

Drumuri minime în grafuri ponderate



Aplicații



- **Data o hartă rutieră, să se determine**
 - un drum minim între două orașe date
 - câte un drum minim între oricare două orașe de pe hartă

Aplicații

- **Determinarea de drumuri minime / distanțe - numeroase aplicații**
 - probleme de rutare
 - robotică
 - procesarea imaginilor
 - strategii jocuri
 - probleme de planificare (drumuri critice)

Cadru

Fie:

- **G** - un graf **orientat** ponderat
- **P** - un drum

$$w(P) = \sum_{e \in E(P)} w(e)$$

costul / ponderea / lungimea drumului P

Cadru

Fie:

- ☐ **G** - un graf **orientat** ponderat
- ☐ **Presupunem că G nu conține circuite de pondere negativă**

Cadru

- Fie $s, v \in V, s \neq v$
- Distanța de la s la v
 - $\delta_G(s, v) = \min \{ w(P) \mid P \text{ este drum de la } s \text{ la } v \}$

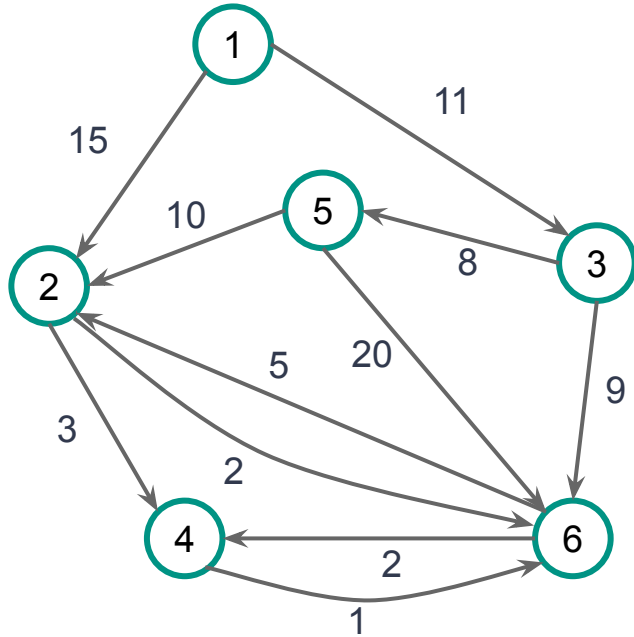
Cadru

- Fie $s, v \in V, s \neq v$
- Distanța de la s la v
 - $\delta_G(s, v) = \min \{ w(P) \mid P \text{ este drum de la } s \text{ la } v \}$
 - $\delta_G(s, s) = 0$

Convenție: $\min \emptyset = \infty$

- Un drum P de la s la v se numește drum minim de la s la v dacă $w(P) = \delta_G(s, v)$

Exemplu



Aplicații

Tipuri de probleme de drumuri minime

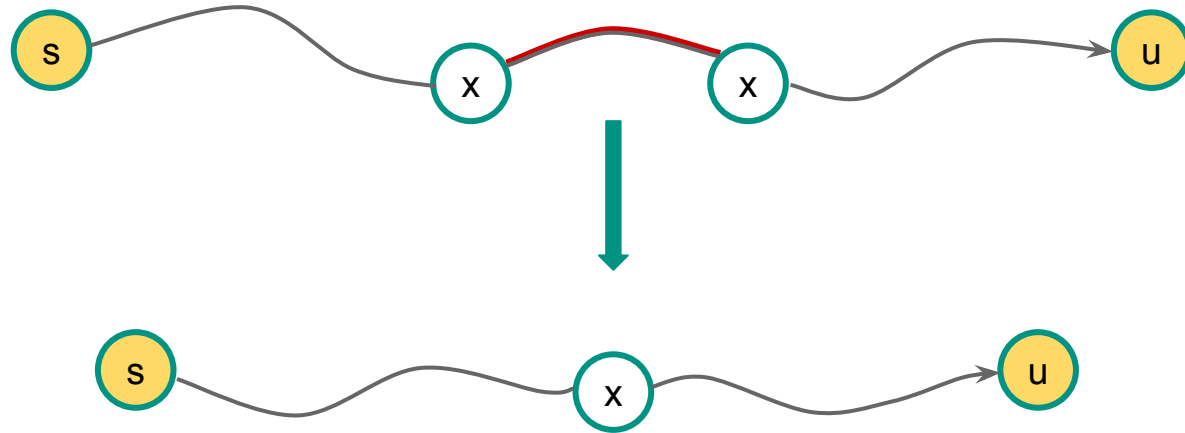
- ☐ între două vârfuri date
- ☐ de la un vârf la toate celelalte
- ☐ între oricare două vârfuri

Situații

- ☐ Sunt acceptate și arce de cost negativ?
- ☐ Graful conține circuite?
- ☐ Graful conține circuite de cost negativ?

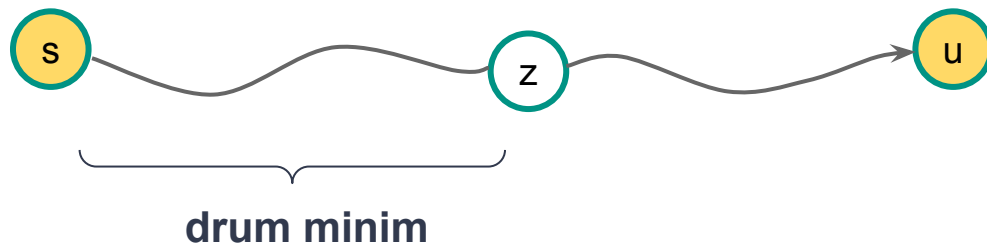
Observații

Observația 1. Dacă P este un drum minim de la s la v și **nu există circuite de cost negativ**, atunci P este drum elementar.

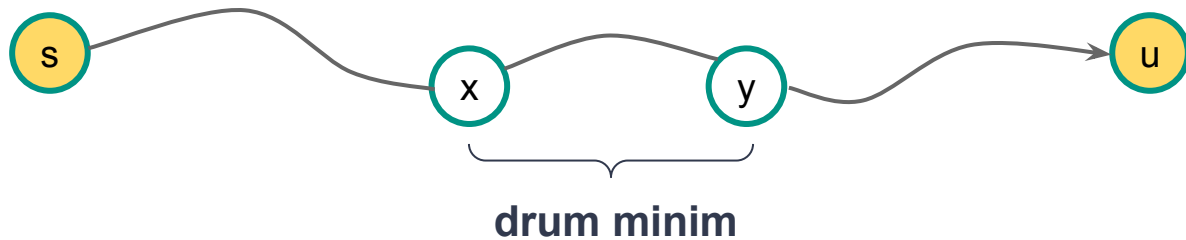


Observații


Observația 2. Dacă P este un drum minim de la s la u și z este un vârf al lui P , atunci subdrumul lui P de la s la z este drum minim de la s la z .



Observații



Drumuri minime de la un
vârf s dat la celelalte vârfuri
(de sursă unică)



Problemă

Problema drumurilor minime de sursă unică (de la s la celelalte vârfuri)

Se dau:

□ un graf **orientat** ponderat $G = (V, E, w)$ cu

$$w : E \rightarrow \mathbb{R}$$

□ un vârf de start **s**

Să se determine distanța de la s la fiecare vârf x al lui G / la un vârf dat t (și un arbore al distanțelor față de s / un drum minim de la s la t).

Drumuri minime de sursă unică s



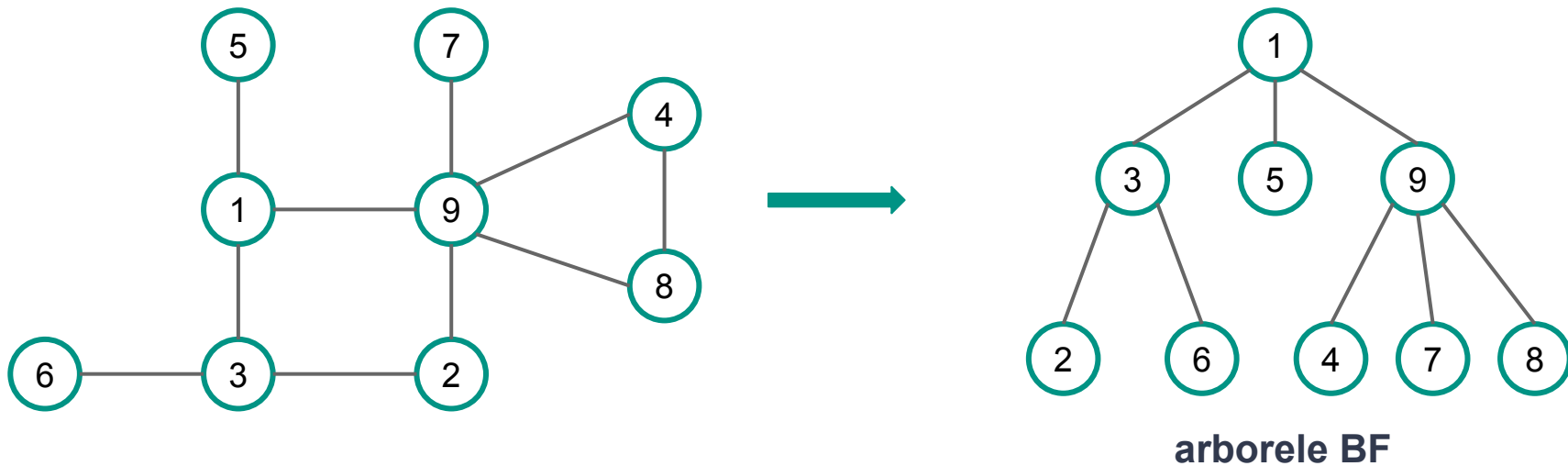
- Dacă G nu este ponderat, cum putem calcula distanțele față de s?

