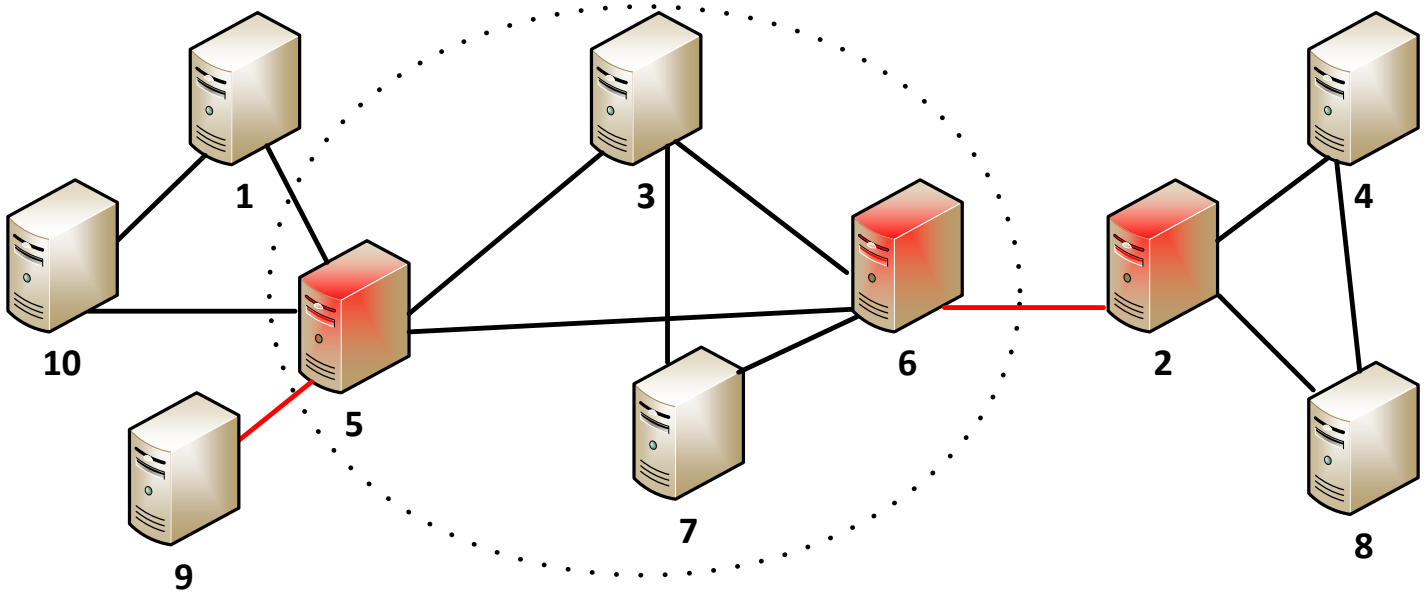
Probleme de conectivitate în grafuri neorientate

2-Conectivitate

- G – graf neorientat
- Puncte și legături vulnerabile



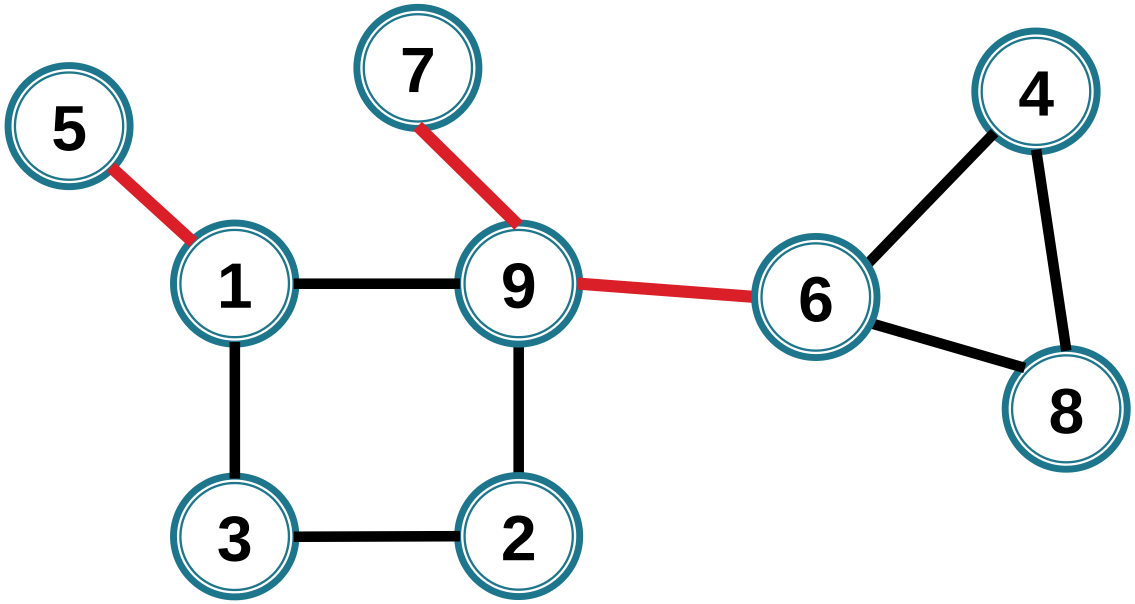
Muchii critice

Muchii critice

- G – graf neorientat
- e ∈ E(G) **critică** (**punte, muchie de articulaţie**/ *bridge*) = prin eliminarea ei crește numărul de componente conexe ale grafului

  nr componente (G – e) >nr. componente (G)

