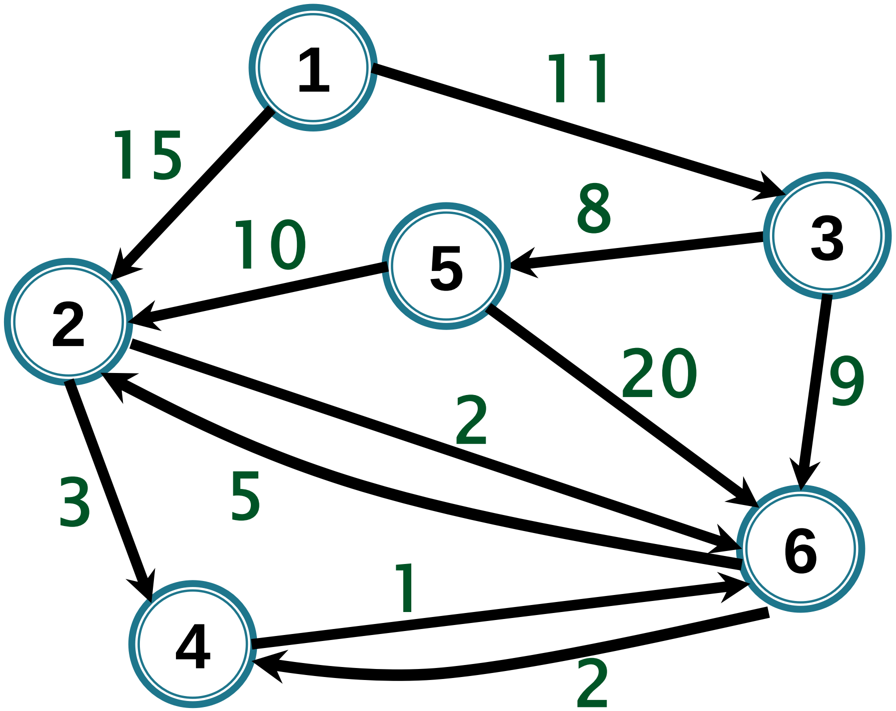
Algoritmul lui Dijkstra

- **Ipoteză:**

  Presupunem că arcele au <u>cost pozitiv</u>

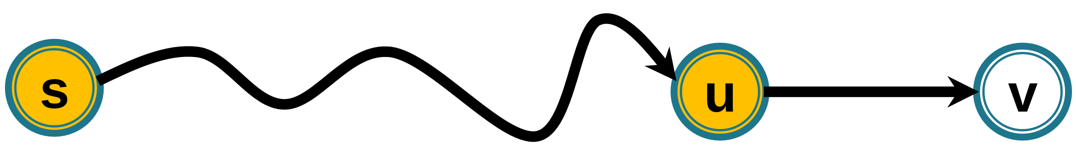
  (graful poate conţine circuite)


1
11
15
3
8
10
5
2
20
9
2
3
5
6
1
4
2

Algoritmul lui Dijkstra

Idee: La un pas este ales ca vârf curent (vizitat) vârful u care 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



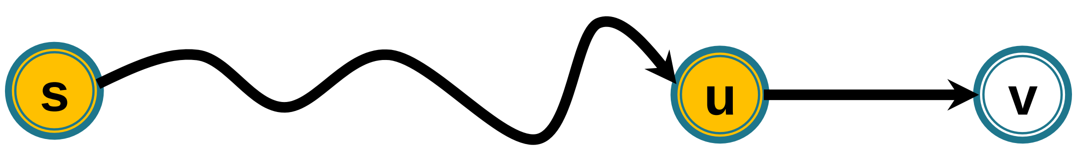
Algoritmul lui Dijkstra

Idee: La un pas este ales ca vârf curent (vizitat) vârful u care 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 ⇒

se actualizează distanțele estimate pentru vecini



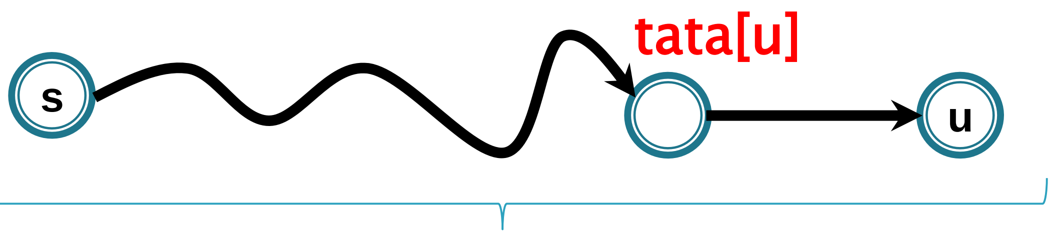
Algoritmul lui Dijkstra

- **generalizare a ideii de parcurgere BF**
- **dacă toate arcele au cost egal Dijkstra $\equiv$ BF**

Pseudocod

Algoritmul lui Dijkstra

- Reţinem pentru fiecare vârf etichetele
  - d[u]
  - tata[u]



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

Algoritmul lui Dijkstra

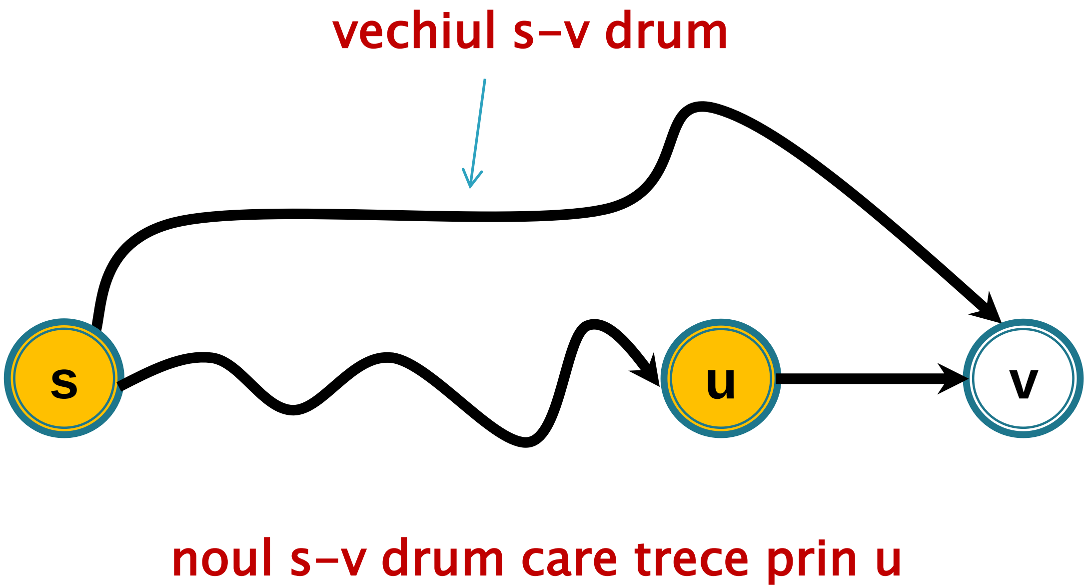
- **La un pas**
  - este selectat un vârf u (neselectat) care “**pare**” cel mai apropiat de s ⇔ are eticheta d minimă

Algoritmul lui Dijkstra

- **La un pas**
  - este selectat un vârf u (neselectat) care "**pare**" cel mai apropiat de s ⇔ are eticheta d minimă
  - Se actualizează etichetele d[v] ale vecinilor lui u – extinzând drumul minim deja găsit de la s la u cu arcul uv
    - **tehnica de <u>relaxare a arcelor</u> care ies din u**

Algoritmul lui Dijkstra

- **Relaxarea unui arc (u, v)** = a verifica dacă d[v] poate fi îmbunătăţit extinzând drumul minim găsit de la s la u cu arcul uv




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

Dijkstra(G, w, s)

```
inițializează mulțimea vârfurilor neselectate Q cu V
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
    d[u] = ∞; tata[u]=0
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
        d[u] = ∞; tata[u]=0
 d[s] = 0
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
        d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa // pentru i=1,n executa
```

