Algorithme de Dijkstra

F.M.

2024/2025

Graphe à pondération positive

Dans ce chapitre, **on ne considère que les graphes $G = (S, A, p)$ où p est une pondération positive** : $\forall u \in A,\ p(u) \geq 0$.

Étant donné un sommet r de G, le problème est de déterminer une r-arborescence de cpm représentée par les tables Π (potentiels) et $\mathcal{A}$ (pères). Puisque la pondération est positive, **G ne contient pas de circuit absorbant**.

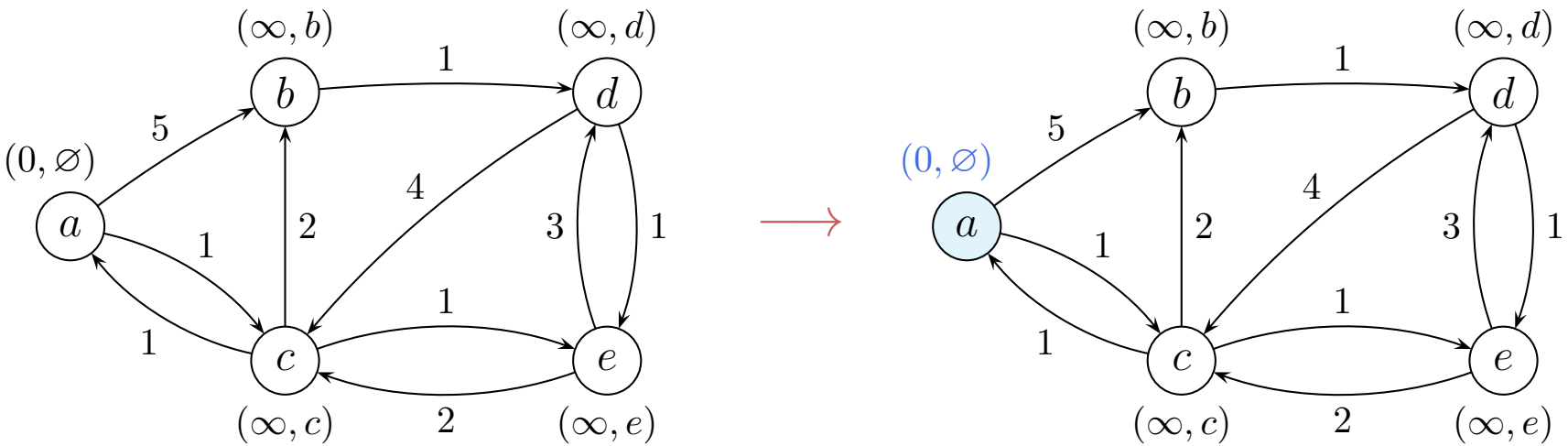
L'algorithme de Ford fournit une solution à ce problème en effectuant un maximum de $n = |S|$ phases de relâchements (une phase consistant à relâcher tous les arcs du graphe).

Dans la suite, on présente un algorithme plus efficace du fait de la positivité de la pondération. On verra qu'une unique phase de relâchements suffit dans ce cas. Cet algorithme est l'algorithme de Dijkstra.

2 / 25

Conséquences d'une pondération positive

Considérons la situation suivante :



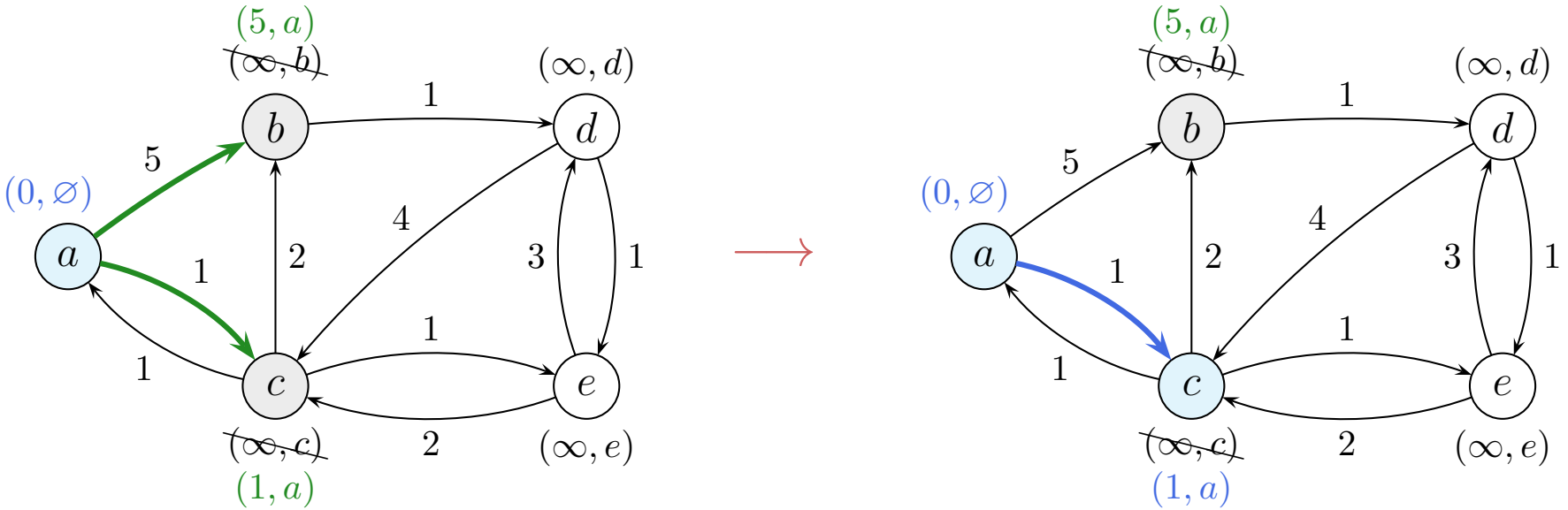
Puisque la pondération est positive, aucun relâchement ne pourra améliorer $\Pi[a]$. Par conséquent : $\boldsymbol{\Pi[a] = \textsc{Dist}(a, a) = 0}$.

