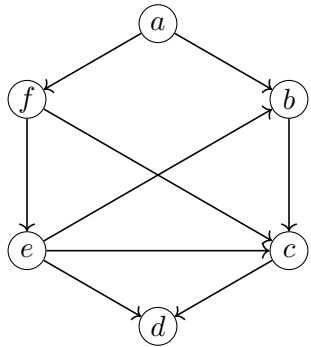
Tri topologique et les DAG (1/3)

Définition

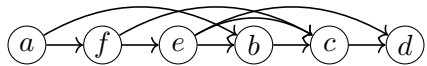
Un *tri topologique* d'un graphe orienté $G = (V, E)$ est un ordre total $\prec$ sur V tel que, pour tout arc $(u, v) \in E$, on a $u \prec v$.



Tri topologique et les DAG (1/3)

Définition

Un *tri topologique* d'un graphe orienté $G = (V, E)$ est un ordre total $\prec$ sur V tel que, pour tout arc $(u, v) \in E$, on a $u \prec v$.



Tri topologique et les DAG (2/3)

Théorème

Un graphe orienté G admet un tri topologique ssi G ne contient pas de circuits.

Démonstration (1/2)

- Si G contient un circuit, G n'admet évidemment pas de tri topologique.
- Inversement, supposons que G est un DAG.
- On définit l'ordre total $\prec$ sur les sommets de G comme suit :
- $u \prec v$ ssi $\text{post}(u) > \text{post}(v)$.

Tri topologique et les DAG (3/3)

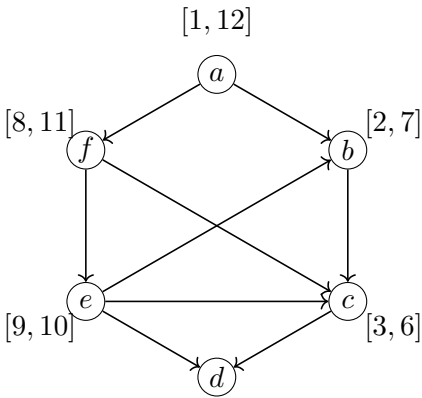
Démonstration (2/2)

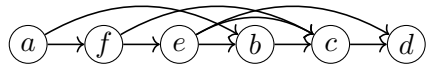
- Soit (u, v) un arc quelconque de G.
- Comme G est un DAG, v n'est pas un ancêtre de u.
- Donc, (u, v) n'est pas un arc retour.
- Par la remarque à la fin de la classification des arcs, $\text{post}(u) > \text{post}(v)$.
- Donc $u \prec v$.

Remarque

Pour trouver un tri topologique d'un DAG G, il suffit de faire un parcours en profondeur, et trier les sommets de G par ordre décroissant de post$(\cdot)$.

Exemple





[1, 12] [8, 11] [9, 10] [2, 7] [3, 6] [4, 5]

Une conséquence du tri topologique

- Le premier sommet dans un tri topologique de G est une source (c'est-à-dire, aucun arc n'entre le sommet).
- De même, le dernier sommet est un puits (c'est-à-dire, aucun arc ne sorte du sommet).

Théorème
Tout graphe orienté acyclique contient au moins une source et au moins un puits.

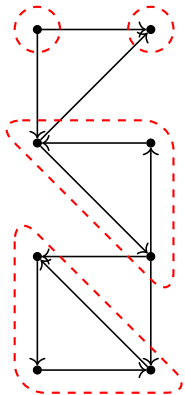
Ce théorème fournit une autre approche au tri topologique :

- Trouver un sommet source de G et supprimer-le de G.
- Répéter jusqu'à ce que le graphe devienne vide.

Connexité dans les graphes orientés

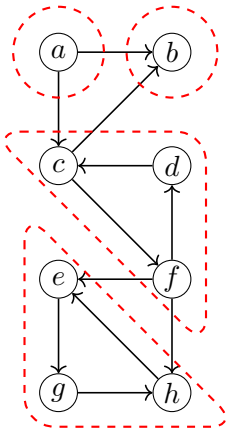
- Nous avons déjà vu la définition de graphes connexes et des composantes connexes.
- Intuitivement, un graphe est connexe s'il ne peut pas être "séparé" sans casser des arêtes.
- Pour les graphes orientés, la notion de connexité est un peu plus compliquée.
- Soit $\sim$ la relation suivante : $u \sim v$ ssi il existe un chemin de u à v et aussi un chemin de v à u.

