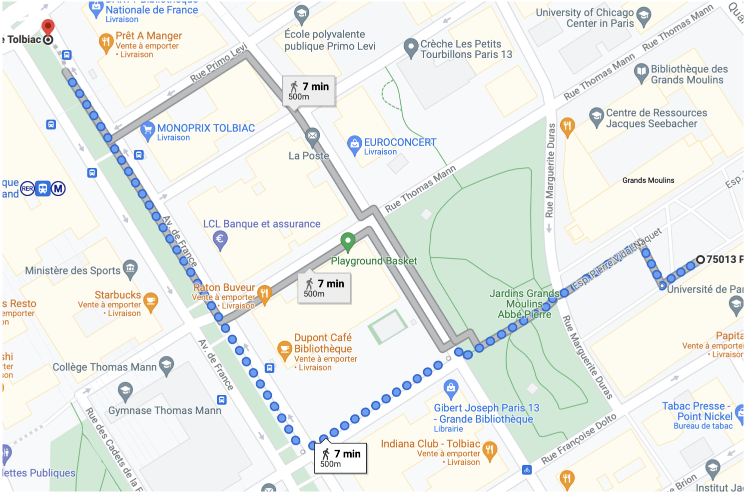
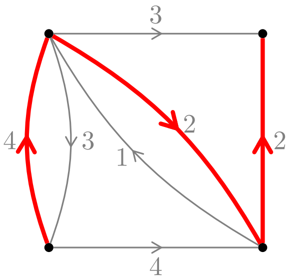
Plus court chemin



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 $\sum_{e \in E(P)} w_e$.

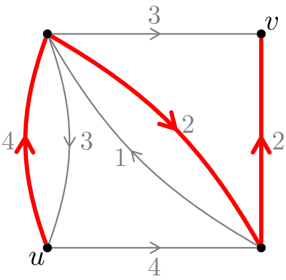


Distance

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

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



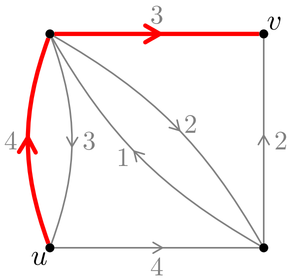
$\mathrm{dist}(u, v) \leq 8$

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

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



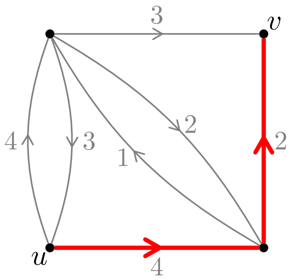
$\mathrm{dist}(u, v) \leq 7$

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

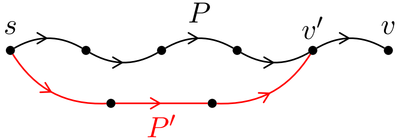
Il peut ne pas exister de plus court chemin de u à v :

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

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.

