Implementarea algoritmului Ford-Fulkerson

Algoritmul Ford–Fulkerson

- **Cum determinăm un lanț f-nesaturat?**

Algoritmul Ford–Fulkerson

- **Spre exemplu, prin parcurgerea grafului, pornind din vârful s și considerând doar arce cu capacitatea reziduală pozitivă** (în raport cu lanțurile construite prin parcurgere, memorate cu vectorul tata)

  = **s-t drum în graful rezidual**

Algoritmul Ford–Fulkerson

- **Spre exemplu, prin parcurgerea grafului, pornind din vârful s și considerând doar arce cu capacitatea reziduală pozitivă** (în raport cu lanțurile construite prin parcurgere, memorate cu vectorul tata)

Parcurgerea BF ⇒

determinăm s-t lanțuri f-nesaturate de lungime minimă

⇒ Algoritmul EDMONDS-KARP

= Ford-Fulkerson, în care lanțul P ales la un pas are lungime minimă

Algoritmul Ford–Fulkerson

- **Spre exemplu, prin parcurgerea grafului, pornind din vârful s și considerând doar arce cu capacitatea reziduală pozitivă** (în raport cu lanțurile construite prin parcurgere, memorate cu vectorul tata)

Alte criterii de construcție lanț ⇒ alți algoritmi

Algoritmul Edmonds-Karp

Implementare

Schema:

```
inițializează_flux_nul()
cât timp (construiește_s-t_lanț_nesat_BF() == true) execută
        revizuiește_flux_lanț()
afișează_flux()
```

Implementare

<u>Schema:</u>

inițializează_flux_nul()

cât timp (construiește_s-t_lanț_nesat_BF() == **true**) **execută**

    revizuiește_flux_lanț()

afișează_flux()

Amintim: a determina un s-t lanț nesaturat folosind BF în G ⇔ a determina un s-t drum folosind BF în graful rezidual G_f

Varianta 1 de implementare

revizuirea fluxului folosind s-t lanțuri din G
(fără a folosi graful rezidual)

Implementare – Varianta 1

construiește_s-t_lanț_nesat_BF() - construiește un s-t lanț nesaturat, prin parcurgerea BF din s

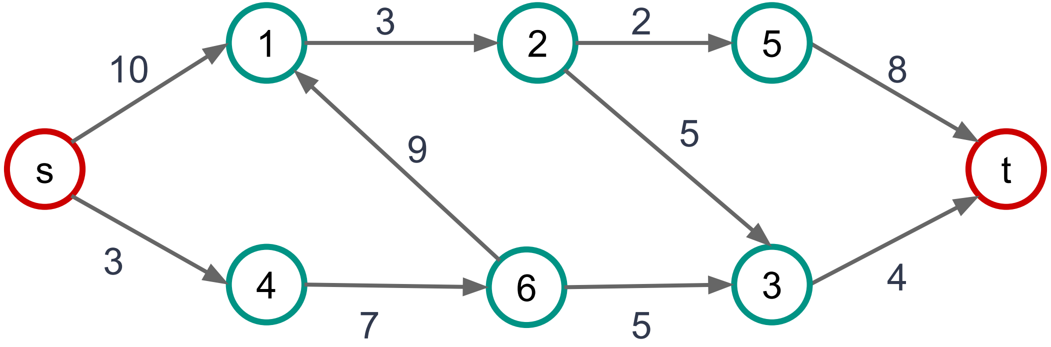
- sunt considerate în parcurgere **doar arce pe care se poate modifica fluxul**, adică având capacitate reziduală pozitivă
- returnează **false** dacă un astfel de lanț nu există (și **true** dacă l-a putut construi)

Implementare – Varianta 1

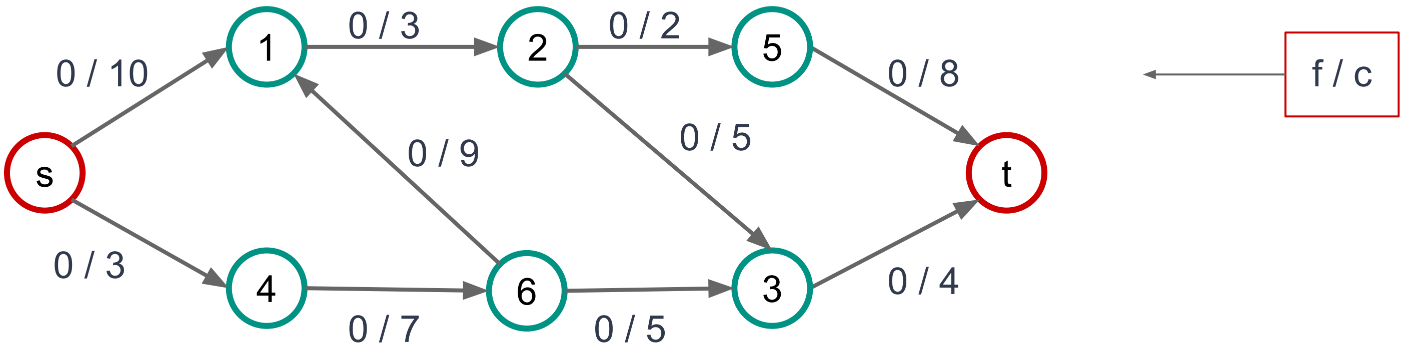
revizuiește_flux_lanț()

- fie P s-t lanțul găsit în **construiește_s-t_lanț_nesat_BF()**
- calculăm **i(P)**
- pentru fiecare arc *e* al lanțului P
  - **creștem** cu i(P) fluxul pe *e* dacă este **arc direct**
  - **scădem** cu i(P) fluxul pe *e* dacă este arc invers

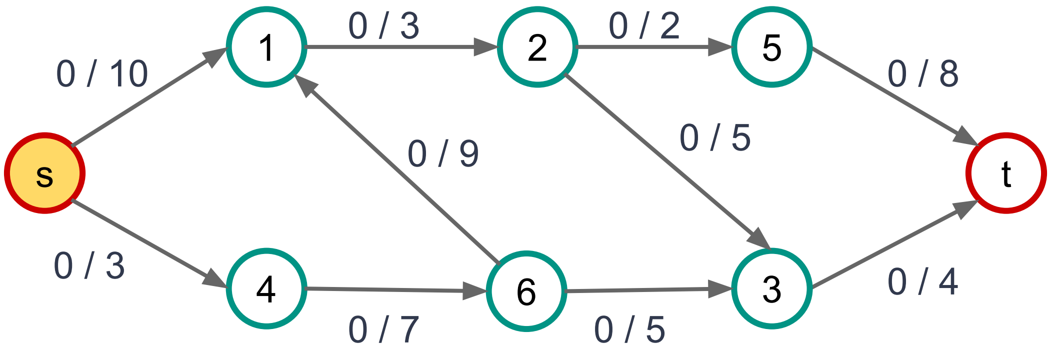
Exemplu
Algoritmul Edmonds-Karp

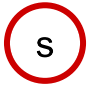

s
1
2
5
t
4
6
3
10
3
3
2
8
9
5
7
5
4

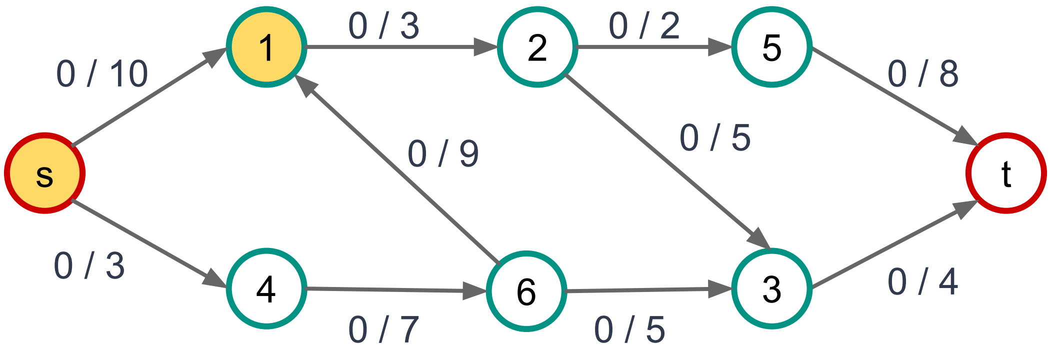
inițializează_flux_nul()

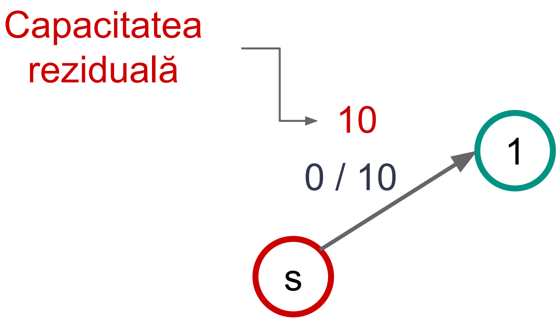

0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 7
0 / 5
0 / 4
s
1
2
5
t
4
6
3
f / c

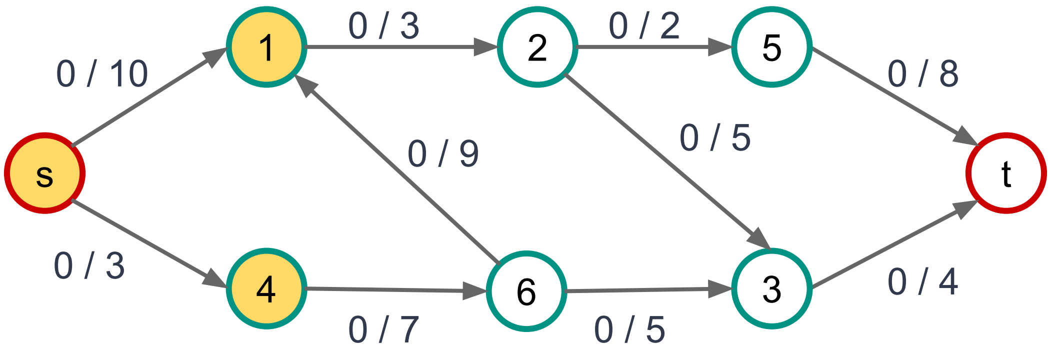
construiește_s-t_lanț_nesat_BF()

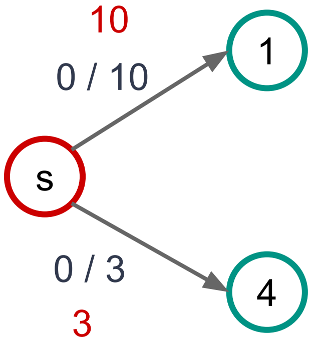

0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 4
0 / 7
0 / 5
s
1
2
5
t
4
6
3

