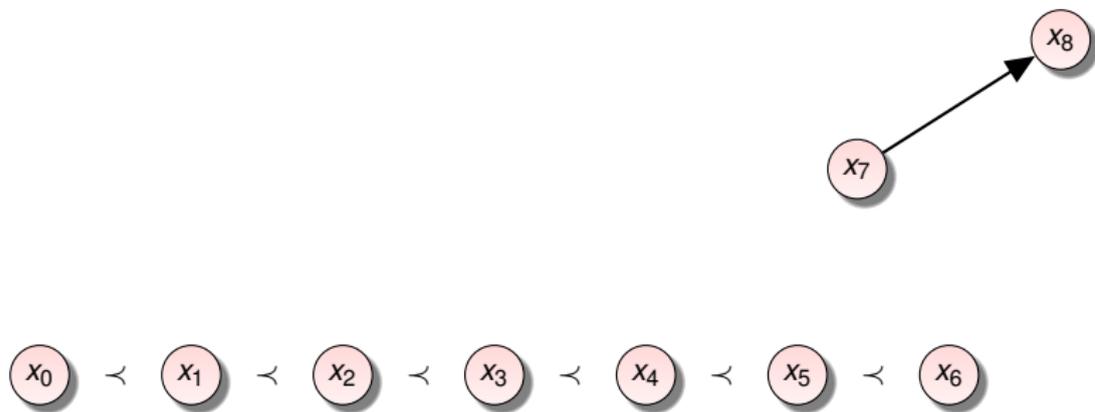
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



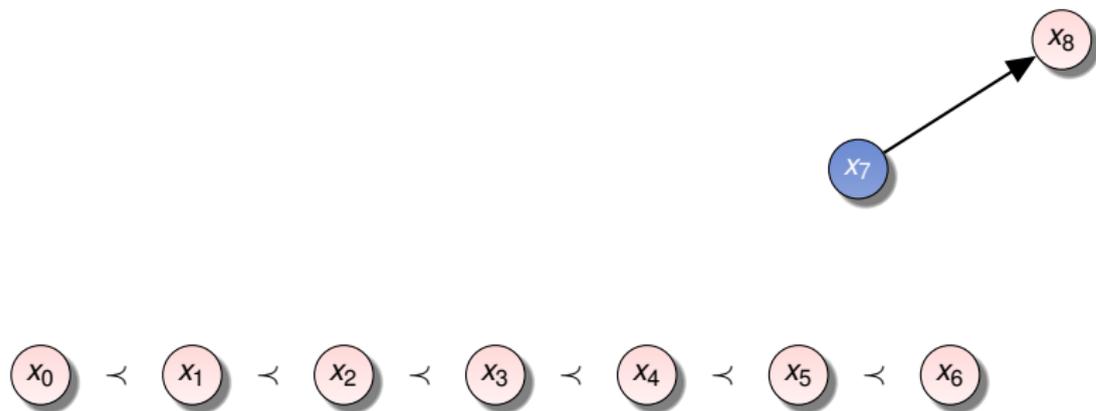
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



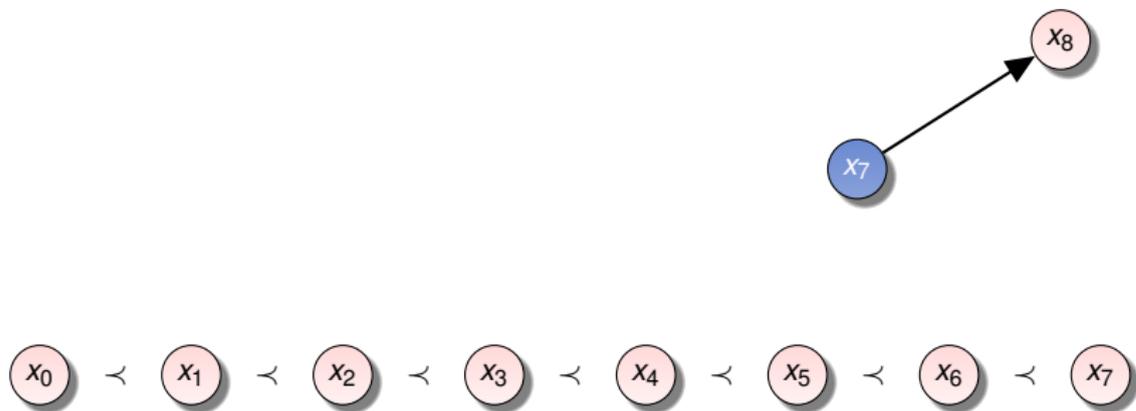
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



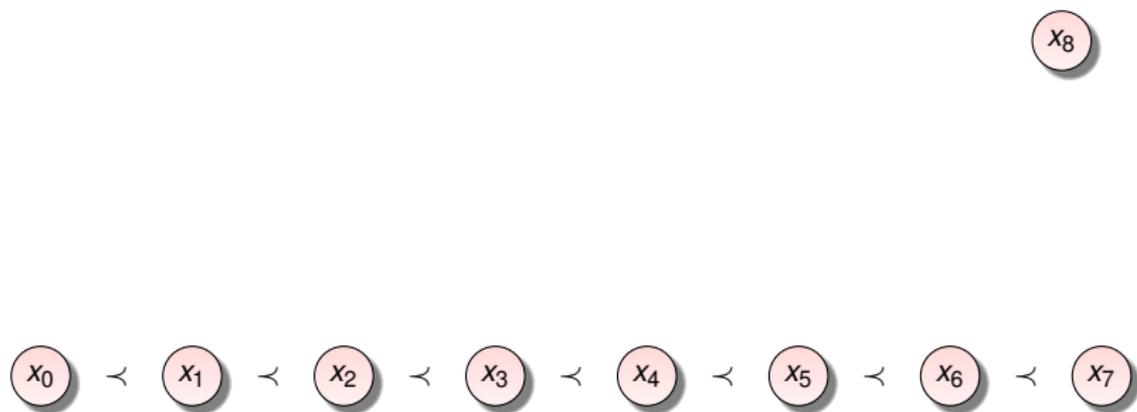
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



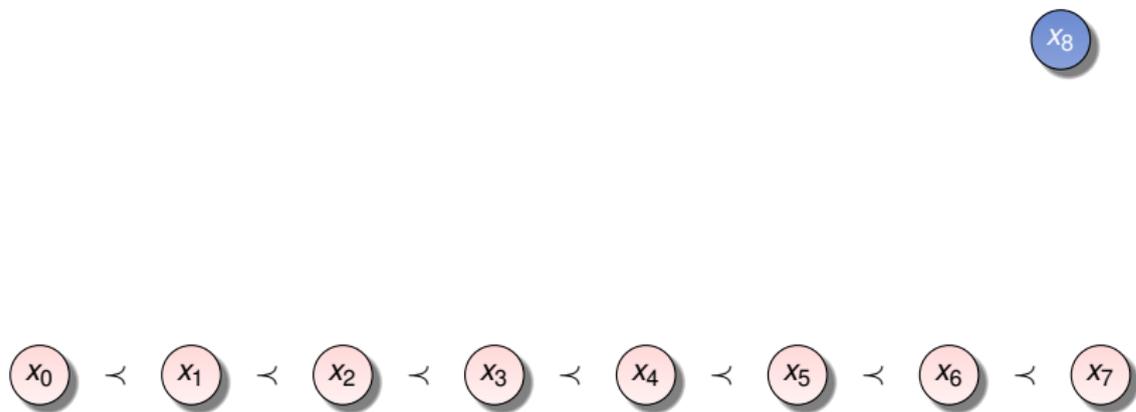
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



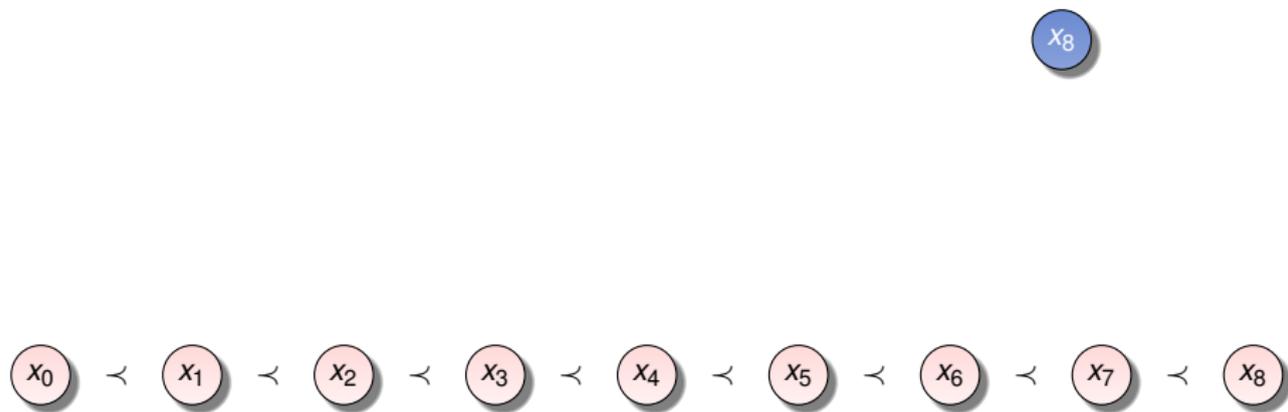
- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer



- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer

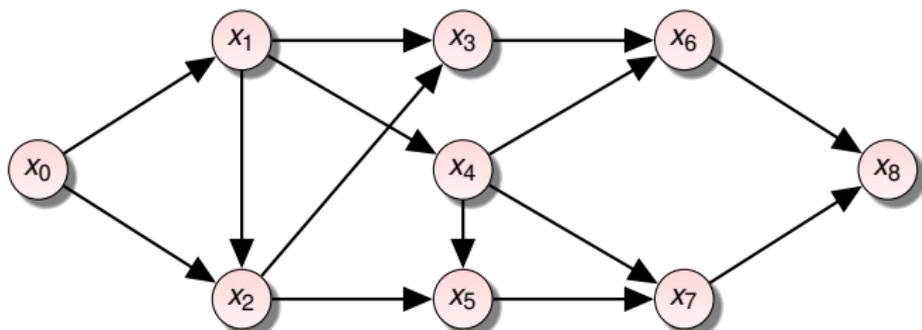


- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer

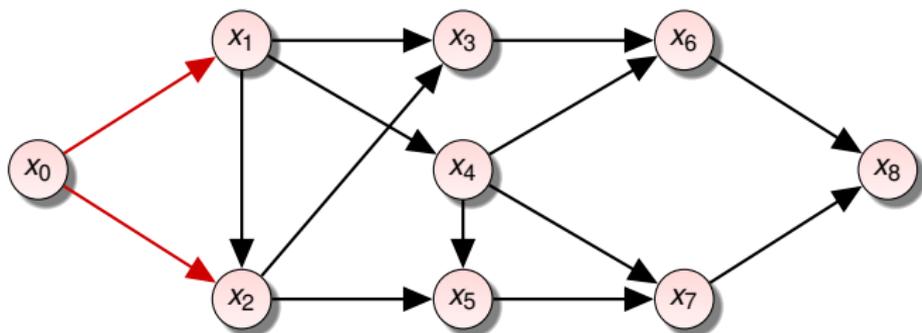


- ▶ DAG \implies il existe au moins un nœud sans parent !
- ▶ Rajouter ce nœud sans parent dans l'ordre topologique
- ▶ Supprimer ce nœud du graphe et réitérer

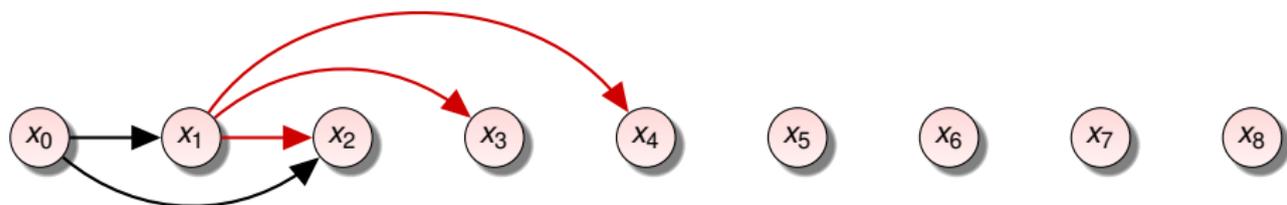
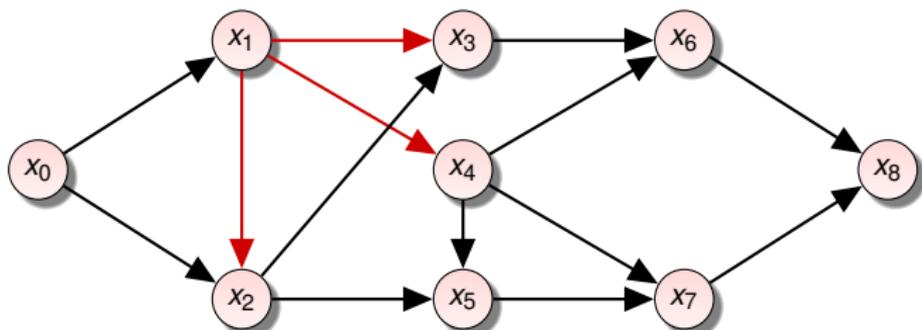
Ordre topologique (suite et fin)



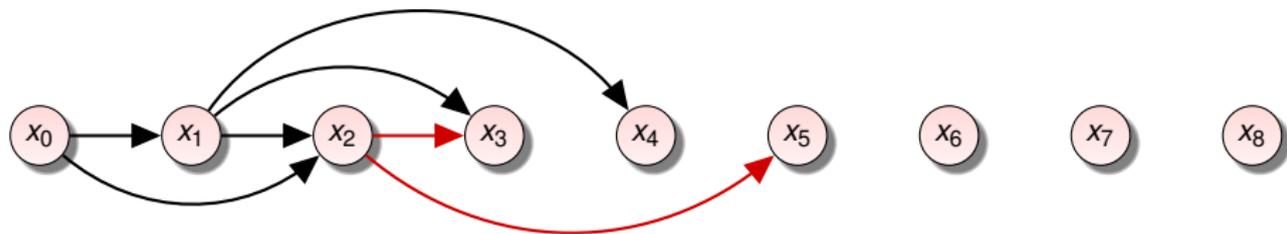
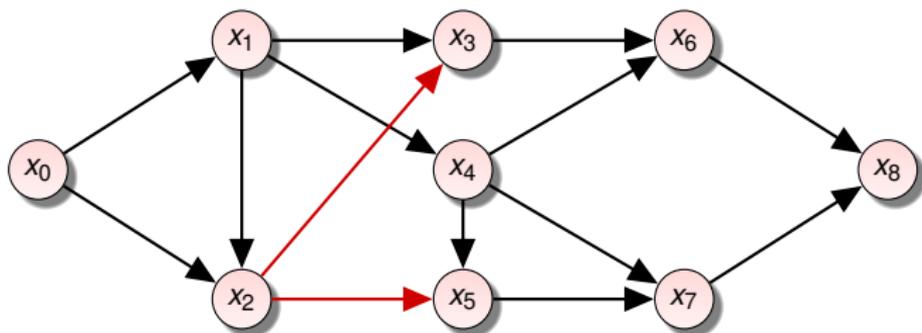
Ordre topologique (suite et fin)



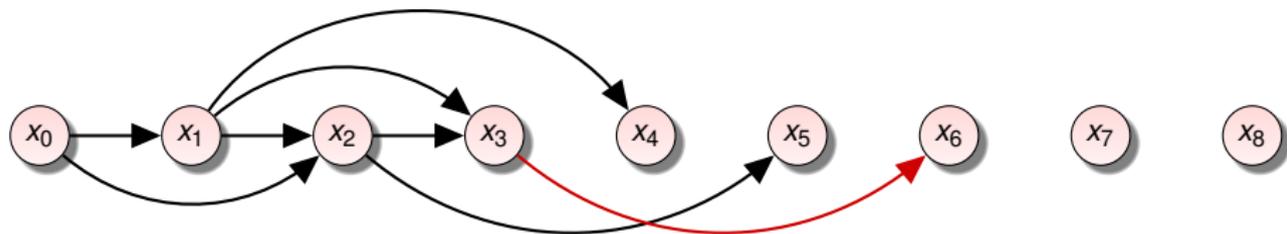
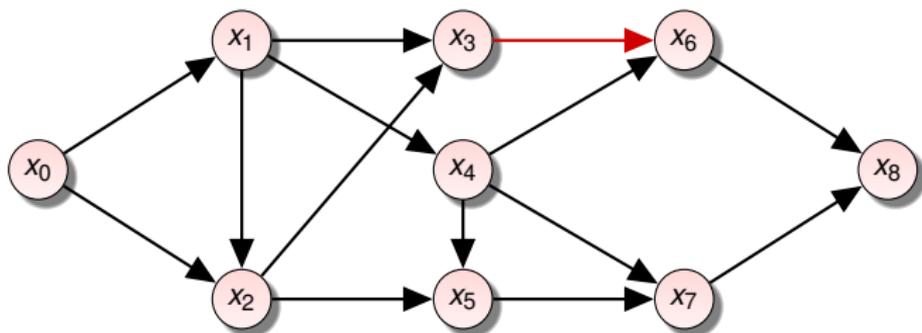
Ordre topologique (suite et fin)