Drumuri minime de sursă unică s

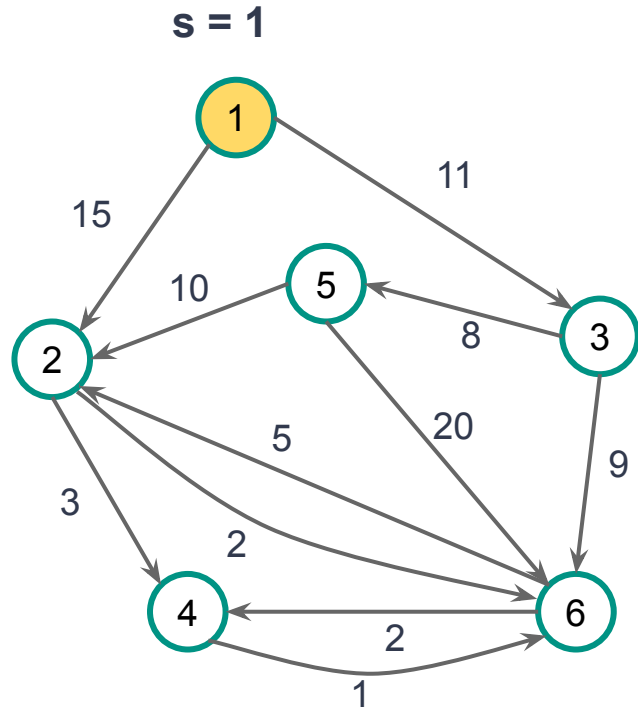
□ Amintim

În cazul unui graf neponderat, problema se rezolvă folosind parcurgerea BF din s

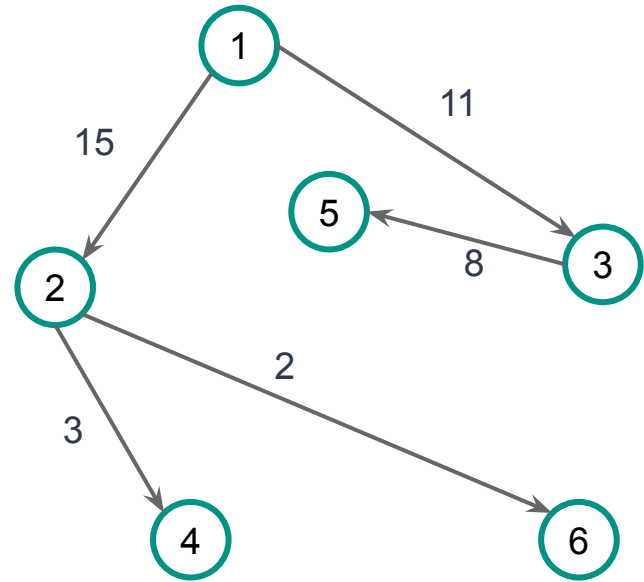
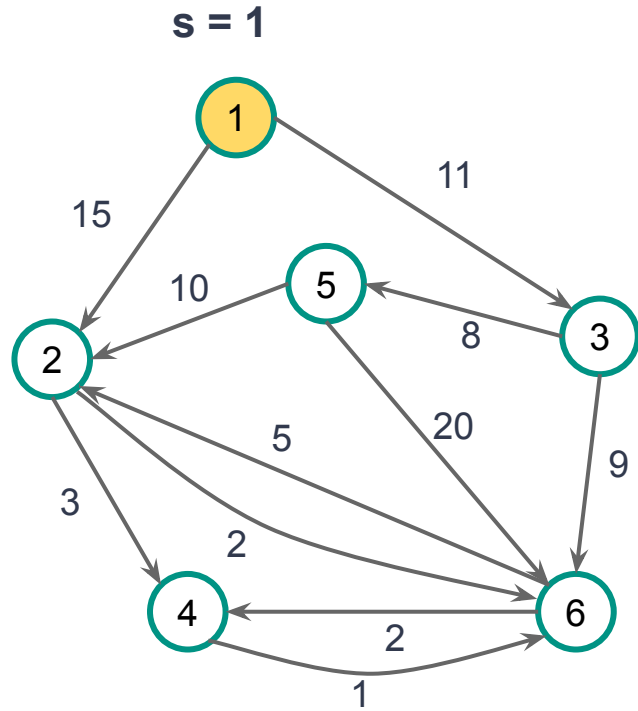
⇒ arborele BF (al distanțelor față de s)



Exemplu



Exemplu



arborele distanțelor față de 1

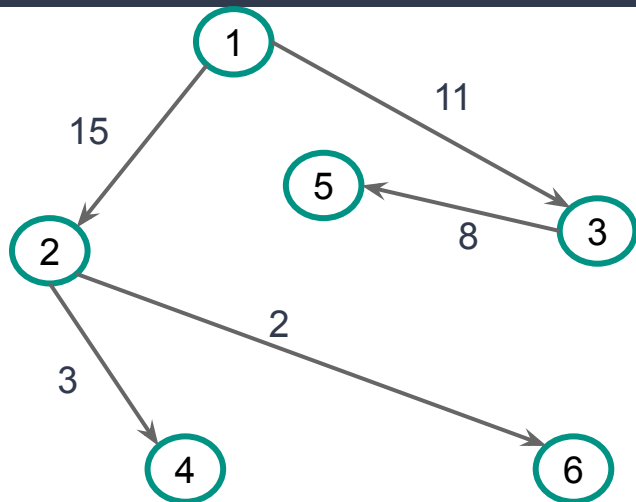
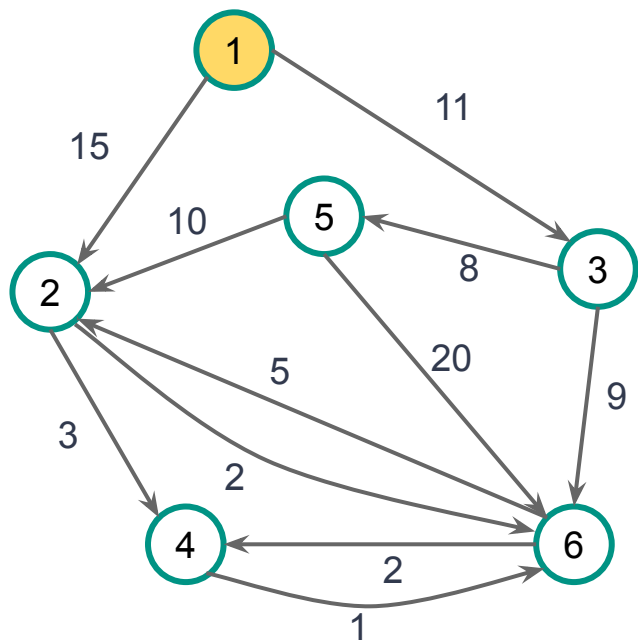
Arborele distanțelor

Definiție. Pentru un vârf dat s , un arbore al distanțelor față de s = un subgraf T al lui G , care conservă distanțele de la s la celelalte vârfuri accesibile din s

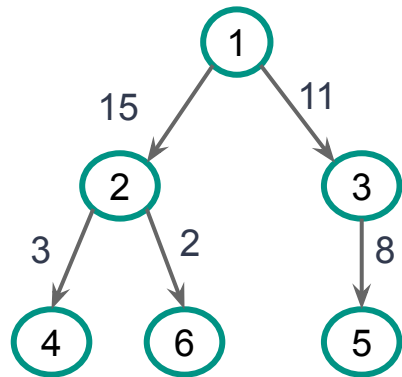
$$\delta_T(s, v) = \delta_G(s, v), \quad \forall v \in V \text{ accesibil din } s,$$

graful neorientat asociat lui T fiind arbore cu rădăcina în s (cu arcele corespunzătoare orientate de la s la frunze).

Arborele distanțelor



**arborele
distanțelor față
de 1**

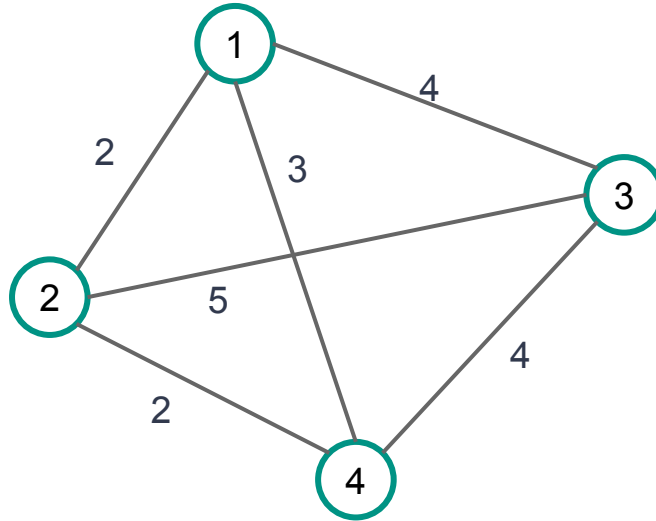


Arborele distanțelor

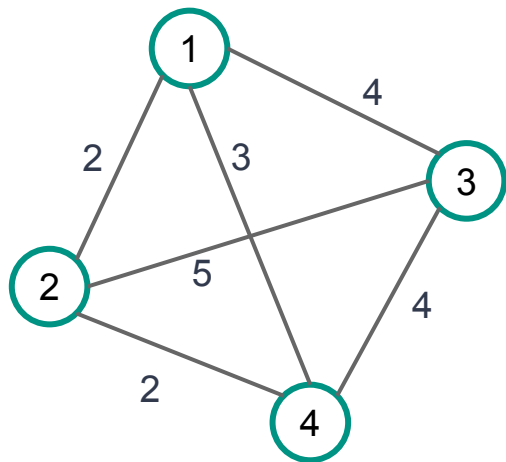
- Presupunem că **toate vârfurile sunt accesibile din s**
- Problema drumurilor minime de sursă unică este echivalentă cu determinarea unui **arbore al distanțelor față de s**