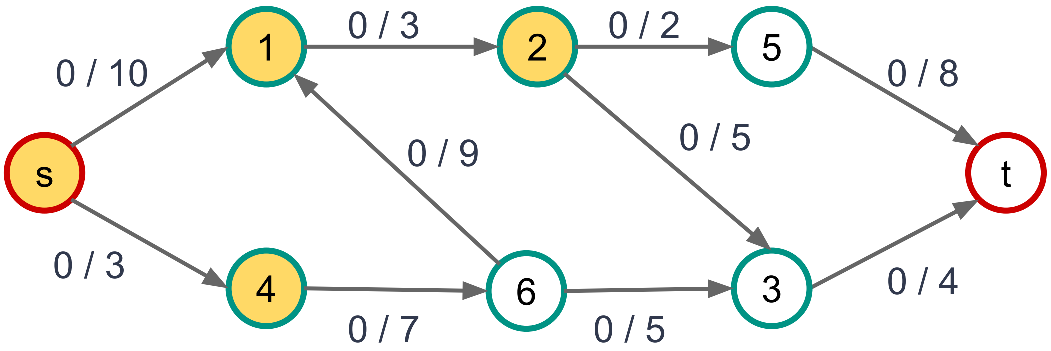

s

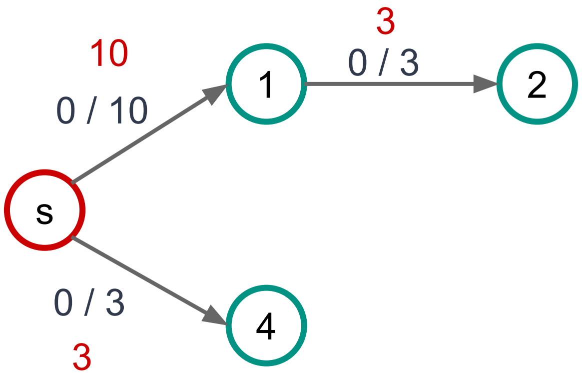

0 / 10
0 / 3
0 / 9
0 / 3
0 / 2
0 / 5
0 / 8
0 / 4
0 / 7
0 / 5
s
1
2
5
t
4
6
3

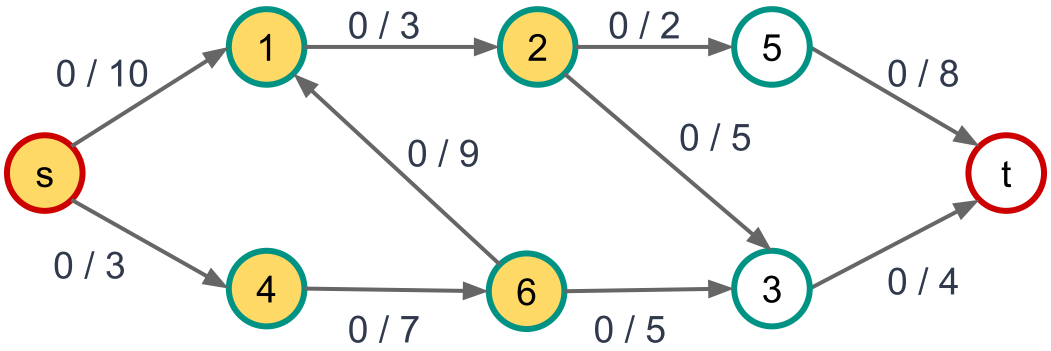

Capacitatea reziduală
10
0 / 10
1
s

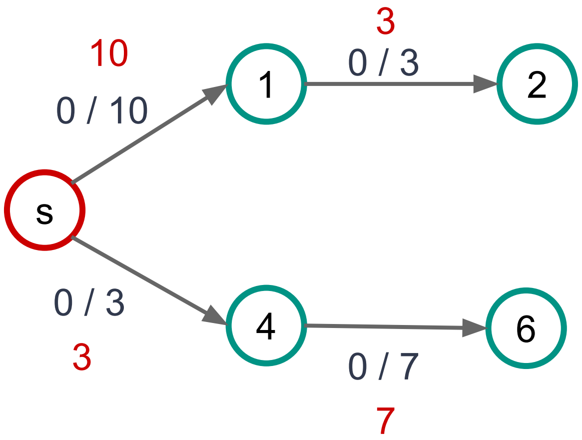

s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 7
0 / 5
0 / 4

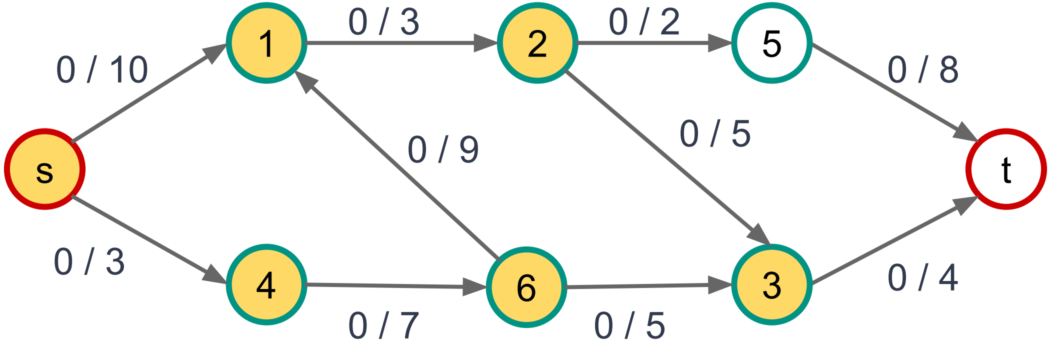

10
0 / 10
1
s
0 / 3
3
4

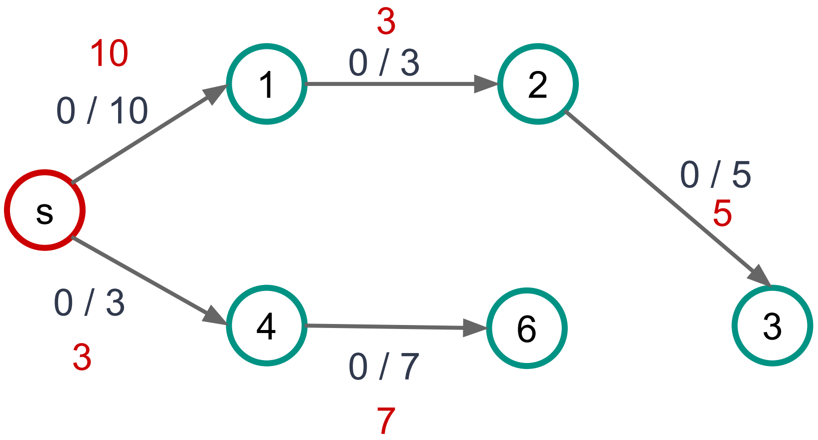

s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 7
0 / 5
0 / 4

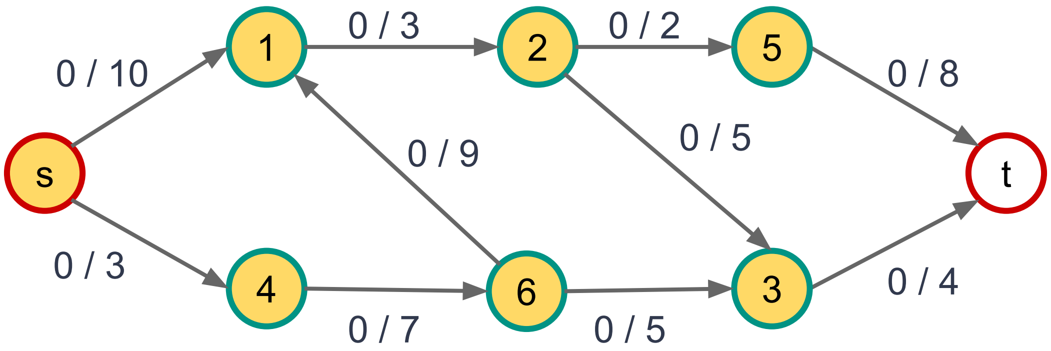

10
0 / 10
3
0 / 3
0 / 3
3
s
1
2
4


s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 7
0 / 5
0 / 4

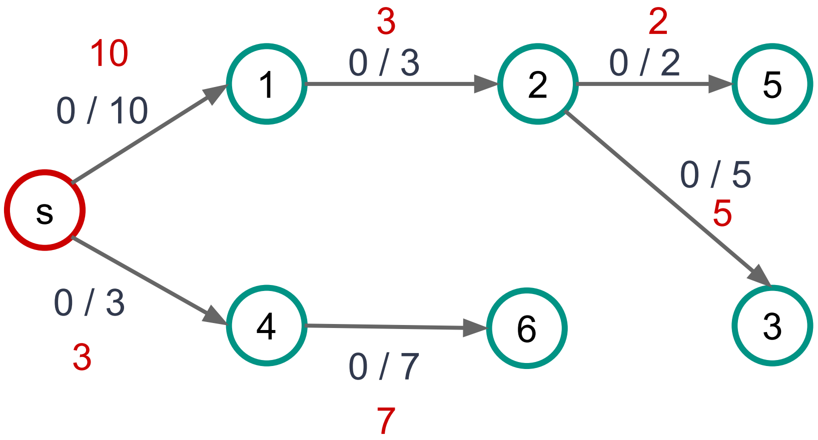

s
1
2
4
6
10
0 / 10
3
0 / 3
0 / 3
3
0 / 7
7


s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 3
0 / 7
0 / 5
0 / 4

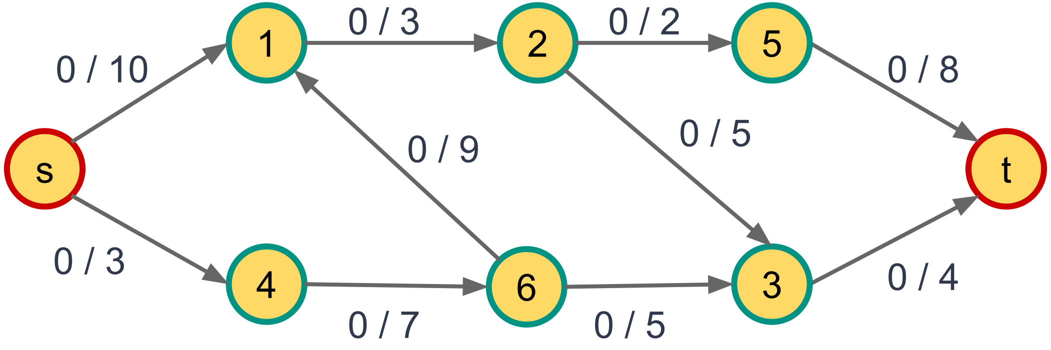

s
1
2
4
6
3
10
0 / 10
3
0 / 3
0 / 5
5
0 / 3
3
0 / 7
7

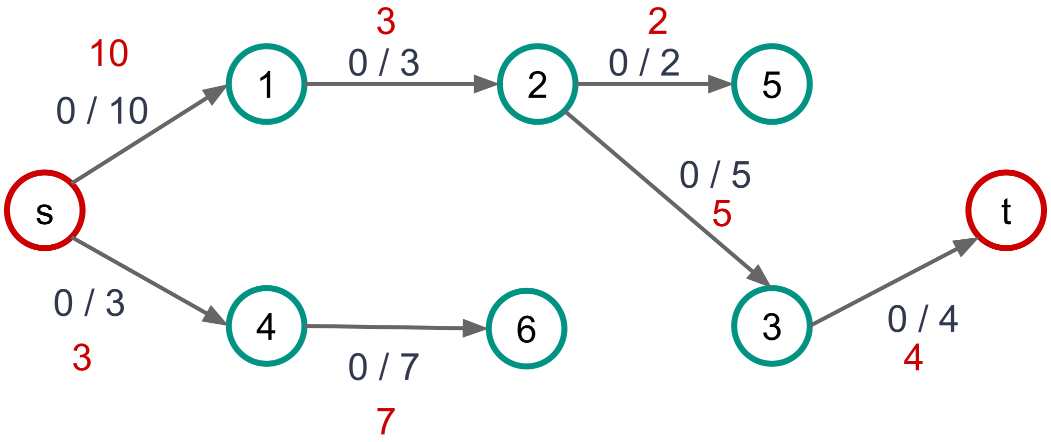


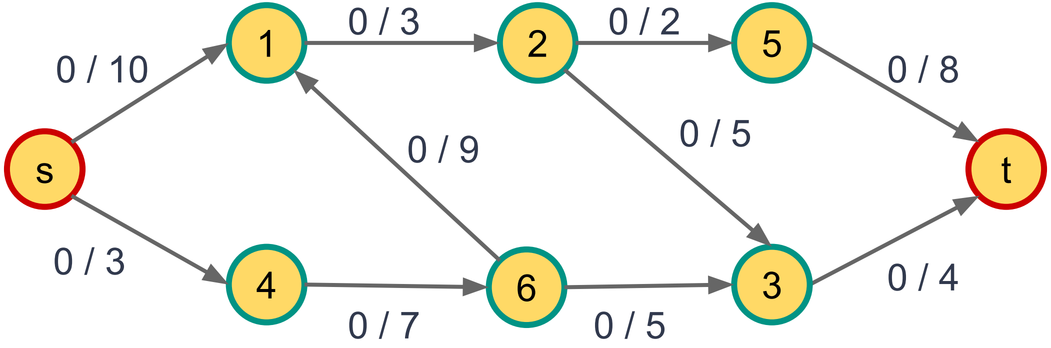
s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 7
0 / 5
0 / 4

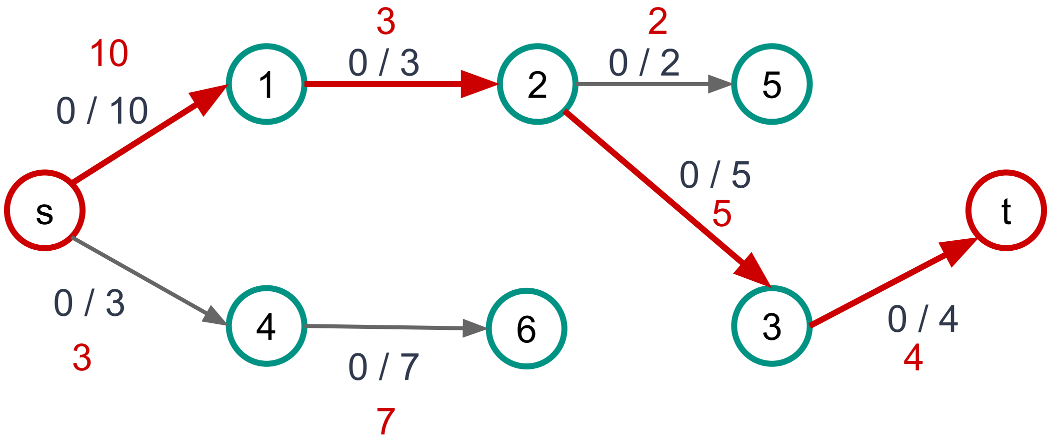


s
1
2
5
4
6
3
10
0 / 10
3
0 / 3
2
0 / 2
0 / 5
5
0 / 3
3
0 / 7
7

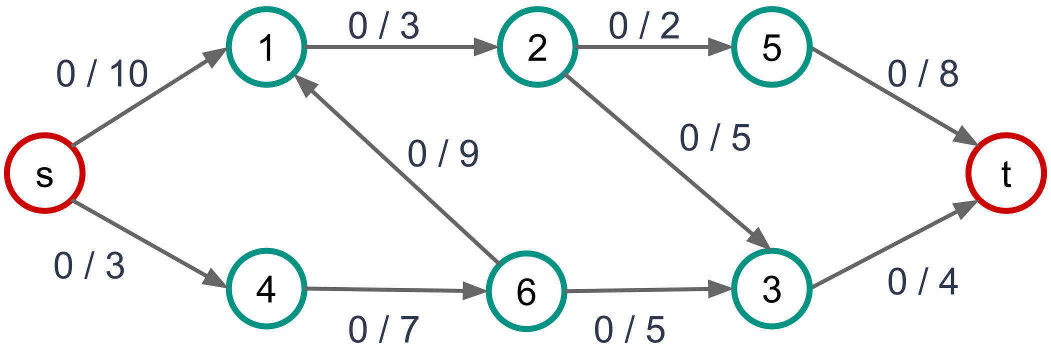

0 / 10
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 3
0 / 7
0 / 5
0 / 4
s
1
2
5
t
4
6
3

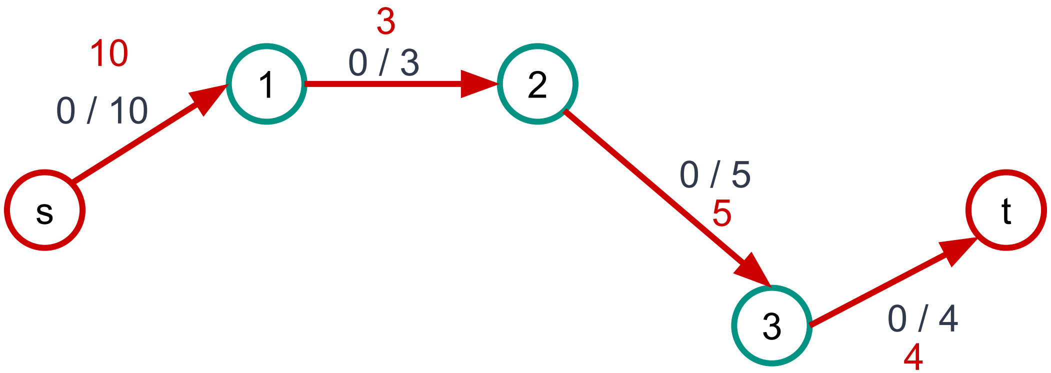

10
3
2
0 / 10
0 / 3
0 / 2
0 / 5
5
0 / 3
3
0 / 7
7
0 / 4
4
s
1
2
5
t
4
6
3


s
1
2
5
t
4
6
3
0 / 10
0 / 3
0 / 2
0 / 8
0 / 9
0 / 5
0 / 3
0 / 7
0 / 5
0 / 4

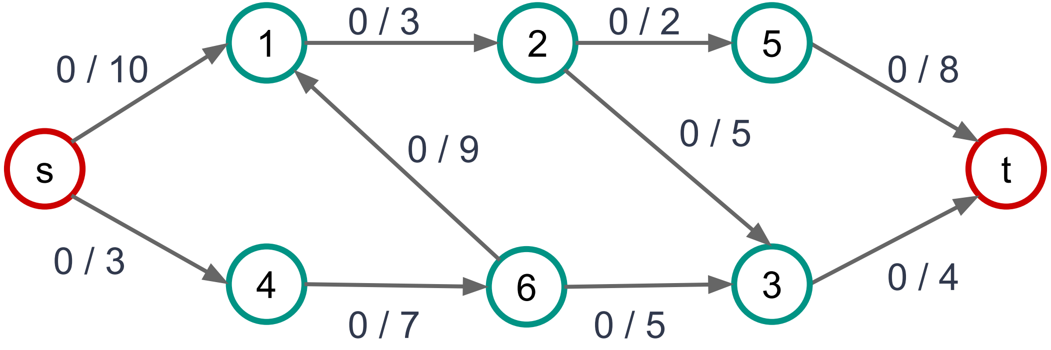

s
1
2
5
t
4
6
3
10
0 / 10
3
0 / 3
2
0 / 2
0 / 5
5
0 / 3
3
0 / 7
7
0 / 4
4

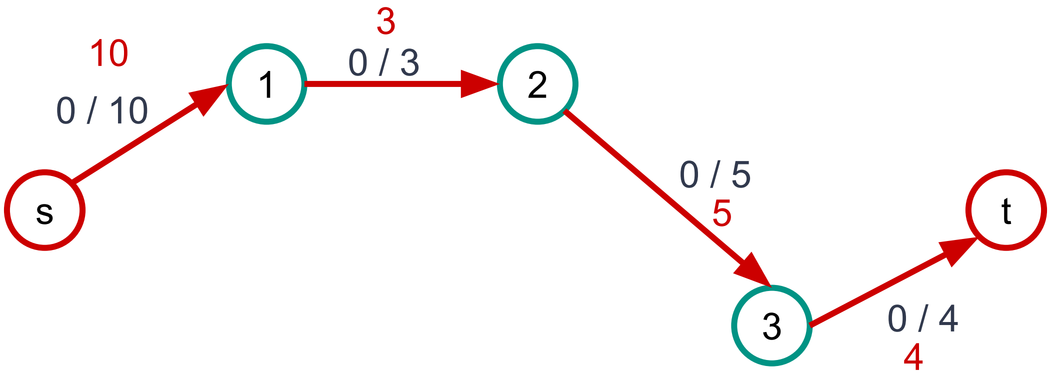
revizuiește_flux_lanț()



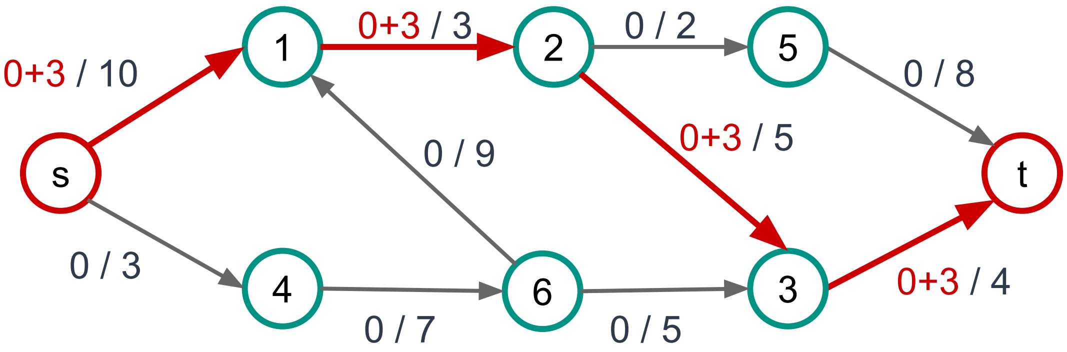


i(P) = ?

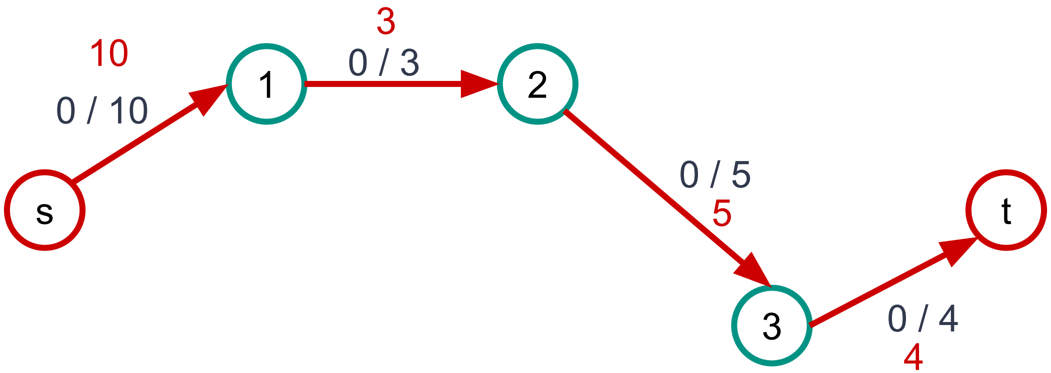




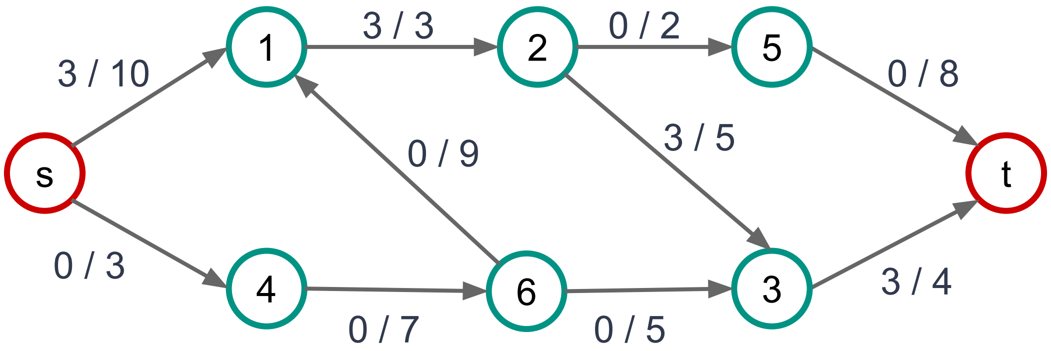
i(P) = min { 10, 3, 5, 4 } = 3



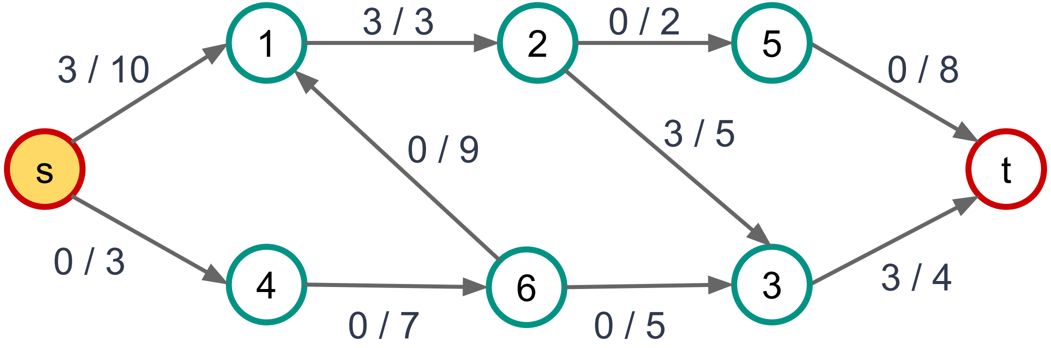
Revizuire flux

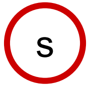


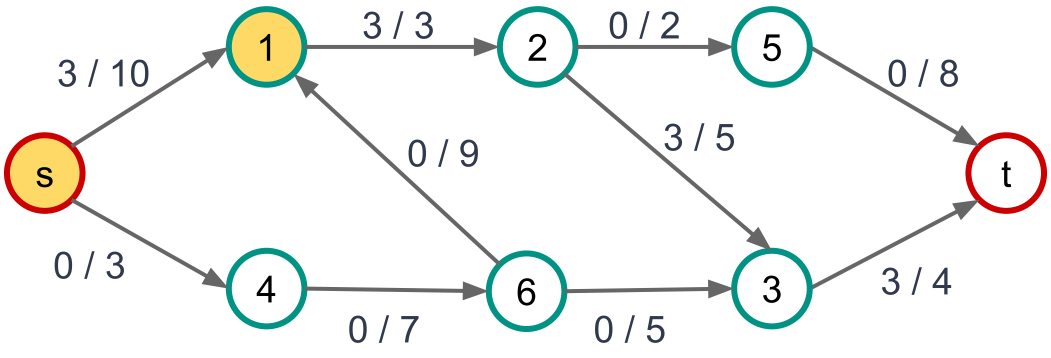
i(P) = min { 10, 3, 5, 4 } = 3

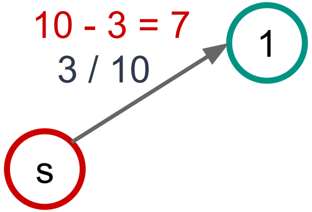

s
1
2
5
t
4
6
3
3 / 10
0 / 3
3 / 3
0 / 9
0 / 7
0 / 2
3 / 5
0 / 5
0 / 8
3 / 4

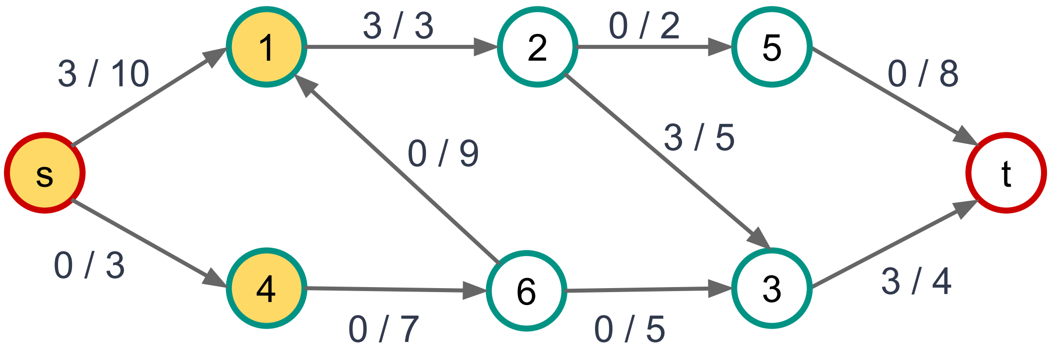
construiește_s-t_lanț_nesat_BF()


s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4

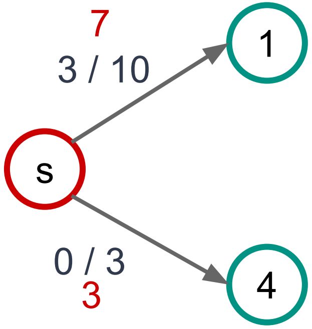

s


3 / 10
1
3 / 3
2
0 / 2
5
0 / 8
s
0 / 9
3 / 5
t
0 / 3
4
0 / 7
6
0 / 5
3
3 / 4

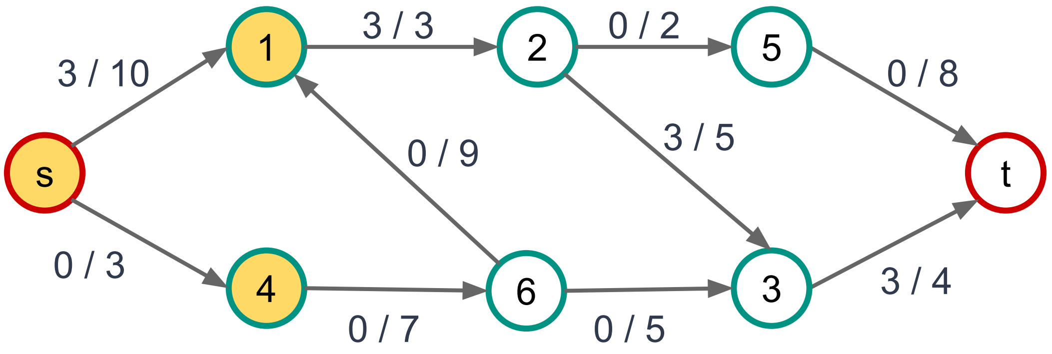

10 - 3 = 7
3 / 10
1
s



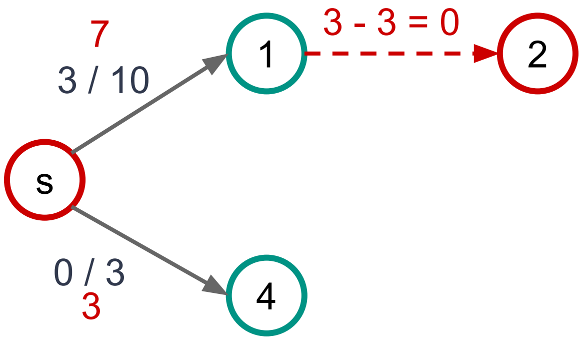
s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4