De plus, **seuls les arcs d'origine a sont améliorants**. On peut les relâcher **une fois pour toutes** (car $\Pi[a] = \textsc{Dist}(a, a)$ à cet instant).

3 / 25

Conséquences d'une pondération positive (suite)

Relâchement des arcs (a, b) et (a, c) :



Le seule possibilité pour aller de a vers c sans utiliser l'arc (a, c) est de passer par b. La positivité de la pondération implique que : **$\Pi[c] = \textsc{Dist}(a, c) = 1$** et **l'arc (a, c) est un cpm allant de a à c**.

Après relâchement des arcs sortant de a, on a pu identifier une distance et un cpm (du fait de la pondération positive).

4 / 25

Bordure et cocycle

Soit $G = (S, A)$ un graphe orienté et considérons un sous-ensemble de sommets $E \subseteq S$.

On appelle cocycle de E l'ensemble des arcs sortant de E.

On note $\mathcal{C}(E)$ cet ensemble qui contient les arcs dont l'origine se trouve dans E et dont le but se trouve à l'extérieur de E.

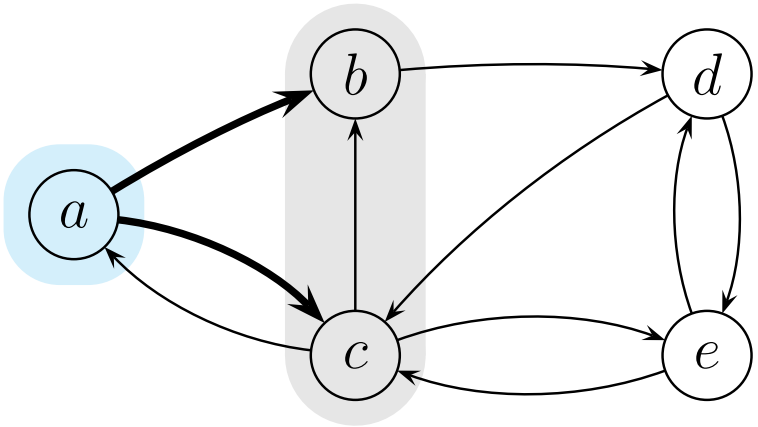
On appelle bordure de E l'ensemble des sommets n'appartenant pas à E mais ayant au moins un prédécesseur dans E.

On note $\mathcal{B}(E)$ cet ensemble qui contient les buts des arcs de $\mathcal{C}(E)$.

Si $E = \{s\}$, on note $\mathcal{C}(s)$ le cocycle de E et $\mathcal{B}(s)$ la bordure de E.

5 / 25

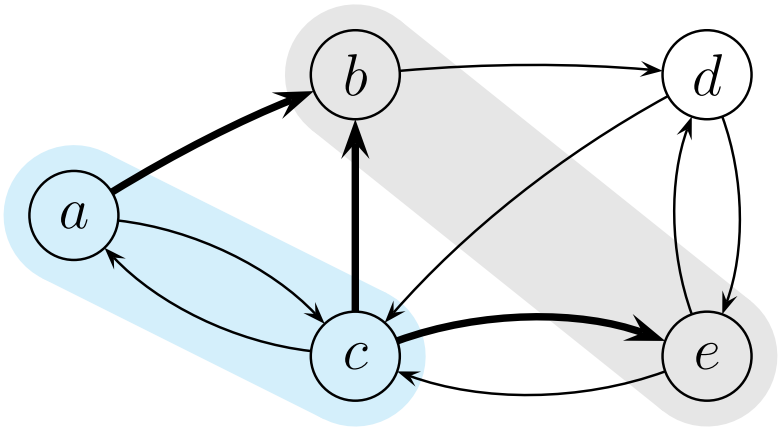
Illustration



$$E = \{a\}$$

$$\mathcal{C}(a) = \{(a, b), (a, c)\}$$

$$\mathcal{B}(a) = \{b, c\} = V^+(a)$$



$$E = \{a, c\}$$

$$\mathcal{C}(a) = \{(a, b), (c, b), (c, e))\}$$

$$\mathcal{B}(a) = \{b, e\}$$

6 / 25

Propriété induite par la pondération positive

Soit $G = (S, A, p)$ un graphe à pondération positive. **Soit T une r-arborescence partielle de chemins de poids minimum** représentée par les tables Π et $\mathcal{A}$. *Partielle* signifie que T contient certains descendants de r mais pas tous.

Soit E l'ensemble des descendants de r se trouvant dans T. Par hypothèse : $\forall x \in E$, $\Pi[x] = \textsc{Dist}(r, x)$.