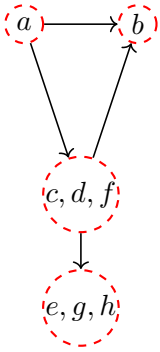
Composantes fortement connexes



- Il est facile de vérifier que $\sim$ est une relation d'équivalence :
  - $\sim$ est symétrique
  - $\sim$ est refléxive
  - $\sim$ est transitive.
- Les classes d'équivalence forment une partition de $V(G)$.
- On les appelle les *composantes fortement connexes*.

Contracter les composantes fortement connexes





Le graphe contracté est un DAG

Théorème

Soit G un graphe orienté quelconque, et soit G' le graphe obtenu en contractant chaque composante fortement connexe à un seul sommet. Alors, G' est un graph orienté acyclique (un DAG).

- Soit $C = (G_1, G_2, \ldots, G_k)$ un circuit dans G'.
- Alors, tous les sommets de G dans la composante fortement connexe G_i sont atteignable depuis tous les sommets de G_j, pour tous $i, j \in \{1, \ldots, k\}$.
- Donc, $V(G_1) \cup V(G_2) \cup \cdots \cup V(G_k)$ appartiennent à une seule composante connexe.
- Donc $k = 1$.

Composantes fortement connexes et parcours en profondeur

Propriété 1
Si la procédure `explorer` est lancée au sommet u, elle se terminera précisément lorsque tous les sommets atteignables depuis u auront été visités.

- Donc, si u est un sommet dans une composante fortement connexe G_i qui est un puit dans le graphe contracté G' (tous les arcs incidents à G_i pointent vers G_i), alors `explore`(u) va parcourir précisément les sommets de G_i.

Composantes fortement connexes et parcours en profondeur

Propriété 2

Soient G_i et G_j des composantes fortement connexes de G. S'il existe un arc d'un sommet de G_i à un sommet de G_j, alors

$$\max\{\text{post}(v) : v \in G_i\} \geq \max\{\text{post}(v) : v \in G_j\}.$$

- Il y a deux cas à considérer.
- Si le DFS visite la composante G_i avant le composante G_j, alors tous les sommets de G_i et G_j seront visités avant de coincer.
- Par conséquent, le nombre post du premier sommet visité dans G_i sera supérieur à celui de n'importe quel sommet de G_j.
- Si G_j est visité en premier, le DFS va coincer après avoir visité l'ensemble de G_j mais avant d'avoir visité l'ensemble de G_i.

Conséquence de la Propriété 2

Propriété 3

Le sommet avec la valeur maximum de post$(\cdot)$ dans une recherche en profondeur appartient à une composante fortement connexe de type source.

- On peut trier les composantes fortement connexes par ordre décroissant de leurs nombres post maximaux.

. . . et si on veut trouver un sommets dans un puit ?

- La Propriété 3 nous aide à trouver un sommet dans une composante fortement connexe de G de type « source ».
- Or, nous avons besoin d'un sommet dans une composante fortement connexe de G de type « puits ».
- Soit G^R le graphe orienté *inverse*, définit comme suit : $G^R = (V, E^R)$, où $(u, v) \in E^R$ ssi $(v, u) \in E$.
- G^R a les même composantes fortement connexes que G.
- En effectuant un parcours en profondeur sur G^R, le sommet avec la valeur maximale de post$(\cdot)$ appartient à une composante fortement connexe de G^R de type « source », c'est-à-dire, à une composante fortement connexe de G de type « puits ».

Vers un algorithme de composantes fortement connexes

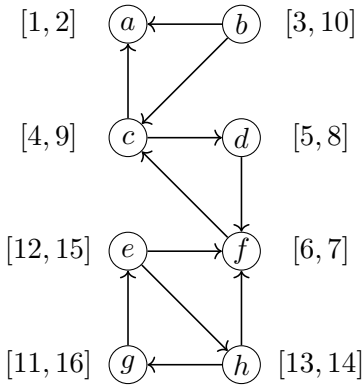
- Comment continuer après l'identification de la première composante fortement connexe de type puits ?
- Il suffit d'utiliser la Propriété 2.
- Une fois que nous avons trouvé la première composante fortement connexe G_1 et que nous l'avons supprimée du graphe, le sommet avec le nombre post maximum dans $G - V(G_1)$ appartiendra à une composante fortement connexe de $G - V(G_1)$.
- Par conséquent, nous pouvons continuer à utiliser les nombres poste$(\cdot)$ du parcours en profondeur initial sur G^R pour produire successivement la deuxième composante fortement connexe, la troisième composante fortement connexe, etc.