Arborele distanțelor

- Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime



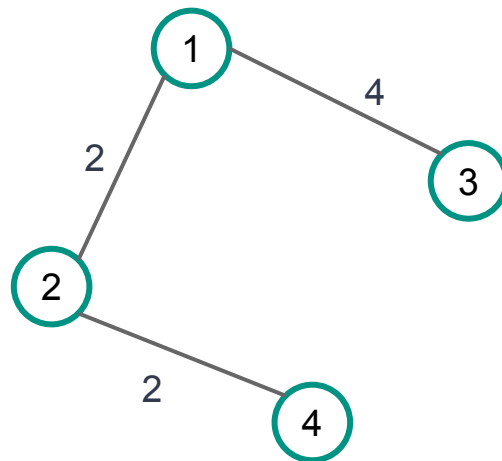
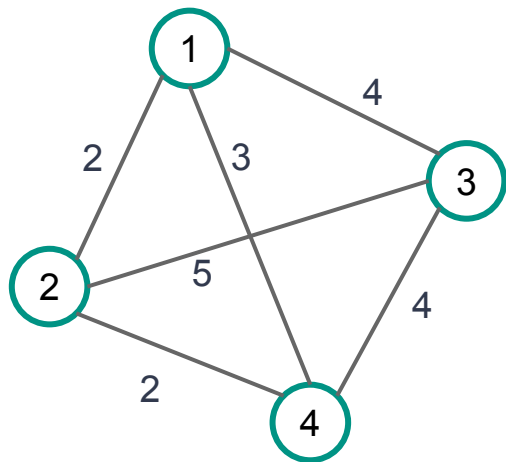
- **Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime**



apcm

arbore al
distanțelor
față de 1

□ **Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime**

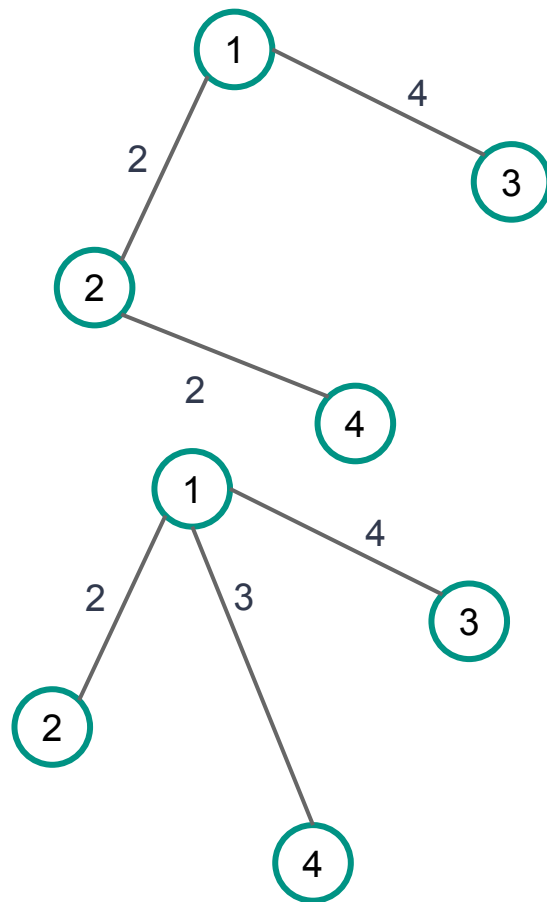
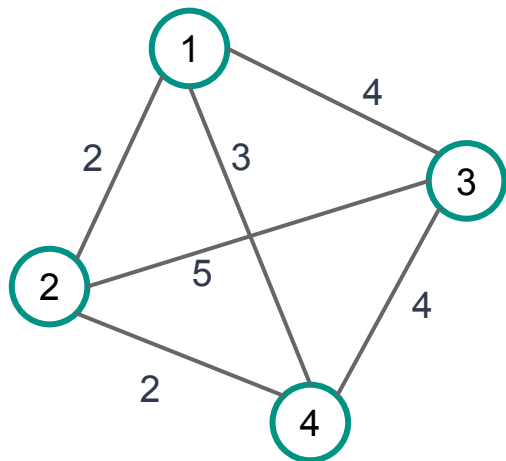


apcm



arbore al
distanțelor
față de 1

□ Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime



apcm

arbore al
distanțelor
față de 1

Drumuri minime de sursă unică s



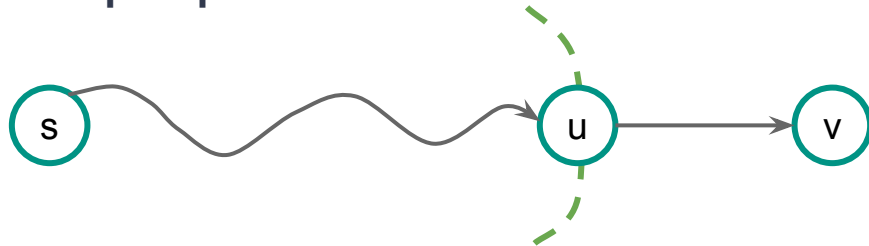
- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

Drumuri minime de sursă unică s



- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

”din aproape în aproape”



Dacă u este predecesor al lui v pe un drum minim de la s la v \Rightarrow

$$\delta(s, v) = \delta(s, u) + w(uv)$$

Știm $\delta(s, u) \Rightarrow$ aflăm și $\delta(s, v)$

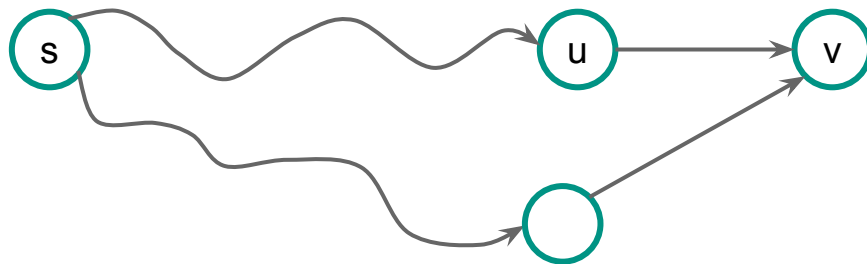
Drumuri minime de sursă unică s



- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

”din aproape în aproape” \Rightarrow când considerăm un vârf v , pentru a calcula $\delta(s, v)$,
ar fi util să știm deja $\delta(s, u)$, pentru u cu $uv \in E$ (?! toate)

$$\delta(s, v) = \min \{ \delta(s, u) + w(u, v) \mid uv \in E \}$$



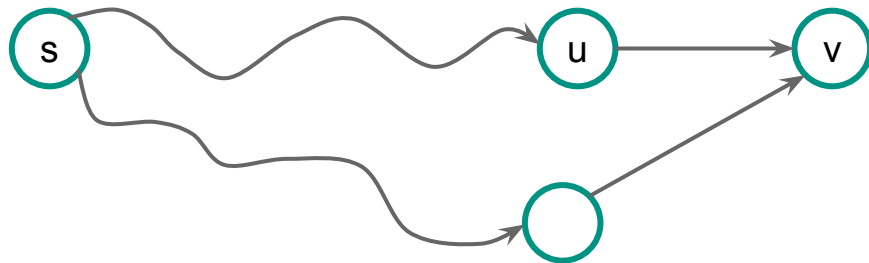
Drumuri minime de sursă unică s

- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

"din aproape în aproape" \Rightarrow când considerăm un vârf v , pentru a calcula $\delta(s, v)$, ar fi util să știm deja $\delta(s, u)$, pentru orice u cu $uv \in E$



Ar fi utilă o ordonare a vârfurilor astfel încât, dacă $uv \in E$, atunci u se află înaintea lui v



Drumuri minime de sursă unică s

- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

”din aproape în aproape” \Rightarrow când considerăm un vârf v , **pentru a calcula $\delta(s, v)$** , ar fi util să știm deja $\delta(s, u)$, **pentru orice u cu $uv \in E$**

Ar fi utilă o ordonare a vârfurilor astfel încât, dacă $uv \in E$, atunci u se află înaintea lui v



O astfel de ordonare nu există dacă graful conține circuite

Drumuri minime de sursă unică s

- În ce ordine considerăm vârfurile pentru a calcula distanțele față de s?

”din aproape în aproape” \Rightarrow când considerăm un vârf v , pentru a calcula $\delta(s, v)$, ar fi util să știm deja $\delta(s, u)$, pentru orice u cu $uv \in E$

Ar fi utilă o ordonare a vârfurilor astfel încât, dacă $uv \in E$, atunci u se află înaintea lui v

O astfel de ordonare nu există dacă graful conține circuite

Dacă există circuite - estimăm distanțele pe parcursul algoritmului și considerăm vârful care este **estimat** a fi cel mai aproape de s



Drumuri minime de sursă unică s

- Algoritmi pentru grafuri orientate cu circuite, dar cu ponderi pozitive - **Dijkstra**
- Algoritmi pentru grafuri orientate fără circuite (cu ponderi reale) DAGs = Directed Acyclic Graphs
- Algoritmi pentru grafuri orientate cu circuite și ponderi reale, care detectează existența de circuite negative - **Bellman-Ford**

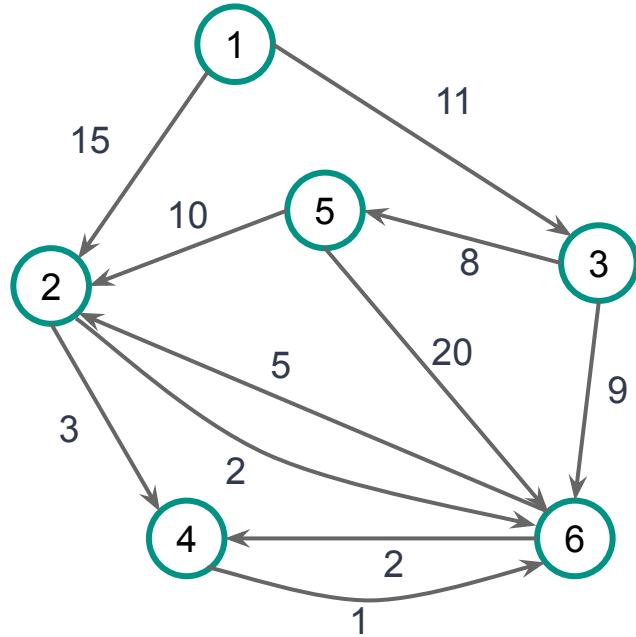
Algoritmul lui Dijkstra

Algoritmul lui Dijkstra

Ipoteză:

Presupunem că arcele au cost pozitiv (graful poate conține circuite)

Exemplu



Algoritmul lui Dijkstra

Idee: La un pas, este ales ca vârf curent (vizitat) vârful u care este estimat a fi cel mai apropiat de s

- Estimarea pentru u = cel mai scurt drum de la s la u determinat până la pasul curent



Algoritmul lui Dijkstra

Idee: La un pas, este ales ca vârf curent (vizitat) vârful u care este estimat a fi cel mai apropiat de s

- **Estimarea pentru u** = cel mai scurt drum de la s la u determinat până la pasul curent
- + se descoperă noi drumuri către vecinii lui \Rightarrow **se actualizează distanțele estimate pentru vecini**



Algoritmul lui Dijkstra

- **Generalizare a ideii de parcurgere BF**
- **Dacă toate arcele au cost egal \Rightarrow Dijkstra \equiv BF**

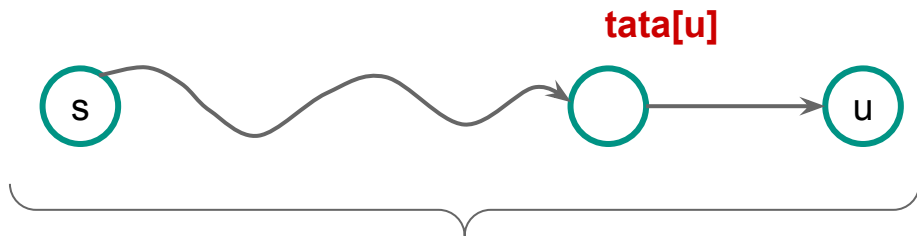
Pseudocod

A dark blue, diagonal shape that starts from the bottom-left corner and extends towards the top-right, covering the lower half of the slide. It has a smooth, slightly curved edge.

Algoritmul lui Dijkstra

Pentru fiecare vârf, reținem etichetele:

- $d[u]$ - etichetă de distanță
- $tata[u]$



$d[u]$ = costul minim al unui drum de la s la u descoperit până la acel moment

$tata[u]$ = **predecesorul** lui u pe drumul de cost minim de la s la u descoperit până la acel moment

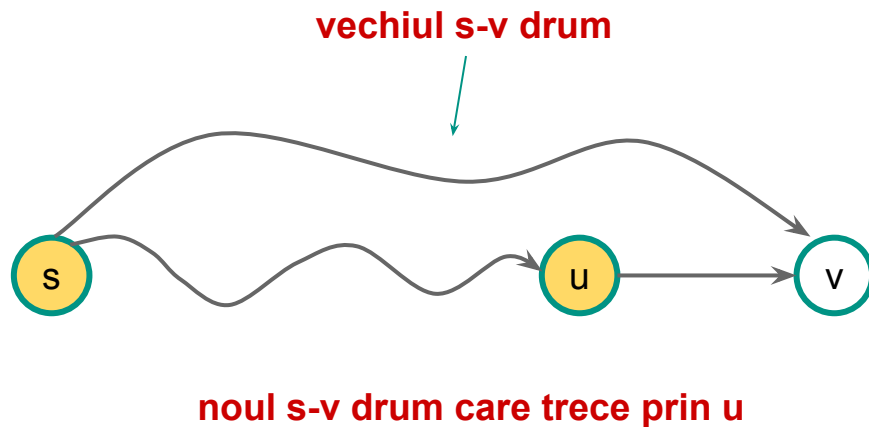
Algoritmul lui Dijkstra

La un pas:

- este selectat un vârf u (neselectat) care “**pare**” cel mai apropiat de $s \Leftrightarrow$ are eticheta d minimă
- se actualizează etichetele $d[v]$ ale vecinilor lui u - considerând drumuri care trec prin u
 - **tehnică de relaxare a arcelor care ies din u**

Algoritmul lui Dijkstra

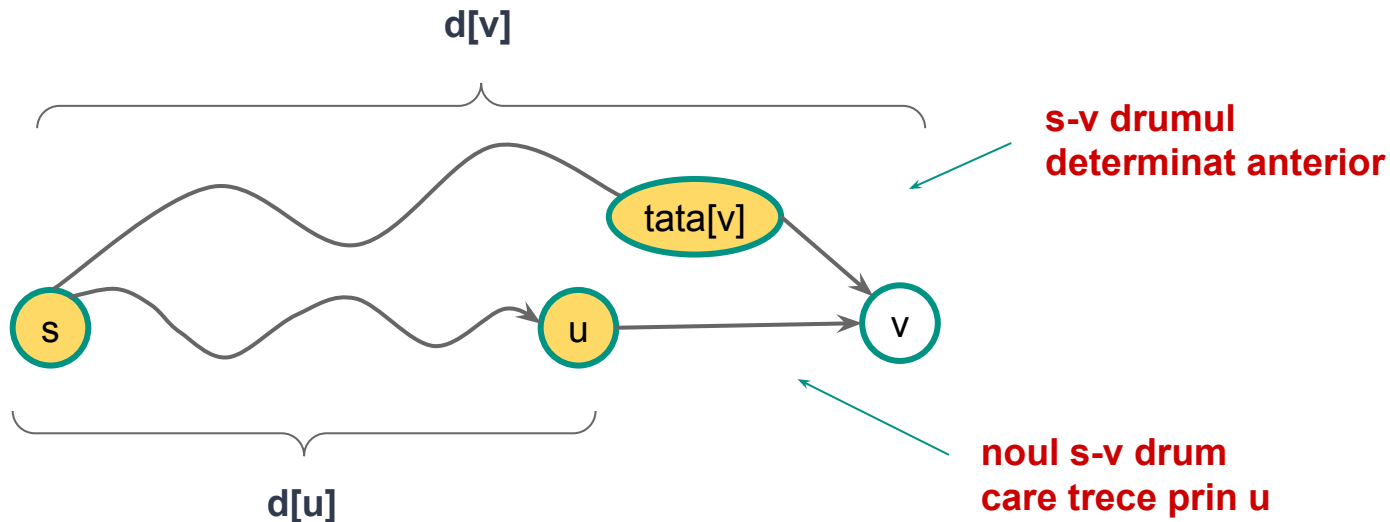
Relaxarea unui arc (u, v) = verificarea dacă $d[v]$ poate fi îmbunătățit, trecând prin vârful u



Algoritmul lui Dijkstra

Relaxarea unui arc (u, v)

dacă $d[u] + w(u, v) < d[v]$ atunci
 $d[v] = d[u] + w(u, v)$
 $tata[v] = u$

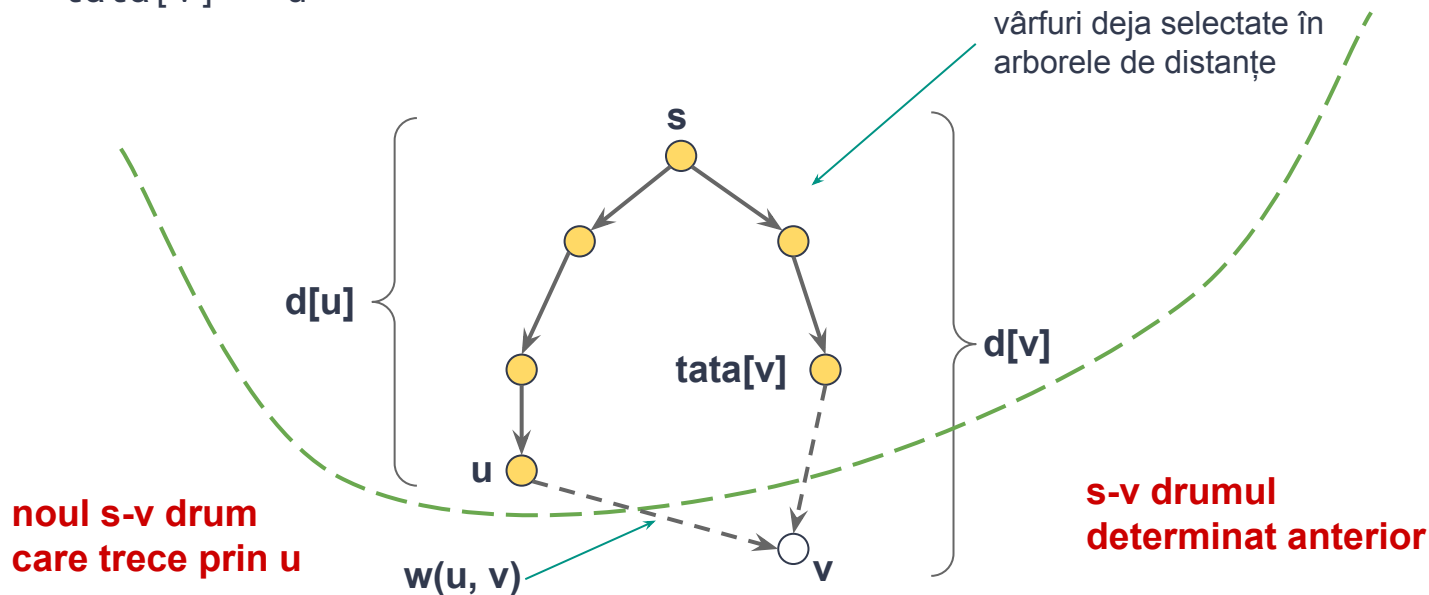


Algoritmul lui Dijkstra

Relaxarea unui arc (u, v)