Après relâchement des arcs de $\mathcal{C}(E)$, le sommet s de $\mathcal{B}(E)$ ayant le plus petit potentiel vérifie : $\Pi[s] = \textsc{Dist}(r, s)$.

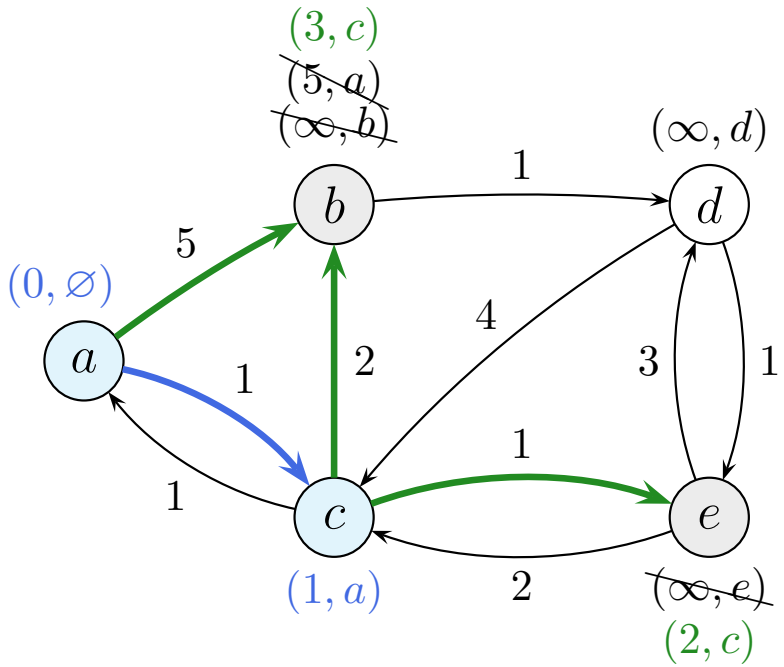
Soit s le sommet de $\mathcal{B}(E)$ ayant le plus petit potentiel après relâchement des arcs de $\mathcal{C}(E)$. Soit (x, s) l'arc de $\mathcal{C}(E)$ vérifiant $\textsc{Dist}(r, x) + p(x, s) = \Pi[s]$. $\Pi[s]$ est le poids du chemin de T allant de r à x prolongé par l'arc (x, s).

Tout autre chemin allant de r à s passe nécessairement par un autre sommet $s' \neq s$ de $\mathcal{B}(E)$. Or $\Pi[s'] \geq \Pi[s]$. Donc cet autre chemin aura un poids supérieur à $\Pi[s]$ du fait de la pondération positive. On a donc $\Pi[s] = \textsc{Dist}(r, s)$.

7 / 25

Illustration de la propriété

Considérons la situation suivante où l'arborescence partielle de cpm ne contient que l'arc (a, c) :



$E = \{a, c\}$ $\quad$ $\mathcal{B}(E) = \{b, e\}$

$\mathcal{C}(E) = \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{e})\}$

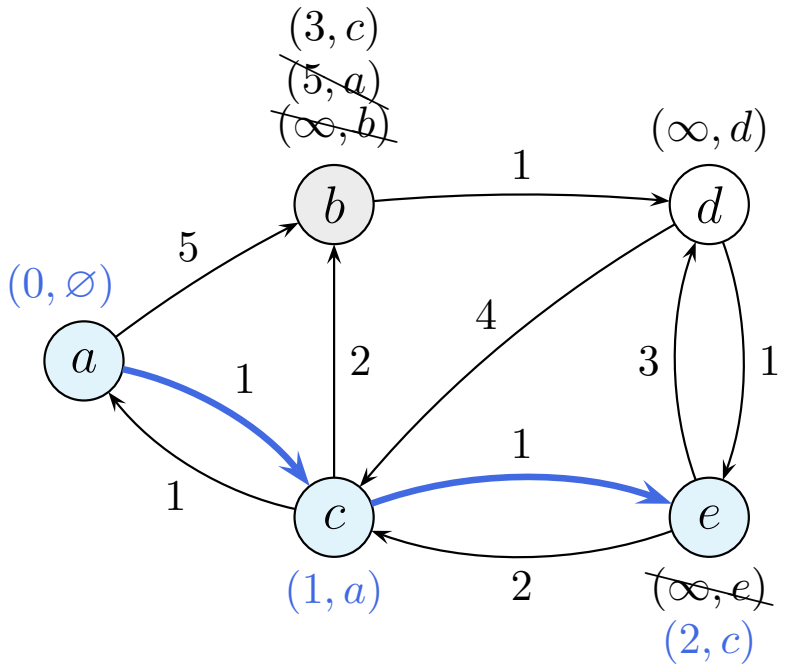
relâchement de (a, b) : $\Pi[b] \leftarrow 5$
relâchement de (c, b) : $\Pi[b] \leftarrow 3$
relâchement de (c, e) : $\Pi[e] \leftarrow 2$

Propriété $\Rightarrow \Pi[e] = \textsc{Dist}(a, e) = 2$

8 / 25

Illustration de la propriété

On peut maintenant **ajouter le sommet e à E** et l'arc (c, e) à l'arborescence :



$$E = \{a, c, e\}$$

C'est ainsi que procède l'algorithme de Dijkstra : il construit progressivement une r-arborescence de cpm en lui ajoutant un sommet (ou un arc) à chaque étape.