- Un graf conex fără punți se numește 2-**muchie conex.**



Muchii critice

- O muchie este critică $\Leftrightarrow$ nu este conținută într-un ciclu

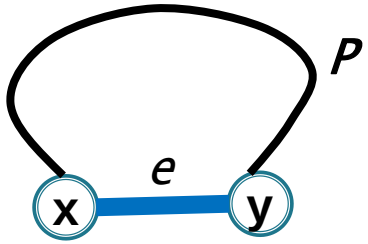
Demonstrație

Muchii critice

- O muchie este critică ⇔ nu este conținută într-un ciclu

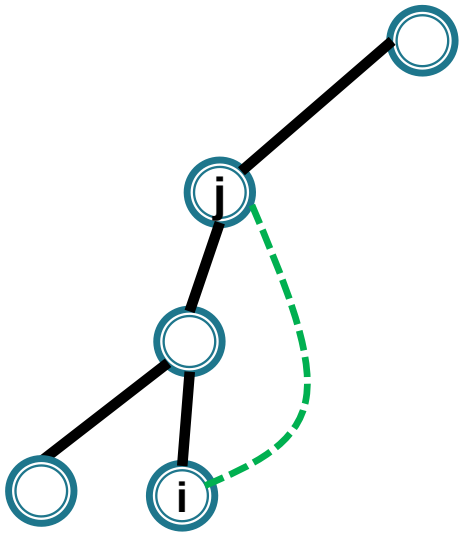
Demonstrație

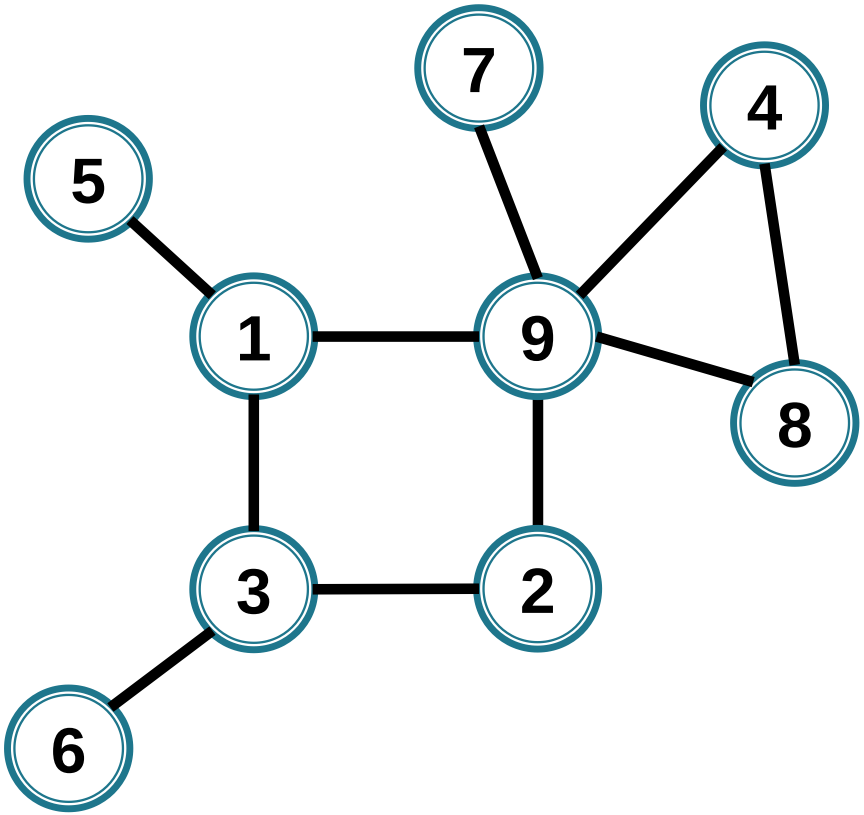
O muchie nu este critică ⇔ este conținută într-un ciclu

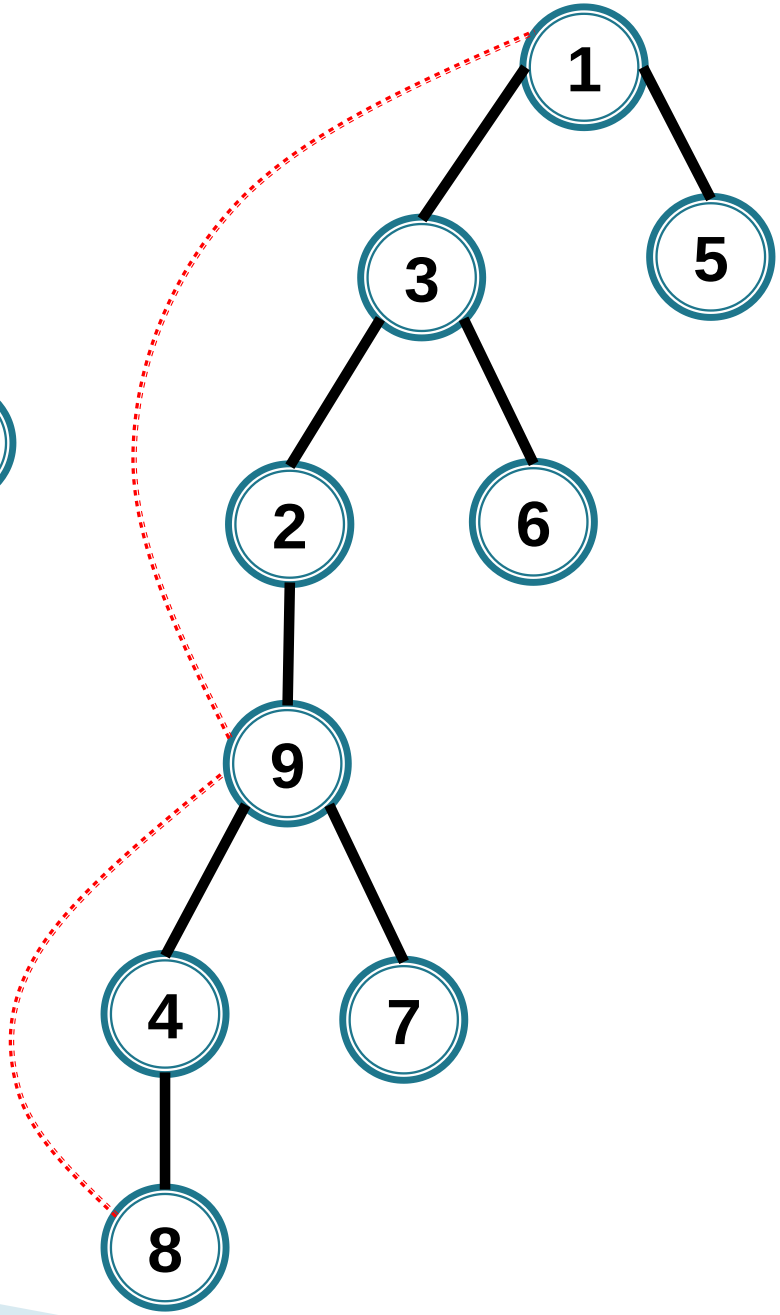


Muchii critice

- Găsirea unui ciclu – parcurgere DF
  - **muchii de avansare** – ale arborelui DF (memorat cu vector tata), prin care se descoperă vârfuri noi
  - **muchii de întoarcere** – închid ciclu, nu pot fi critice