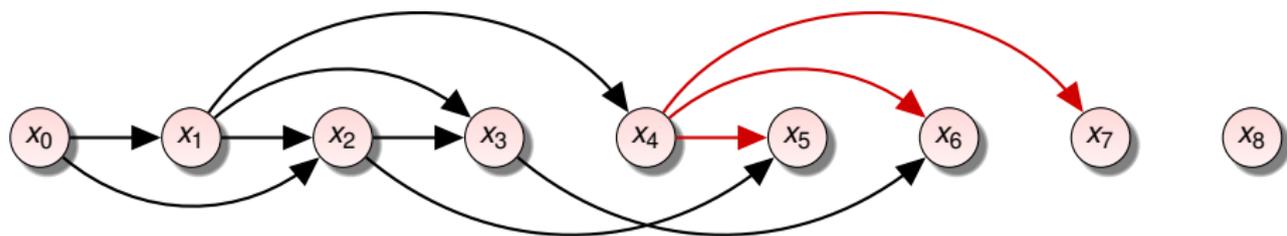
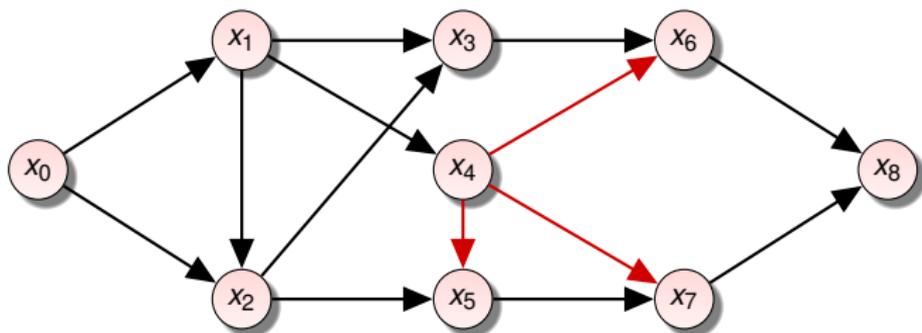
Ordre topologique (suite et fin)



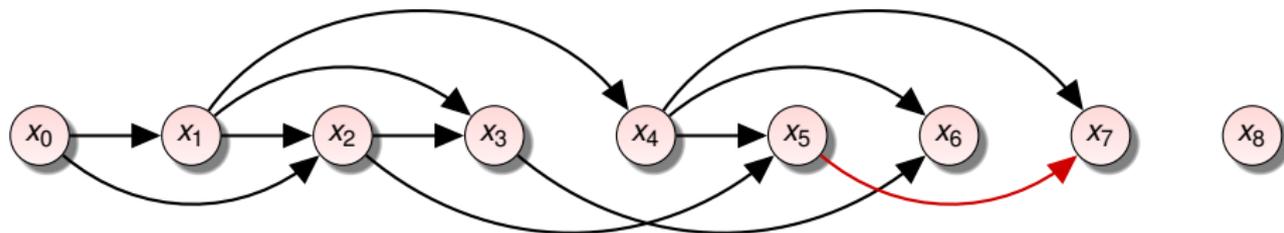
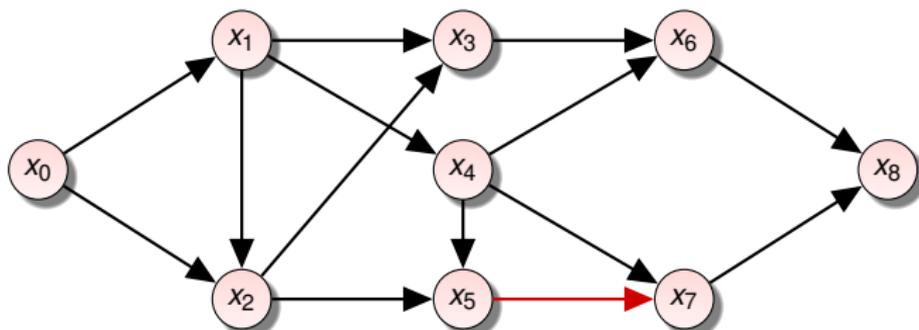
Ordre topologique (suite et fin)



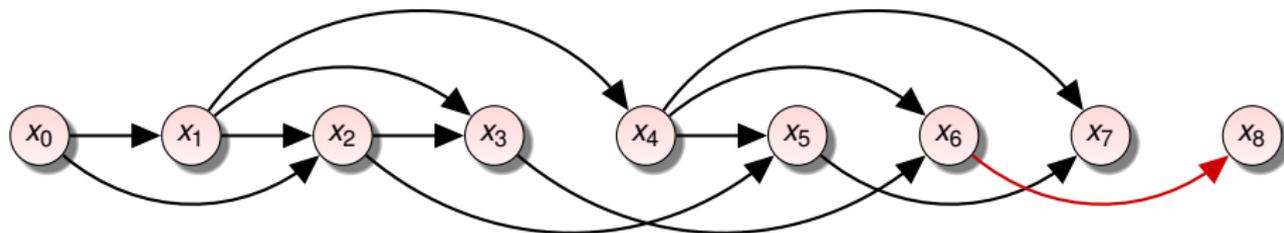
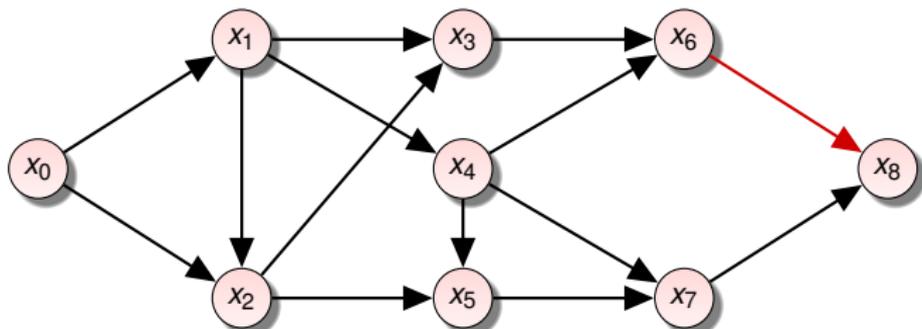
Ordre topologique (suite et fin)



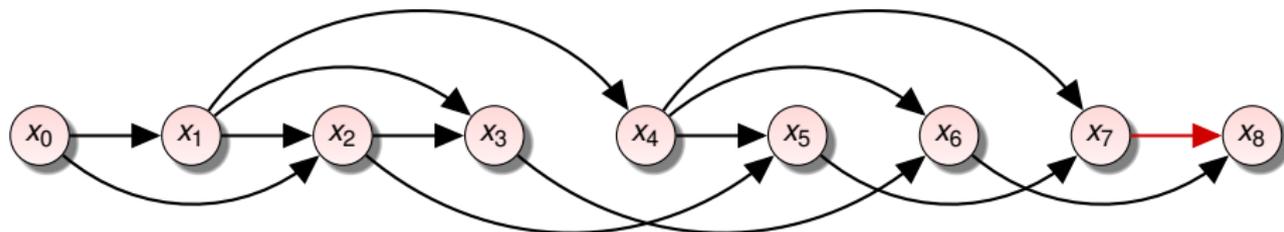
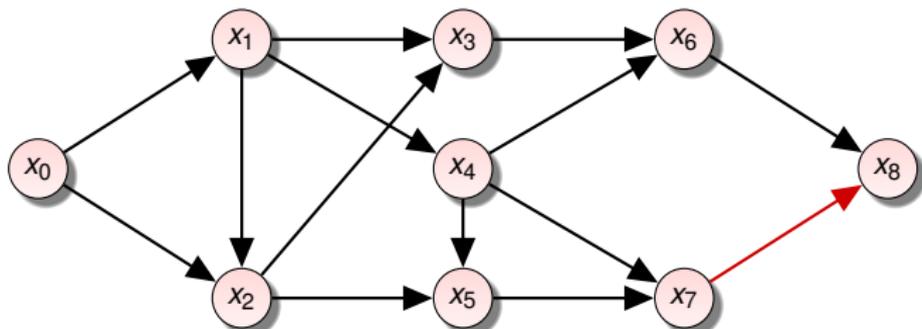
Ordre topologique (suite et fin)



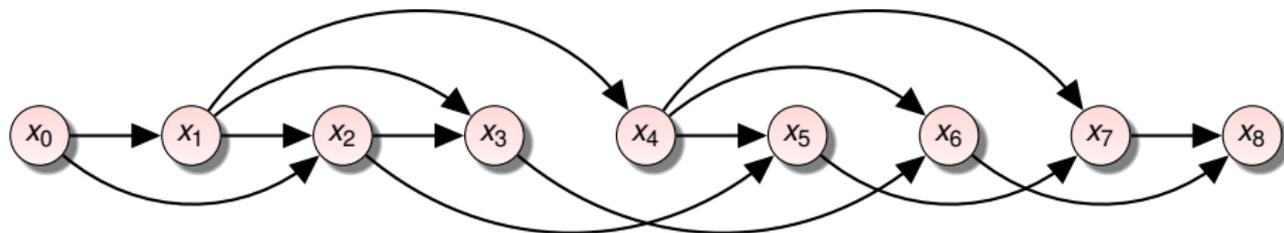
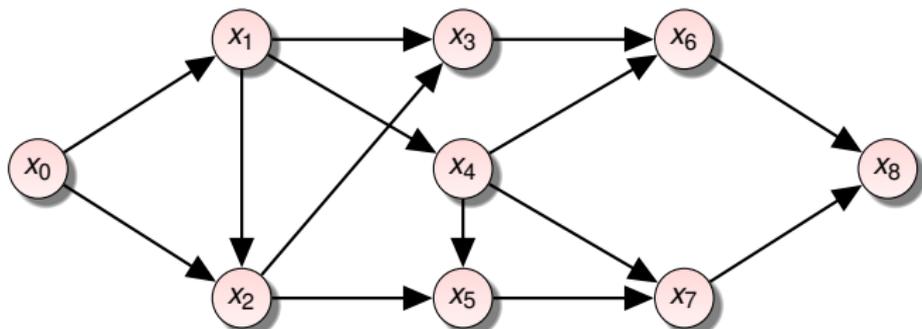
Ordre topologique (suite et fin)



Ordre topologique (suite et fin)



Ordre topologique (suite et fin)



Par construction : arcs uniquement de la gauche vers la droite !

- ▶ Théorie des graphes \implies Claude Berge



Théorie des graphes et ses applications (1958)
Dunod, Paris



- ▶ Introduction des « graphes parfaits »

- ▶ Théorie des graphes \implies Claude Berge



Théorie des graphes et ses applications (1958)
Dunod, Paris

- ▶ Introduction des « graphes parfaits »
- ▶ Prix Euler en 1993



- ▶ Théorie des graphes \implies Claude Berge



Théorie des graphes et ses applications (1958)
Dunod, Paris

- ▶ Introduction des « graphes parfaits »
- ▶ Prix Euler en 1993
- ▶ Membre fondateur de l'Oulipo en 1960
(Georges Perec, Raymond Queneau, Italo Calvino, *etc.*)



- ▶ Théorie des graphes \implies Claude Berge



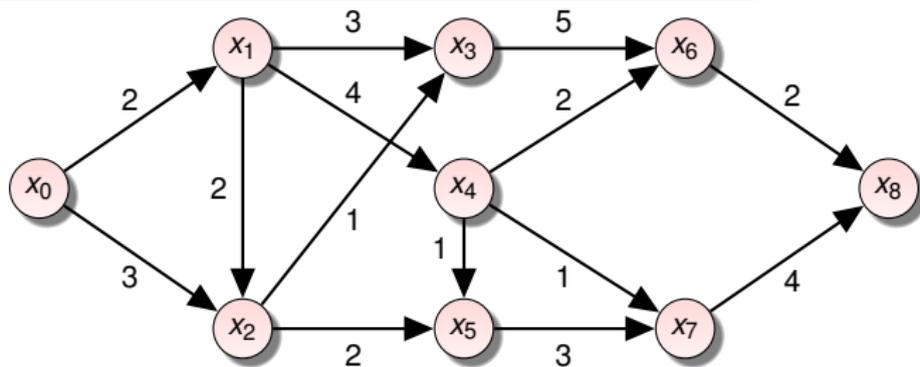
Théorie des graphes et ses applications (1958)
Dunod, Paris



- ▶ Introduction des « graphes parfaits »
- ▶ Prix Euler en 1993
- ▶ Membre fondateur de l'Oulipo en 1960
(Georges Perec, Raymond Queneau, Italo Calvino, *etc.*)
- ▶ Romans policiers et poèmes
Qui a tué le Duc de Densmore ?
La reine aztèque ou contraintes pour un sonnet à longueur variable

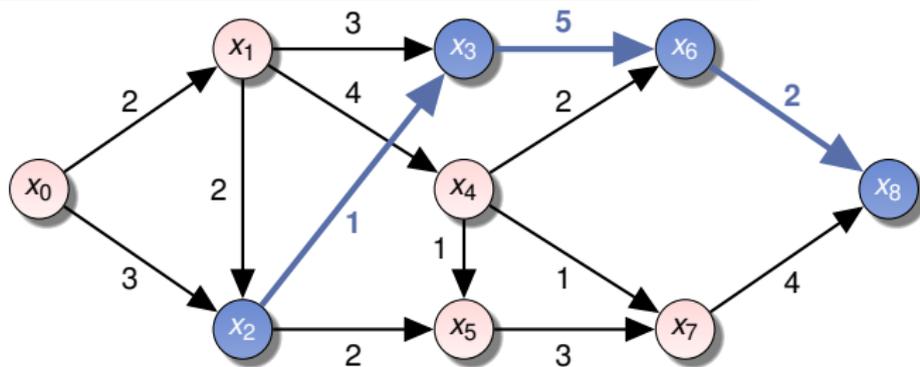
GPS : idée pour calculer le plus court chemin

Problème : aller le plus rapidement de x_0 à x_8 :



GPS : idée pour calculer le plus court chemin

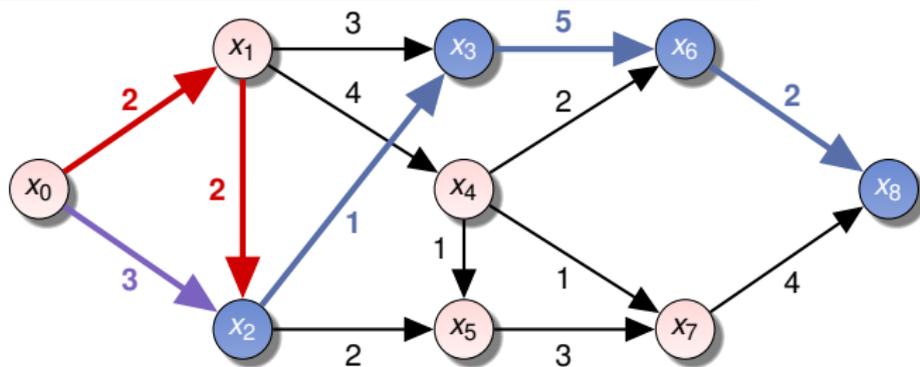
Problème : aller le plus rapidement de x_0 à x_8 :



► **Idée** : Si chemin passe par x_2 suivant le chemin bleu

GPS : idée pour calculer le plus court chemin

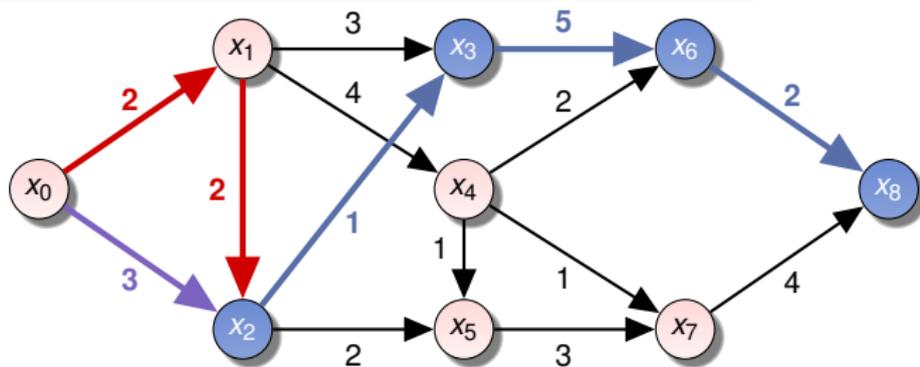
Problème : aller le plus rapidement de x_0 à x_8 :



- **Idée :** Si chemin passe par x_2 suivant le chemin bleu
Chemin optimal = Chemin rouge puis bleu ou
violet puis bleu ?