9 / 25

Principe de l'algorithme de Dijkstra

L'algorithme de Dijkstra détermine une r-arborescence de chemins de poids minimum (représentée par les tables $\mathcal{A}$ et Π) dans un graphe $G = (S, A, p)$ à **pondération positive**. Il utilise :

- **Un ensemble de sommet E** : $\forall s \in E$, $\Pi[s] = \textsc{Dist}(r, s)$.
- **Un ensemble B représentant la bordure de E**.

Chaque itération consiste à :

(1) **Extraire de B le sommet s de plus petit potentiel puis ajouter s à E**. À cet instant, $\Pi[s] = \textsc{Dist}(r, s)$ et s est dit **examiné**.

(2) **Relâcher une fois pour toutes les arcs sortant de s** (les autres arcs de $\mathcal{C}(E)$ ont été relâchés lors des itérations précédentes) et **mettre à jour B** (car E est modifié).

La positivité de la pondération permet de définir un ordre de relâchement des arcs de sorte que, une et seule phase de relâchements suffit pour calculer toutes les distances.

10 / 25

Algorithme de Dijkstra : initialisations

✎ **initialisation des tables**

pour tout $s \in S$ **faire**

$\quad \Pi[s] \leftarrow \infty \; ; \; \mathcal{A}[s] \leftarrow s$

fin pour

$\Pi[r] \leftarrow 0 \; ; \; \mathcal{A}[r] \leftarrow \varnothing$

✎ **initialisation des ensembles**

$E \leftarrow \emptyset \; ; \; B \leftarrow \{r\}$

Initialement E est vide (aucun sommet n'est examiné) et on place le sommet r dans la bordure B. Ainsi, lors de la première itération, le sommet r sera le premier sommet à basculer de B vers E.

11 / 25

Algorithme de Dijkstra : itérations

✎ **Itérations : une seule phase de relâchements**

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

  $s \leftarrow$ le sommet de B ayant le plus petit potentiel

  ✎ **ici s vérifie $\Pi[s] = \textsc{Dist}(r, s)$ et s est dit examiné**

  $B \leftarrow B - \{s\}$; $E \leftarrow E \cup \{s\}$

  ✎ **relâchement des arcs sortant de s et m.a.j. de B**

  pour tout $x \in V^+(s) \cap \bar{E}$ **faire**

    si $\Pi[s] + p(s, x) < \Pi[x]$ **alors**

      $\Pi[x] \leftarrow \Pi[s] + p(s, x)$; $\mathcal{A}[x] \leftarrow s$; $B \leftarrow B \cup \{x\}$

    fin si

  fin pour

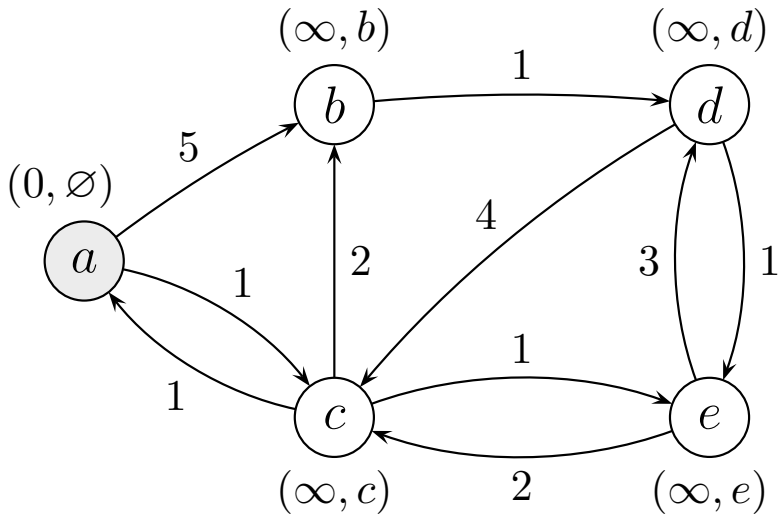
fin tq

On ne relâche que les arcs sortant de s (les autres arcs sortant de E ont déjà été relâchés). L'ensemble E est superflu (uniquement un rôle pédagogique) et B suffit. En effet, si x est un successeur de s appartenant à E alors $\Pi[x] = \textsc{Dist}(r, x)$ et (s, x) ne peut être améliorant.

