# Graphes pondérés

Benjamin Monmege

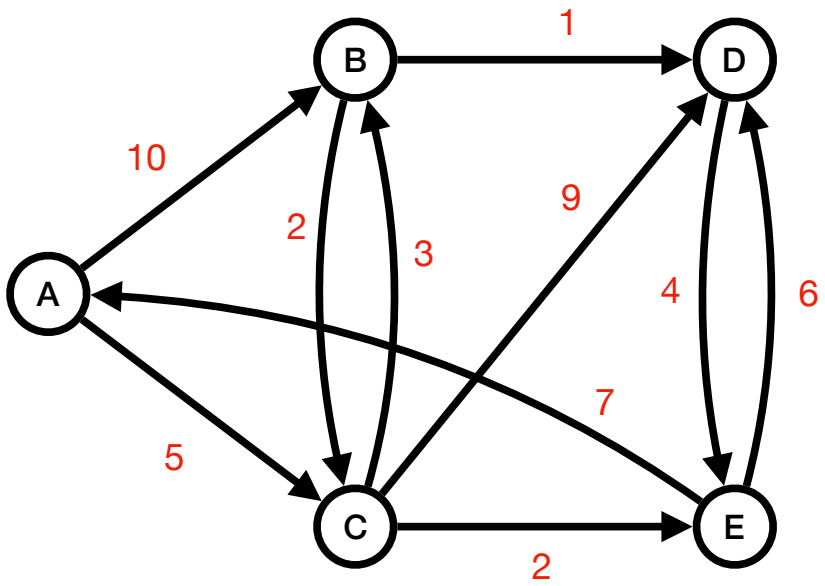
# Graphes pondérés

L'algorithme de parcours en largeur garantit d'obtenir un *plus court chemin* en terme du nombre d'arcs visités...

# Graphes pondérés

L'algorithme de parcours en largeur garantit d'obtenir un *plus court chemin* en terme du nombre d'arcs visités...

... mais vis-à-vis d'un problème de recherche d'itinéraire dans une ville, on peut vouloir mesurer une distance plus finement qu'en fonction du nombre d'arcs, par des *poids*:



# Représentation des graphes pondérés

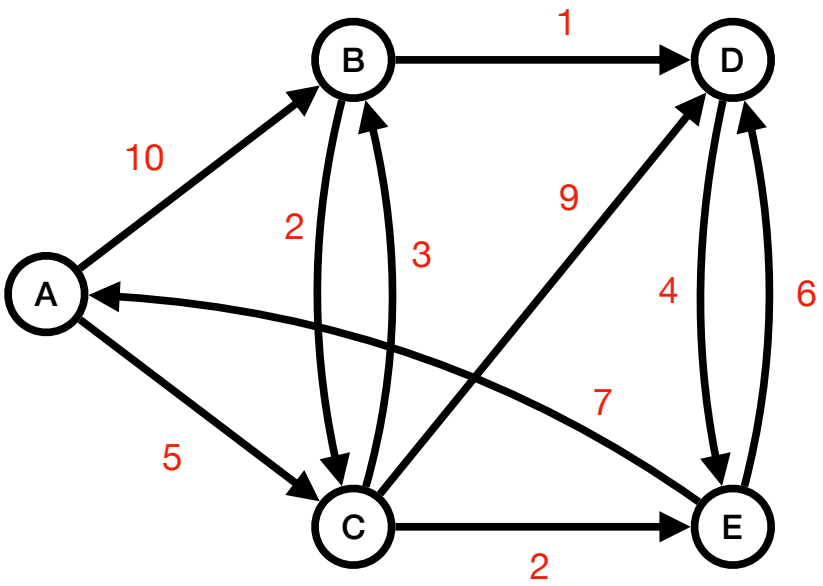
- Matrice d'adjacence où la case $(u, v)$ abrite le poids de l'arc de $u$ à $v$, ou $\infty$ s'il n'y a pas de tel arc
- Liste de successeurs, où chaque successeur $v$ de la liste $u$ est étiqueté par le poids de l'arc de $u$ à $v$ (dictionnaire de listes de paires)

# Représentation des graphes pondérés

- Matrice d'adjacence où la case $(u, v)$ abrite le poids de l'arc de $u$ à $v$, ou $\infty$ s'il n'y a pas de tel arc
- Liste de successeurs, où chaque successeur $v$ de la liste $u$ est étiqueté par le poids de l'arc de $u$ à $v$ (dictionnaire de listes de paires)
- Longueur d'un plus court chemin de $u$ à $v$ :

$$\delta(u, v) = \begin{cases} \min\{\mathit{poids}(c) \mid c \text{ chemin de } u \text{ à } v\} & \text{s'il existe un chemin de } u \text{ à } v \\ \infty & \text{sinon} \end{cases}$$

# L'algorithme de parcours en largeur ne suffit plus à calculer des plus courts chemins !




Exemple au tableau. . .

# Différents algorithmes de plus courts chemins

- **Algorithme de Dijkstra** : plus courts chemins à origine unique dans des graphes à poids positifs
- **Algorithme de Bellman-Ford** : plus courts chemins à origine unique
- **Algorithme de Floyd-Warshall** : plus courts chemins pour tous couples de sommets (avec détection des cycles de poids strictement négatif)

# Algorithmes de Dijkstra à partir d'une source $s$

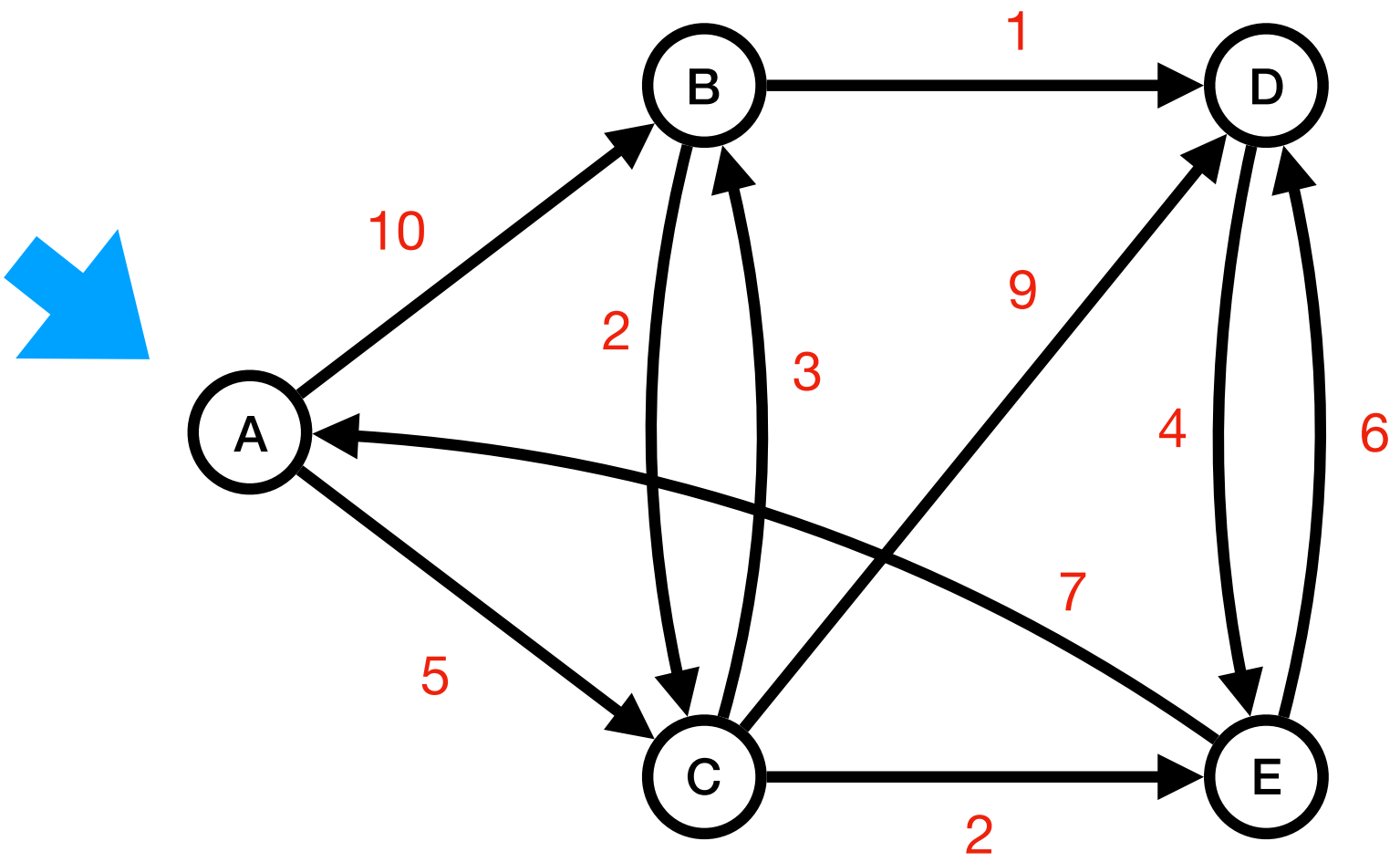
- Maintenir un attribut $d(u)$ pour tout sommet $u$, qui est un majorant de la longueur d'un plus court chemin de $s$ à $u$
- Maintenir un attribut $\pi(u)$ pour tout sommet $u$, qui permet de représenter une arborescence (de plus courts chemins) de racine $s$