Un algorithme de composantes fortement connexes

Voici donc un algorithme pour déterminer les composantes fortement connexes de G :

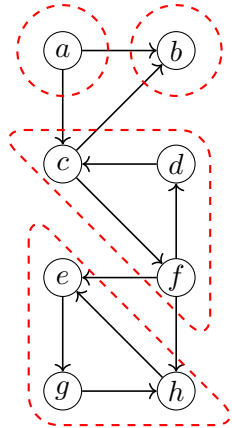
1. Exécuter un parcours en profondeur sur G^R.
2. Exécuter un parcours en profondeur sur sur le graphe non-orienté sous-jacent de G, en traitant les sommets par nombre post (trouvé dans l'étape 1) décroissant.

Exemple



G^R

g, e, h, b, c, d, f, a



G

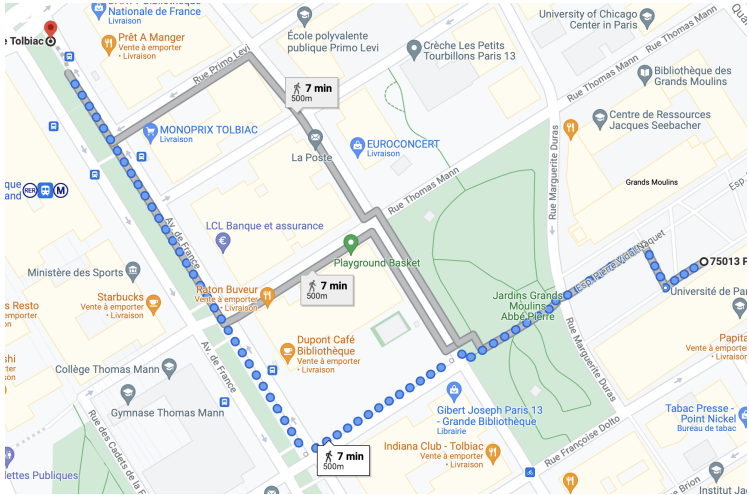
$\{g, e, h\}, \{b\}, \{c, d, f\}, \{a\}$

Applications du parcours en profondeur : résumé

Nous avons vu 3 applications du parcours en profondeur :

1. Détection de circuits dans les graphes orientés :
   - Un DFS révèle un arc retour ssi le graphe contient un circuit.
2. Tri topologique des graphes acycliques orientés :
   - Il suffit de faire un DFS qui garde trace des temps de visite, et ensuite trier les sommets par nombre post décroissant.
3. Calcul des composantes connexes d'un graphe orienté :
   - On fait DFS sur le graphe inverse G^R et on trie les sommets selon l'ordre décroissant de nombre post
   - Ensuite on fait DFS sur G en utilisant l'ordre trouvé dans la première étape.

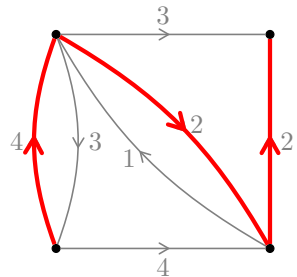
Plus court chemin



Chemins et circuits pondérés

Définition

- Soit $G = (V, E)$ un graphe orienté pondéré (avec pondération $w \in \mathbb{R}^{|E|}$).
- Soit $P \subseteq$ un chemin dans G.
- La *longueur* (ou poids) du chemin P est définie comme $\sum_{e \in E(P)} w_e$.