GPS : idée pour calculer le plus court chemin

Problème : aller le plus rapidement de x_0 à x_8 :

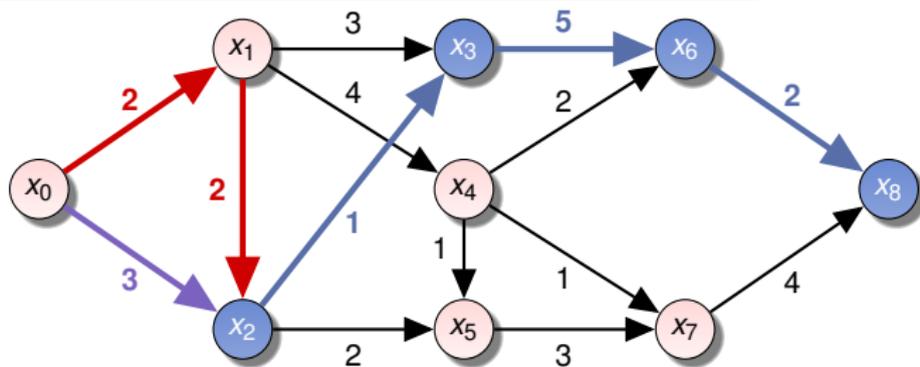


- **Idée :** Si chemin passe par x_2 suivant le chemin bleu
Chemin optimal = Chemin rouge puis bleu ou
violet puis bleu ?

Chemin optimal passant par x_2 =
chemin optimal de x_0 à x_2 + chemin optimal de x_2 à x_8

GPS : idée pour calculer le plus court chemin

Problème : aller le plus rapidement de x_0 à x_8 :

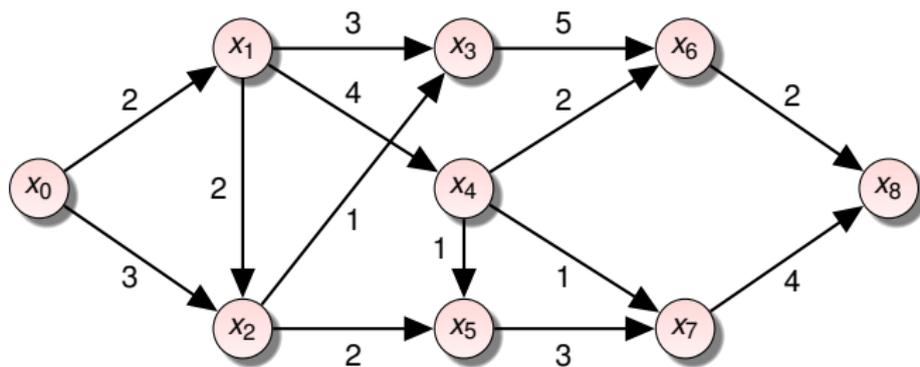


- **Idée :** Si chemin passe par x_2 suivant le chemin bleu
Chemin optimal = Chemin rouge puis bleu ou
violet puis bleu ?

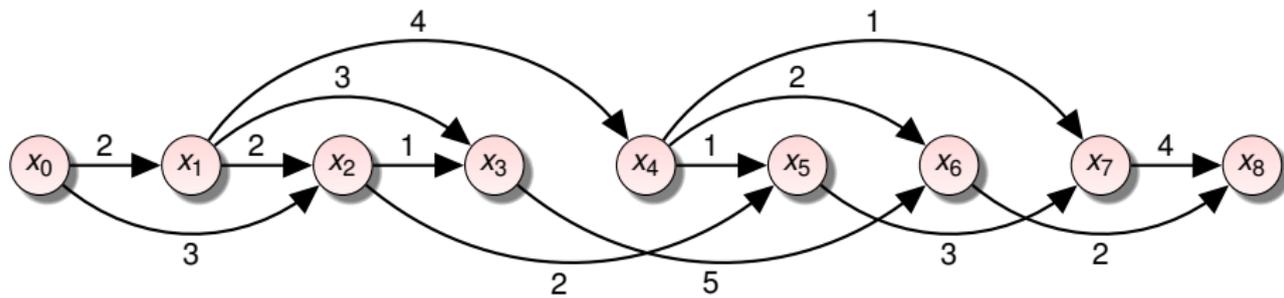
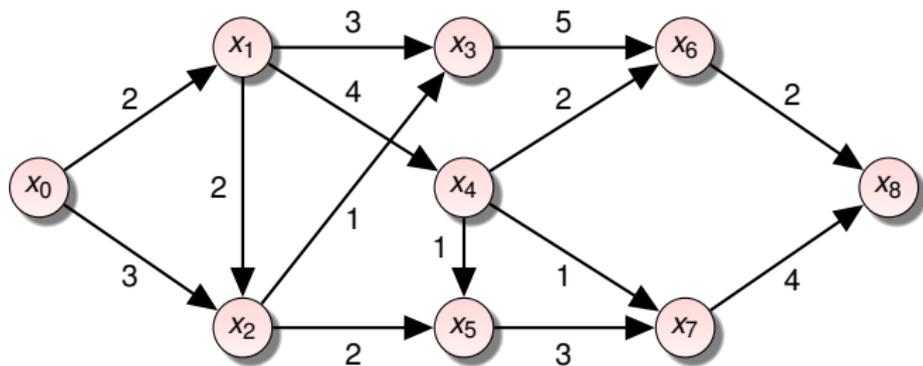
Chemin optimal passant par x_2 =
chemin optimal de x_0 à x_2 + chemin optimal de x_2 à x_8

⇒ Plus rapide que de calculer tous les chemins passant par x_2 !

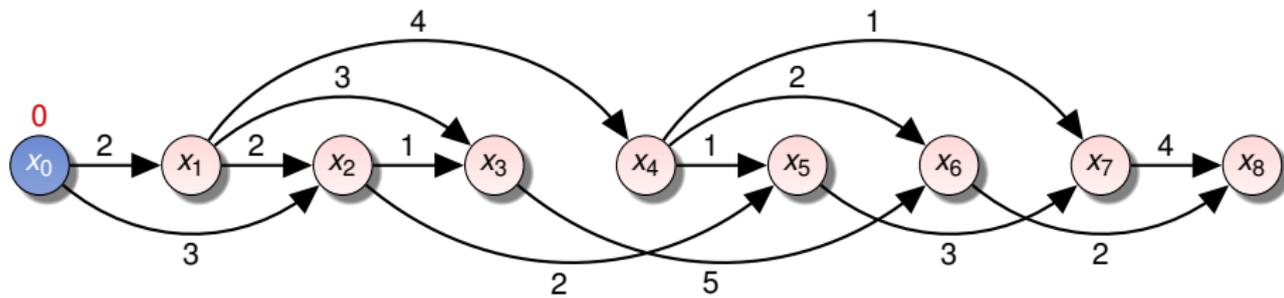
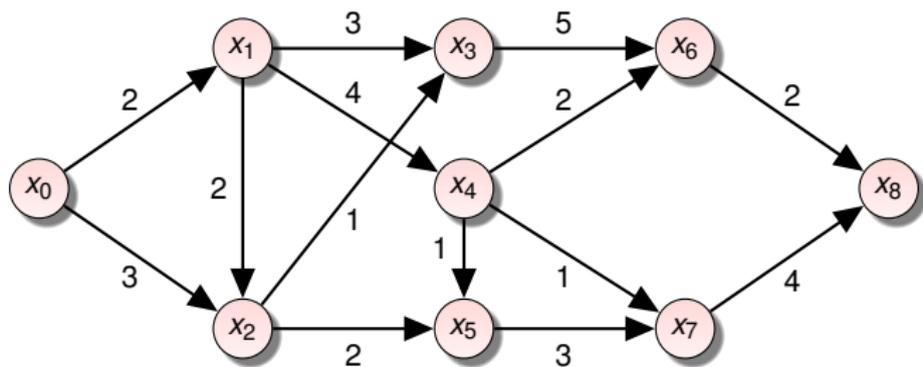
Calcul rapide du plus court chemin



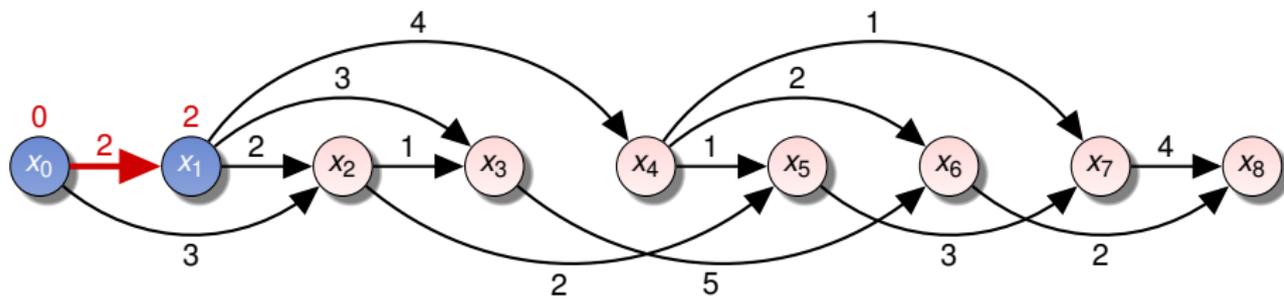
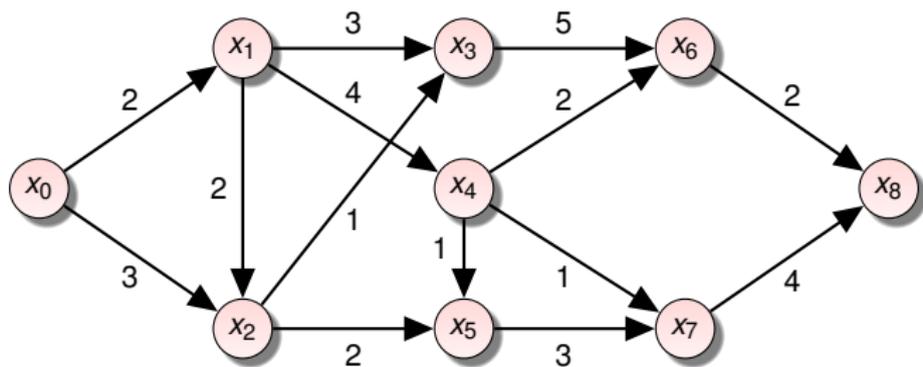
Calcul rapide du plus court chemin



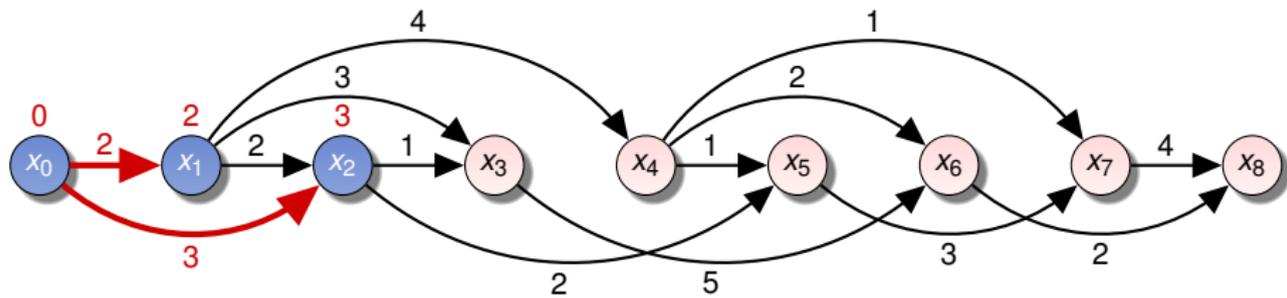
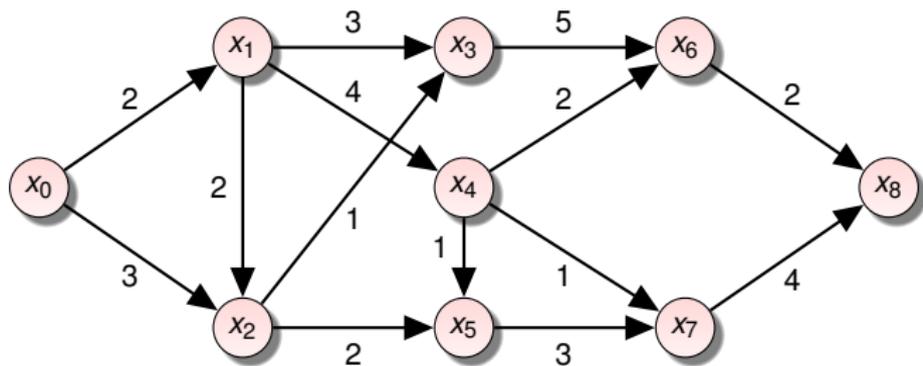
Calcul rapide du plus court chemin



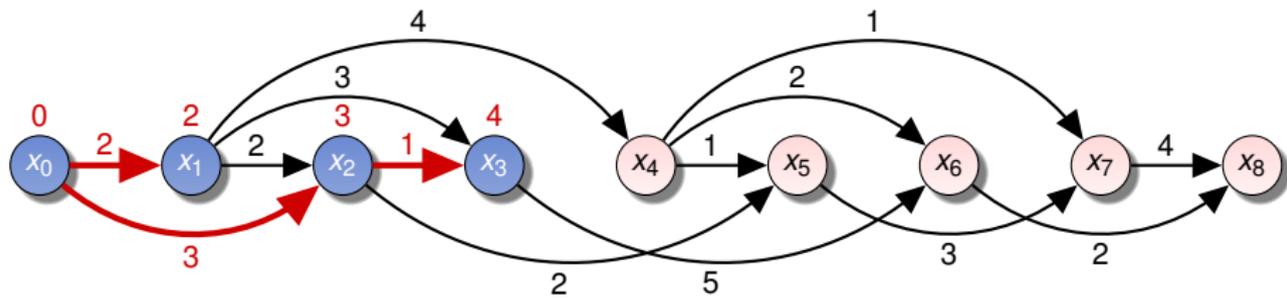
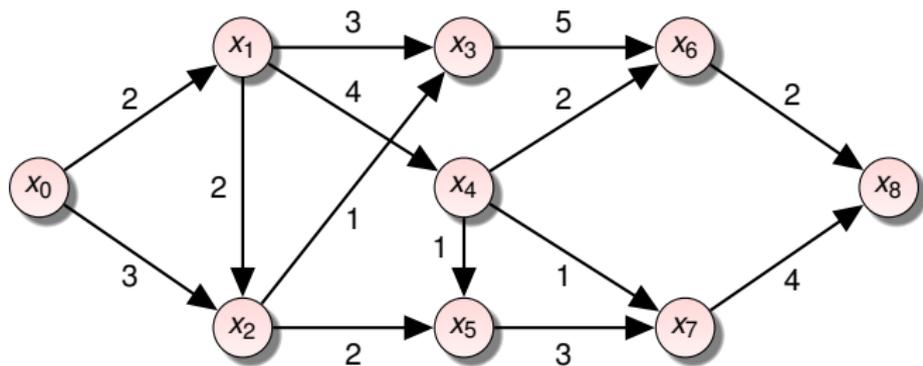
Calcul rapide du plus court chemin