# Algorithmes de Dijkstra à partir d'une source $s$

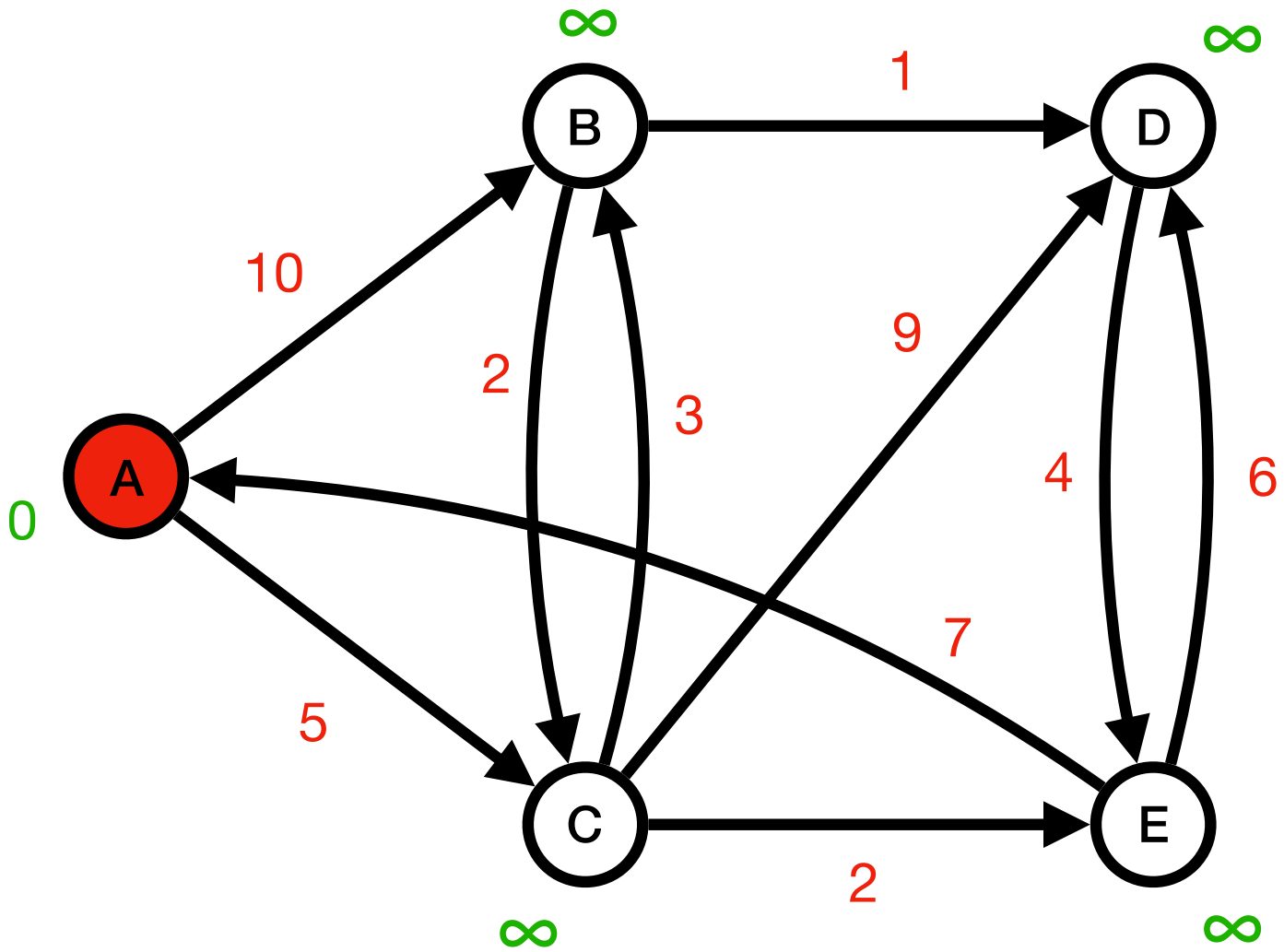
- Maintenir un attribut $d(u)$ pour tout sommet $u$, qui est un majorant de la longueur d'un plus court chemin de $s$ à $u$
- Maintenir un attribut $\pi(u)$ pour tout sommet $u$, qui permet de représenter une arborescence (de plus courts chemins) de racine $s$
- Appliquer une procédure de *relâchement*:

```
fonction relâcher(u, v, poids_arc):
    si d(v) > d(u) + poids_arc(u,v)
    alors d(v) = d(u) + poids_arc(u,v); pi(v) = u
```