Raportat la vârfuri deja selectate - similar Prim

dacă $d[u] + w(u, v) < d[v]$ atunci
 $d[v] = d[u] + w(u, v)$
 $tata[v] = u$



Algoritmul lui Dijkstra

Dijkstra(G , w , s)

 inițializează mulțimea vârfurilor neselectate Q cu V

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

 inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = \emptyset$

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = \emptyset$

$d[s] = 0$

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută** // \Leftrightarrow **pentru** $i=1, n$ **execută**

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută**

$u =$ extrage vârf cu eticheta d minimă din Q

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută**

$u =$ extrage vârf cu eticheta d minimă din Q

pentru fiecare $uv \in E$ **execută** // relaxare uv

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută**

$u =$ extrage vârf cu eticheta d minimă din Q

pentru fiecare $uv \in E$ **execută**

dacă ~~$v \in Q$ și~~ $d[u] + w(u, v) < d[v]$ **atunci**

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

$tata[v] = u$

Algoritmul lui Dijkstra

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută**

$u =$ extrage vârf cu eticheta d minimă din Q

pentru fiecare $uv \in E$ **execută**

dacă $d[u] + w(u, v) < d[v]$ **atunci**

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

$tata[v] = u$

scrie $d, tata$

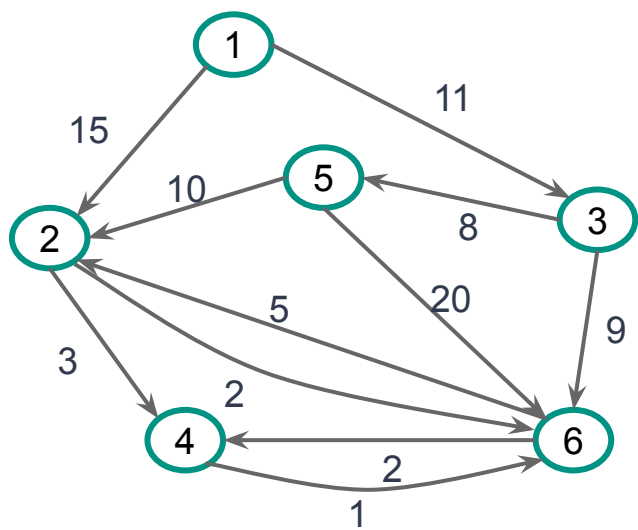
// scrie drum minim de la s la un vârf t dat folosind **tata**

Algoritmul lui Dijkstra

Observatie

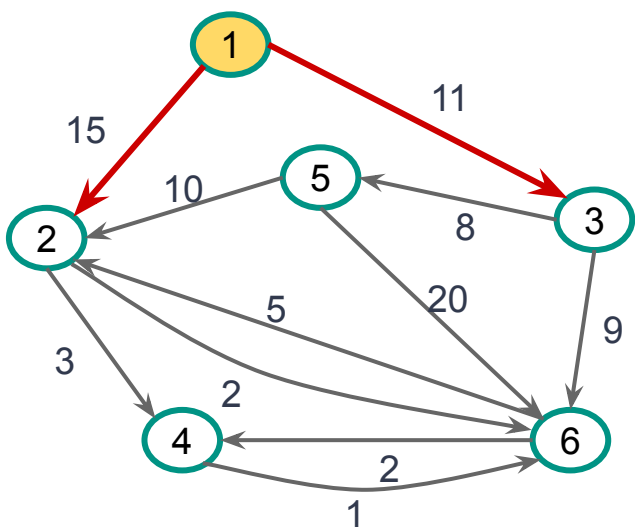
Vom demonstra că, atunci când u este extras din Q , eticheta lui $d[u]$ este chiar egală cu $\delta(s, u)$ (este corectă) și **nu se va mai actualiza** $\Rightarrow \cancel{v \in Q}$

Exemplu



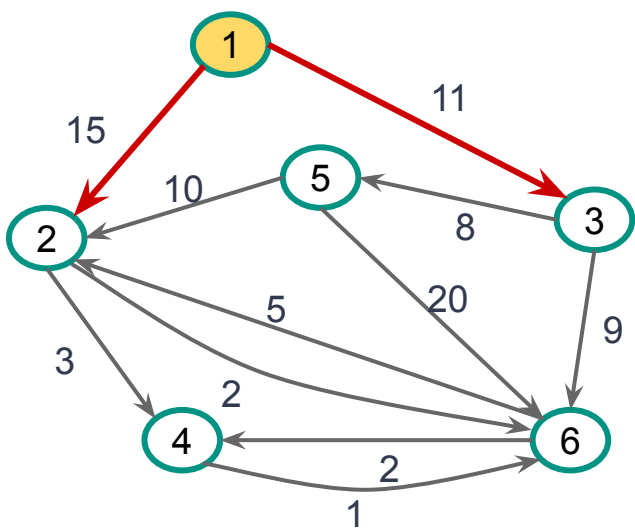
$d/tata =$

1	2	3	4	5	6
[0/0,	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0$]



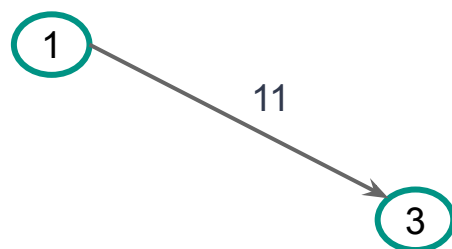
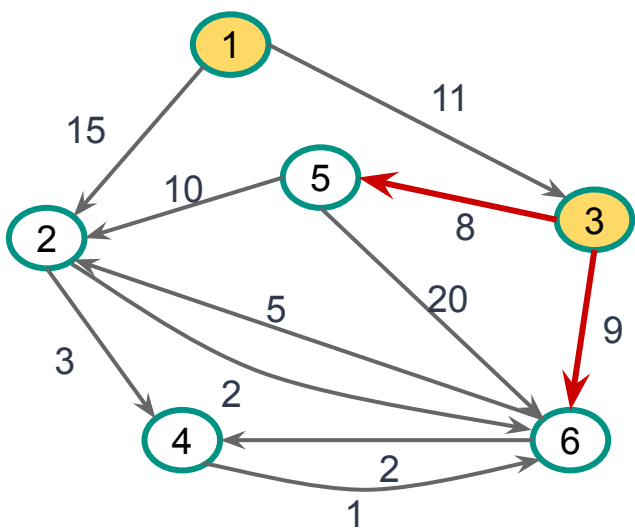
	1	2	3	4	5	6
d/tata	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Selectăm 1						

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



	1	2	3	4	5	6
d/tata	[0/0 ,	$\infty/0$,	$\infty/0$,	$\infty/0$,	$\infty/0$,	$\infty/0$]
Selectăm 1	[- ,	15/1 ,	11/1 ,	$\infty/0$,	$\infty/0$,	$\infty/0$]

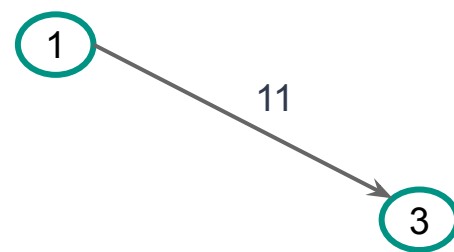
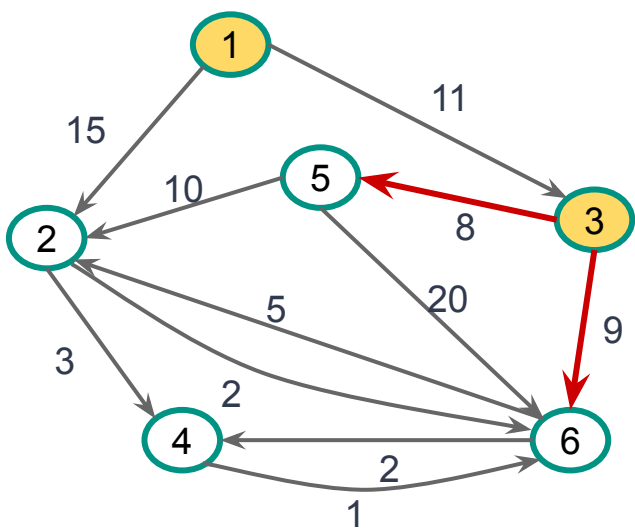
$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3						

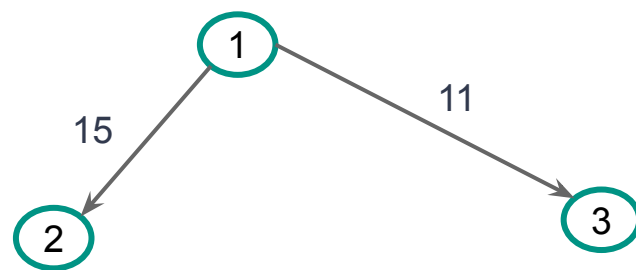
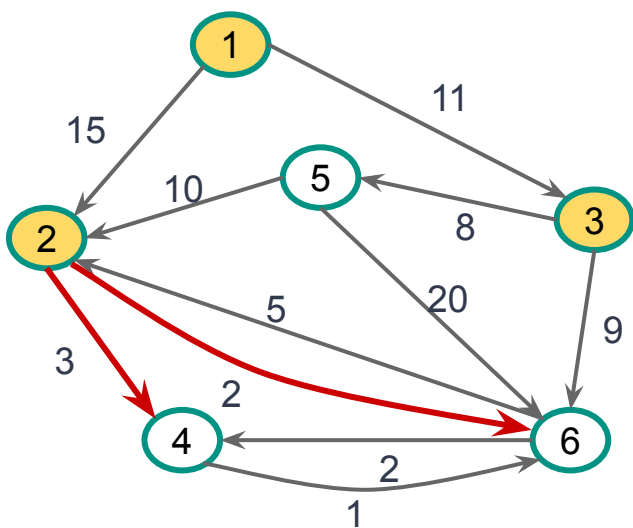
$$d[5] = \min \{ d[5], d[3] + w(3, 5) \}$$