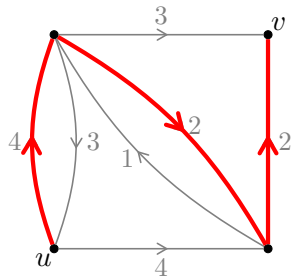


Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, E)$ (avec pondération $w \in \mathbb{R}^{|E|}$). La *distance* de u à v est définie comme

$$\mathrm{dist}(u, v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



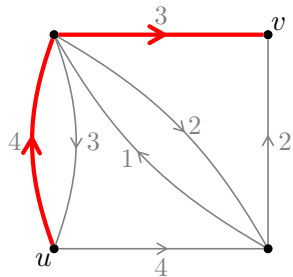
$\mathrm{dist}(u, v) \leq 8$

Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, E)$ (avec pondération $w \in \mathbb{R}^{|E|}$). La *distance* de u à v est définie comme

$$\mathrm{dist}(u, v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



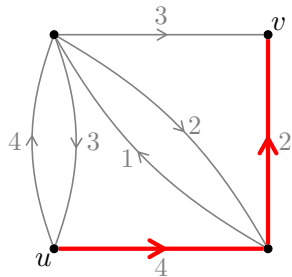
$\mathrm{dist}(u, v) \leq 7$

Distance

Définition

Soient u, v deux sommets dans un graphe orienté pondéré $G = (V, E)$ (avec pondération $w \in \mathbb{R}^{|E|}$). La *distance* de u à v est définie comme

$$\mathrm{dist}(u, v) = \min\{w(P) : P \text{ est un chemin de } u \text{ à } v\}$$



$\mathrm{dist}(u, v) = 6$

Le problème du plus court chemin

Problème

Étant donné un graphe orienté $G = (V, E)$ pondéré (avec pondération $w \in \mathbb{R}^{|E|}$) et deux sommets $u \neq v$ dans V, trouver un plus court chemin (« chaîne orientée ») de u vers v.

Remarque

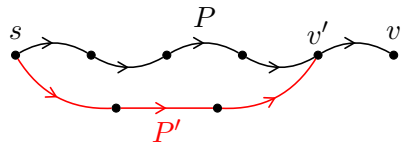
Il peut ne pas exister de plus court chemin de u à v :

- S'il n'y a aucun chemin de u à v : $\mathrm{dist}(u, v) = \infty$
- S'il y a un circuit de poids négatif (circuit absorbant) : $\mathrm{dist}(u, v) = -\infty$

Principe de sous-optimalité

Observation

Si P est un plus court chemin de s vers v alors, en notant v' le prédécesseur de v dans ce chemin, le sous-chemin de P qui va de s vers v' est un plus court chemin de s vers v'.



Démonstration par l'absurde

S'il existe P' de s vers v' de poids strictement inférieur au sous-chemin de P de s vers v' alors en concaténant P' à (v, v') on aurait un chemin de s à v' de poids strictement inférieur à celui de P, contradiction.