12 / 25

Trace de l'algorithme de Dijkstra

Initialisations



E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a

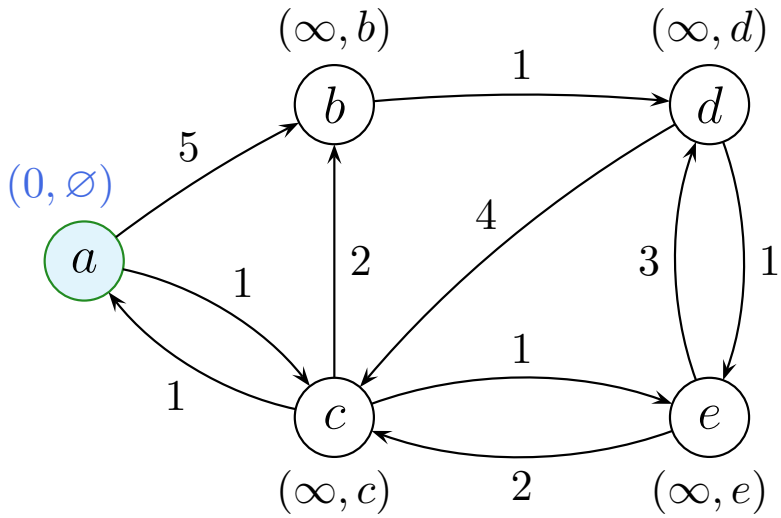
En pratique, on déroule l'algorithme sur le diagramme et on remplit en parallèle un tableau de résultats (la trace de l'algorithme). À chaque itération, on modifie les annotations des sommets (en cas d'amélioration) et on ajoute une ligne au tableau dans laquelle on reporte les résultats obtenus.

13 / 25

Trace de l'algorithme de Dijkstra

Itération 1

a bascule de B vers E
$\Rightarrow \textsc{Dist}(a, a) = \Pi[a] = 0$



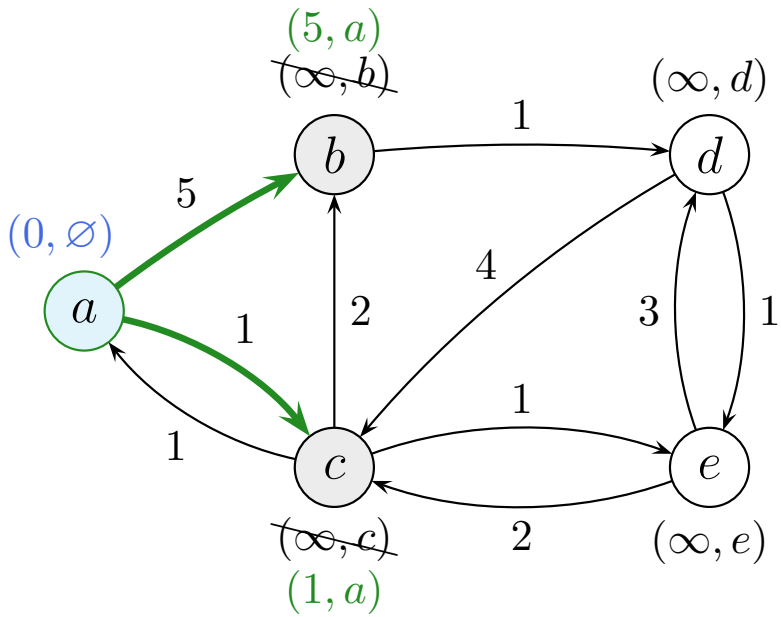
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
$\boldsymbol{a}$	$0, \varnothing$					

14 / 25

Trace de l'algorithme de Dijkstra

Itération 1

relâchement de (a, b) et de (a, c)
mise à jour de B



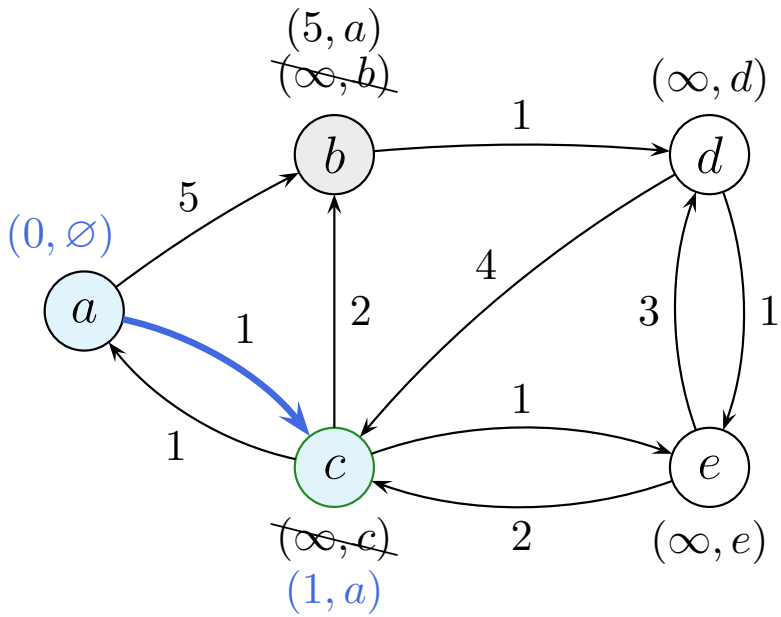
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
$\boldsymbol{a}$	$0, \varnothing$	$\mathbf{5, a}$	$\mathbf{1, a}$	∞, d	∞, e	$b\ c$