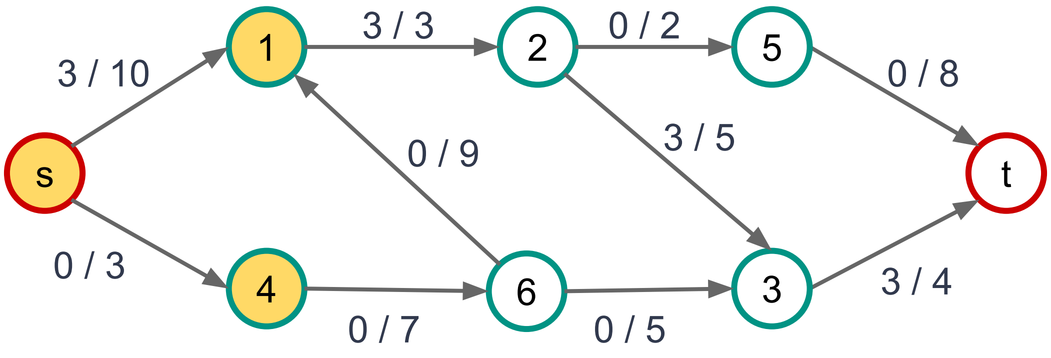
7
3 / 10
1
s
0 / 3
3
4

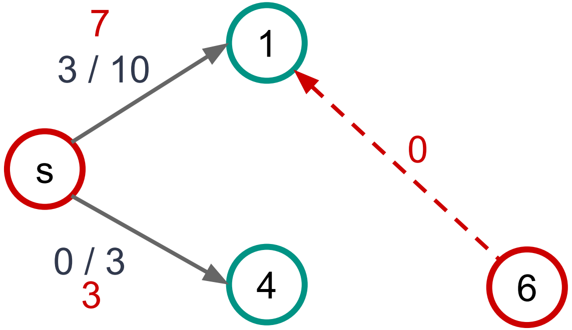


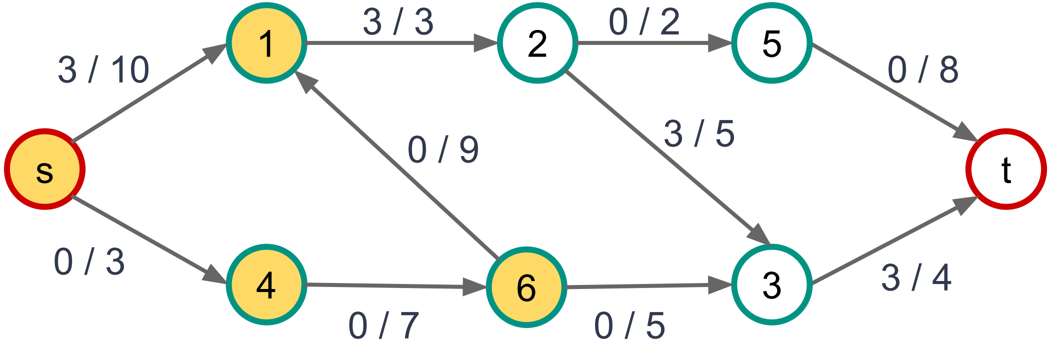
s
1
2
5
t
4
6
3
3 / 10
0 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 7
0 / 5
3 / 4

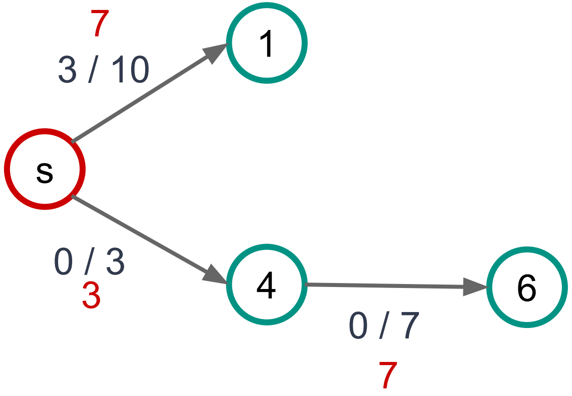


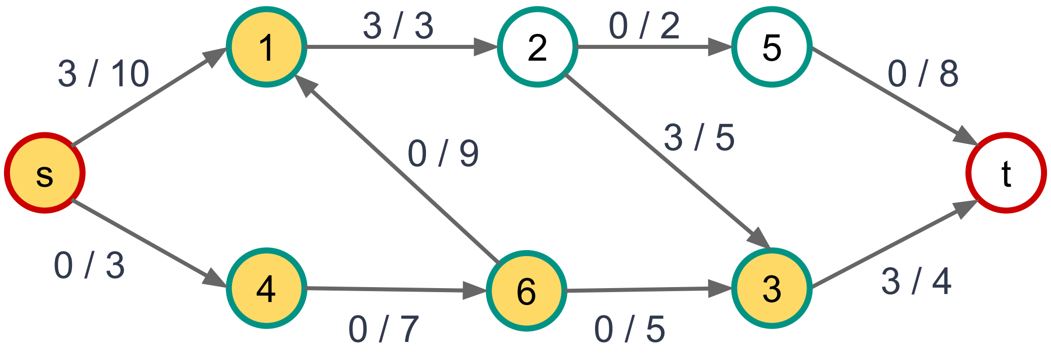
7
3 / 10
3 - 3 = 0
s
1
2
4
0 / 3
3


s
1
2
5
t
4
6
3
3 / 10
0 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 7
0 / 5
3 / 4

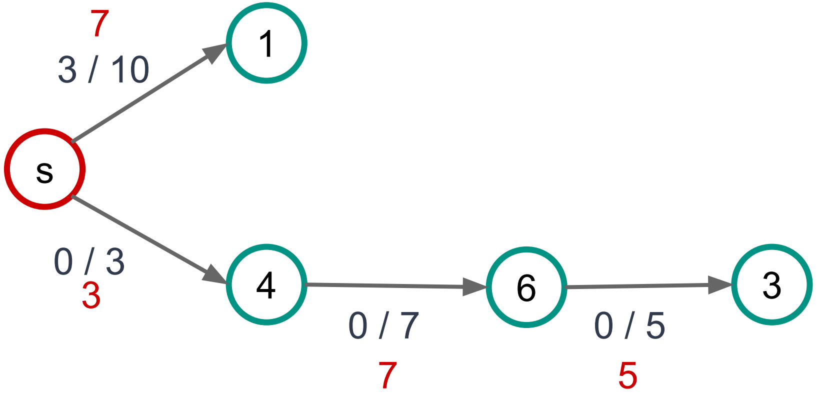

7
3 / 10
0 / 3
3
0
s
1
4
6


3 / 10
0 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 7
0 / 5
3 / 4
s
1
2
5
t
4
6
3

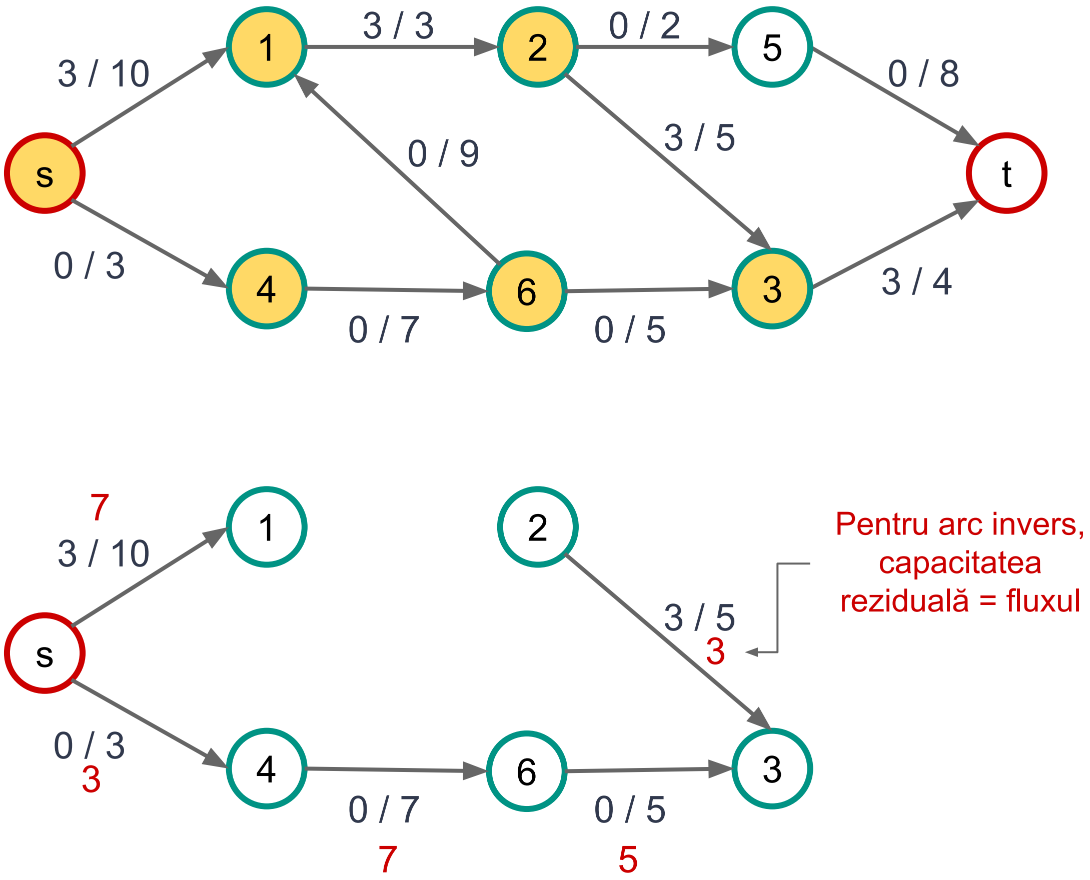

7
3 / 10
0 / 3
3
0 / 7
7
s
1
4
6



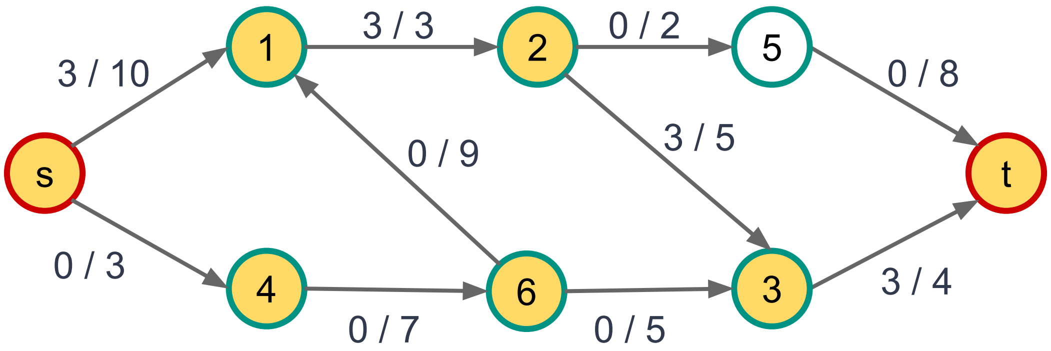
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4
s
1
2
5
t
4
6
3



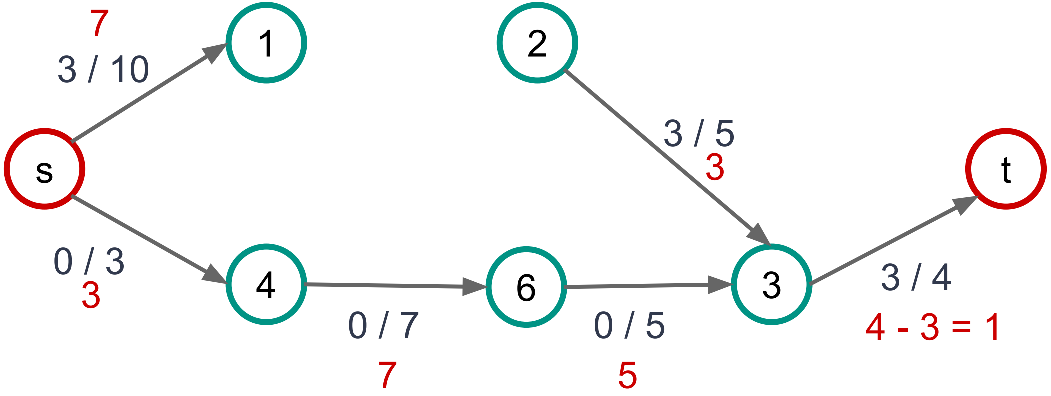
7
3 / 10
0 / 3
3
0 / 7
7
0 / 5
5
s
1
4
6
3



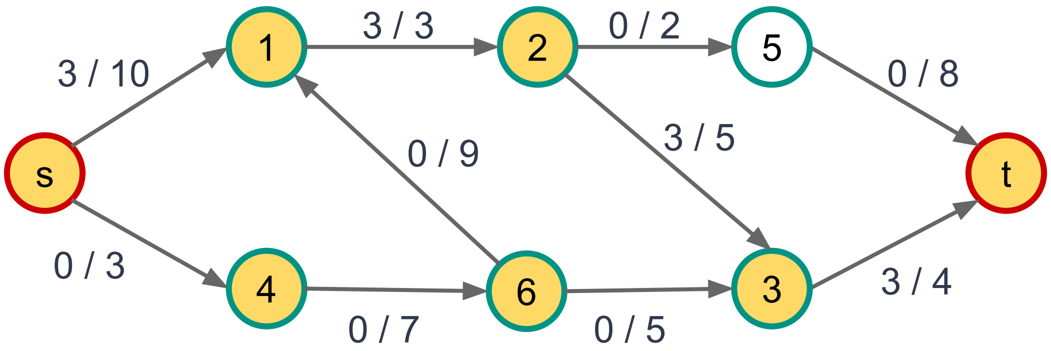
s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4
7
3 / 10
3 / 5
3
0 / 3
3
0 / 7
7
0 / 5
5
Pentru arc invers,
capacitatea
reziduală = fluxul



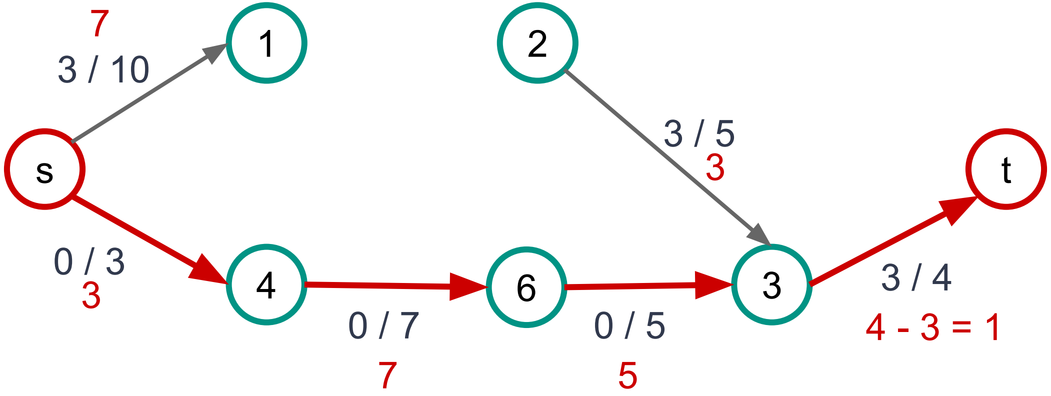
s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4



s
1
2
t
4
6
3
7
3 / 10
3 / 5
3
0 / 3
3
0 / 7
7
0 / 5
5
3 / 4
4 - 3 = 1

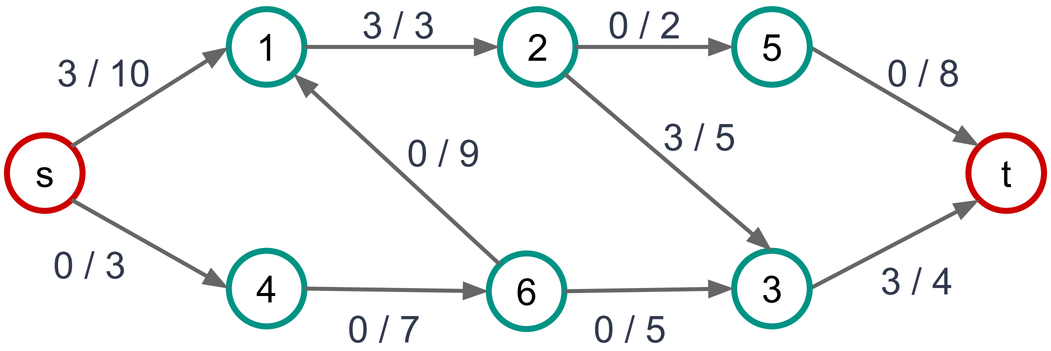


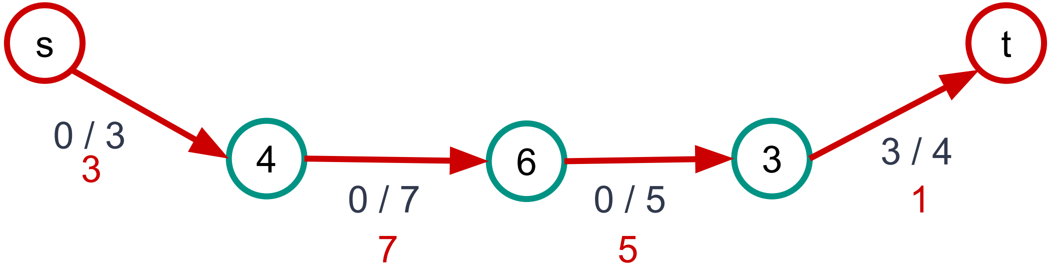
s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
0 / 3
0 / 7
0 / 5
3 / 4



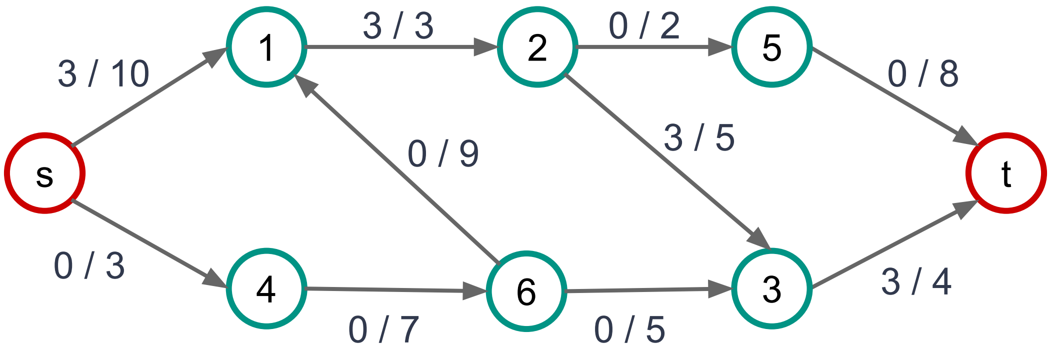
s
1
2
t
4
6
3
7
3 / 10
3 / 5
3
0 / 3
3
0 / 7
7
0 / 5
5
3 / 4
4 - 3 = 1

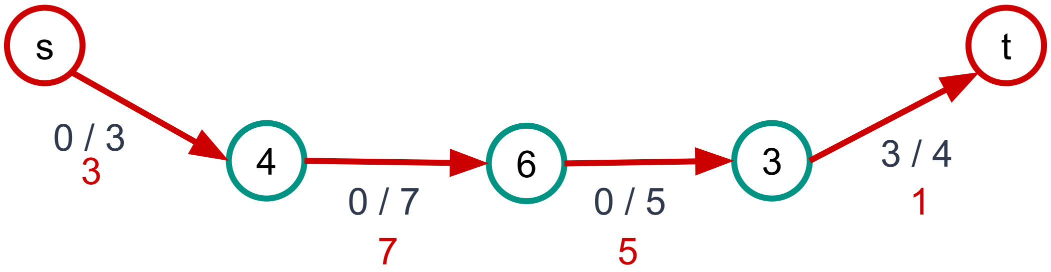
revizuiește_flux_lanț()



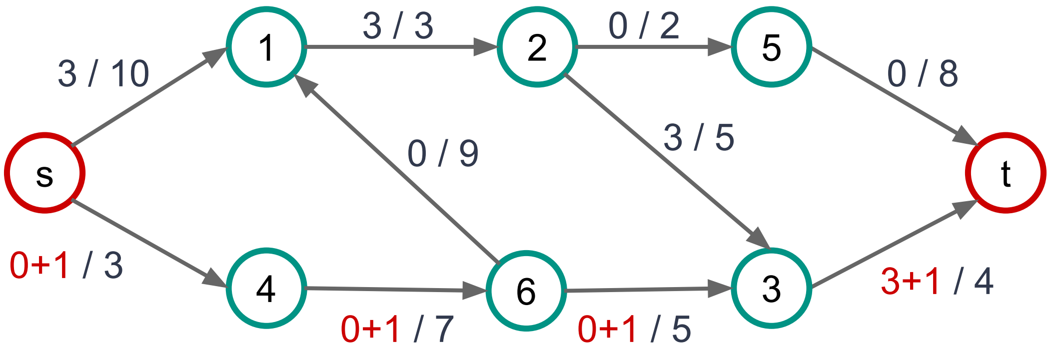


i(P) = ?