Algorithme de Dijkstra

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

Sorties : Distances de s aux autres sommets

$S \leftarrow \emptyset$

$D[s] \leftarrow 0$

pour tous les $u \in V \setminus \{s\}$ **faire**

  $D[u] \leftarrow +\infty$

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

  Trouver $u \in V \setminus S$ tel que $D[u]$ est minimum

  $S \leftarrow S \cup \{u\}$

  pour tous les $v \in V \setminus S$ ***tels que*** $(u, v) \in E$ **faire**

    $D[v] \leftarrow \min(D[u], D[v] + \ell(u, v))$

retourner D

Illustration de l'algorithme de Dijkstra

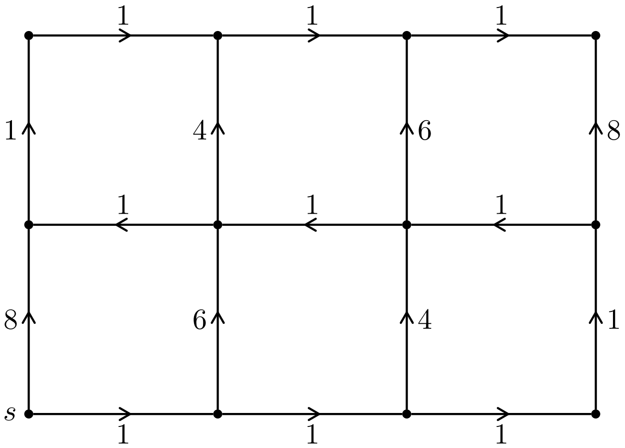


Illustration de l'algorithme de Dijkstra

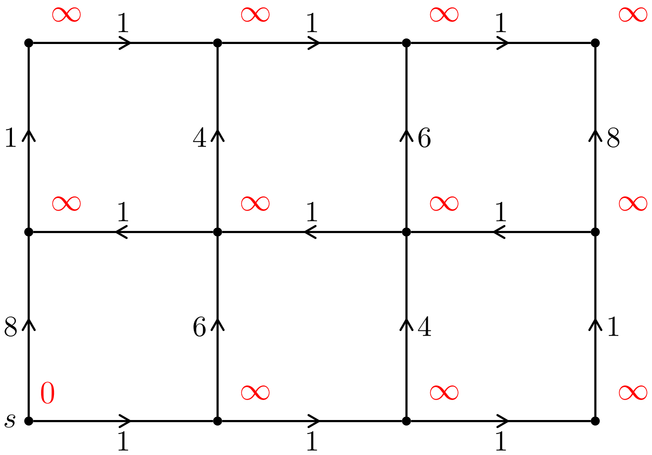


Illustration de l'algorithme de Dijkstra

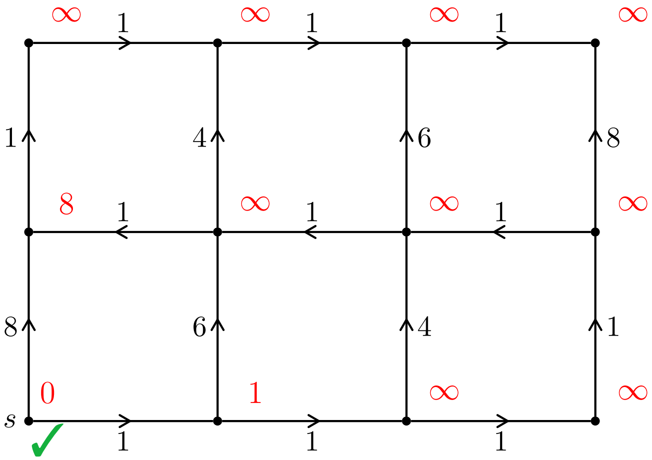


Illustration de l'algorithme de Dijkstra

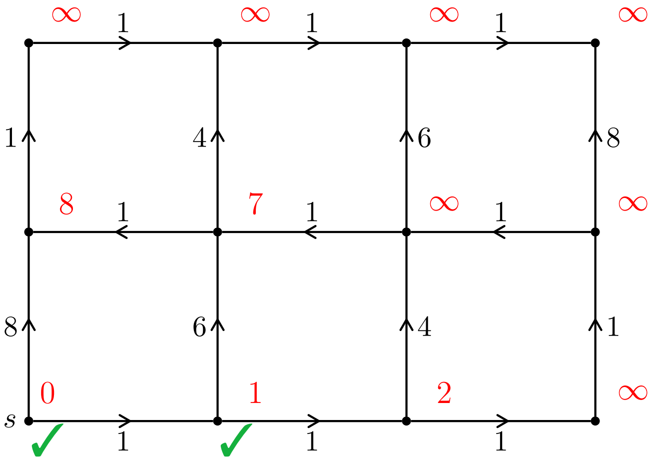


Illustration de l'algorithme de Dijkstra

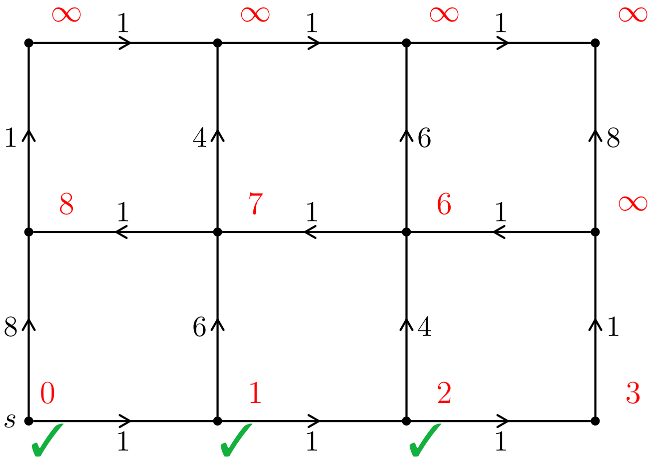


Illustration de l'algorithme de Dijkstra

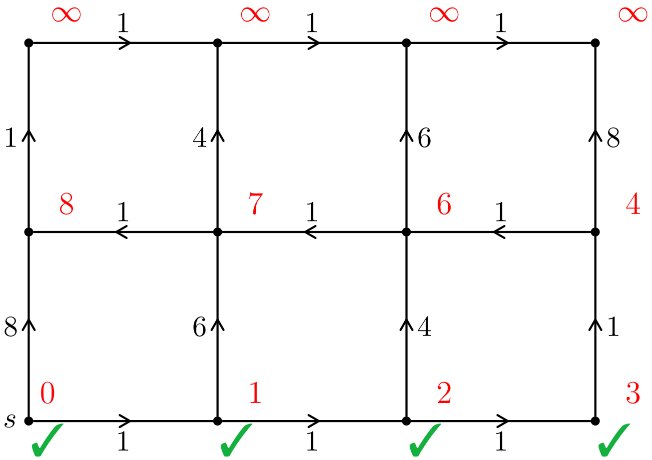


Illustration de l'algorithme de Dijkstra

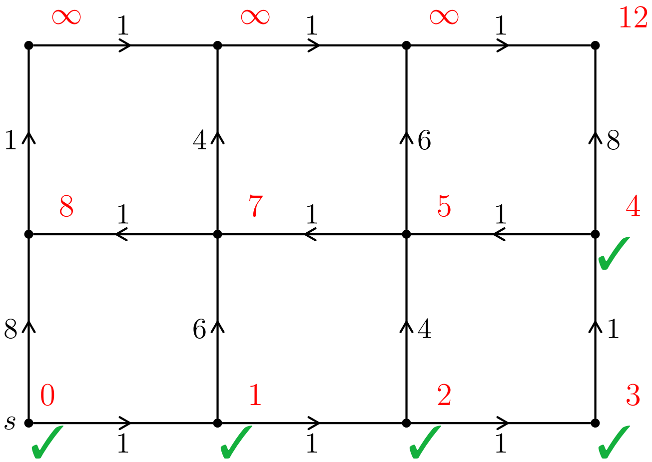