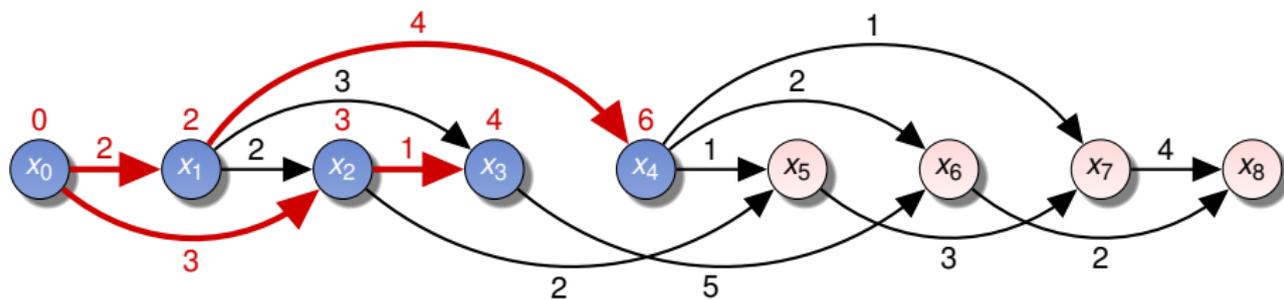
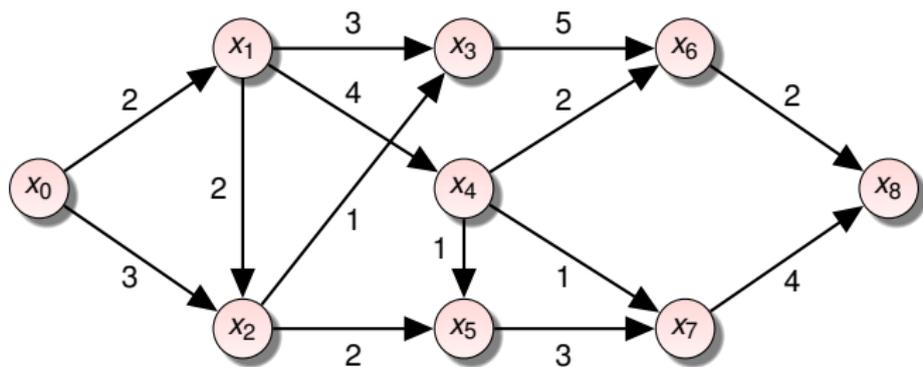
Calcul rapide du plus court chemin



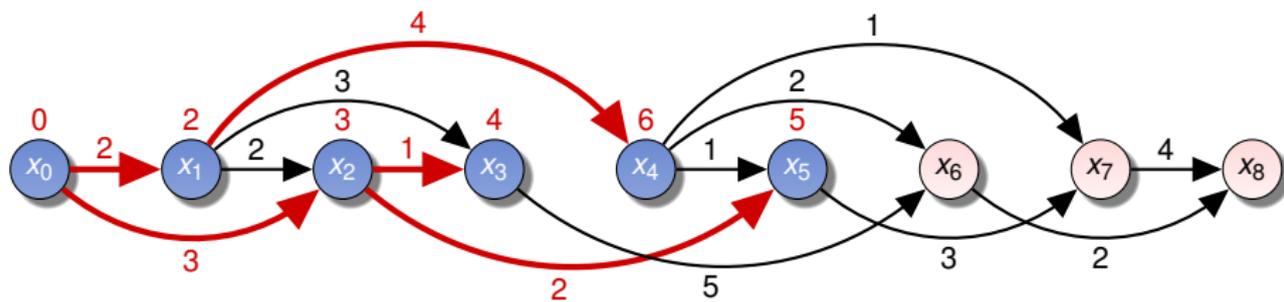
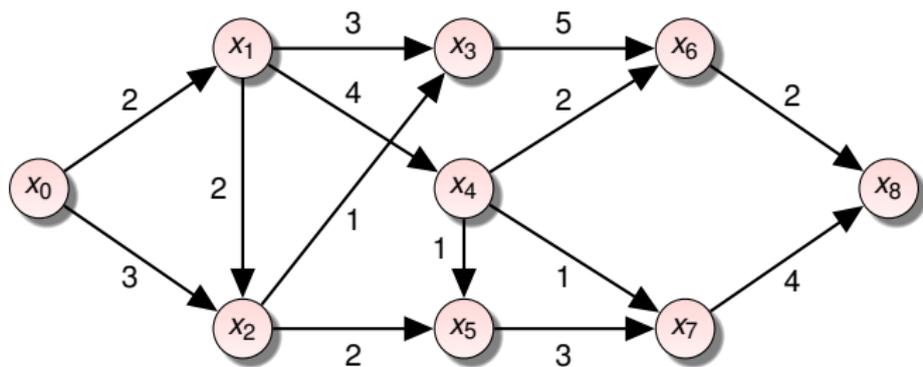
Calcul rapide du plus court chemin



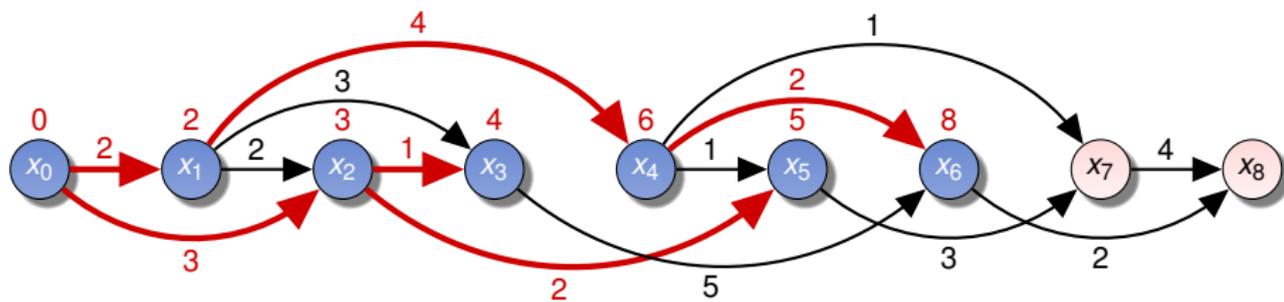
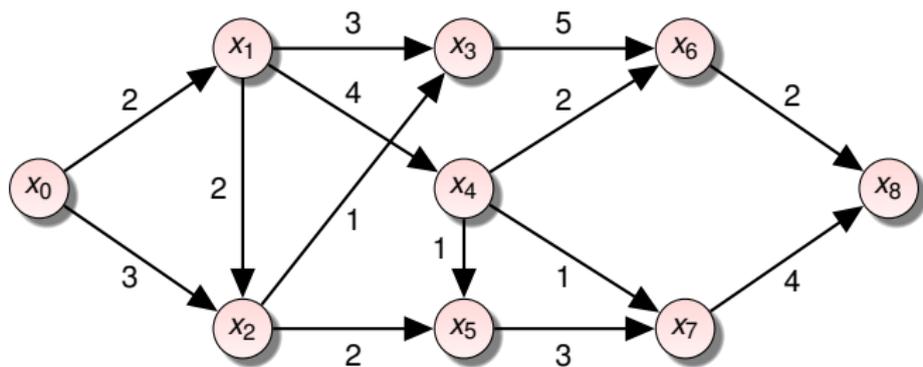
Calcul rapide du plus court chemin



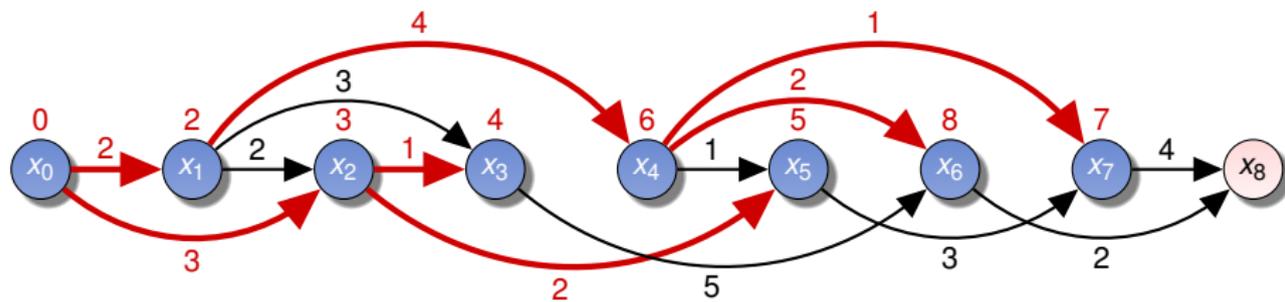
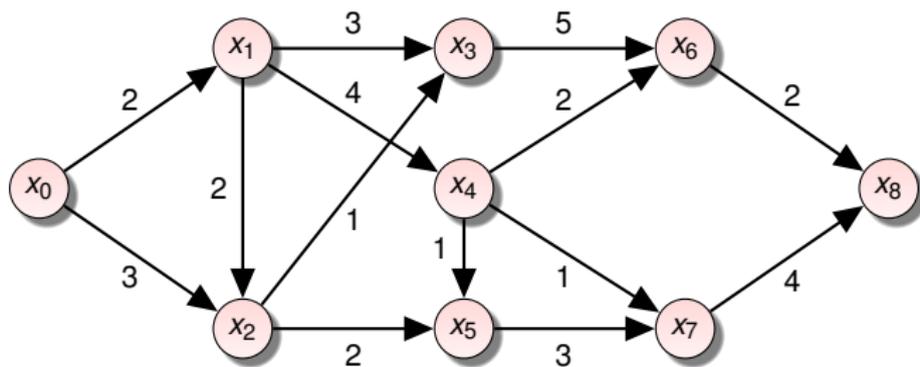
Calcul rapide du plus court chemin



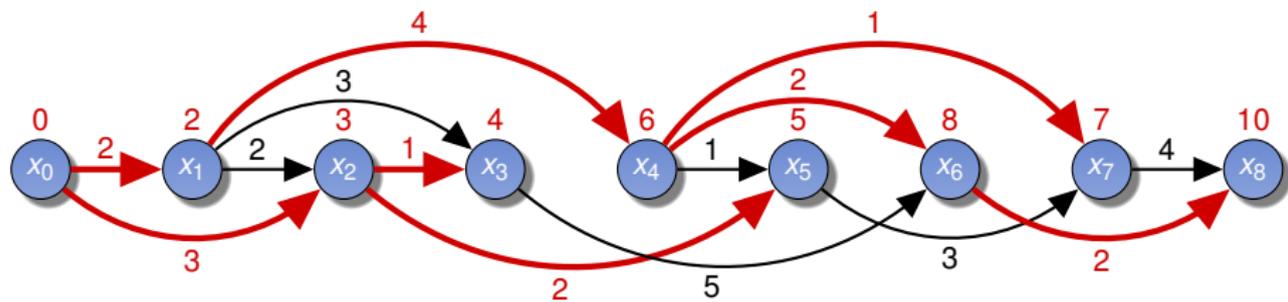
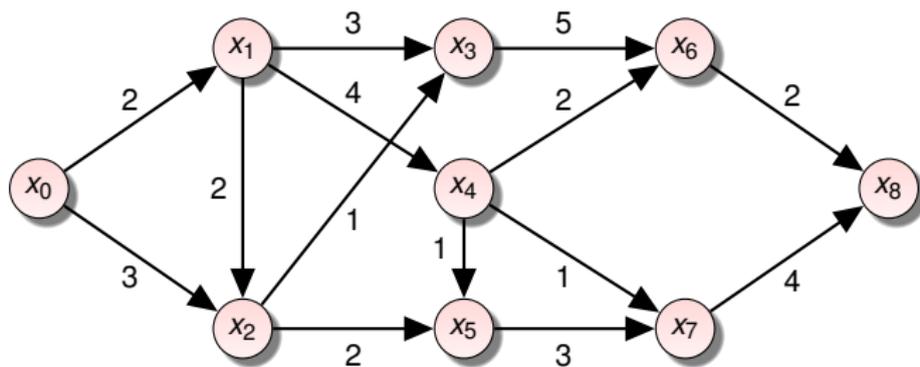
Calcul rapide du plus court chemin



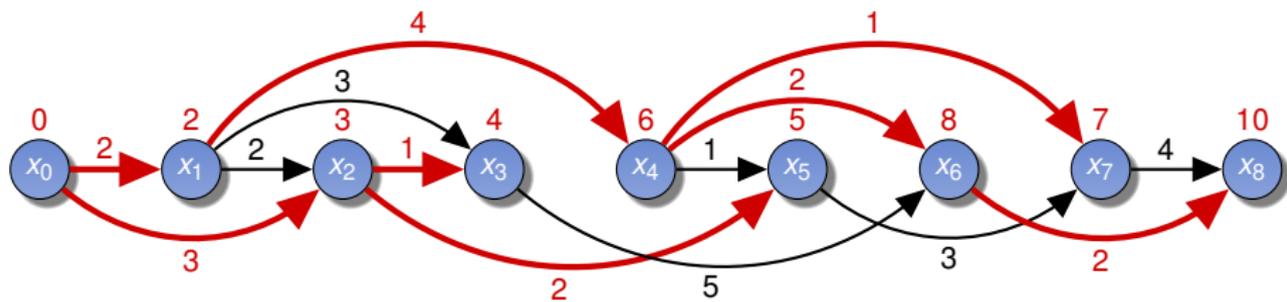
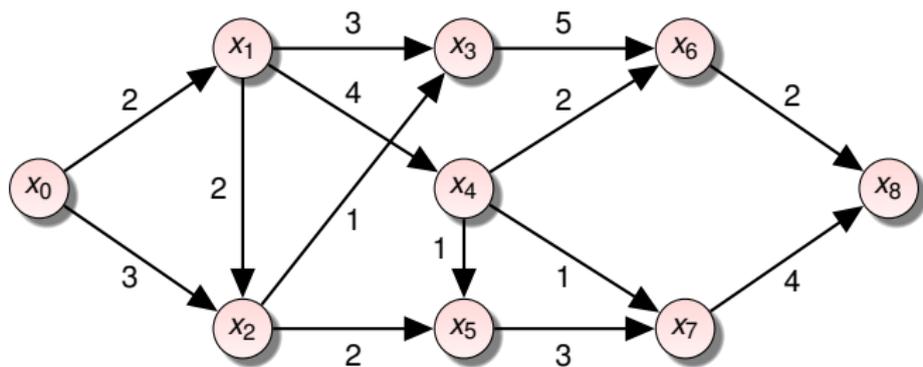
Calcul rapide du plus court chemin



Calcul rapide du plus court chemin



Calcul rapide du plus court chemin



Chemin optimal : $x_8 \leftarrow x_6 \leftarrow x_4 \leftarrow x_1 \leftarrow x_0$

Algorithme rapide du plus court chemin (1/2)

- ▶ V : ensemble des nœuds, E : ensemble des arcs
- ▶ Poids : tableau contenant le temps de trajet de chaque arc

```
1 fonction init (V, Temps, Parent) :
2   n ← nombre de nœuds de V
3   pour i variant de 0 à n-1 faire #  $i \iff$  noeud  $x_i$ 
4     Temps[i] ←  $+\infty$  # temps pour aller de  $x_0$  à  $x_i$ 
5     Parent[i] ← -1 # -1 = parent de  $x_i$  non affecté
6   fait
7   Temps[0] ← 0
```

```
1 fonction calc_parent (V, E, Poids, Temps, Parent) :
2   init (V, Temps, Parent)
3   n ← nombre de nœuds de V
4   pour i variant de 1 à n-1 faire
5     pour tout j tel que  $(x_j, x_i) \in E$  faire # arc  $(x_j, x_i)$  existe
6       si Temps[j] + Poids[j, i] < Temps[i] alors
7         Temps[i] ← Temps[j] + Poids[j, i]
8         Parent[i] ← j
9     finsi
10  fait
11  fait
12  # ici, Temps[i] = temps optimal pour aller de  $x_0$  à  $x_i$ 
13  # Parent[i] = noeud précédant  $x_i$  sur le chemin optimal
```

Algorithme rapide du plus court chemin (2/2)

```
1  fonction plus_court_chemin (V, E, Poids) :
2    n ← nombre de nœuds de V
3    créer deux tableaux Temps et Parent de longueur n
4
5    # phase 1 : on calcule les arcs rouges de la
6    # gauche vers la droite
7    calc_parent (V, E, Poids, Temps, Parent)
8
9    # phase 2 : on parcourt ces arcs de la droite
10   # vers la gauche
11   Chemin ← {n-1} # noeud arrivée
12   i = n-1
13   tant que i ≠ 0 faire
14     Chemin ← {Parent[i]} ∪ Chemin
15     i ← Parent[i]
16   fait
17
18   retourner Chemin
```

Algorithme rapide du plus court chemin (2/2)

```
1  fonction plus_court_chemin (V, E, Poids) :
2    n ← nombre de nœuds de V
3    créer deux tableaux Temps et Parent de longueur n
4
5    # phase 1 : on calcule les arcs rouges de la
6    # gauche vers la droite
7    calc_parent (V, E, Poids, Temps, Parent)
8
9    # phase 2 : on parcourt ces arcs de la droite
10   # vers la gauche
11   Chemin ← {n-1} # noeud arrivée
12   i = n-1
13   tant que i ≠ 0 faire
14     Chemin ← {Parent[i]} ∪ Chemin
15     i ← Parent[i]
16   fait
17
18   retourner Chemin
```

- Parcours optimal de x_0 à x_k , $k \neq n-1$:
remplacer $n-1$ sur les lignes 11 et 12 par k .

