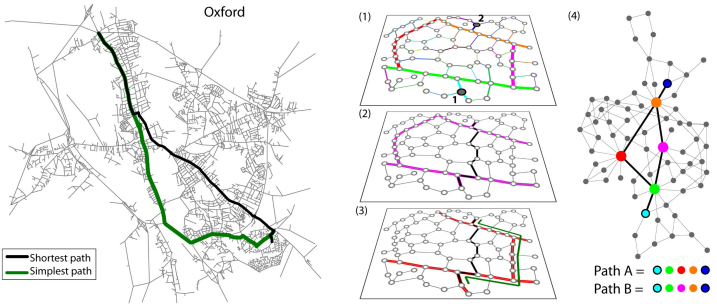
RO03 : Recherche opérationnelle

Problèmes de cheminement

Antoine Jouglet, David Savourey



Introduction

Exemples d'applications

- Rechercher un meilleur itinéraire (le plus court, le plus rapide, le moins polluant, le plus bucolique, *etc.*) ;
- Trouver le meilleur moment pour revendre sa voiture en fonction du prix de revente et des frais d'entretien ;
- Optimiser un portefeuille de placements financiers ;
- Calculer les meilleures actions possibles à un jeu solitaire déterministe.

Problème central

Trouver un plus court chemin dans un graphe est une brique de base dans de nombreux domaines!

- Optimisation combinatoire
- Théorie des jeux
- Informatique
- Économie
- *etc.*

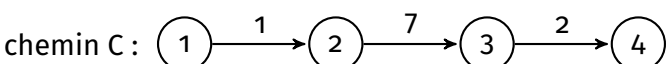
Longueur et valeur d'un chemin

Définition

On appelle **longueur d'un chemin** le nombre d'arcs qui le composent.

Définition

On appelle **valeur d'un chemin** la somme des valuations des arcs du chemin.



$l(C) = 3$ et $v(C) = 10$

Chemins minimaux

Définition

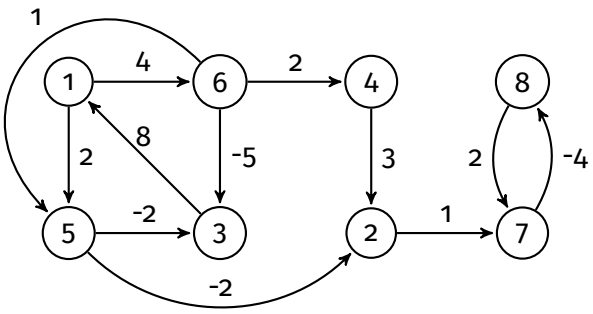
On dit indifférement d'un chemin entre i et j dont la valeur est inférieure ou égale à celle de tout autre chemin entre i et j qu'il est **un chemin minimal** ou bien **un chemin de valeur minimale** ou bien **un plus court chemin** entre i et j.

Problèmes de cheminement

On rencontre principalement 3 problèmes :

- Trouver un plus court chemin entre i et j
- **Trouver un plus court chemin entre i et tous les sommets du graphe**
- Trouver un plus court chemin entre tous les couples de sommets du graphe

Exemple illustré



- Trouver un chemin minimal entre les sommets 1 et 2.
- Existe-t-il un chemin minimal entre 1 et 7 ?

Résultats théoriques

Conventions, notations

- On notera **pcc** pour plus court chemin ;
- On notera **pcc(i)** un plus court chemin entre 1 et i.
- On notera **pcc(i,j)** un plus court chemin entre i et j.

- On notera λ_i^* la valeur d'un pcc(i).
- On notera λ_i^{*k} la valeur d'un pcc(i) utilisant au plus k arcs.

Remarque

On a $\lambda_i^* = \lambda_i^{*n-1}$.

Circuit absorbant

On se situe dans un graphe orienté où les valuations des arcs peuvent être positives, nulles ou négatives.

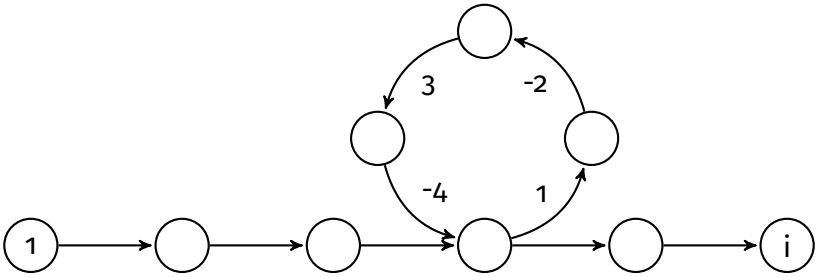
Définition

On appelle **circuit absorbant** un circuit dont la valeur est strictement négative.

Théorème

Soit G un graphe orienté quelconque. **L'existence de plus courts chemins est équivalente à l'existence de chemins** entre toutes paires de sommets de G si et seulement si G **ne contient pas de circuit absorbant**.

Circuit absorbant



plus on emprunte le circuit, plus le chemin diminue de valeur!

Preuve du théorème

$\Rightarrow$ Si G contient un circuit absorbant passant i et j, alors tout chemin entre et i et j peut être « strictement amélioré » en empruntant le circuit. Donc un plus court chemin entre i et j ne peut pas exister.

$\Leftarrow$ Si G ne contient pas de circuit absorbant, alors tout chemin non élémentaire « est dominé » par un chemin élémentaire. Soient i et j deux sommets tels qu'il existe un chemin C entre i et j. Alors il existe un chemin élémentaire entre i et j de valeur plus petite que celle de C. Pour trouver un plus court chemin, on peut donc se restreindre aux chemins élémentaires. Comme ils sont en nombre fini, on peut en calculer le minimum. Ceci garantie bien l'existence d'un plus court chemin.

Propriété caractéristique des chemins minimaux

Théorème

Tout sous-chemin d'un plus court chemin est un plus court chemin.

Preuve (par l'absurde)
Soit C un pcc(i,j). On note $C_{[kl]}$ la portion de C reliant le sommet k au sommet l. Supposons que $C_{[kl]}$ ne soit pas un pcc(k,l). Alors il existe un chemin μ entre k et l avec $v(\mu) < v(C_{[kl]})$. On considère le chemin C' obtenu en remplaçant dans C $C_{[kl]}$ par μ. On voit que $v(C') = v(C) - v(C_{[kl]}) + v(\mu)$. Donc $v(C') < v(C)$. Donc C n'est pas un pcc(i,j), car il est dominé par C'.

Valeurs des plus courts chemins

Théorème

Soit $G = (V, E)$ un graphe sans circuit absorbant. On se donne des λ_i qui correspondent à des valeurs de chemin entre 1 et i. Les valeurs $\{\lambda_i, i \in [1, n]\}$ sont **les valeurs des plus courts chemins** issus de 1 si et seulement si :

- $\lambda_1 = 0$;
- $\forall (i, j) \in E, \lambda_j \leqslant \lambda_i + v_{ij}$.

Preuve du théorème

$\Longrightarrow$ Supposons que les λ_i sont les valeurs des pcc, *i.e.* $\lambda_i = \lambda_i^*$. On a tout d'abord $\lambda_1 = 0$. Par ailleurs, soit (i, j) un arc de E. Soit C un pcc(j). Il est de valeur λ_j. Si C passe par i, alors on sait que $C_{[1i]}$ est un pcc(i), donc de valeur λ_i. Finalement, on obtient $\lambda_j = \lambda_i + v_{ij}$. Si C ne passe pas par i, alors le chemin obtenu en concaténant un pcc(i) et l'arc (i, j) est dominé par C, *i.e.* $\lambda_j \leqslant \lambda_i + v_{ij}$. Dans tous les cas, on a donc bien $\forall i, j) \in E, \lambda_j \leqslant \lambda_i + v_{ij}$.

Preuve du théorème (suite)

$\Leftarrow$ Supposons que $\lambda_1 = 0$ et $\forall (i,j) \in E, \lambda_j \leqslant \lambda_i + v_{ij}$. Supposons qu'il existe k tel que λ_k ne soit pas la valeur d'un pcc(k). Soit v l'indice du plus petit λ_k dont la valeur n'est pas celle d'un pcc(k). Soit C un pcc(v) . On a donc $\lambda_v > v(C)$. On note (u,v) le dernier arc de C. On sait que $C_{[1u]}$ est un pcc(u) et que sa valeur est égale à λ_u, car $\lambda_u < \lambda_v$ et on a supposé que λ_v était le plus petit λ_k qui n'était pas une valeur de pcc. Ainsi, on a $v(C) = \lambda_u + v_{uv}$. Finalement, on obtient $\lambda_v > \lambda_u + v_{uv}$, ce qui est impossible car on a supposé que $\forall (i,j) \in E, \lambda_j \leqslant \lambda_i + v_{ij}$.

Corollaire

Corollaire

L'ensemble des arcs (i,j) pour lesquels $\lambda_j = \lambda_i + v_{ij}$ est l'ensemble des arcs appartenant à des chemins minimaux.

$\Leftarrow$ Soit un arc (i,j) tel que $\lambda_j = \lambda_i + v_{ij}$. Alors la concaténation du chemin de valeur λ_i menant à i et de l'arc (i,j) est un chemin de valeur λ_j. Or $\lambda_j = \lambda_j^*$, donc il existe un pcc(j) passant par l'arc (i,j).