Illustration de l'algorithme de Dijkstra

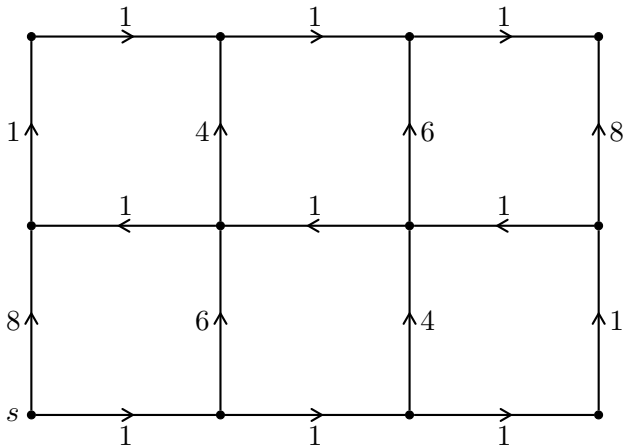


Illustration de l'algorithme de Dijkstra

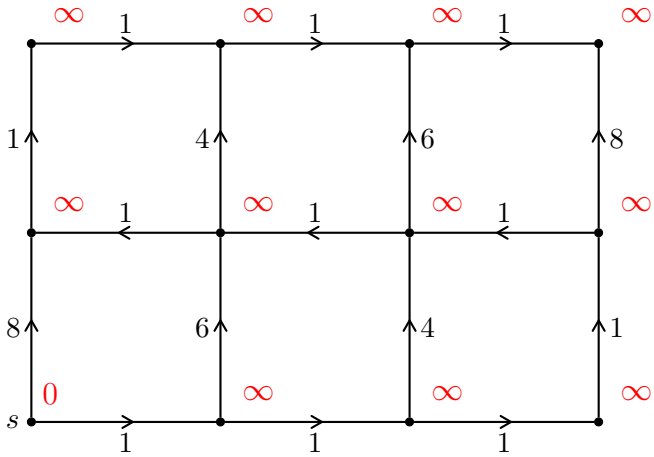


Illustration de l'algorithme de Dijkstra

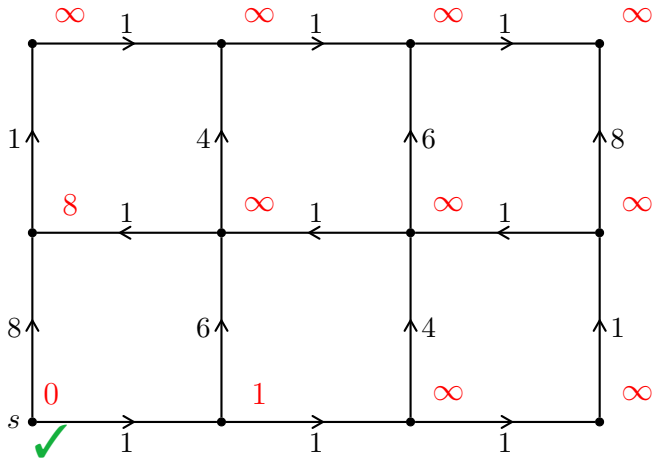


Illustration de l'algorithme de Dijkstra

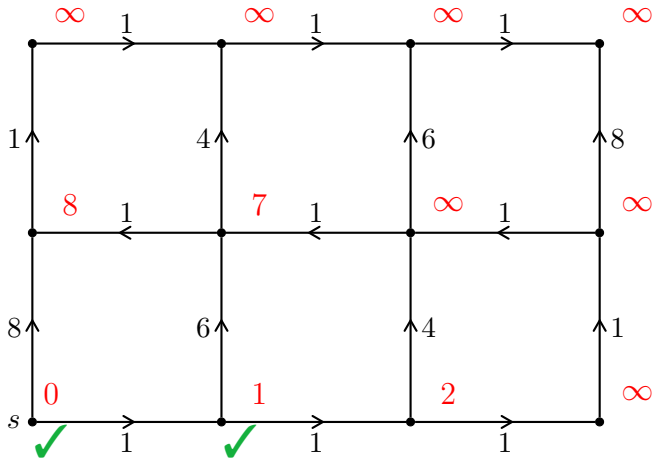


Illustration de l'algorithme de Dijkstra