Programmation dynamique : 2 phases

Phase 1

- ▶ On calcule itérativement les informations utiles pour la solution optimale

Programmation dynamique : 2 phases

Phase 1

- ▶ On calcule itérativement les informations utiles pour la solution optimale
- ▶ Pour cela : formule mathématique

Exemple du plus court chemin : En chaque noeud x_i :

$$\begin{cases} \text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\} \\ \text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\} \end{cases}$$

Programmation dynamique : 2 phases

Phase 1

- ▶ On calcule itérativement les informations utiles pour la solution optimale
- ▶ Pour cela : formule mathématique

Exemple du plus court chemin : En chaque noeud x_i :

$$\begin{cases} \text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\} \\ \text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\} \end{cases}$$

- ▶ Formule mathématique simple

Programmation dynamique : 2 phases

Phase 1

- ▶ On calcule itérativement les informations utiles pour la solution optimale

- ▶ Pour cela : formule mathématique

Exemple du plus court chemin : En chaque noeud x_i :

$$\begin{cases} \text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\} \\ \text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\} \end{cases}$$

- ▶ Formule mathématique simple
- ▶ Calculs effectués dans un ordre précis :

Exemple du plus court chemin : de la gauche vers la droite

Programmation dynamique : 2 phases

Phase 1

- ▶ On calcule itérativement les informations utiles pour la solution optimale
- ▶ Pour cela : formule mathématique

Exemple du plus court chemin : En chaque noeud x_i :

$$\begin{cases} \text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\} \\ \text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\} \end{cases}$$

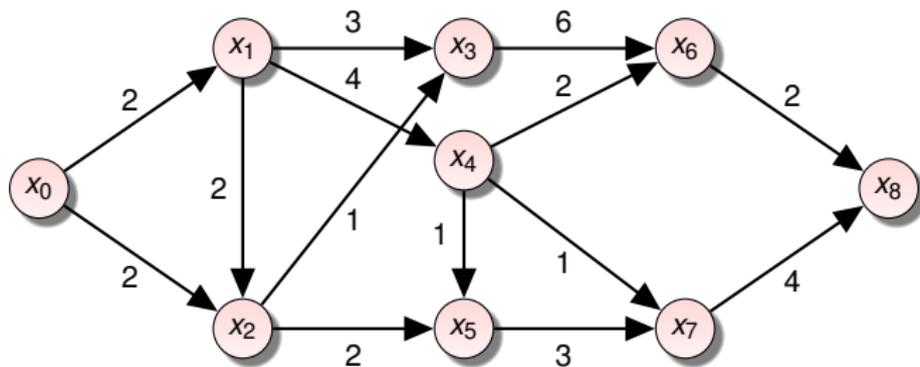
- ▶ Formule mathématique simple
- ▶ Calculs effectués dans un ordre précis :

Exemple du plus court chemin : de la gauche vers la droite

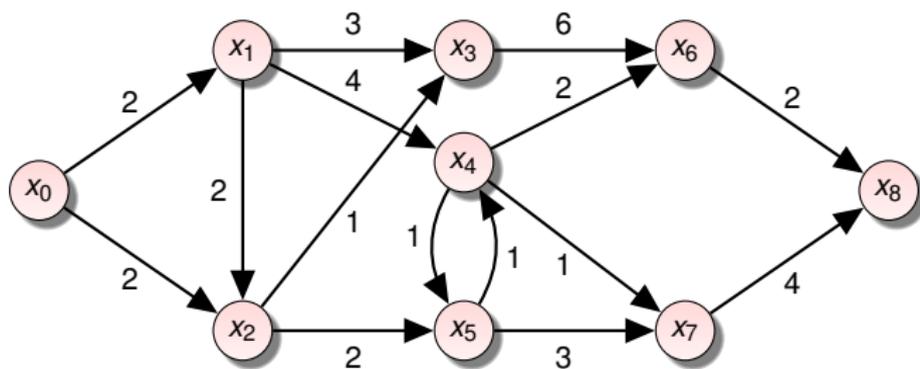
Phase 2

- ▶ On utilise les infos de la phase 1 pour calculer la solution optimale
- ▶ Calculs itératifs dans l'ordre inverse de la phase 1

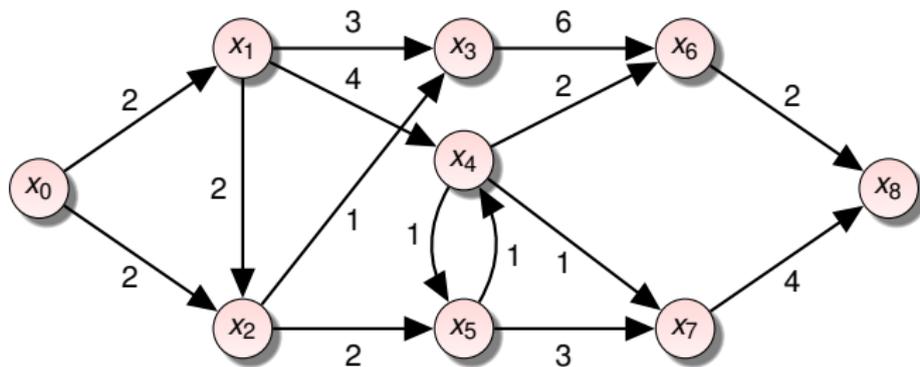
Plus court chemin : 2 sens de circulation



Plus court chemin : 2 sens de circulation

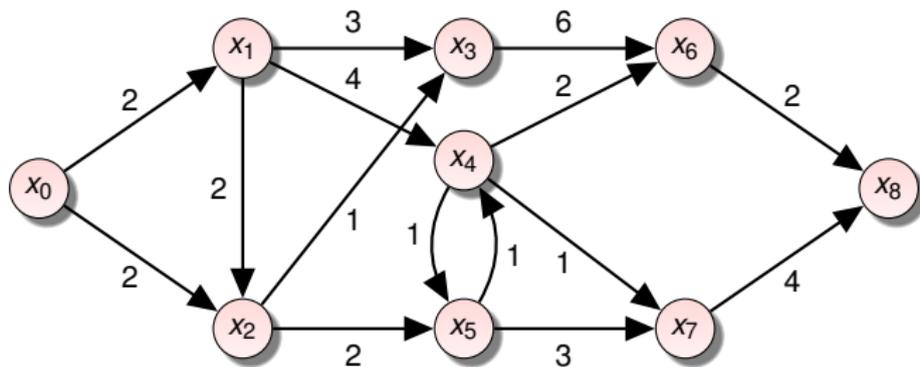


Plus court chemin : 2 sens de circulation



\implies Il n'existe plus d'ordre topologique !

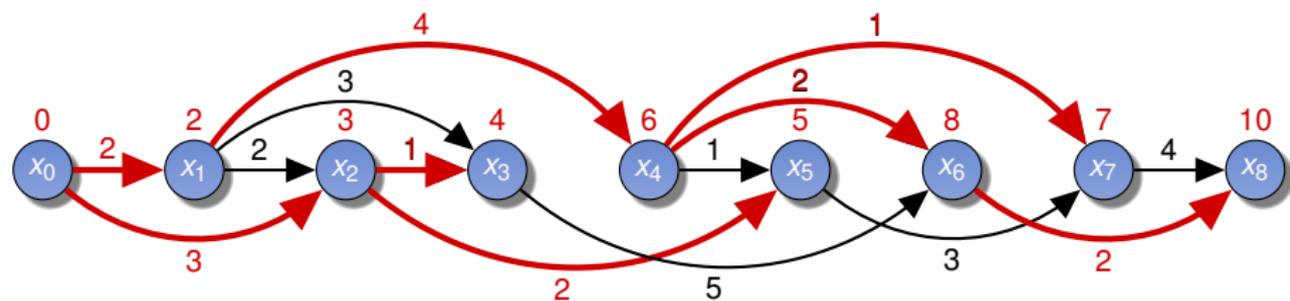
Plus court chemin : 2 sens de circulation



⇒ Il n'existe plus d'ordre topologique !

⇒ Notre algo de plus court chemin ne fonctionne plus.

Retour sur les sens uniques (1/2)

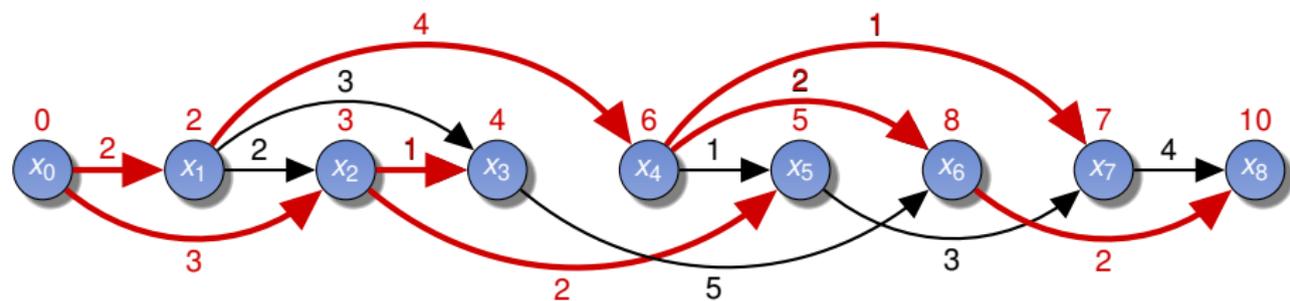


► Algorithme : en chaque noeud x_i , on calcule :

$$\text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\}$$

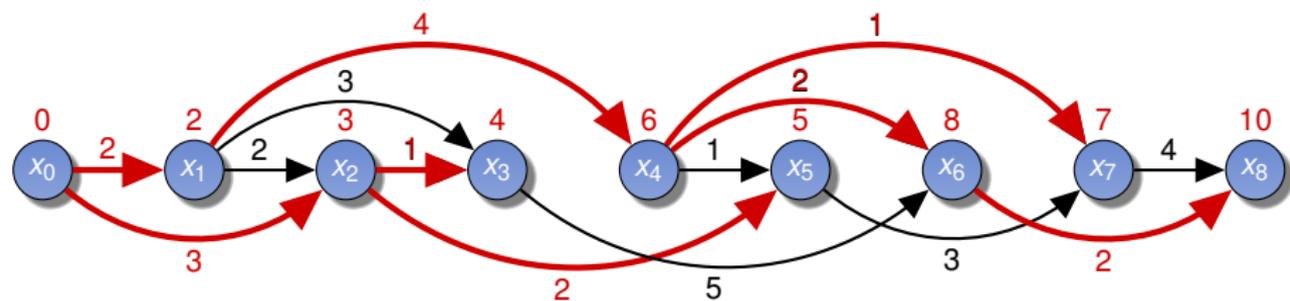
$$\text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\}$$

Retour sur les sens uniques (1/2)



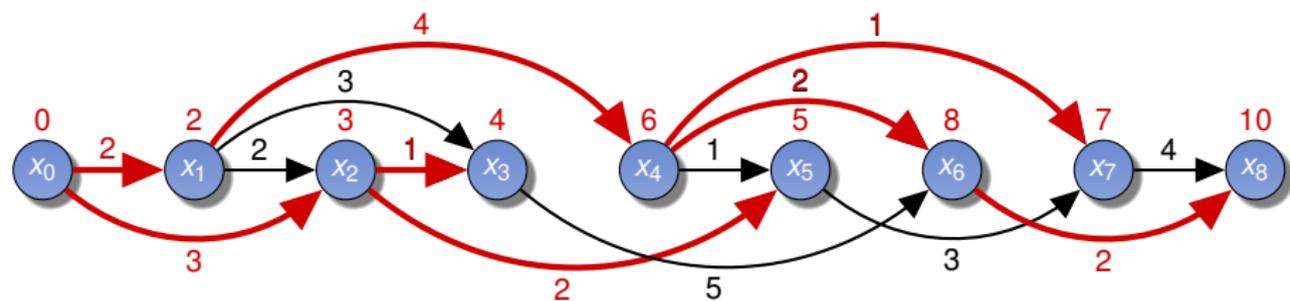
- ▶ Algorithme : en chaque noeud x_i , on calcule :
$$\text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\}$$
$$\text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\}$$
- ▶ Calcul correct car les $\text{Temps}(x_j)$ précédents sont corrects

Retour sur les sens uniques (1/2)



- ▶ Algorithme : en chaque noeud x_i , on calcule :
$$\text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\}$$
$$\text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\}$$
- ▶ Calcul correct car les $\text{Temps}(x_j)$ précédents sont corrects
- ▶ L'ordre topologique garantit qu'en chaque x_i ce calcul est juste.

Retour sur les sens uniques (1/2)

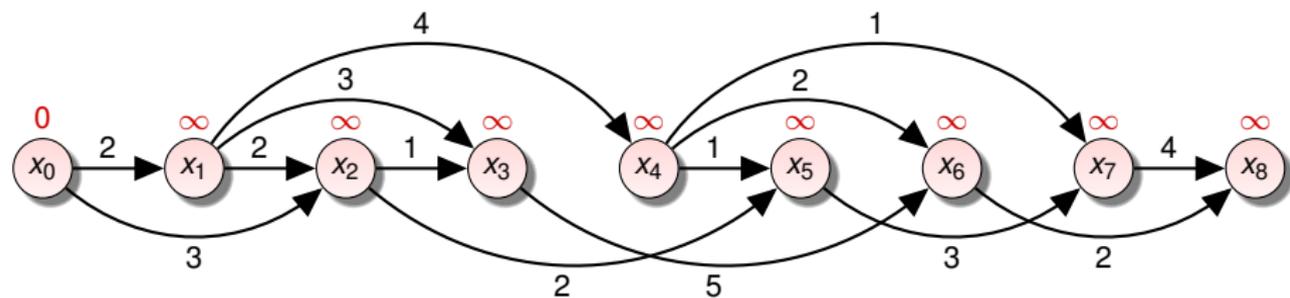


- ▶ Algorithme : en chaque noeud x_i , on calcule :
$$\text{Temps}(x_i) = \min\{\text{Temps}(x_j) + \text{Poids}(\text{arc}(x_j, x_i)) : (x_j, x_i) \in E\}$$
$$\text{Parent}(x_i) = \text{Argmin}\{\text{Temps}(x_j) + \text{Poids}((x_j, x_i)) : (x_j, x_i) \in E\}$$
- ▶ Calcul correct car les $\text{Temps}(x_j)$ précédents sont corrects
- ▶ L'ordre topologique garantit qu'en chaque x_i ce calcul est juste.