# Algorithme de Dijkstra

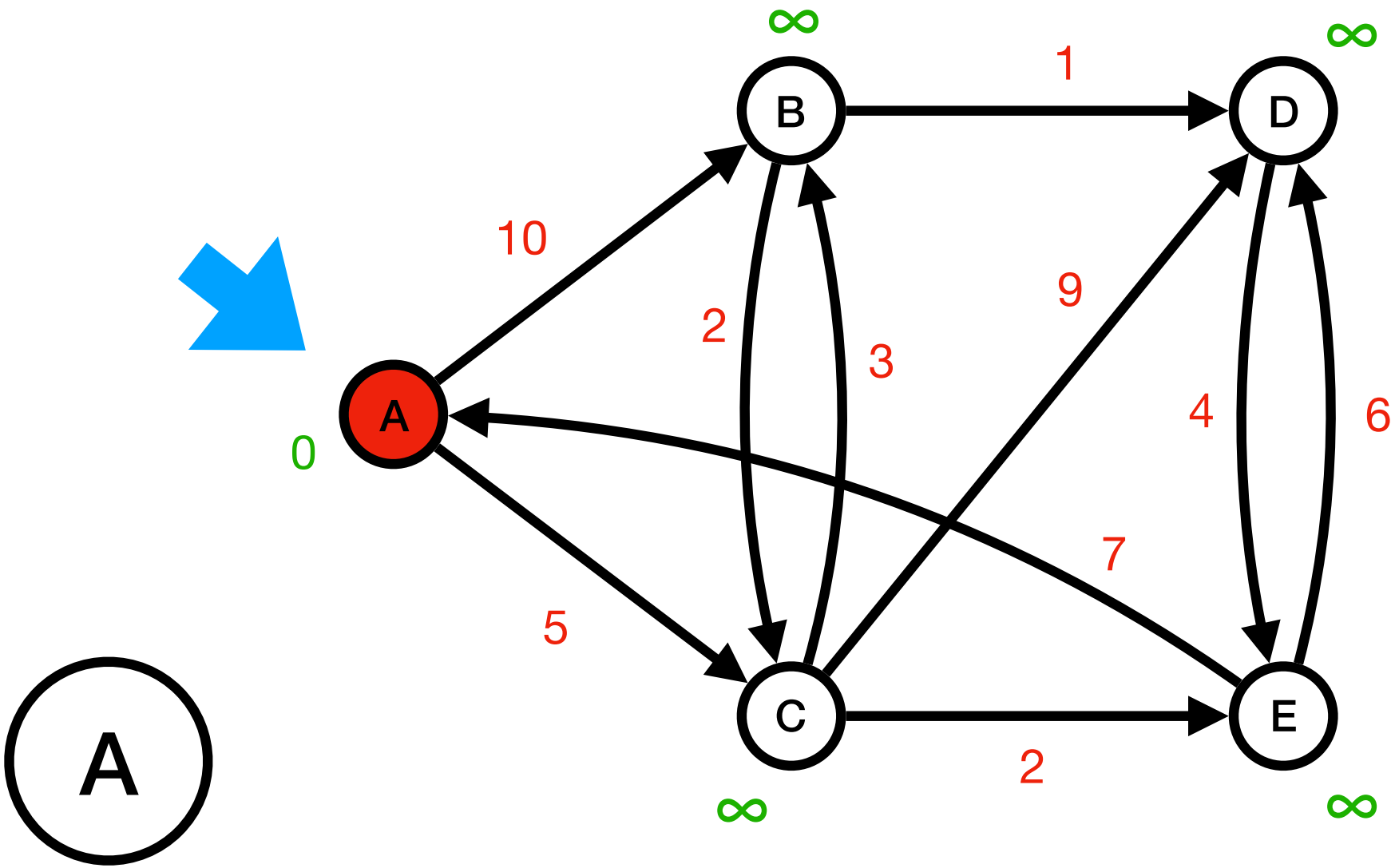


# Algorithme de Dijkstra



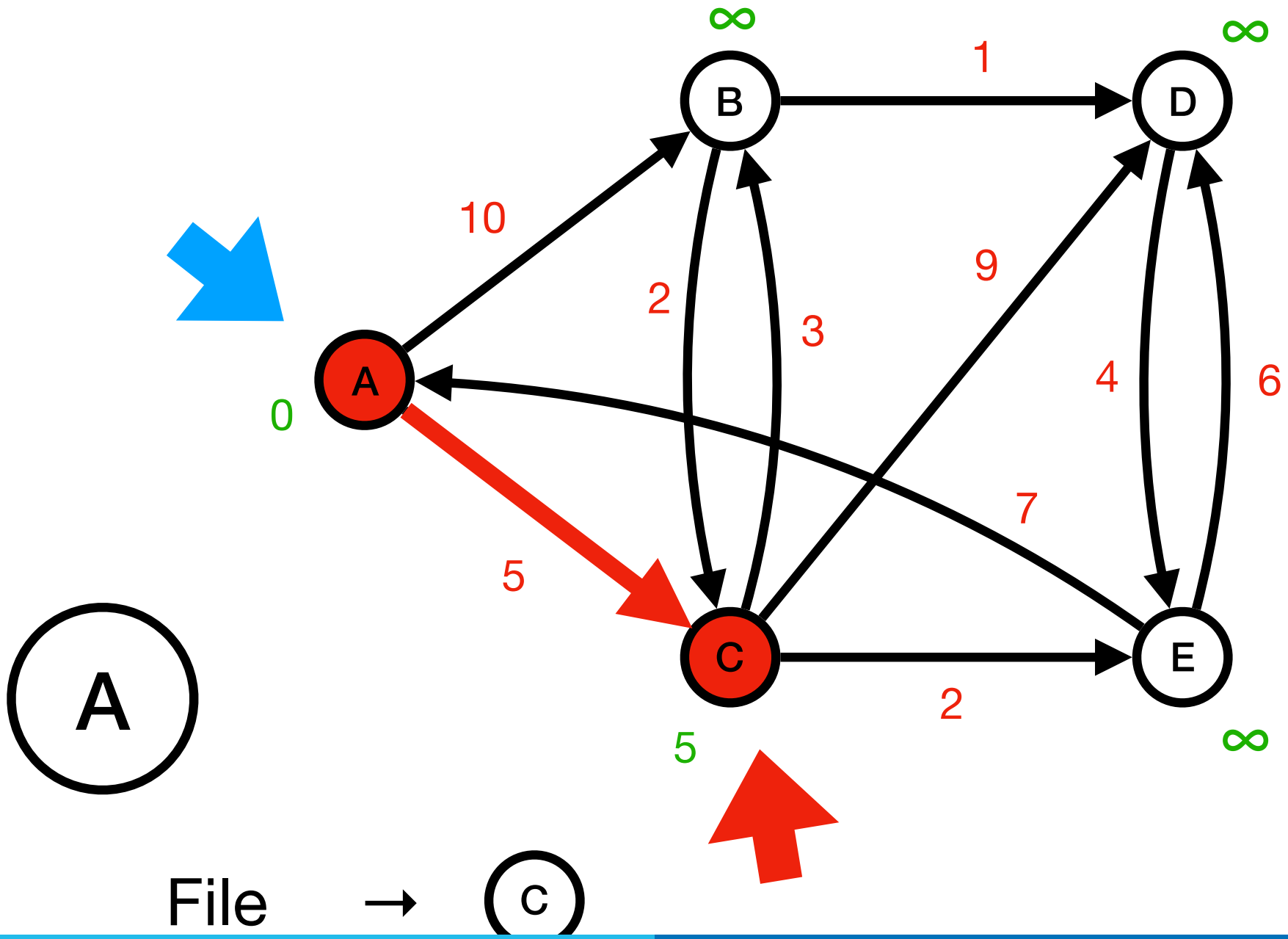

File → A

# Algorithme de Dijkstra

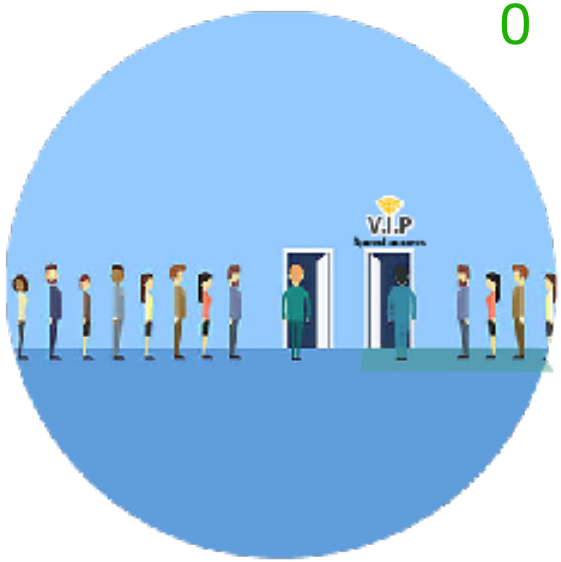