Illustration de l'algorithme de Dijkstra

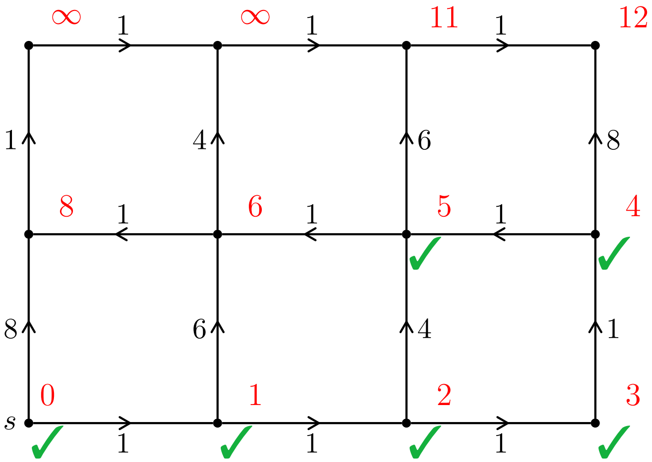


Illustration de l'algorithme de Dijkstra

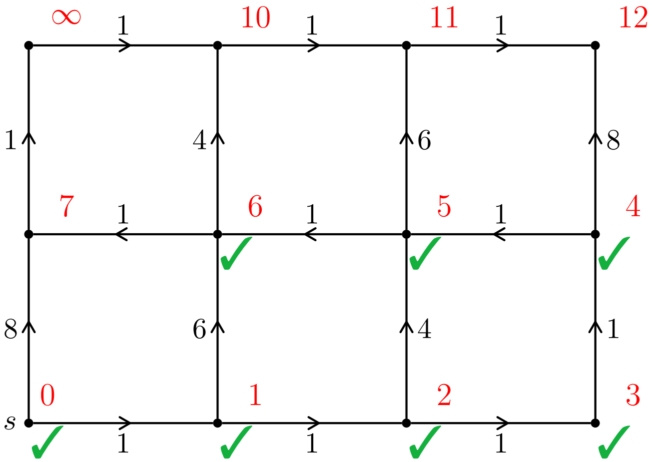


Illustration de l'algorithme de Dijkstra

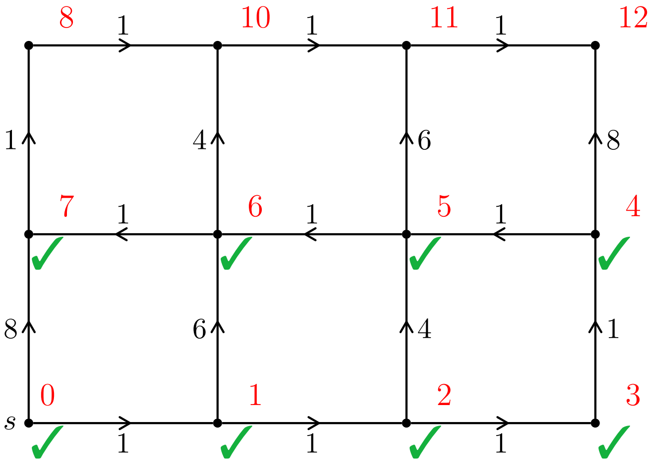


Illustration de l'algorithme de Dijkstra

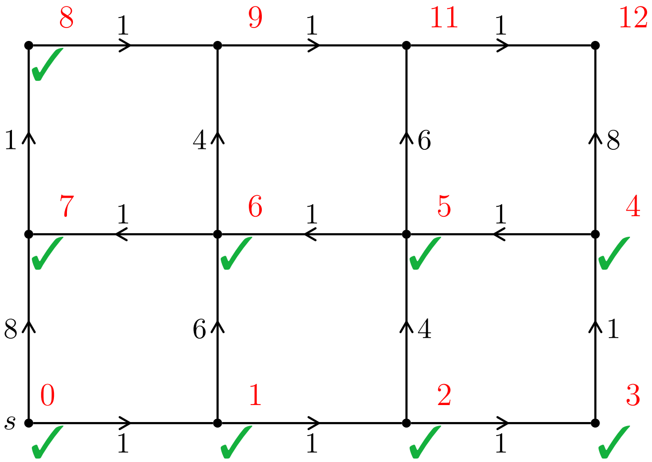


Illustration de l'algorithme de Dijkstra

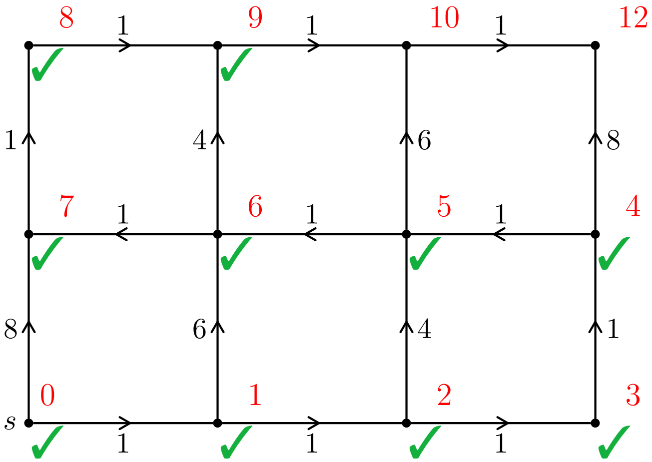


Illustration de l'algorithme de Dijkstra

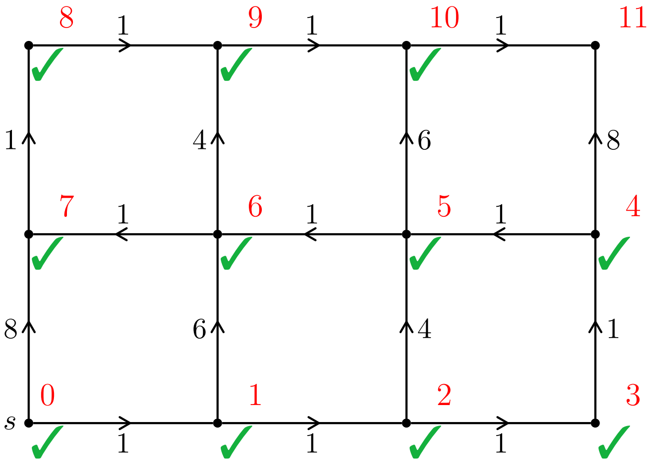
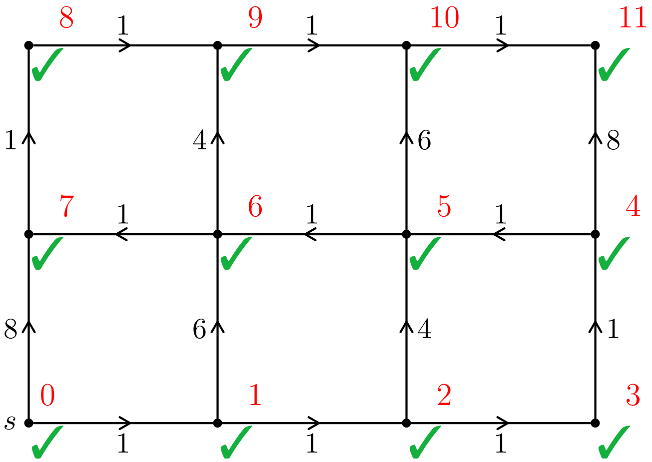