$\Rightarrow$ Soit un arc (i,j) par lequel passe un pcc(k), que l'on note C. Alors $C_{[1i]}$ est un pcc(i) et $C_{[1j]}$ est un pcc(j). Donc $\lambda_j = \lambda_i + v_{ij}$.

Algorithmes

3 algorithmes

Trouver un plus court chemin entre i et tous les sommets du graphe

Algorithme de Ford :

- valuations quelconques
- $O(nm)$
- corrections d'étiquettes

Algorithme de Dijsktra :

- valuations positives ou nulles
- $O(mlog(n))$ ou $O(n^2)$
- fixation d'étiquettes

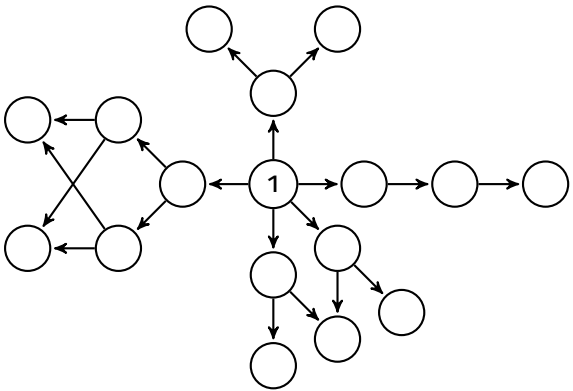
Algorithme de Bellman :

- sans circuit
- $O(m)$
- fixation d'étiquettes

Principe des 3 algorithmes

Les 3 algorithmes ont des **contextes d'utilisation différents**, mais ils reposent sur le même principe :

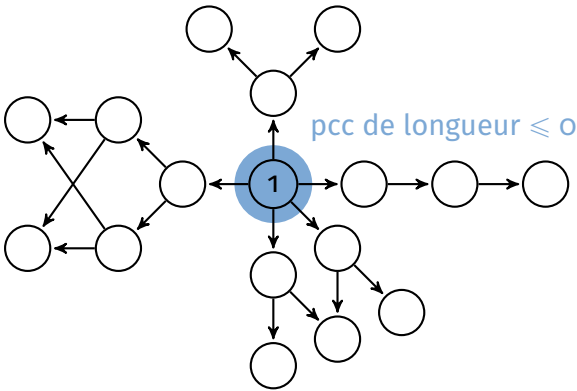
on découvre les plus courts chemins selon les longueurs croissantes



Principe des 3 algorithmes

Les 3 algorithmes ont des **contextes d'utilisation différents**, mais ils reposent sur le même principe :

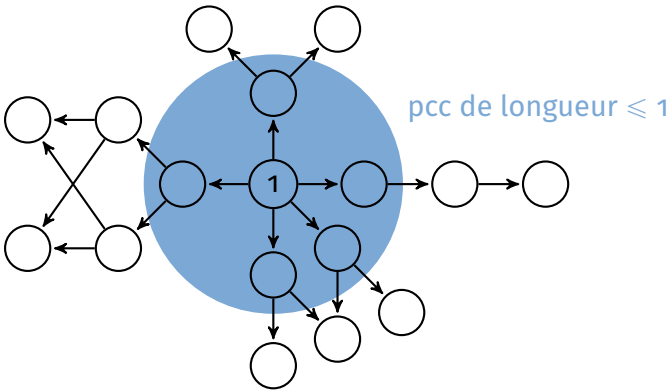
on découvre les plus courts chemins selon les longueurs croissantes



Principe des 3 algorithmes

Les 3 algorithmes ont des **contextes d'utilisation différents**, mais ils reposent sur le même principe :

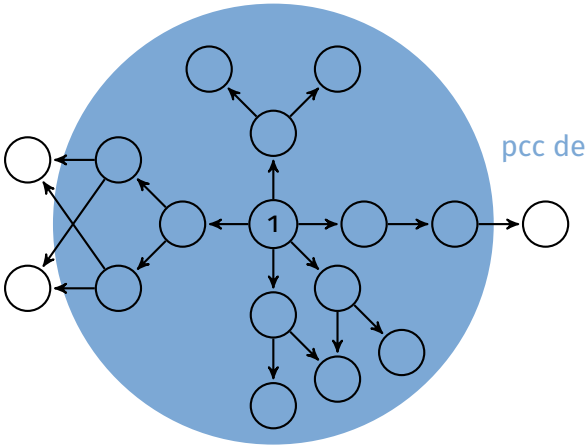
on découvre les plus courts chemins selon les longueurs croissantes



Principe des 3 algorithmes

Les 3 algorithmes ont des **contextes d'utilisation différents**, mais ils reposent sur le même principe :

on découvre les plus courts chemins selon les longueurs croissantes



Algorithme de Ford : préambule

Contexte d'utilisation :
On se situe dans un graphe où les valuations sont quelconques. ou nulles. Ce graphe peut contenir des **arcs avec des valuations strictement négatives** (sinon il vaut mieux utiliser Dijkstra) et/ou **peut contenir des circuits** (sinon il vaut mieux utiliser Bellman).

Variables dans l'algo :

- $\lambda[i]$ est la valeur du meilleur chemin trouvé par l'algorithme
- $p[i]$ est le prédecesseur de i dans le meilleur chemin trouvé par l'algorithme

Algorithme de Ford

Algorithme : Algorithme de Ford

Entrée : un graphe orienté $G(V, E)$ et des valuations v quelconques

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$

pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;

répéter

  $modif \leftarrow FAUX$

  pour i **de 1 à** n **faire**

    si $p[i] \neq -1$ **alors**

      pour chaque successeur j **de** i **faire**

        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**

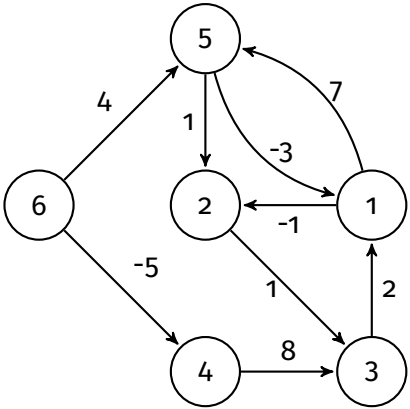
          $p[j] \leftarrow i$;

          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;

          $modif \leftarrow VRAI$

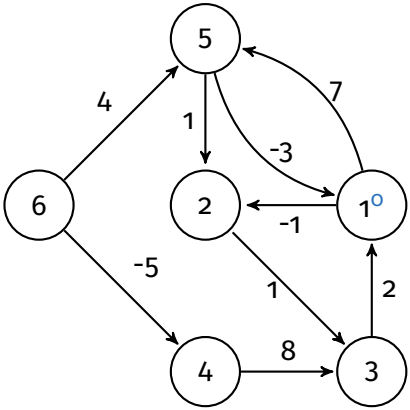
jusqu'à $modif = FAUX$;

Algorithme de Ford : exemple



$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

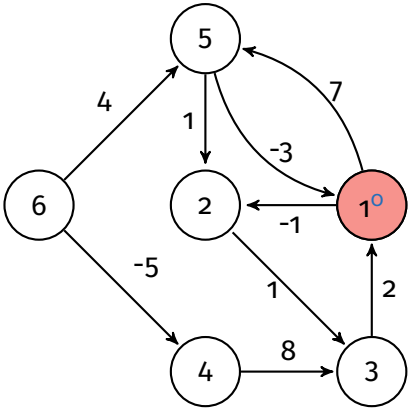
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

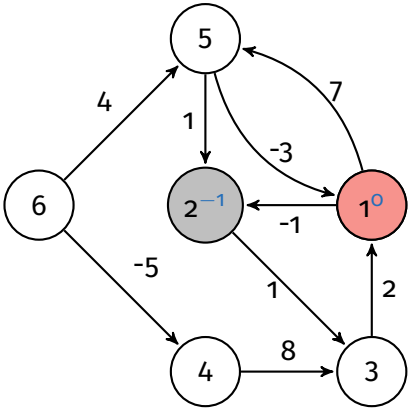
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

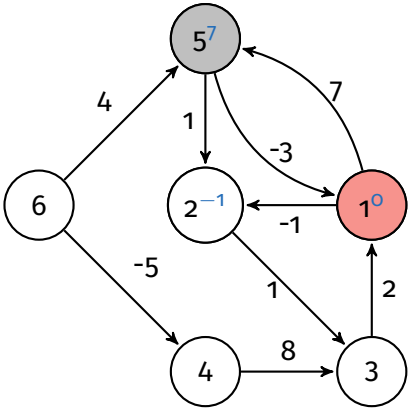
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

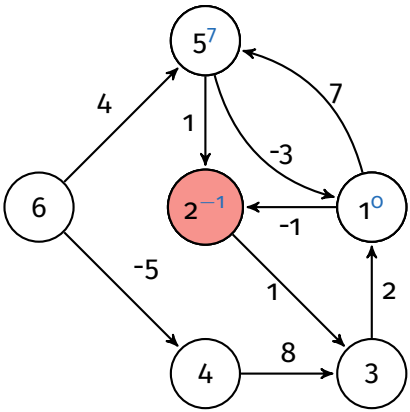
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

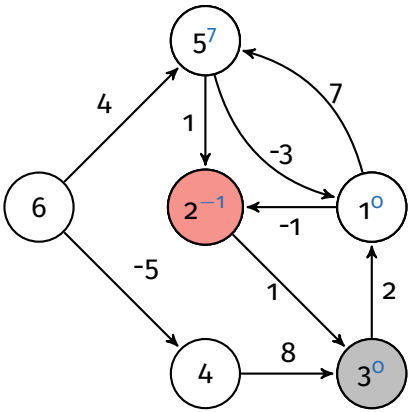
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