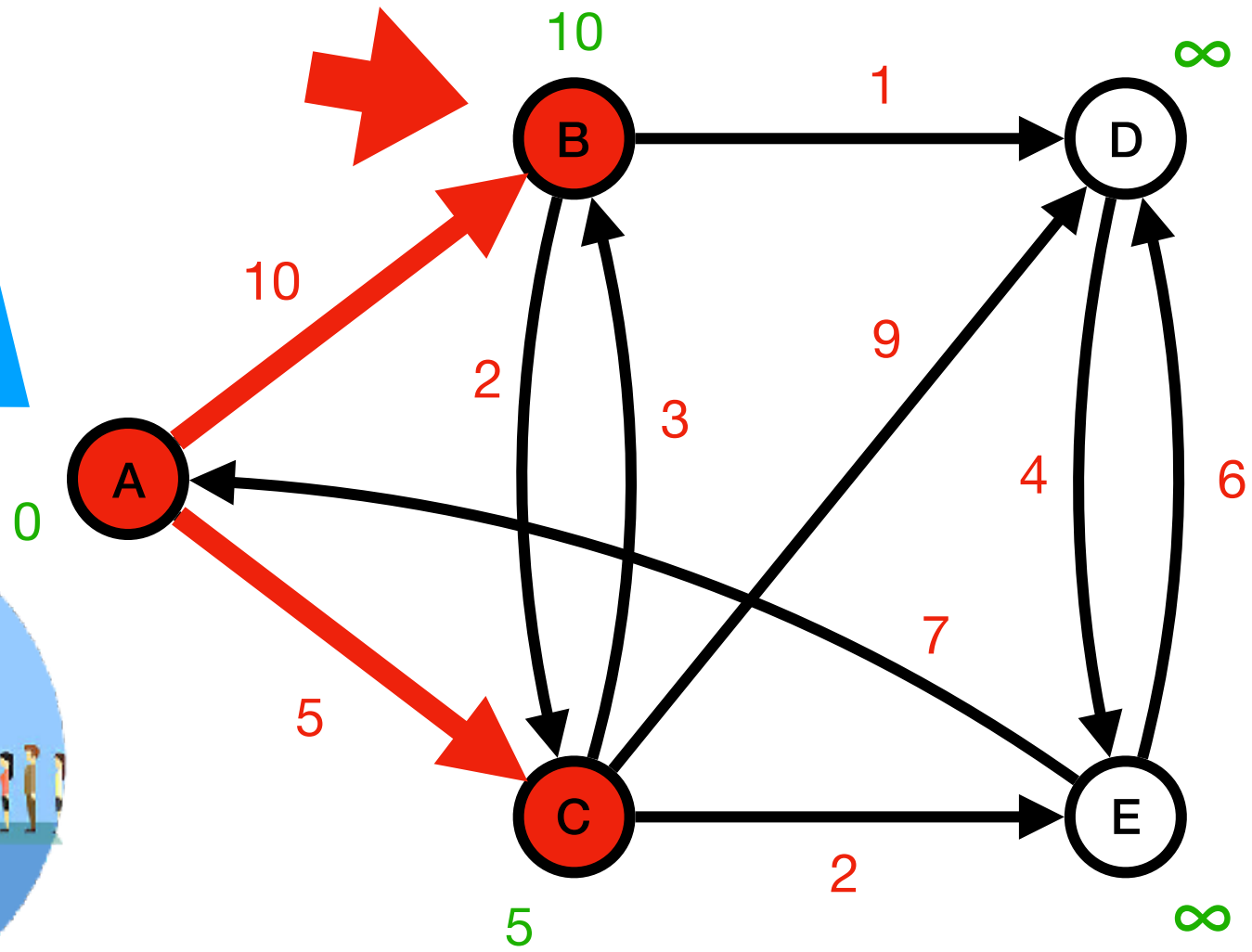

File →

# Algorithme de Dijkstra



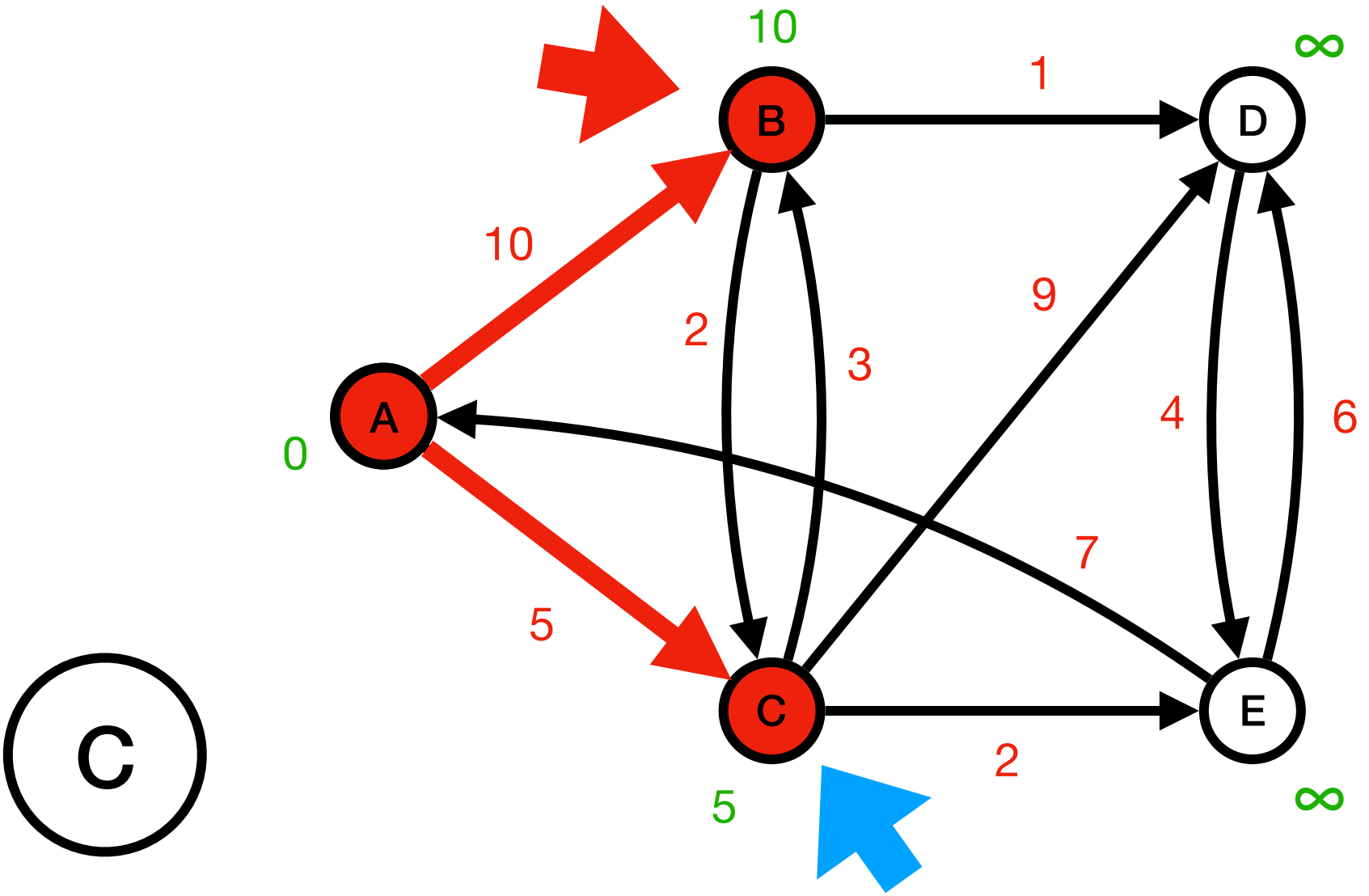
# Algorithme de Dijkstra



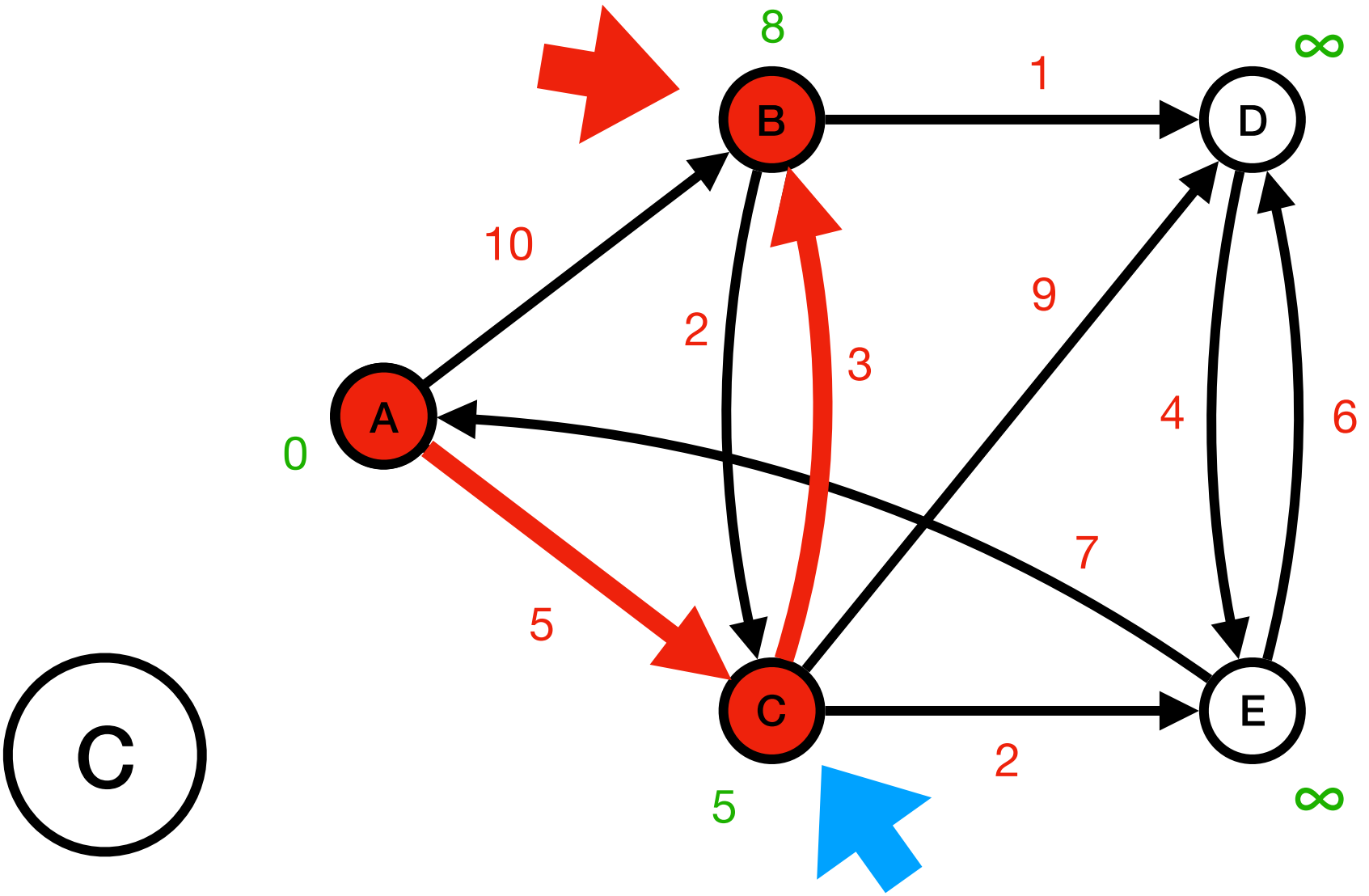