Dijkstra(G, w, s)

```
  inițializează mulțimea vârfurilor neselectate Q cu V
  pentru fiecare u∈V executa
        d[u] = ∞; tata[u]=0
   d[s] = 0
   cat timp Q ≠ ∅ executa
        u = extrage vârf cu eticheta d minimă din Q
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
    d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
    u = extrage vârf cu eticheta d minimă din Q
    pentru fiecare uv∈E executa  //relaxare uv
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
       u = extrage vârf cu eticheta d minimă din Q
       pentru fiecare uv∈E executa
              daca v∈Q si d[u]+w(u,v)<d[v] atunci
                     d[v] = d[u]+w(u,v)
                     tata[v] = u
```

Dijkstra(G, w, s)

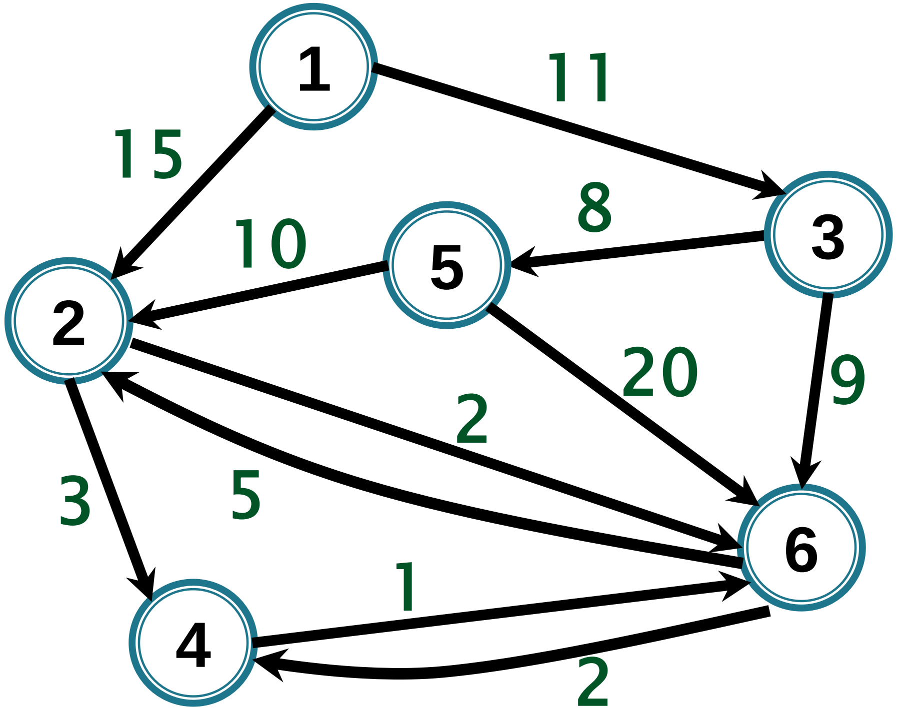
```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
     d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
     u = extrage vârf cu eticheta d minimă din Q
     pentru fiecare uv∈E executa
          daca d[u]+w(u,v)<d[v] atunci
                    d[v] = d[u]+w(u,v)
                    tata[v] = u
 scrie d, tata

 //scrie drum minim de la s la t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

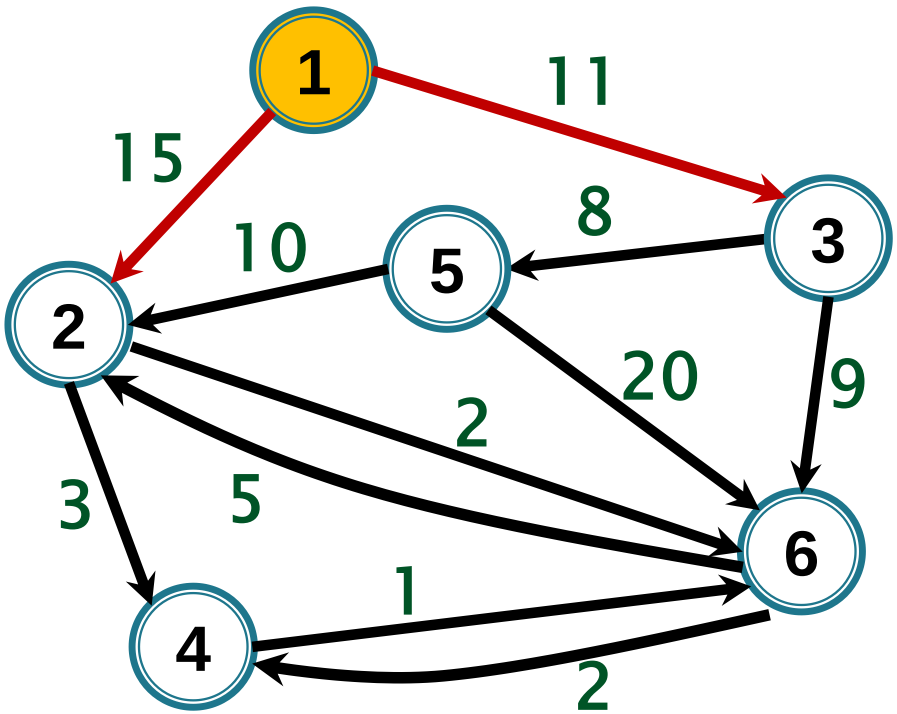
- **Observație**

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