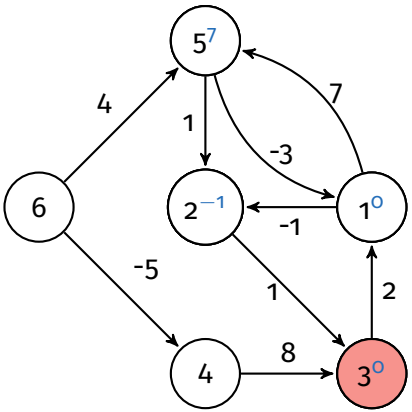
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

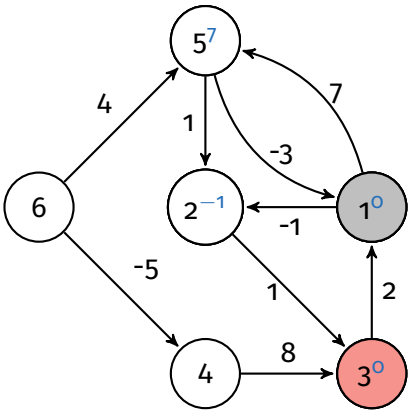
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

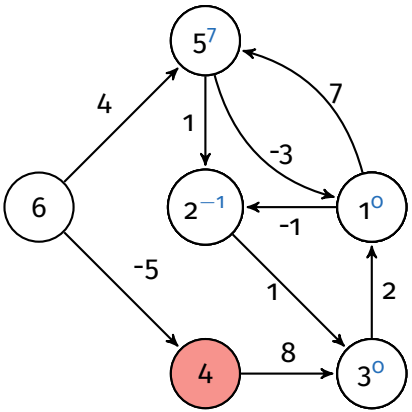
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

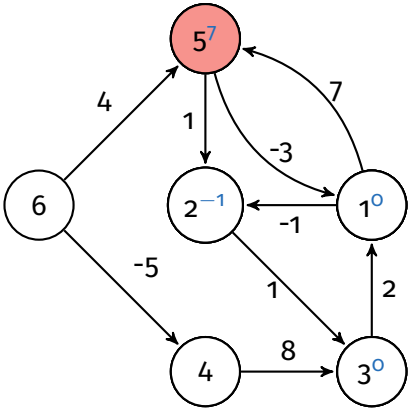
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

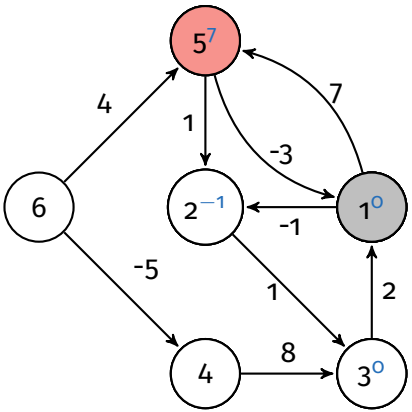
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

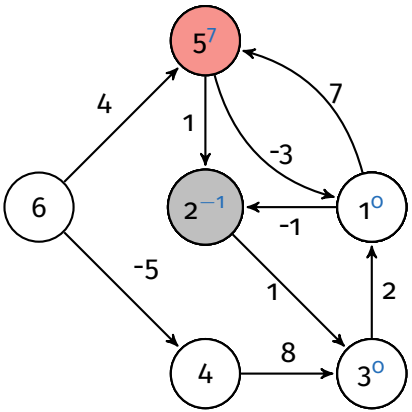
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

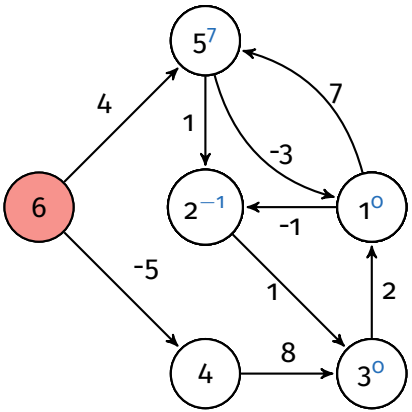
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

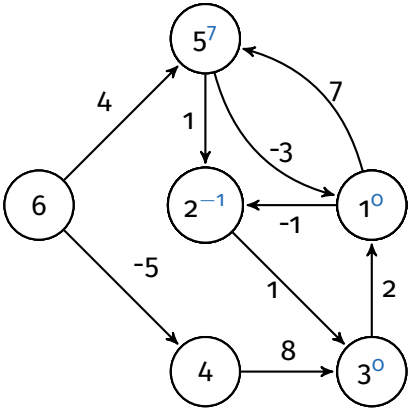
Algorithme de Ford : exemple



modif = *VRAI*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

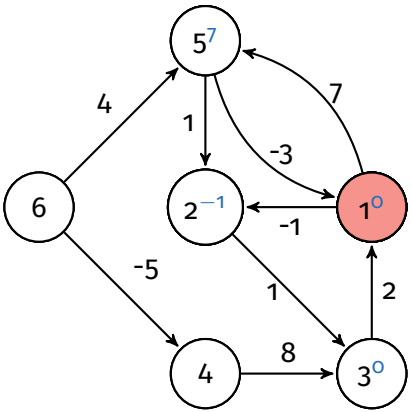
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

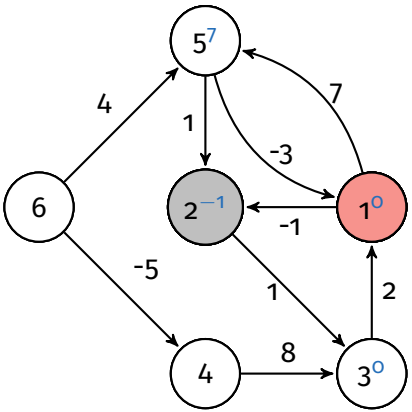
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

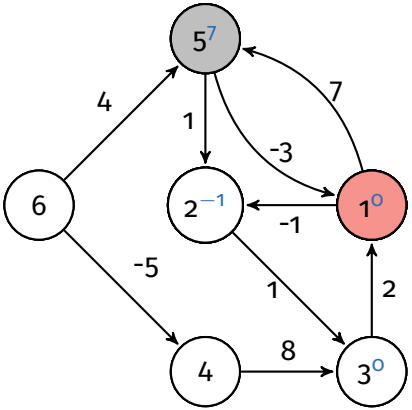
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

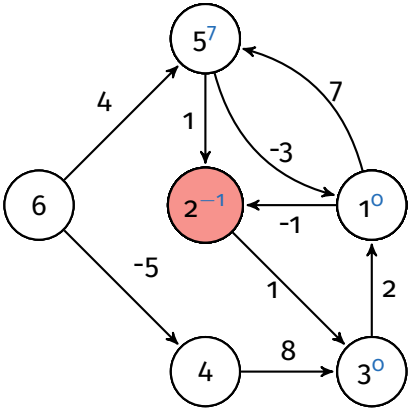
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour *i* **de 2 à** *n* **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour *i* **de 1 à** *n* **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur *j* **de** *i* **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

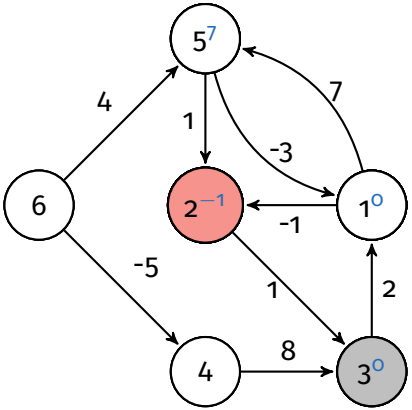
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

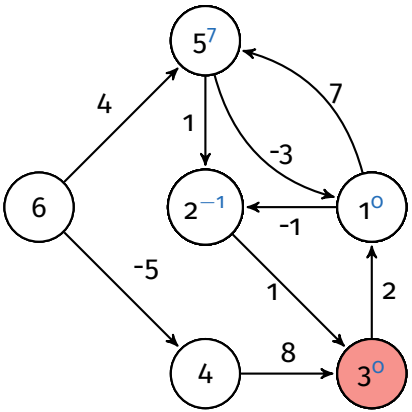
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

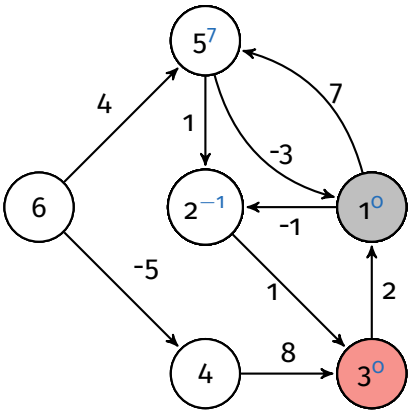
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

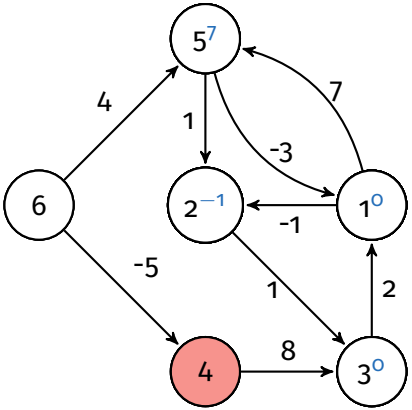
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