Ordre topologique : utiliser les temps précédents pour mettre à jour $\text{Temps}(x_i)$

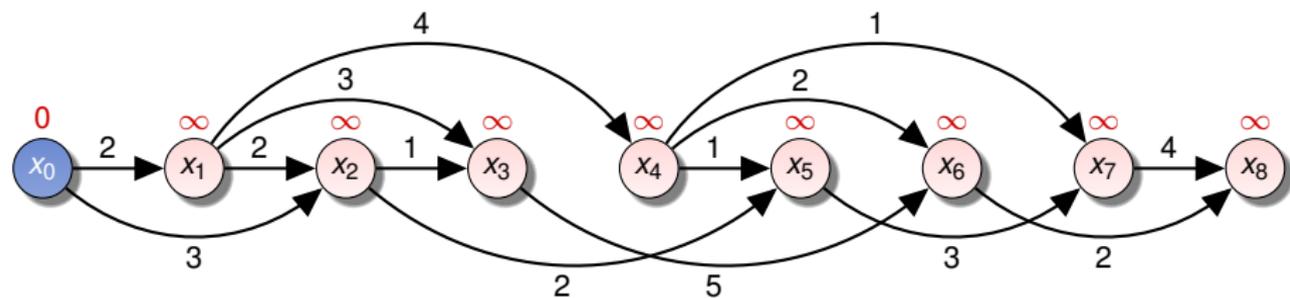
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Autre méthode pour calculer $\text{Temps}(x_i)$

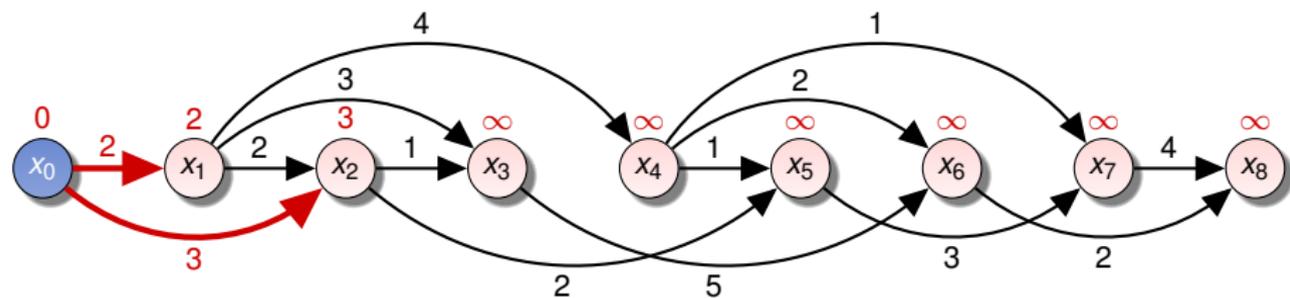
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

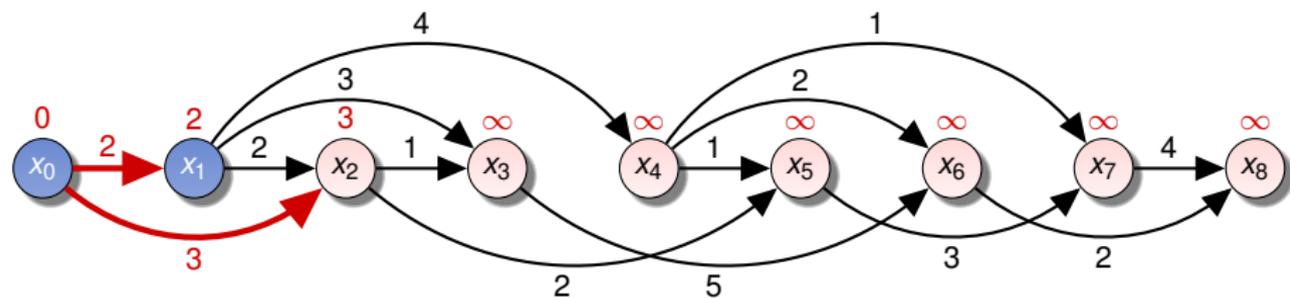
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



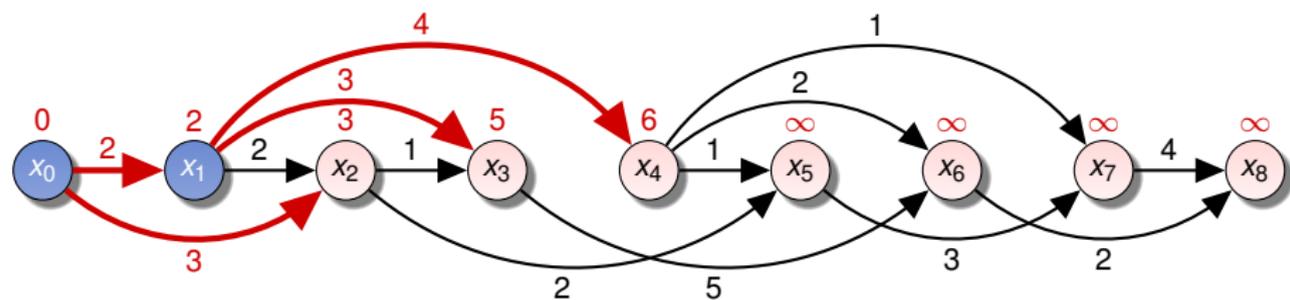
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Autre méthode pour calculer $\text{Temps}(x_i)$

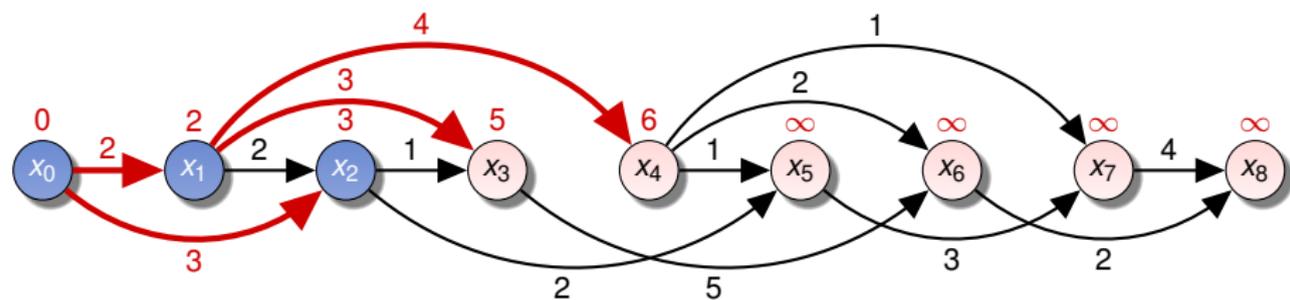
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

Autre méthode pour calculer $\text{Temps}(x_i)$

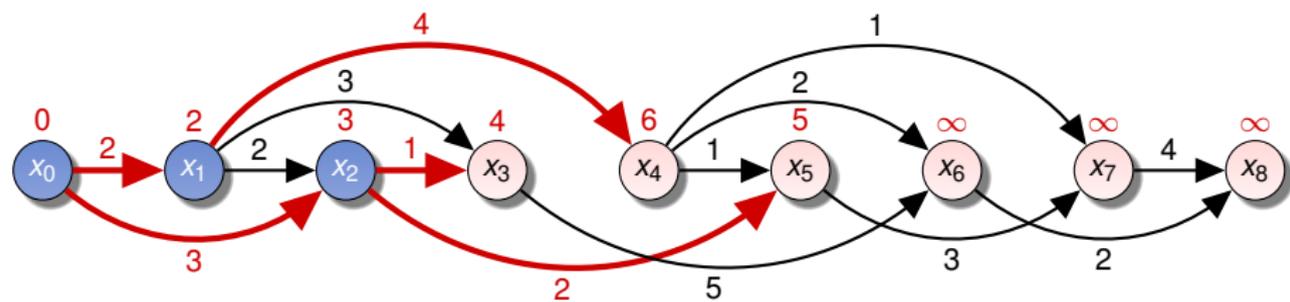
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

Autre méthode pour calculer $\text{Temps}(x_i)$

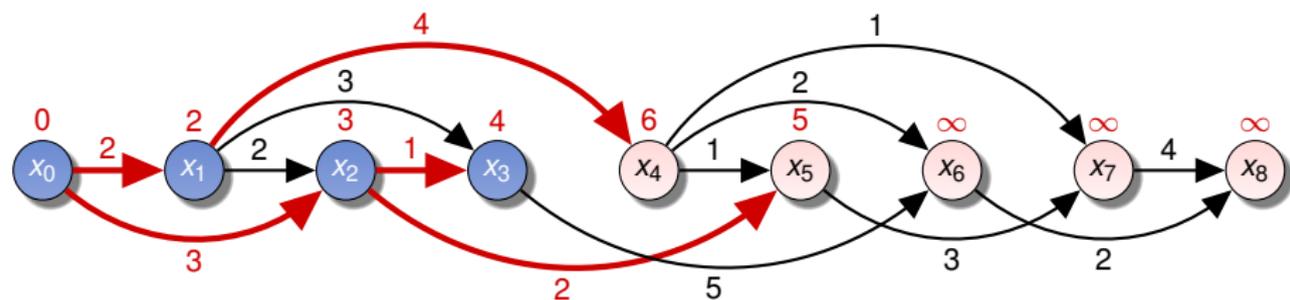
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

Autre méthode pour calculer $\text{Temps}(x_i)$

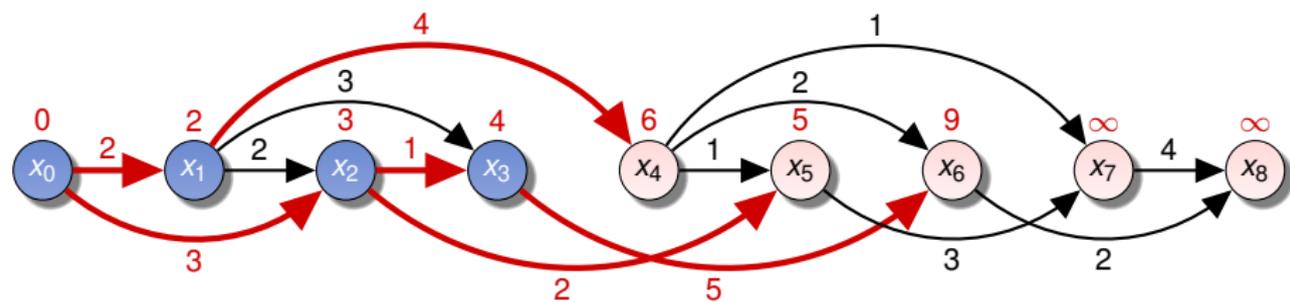
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

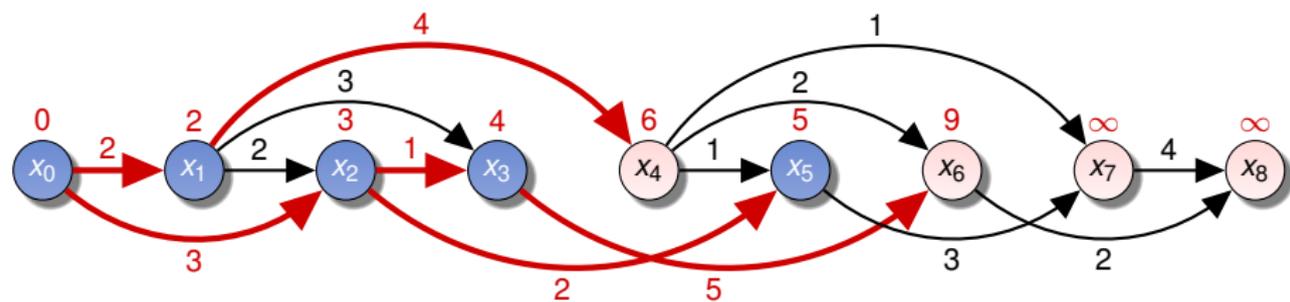
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Autre méthode pour calculer $\text{Temps}(x_i)$

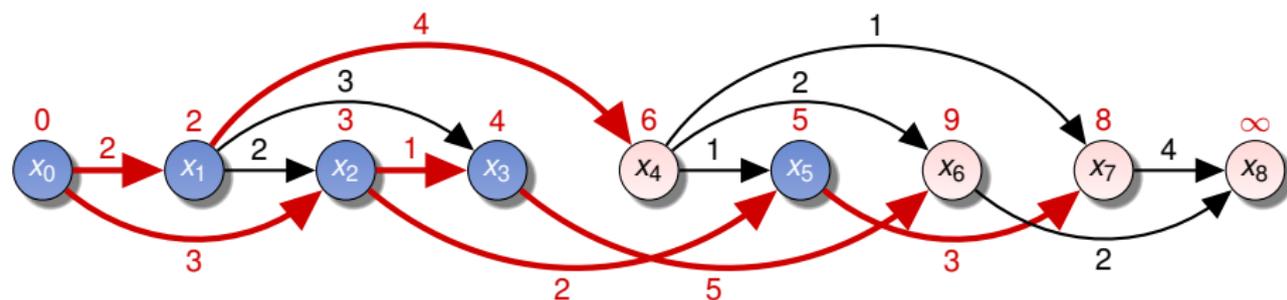
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

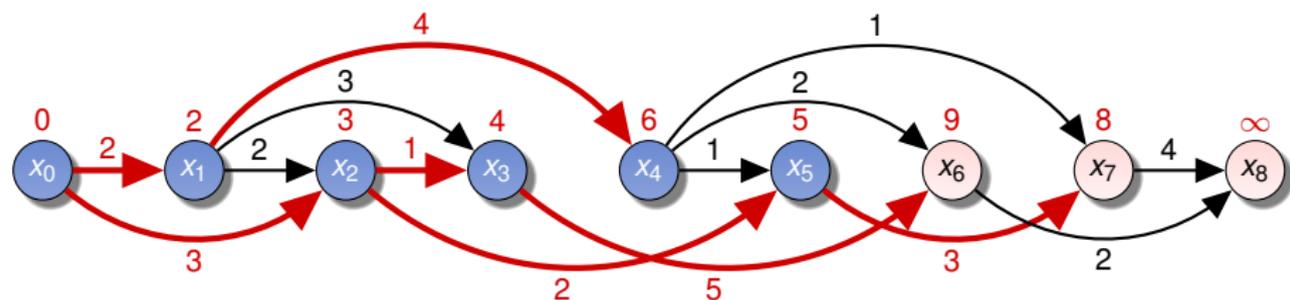
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



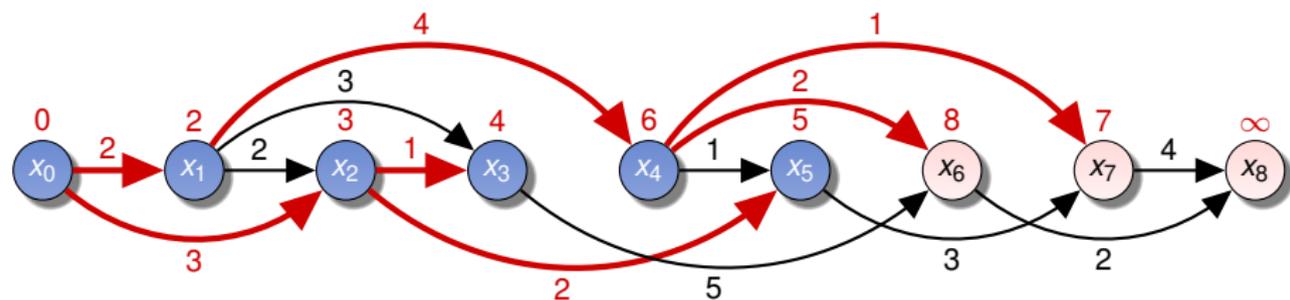
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



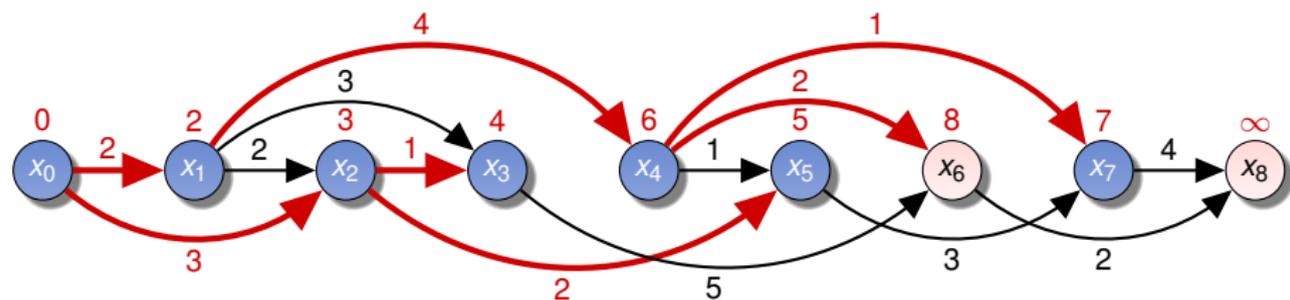
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



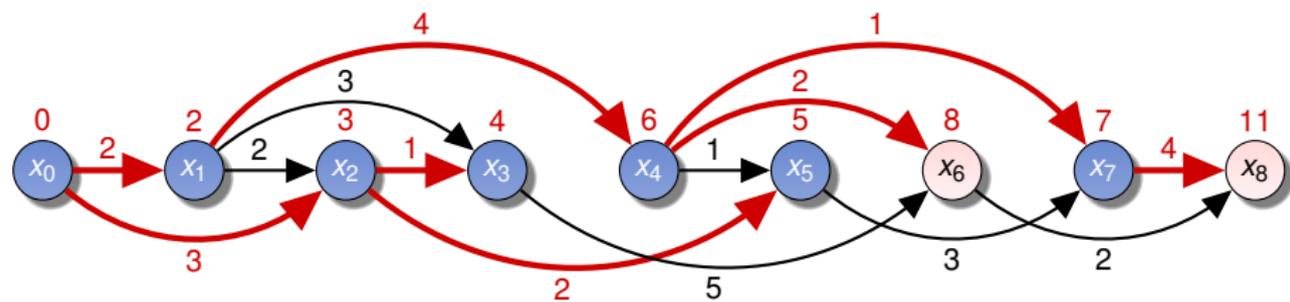
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



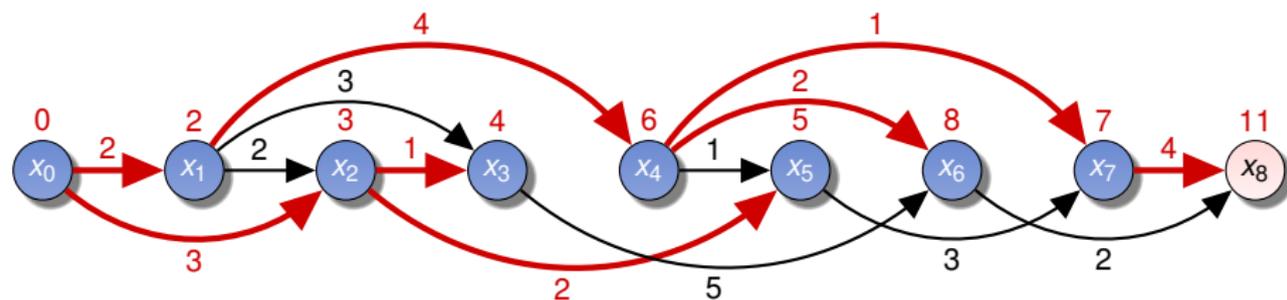
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