Exemplu

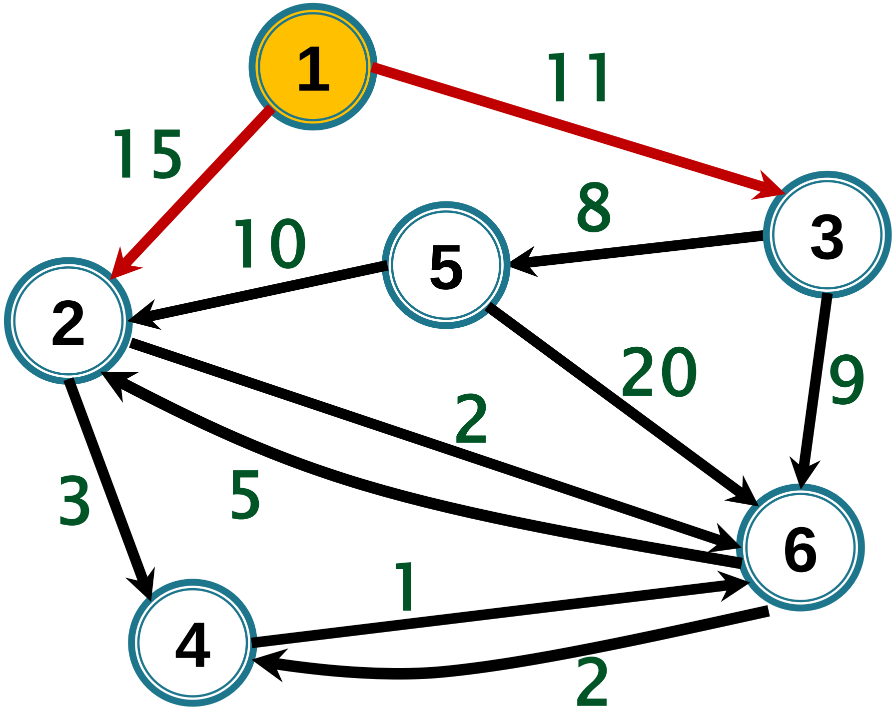


d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]



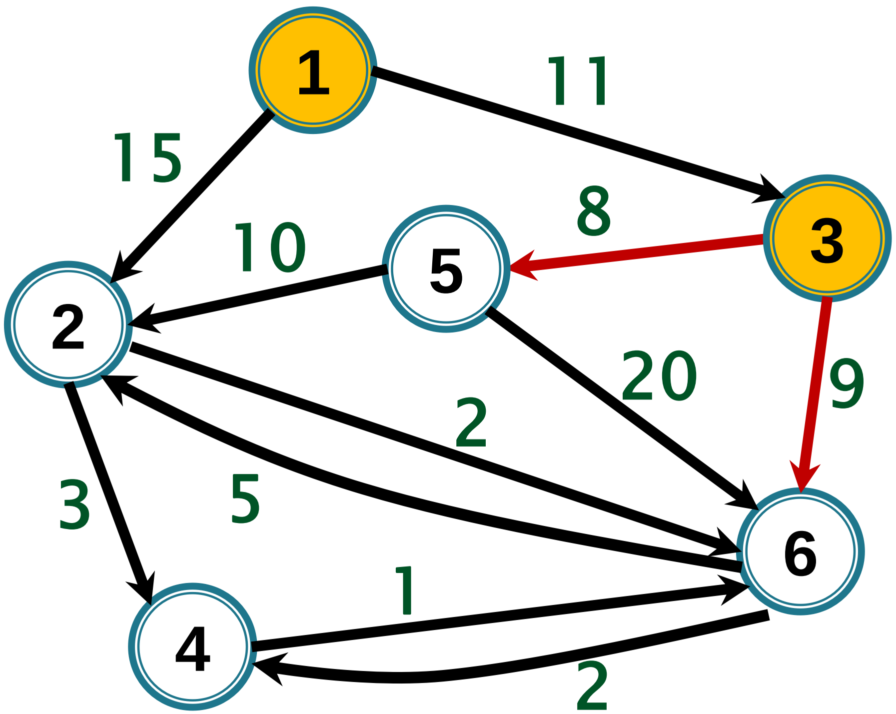
d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:						

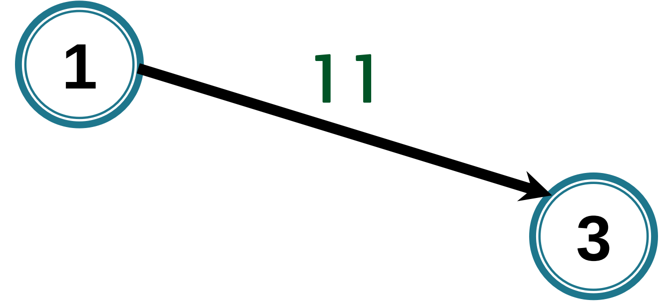
d[v] = min{d[v],d[u]+w(u,v)}



d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]

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

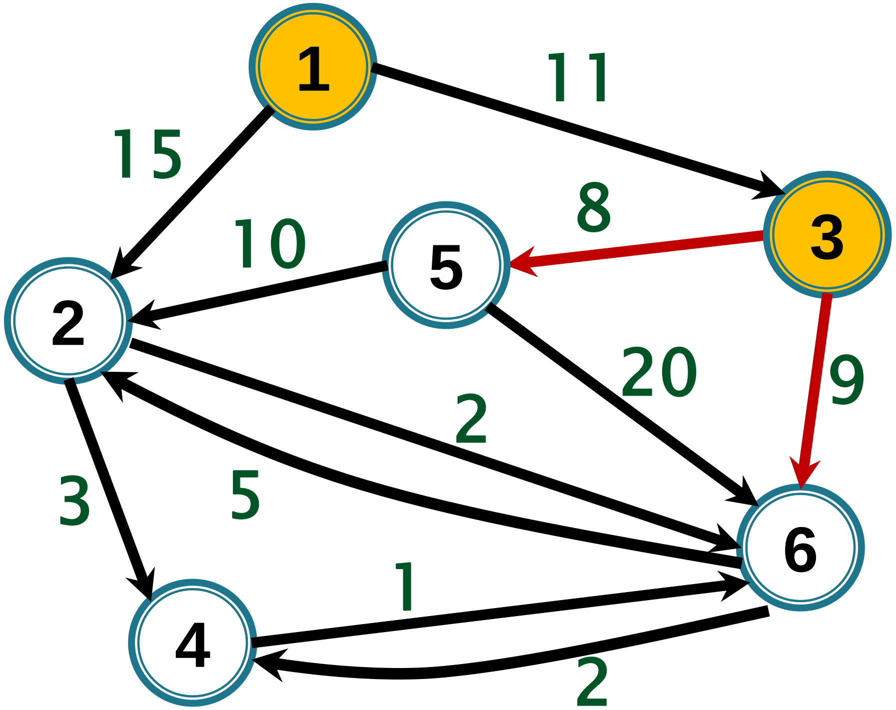


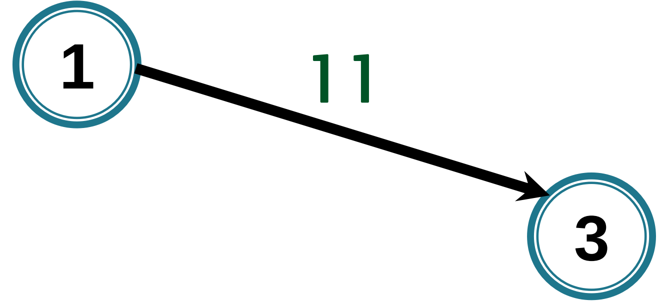


d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3						

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

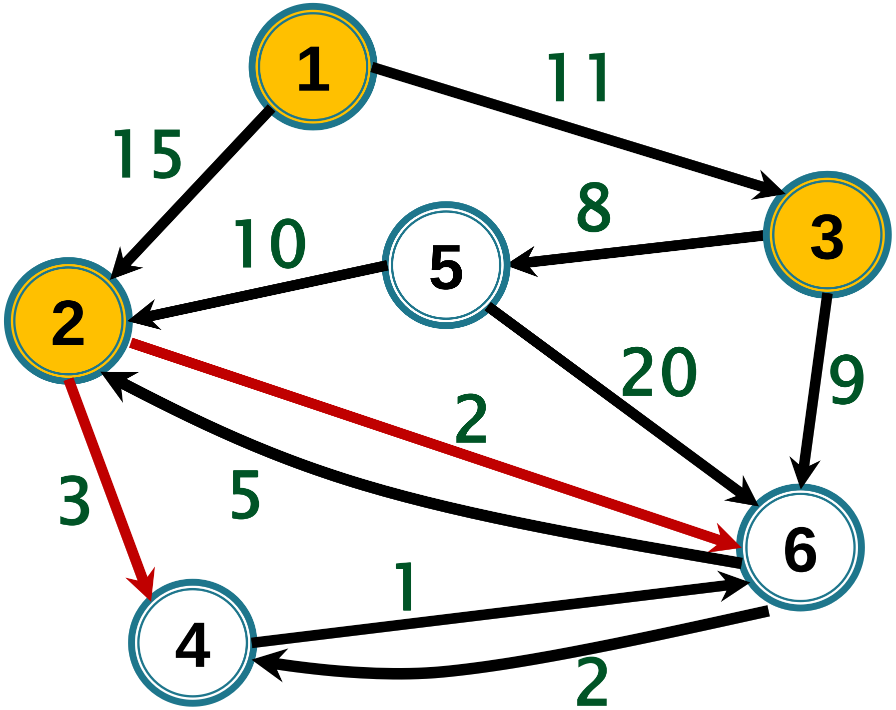