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



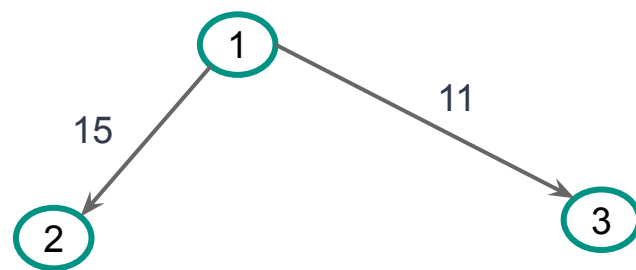
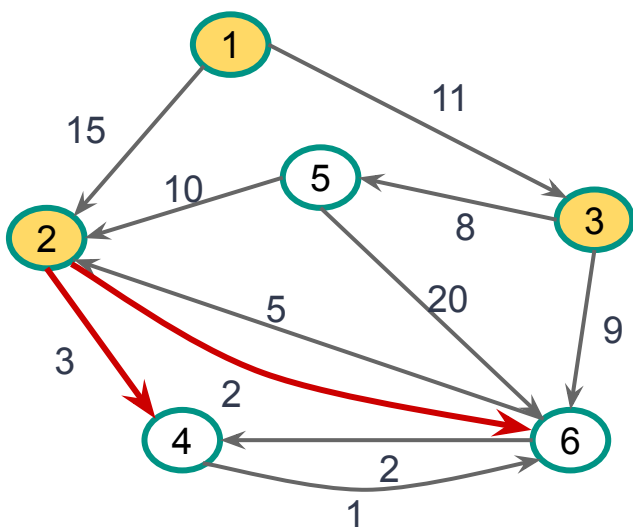
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



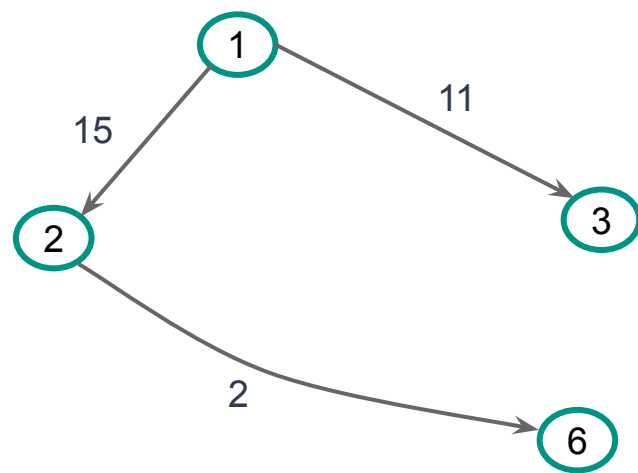
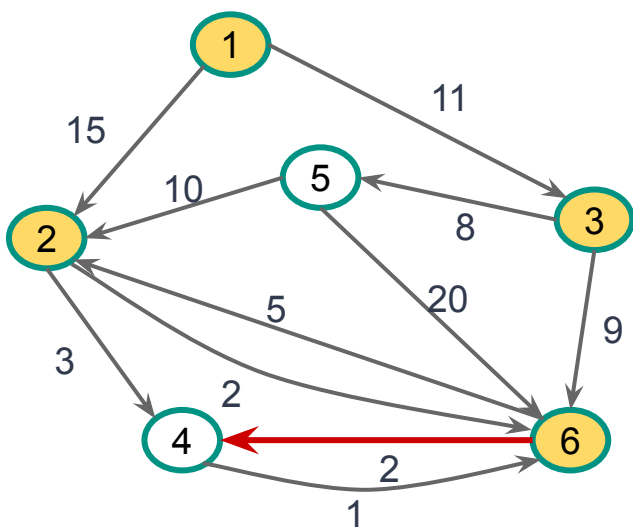
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2						

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



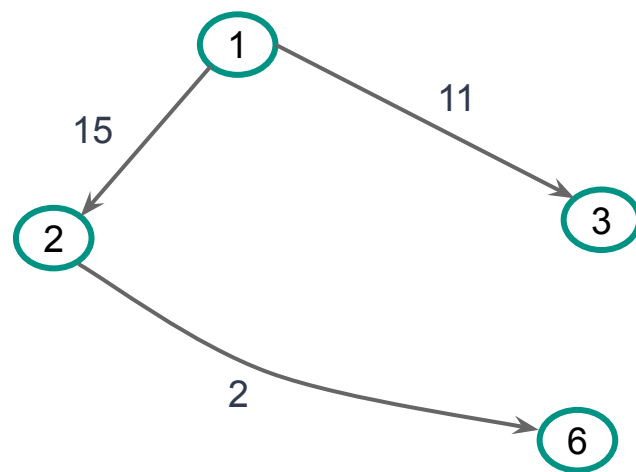
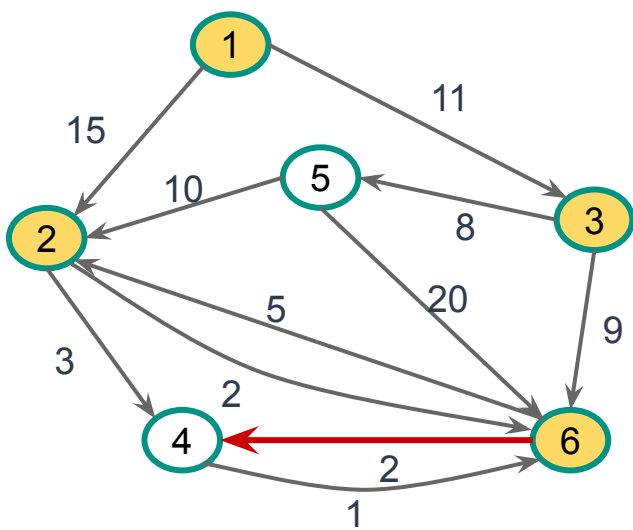
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



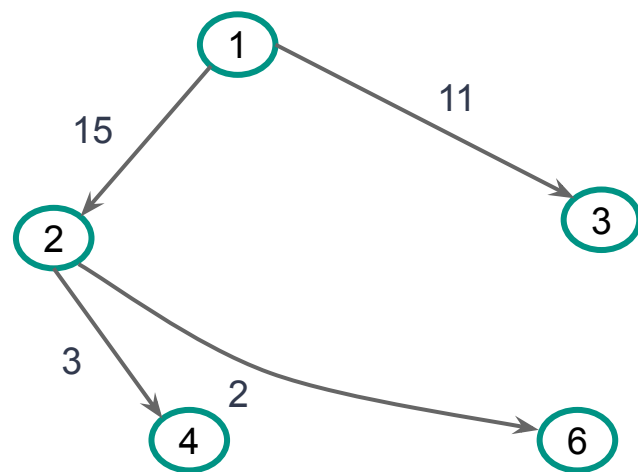
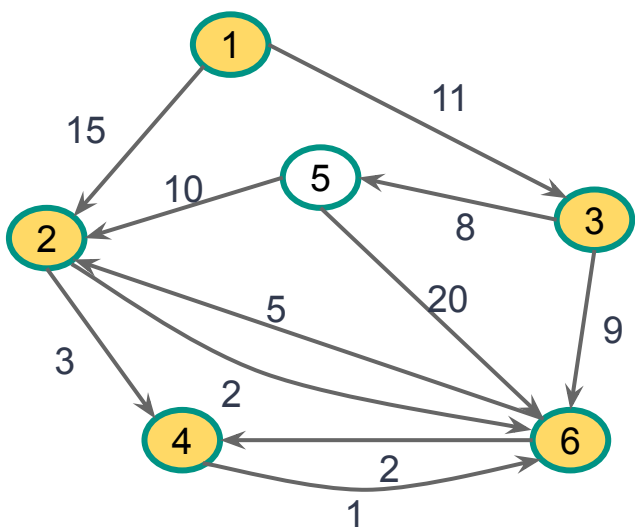
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6						

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



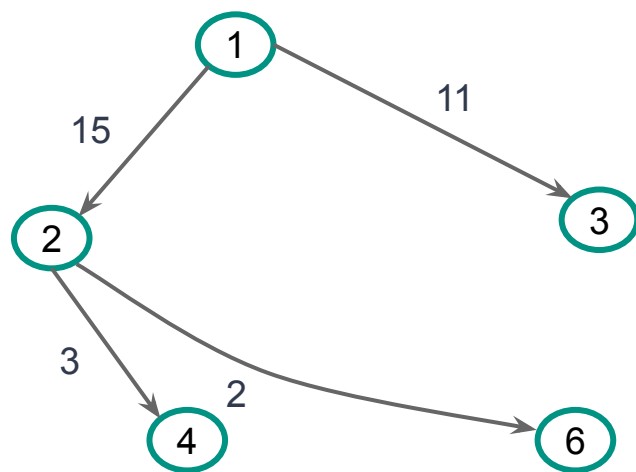
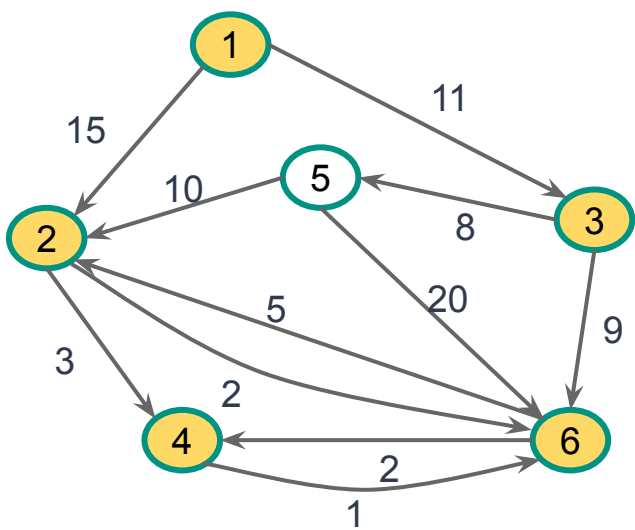
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6	[- ,	- ,	- ,	18/2,	19/3,	-]