Illustration de l'algorithme de Dijkstra

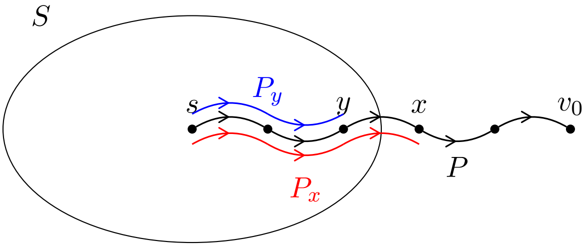


Distances partielles

Définition

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

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



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

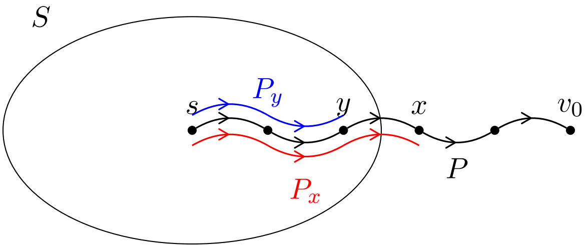
Lemme

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_y le sous-chemin de P 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 $d(y) + \ell(y, x)$.
- Ceci entraîne que $D(x) \leq d(y) + \ell(y, x) \leq d(v_0) < D(v_0)$.
  - 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 le choix de $D(v_0)$.

Les files de priorité

Une *file de priorité* est un type abstrait élémentaire sur laquelle on peut effectuer les opérations suivantes :

- **Insert** insérer un élément
- **Decrease-key** diminuer la valeur de la clé d'un élément particulier
- **Delete-min** retourner l'élément ayant la plus petite clé et supprimer-le de la file
- **Make-queue** créer une file d'attente prioritaire à partir des éléments donnés, avec les valeurs de clés données.

Complexité des opérations dans files de priorité

implémentation	deletemin	insert	decreasekey
liste	$O(n)$	$O(1)$	$O(1)$
tas binaire	$O(\log n)$	$O(\log n)$	$O(\log n)$
tas de Fibonacci	$O(\log n)$	$O(1)$	$O(1)$

Implémentation de Dijkstra avec une file de priorité

pour tous les $u \in V$ **faire**

  $D[u] \leftarrow +\infty$

  $\texttt{prev}[u] \leftarrow \emptyset$

$D[s] \leftarrow 0$

$H \leftarrow \text{makequeue}(V)$

tant que $H \neq \emptyset$ **faire**

  $u \leftarrow \text{deletemin}(H)$

  pour tous les $(u, v) \in E$ **faire**

    si $D[v] > D[u] + w(u, v)$ **alors**

      $D[v] = D[u] + w(u, v)$

      $\texttt{prev}[v] \leftarrow u$

      $\text{decreasekey}(H, v)$

Complexité des opérations dans files de priorité

implémentation	deletemin	insert	decreasekey
liste	$O(n)$	$O(1)$	$O(1)$
tas binaire	$O(\log n)$	$O(\log n)$	$O(\log n)$
tas de Fibonacci	$O(\log n)$	$O(1)$	$O(1)$

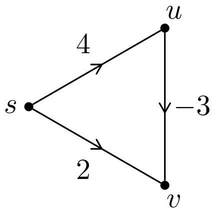
Complexité de l'algorithme de Dijkstra

- L'algorithme de Dijkstra est très similaire au parcours en largeur.
- Cependant, il est plus lent car les files de priorité sont plus exigeantes que les files simples de BFS.
- Puisque makequeue prend au plus autant de temps que $|V|$ opérations d'insertion, nous obtenons un total de $|V|$ deletemin et $|V| + |E|$ opérations d'insertion/decreasekey.
- On obtient ainsi les complexités suivantes de Dijkstra selon l'implémentation de la file de priorité :

liste	$O(n^2)$
tas binaire	$O((n+m)\log n)$
tas de Fibonacci	$O(m + n\log n)$

Arcs négatifs

- L'algorithme de Dijkstra fonctionne en partie parce que le plus court chemin entre le point de départ s et un sommet v doit passer exclusivement par des sommets plus proches que v.
- Ce n'est plus le cas lorsque la longueur des arêtes peut être négative.
- Dans cet exemple, le plus court chemin de s à v passe par u, un sommet plus éloigné