15 / 25

Trace de l'algorithme de Dijkstra

Itération 2

c bascule de B vers E
$\Rightarrow \textsc{Dist}(a, c) = \Pi[c] = 1$



E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
$\boldsymbol{c}$			$1, a$			

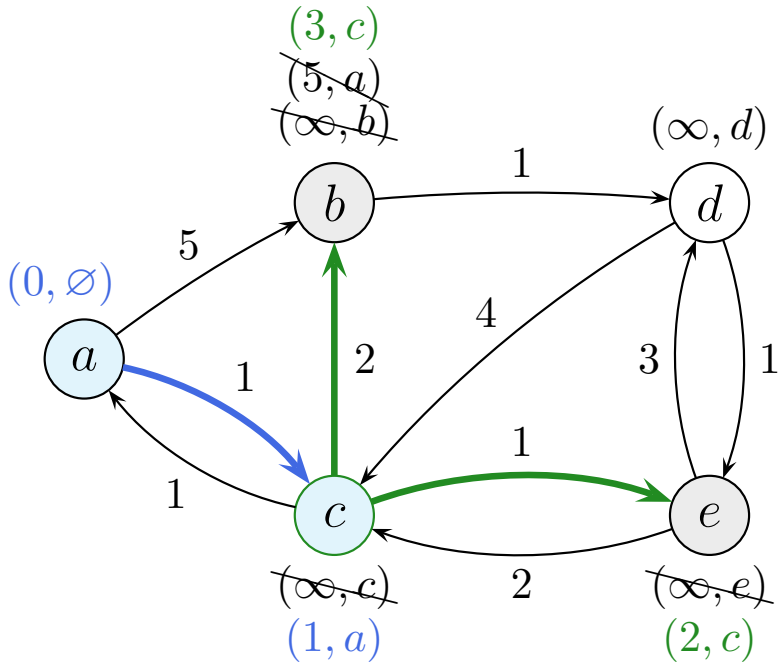
Les arcs de l'arborescence des cpm sont colorés en bleu afin de visualiser les étapes de sa construction.

16 / 25

Trace de l'algorithme de Dijkstra

Itération 2

relâchement de (c, b) et de (c, e)
mise à jour de B



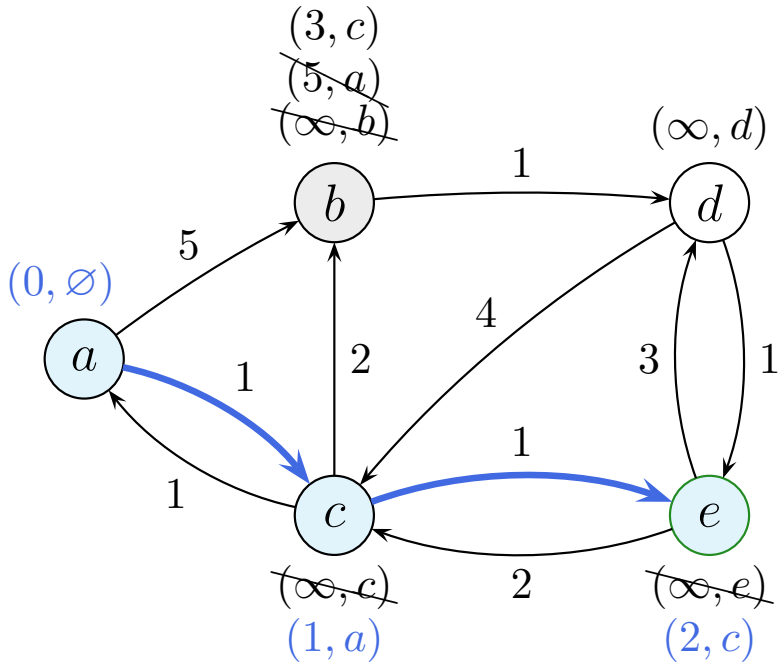
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
$\boldsymbol{c}$		$\mathbf{3, c}$	$1, a$	∞, d	$\mathbf{2, c}$	$b\ e$

17 / 25

Trace de l'algorithme de Dijkstra

Itération 3

e bascule de B vers E
$\Rightarrow \textsc{Dist}(a, e) = \Pi[e] = 2$



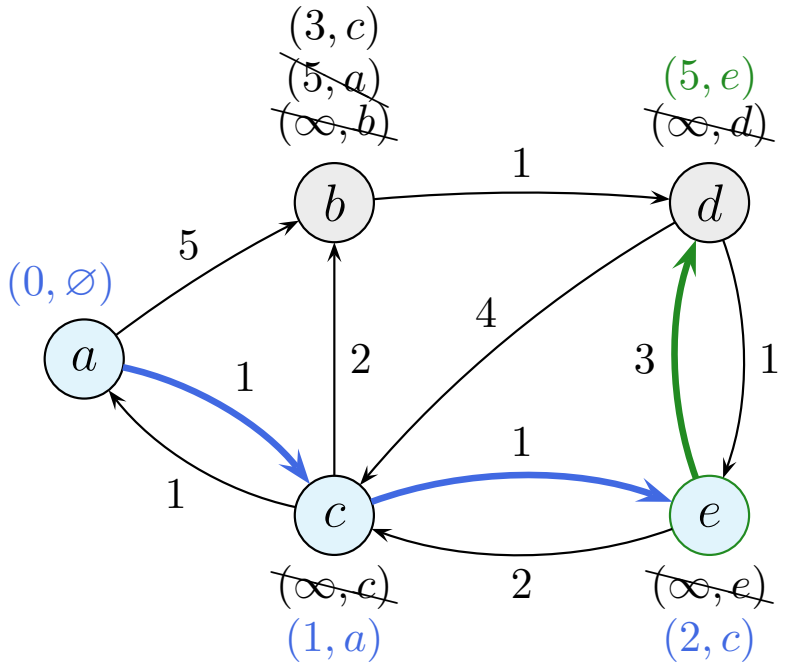
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
c		$3, c$	$1, a$	∞, d	$2, c$	$b\ e$
$\boldsymbol{e}$					$2, c$	