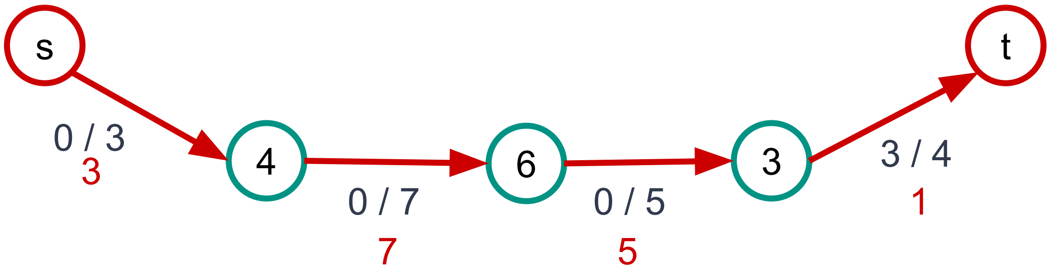




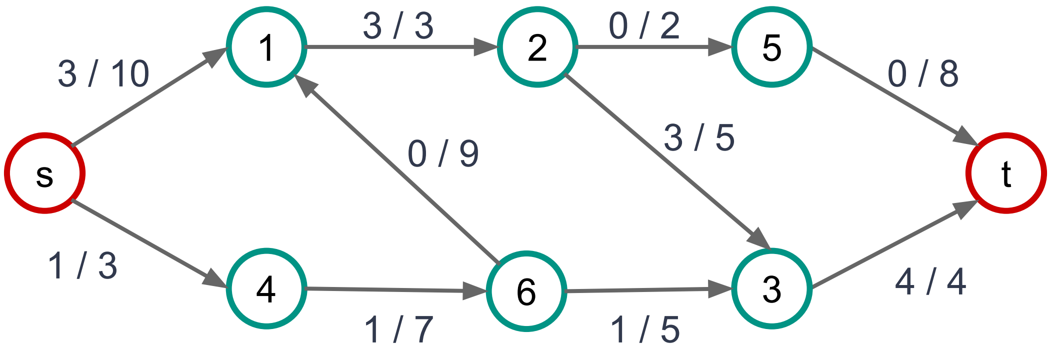
i(P) = min { 3, 7, 5, 1 } = 1



Revizuire flux

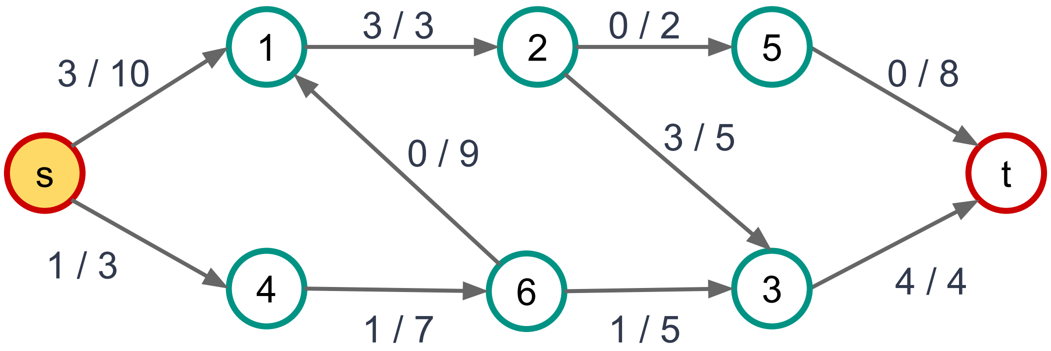


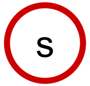
i(P) = min { 3, 7, 5, 1 } = 1

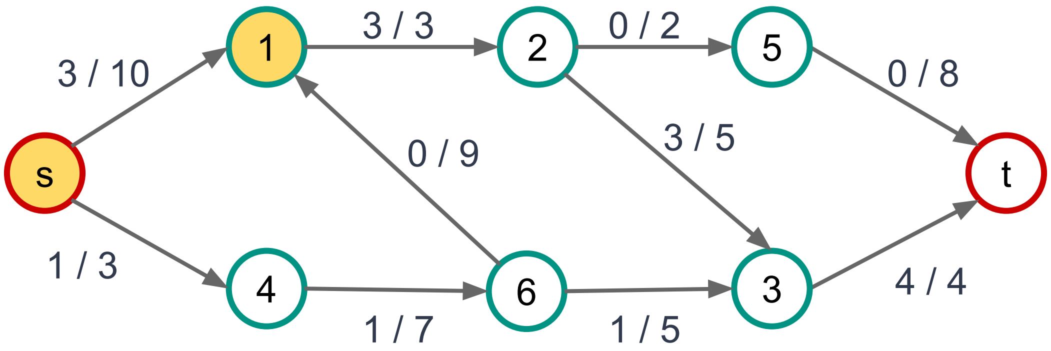


Revizuire flux

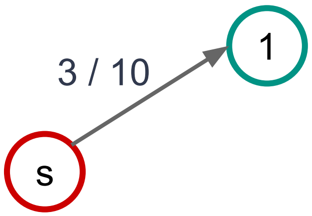
construiește_s-t_lanț_nesat_BF()


3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
4 / 4
1 / 7
1 / 5
s
1
2
5
t
4
6
3

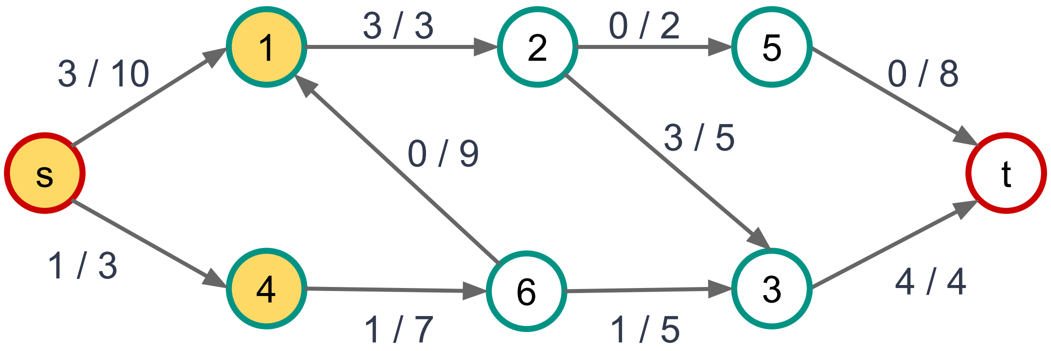

s

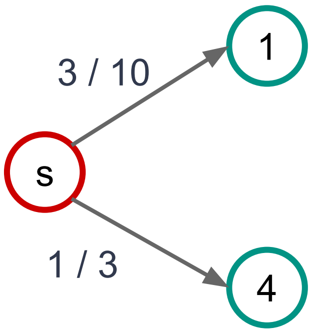


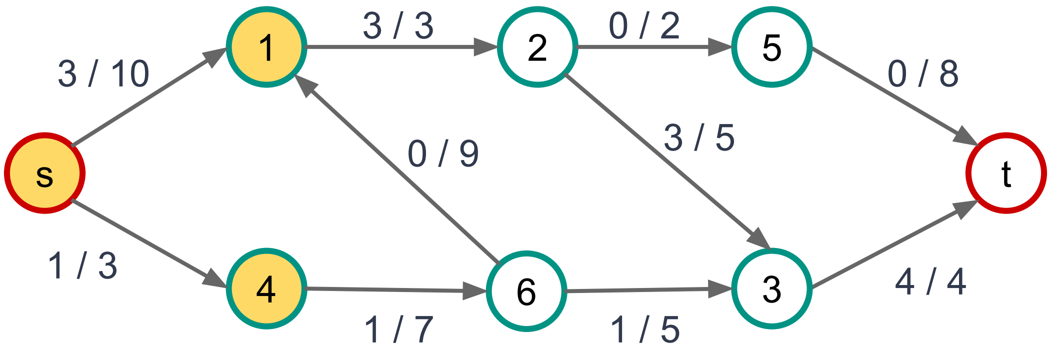
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4
s
1
2
5
t
4
6
3

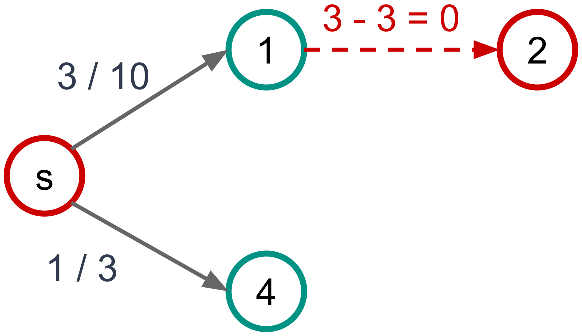


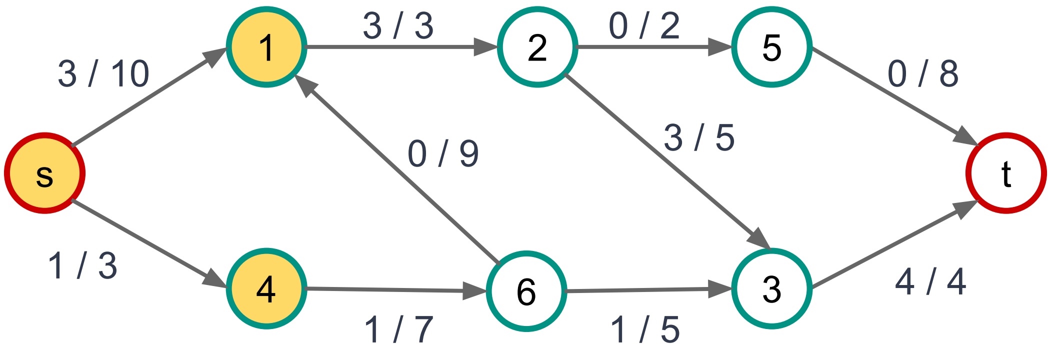
3 / 10
1
s

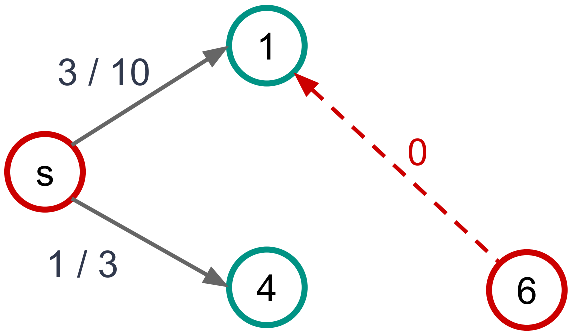

3 / 10
1 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 7
1 / 5
4 / 4
s
1
2
5
t
4
6
3

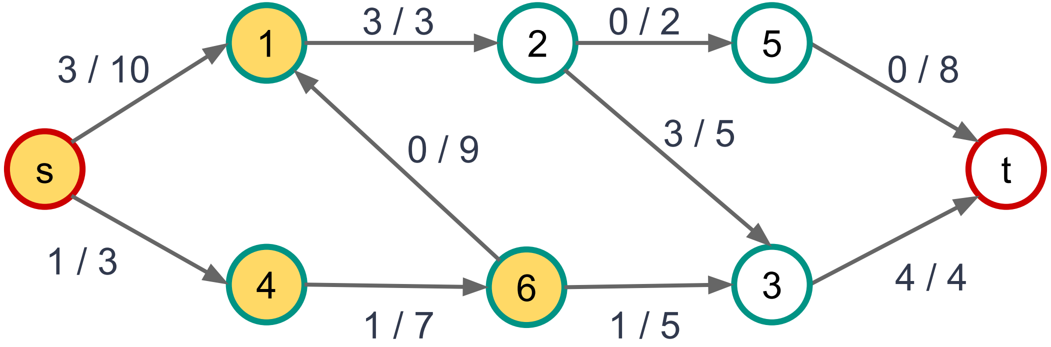

3 / 10
1 / 3
s
1
4

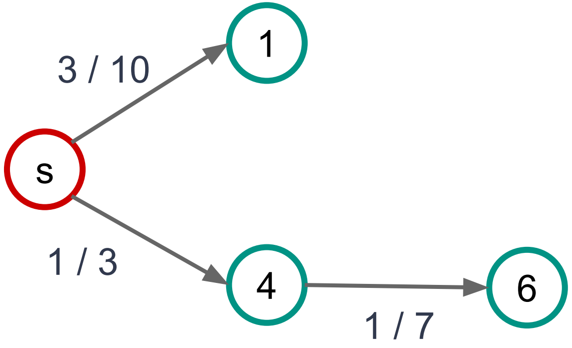

3 / 3
0 / 2
3 / 10
1
2
5
0 / 8
0 / 9
3 / 5
s
t
1 / 3
4
6
3
4 / 4
1 / 7
1 / 5

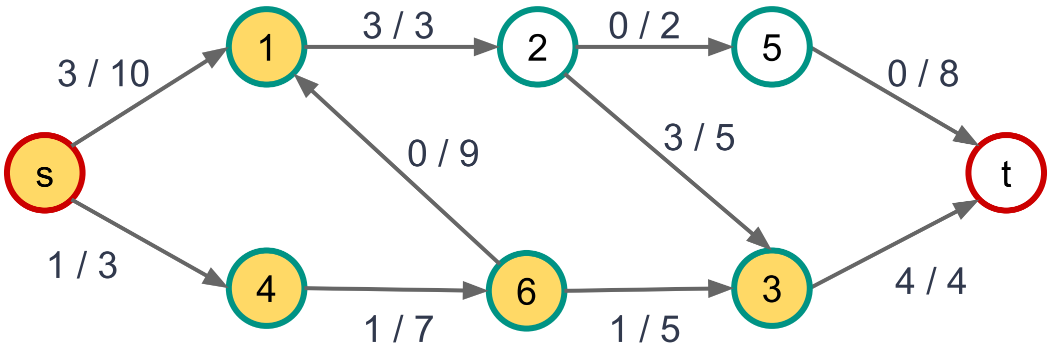

3 - 3 = 0
3 / 10
1
2
s
1 / 3
4


s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4

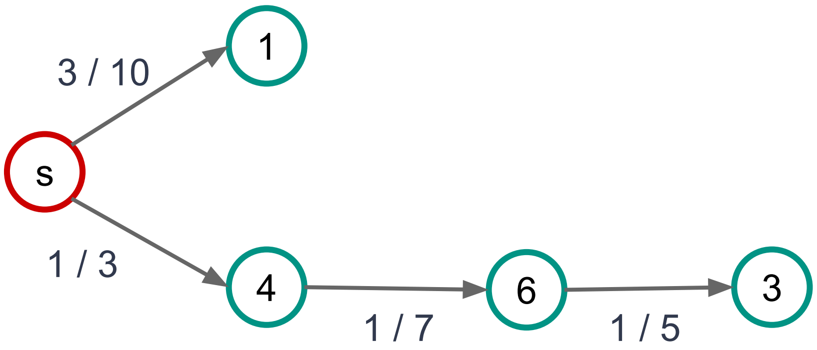

s
1
4
6
3 / 10
1 / 3
0


s
1
2
5
t
4
6
3
3 / 10
1 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 7
1 / 5
4 / 4

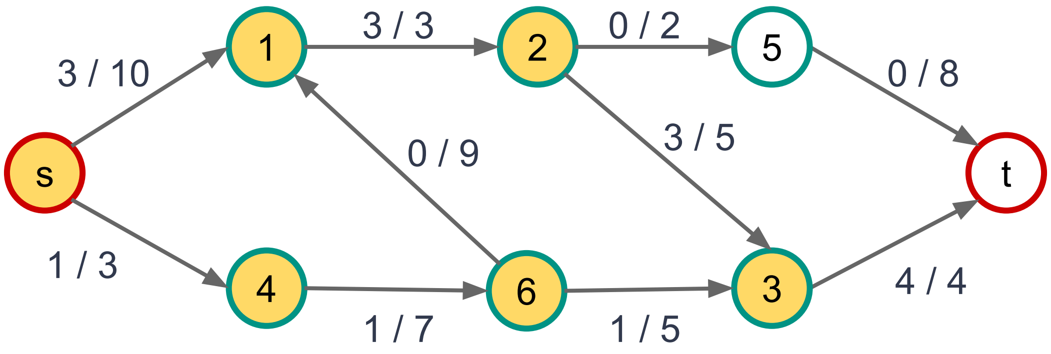

s
1
4
6
3 / 10
1 / 3
1 / 7

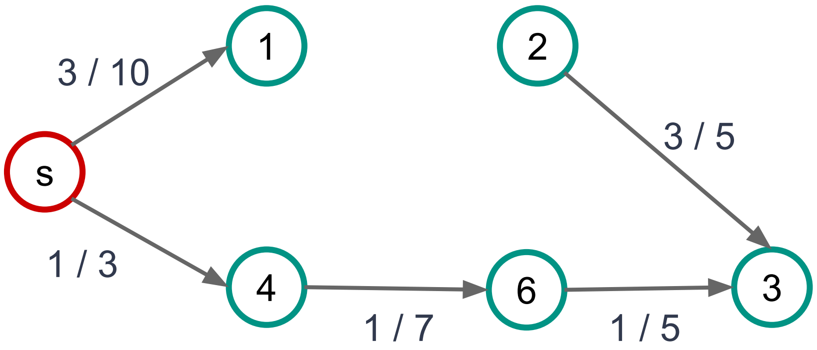


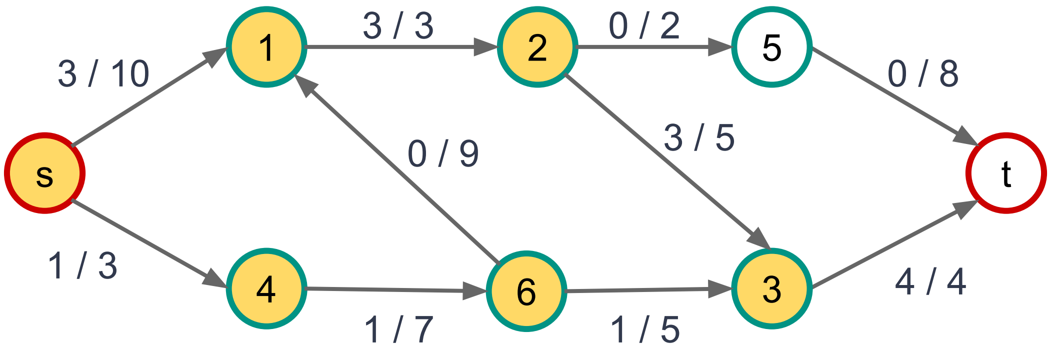
s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4