Dijkstra vu comme une séquence de mises à jour

- Pour tout sommet v, la valeur de $D[v]$ est toujours supérieure ou égale à $\mathrm{dist}(s, v)$.
- Les valeurs de $D[v]$ changent uniquement grâce à la “mise à jour” le long d’un arc :

  Procédure $\texttt{maj}(u, v)$**:**

  $D[v] \leftarrow \min\{D[v], D[u] + \ell(u, v)\}$

- Cette opération, basée sur le principe de sous-optimalité, satisfait les propriétés suivantes :
  - Elle donne la distance correcte de s à v dans le cas particulier où u est l’avant-dernier noeud du plus court chemin vers v, et $D[u] = \mathrm{dist}(s, u)$.
  - Elle ne rendra jamais $D[v]$ trop petit.

Plus courts chemins dans les graphes avec arcs négatifs

- Soit t un sommet quelconque dans G, et soit $P = (s, u_1, \ldots, u_k, t)$ un plus court chemin de s à t.
- P comprend au plus $n - 1$ arcs (sinon, P n'est pas élémentaire).
- Si la séquence de `maj` comprend $(s, u_1), (u_1, u_2), (u_2, u_3), \ldots, (u_k, t)$, dans cet ordre (mais pas nécessairement de manière consécutive), alors, grâce à la première propriété de la diapo précédente, $D[t] = \mathrm{dist}(s, t)$, *même s'il y a des arcs négatifs* !
- Il suffit donc de mettre à jour tous les arcs $n - 1$ fois.