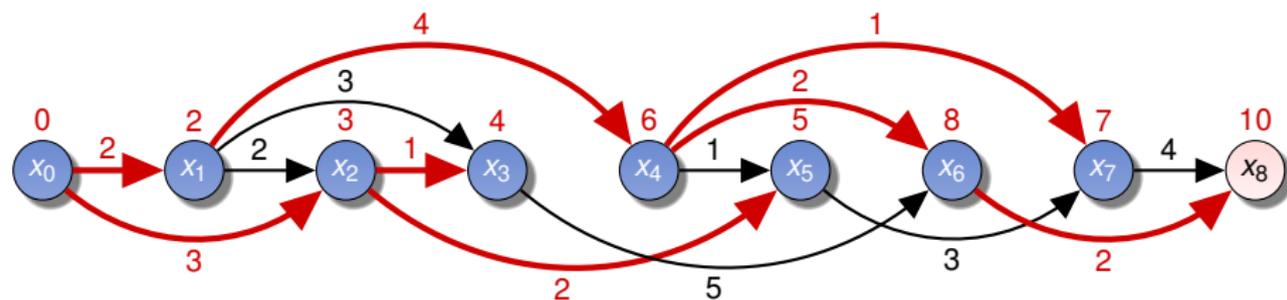
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



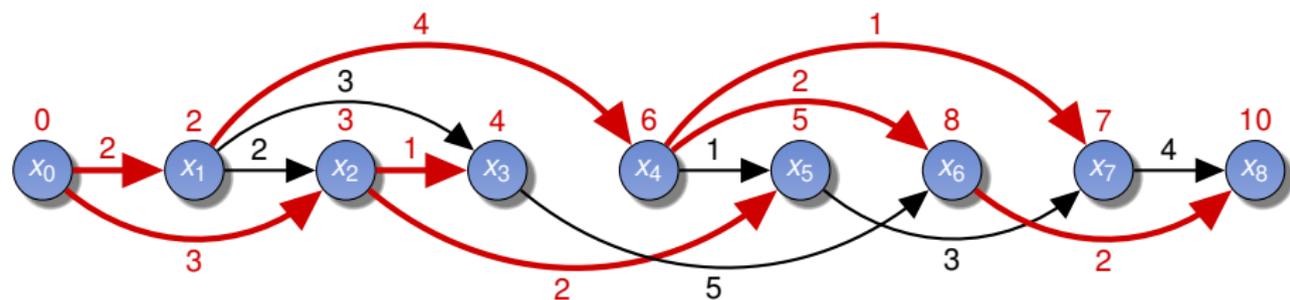
Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Autre méthode pour calculer $\text{Temps}(x_i)$

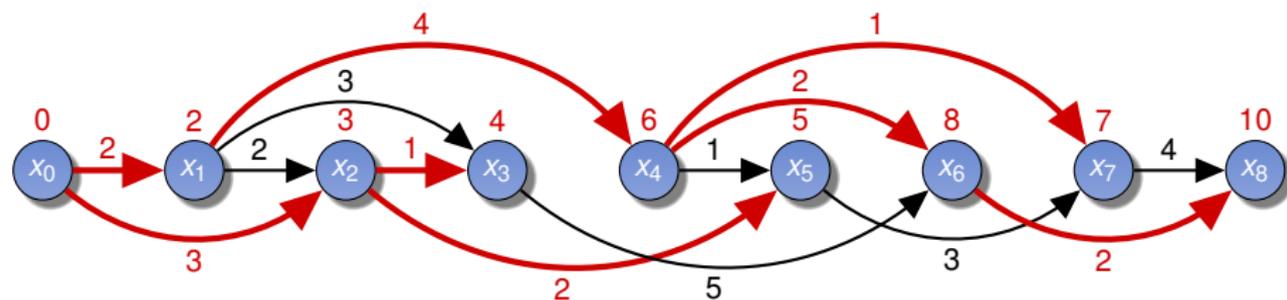
- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Retour sur les sens uniques (2/2)

Autre méthode pour calculer $\text{Temps}(x_i)$

- 1 Appliquer la fonction `init` pour initialiser les Temps
- 2 Choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum
- 3 Mettre à jour les nœud x_k enfants de x_i :
$$\text{Temps}(x_k) \leftarrow \min\{\text{Temps}(x_k), \text{Temps}(x_i) + \text{Poids}(x_i, x_k)\}$$
- 4 Revenir en 2



Chemin optimal : $x_8 \leftarrow x_6 \leftarrow x_4 \leftarrow x_1 \leftarrow x_0$

Pourquoi cela fonctionne ?

Règle : choisir le nœud x_i rose dont $\text{Temps}(x_i)$ est minimum

Invariant de boucle

- ▶ À chaque étape :
 - ▶ si x_i = nœud rose dont $\text{Temps}(x_i)$ minimum
 - ▶ alors remonter de x_i à x_0 en passant par les arcs rouges = chemin optimal de x_0 à x_i
⇒ $\text{Temps}(x_i)$ = temps du trajet optimal de x_0 à x_i

- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus

- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus
- ▶ Si $\text{Temps}(x_i) \neq$ temps du chemin optimal de x_0 à x_i
 \implies il existe au moins un nœud rose sur ce chemin optimal C

- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus
- ▶ Si $\text{Temps}(x_i) \neq$ temps du chemin optimal de x_0 à x_i
 \implies il existe au moins un nœud rose sur ce chemin optimal C
Soit x_k le premier nœud rose de C

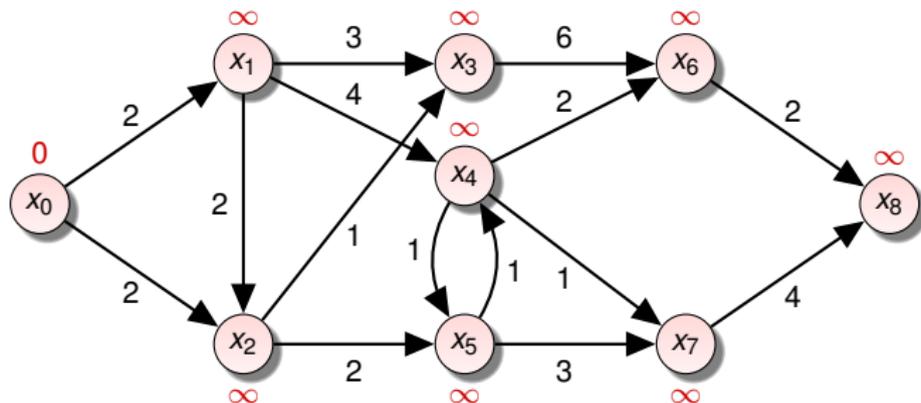
- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus
- ▶ Si $\text{Temps}(x_i) \neq$ temps du chemin optimal de x_0 à x_i
 - \implies il existe au moins un nœud rose sur ce chemin optimal C
 - Soit x_k le premier nœud rose de C
 - $\implies x_k$ a un parent bleu $\implies \text{Temps}(x_k) \neq +\infty$

- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus
- ▶ Si $\text{Temps}(x_i) \neq$ temps du chemin optimal de x_0 à x_i
 - \implies il existe au moins un nœud rose sur ce chemin optimal C
 - Soit x_k le premier nœud rose de C
 - $\implies x_k$ a un parent bleu $\implies \text{Temps}(x_k) \neq +\infty$
 - Mais $\text{Temps}(x_k) < \text{Temps}(x_i)$ car les poids des arcs sur le chemin de x_k à x_i sont > 0

- ▶ Démonstration par récurrence que $\text{Temps}(x_i) =$ temps du chemin optimal de x_0 à x_i *passant uniquement* par des nœuds bleus
- ▶ Si $\text{Temps}(x_i) \neq$ temps du chemin optimal de x_0 à x_i
 - \implies il existe au moins un nœud rose sur ce chemin optimal C
 - Soit x_k le premier nœud rose de C
 - $\implies x_k$ a un parent bleu $\implies \text{Temps}(x_k) \neq +\infty$
 - Mais $\text{Temps}(x_k) < \text{Temps}(x_i)$ car les poids des arcs sur le chemin de x_k à x_i sont > 0
 - \implies contradiction : on aurait dû choisir x_k et non x_i

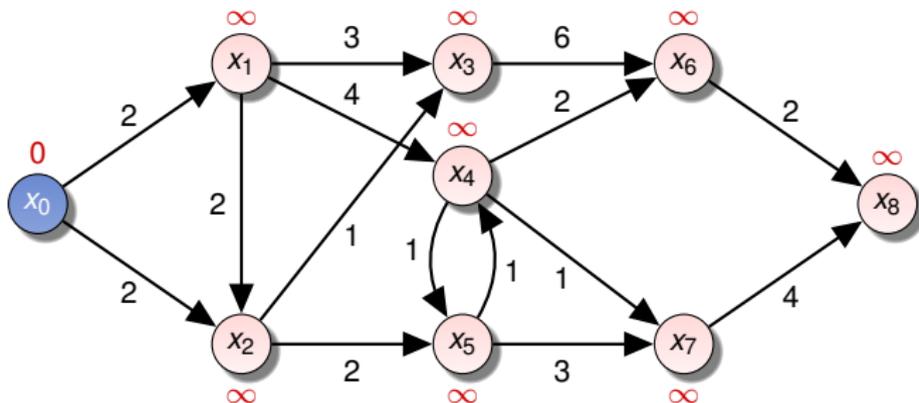
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



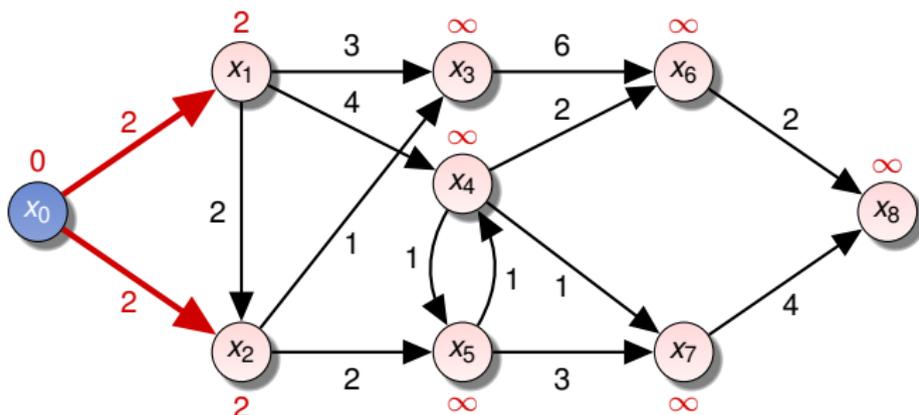
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



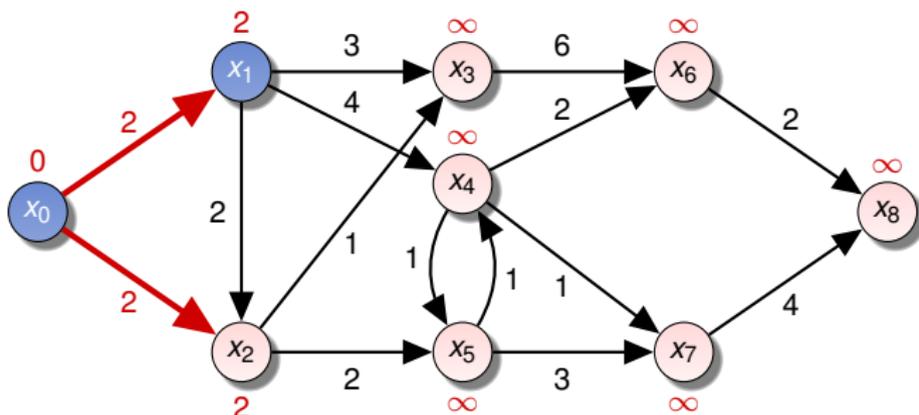
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



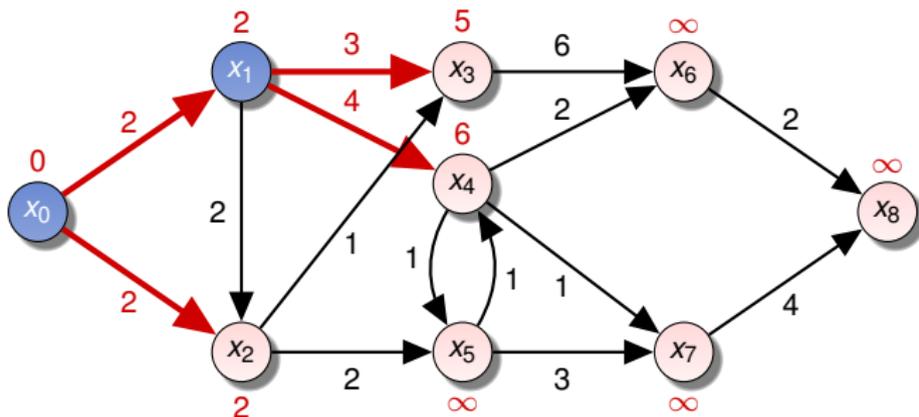
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



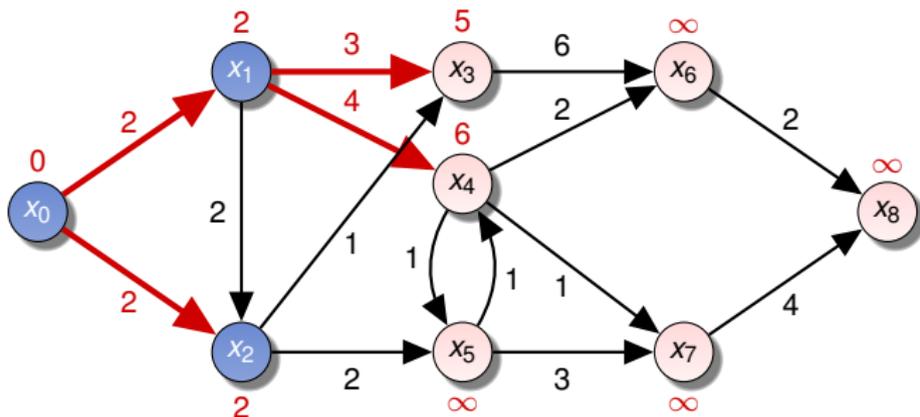
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



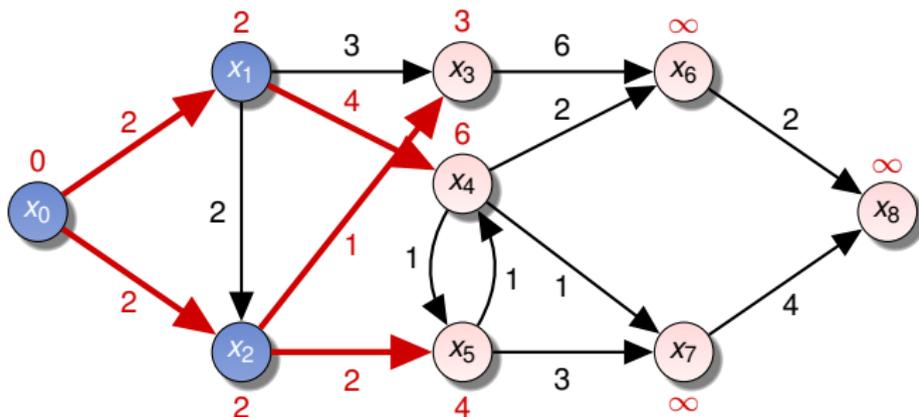
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



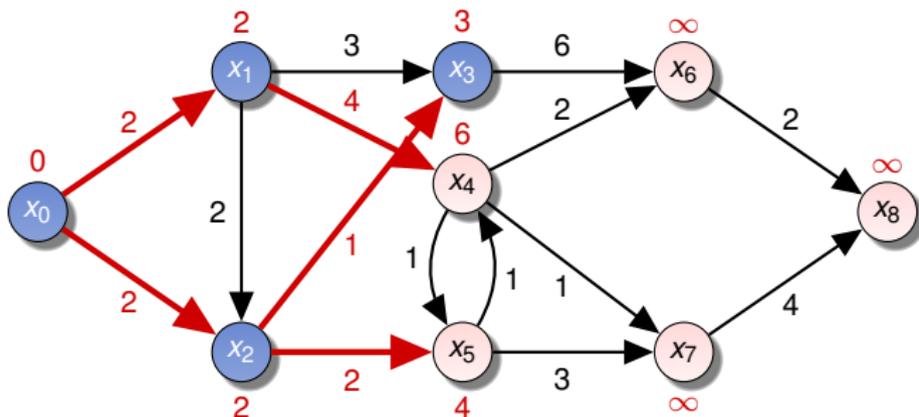
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