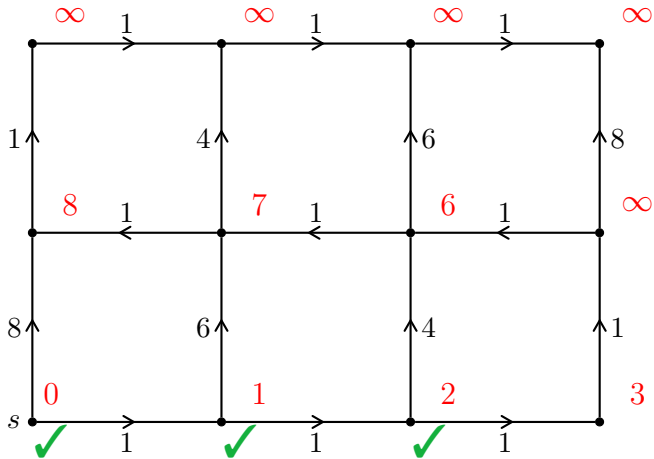


Illustration de l'algorithme de Dijkstra

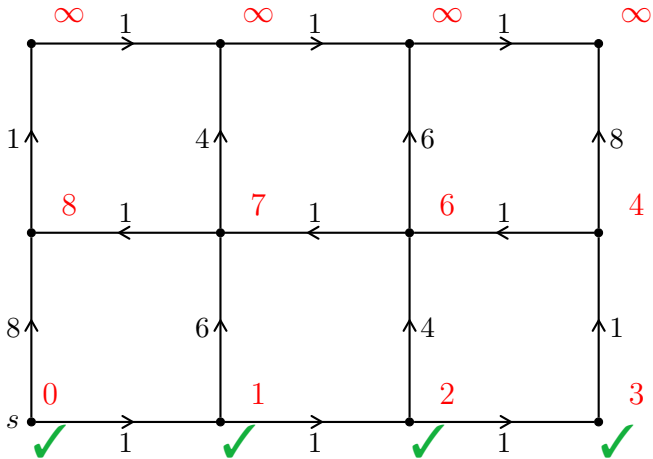


Illustration de l'algorithme de Dijkstra

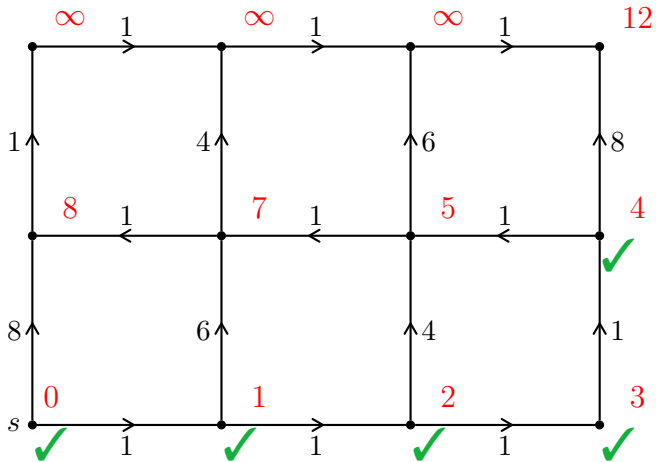


Illustration de l'algorithme de Dijkstra

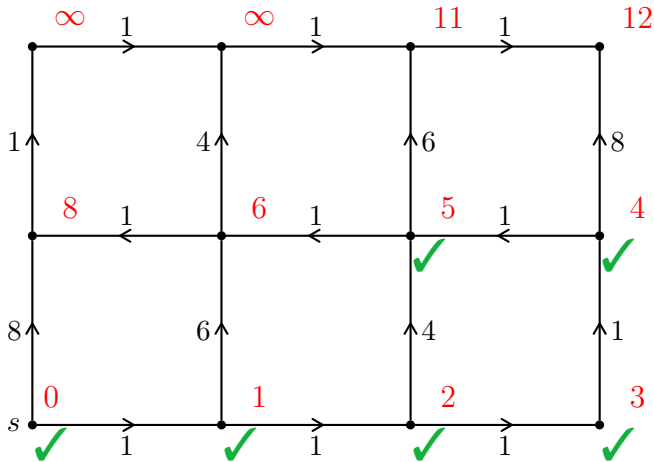


Illustration de l'algorithme de Dijkstra

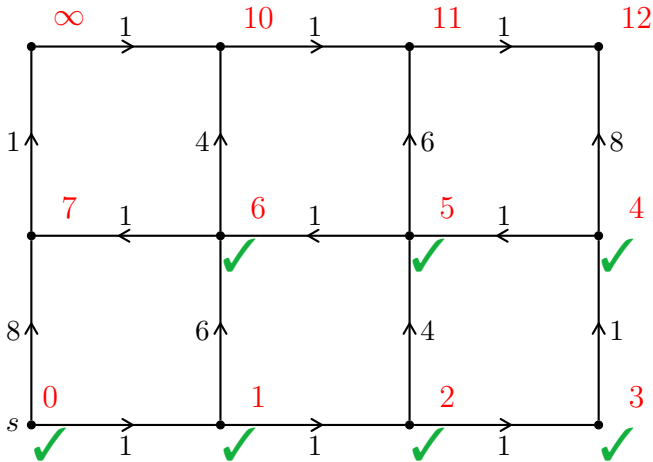