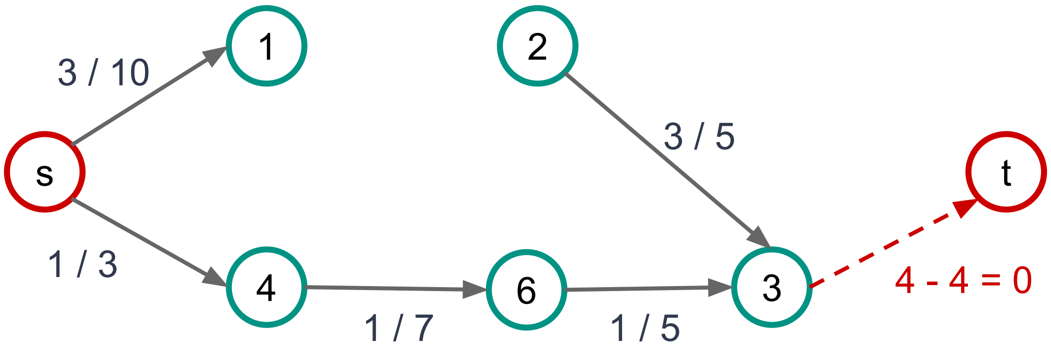


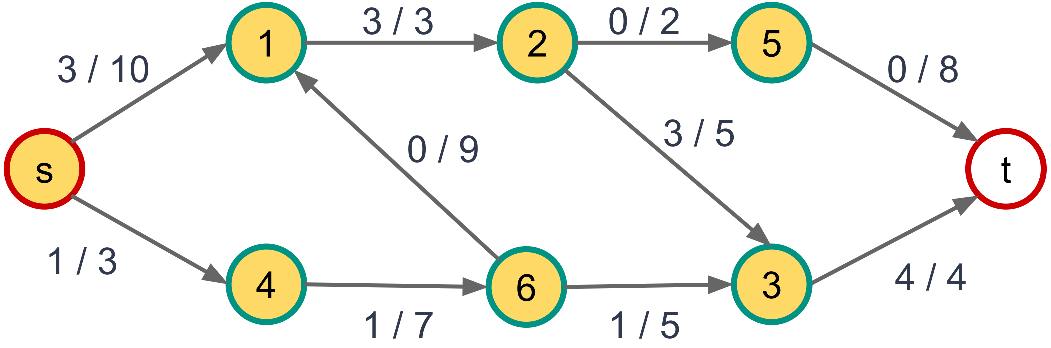
s
1
4
6
3
3 / 10
1 / 3
1 / 7
1 / 5

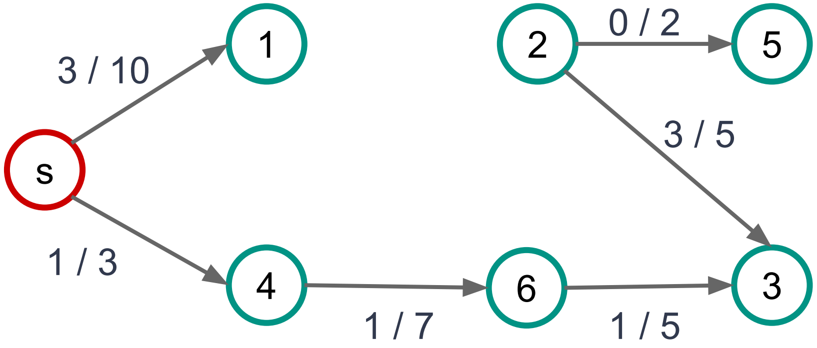

3 / 10
1 / 3
3 / 3
0 / 9
1 / 7
0 / 2
3 / 5
1 / 5
0 / 8
4 / 4
s
1
2
5
t
4
6
3

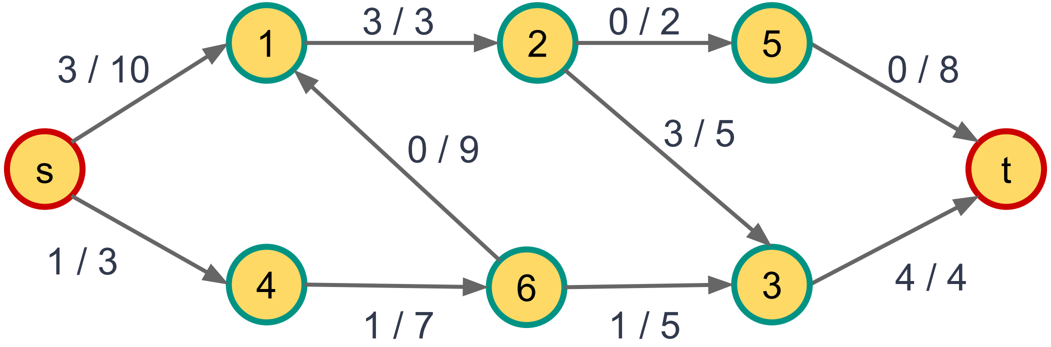

3 / 10
1 / 3
1 / 7
3 / 5
1 / 5
s
1
2
4
6
3


s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4

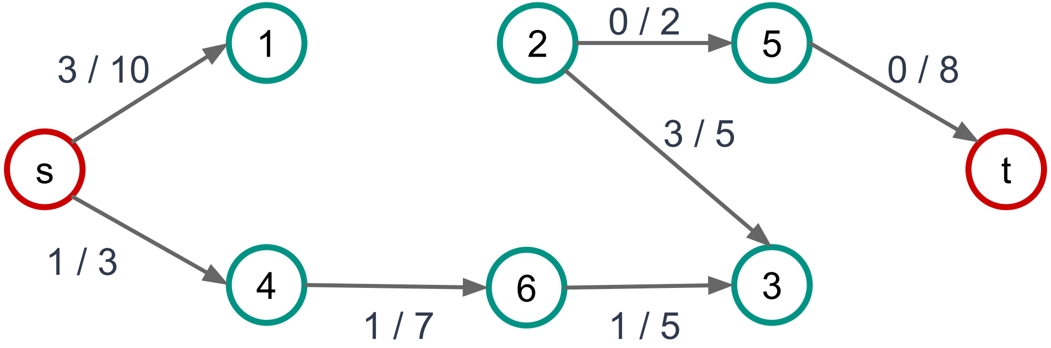

s
1
2
t
4
6
3
3 / 10
3 / 5
1 / 3
1 / 7
1 / 5
4 - 4 = 0


s
1
2
5
t
4
6
3
3 / 10
1 / 3
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 7
1 / 5
4 / 4

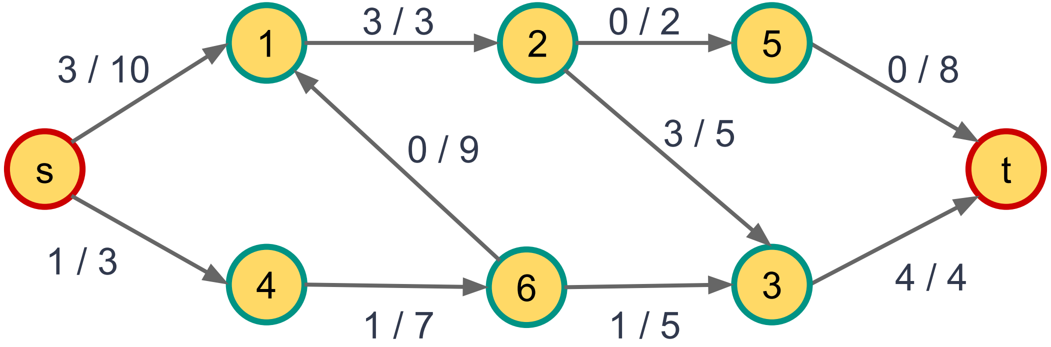

s
1
2
5
4
6
3
3 / 10
1 / 3
0 / 2
3 / 5
1 / 7
1 / 5

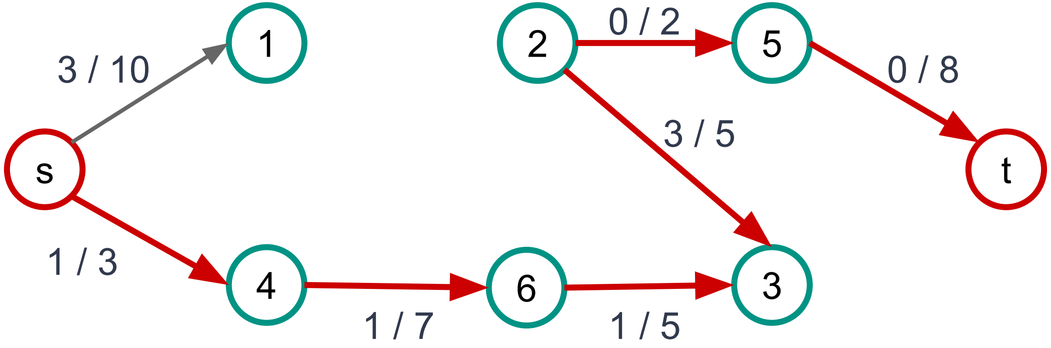


s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4

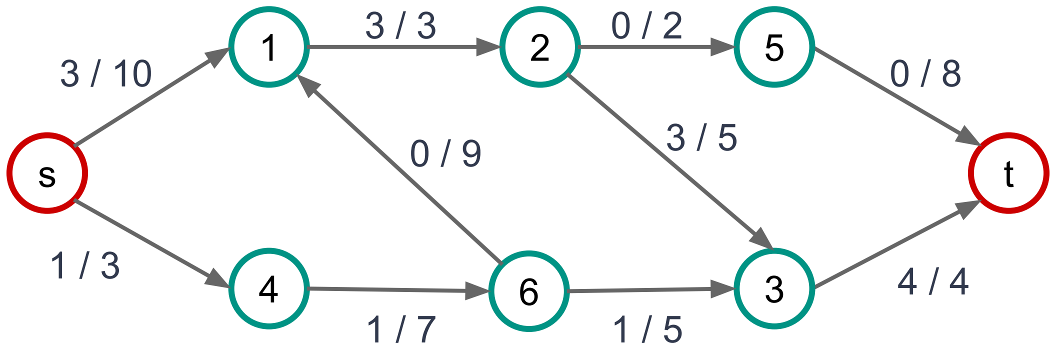


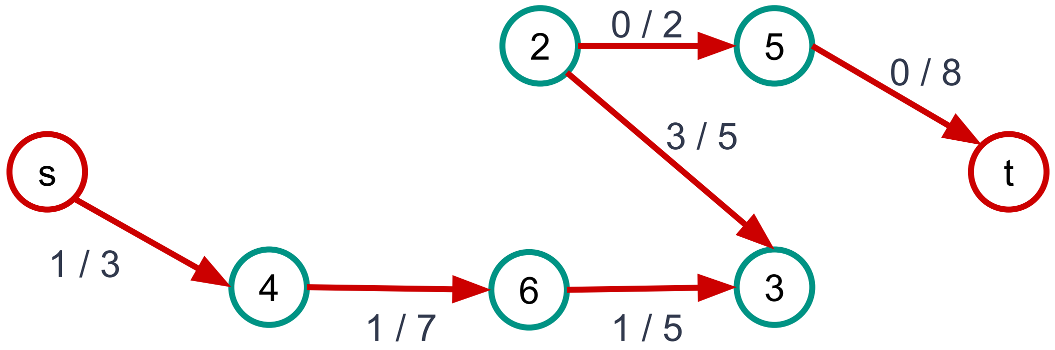
s
1
2
5
t
4
6
3
3 / 10
0 / 2
0 / 8
3 / 5
1 / 3
1 / 7
1 / 5

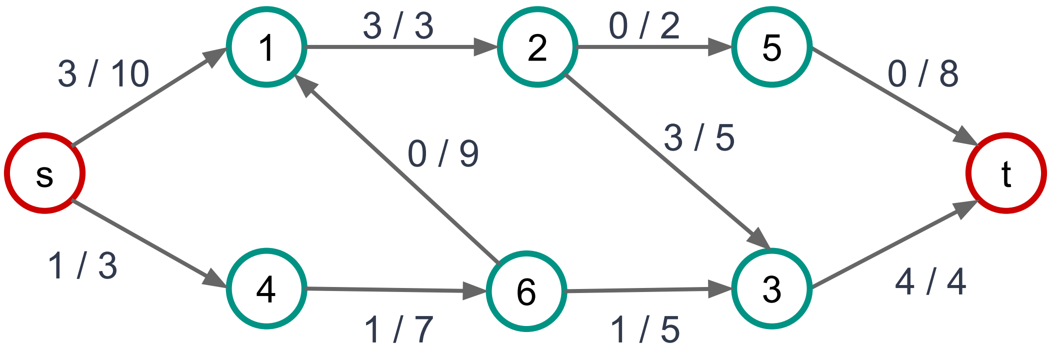

s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4

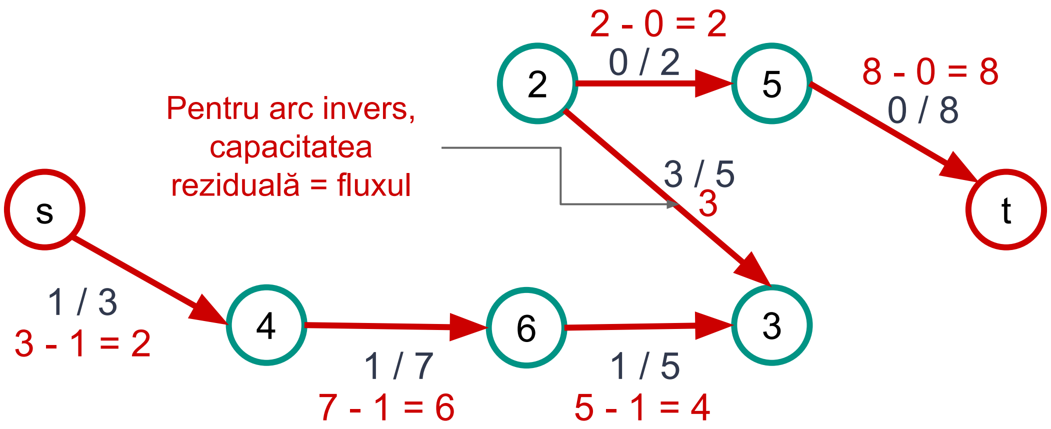

s
1
2
5
t
4
6
3
3 / 10
0 / 2
0 / 8
3 / 5
1 / 3
1 / 7
1 / 5

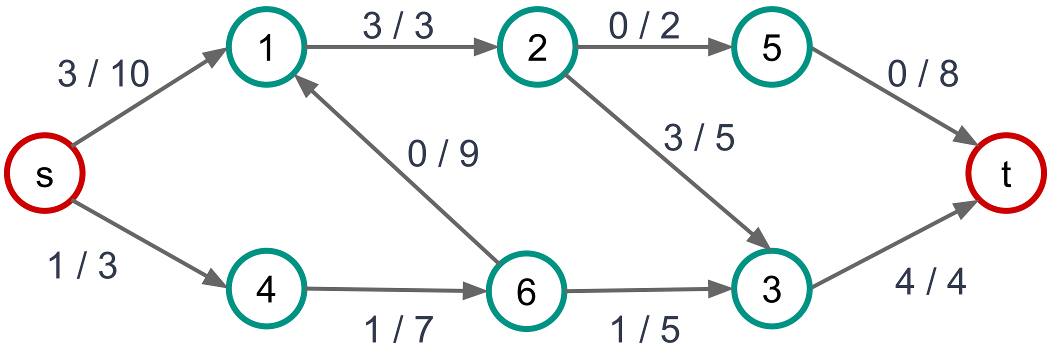
revizuiește_flux_lanț()

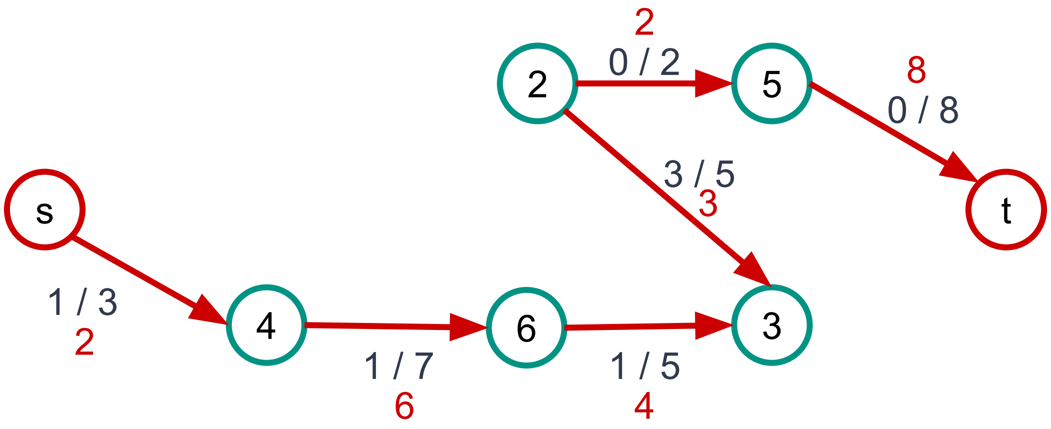

s
1
2
5
t
4
6
3
3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4


2
5
s
t
4
6
3
0 / 2
0 / 8
3 / 5
1 / 3
1 / 7
1 / 5

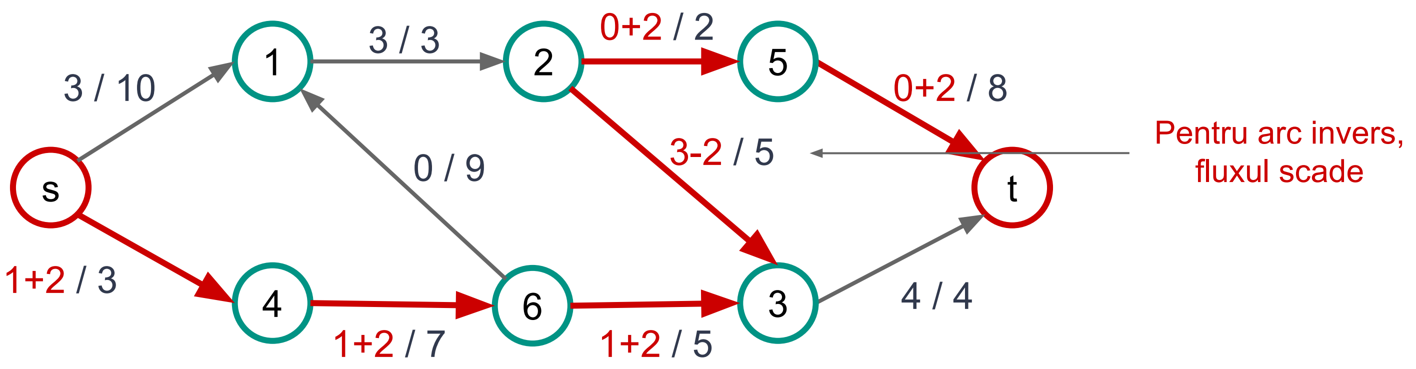

3 / 10
3 / 3
0 / 2
0 / 8
0 / 9
3 / 5
1 / 3
1 / 7
1 / 5
4 / 4
s
1
2
5
t
4
6
3

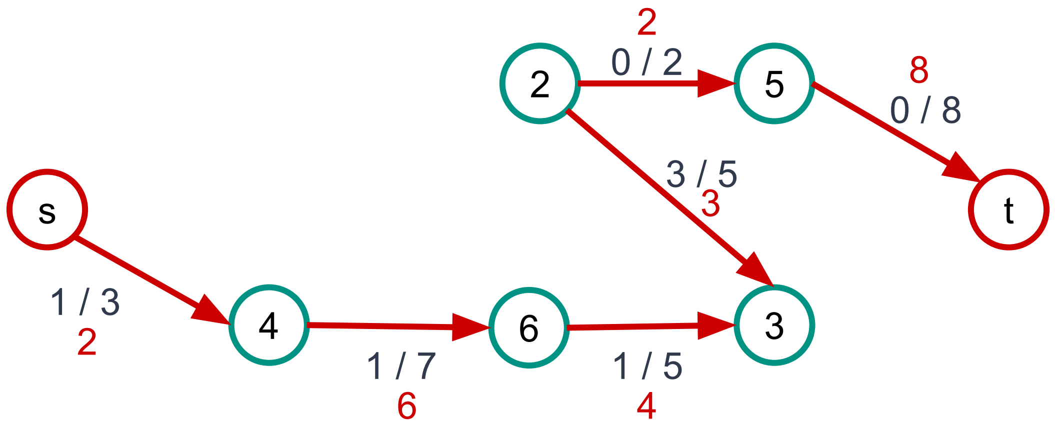

2 - 0 = 2
0 / 2
8 - 0 = 8
0 / 8
Pentru arc invers,
capacitatea
reziduală = fluxul
3 / 5
3
1 / 3
3 - 1 = 2
1 / 7
7 - 1 = 6
1 / 5
5 - 1 = 4
s
2
5
t
4
6
3