Algorithme de Bellman–Ford

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**

  $D[x] \leftarrow +\infty$

  $\texttt{prev}[u] \leftarrow \emptyset$

repéter $|V| - 1$ **fois**

  pour tous les $e \in E$ **faire**

    $\texttt{maj}(e)$

retourner $D, \texttt{prev}$

Illustration de l'algorithme de Bellman–Ford

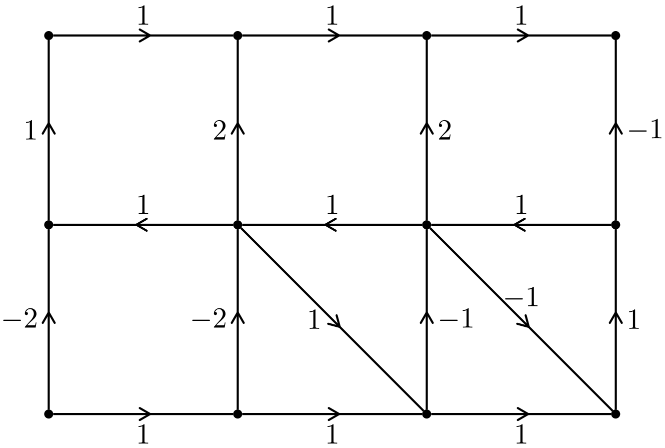


Illustration de l'algorithme de Bellman–Ford

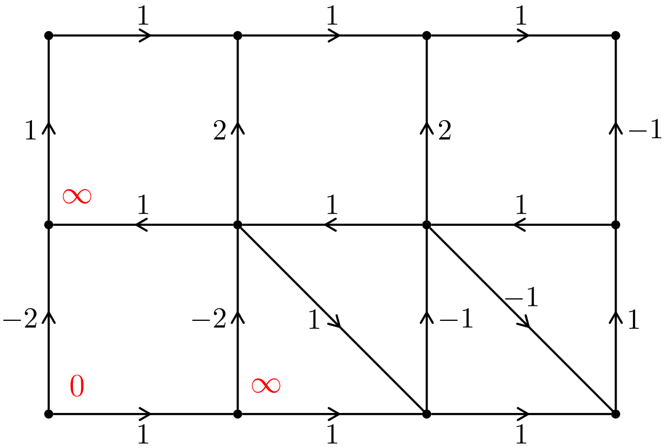


Illustration de l'algorithme de Bellman–Ford

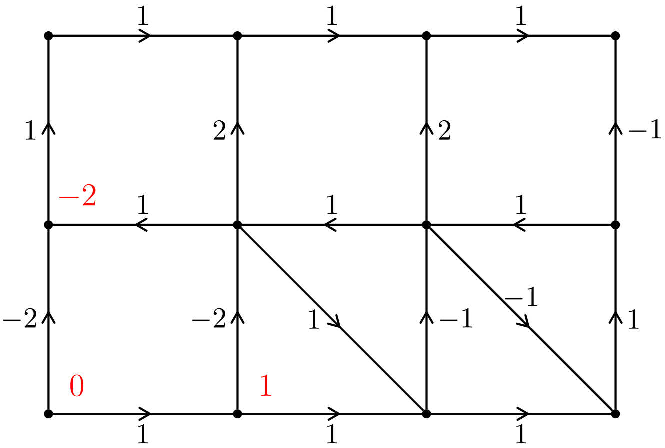


Illustration de l'algorithme de Bellman–Ford

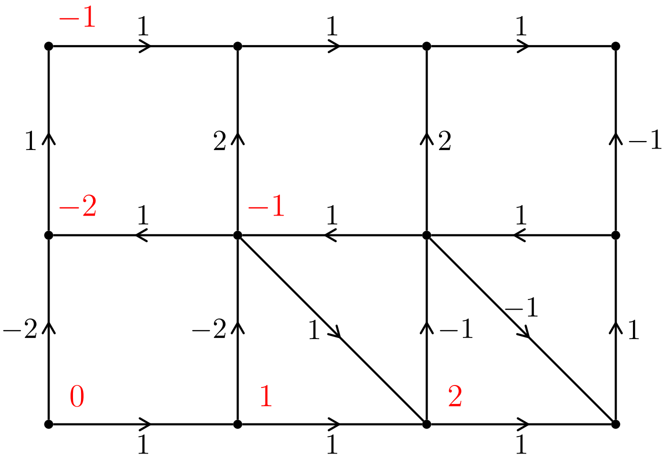


Illustration de l'algorithme de Bellman–Ford

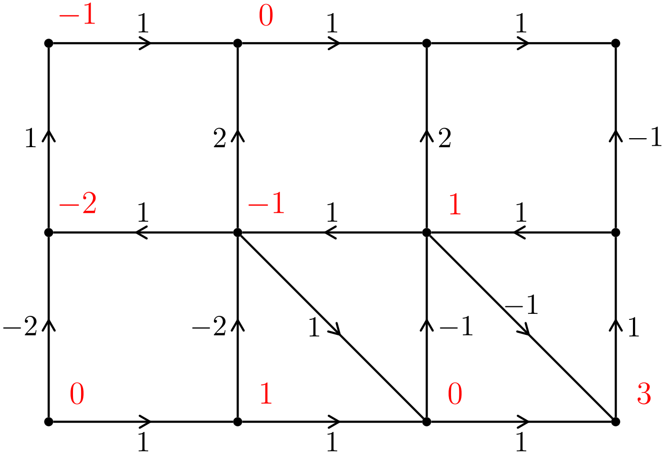


Illustration de l'algorithme de Bellman–Ford

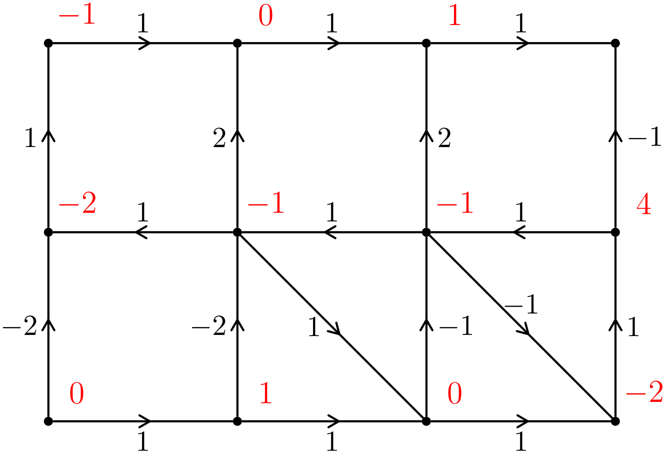


Illustration de l'algorithme de Bellman–Ford

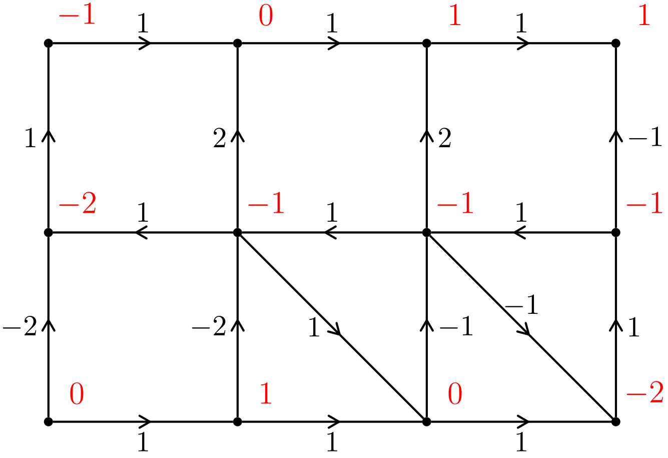
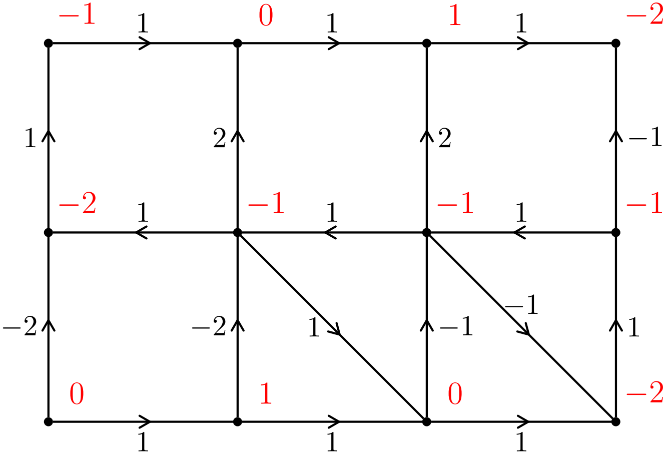


Illustration de l'algorithme de Bellman–Ford



Complexité et correction de l'algorithme de Bellman–Ford

- La complexité de Bellman–Ford est de $O(nm)$.
- Pour la correction de l'algorithme de Bellman–Ford, il suffit de prouver le lemme suivant.
- La démonstration peut se faire par récurrence.

Lemme

Après i itérations de la boucle principale :

- Si $D[u] \neq +\infty$, alors $D[u]$ est la longueur d'un chemin de s à u ;
- S'il existe un chemin de s à u comprenant au plus i arcs, alors la valeur de $D[u]$ est inférieure ou égale à la longueur d'un plus court chemin de s à u comprenant au plus i arcs.

Détection de cycles négatifs