Illustration de l'algorithme de Dijkstra

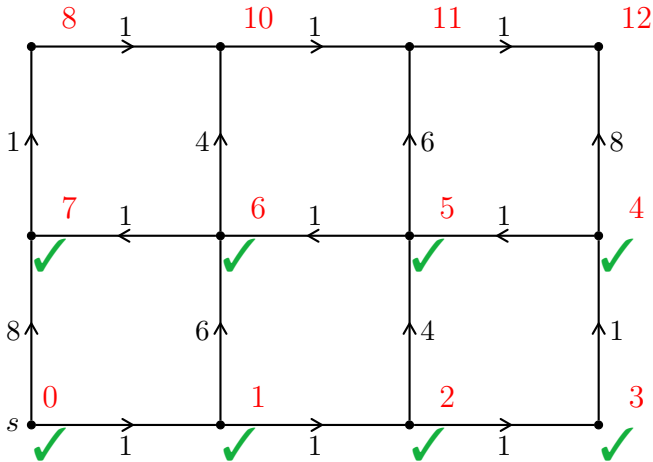


Illustration de l'algorithme de Dijkstra

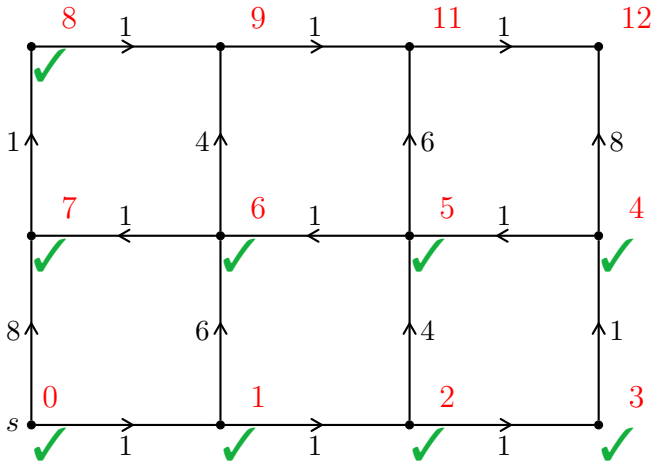


Illustration de l'algorithme de Dijkstra

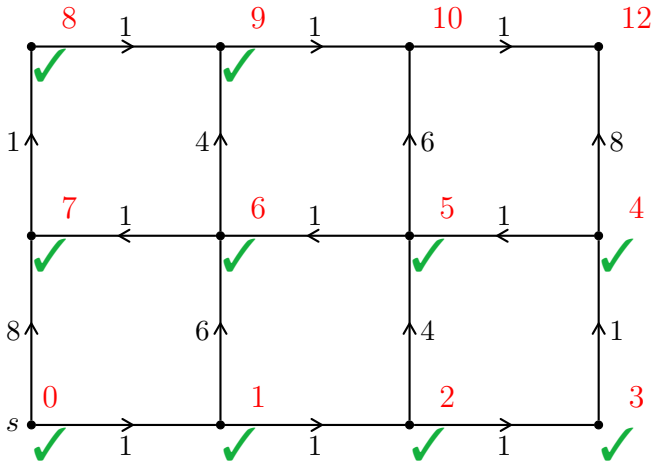


Illustration de l'algorithme de Dijkstra

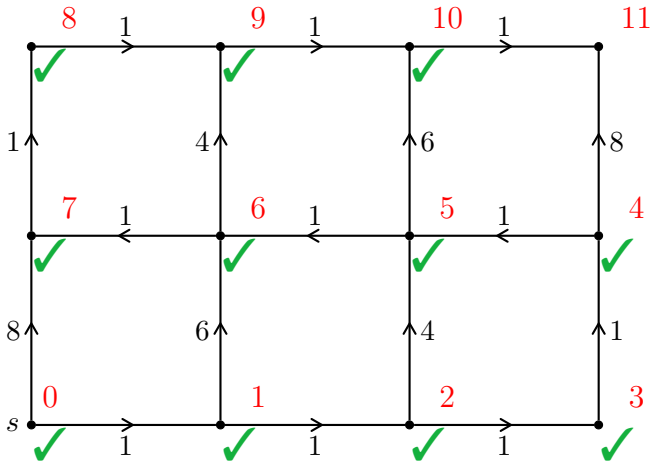
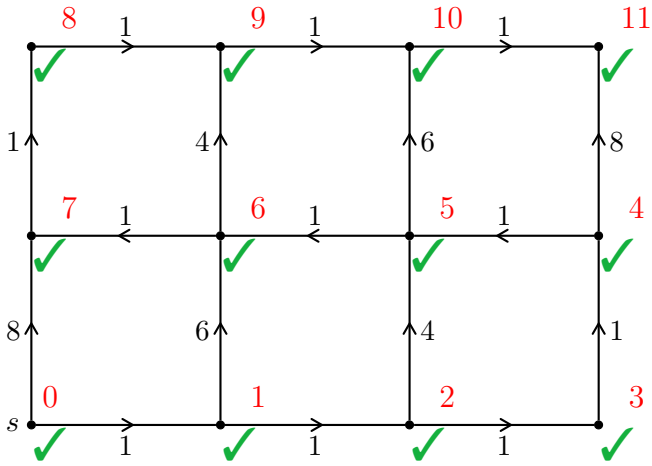