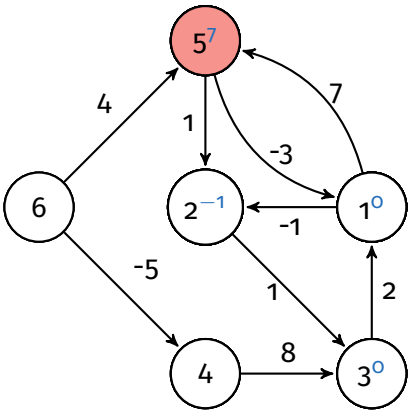
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

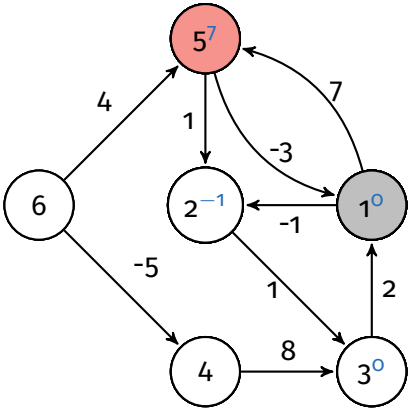
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  modif $\leftarrow$ *FAUX*
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          modif $\leftarrow$ *VRAI*
jusqu'à *modif* = *FAUX*;

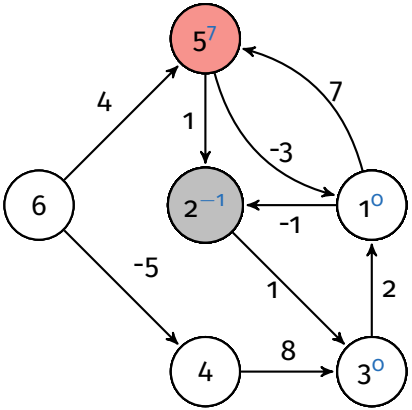
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

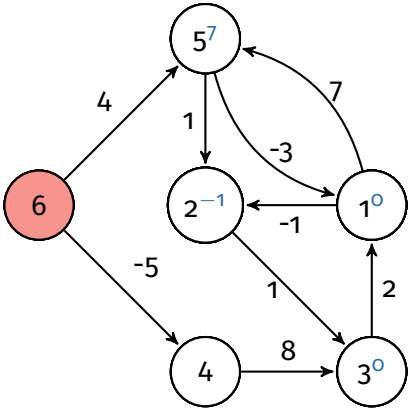
Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

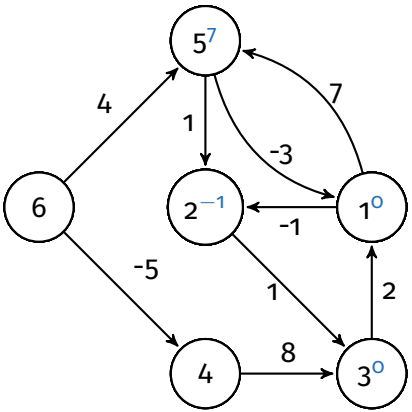
Algorithme de Ford : exemple



modif = *FAUX*

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

Algorithme de Ford : exemple



$modif = FAUX$

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + v_{ij}$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$;

Algorithme de Ford : validité

Théorème

Si le graphe est sans circuit absorbant, l'algorithme calcule les valeurs des plus courts chemins.

Algorithme de Ford : preuve

Invariant

À la fin de l'itération k, si $p[i] \neq -1$, alors $\lambda[i] = \lambda_i^{*k}$.

Preuve (par récurrence)

Supposons qu'à la fin de l'itération k, on ait $\forall i, \lambda[i] = \lambda_i^{*k}$ quand $p[i] \neq -1$. On note $\lambda[i]^k$ la valeur de $\lambda[i]$ à la fin de l'itération k. Plaçons nous désormais à la fin de l'itération $k+1$.

Nous allons examiner deux cas :

a. : $\lambda_i^{*k} > \lambda_i^{*k+1}$

b. : $\lambda_i^{*k} = \lambda_i^{*k+1}$

Algorithme de Ford : preuve (suite)

Preuve du cas a.

Dans le cas a., on appelle C le pcc à $k+1$ arcs, qui doit être strictement plus petit que tous les chemins à k arcs au plus. Par définition, $v(C) = \lambda_i^{*k+1}$. Le sous-chemin formé des k premiers arcs de C, noté C', doit être lui-même un pcc à k arcs, sinon C serait améliorable. On appelle u le dernier sommet de C'. On a alors $v(C) = v(C') + v_{ui}$. Or $v(C) = \lambda_i^{*k+1}$ et $v(C') = \lambda_u^{*k}$. Lors de la $k+1^e$ itération, lorsque l'on va étudier l'arc (u, i), on va fixer $\lambda[i]^{k+1} = \lambda[u]^k + v_{ui}$. Donc on aura $\lambda[i]^{k+1} = v(C') + v_{ui}$, soit $\lambda[i]^{k+1} = \lambda_i^{*k+1}$.

Algorithme de Ford : preuve (suite et fin)

Preuve du cas b.

Dans le cas b., le pcc à $k+1$ arcs au plus a strictement moins de $k+1$ arcs. On a $\lambda[i]^k = \lambda_i^{*k}$ par hypothèse de récurrence. On sait également qu'aucun arc (u,i) ne peut améliorer la valeur de $\lambda[i]^{k+1}$ (c'est ce que dit le cas b.). Donc $\lambda[i]^{k+1} = \lambda[i]^k = \lambda_i^{*k} = \lambda_i^{*k+1}$.

Algorithme de Ford : complexité

- **boucle pour du début** : $O(n)$
- **nb itérations boucle répeter** : au plus $n-1$ itérations (il existe toujours un plus court chemin élémentaire)
- **une itération de la boucle répeter** : pour chaque sommet, on regarde ses successeurs : $\sum_i d^+(i) = m$ donc $O(m)$.
- **complexité de l'algo** : $O(nm)$

```
λ[1] ← 0 ; p[1] ← 1
pour i de 2 à n faire λ[i] ← ∞ ;
  p[i] ← −1 ;
répéter
    modif ← FAUX
    pour i de 1 à n faire
        si p[i] ≠ −1 alors
            pour chaque successeur j
              de i faire
                si λ[j] > λ[i] + v_ij alors
                    p[j] ← i ;
                    λ[j] ← λ[i] + vij ;
                    modif ← VRAI
jusqu'à modif = FAUX ;
```

Algorithme de Ford modifié

L'algorithme de Ford ne termine en cas de circuit absorbant. On peut empêcher cela en limitant le nombre d'itérations à $n - 1$.

$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 1$; $iter \leftarrow 0$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$; $p[i] \leftarrow -1$;
répéter
  $modif \leftarrow FAUX$; $iter \leftarrow iter + 1$
  pour i **de 1 à** n **faire**
    si $p[i] \neq -1$ **alors**
      pour chaque successeur j **de** i **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
          $p[j] \leftarrow i$;
          $\lambda[j] \leftarrow \lambda[i] + vij$;
          $modif \leftarrow VRAI$
jusqu'à $modif = FAUX$ **et** $iter < n$;
si $iter = n$ **alors** $exit$("circ. absorb.");

Algorithme de Dijsktra : préambule

Contexte d'utilisation :
On se situe dans un graphe où les **valuations sont positives ou nulles**. Ce graphe **peut contenir des circuits** (sinon il vaut mieux utiliser l'algorithme de Bellman).

Variables dans l'algo :

- $\lambda[i]$ est la valeur du meilleur chemin trouvé par l'algorithme
- *atteints* est une file de priorité contenant les sommets pour lesquels on a trouvé un chemin
- *fixe*$[i]$ est vrai ssi on a trouvé un pcc(i) (donc $\lambda[i] = \lambda_i^*$)

File de priorité

Cet algorithme utilise **une file de priorité** : c'est une structure de données qui permet de stocker une collection d'éléments avec une priorité associée. On peut effectuer les opérations suivantes sur une file de priorité :

- savoir si elle est vide : $\mathit{empty}()$
- ajouter un élément avec une certaine priorité : $\mathit{add}(x, p)$
- retirer l'élément de plus grande/petite priorité : $\mathit{extractMinPriority}()$

Dans notre cas, les éléments seront les sommets du graphe ; les priorités seront les $\lambda[i]$ associés et on retirera l'élément de plus petite priorité.