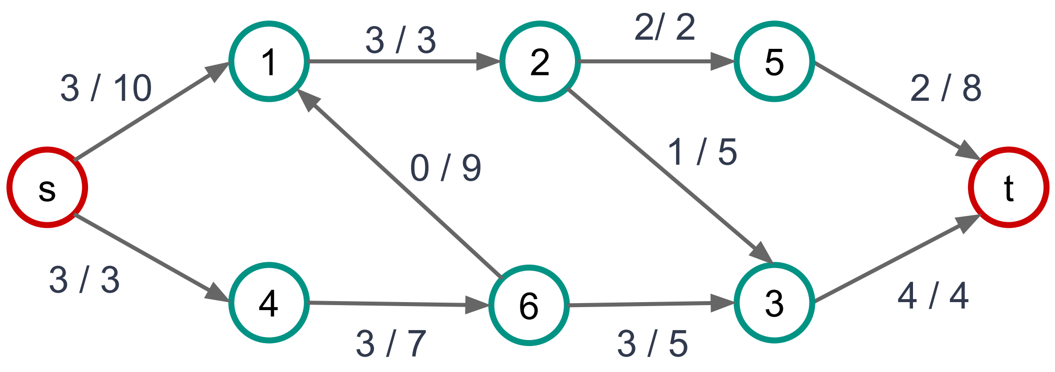


$$i(P) = \min \{ 2, 6, 4, 3, 2, 8 \} = 2$$



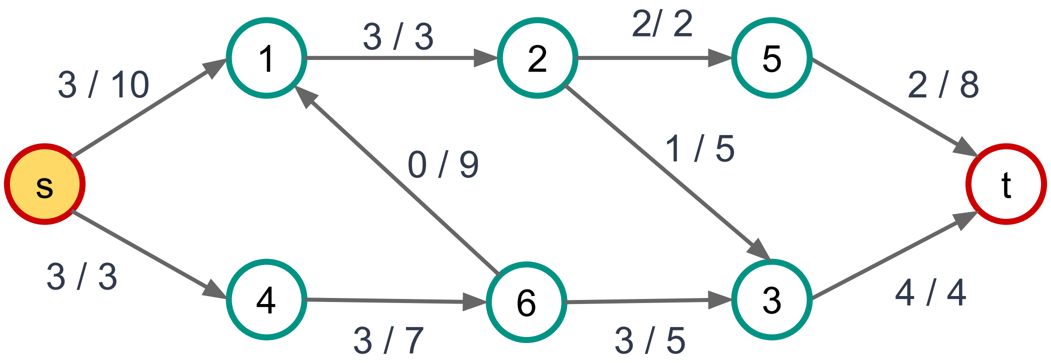


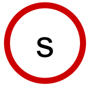
i(P) = min { 2, 6, 4, 3, 2, 8 } = 2

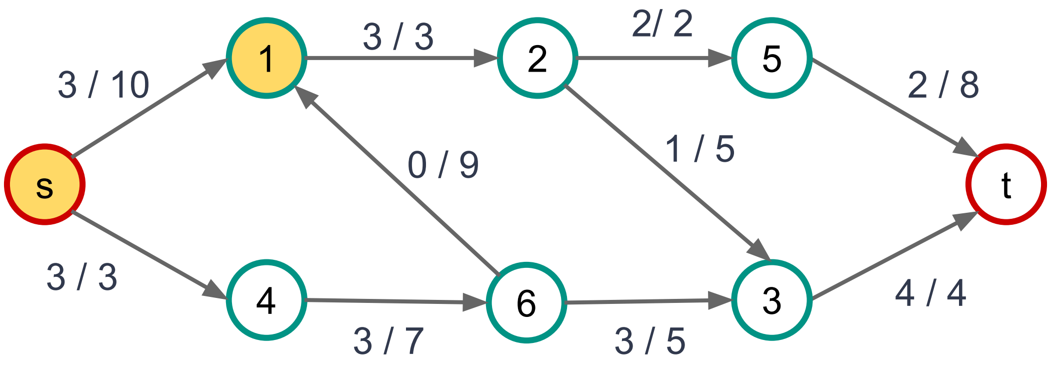


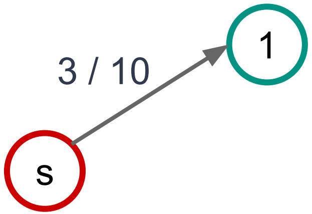
s
1
2
5
t
4
6
3
3 / 10
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 3
3 / 7
3 / 5
4 / 4

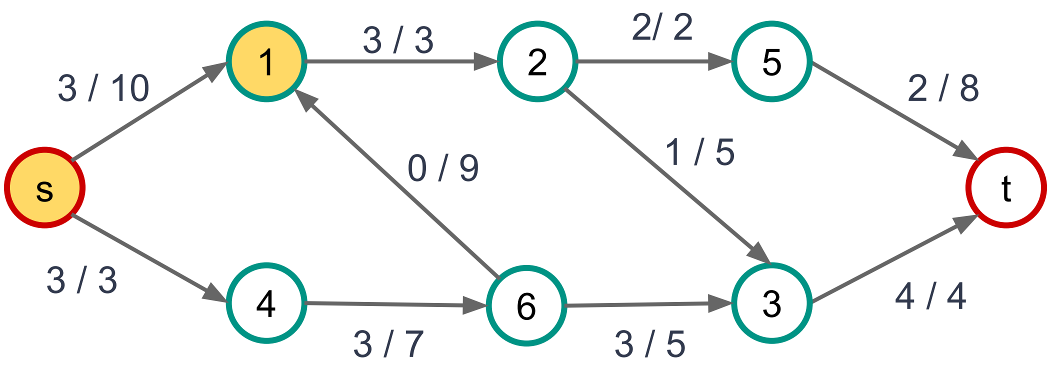
construiește_s-t_lanț_nesat_BF()

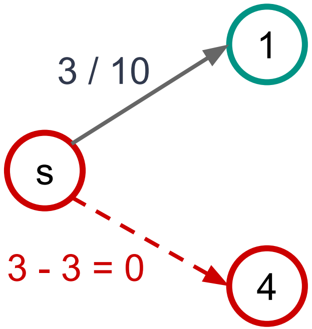

s
1
2
5
t
4
6
3
3 / 10
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 3
3 / 7
3 / 5
4 / 4

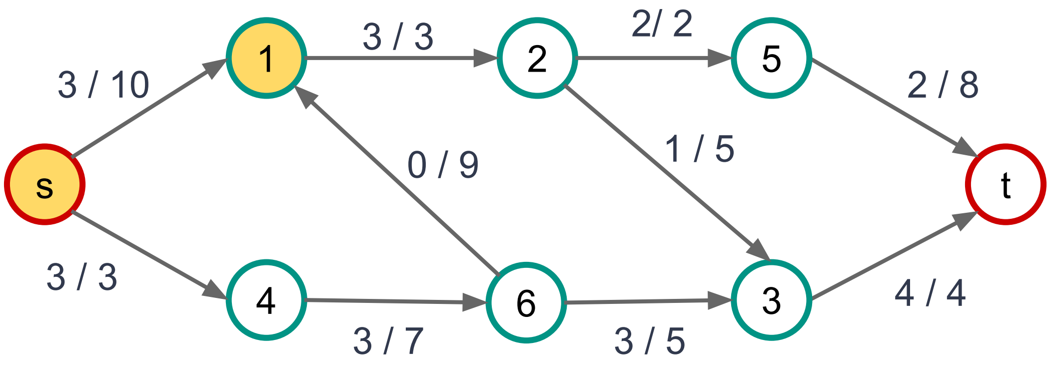

s

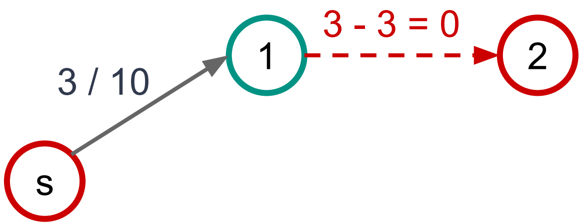

s
1
2
5
t
4
6
3
3 / 10
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 3
3 / 7
3 / 5
4 / 4

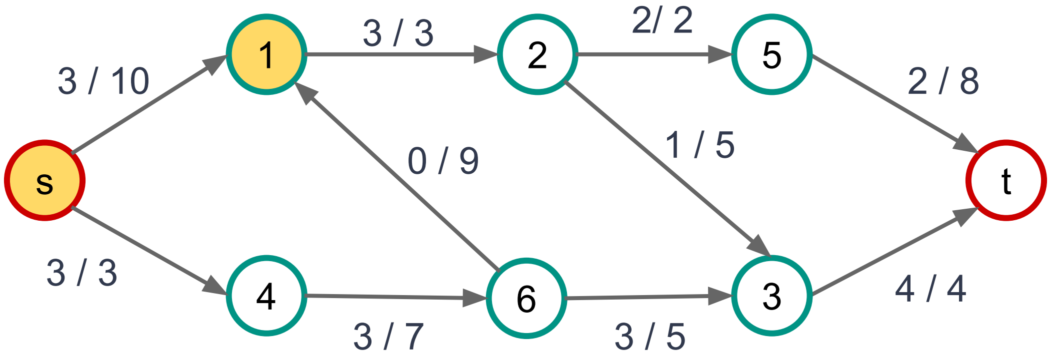

s
1
3 / 10

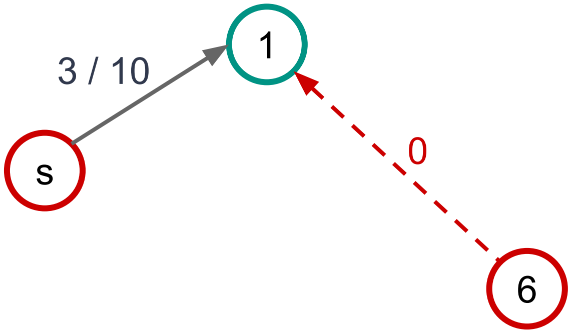

s
1
2
5
t
4
6
3
3 / 10
3 / 3
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 7
3 / 5
4 / 4


s
1
4
3 / 10
3 - 3 = 0


s
1
2
5
t
4
6
3
3 / 10
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 3
4 / 4
3 / 7
3 / 5


s
1
2
3 / 10
3 - 3 = 0


s
1
2
5
t
4
6
3
3 / 10
3 / 3
2/ 2
2 / 8
0 / 9
1 / 5
3 / 3
3 / 7
3 / 5
4 / 4


s
1
6
3 / 10
0

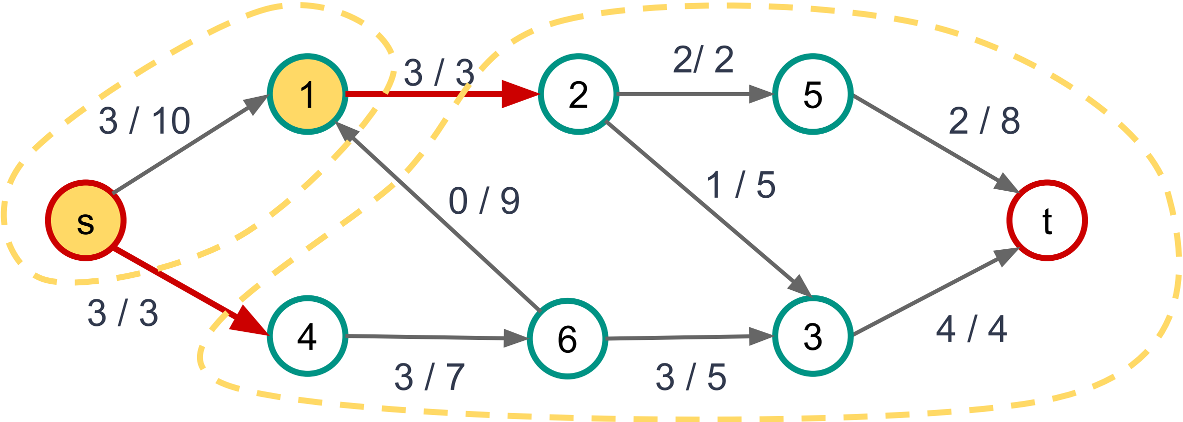
t nu este accesibil din s ⇒ STOP

- f este flux maxim

t nu este accesibil din s ⇒ STOP

- f este flux maxim
- tăietura determinată de vârfurile accesibile din s, la ultimul pas, prin lanțuri f-nesaturate, este tăietură minimă (= din vârfurile vizitate la ultimul pas)

Vom demonstra!



Tăietură minimă

Sugestii de implementare

Algoritmul Edmonds-Karp

Implementare

Memorăm lanțurile (arborele BF), folosind vectorul **tata**

<u>Convenție</u> - pentru arcele inverse (i, j) ținem minte tatăl cu semnul minus

tata[j] = -i

construiește_s-t_lanț_nesat_BF()

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută           // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută                  // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută             // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
                dacă j = t atunci
                    STOP și returnează true (1)
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută                   // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
                dacă j = t atunci
                    STOP și returnează true (1)
        pentru ji ∈ E execută                   // arc invers
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută                    // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
                dacă j = t atunci
                    STOP și returnează true (1)
        pentru ji ∈ E execută                    // arc invers
            dacă viz[j] = 0 și f(ji) > 0 atunci
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută                // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
                dacă j = t atunci
                    STOP și returnează true (1)
        pentru ji ∈ E execută                // arc invers
            dacă viz[j] = 0 și f(ji) > 0 atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← -i
```