- Une légère modification de l'algorithme de Bellman–Ford nous permet de détecter les cycles négatifs.
- Après avoir fait les $|V|-1$ itérations de la boucle, faire une itération supplémentaire.
- Un cycle négatif existe dans G ssi il y a au moins un changement dans le tableau D lors de la dernière itération.
- Détecter les cycles négatifs a des applications importantes dans la vie réele.
- Une application classique est l'arbitrage de devises (voir le TD).

Deix classes naturelles de graphes orientés sans cycles négatifs

- Il y a deux classes naturelles de graphes orientés sans cycles négatifs :
  - les graphes sans arcs négatifs
  - les graphes sans cycles orientés.
- Dans les graphes sans arcs négatifs, on peut utiliser l'algorithme de Dijkstra.

Plus court chemin dans les graphes orientés acycliques (DAG)

- Il faut effectuer une séquence de mises à jour qui inclut chaque plus court chemin comme sous-séquence.
- Dans tout chemin d'un DAG, les sommets apparaissent dans un ordre topologique croissant.
- Par conséquent, il suffit de faire un tri topologique du DAG par une recherche en profondeur, et puis de parcourir les sommets dans l'ordre topologique, en mettant chaque fois à jour tous les arcs sortants du sommet.
- La complexité de cet algorithme est de $O(n+m)$.

Algorithme de plus court chemin dans les DAG

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

Sorties : Distances de s aux autres sommets

$S \leftarrow \emptyset$

$D[s] \leftarrow 0$

Tri topologique de G

pour tous les $v \in V$ *dans l'ordre topologique* **faire**

  pour tous les $e \in E$ **faire**

    `maj`(e)

retourner D