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



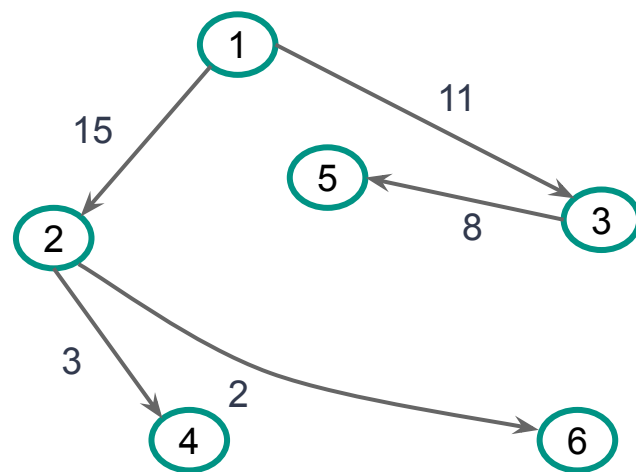
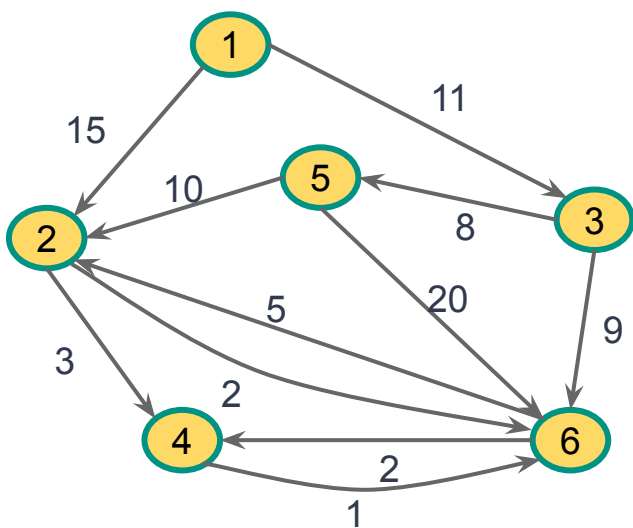
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6	[- ,	- ,	- ,	18/2,	19/3,	-]
Selectăm 4						

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



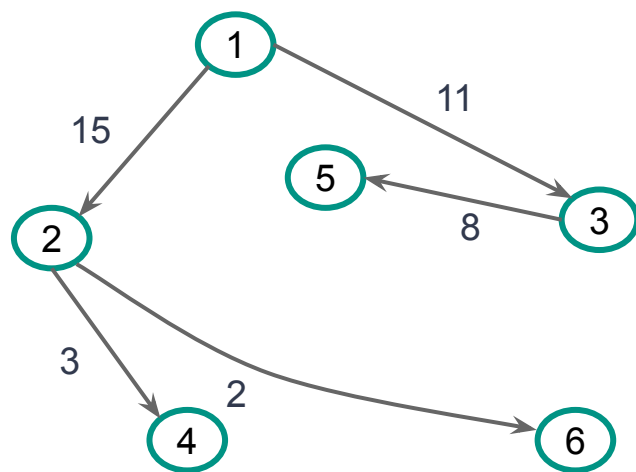
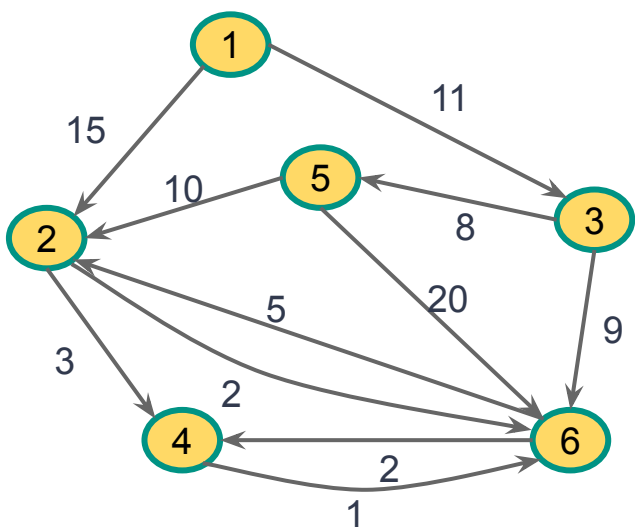
	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6	[- ,	- ,	- ,	18/2,	19/3,	-]
Selectăm 4	[- ,	- ,	- ,	- ,	19/3,	-]

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$

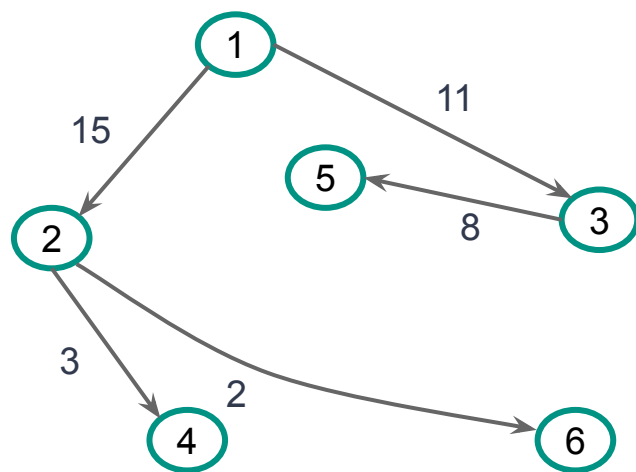
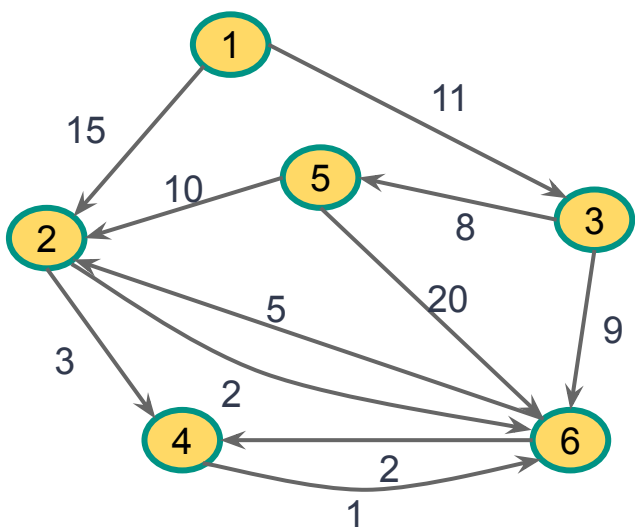


	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6	[- ,	- ,	- ,	18/2,	19/3,	-]
Selectăm 4	[- ,	- ,	- ,	- ,	19/3,	-]
Selectăm 5						

$$d[v] = \min \{ d[v], d[u] + w(u, v) \}$$



	1	2	3	4	5	6
d/tata	[0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]
Selectăm 1	[- ,	15/1,	11/1,	∞ /0,	∞ /0,	∞ /0]
Selectăm 3	[- ,	15/1,	- ,	∞ /0,	19/3,	20/3]
Selectăm 2	[- ,	- ,	- ,	18/2,	19/3,	17/2]
Selectăm 6	[- ,	- ,	- ,	18/2,	19/3,	-]
Selectăm 4	[- ,	- ,	- ,	- ,	19/3,	-]
Selectăm 5	[- ,	- ,	- ,	- ,	- ,	-]



d/tata
SOLUȚIE

	1	2	3	4	5	6
[0/0,						
15/1,						
11/1,						
18/2,						
19/3,						
17/2]						

Un drum minim de la 1 la 6?

Algoritmul lui Dijkstra

Observații

- **Dacă vârful u curent are eticheta $d[u] = \infty$, algoritmul se poate opri**
- Vectorul **tata** memorează **arborele distanțelor față de s** (vârfurile neaccesibile din s rămân cu tata 0)

Algoritmul lui Dijkstra – Complexitate

Dijkstra(G, w, s)

inițializează mulțimea vârfurilor neselectate Q cu V

pentru fiecare $u \in V$ **execută**

$d[u] = \infty$; $tata[u] = 0$

$d[s] = 0$

cât timp $Q \neq \emptyset$ **execută**

$u =$ extrage vârf cu eticheta d minimă din Q

pentru fiecare $uv \in E$ **execută**

dacă $d[u] + w(u, v) < d[v]$ **atunci**

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

$tata[v] = u$

scrie $d, tata$

// scrie drum minim de la s la un vârf t dat folosind **tata**

Algoritmul lui Dijkstra – Complexitate



- Cum memorăm Q = vârfurile încă neselectate?