18 / 25

Trace de l'algorithme de Dijkstra

Itération 3

relâchement de (e, d)
mise à jour de B



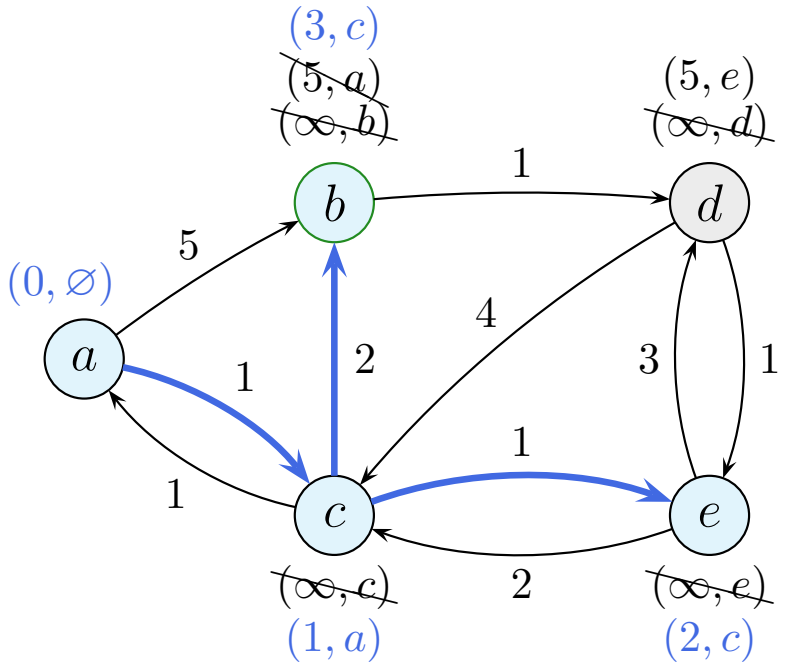
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
c		$3, c$	$1, a$	∞, d	$2, c$	$b\ e$
$\boldsymbol{e}$		$3, c$		$\mathbf{5, e}$	$2, c$	$b\ d$

19 / 25

Trace de l'algorithme de Dijkstra

Itération 4

b bascule de B vers E
$\Rightarrow \textsc{Dist}(a, b) = \Pi[b] = 3$



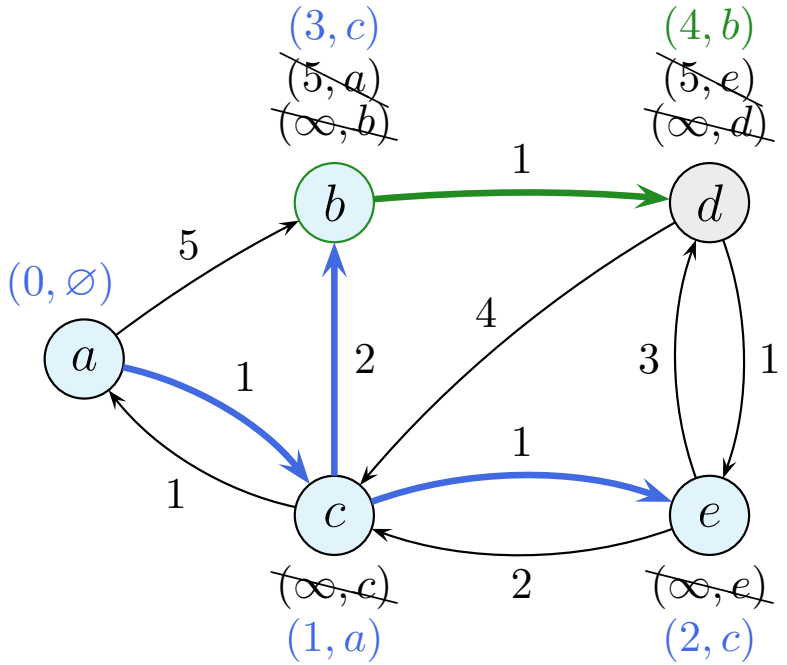
E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
c		$3, c$	$1, a$	∞, d	$2, c$	$b\ e$
e		$3, c$		$5, e$	$2, c$	$b\ d$
$\boldsymbol{b}$		$3, c$				

20 / 25

Trace de l'algorithme de Dijkstra

Itération 4

relâchement de (b, d)
mise à jour de B



E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
c		$3, c$	$1, a$	∞, d	$2, c$	$b\ e$
e		$3, c$		$5, e$	$2, c$	$b\ d$
$\boldsymbol{b}$		$3, c$		$\mathbf{4, b}$		d