Algorithme de Dijsktra

Algorithme : Algorithme de Dijsktra

Entrée : un graphe orienté $G(V, E)$ et des valuations v positives ou nulles

$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$

pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;

tant que *atteints.empty*$() = FALSE$ **faire**

    $i \leftarrow$ *atteints.extractMinPriority*$()$

    si *fixe*$[i] = FAUX$ **alors**

        fixe$[i] \leftarrow VRAI$

        pour chaque successeur j **de** i **faire**

            $d \leftarrow \lambda[i] + v_{ij}$

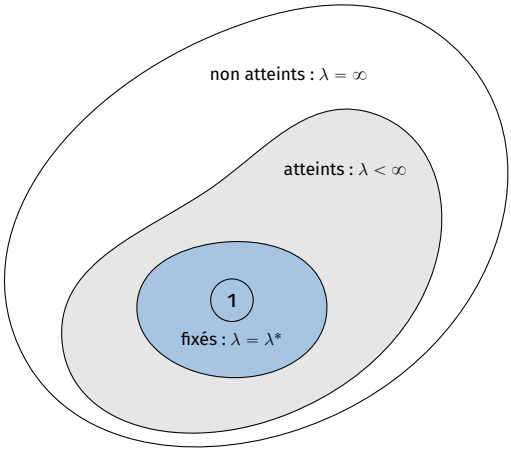
            si $d < \lambda[j]$ **alors**

                $\lambda[j] \leftarrow d$

                atteints.add(j, d)

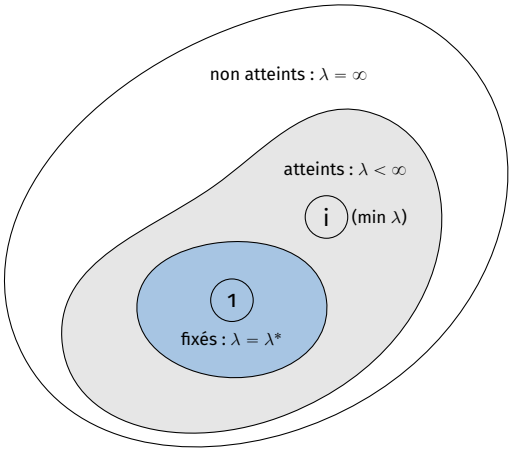
Algorithme de Dijsktra : comment ça marche ?

- **fixés** : on a déja trouvé un pcc
- **atteints** : on a déja trouvé un chemin
- **non atteints** : on a pas trouvé de chemin



Algorithme de Dijsktra : comment ça marche ?

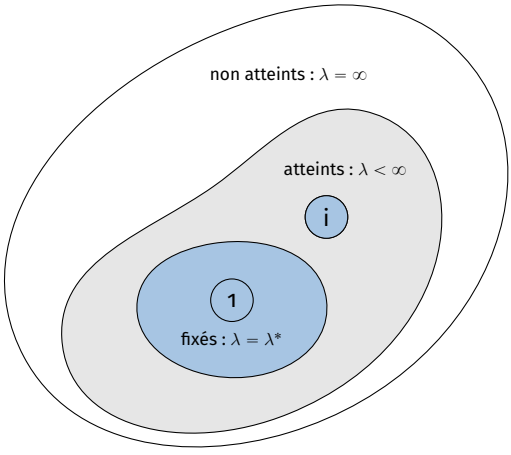
on choisit le sommet *i* déjà atteint de plus faible distance à 1.



Algorithme de Dijsktra : comment ça marche ?

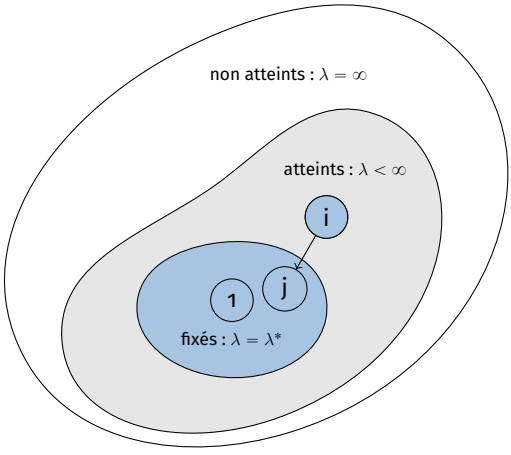
on ne pourra plus améliorer ce sommet.
donc on le fixe.

on va examiner les voisins de *i*.



Algorithme de Dijsktra : comment ça marche ?

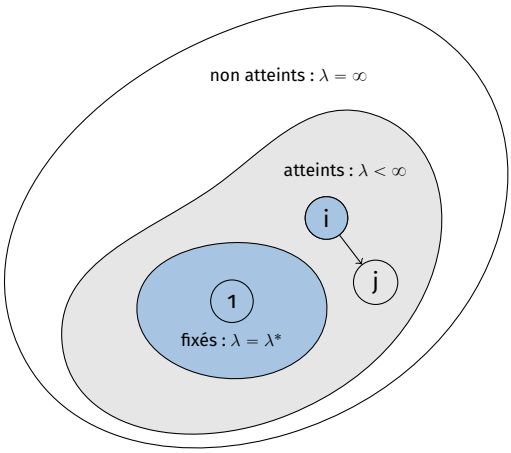
si *j* est déjà fixé, le chemin passant par *i* ne peut pas amélioré *j*, donc on l'ignore.



Algorithme de Dijsktra : comment ça marche ?

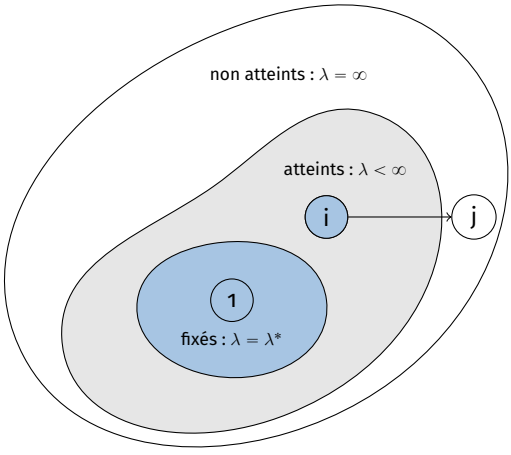
si j est atteint, 2 cas sont possibles :

- **le chemin par i améliore λ_j** : on ajoute/met à jour j avec $\lambda_j = \lambda_i + v_{ij}$;
- **le chemin par i n'améliore pas λ_j** : on l'ignore.

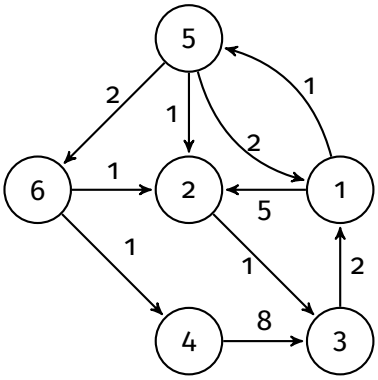


Algorithme de Dijsktra : comment ça marche ?

si j n'est pas encore atteint, on l'ajoute avec $\lambda_j = \lambda_i + v_{ij}$

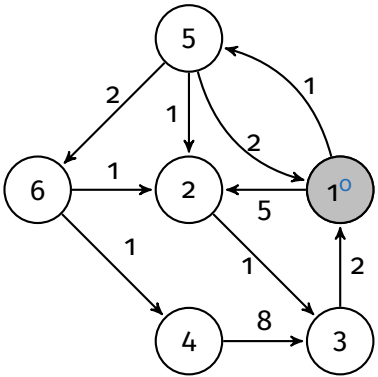


Algorithme de Dijkstra : exemple



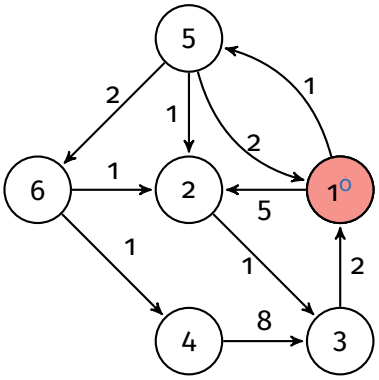
$\lambda[1] \leftarrow 0$; $atteints.add(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que $atteints.empty() = FALSE$ **faire**
  $i \leftarrow atteints.extractMinPriority()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        $atteints.add(j, d)$

Algorithme de Dijkstra : exemple



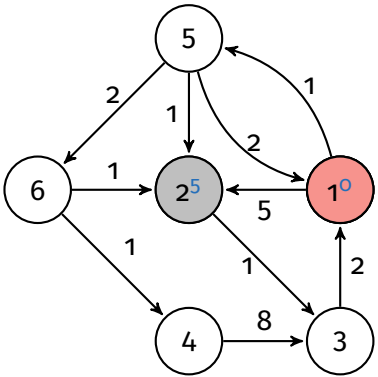
$\lambda[1] \leftarrow 0$; $atteints.add(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que $atteints.empty() = FALSE$ **faire**
  $i \leftarrow atteints.extractMinPriority()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        $atteints.add(j, d)$

Algorithme de Dijkstra : exemple



$\lambda[1] \leftarrow 0$; $atteints.add(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que $atteints.empty() = FALSE$ **faire**
  $i \leftarrow atteints.extractMinPriority()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        $atteints.add(j, d)$

Algorithme de Dijkstra : exemple

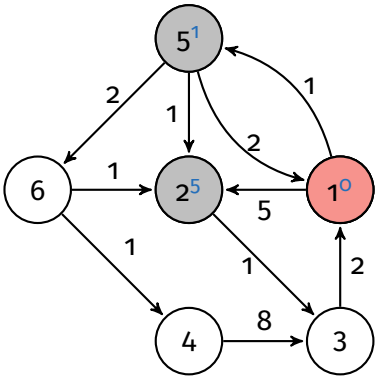


atteints

2^5

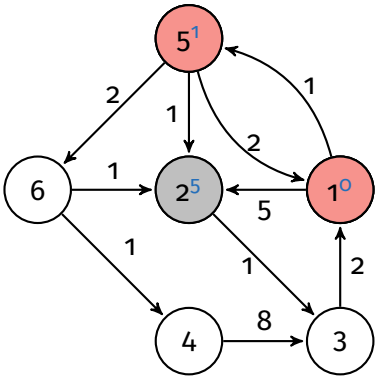
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