```
construiește_s-t_lanț_nesat_BF()
    pentru v ∈ V execută
        tata[v] ← 0;  viz[v] ← 0
    coada C ← ∅
    adaugă(s, C)
    viz[s] ← 1
    cât timp C ≠ ∅ execută
        i ← extrage(C)
        pentru ij ∈ E execută                   // arc direct
            dacă viz[j] = 0 și c(ij) > f(ij) atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← i
                dacă j = t atunci
                    STOP și returnează true (1)
        pentru ji ∈ E execută                   // arc invers
            dacă viz[j] = 0 și f(ji) > 0 atunci
                adaugă(j, C)
                viz[j] ← 1
                tata[j] ← -i
                dacă j = t atunci
                    STOP și returnează true (1)
returnează false (0)
```

Algoritmul Edmonds–Karp

Complexitate

- Algoritm generic Ford-Fulkerson - **$O(mL)$ / $O(nmC)$**
- Implementare Edmonds-Karp - **$O(nm^2)$**

Implementare

Schema:

inițializează_flux_nul()

cât timp (construiește_s-t_lanț_nesat_BF() == **true**) **execută**

    revizuiește_flux_lanț()

afișează_flux()

Amintim: a determina un s-t lanț nesaturat folosind BF în G ⇔ a determina un s-t drum folosind BF în graful rezidual G_f

Varianta 2 de implementare

revizuirea fluxului folosind s-t drumuri din G_f (în graful rezidual)

Implementare folosind graf rezidual

<u>Schema devine:</u>

inițializează_flux_nul()

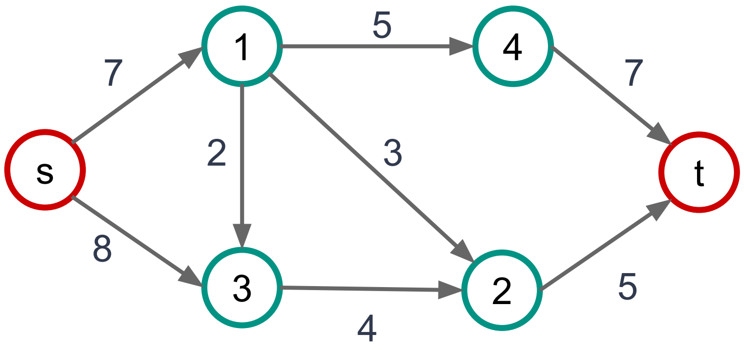
cât timp (construiește_s-t_drum_în_G_fBF() == **true**) **execută**

    revizuiește_flux_lanț()

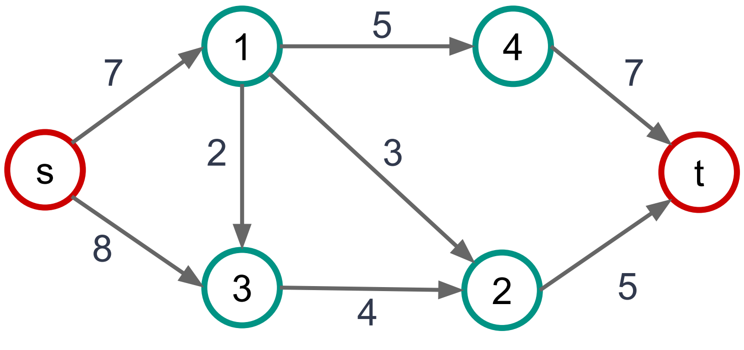
    actualizează G_f

afișează_flux()

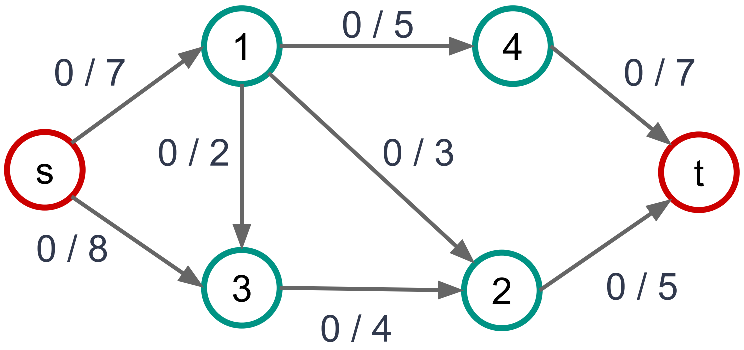
Detaliem această schemă.

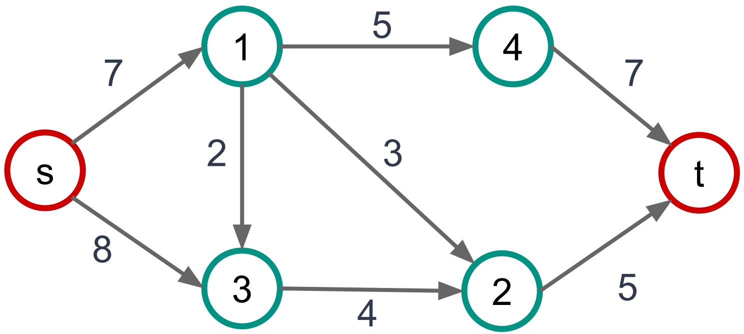


Graful rezidual G_f

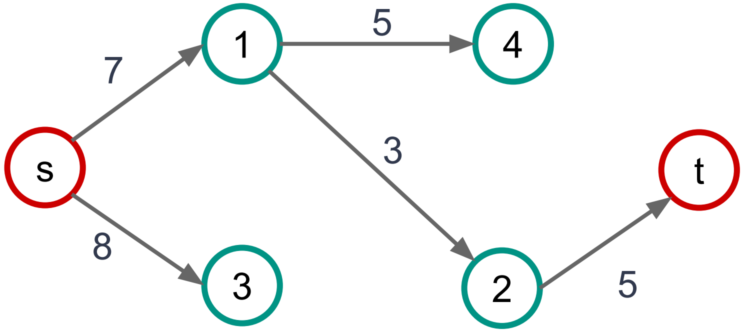


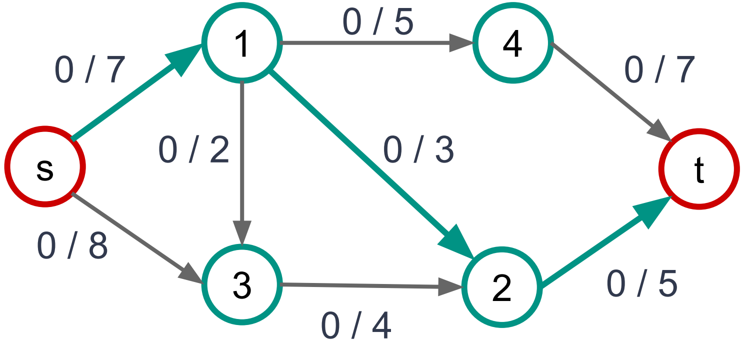
Graful rezidual G_f



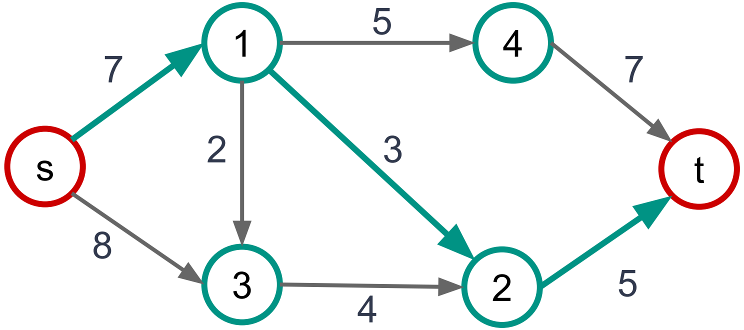


BF(s) - în graful rezidual

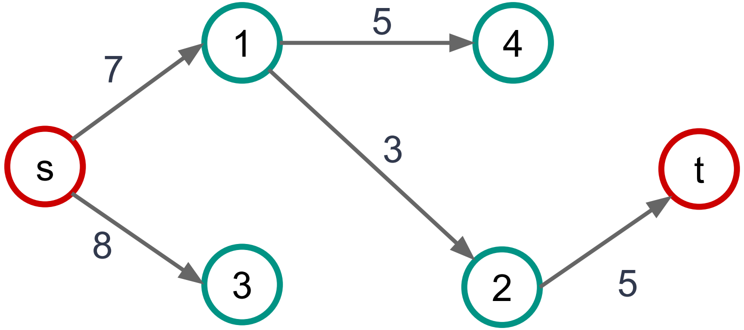




Graful rezidual G_f



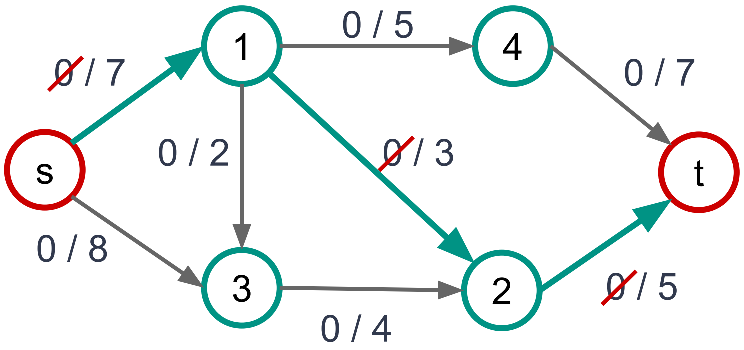
BF(s) - în graful rezidual

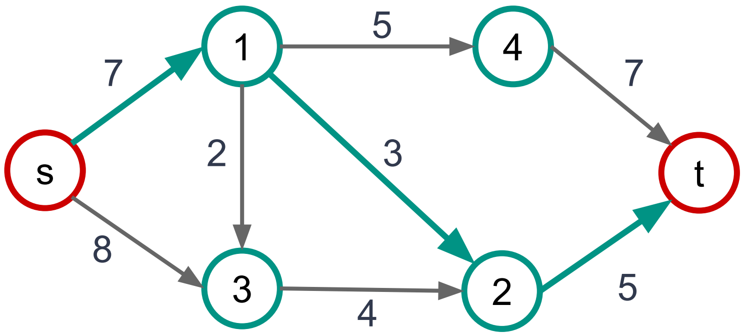


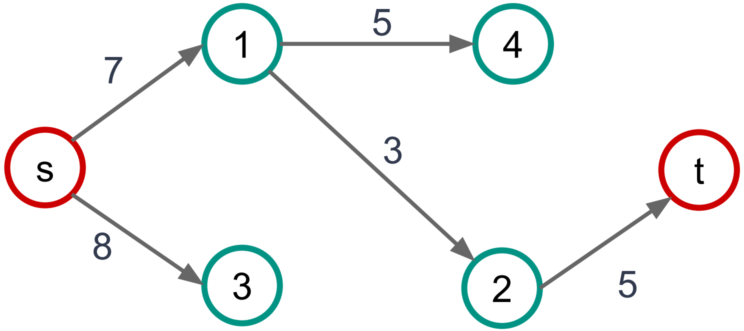
Drumul de creștere [s, 1, 2, t] - capacitate reziduală 3

Revizuim fluxul

Graful rezidual G_f



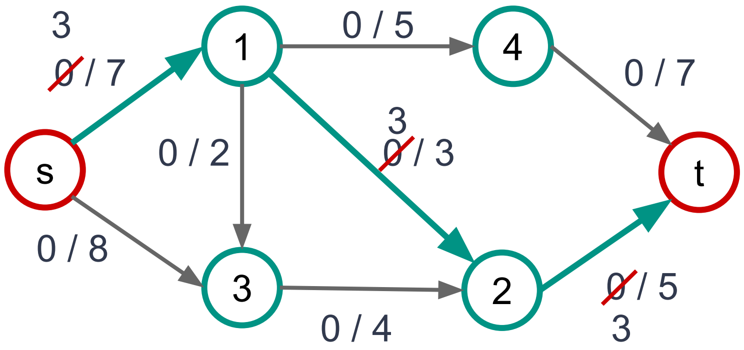




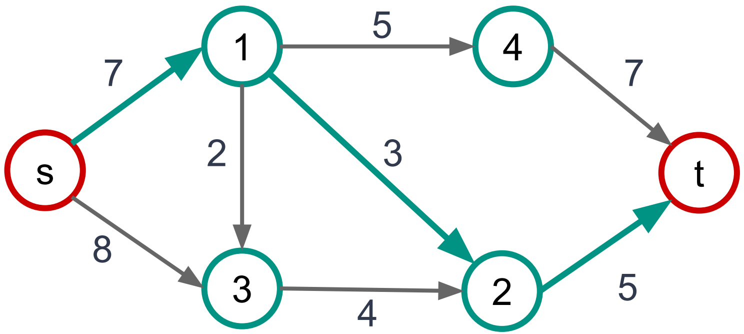
BF(s) - în graful rezidual

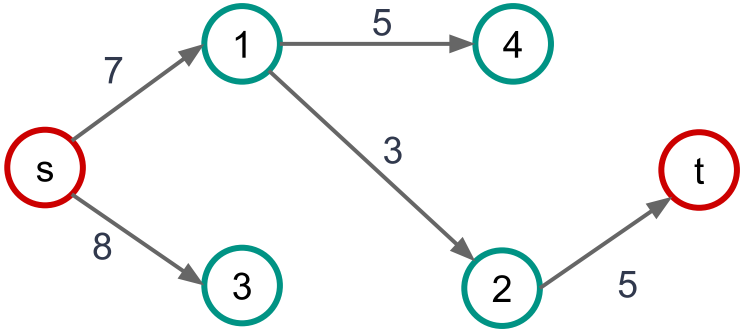
Drumul de creștere [s, 1, 2, t] - capacitate reziduală 3

Revizuim fluxul



Graful rezidual G_f

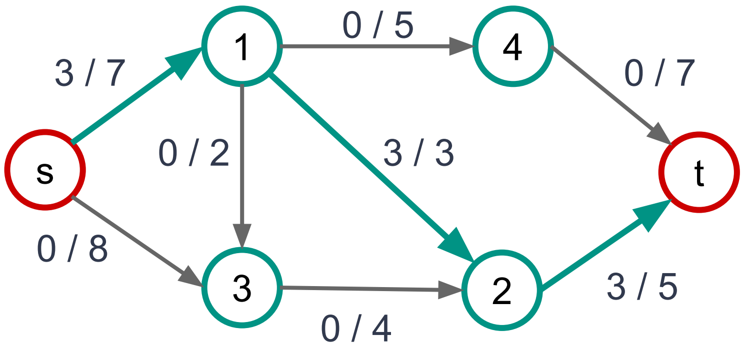




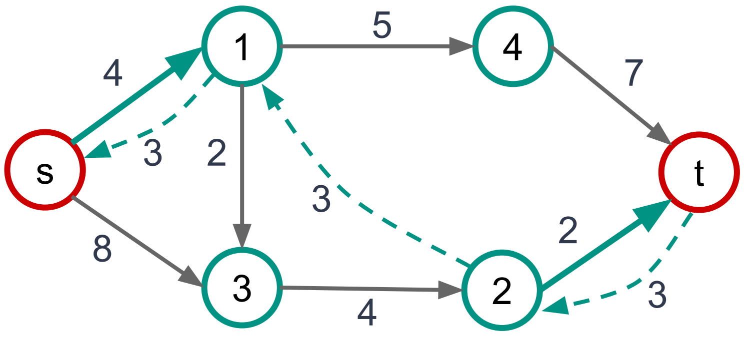
BF(s) - în graful rezidual

Drumul de creștere [s, 1, 2, t] - capacitate reziduală 3

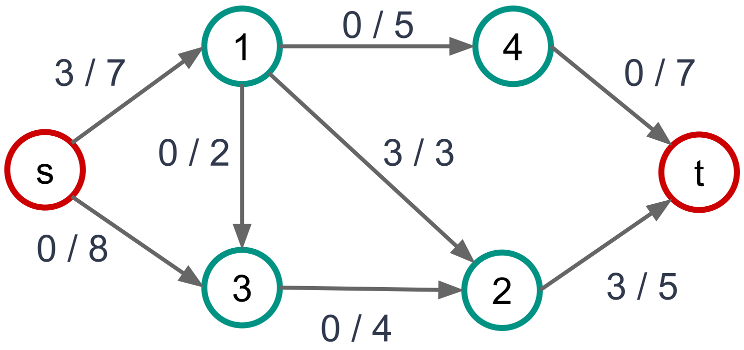
Revizuim fluxul



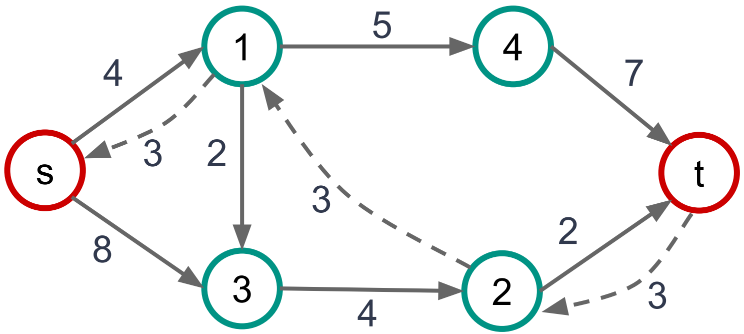
Graful rezidual G_f



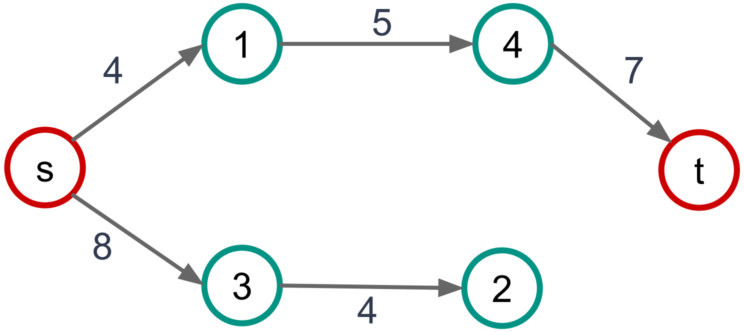
Actualizăm rețeaua reziduală



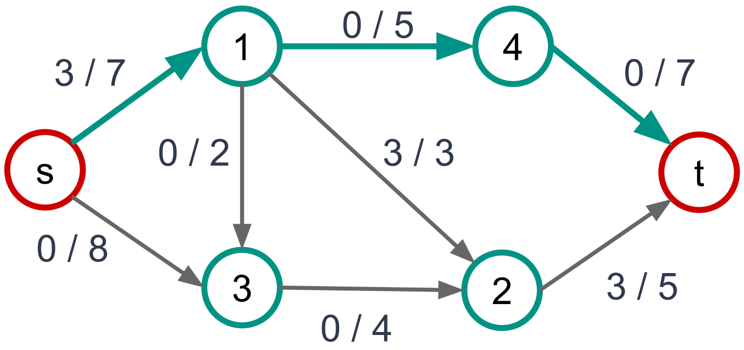
Graful rezidual G_f

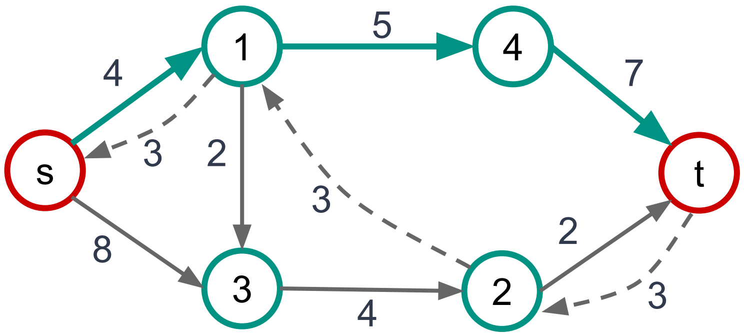


BF(s) - în graful rezidual

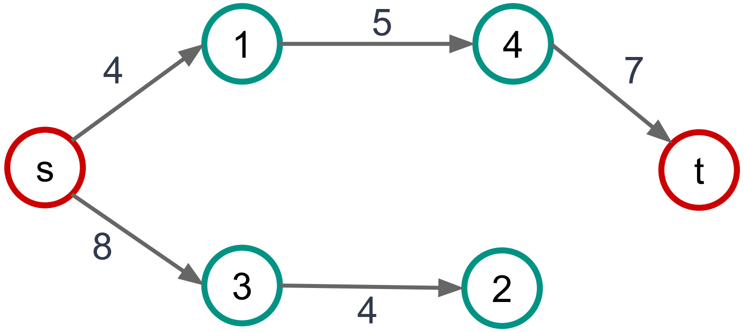


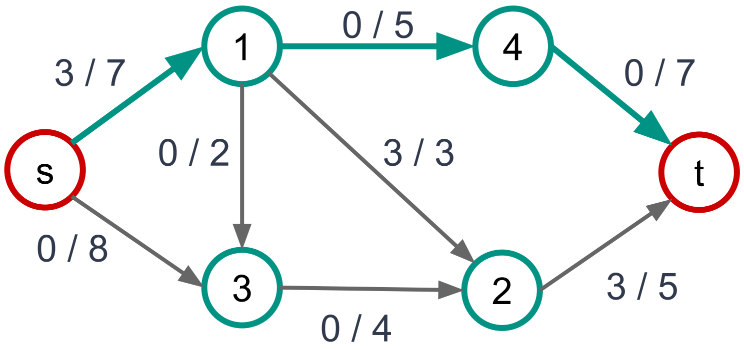
Graful rezidual G_f



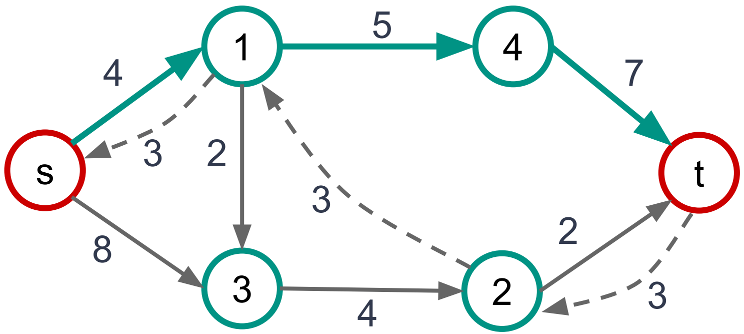


BF(s) - în graful rezidual

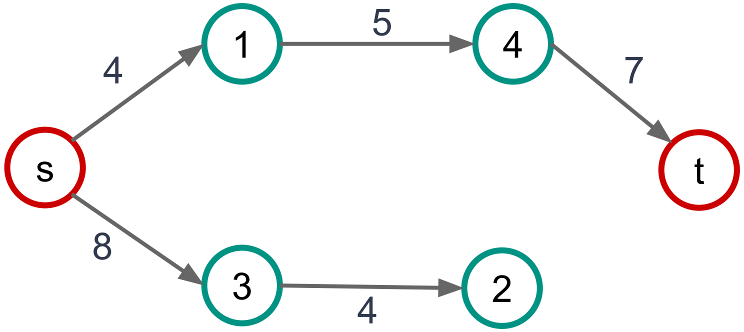




Graful rezidual G_f

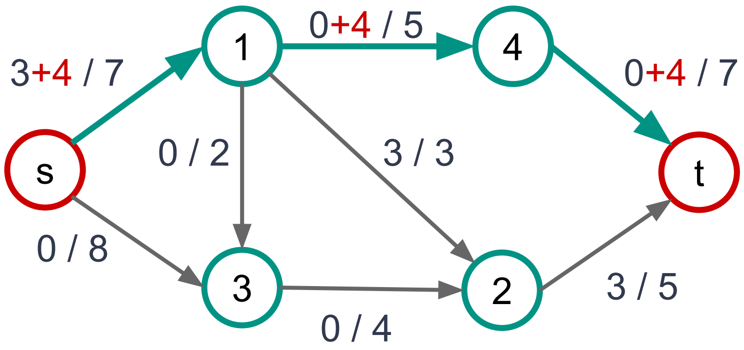


BF(s) - în graful rezidual

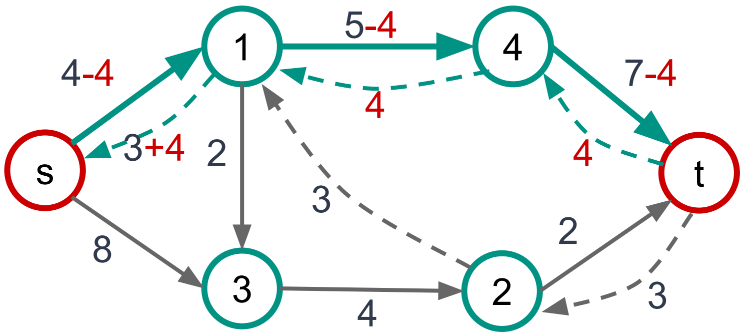


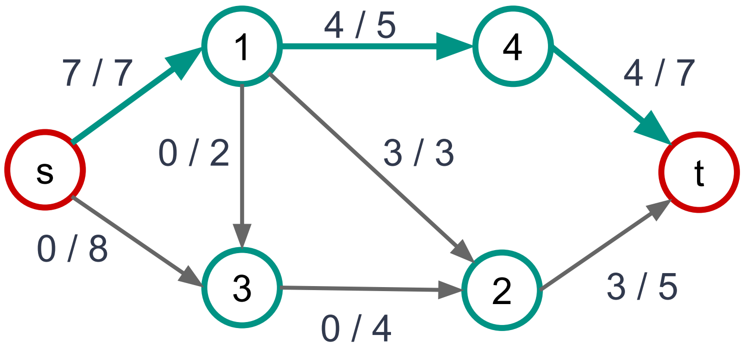
Drumul de creștere [s, 1, 4, t] - capacitate reziduală 4

Revizuim fluxul

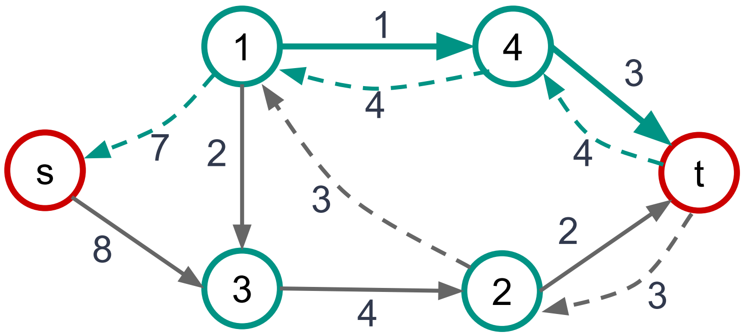


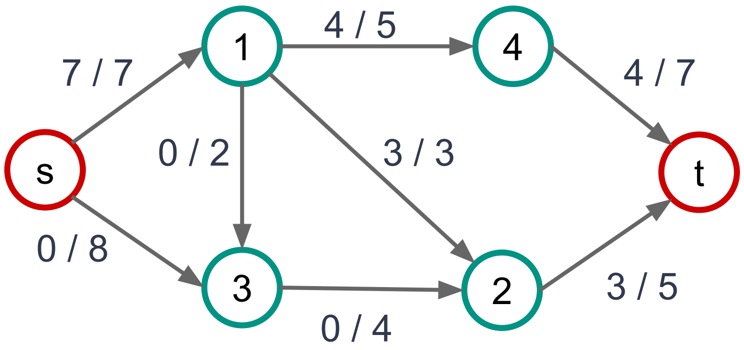
Graful rezidual G_f



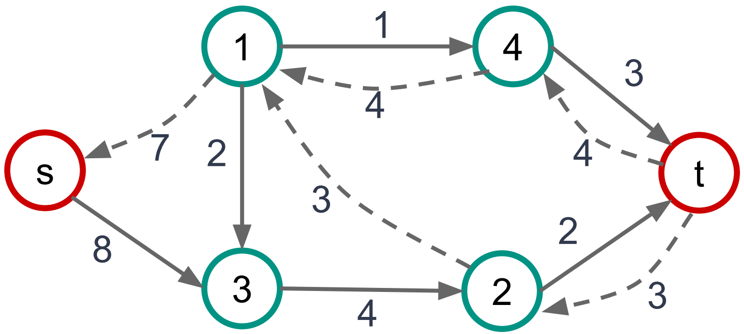


Graful rezidual G_f

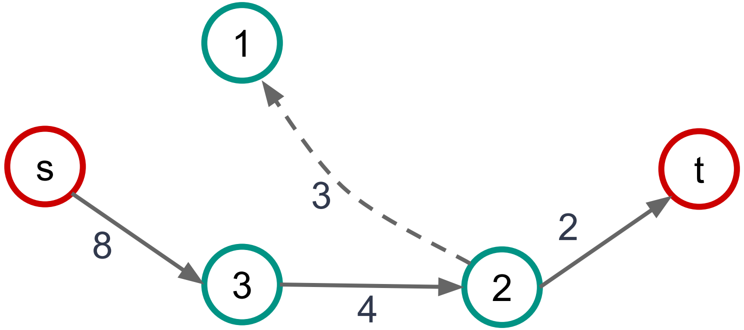


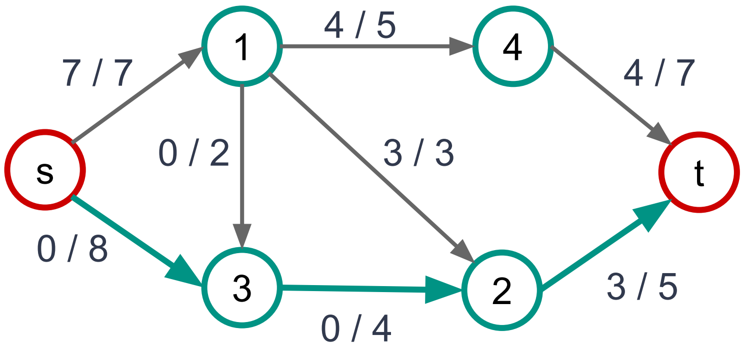


Graful rezidual G_f

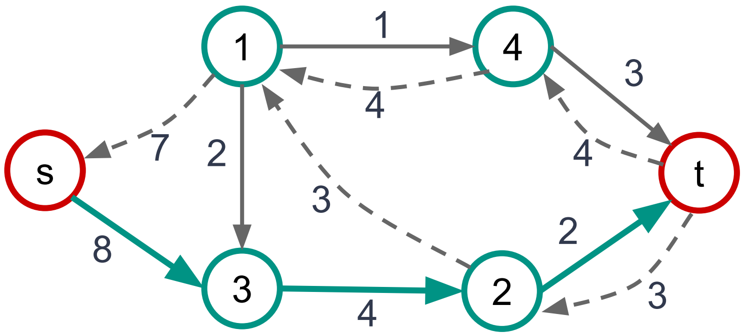


BF(s) - în graful rezidual

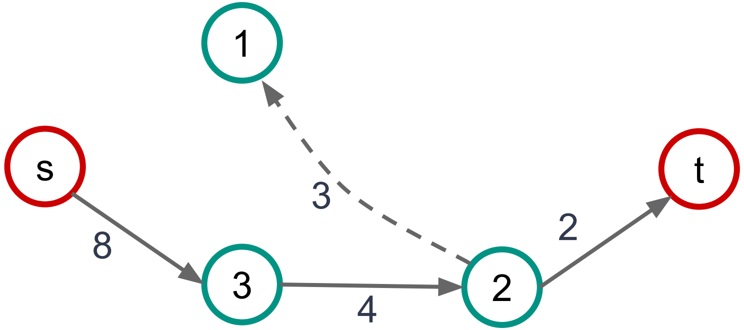




Graful rezidual G_f

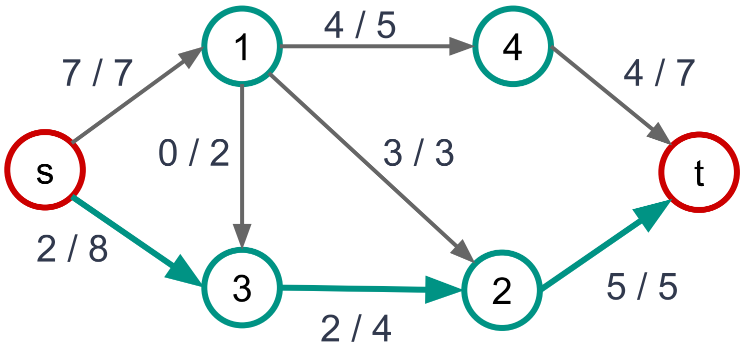


BF(s) - în graful rezidual

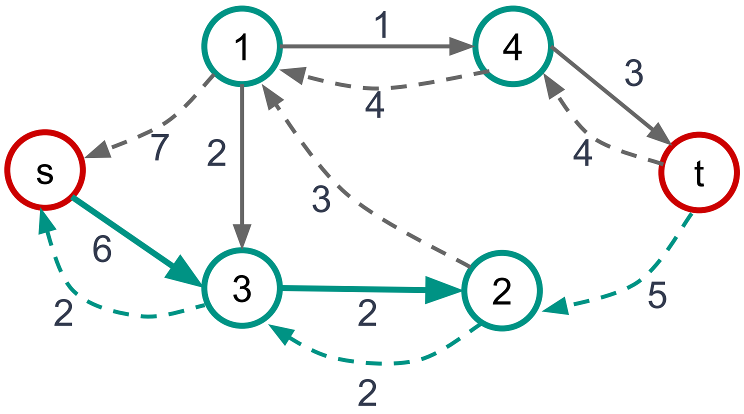


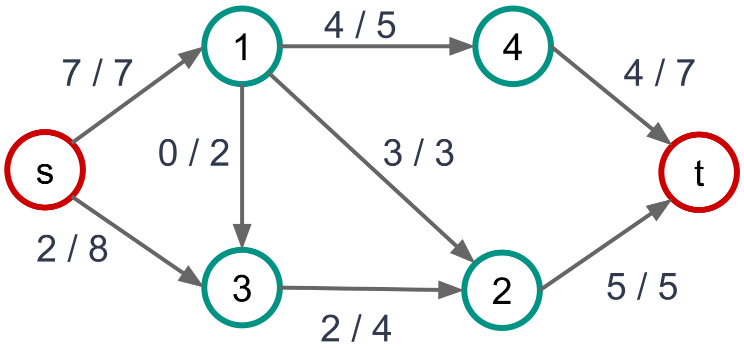
Drumul de creștere [s, 3, 2, t] - capacitate reziduală 2

Revizuim fluxul

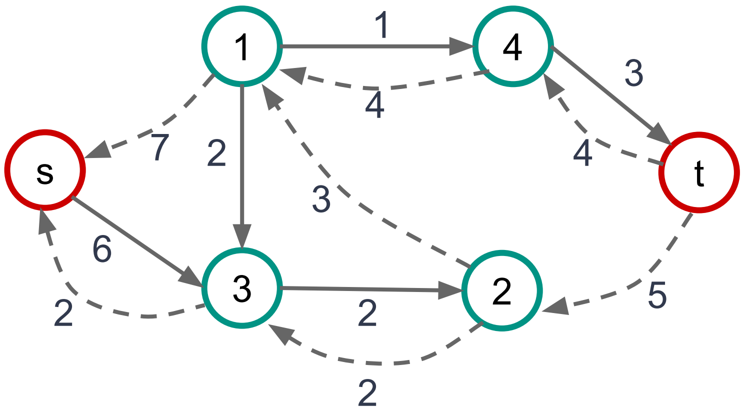


Graful rezidual G_f

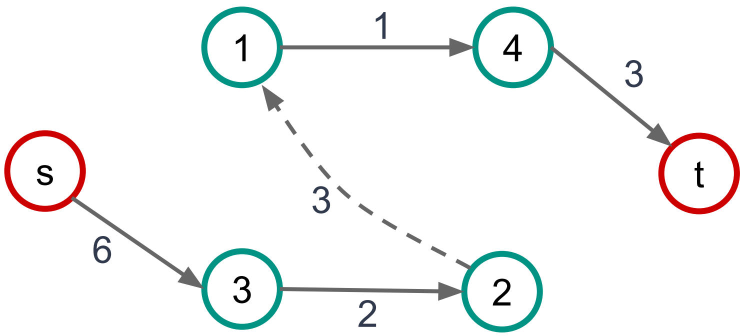


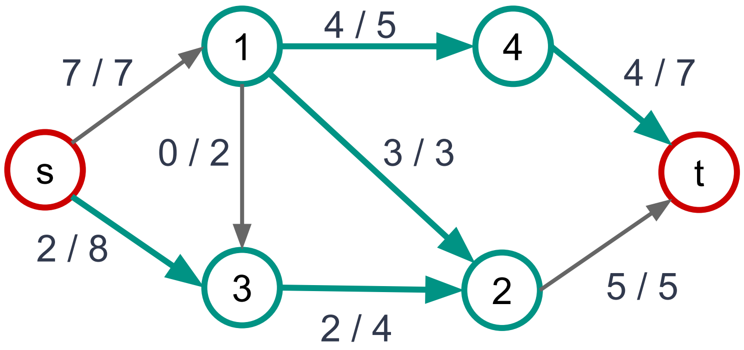


Graful rezidual G_f

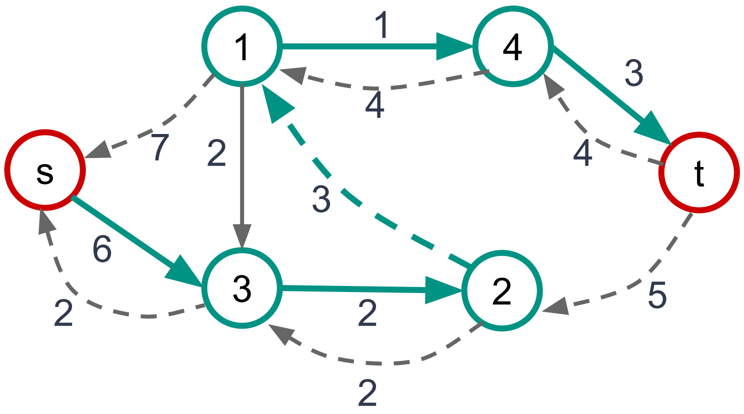


BF(s) - în graful rezidual

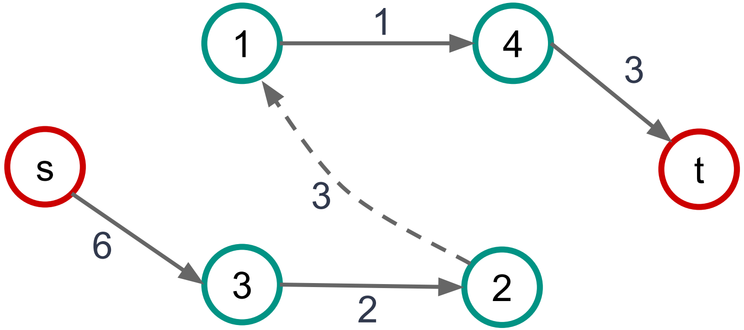




Graful rezidual G_f

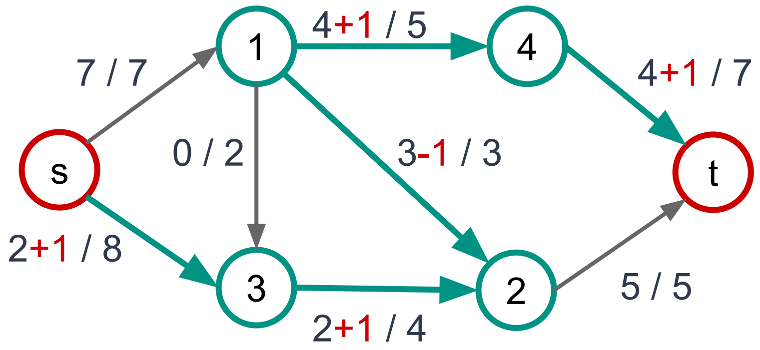


BF(s) - în graful rezidual

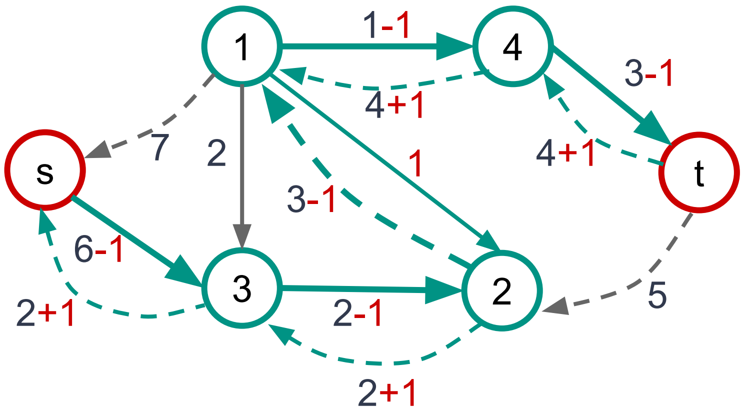


Drumul de creștere [s, 3, 2, 1, 4, t] - capacitate reziduală 1

Revizuim fluxul

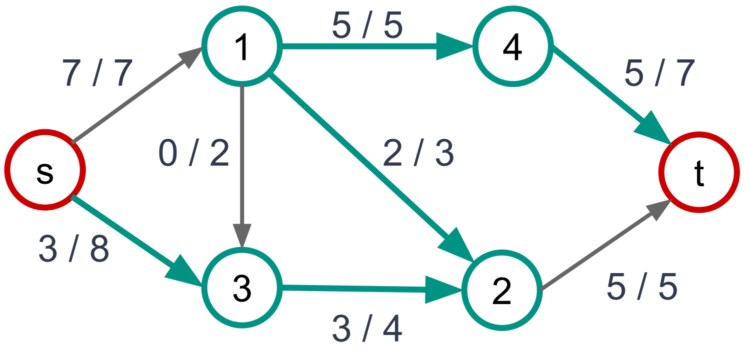


Graful rezidual G_f

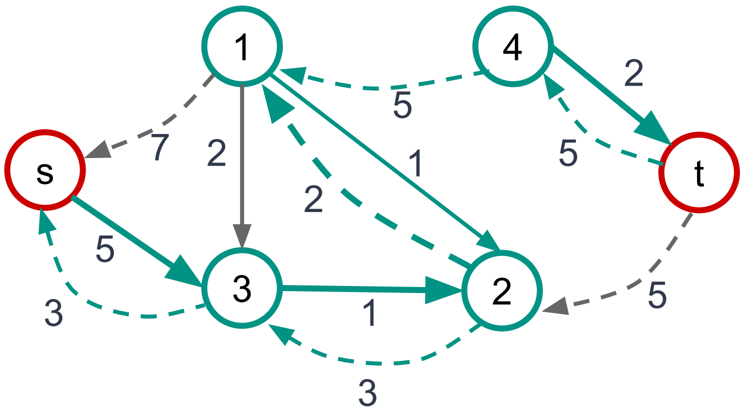


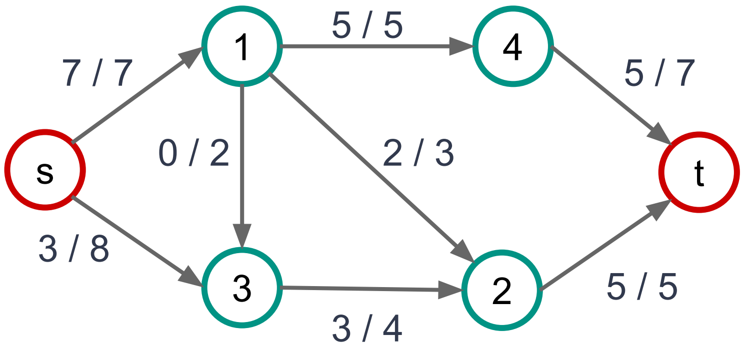
actualizare G_f:

```
pentru e ∈ E(P) ⊆ E(G_f) execută
     c_f(e) ← c_f(e) - cfP
     dacă c_f(e) = 0 atunci
          elimină e din G_f      // se ignoră în parcurgere
     c_f(e^-1) ← c_f(e^-1) + cfP
     dacă c_f(e^-1) > 0 și e^-1 ∉ G_f atunci
          adaugă e^-1 la G_f
```

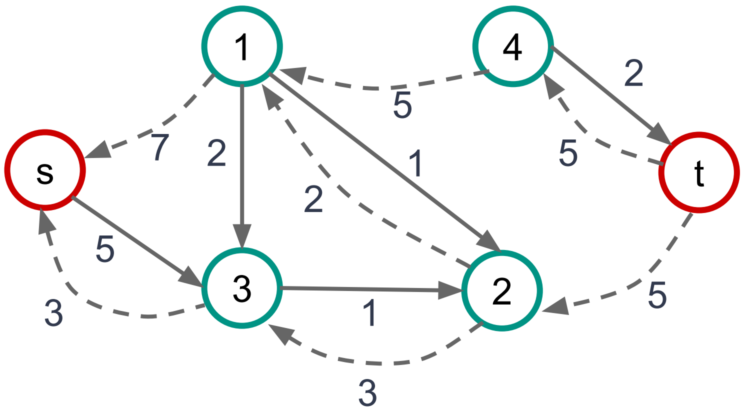


Graful rezidual G_f

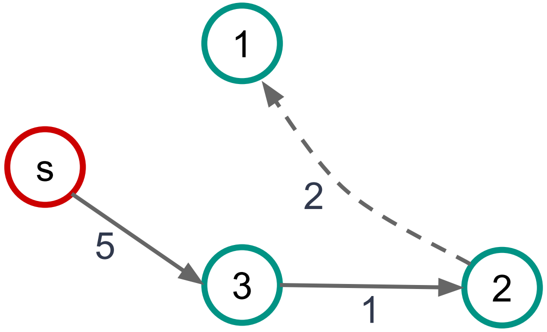


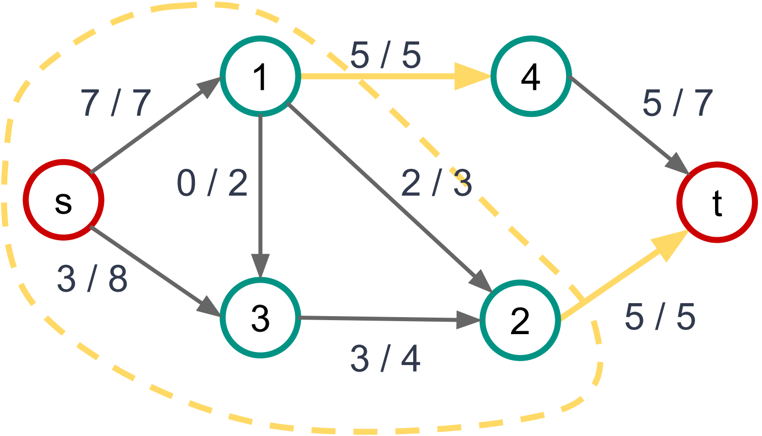


Graful rezidual G_f

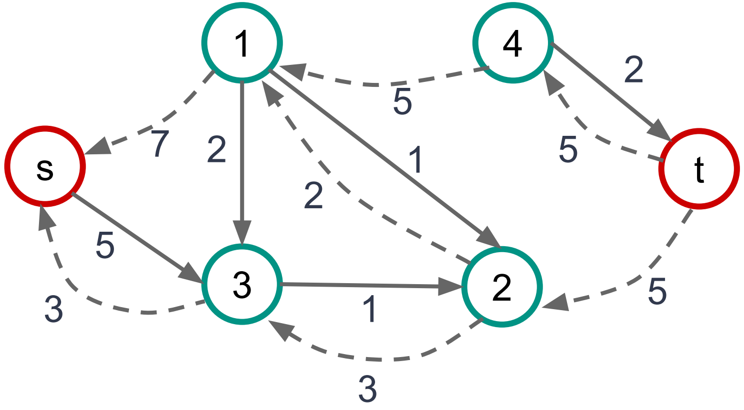


BF(s) - în graful rezidual

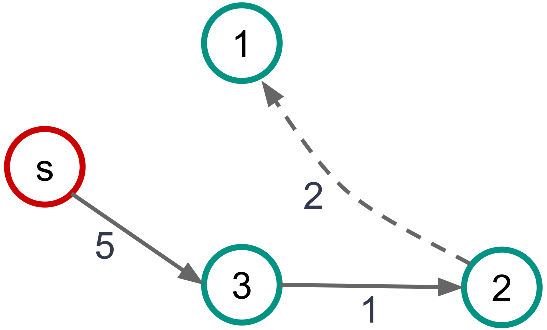




Graful rezidual G_f



BF(s) - în graful rezidual



Nu mai există drum de creștere ⇒ s-t flux maxim (valoare 10) + s-t tăietură minimă (de capacitate tot 10, determinată de vârfurile accesibile din s în G_f : S = { s, 1, 3, 2 })