d[v] = min{d[v],d[u]+w(u,v)}

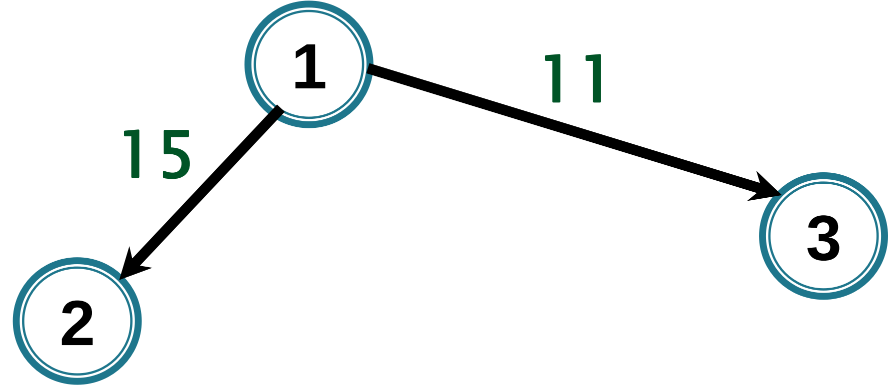




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]

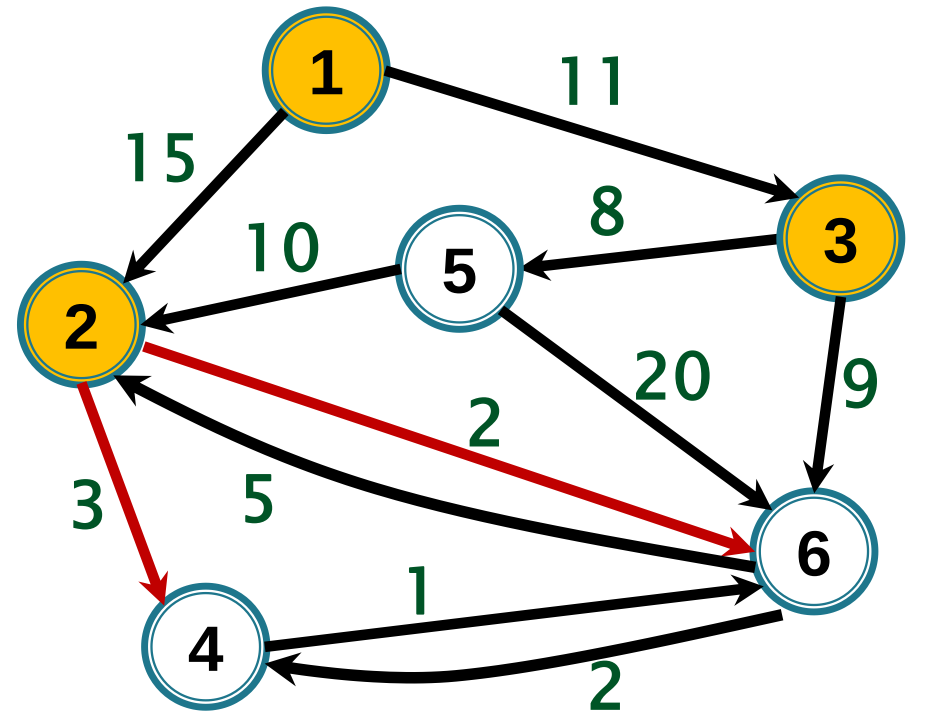
d[v] = min{d[v],d[u]+w(u,v)}

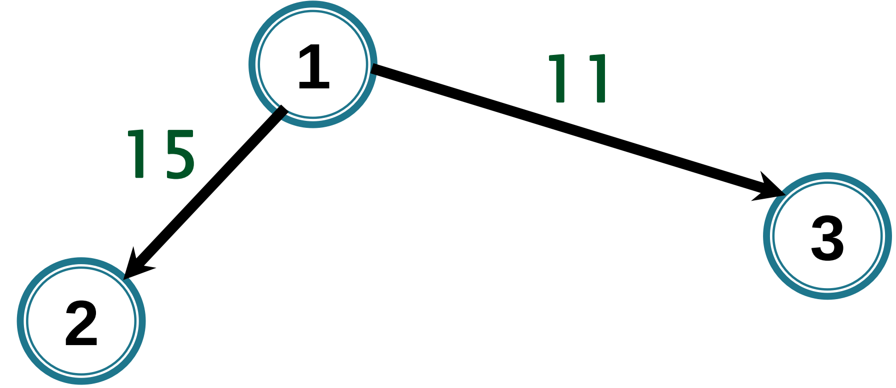




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:						

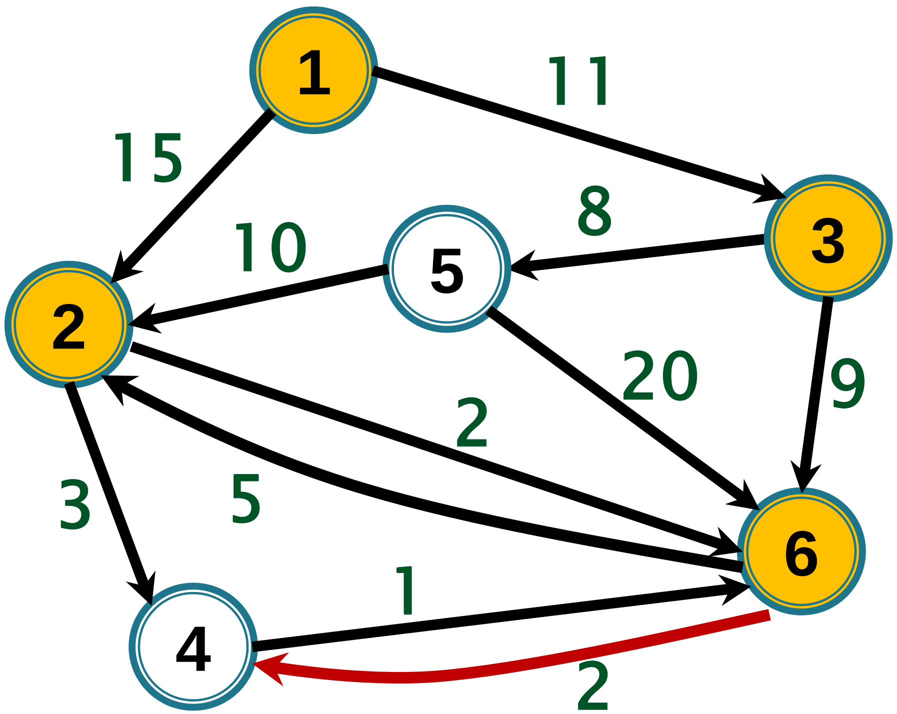
d[v] = min{d[v],d[u]+w(u,v)}

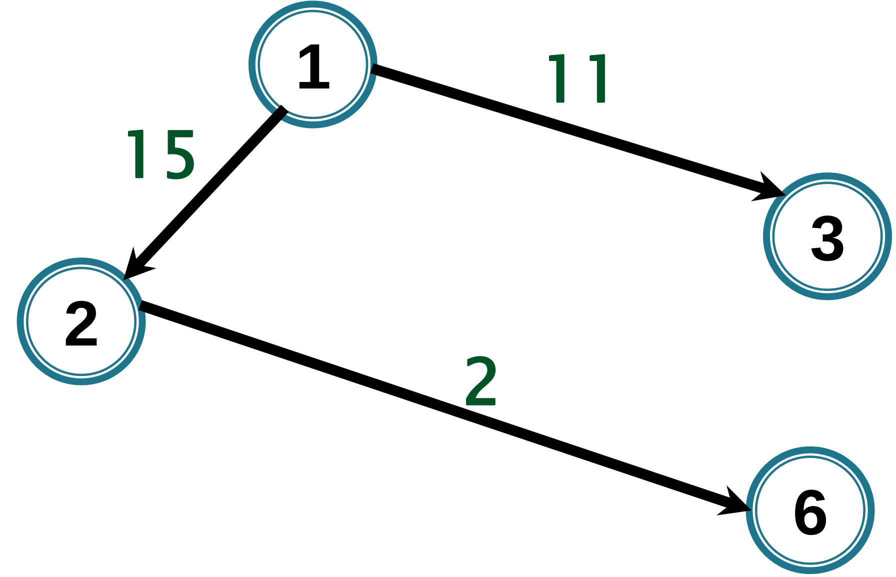




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]

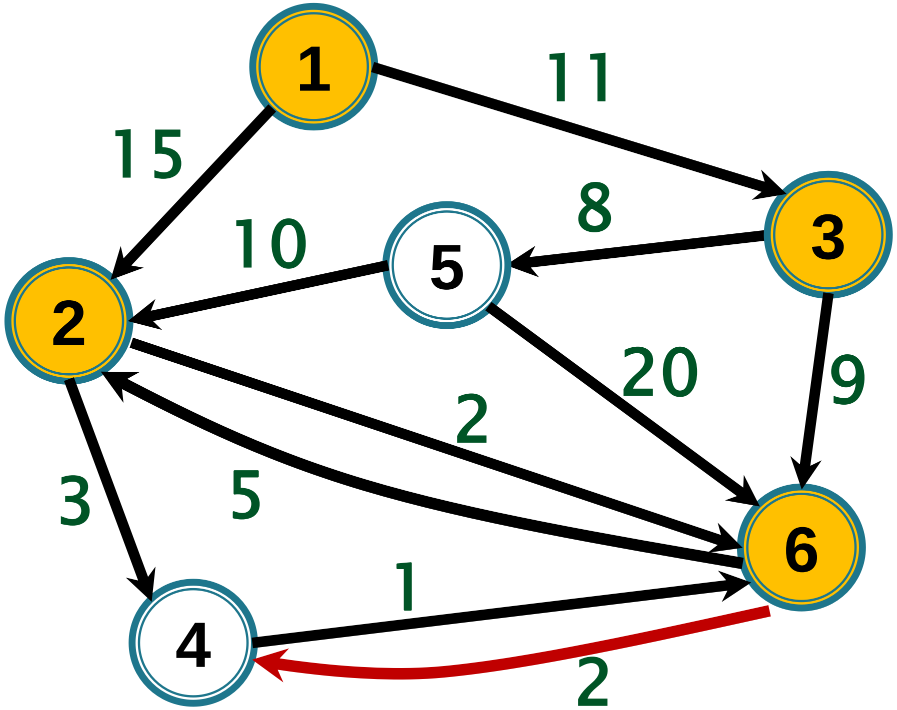
d[v] = min{d[v],d[u]+w(u,v)}

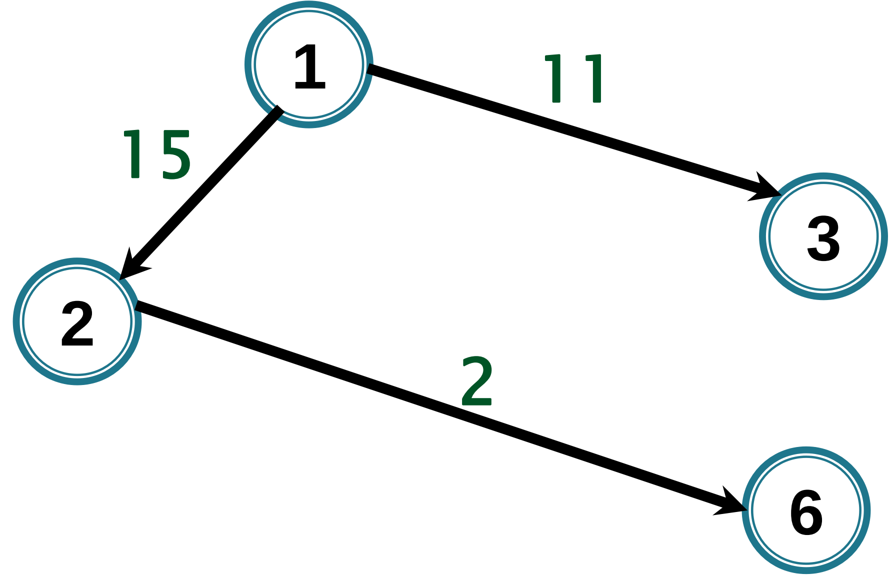




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:						