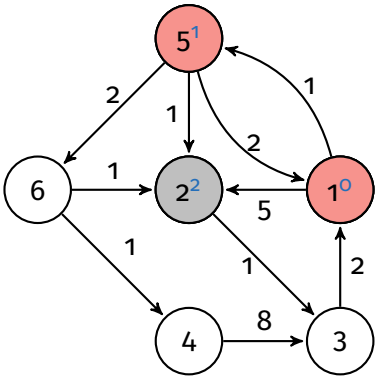
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



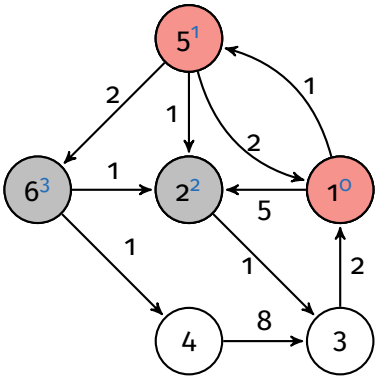
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
    $i \leftarrow$ *atteints.extractMinPriority*$()$
    si $fixe[i] = FAUX$ **alors**
        $fixe[i] \leftarrow VRAI$
        pour chaque successeur j **de** i **faire**
            $d \leftarrow \lambda[i] + v_{ij}$
            si $d < \lambda[j]$ **alors**
                $\lambda[j] \leftarrow d$
                atteints.add(j, d)

Algorithme de Dijkstra : exemple

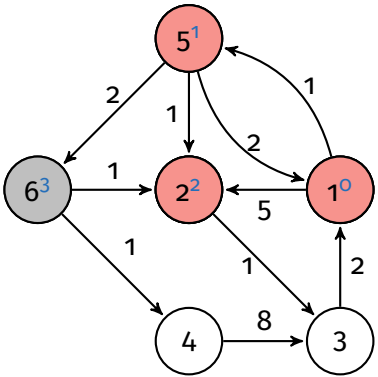


atteints

2^5 2^2 6^3

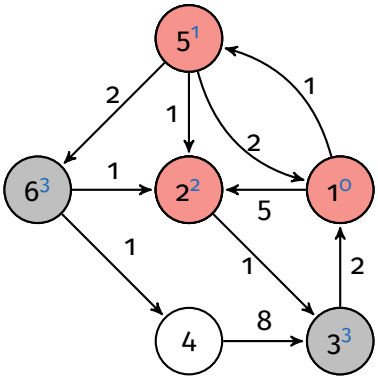
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() =$ *FALSE* **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si *fixe*$[i] =$ *FAUX* **alors**
    fixe$[i] \leftarrow$ *VRAI*
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



$\lambda[1] \leftarrow 0$; $atteints.add(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que $atteints.empty() = FALSE$ **faire**
  $i \leftarrow atteints.extractMinPriority()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        $atteints.add(j, d)$

Algorithme de Dijkstra : exemple

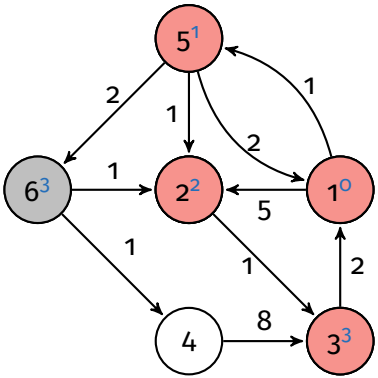


atteints

2^5 6^3 3^3

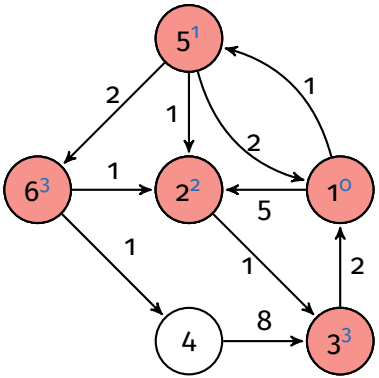
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() =$ *FALSE* **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si *fixe*$[i] =$ *FAUX* **alors**
    fixe$[i] \leftarrow$ *VRAI*
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



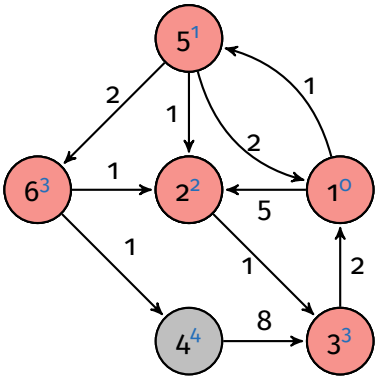
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si *fixe*$[i] = FAUX$ **alors**
    fixe$[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



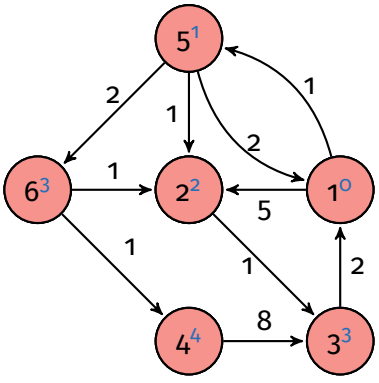
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si *fixe*$[i] = FAUX$ **alors**
    fixe$[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



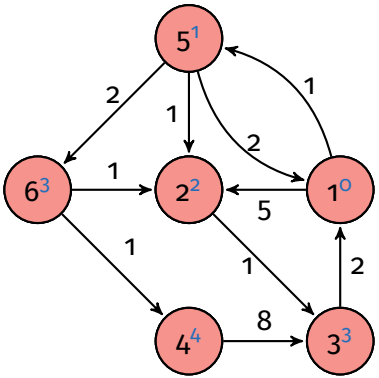
$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si *fixe*$[i] = FAUX$ **alors**
    fixe$[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijkstra : exemple



$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
    $i \leftarrow$ *atteints.extractMinPriority*$()$
    si *fixe*$[i] = FAUX$ **alors**
        fixe$[i] \leftarrow VRAI$
        pour chaque successeur j **de** i **faire**
            $d \leftarrow \lambda[i] + v_{ij}$
            si $d < \lambda[j]$ **alors**
                $\lambda[j] \leftarrow d$
                atteints.add(j, d)

Algorithme de Dijkstra : exemple



$\lambda[1] \leftarrow 0$; *atteints.add*$(1, 0)$
pour i **de 2 à** n **faire** $\lambda[i] \leftarrow \infty$;
tant que *atteints.empty*$() = FALSE$ **faire**
  $i \leftarrow$ *atteints.extractMinPriority*$()$
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        atteints.add(j, d)

Algorithme de Dijsktra : preuve

On va prouver l'invariant suivant :

Invariant

Quand l'algorithme exécute *fixe*$[i] \leftarrow$ *VRAI*, $\lambda[i] = \lambda_i^*$.

Preuve (par l'absurde)
Supposons qu'il existe un chemin de valeur strictement plus petite que $\lambda[i]$ menant à x_i. Ce chemin a nécessairement un arc (v, w) où v est fixé et w atteint. Si ce chemin est meilleur, on a doit avoir $\lambda[w] < \lambda[i]$. Donc w aurait dû sortir avant i de *atteints*.

Algorithme de Dijkstra : complexité

- **boucle pour du début** : $O(n)$
- **nb. itér. tant que** : m au maximum
- **1** *extractMinPriority* : $O(log(n))$
- **tous les** *extractMinPriority* : $O(mlog(n))$
- **1 itér. succ de i** : $O(d^+(i))$
- **toutes les itér. succ de i** : $O(m)$
- **complexité de l'algo** : $O(mlog(n))$

```
λ[1] ← 0 ; atteints.add(1, 0)
pour i de 2 à n faire λ[i] ← ∞;
tant que atteints.empty() = FALSE
 faire
    i ← atteints.extractMinPriority()
    si fixe[i] = FAUX alors
        fixe[i] ← VRAI
        pour chaque successeur j de i
         faire
            d ← λ[i] + v_ij
            si d < λ[j] alors
                λ[j] ← d
                atteints.add(j, d)
```

Algorithme de Bellman : préambule

Contexte d'utilisation :
On se situe dans un **graphe sans circuit**. On sait donc qu'on peut trouver un ordre topologique sur les sommets en temps linéaire.