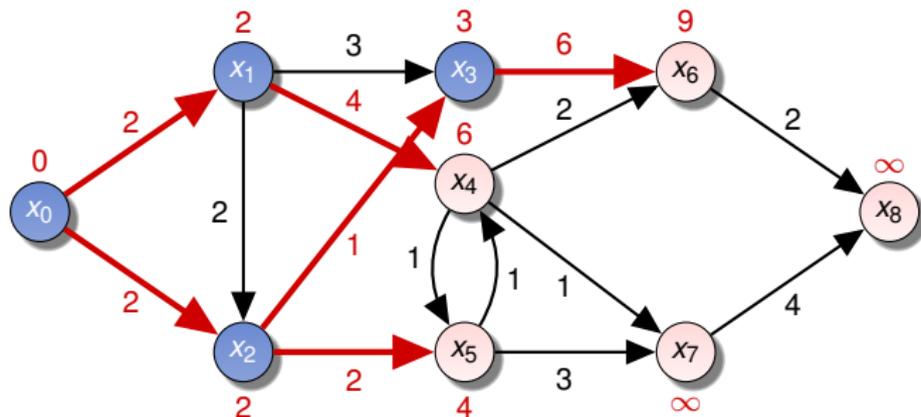
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



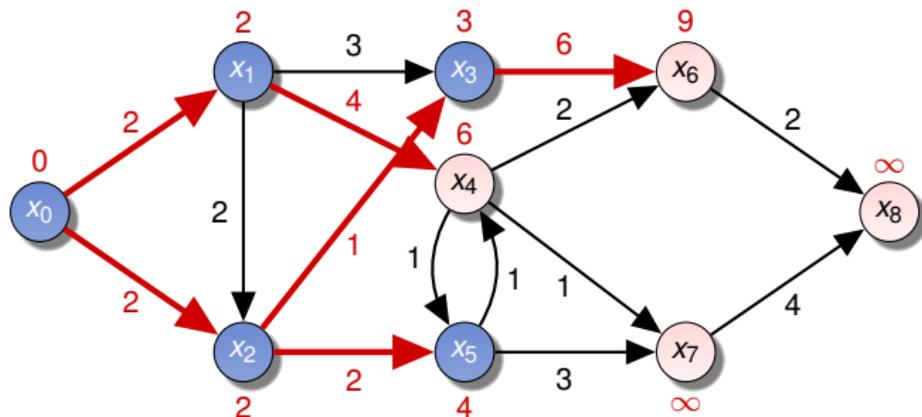
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



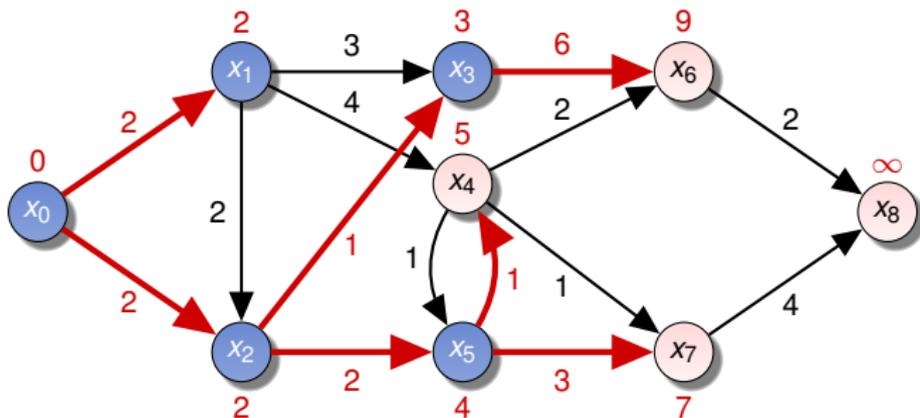
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



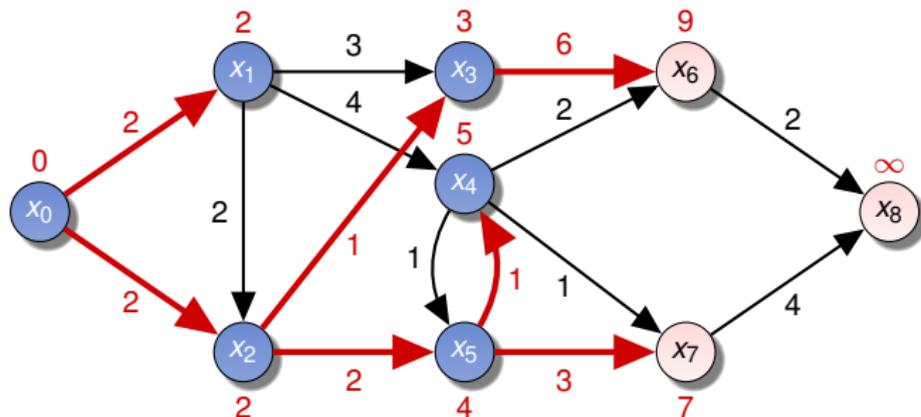
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



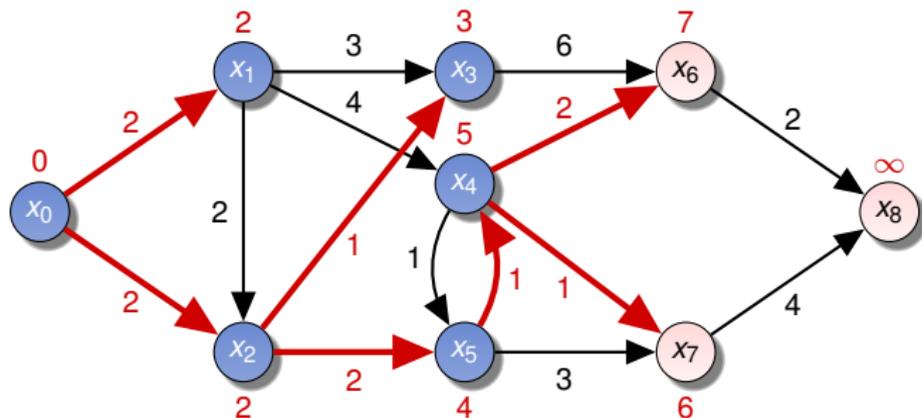
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



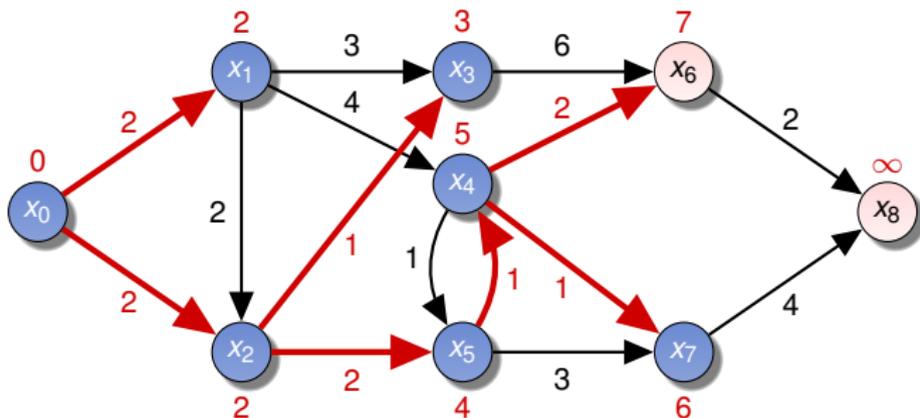
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



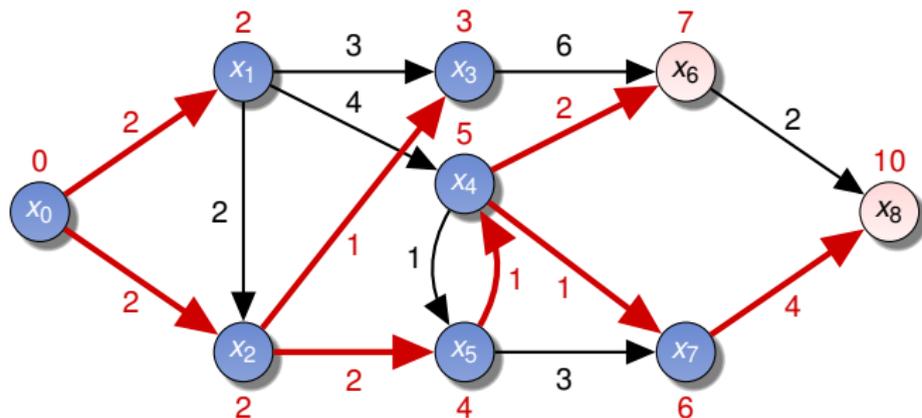
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



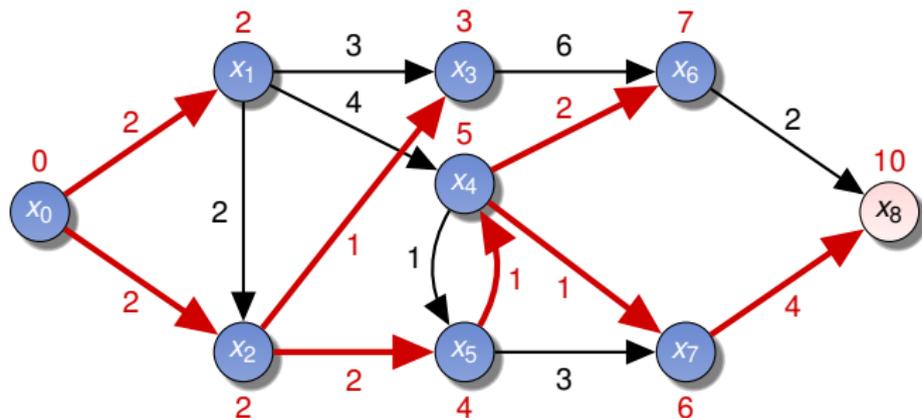
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



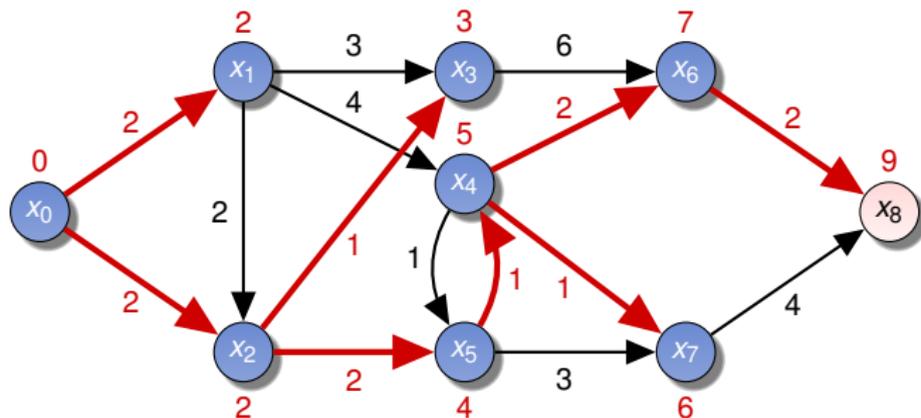
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



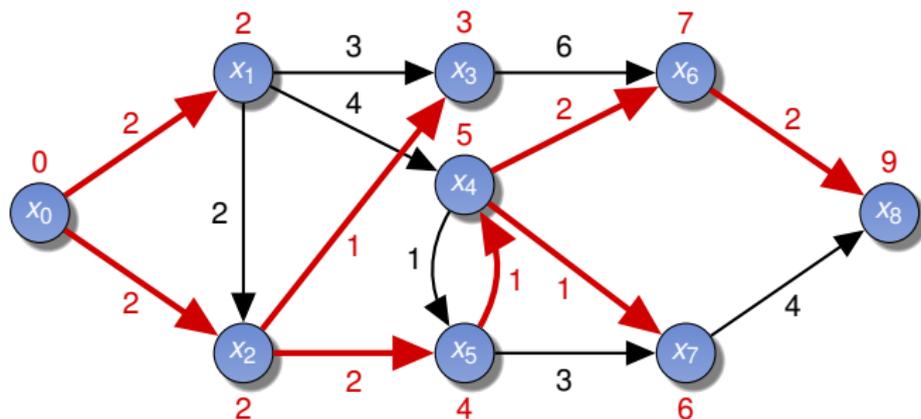
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



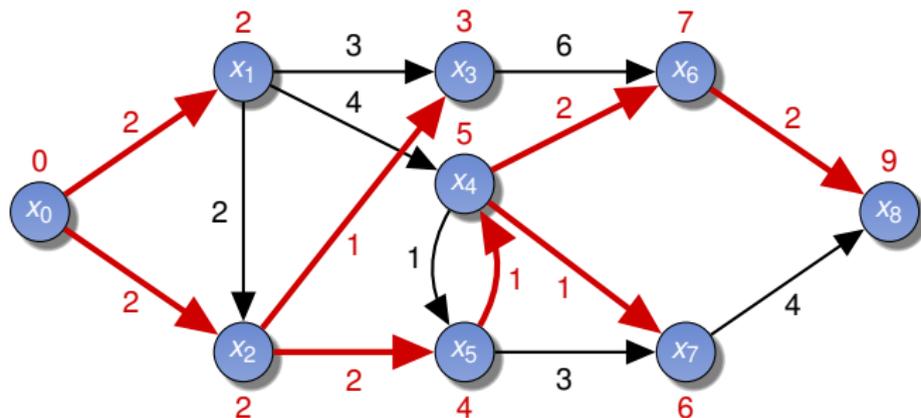
Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



Retour sur les 2 sens de circulation

On peut appliquer l'algorithme que l'on vient de voir !



Chemin optimal : $x_8 \leftarrow x_6 \leftarrow x_4 \leftarrow x_5 \leftarrow x_2 \leftarrow x_0$

Algorithme de Dijkstra (1959)

```
1  fonction calc_parent_Dijkstra (V, E, Poids,  
2                                Temps, Parent) :  
3      init (V, Temps, Parent)  
4      S ← ∅ # les noeuds en bleu  
5      Q ← V # les noeuds en rose  
6  
7      tant que Q ≠ ∅ faire  
8          # choisir un noeud rose i que l'on va  
9          # transformer en bleu  
10         soit i ∈ { noeuds v de Q t.q. Temps[v] minimal }  
11         S ← S ∪ {i}  
12         Q ← Q \ {i}  
13  
14         # mettre à jour les enfants de i  
15         pour tout noeud j tel que (i,j) ∈ E faire  
16             si Temps[j] > Temps[i] + Poids(i,j) alors  
17                 Temps[j] ← Temps[i] + Poids(i,j)  
18                 Parent[j] ← i  
19         finsi  
20     fait  
21 fait
```

Complexité pire cas : $\Theta((|E| + |V|) \log(|V|))$ si Q bien implémenté

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO!

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO!
- ▶ Algorithmes de graphes (plus court chemin...)

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO !
- ▶ Algorithmes de graphes (plus court chemin...)
- ▶ Programmation concurrente : introduction des sémaphores en 1962

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO !
- ▶ Algorithmes de graphes (plus court chemin...)
- ▶ Programmation concurrente : introduction des sémaphores en 1962
- ▶ Algorithmique distribuée, compilateurs, vérification de programmes, *etc.*

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO !
- ▶ Algorithmes de graphes (plus court chemin...)
- ▶ Programmation concurrente : introduction des sémaphores en 1962
- ▶ Algorithmique distribuée, compilateurs, vérification de programmes, *etc.*
- ▶ **Petites phrases** :
 - ▶ Tester un programme démontre la présence de bugs, pas leur absence.

- ▶ **Historique** : Edsger Wybe Dijkstra (1930–2002) :
Chercheur hollandais en informatique
Une des personnes les plus influentes en info



Structured Programming (1972)
Academic Press



- ▶ Programmation structurée : if/then/else, boucles mais pas de GOTO !
- ▶ Algorithmes de graphes (plus court chemin. . .)
- ▶ Programmation concurrente : introduction des sémaphores en 1962
- ▶ Algorithmique distribuée, compilateurs, vérification de programmes, *etc.*
- ▶ **Petites phrases** :
 - ▶ Tester un programme démontre la présence de bugs, pas leur absence.
 - ▶ La question de savoir si les machines peuvent penser. . . est à peu près aussi pertinente que celle de savoir si les sous-marins peuvent nager.