Illustration de l'algorithme de Dijkstra



Algorithme de Dijkstra

Entrées : Graphe orienté $G = (V, E)$ avec pondération $w \in \mathbb{R}^+$, un sommet $s \in V$

Sorties : Distances de s aux autres sommets

$S \leftarrow \emptyset$;

$D[s] \leftarrow 0$;

pour tous les $x \neq s$ **faire**

  $D[x] \leftarrow +\infty$;

tant que $S \neq V$ **faire**

  Trouver $x \notin S$ tel que $D[x]$ est minimum;

  pour tous les $y \notin S$ *tel que* $(x, y) \in E$ **faire**

    $D[y] \leftarrow \min(D[y], D[x] + w(x, y))$;

  $S \leftarrow S \cup \{x\}$

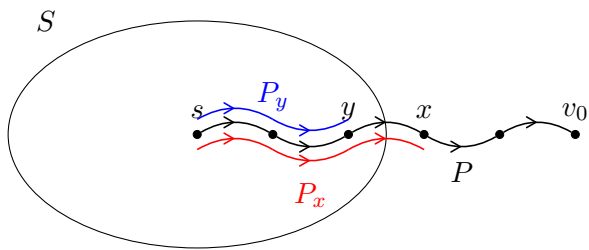
retourner D;

Distances partielles

Définition

Soit $G = (V, E)$ un graphe orienté avec pondération $w \in \mathbb{R}^+$, et $s \in S \subseteq V$. Pour tout $u \in V$, on note $d(u) = \mathrm{dist}(s, u)$. Pour tout sommet $v \notin S$, on définit

$$D(v) = \min \{d(u) + w(u, v) \colon u \in S \text{ et } (u, v) \in E\}.$$



Justification de l'algorithme de Dijkstra (1/3)

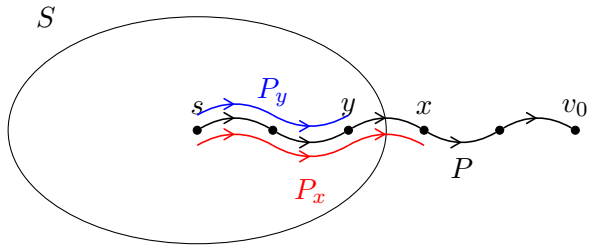
Lemme

Soit v_0 tel que $D(v_0)$ est minimum, parmi tous les sommets dans $V \setminus S$. Alors, $D(v_0) = d(v_0)$.

Démonstration

- Soit v_0 tel que $D(v_0)$ est minimum.
- Comme $D(v_0)$ est la longueur d'un chemin de s à v_0, on a $D(v_0) \geq d(v_0)$.
- Si $D(v_0) \leq d(v_0)$ alors la preuve est terminé.
- Sinon, on suppose $D(v_0) > d(v_0)$.
- Soit P un plus court chemin de s vers v_0 (son poids est alors $d(v_0)$).
- Soit x le premier sommet de P qui n'appartient pas à S.

Justification de l'algorithme de Dijkstra (2/3)



Démonstration (suite)

- Soit $y \in S$ le prédécesseur de x dans ce chemin.
- Soit P' le sous-chemin de P_y de s vers y.
- P_y est un plus court chemin de s vers y (principe de sous optimalité).
- Soit P_x le sous chemin de P de s vers x.

Justification de l'algorithme de Dijkstra (3/3)

Démonstration (suite)

- La longueur de P_x est $d(y) + w(y, x)$.
- Ceci entraîne que $D(x) \leq d(y) + w(y, x) \leq d(v_0) < D(v_0)$.
  - Première inégalité : par définition de D.
  - Deuxième inégalité : tous les poids sont ≥ 0.
  - Troisième inégalité : par hypothèse.
- Implique $D(x) < D(v_0)$ — contradiction avec la minimalité de $D(v_0)$.