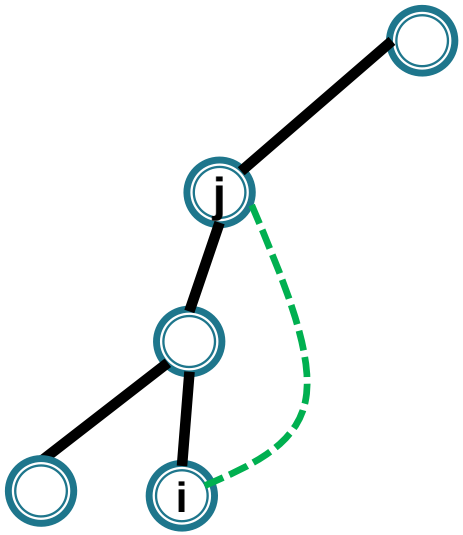



7
4
5
1
9
8
3
2
6


1
3
5
2
6
9
4
7
8

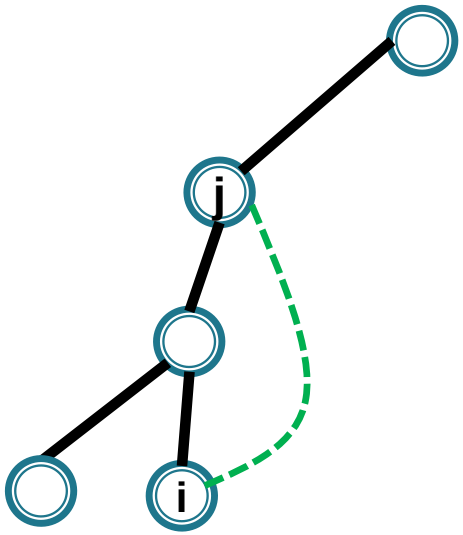
Muchii critice

- Găsirea unui ciclu – parcurgere DF – **O(n)**
  - **muchii de avansare** – ale arborelui DF (memorat cu vector tata), prin care se descoperă vârfuri noi
  - **muchii de întoarcere** – închid ciclu



Muchii critice

- Găsirea unui ciclu – parcurgere DF – **O(n)**
  - **muchii de avansare** – ale arborelui DF (memorat cu vector tata), prin care se descoperă vârfuri noi
  - **muchii de întoarcere** – închid ciclu, nu pot fi critice



Doar muchiile de avansare pot fi critice

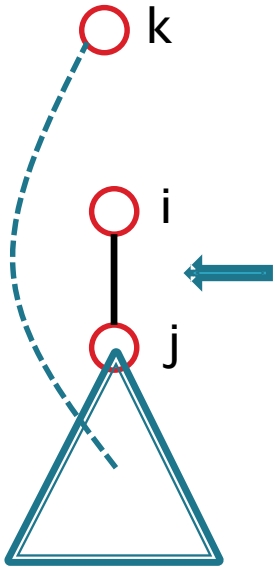
Muchii critice

Cum testăm dacă o muchie de avansare (i,j) este critică?

Muchii critice

Cum testăm dacă o muchie de avansare (i,j) este critică?

- **nu este conținută într-un ciclu închis de o muchie de întoarcere**



Muchii critice

O muchie de avansare (i,j) este critică

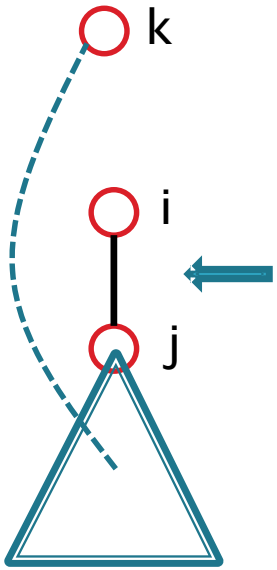
⇔

nu este conținută într-un ciclu închis de o muchie de întoarcere

⇔

nu există nicio muchie de întoarcere cu

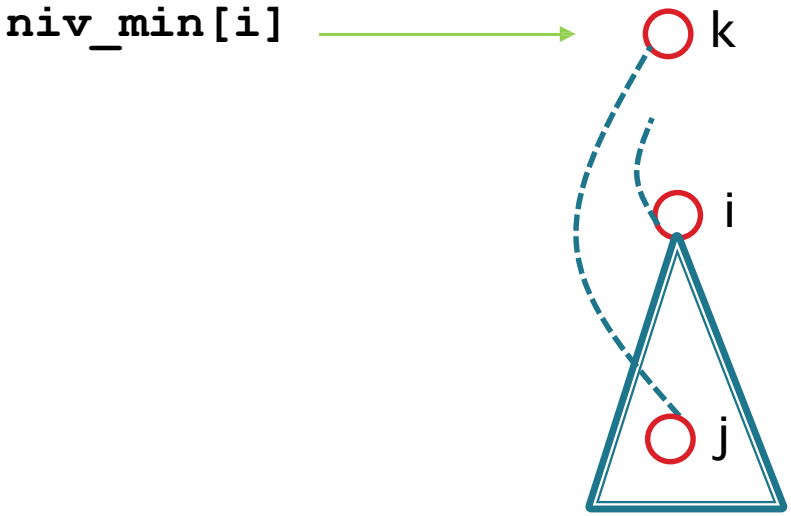
- o extremitate în **j sau într-un descendent al lui j** și
- cealaltă extremitate **în i sau într-un ascendent al lui i** (într-un vârf de **pe un nivel mai mic sau egal cu nivelul lui i**)



Muchii critice

Memorăm pentru fiecare vârf i:

`niv_min[i]` = *intuitiv: cât de sus putem ajunge din i mergând în sensul parcurgeii DF*