File de priorité → C B

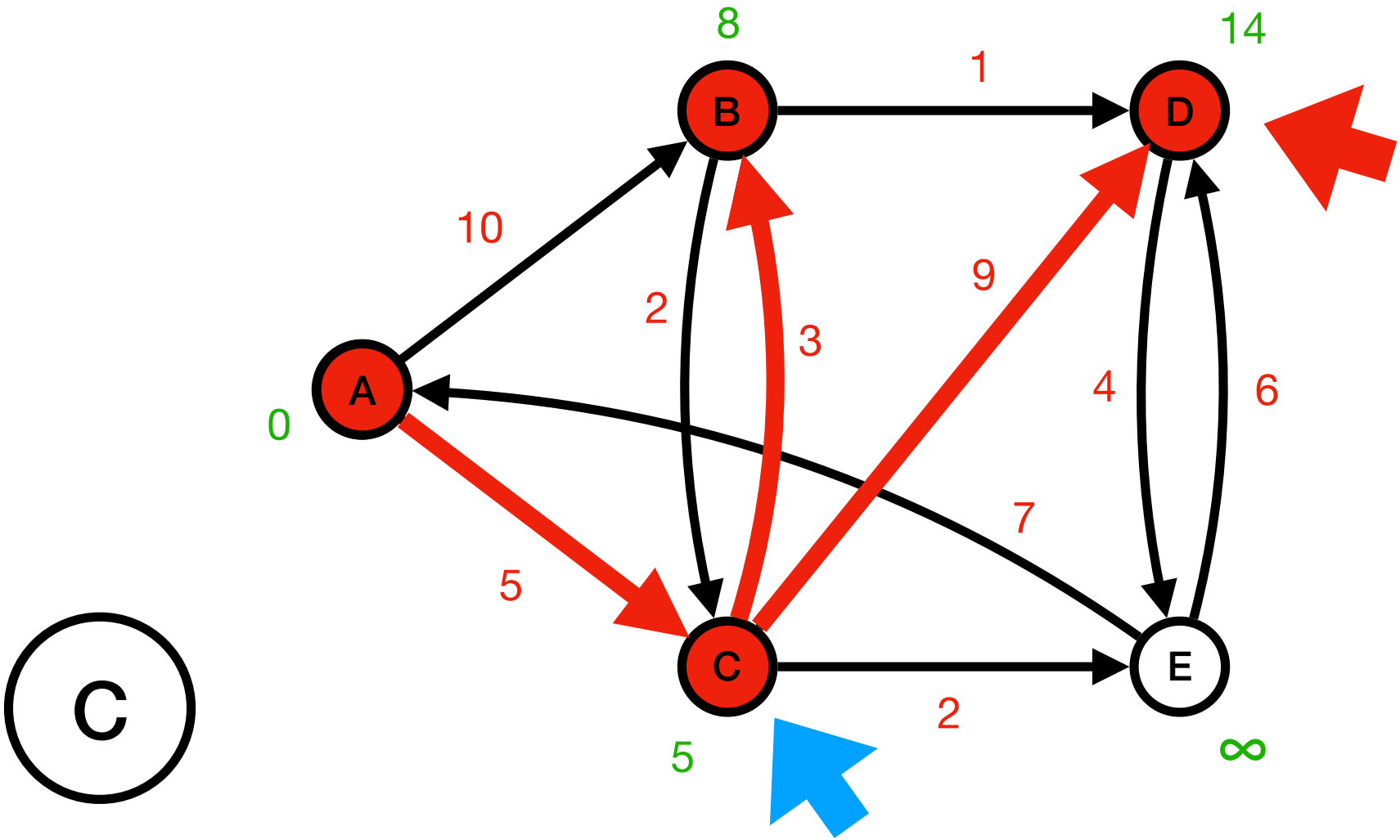
# Algorithme de Dijkstra



# Algorithme de Dijkstra



# Algorithme de Dijkstra

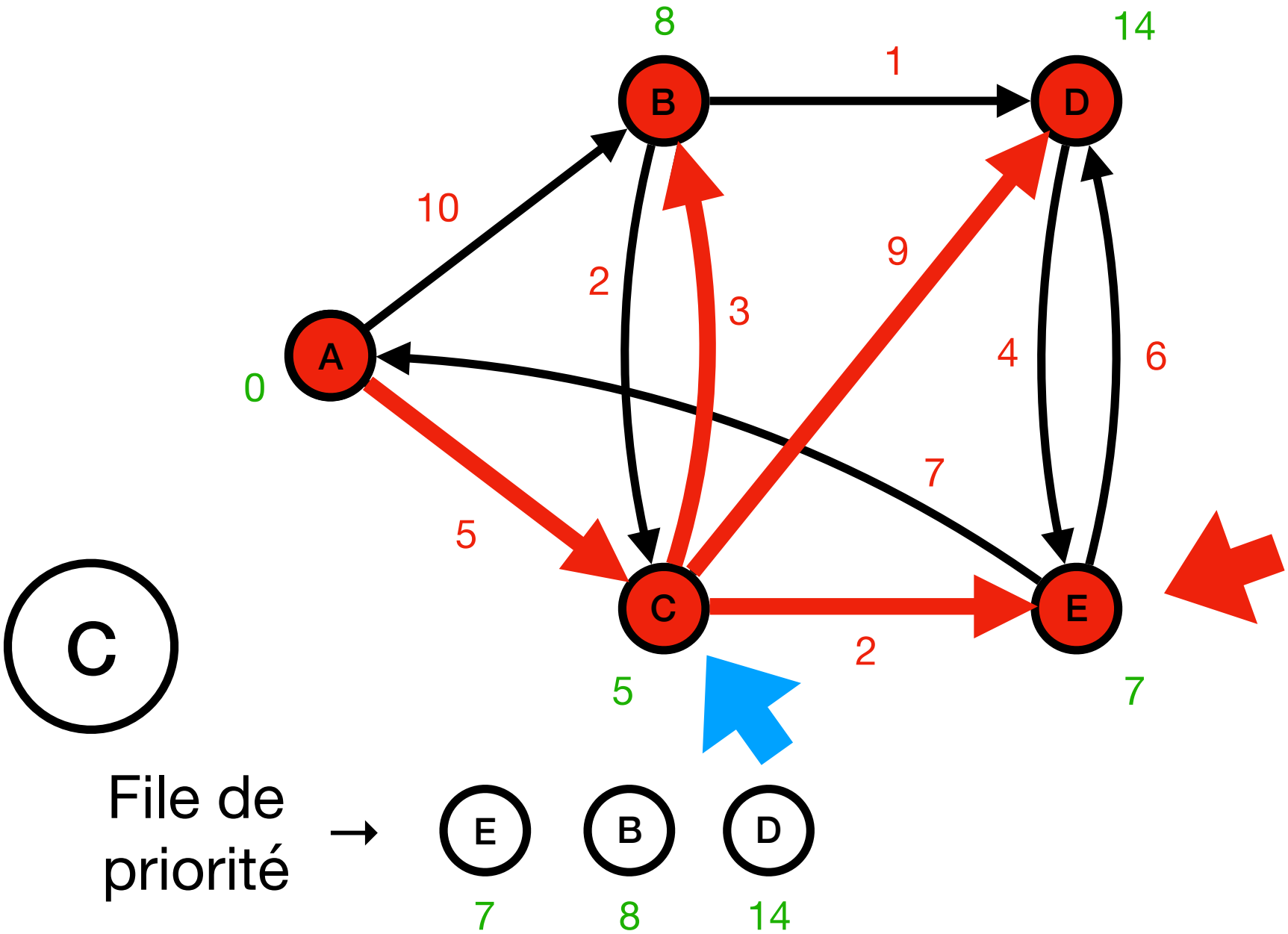



C