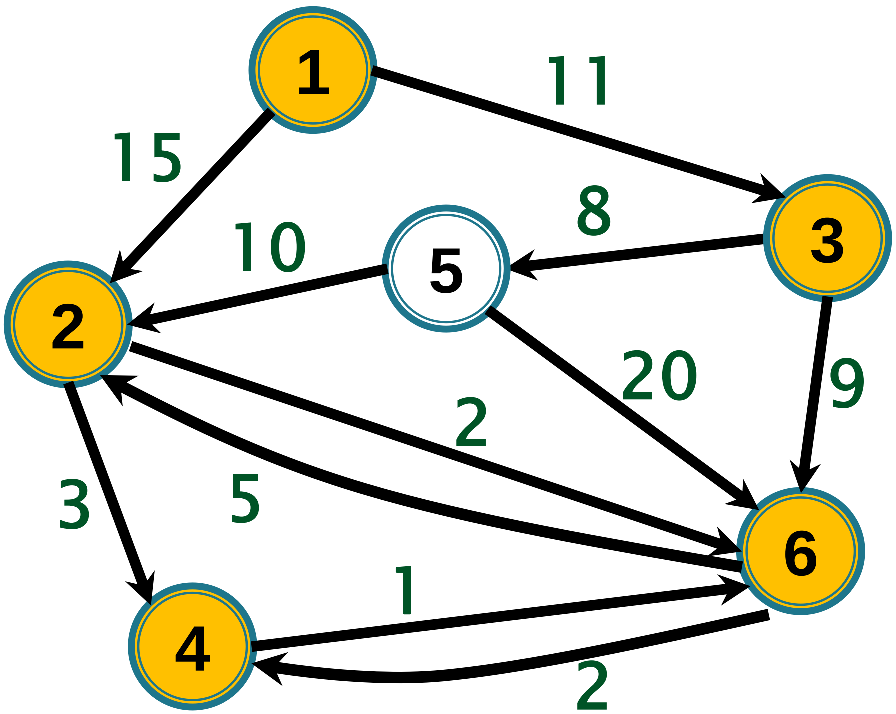
d[v] = min{d[v],d[u]+w(u,v)}

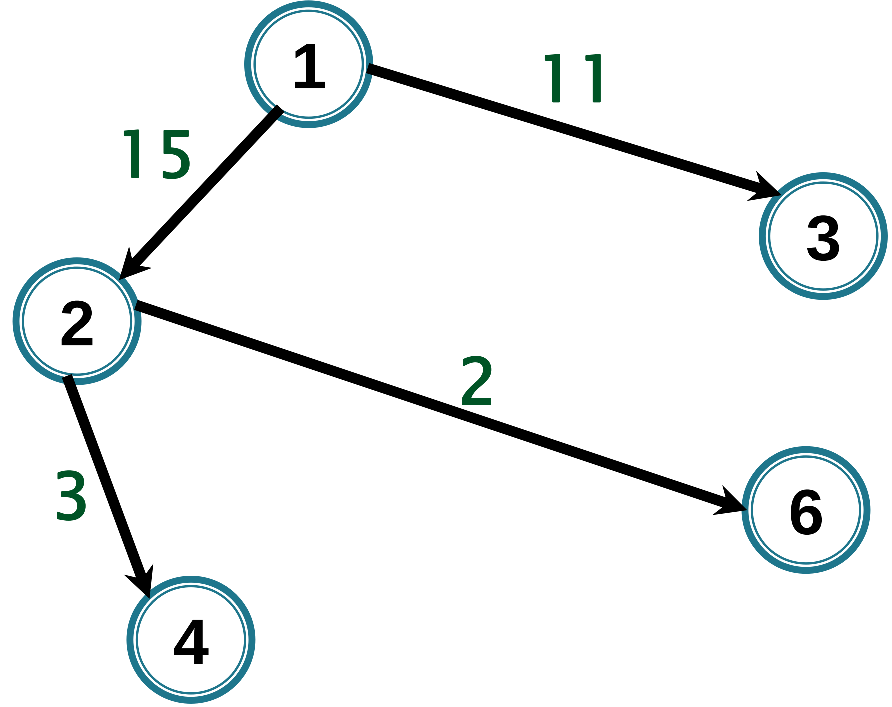




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]

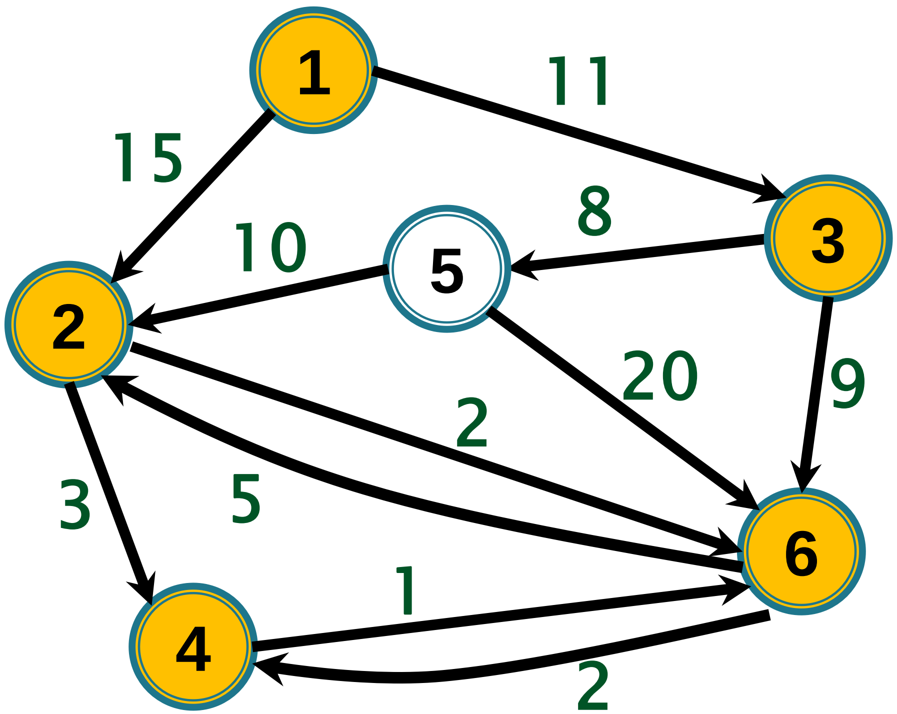
d[v] = min{d[v],d[u]+w(u,v)}

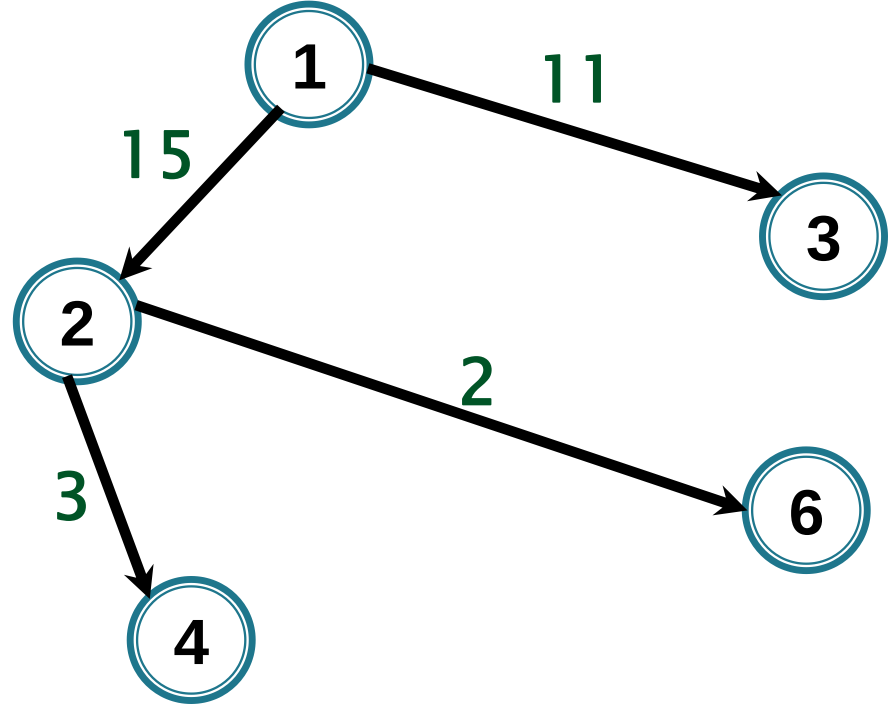




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:						

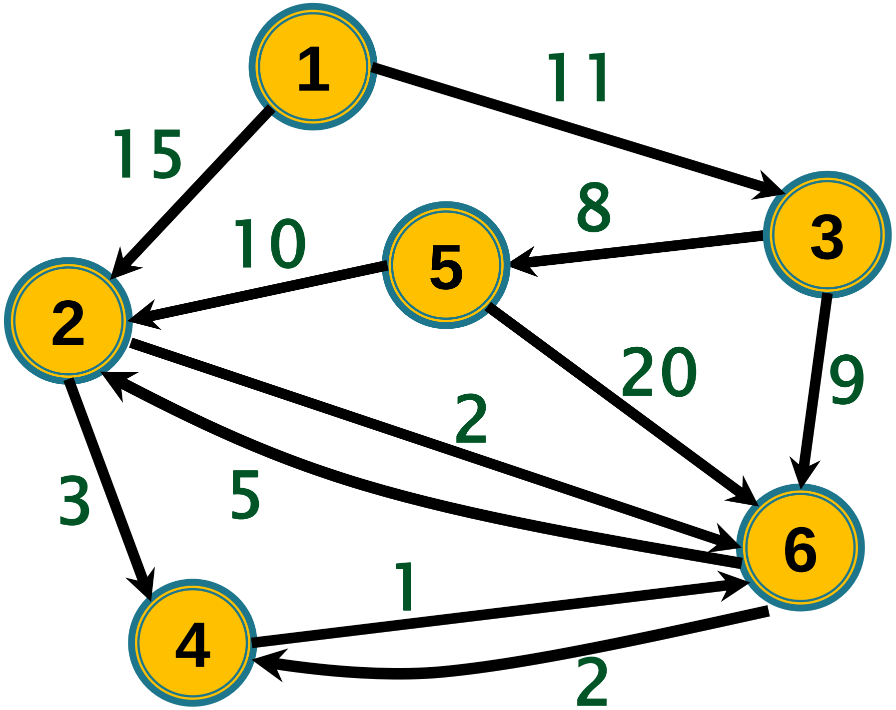
d[v] = min{d[v],d[u]+w(u,v)}

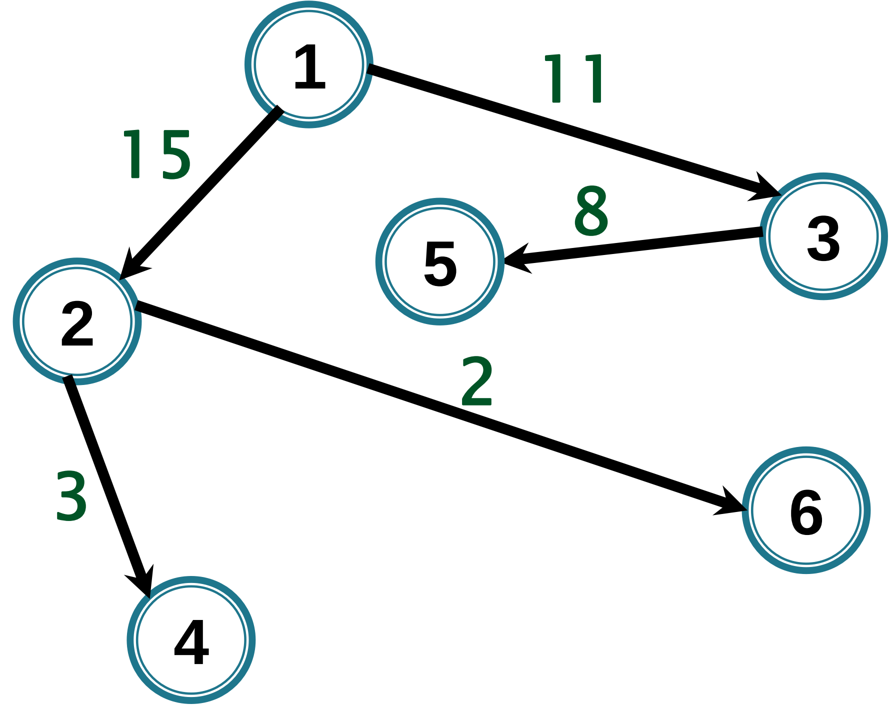




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]

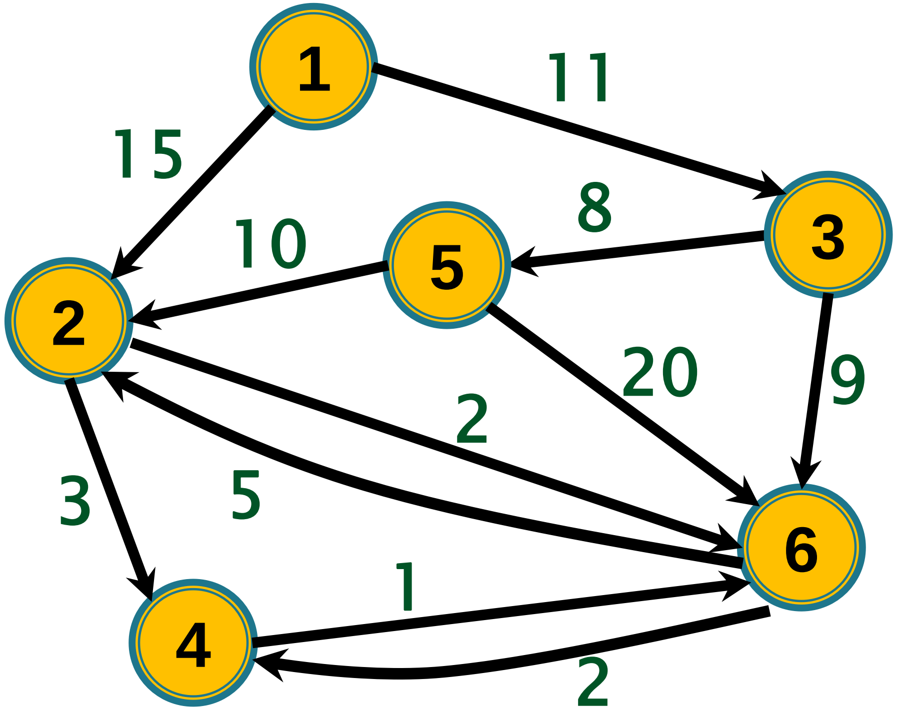
d[v] = min{d[v],d[u]+w(u,v)}

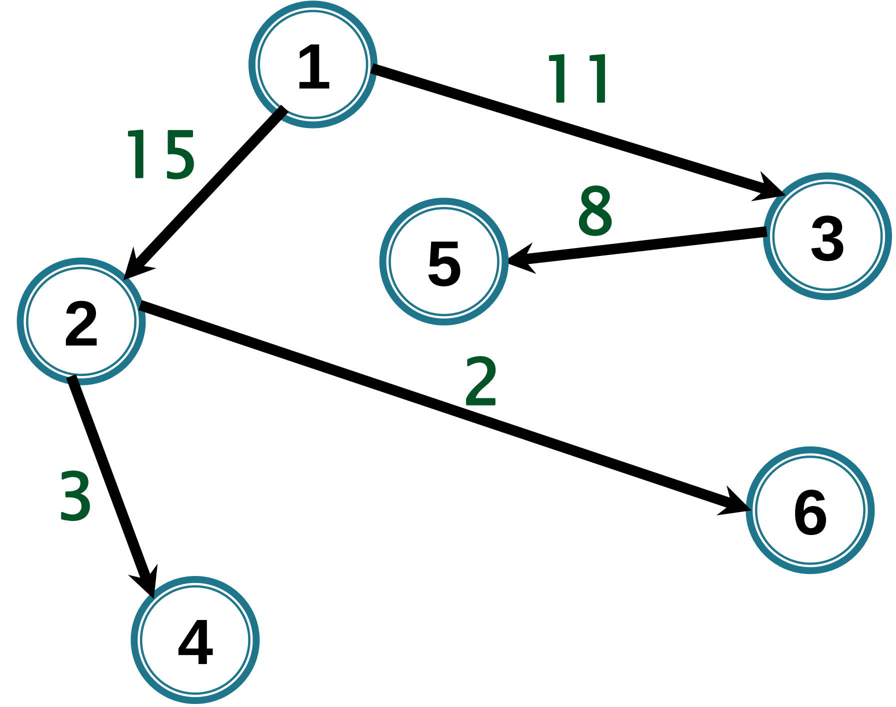