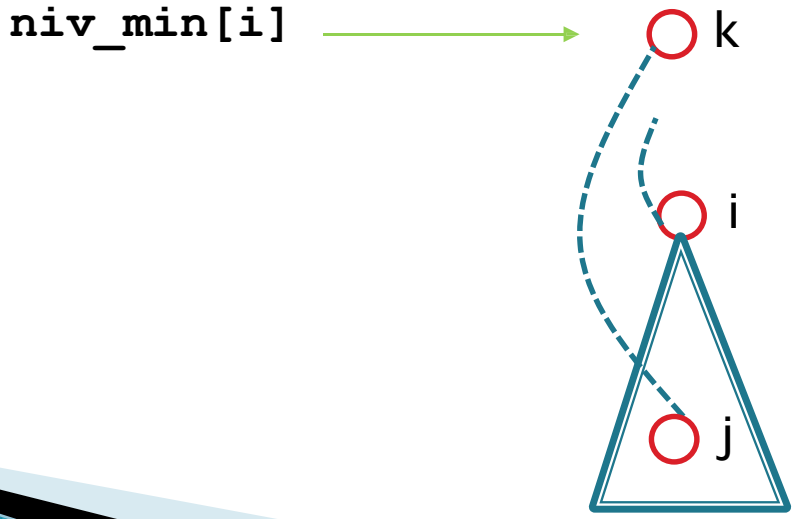


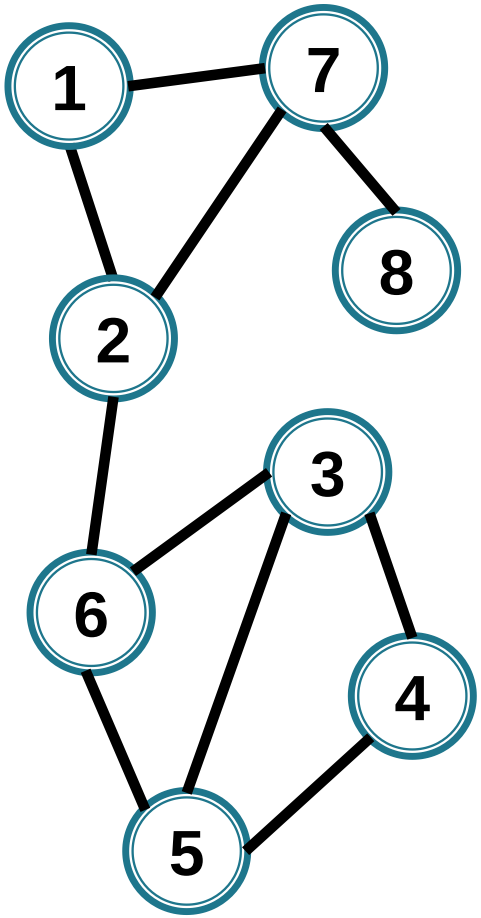
Muchii critice

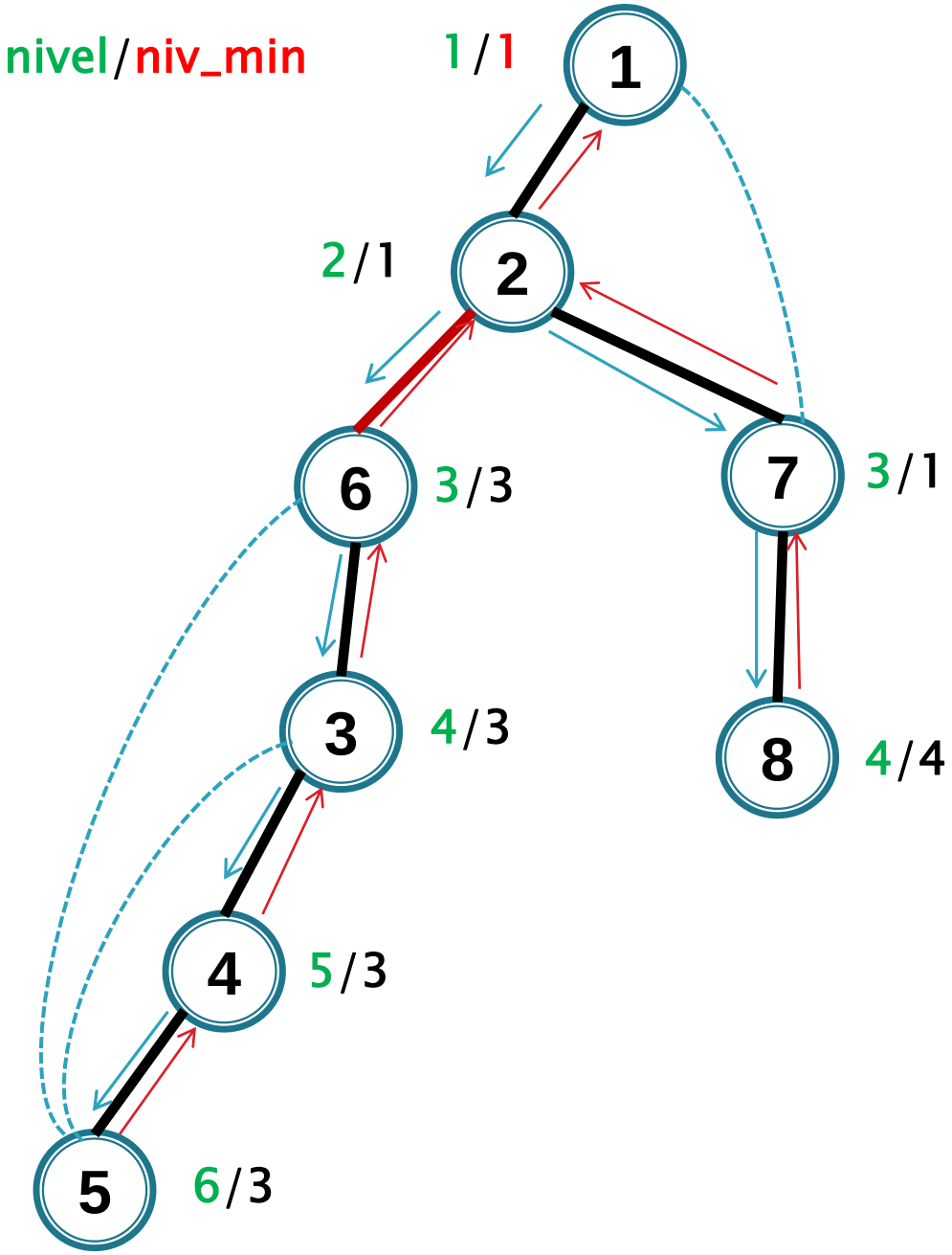
Memorăm pentru fiecare vârf i:

`niv_min[i]` = nivelul minim al unui vârf care este extremitate a unei muchii de întoarcere din i sau dintr-un descendent al lui i

= nivelul minim la care se închide un ciclu elementar care conține vârful i printr-o muchie de întoarcere

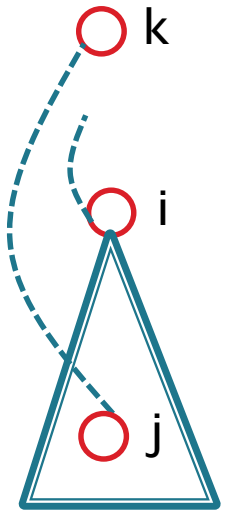



1
7
8
2
3
6
4
5


nivel/niv_min
1/1
1
2/1
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3

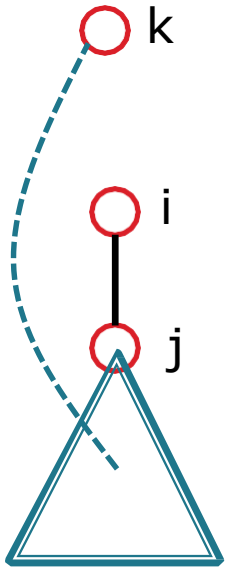
Muchii critice

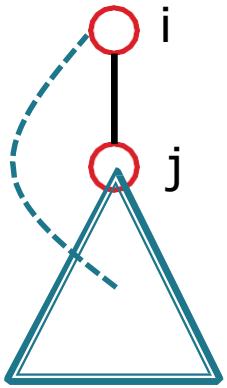
- **`nivel[i]`** = nivelul lui i în arborele DF
- **`niv_min[i]`** = min { **`nivel[i]`**, A, B}
  - A = min {**`nivel[k]`** | **`ik`** muchie de întoarcere}
  - B = min {**`nivel[k]`** | **`j`** **descendent** al lui **`i`**, **`jk`** muchie de întoarcere}

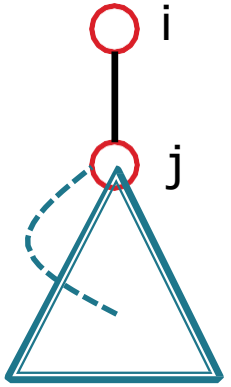


Muchii critice

O **muchie de avansare** ij este critică ⇔ ?