Variables dans l'algo :

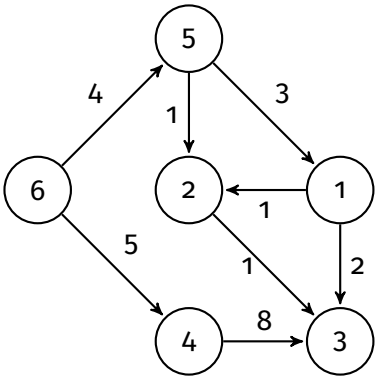
- $\lambda[i]$ est la valeur du meilleur chemin trouvé par l'algorithme
- $p[i]$ est le prédecesseur de i dans le meilleur chemin trouvé par l'algorithme

Algorithme de Bellman

Algorithme : Algorithme de Bellman

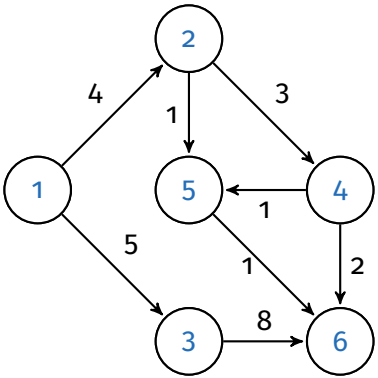
Entrée : un graphe orienté $G(V, E)$ sans circuit
Renuméroter les sommets selon un ordre topologique
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de 2 à** n **faire**
    $\lambda[j] \leftarrow \infty$
    pour chaque prédecesseur i **de** j **faire**
        si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
            $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
            $p[j] \leftarrow i$

Algorithme de Bellman : exemple



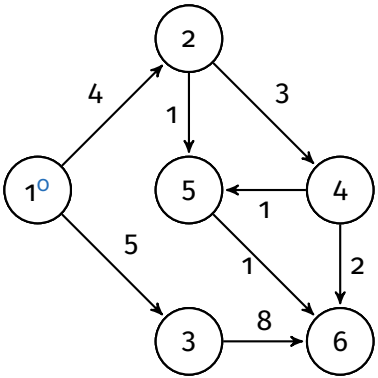
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



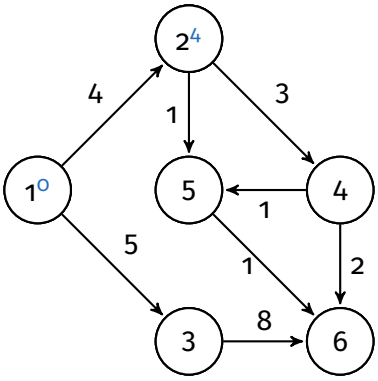
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de 2 à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



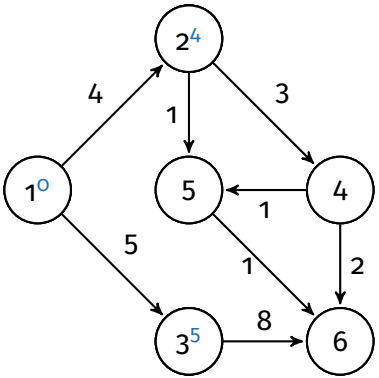
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de 2 à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



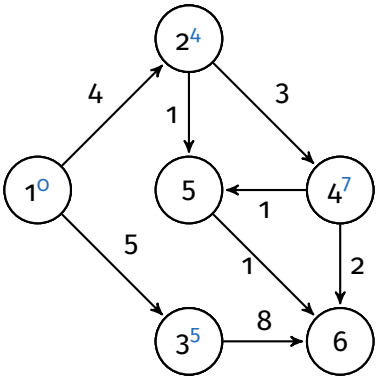
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



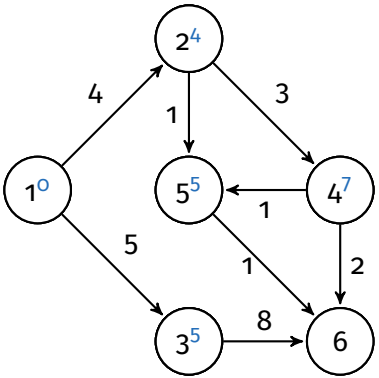
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



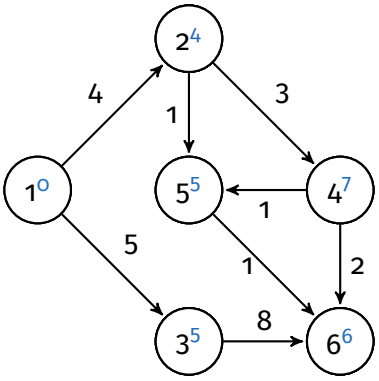
Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : exemple



Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de** 2 **à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme de Bellman : complexité

- **ordre topologique** : $O(n)$
- **nb itérations boucle pour** : n itér.
- **une itération de la boucle** : $O(d^+(i))$
- **complexité de l'algo** : $O(m)$

Renuméroter les sommets topologiquement
$\lambda[1] \leftarrow 0$; $p[1] \leftarrow 0$
pour j **de 2 à** n **faire**
  $\lambda[j] \leftarrow \infty$
  pour chaque prédecesseur i **de** j **faire**
    si $\lambda[j] > \lambda[i] + v_{ij}$ **alors**
      $\lambda[j] \leftarrow \lambda[i] + v_{ij}$
      $p[j] \leftarrow i$

Algorithme *A**

Algorithme A^* : Contexte

Il y a beaucoup de contexte où l'**on connait plus que le graphe des distances**. Par exemple, pour la recherche d'itinéraire optimal, on connait la position absolue des sommets du graphe sur une carte.

On cherche souvent **un pcc entre une source fixée et une destination fixée** (par exemple encore la recherche d'itinéraire optimal).

Algorithme A^* : Principe

On cherche un pcc(1,n)

Fonctionnement similaire à l'algorithme de Dijsktra

Évaluations des sommets différentes, pour un sommet i :

- Dijsktra : la valeur $\lambda[i]$ du meilleur chemin trouvé de 1 à i
- A^* : la valeur $\lambda[i]$ du meilleur chemin trouvé de 1 à i + une évaluation par défaut $def(i)$ de la valeur entre i et n

$$priorite(i) = \lambda[i] + def(i)$$

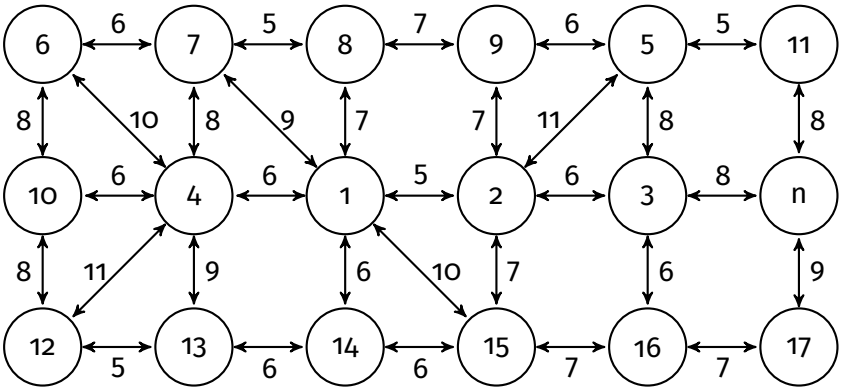
Algorithme A^*

on suppose que l'on cherche un pcc$(1, n)$

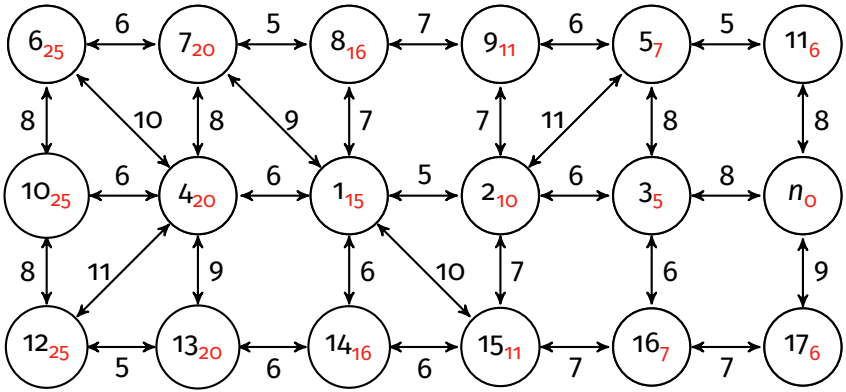
Algorithme : Algorithme A^*

$\lambda[1] \leftarrow 0$; $atteints.add(1, 0 + def(1, n))$
tant que $atteints.empty() = FALSE$ **faire**
  $i \leftarrow atteints.extractMinPriority()$
  si $i = n$ **alors** exit;
  si $fixe[i] = FAUX$ **alors**
    $fixe[i] \leftarrow VRAI$
    pour chaque successeur j **de** i **faire**
      $d \leftarrow \lambda[i] + v_{ij}$
      si $fixe[j] = FAUX$ **et** $d < \lambda[j]$ **alors**
        $\lambda[j] \leftarrow d$
        $atteints.add(j, d + def(j, n))$