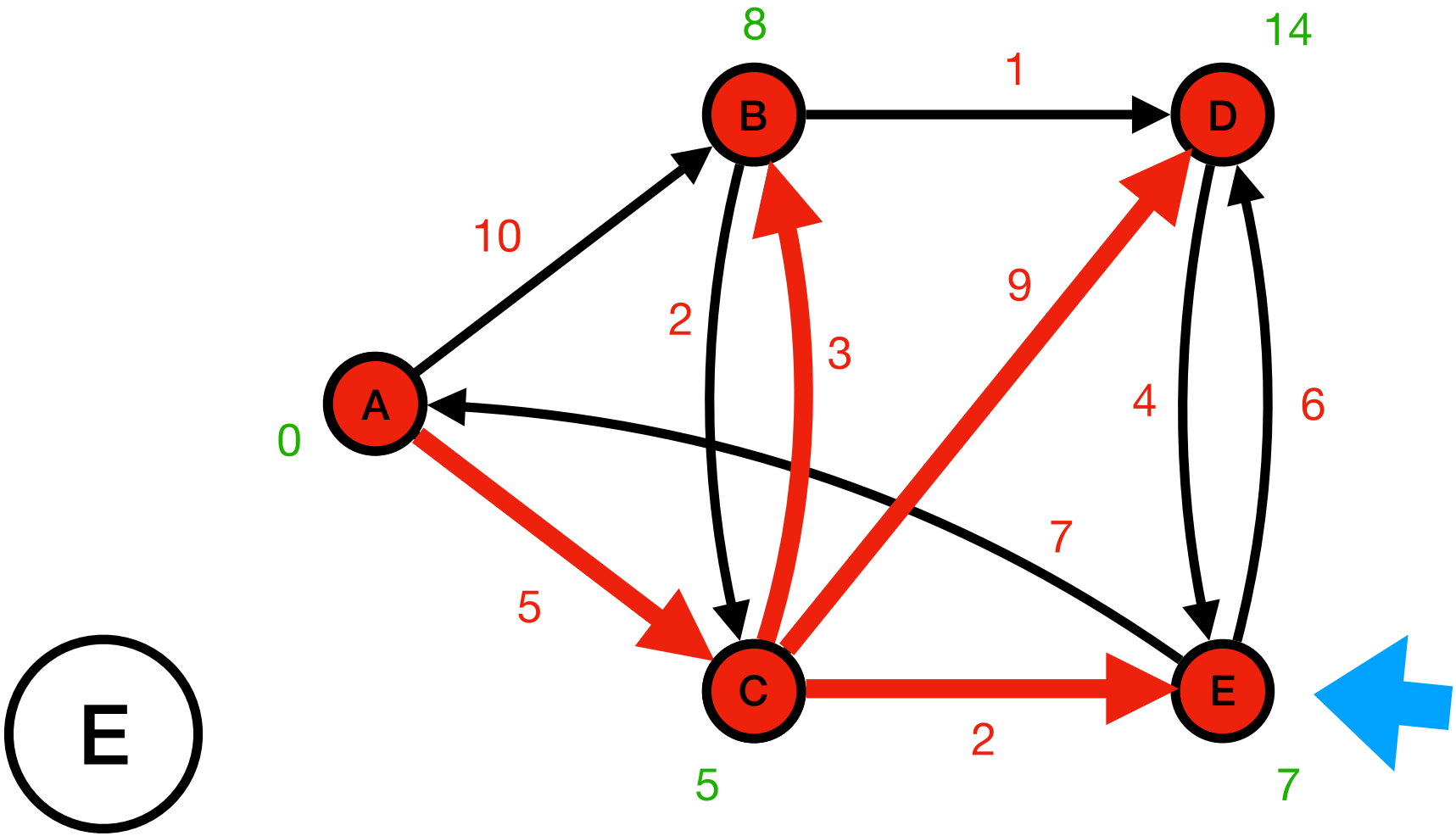
File de priorité → B D

8 14

# Algorithme de Dijkstra



# Algorithme de Dijkstra

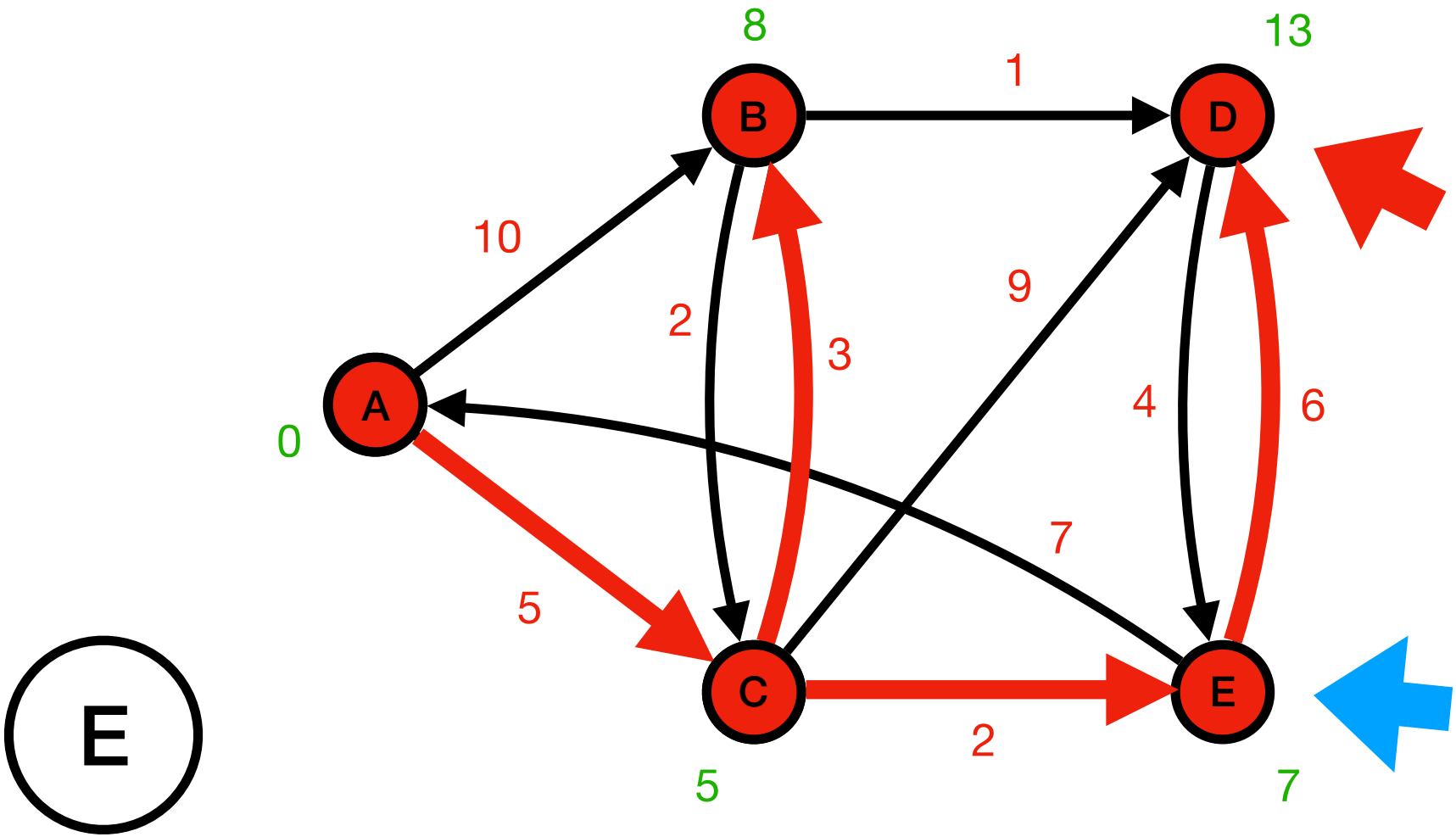



E

File de priorité → B D

8 14