Algoritmul lui Dijkstra – Complexitate

Q poate fi (ca și în cazul algoritmului lui Prim)

- **vector**
 - $Q[u] = 1$, dacă u este selectat ($u \notin Q$)
0, altfel ($u \in Q$)
- **min-ansamblu (heap)**

Algoritmul lui Dijkstra – Complexitate

Varianta 1 - reprezentarea lui Q ca vector

$Q[u] = 1$, dacă u este selectat ($u \notin Q$)
0, altfel ($u \in Q$)

- ☐ **Inițializare Q** →
 - ☐ **n * extragere vârf minim** →
 - ☐ **actualizare etichete vecini** →
-

Algoritmul lui Dijkstra – Complexitate

Varianta 1 - reprezentarea lui Q ca vector

$Q[u] = 1$, dacă u este selectat ($u \notin Q$)
0, altfel ($u \in Q$)

- ☐ **Inițializare Q** $\rightarrow O(n)$
 - ☐ **n * extragere vârf minim** \rightarrow
 - ☐ **actualizare etichete vecini** \rightarrow
-

Algoritmul lui Dijkstra – Complexitate

Varianta 1 - reprezentarea lui Q ca vector

$Q[u] = 1$, dacă u este selectat ($u \notin Q$)
 0 , altfel ($u \in Q$)

- ☐ **Inițializare Q** $\rightarrow O(n)$
- ☐ **n * extragere vârf minim** $\rightarrow O(n^2)$
- ☐ **actualizare etichete vecini** \rightarrow _____

Algoritmul lui Dijkstra – Complexitate

Varianta 1 - reprezentarea lui Q ca vector

$Q[u] = 1$, dacă u este selectat ($u \notin Q$)
0, altfel ($u \in Q$)

- **Inițializare Q** → $O(n)$
 - **n * extragere vârf minim** → $O(n^2)$
 - **actualizare etichete vecini** → $O(m)$
-

Algoritmul lui Dijkstra – Complexitate

Varianta 1 - reprezentarea lui Q ca vector

$Q[u] = 1$, dacă u este selectat ($u \notin Q$)
0, altfel ($u \in Q$)

- **Inițializare Q** → $O(n)$
- **n * extragere vârf minim** → $O(n^2)$
- **actualizare etichete vecini** → $O(m)$

$O(n^2)$

Algoritmul lui Dijkstra – Complexitate

Varianta 2 - reprezentarea lui Q ca min-heap

- ☐ Inițializare Q →
 - ☐ n * extragere vârf minim →
 - ☐ actualizare etichete vecini →
-

Algoritmul lui Dijkstra – Complexitate

```
Dijkstra(G, w, s)  // - Q min-heap în raport cu d
    pentru fiecare u ∈ V execută
        d[u] = ∞; tata[u] = 0
    d[s] = 0
    Q = V    // creare heap cu cheile din d
    cât timp Q ≠ ∅ execută
        u = extrage_min(Q)
        pentru fiecare uv ∈ E execută
            dacă d[u] + w(u, v) < d[v] atunci
                d[v] = d[u] + w(u, v)
                repara(Q, v)
                tata[v] = u

    scrie d, tata
    // scrie drum minim de la s la un vârf t dat folosind tata
```

Algoritmul lui Dijkstra – Complexitate

Varianta 2 - reprezentarea lui Q ca min-heap

- ☐ Inițializare Q → $O(n)$
- ☐ n * extragere vârf minim → $O(n \log n)$
- ☐ actualizare etichete vecini → _____

Algoritmul lui Dijkstra – Complexitate

Varianta 2 - reprezentarea lui Q ca min-heap

□	Inițializare Q	→	$O(n)$
□	n * extragere vârf minim	→	$O(n \log n)$
□	actualizare etichete vecini	→	$O(m \log n)$
	!! + actualizare Q		<hr/>
			$O(m \log n)$

Algoritmul lui Dijkstra

Observație. Pentru a determina drumul minim între două vârfuri s și t **date**, putem folosi algoritmul lui Dijkstra, cu următoarea modificare:

- dacă vârful u ales este chiar t , **algoritmul se oprește**
- drumul de la s la t se afișează folosind vectorul $tata$ (vezi BF)

Algoritmul lui Dijkstra

- **Dijkstra \approx Prim** (versiunea $O(n^2)$ / $O(m \log n)$)

Algoritmul lui Dijkstra



- **Algoritmul funcționează și pentru grafuri neorientate?**

Algoritmul lui Dijkstra



- ☐ De ce nu funcționează corect algoritmul dacă avem arce cu cost negativ? Exemplu.
- ☐ Cum putem rezolva problema dacă avem și arce de cost negativ?

Algoritmul lui Dijkstra

- ❑ De ce nu funcționează corect algoritmul dacă avem arce cu cost negativ? Exemplu.
- ❑ Cum putem rezolva problema dacă avem și arce de cost negativ?



Putem aduna o constantă la costul fiecărui arc, astfel încât toate arcele să aibă cost pozitiv. **Drumul minim între 2 vârfuri rămâne la fel?**

Algoritmul lui Dijkstra

- De ce nu funcționează corect algoritmul dacă avem arce cu cost negativ? Exemplu.
- Cum putem rezolva problema dacă avem și arce de cost negativ?

Putem aduna o constantă la costul fiecărui arc, astfel încât toate arcele să aibă cost pozitiv. Drumul minim între 2 vârfuri rămâne la fel?

- NU



Algoritmul lui Dijkstra

Cum putem rezolva problema dacă avem și arce de cost negativ?



Algoritmul BELLMAN-FORD

(suplimentar)

- La un pas, nu relaxăm arcele dintr-un vârf selectat u , ci **din toate vârfurile** (deci relaxăm toate arcele)

Algoritmul lui Dijkstra

Cum putem rezolva problema dacă avem și arce de cost negativ?



Algoritmul BELLMAN-FORD

(suplimentar)

- La un pas, nu relaxăm arcele dintr-un vârf selectat u , ci **din toate vârfurile** (deci relaxăm toate arcele)

pentru $i = 1, n-1$ **execută**

pentru fiecare $uv \in E$ **execută**

dacă $d[u] + w(u, v) < d[v]$ **atunci**

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

$tata[v] = u$

Algoritmul lui Dijkstra

Cum putem rezolva problema dacă avem și arce de cost negativ?



Algoritmul BELLMAN-FORD

- La un pas, nu relaxăm arcele dintr-un vârf selectat u , ci **din toate vârfurile** (deci relaxăm toate arcele)

pentru $i = 1, n-1$ execută

 pentru fiecare $uv \in E$ execută

 dacă $d[u] + w(u, v) < d[v]$ atunci

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

$tata[v] = u$

- După pasul $i \rightarrow d[u] =$ costul minim al unui $s-u$ drum cu cel mult i arce

Algoritmul lui Dijkstra - Corectitudine

Algoritmul lui Dijkstra – Corectitudine

Lema 1. Pentru orice $u \in V$, la orice pas al algoritmului lui Dijkstra, avem:

- **dacă $d[u] < \infty$** , există un drum de la s la u în G , de cost $d[u]$, iar acesta se poate determina din vectorul $tata$:
 - $tata[u] =$ predecesorul lui u pe un drum de la s la u de cost $d[u]$
- **$d[u] \geq \delta(s, u)$**

Algoritmul lui Dijkstra – Corectitudine

Lema 1. Pentru orice $u \in V$, la orice pas al algoritmului lui Dijkstra, avem:

- **dacă $d[u] < \infty$** , există un drum de la s la u în G , de cost $d[u]$, iar acesta se poate determina din vectorul $tata$:
 - $tata[u] =$ predecesorul lui u pe un drum de la s la u de cost $d[u]$
- **$d[u] \geq \delta(s, u)$**

Consecință. Dacă, la un pas al algoritmului, avem, pentru un vârf u , relația **$d[u] = \delta(s, u)$** , atunci $d[u]$ nu se mai modifică până la final.

Algoritmul lui Dijkstra – Corectitudine

Teoremă

Fie $G = (V, E, w)$ un graf orientat ponderat cu
 $w : E \rightarrow \mathbb{R}_+$ și $s \in V$ fixat.

La finalul algoritmului lui Dijkstra, avem:

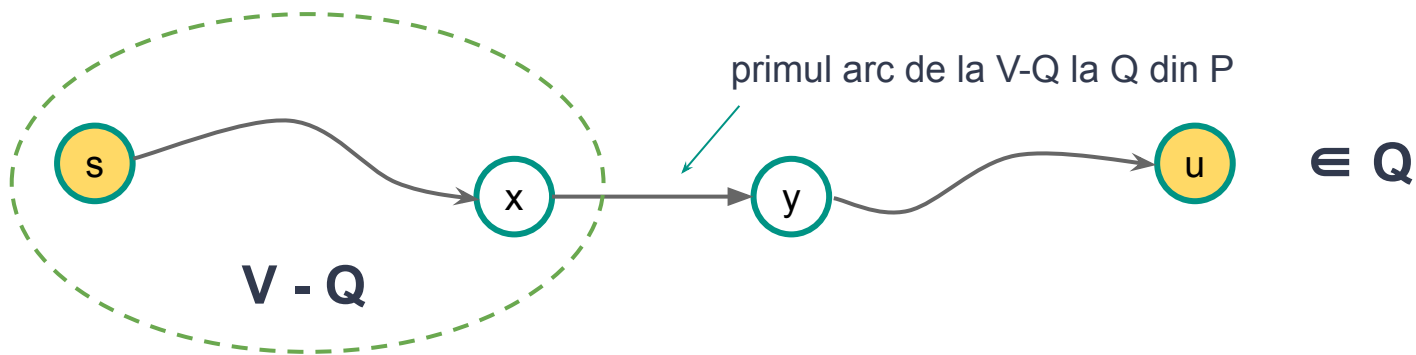
$$d[u] = \delta(s, u), \text{ pentru orice } u \in V$$

și tata memorează un arbore al distanțelor față de s .

Algoritmul lui Dijkstra – Corectitudine

Demonstrație (idee). Inducție: $d[x] = \delta(s, x)$, $\forall x \notin Q$ (= deja selectat)

Când un vârf u este selectat: fie P un s - u drum minim



din modul în care este ales u

după relaxarea lui xy (mai mult, are loc chiar egalitate:
 $d[y] = \delta(s, x) + w(x, y) = w(s \xrightarrow{P} y) = \delta(s, y)$)

$$d[u] \leq d[y] \leq d[x] + w(x, y) = \delta(s, x) + w(x, y) = w(s \xrightarrow{P} y) \leq w(P) \leq d[u]$$

$$\Rightarrow d[u] = d[y] = w(P) = \delta(s, u)$$

ipoteza de inducție pentru x