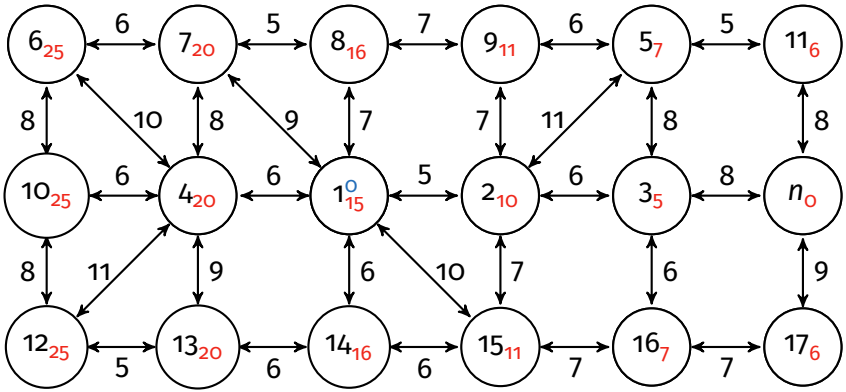
Algorithme A^* : exemple



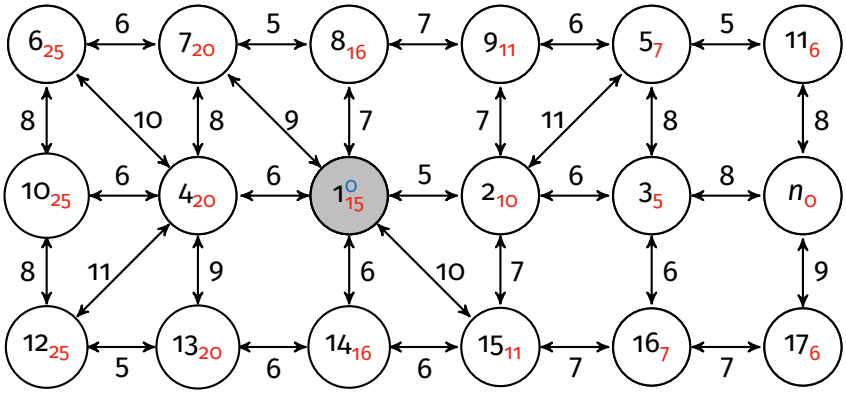
Algorithme A^* : exemple



Algorithme A^* : exemple



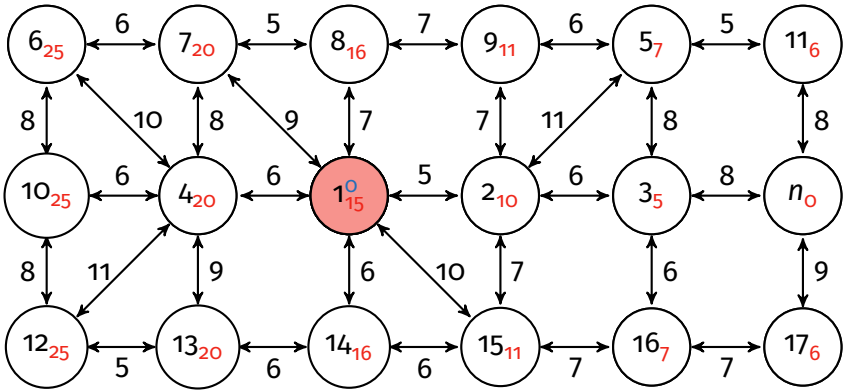
Algorithme A^* : exemple



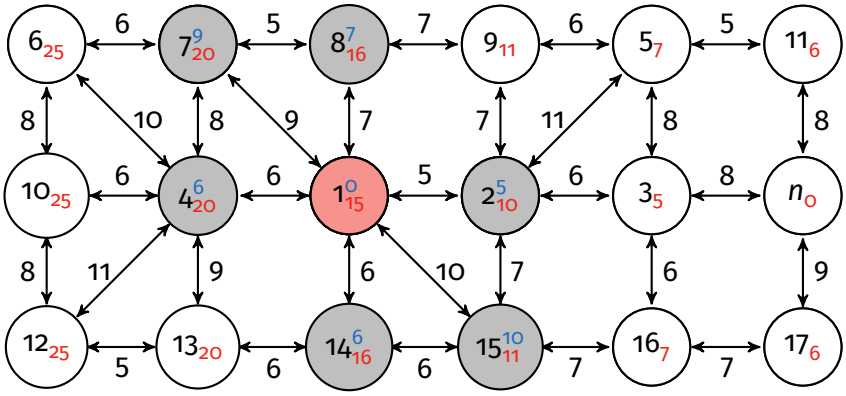
atteints

1^{o}_{15}

Algorithme A^* : exemple



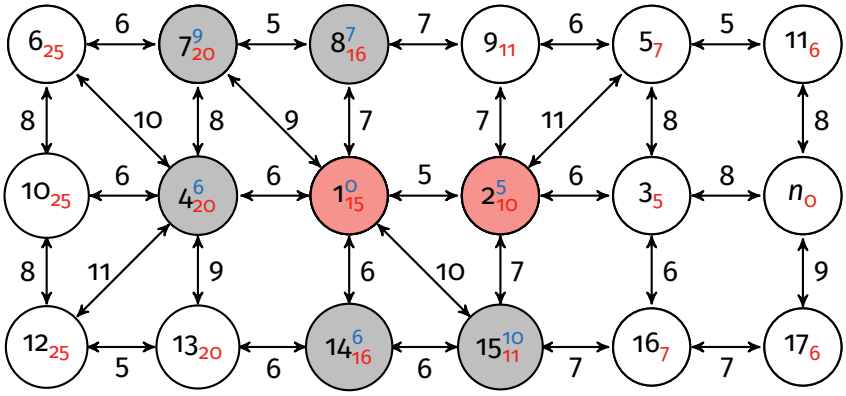
Algorithme A^* : exemple



atteints

2^{5}_{10} 4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11}

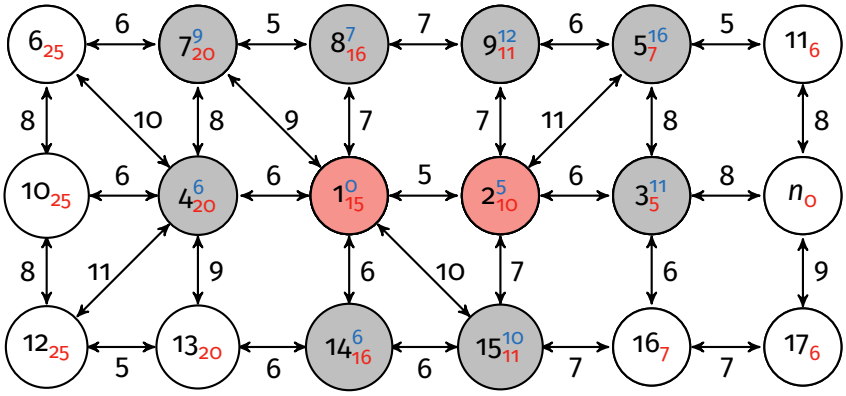
Algorithme A^* : exemple



atteints

4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11}

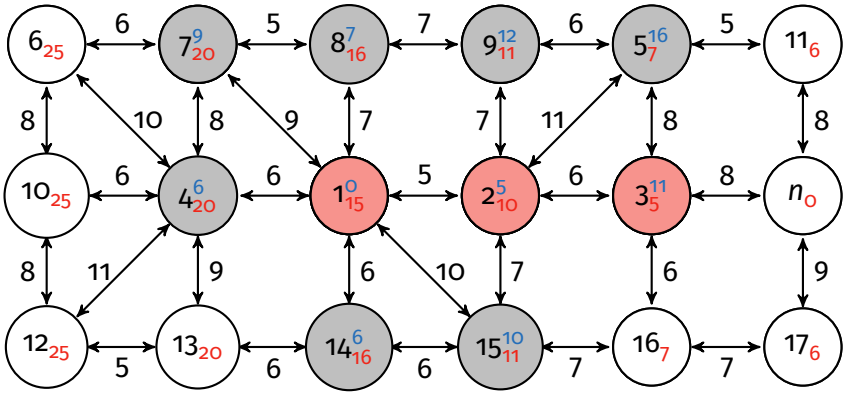
Algorithme A^* : exemple



atteints

4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11} 3^{11}_{5} 9^{12}_{11} 5^{16}_{7}

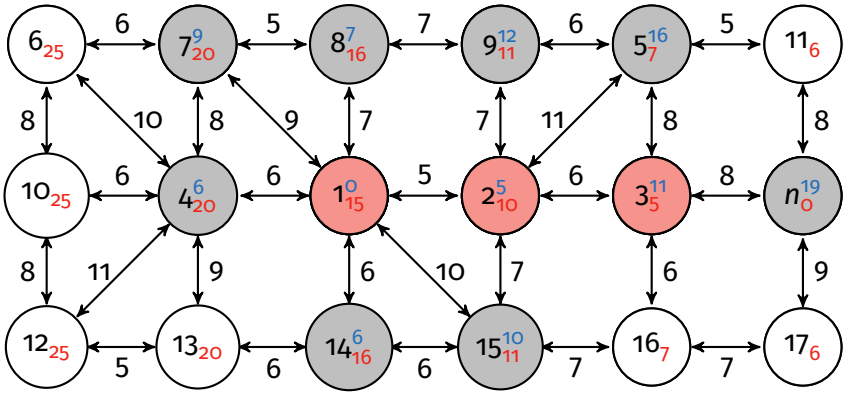
Algorithme A^* : exemple



atteints

4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11} 9^{12}_{11} 5^{16}_{7}

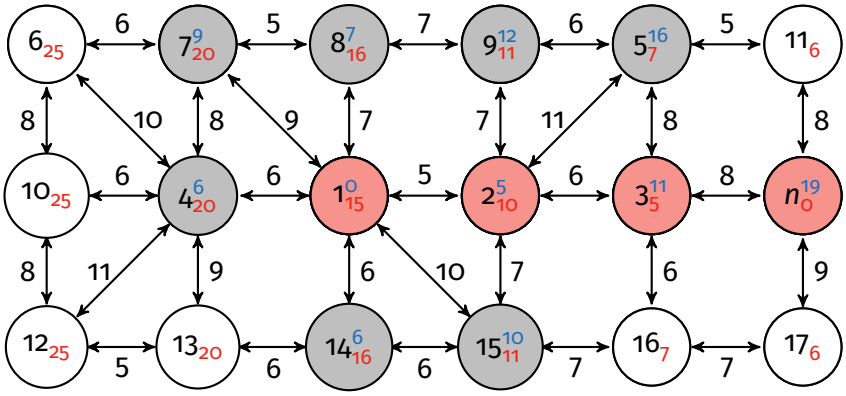
Algorithme A^* : exemple



atteints

4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11} 9^{12}_{11} 5^{16}_{7} n^{19}_{0}

Algorithme A^* : exemple



atteints

4^{6}_{20} 7^{9}_{20} 8^{7}_{16} 14^{6}_{16} 15^{10}_{11} 9^{12}_{11} 5^{16}_{7}