# Algorithme de Dijkstra

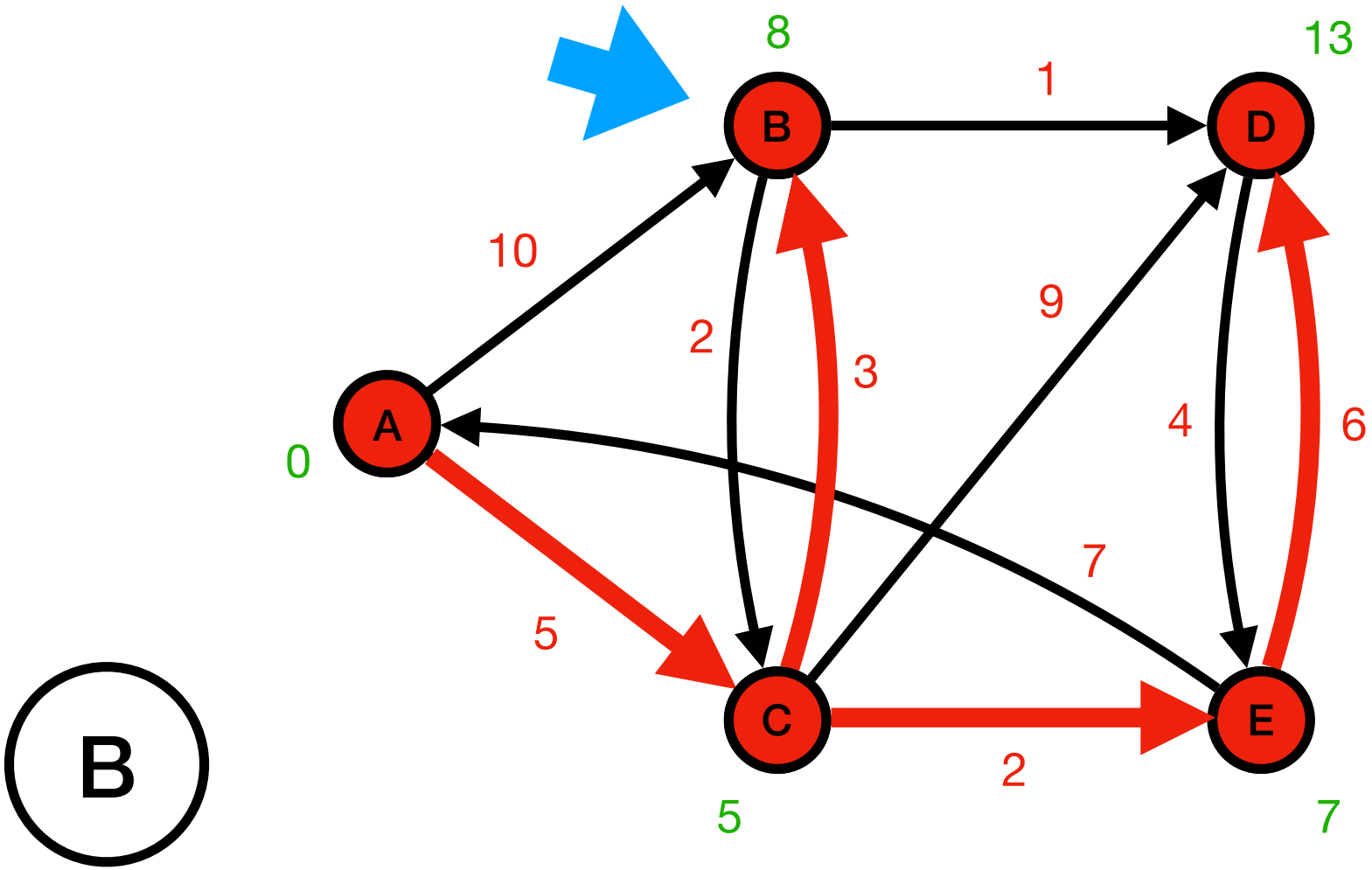



E

File de priorité → B D

8 13

# Algorithme de Dijkstra



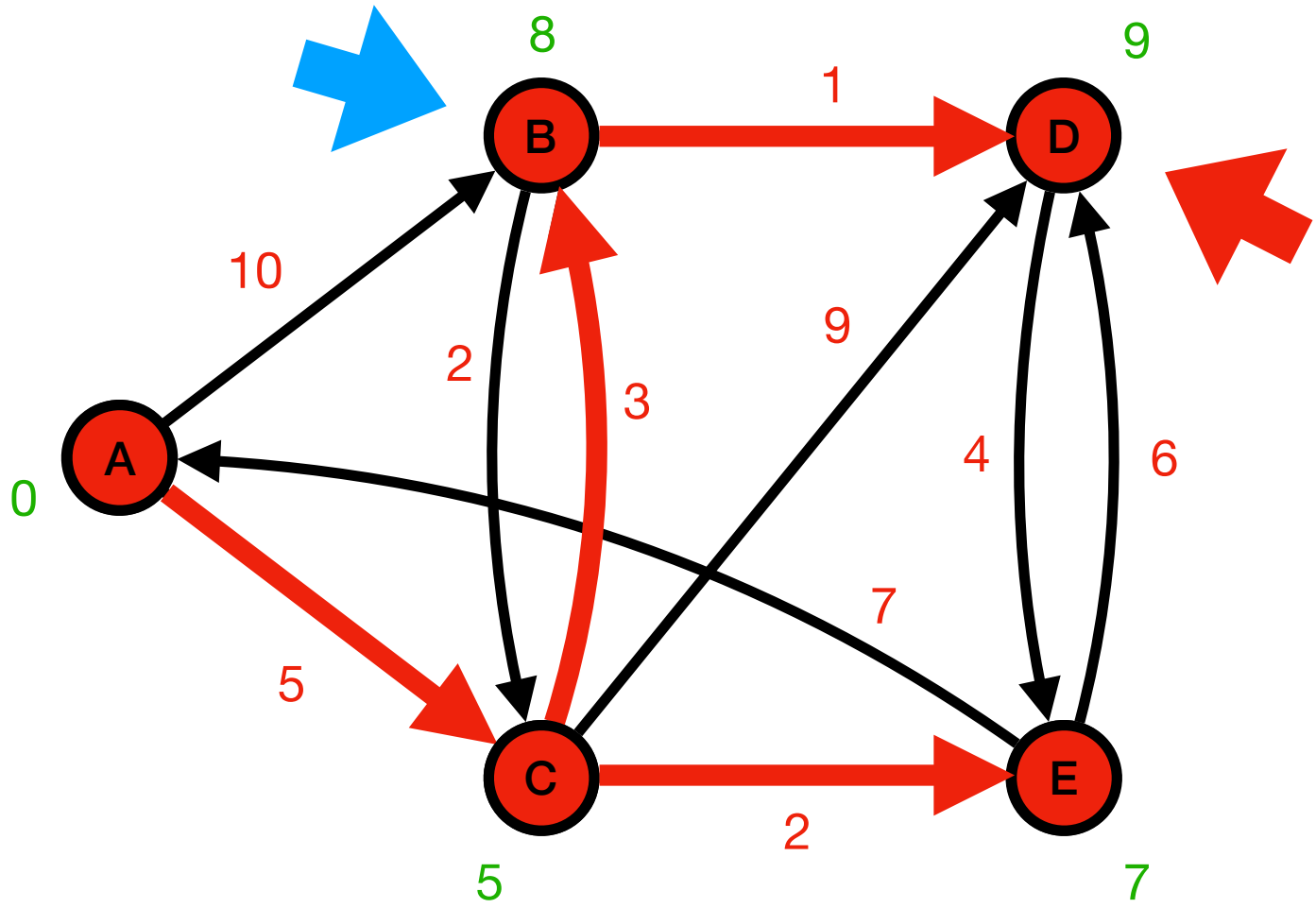


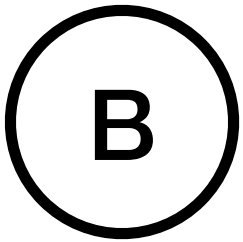