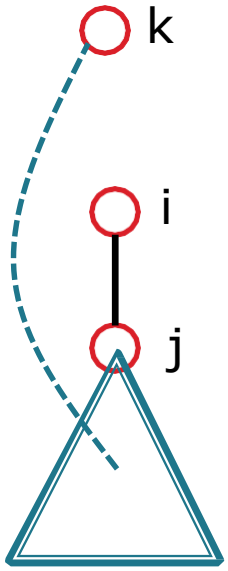




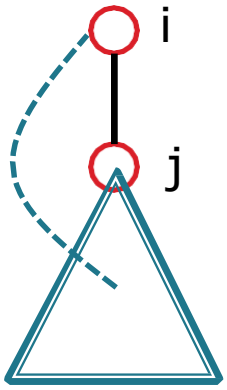


Muchii critice

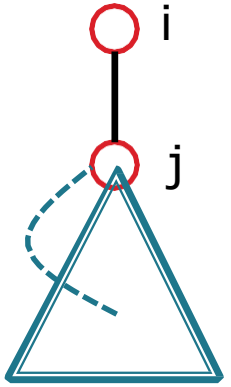
O **muchie de avansare** ij este critică ⇔ ?



NU este critică
`niv_min[j]<nivel[i]`



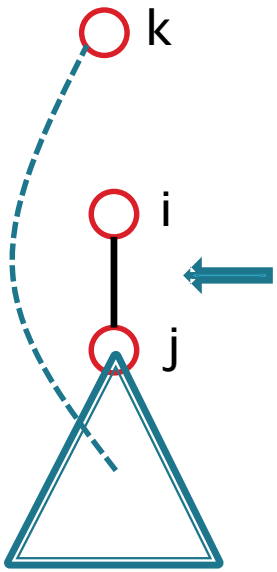
NU este critică
`niv_min[j]=nivel[i]`



ESTE critică
`niv_min[j]>nivel[i]`

Muchii critice

O **muchie de avansare** ij este critică ⇔ `niv_min[j] > nivel[i]`



Muchii critice

Cum calculăm eficient `niv_min[i]` ?

`niv_min[i]` = min { `nivel[i]`, A, B}

A = min {`nivel[k]` | `ik` muchie de întoarcere}

B = min {`nivel[k]` | `j` **descendent** al lui `i`,
`jk` muchie de întoarcere}

Muchii critice

Cum calculăm eficient `niv_min[i]` ?

`niv_min[i]` = min { `nivel[i]`, A, B}

A = min {`nivel[k]` | `ik` muchie de întoarcere}

B = min {`nivel[k]` | `j` **descendent** al lui `i`,
`jk` muchie de întoarcere}

B se poate calcula recursiv

Muchii critice

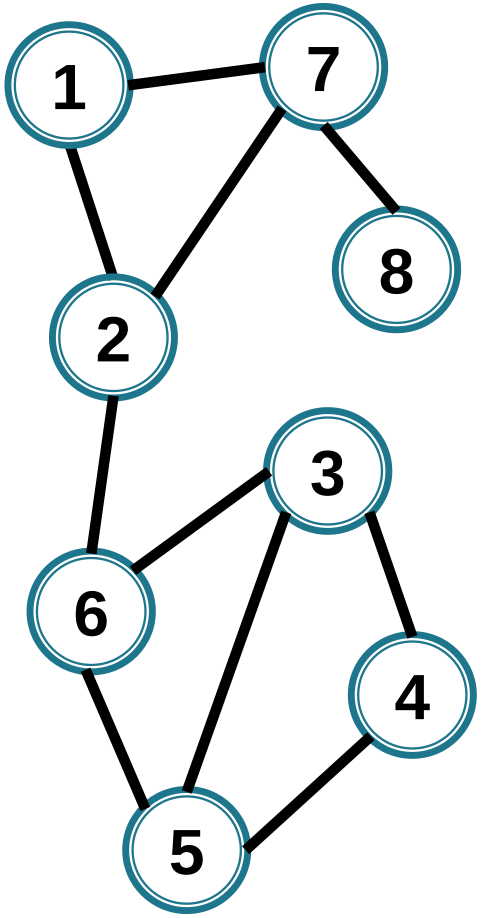
Cum calculăm eficient `niv_min[i]` ?

`niv_min[i]` = min { `nivel[i]`, A, B}

A = min {`nivel[k]` | `ik` muchie de întoarcere}

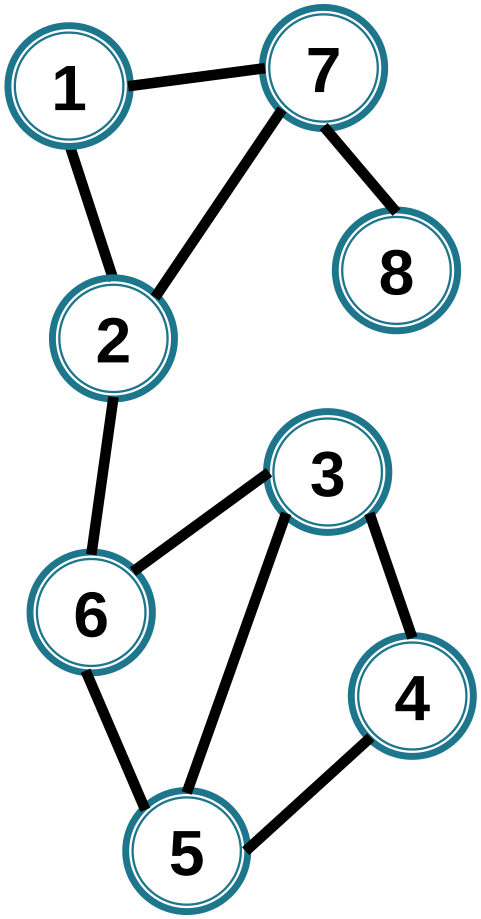
B = min {`nivel[k]` | `j` **descendent** al lui `i`, `jk` muchie de întoarcere}