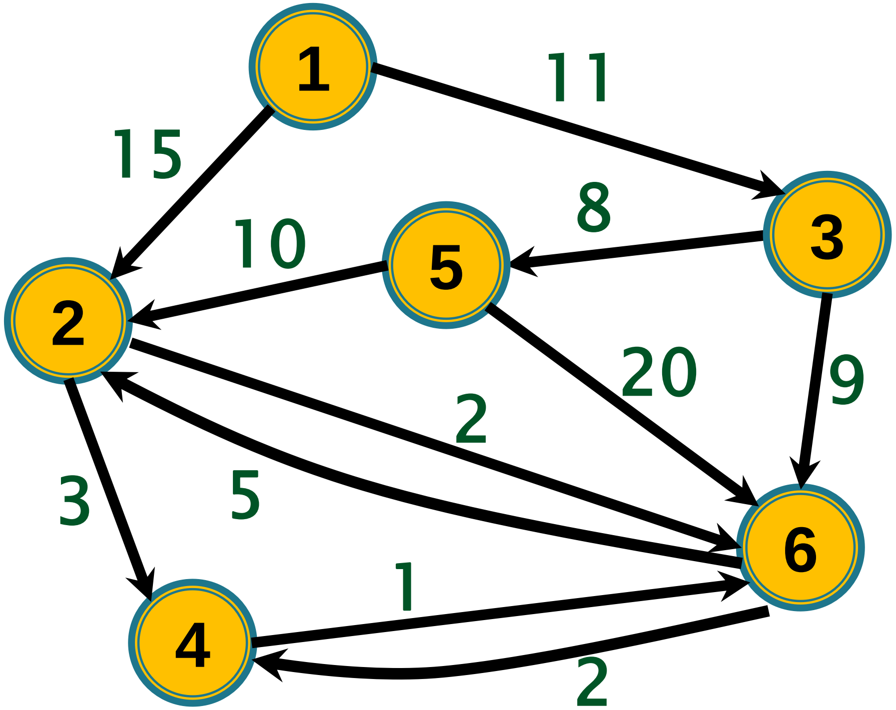
d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]
Sel. 5:						

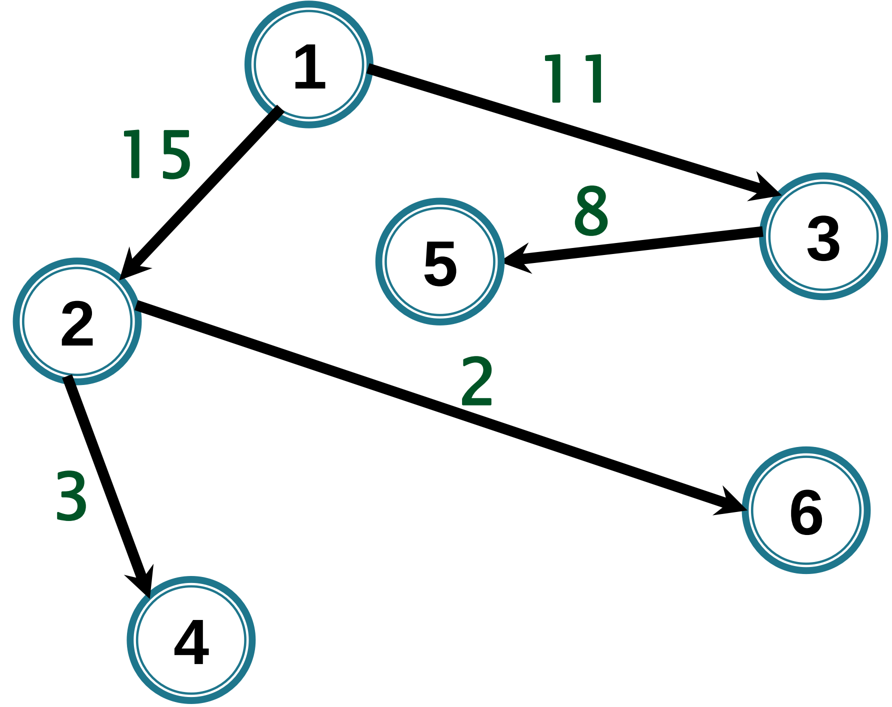
d[v] = min{d[v],d[u]+w(u,v)}





d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]
Sel. 5:	[– ,	– ,	– ,	– ,	– ,	–]





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

Un drum minim de la 1 la 6?

Algoritmul lui Dijkstra

- **Observații.**

1. Dacă vârful u curent are eticheta d[u] = ∞, algoritmul se poate opri

2. Vectorul tata memorează arborele distanțelor față de s
(vârfurile neaccesibile din s rămân cu tata 0)

Complexitate

Dijkstra(G, w, s)