File de priorité → 


13

# Algorithme de Dijkstra

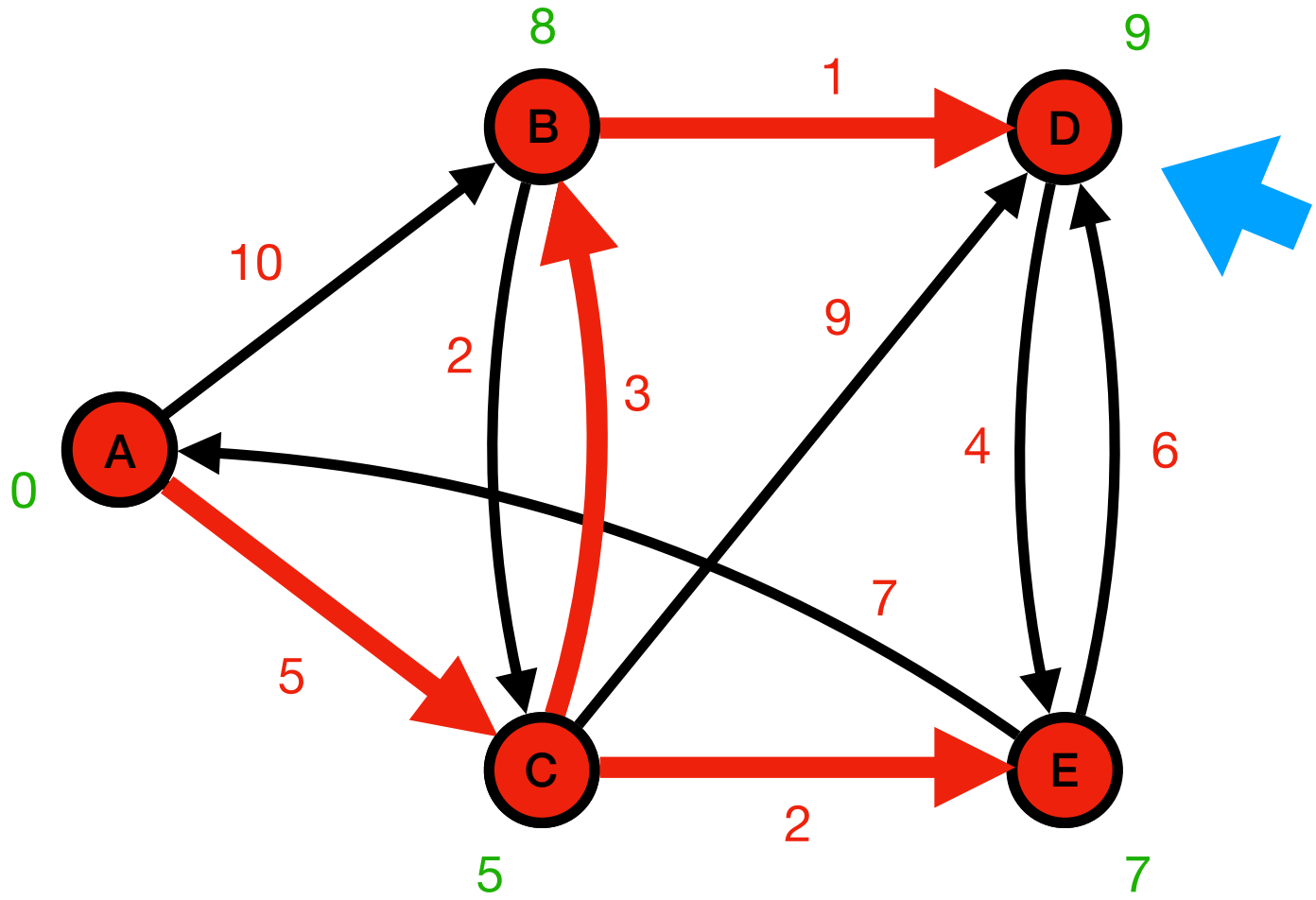


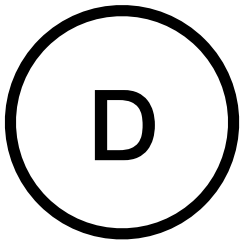




File de priorité → 


9

# Algorithme de Dijkstra




File de priorité →