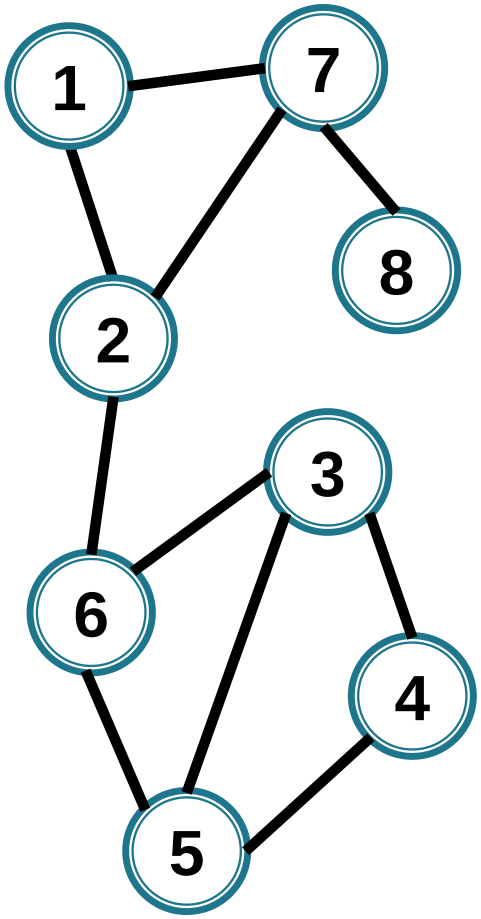
B = min {`niv_min[j]` | j fiu al lui i}


1
7
8
2
3
6
4
5

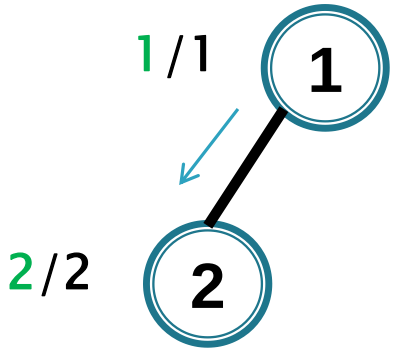
nivel/niv_min

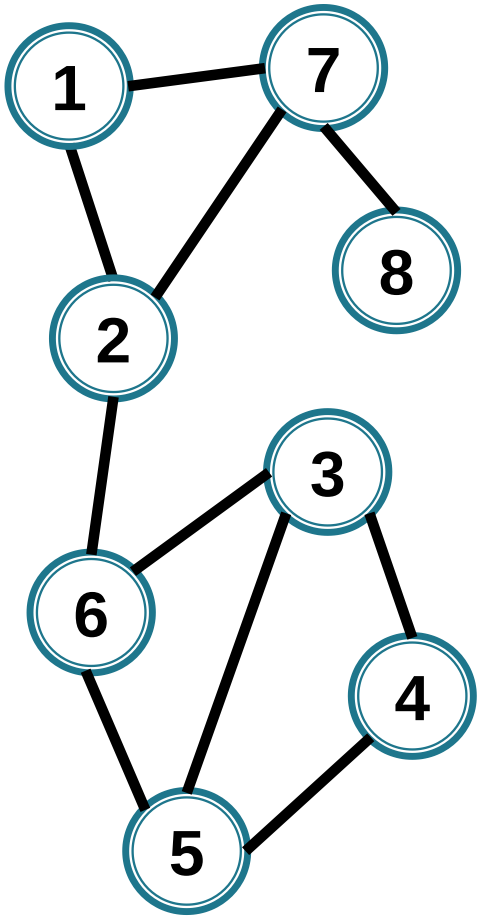
1/1



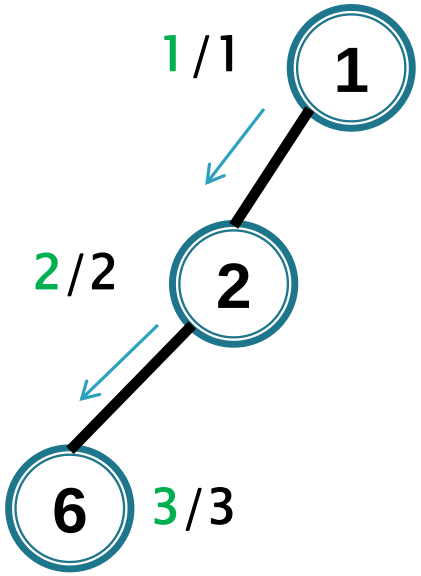