```
 inițializează mulțimea vârfurilor neselectate Q cu V
  pentru fiecare u∈V executa
      d[u] = ∞; tata[u]=0
  d[s] = 0
  cat timp Q ≠ ∅ executa
      u = extrage vârf cu eticheta d minimă din Q
      pentru fiecare uv∈E executa
            daca 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 t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

Cum memorăm Q = vârfurile încă neselectate?

Algoritmul lui Dijkstra

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 – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- Inițializare Q ->
- n * extragere vârf minim ->
- actualizare etichete vecini ->

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

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

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

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

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

Q[u] = 1, dacă u este selectat în V(T)

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 – Q min-heap

- **Inițializare Q** ->
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

Dijkstra(G, w, s) - Q min-heap in raport cu d

```
pentru fiecare u∈V executa
    d[u] = ∞; tata[u]=0
d[s] = 0
Q = V //creare heap cu cheile din d
cat timp Q ≠ ∅ executa
    u = extrage_min(Q)
    pentru fiecare uv∈E executa
        daca 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 t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

Complexitate – Q min-heap

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** -> O(n log n)
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – Q min-heap

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** -> O(n log n)
- **actualizare etichete vecini** -> O(m log n)

  !!+ actualizare Q

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 ≈ Prim** (versiunea $O(n^2)$/$O(m \log n)$)

Algoritmul lui Dijkstra

Algoritmul funcţionează şi pentru grafuri neorientate?

Algoritmul lui Dijkstra

- De ce nu funcţionează corect algoritmul dacă avem arce cu cost negativ + exemplu?

Algoritmul lui Dijkstra

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

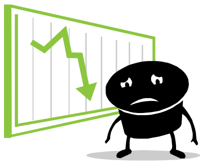
Algoritmul lui Dijkstra

- **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

- 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]()

Corectitudinea Algoritmului lui Dijkstra

Corectitudine

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

  a) dacă $d[u] < \infty$, există un drum de la s la u în G de cost d[u] și acesta se poate determina din vectorul tata:

  tata[u]= predecesorul lui u pe un drum de la s la u de cost d[u]

  b) $d[u] \geq \delta(s,u)$

Corectitudine

- **Demonstrație:** Rezultă din faptul că actualizarea unei etichete d[v] corespunde extinderii unui drum deja determinat de la s la u prin adăugarea arcului uv.
- Detaliat – Inducție

Corectitudine

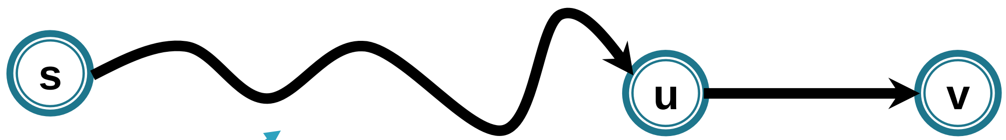
- **Demonstrație:** inducție după numărul de iterații (executii cat timp)

- Inițial d[s] = 0 = w([s]) (restul etichetelor sunt ∞)

- La prima iterație este extras din Q vârful s – pentru el afirmația se verifică

Corectitudine

- **Demonstrație:** Idee inducție

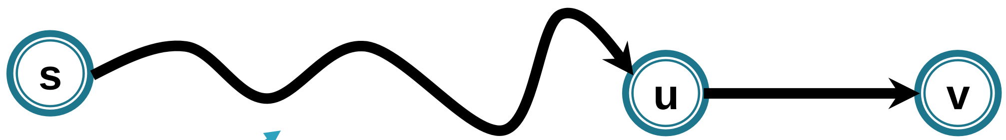


există P de la s la u cu w(P) = d[u]

relaxare uv

Corectitudine

- **Demonstrație: Idee inducție**



există P de la s la u cu w(P) = d[u]

După relaxare uv

$d[v] = d[u] + w(uv) = w(\,[s\ \underline{P}\ u;\ v])$

tata[v] = u

⇓

se verifică proprietatea și pentru v

Corectitudine

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

  a) dacă $d[u] < \infty$, există un drum de la s la u în G de cost d[u] și acesta se poate determina din vectorul tata:

  tata[u]= predecesorul lui u pe un drum de la s la u de cost d[u]

  b) $d[u] \geq \delta(s,u)$

- **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.

Corectitudine

- **Teoremă**

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

  $w : E \rightarrow \mathbb{R}_+$ și $s \in V$ fixat.

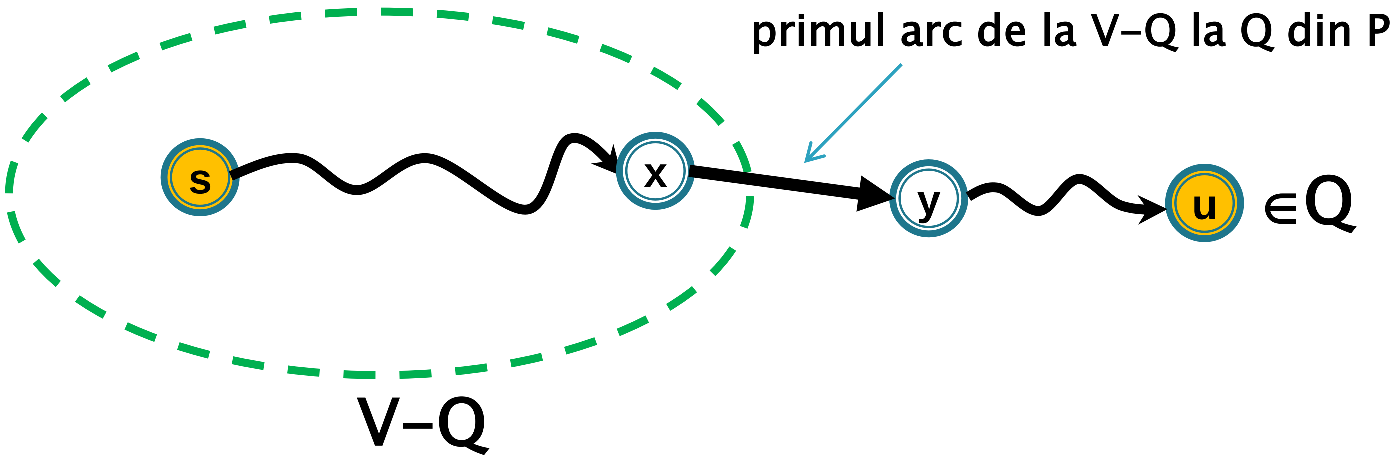
  La finalul algoritmul lui Dijkstra avem:

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

  și tata memorează un arbore al distanțelor față de s.

Corectitudine

- **Demonstraţie (idee).** Inducţie: $d[x] = \delta(s, x) \ \forall\, x \notin Q$ (=deja selectat)
- Când un vârf u este selectat: fie P un s-u drum minim



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

din modul în care este ales u

$$d[u] \le d[y] \le d[x] + w(x,y) = \delta(s,x) + w(x,y) = w(s \overset{P}{-} y)$$

$$\le w(P) \le d[u]$$

ipoteza de inductie pentru x

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

Drumuri maxime

- Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la s la celelalte vârfuri?

Drumuri maxime

Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la s la celelalte vârfuri

- Modificăm astfel doar inițializarea distanțelor (cu -∞ în loc de + ∞) și inversam condiția de la relaxarea arcelor pentru a calcula maxim în loc de minim

Corectitudine – probabil similar cu Dijkstra?!!

Drumuri maxime

Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la S la celelalte vârfuri

- Modificăm astfel doar inițializarea distanțelor (cu -∞ în loc de + ∞) și inversam condiția de la relaxarea arcelor pentru a calcula maxim în loc de minim

- **Corectitudine** – probabil similar cu Dijkstra?!!