21 / 25

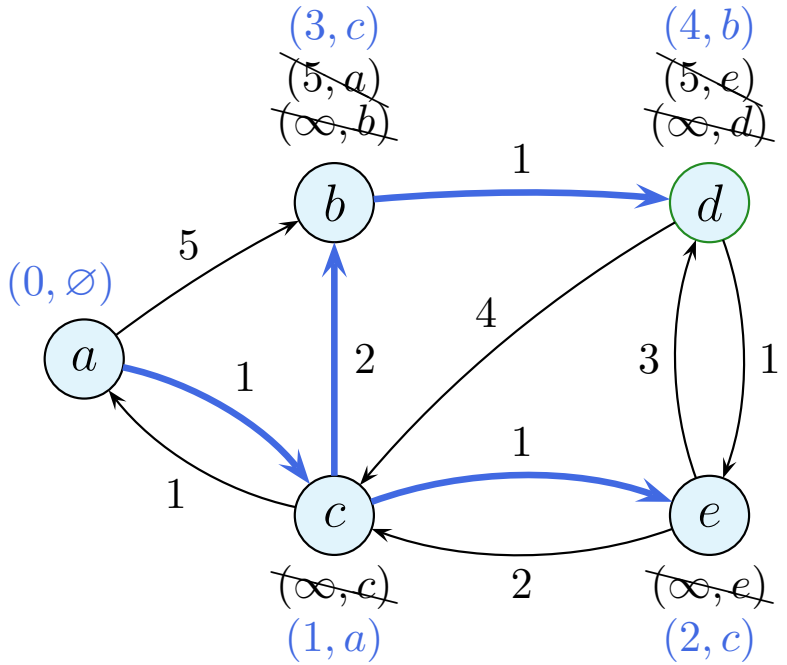
Trace de l'algorithme de Dijkstra

Itération 5

d bascule de B vers E
$\Rightarrow \textsc{Dist}(a, d) = \Pi[d] = 4$

aucun relâchement

$B = \emptyset \Rightarrow$ fin de l'algorithme

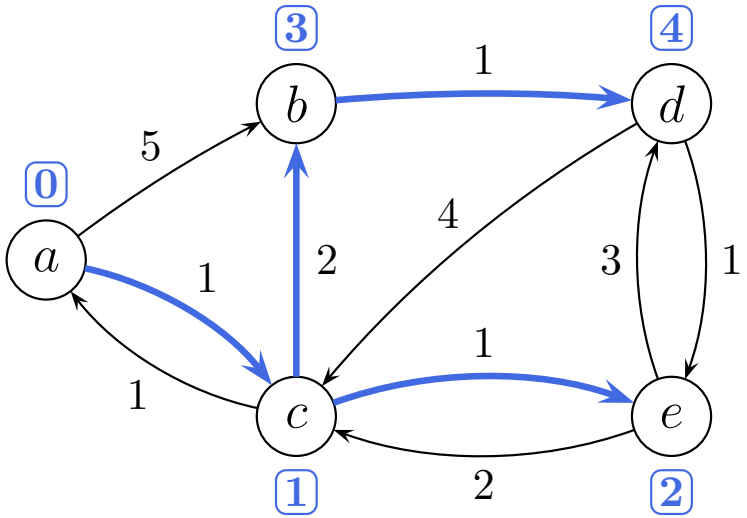


E	a	b	c	d	e	B
$\emptyset$	$0, \varnothing$	∞, b	∞, c	∞, d	∞, e	a
a	$0, \varnothing$	$5, a$	$1, a$	∞, d	∞, e	$b\ c$
c		$3, c$	$1, a$	∞, d	$2, c$	$b\ e$
e		$3, c$		$5, e$	$2, c$	$b\ d$
b		$3, c$		$4, b$		d
$\boldsymbol{d}$				$4, b$		$\emptyset$

22 / 25

Trace de l'algorithme de Dijkstra

on a obtenu
une a-arborescence de cpm



s	a	b	c	d	e
$\Pi[s]$	**0**	**3**	**1**	**4**	**2**
$\mathcal{A}[s]$	$\varnothing$	c	a	b	c

23 / 25

Implémentation de l'algorithme de Dijkstra

À chaque itération, il faut rechercher dans B le sommet de plus petit potentiel. Cette recherche est la partie la plus *coûteuse* de l'algorithme.

Une solution est de représenter B par un tableau de sommets triés dans l'ordre croissant des potentiels. Mais la suppression d'un sommet et l'insertion d'un nouveau sommet sont des opérations coûteuses.

Il est préférable d'utiliser une **file de priorité**. L'idée est d'associer une priorité à chaque élément et de faire en sorte que l'élément en tête de file soit toujours celui de plus forte priorité. Cette file est implémentée de sorte que les opérations de suppression (défiler, pop, get) et d'insertion (enfiler, push, put) soient très rapides.

Dans l'algorithme de Dijkstra, la priorité d'un sommet est définie par son potentiel : plus le potentiel est petit (resp. grand) et plus la priorité est grande (resp. petite).

24 / 25

En conclusion

L'algorithme de Dijkstra ne fonctionne que si tous les poids sont positifs (si des poids sont négatifs, l'algorithme ne fonctionne pas). Il est plus efficace que l'algorithme de Ford dans ce cas.

L'algorithme de Dijkstra peut être utilisé pour rechercher des chaînes de poids minimum dans des graphes non orientés à pondération positive.

L'algorithme de Dijkstra ne permet pas de rechercher des chemins de poids maximum : tout circuit est absorbant quand la pondération est positive.

Quand les poids des arcs sont des probabilités, le poids d'un chemin est obtenu en multipliant les poids des arcs du chemin : le poids d'un chemin est alors une probabilité. L'algorithme de Dijkstra peut être utilisé pour rechercher des chemins de probabilité maximum. En effet, tout circuit a un poids inférieur à 1 et ne peut être absorbant.

25 / 25