nivel/niv_min

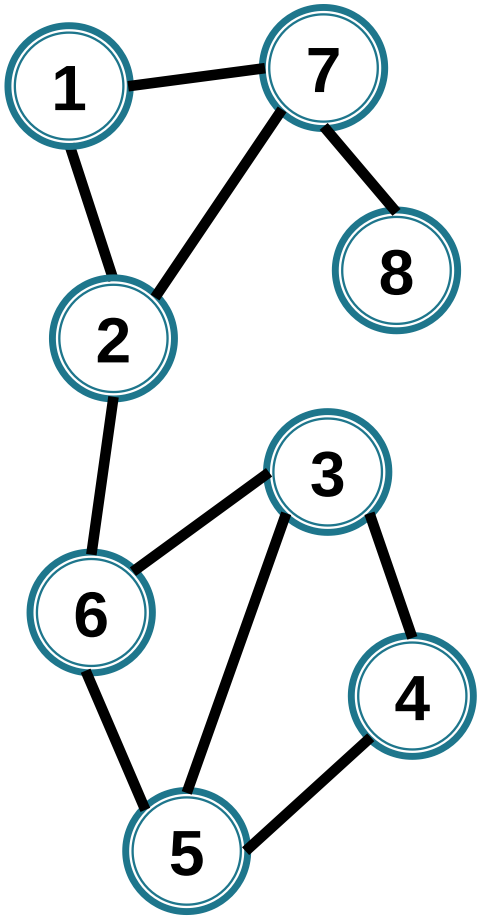


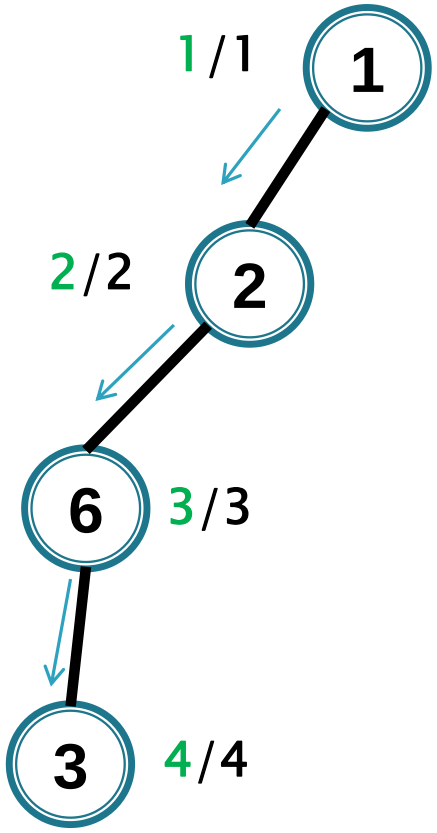


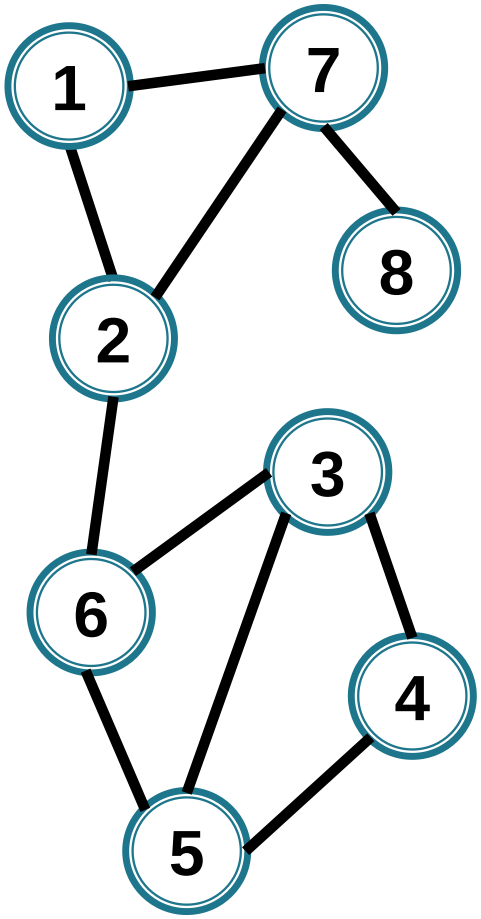
nivel/niv_min



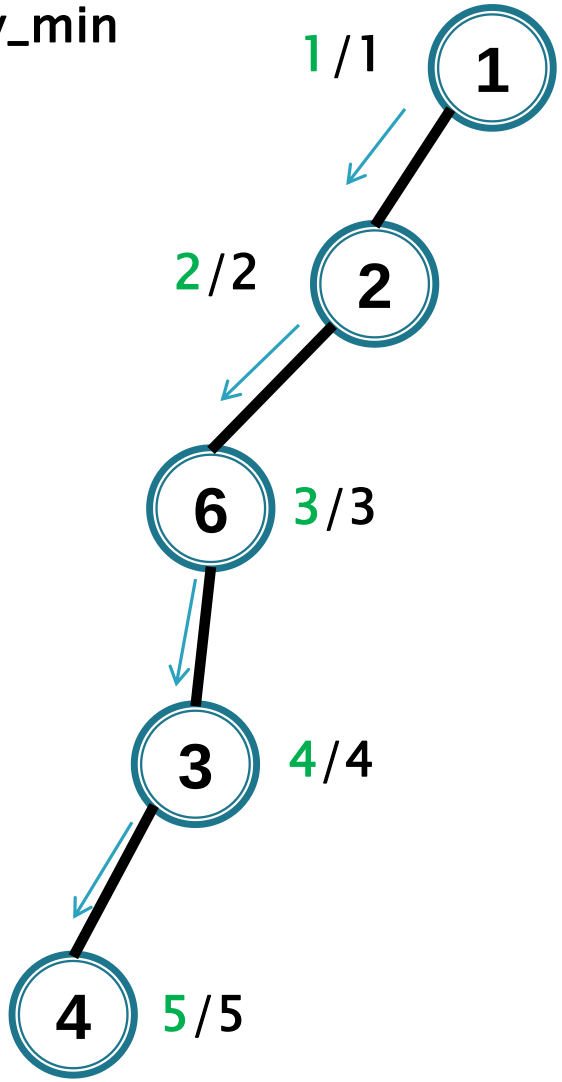
nivel/niv_min

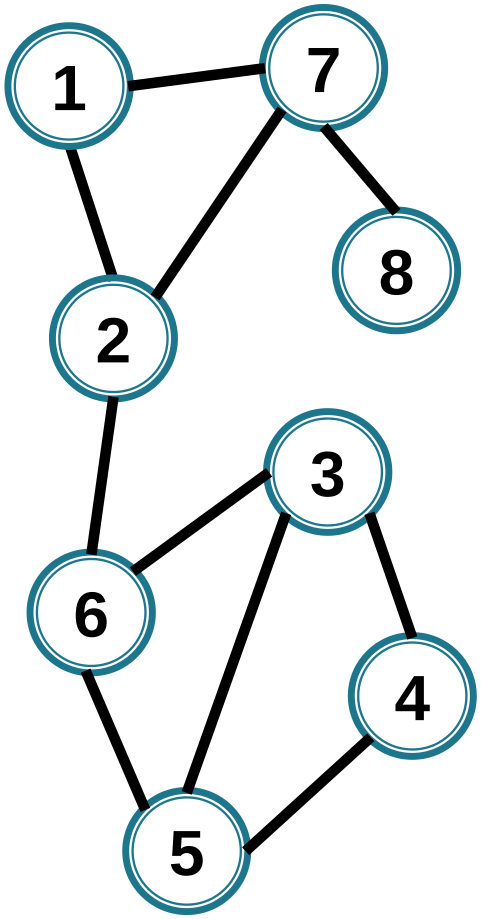




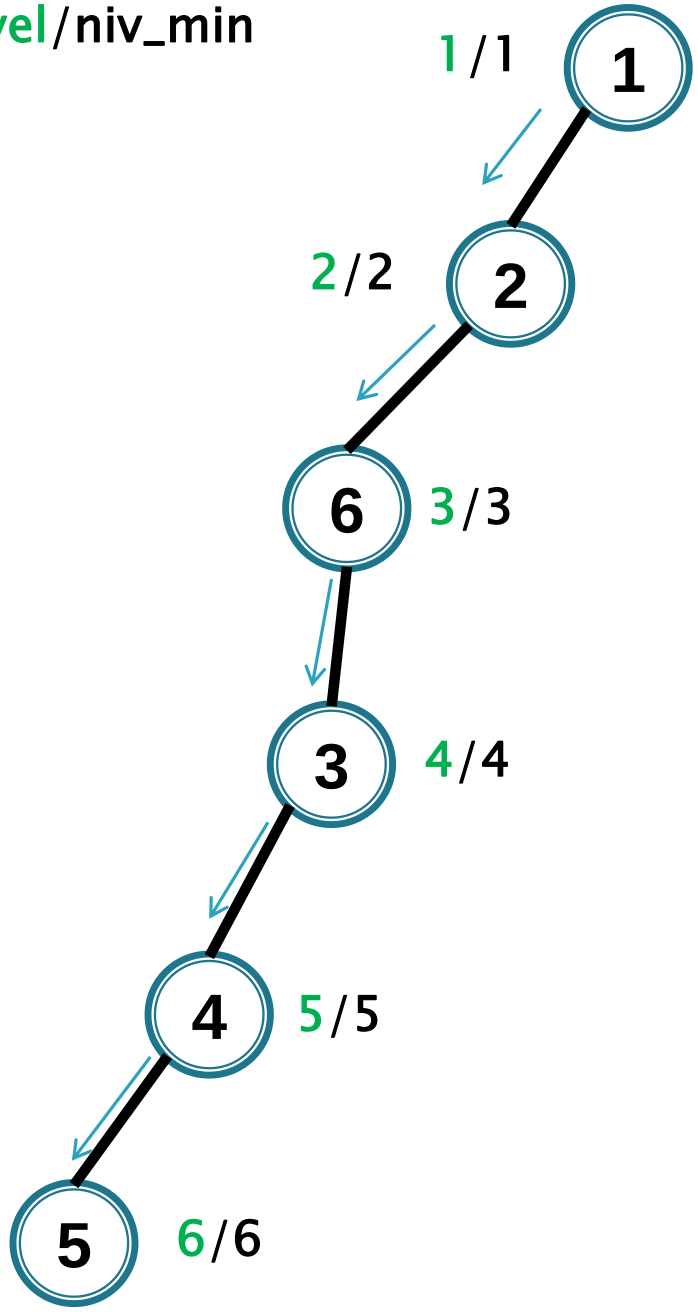


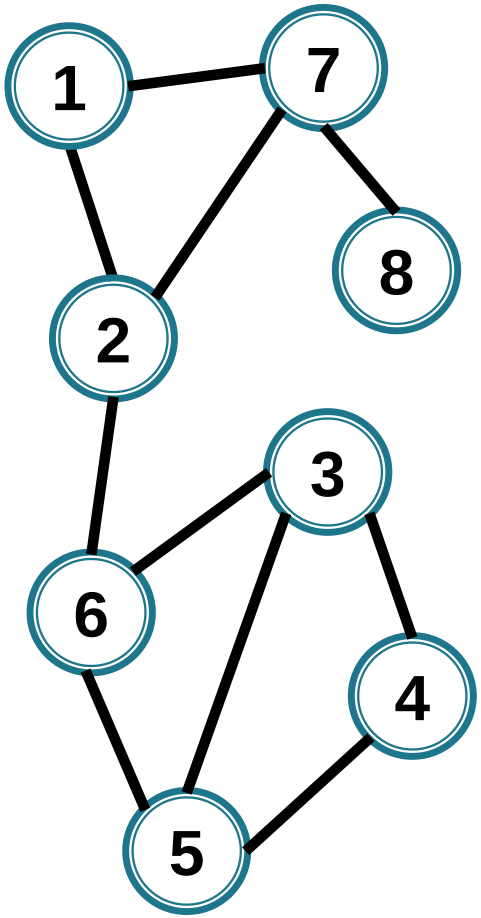
nivel/niv_min



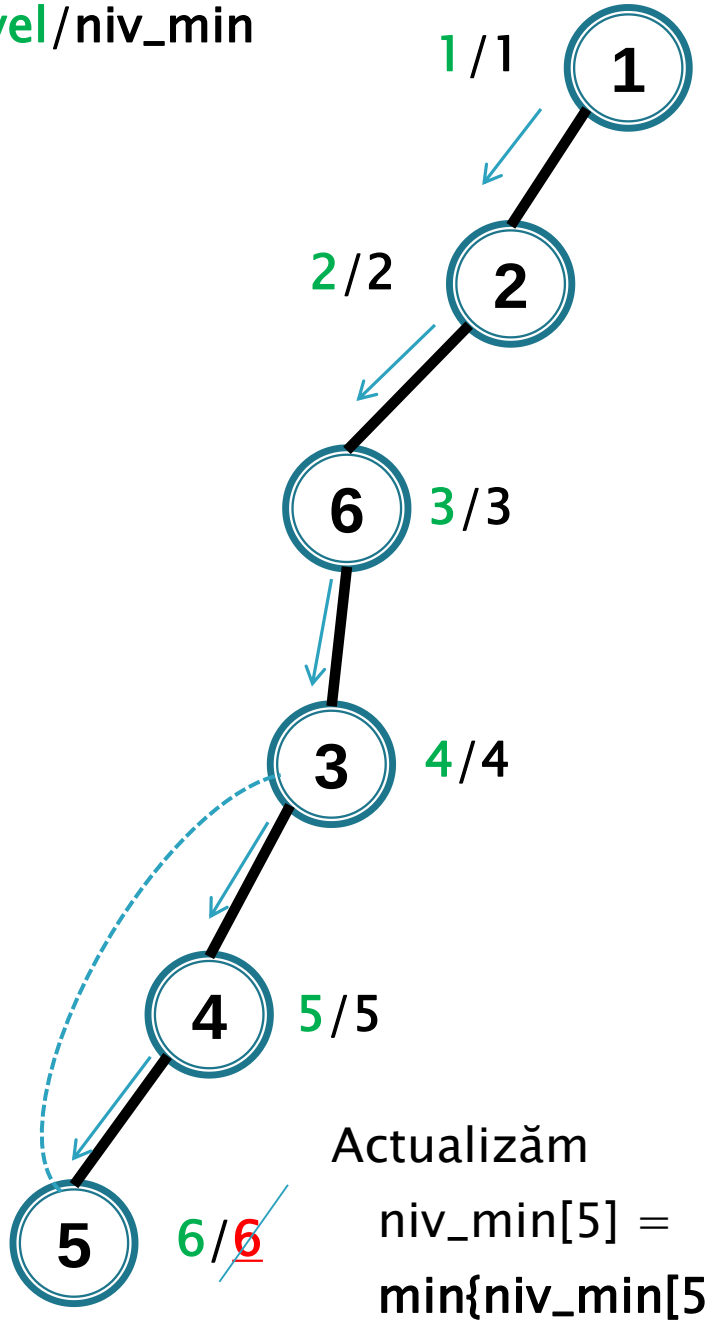


nivel/niv_min





nivel/niv_min

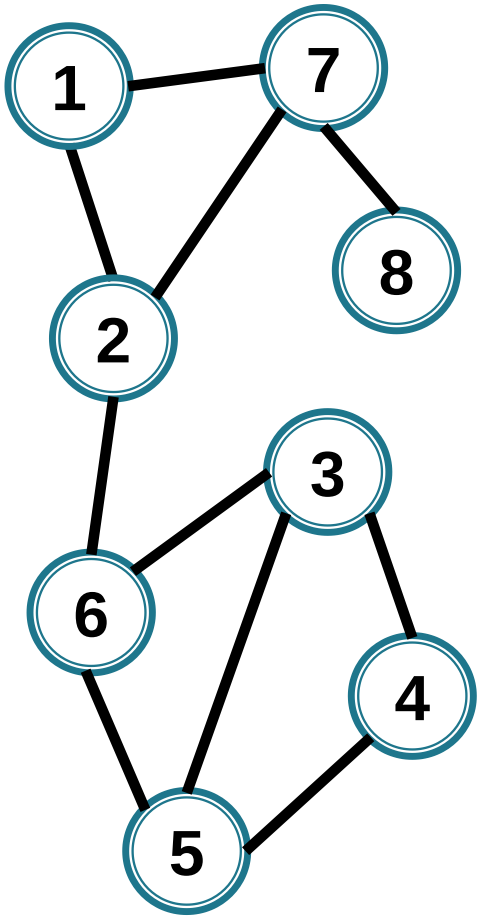


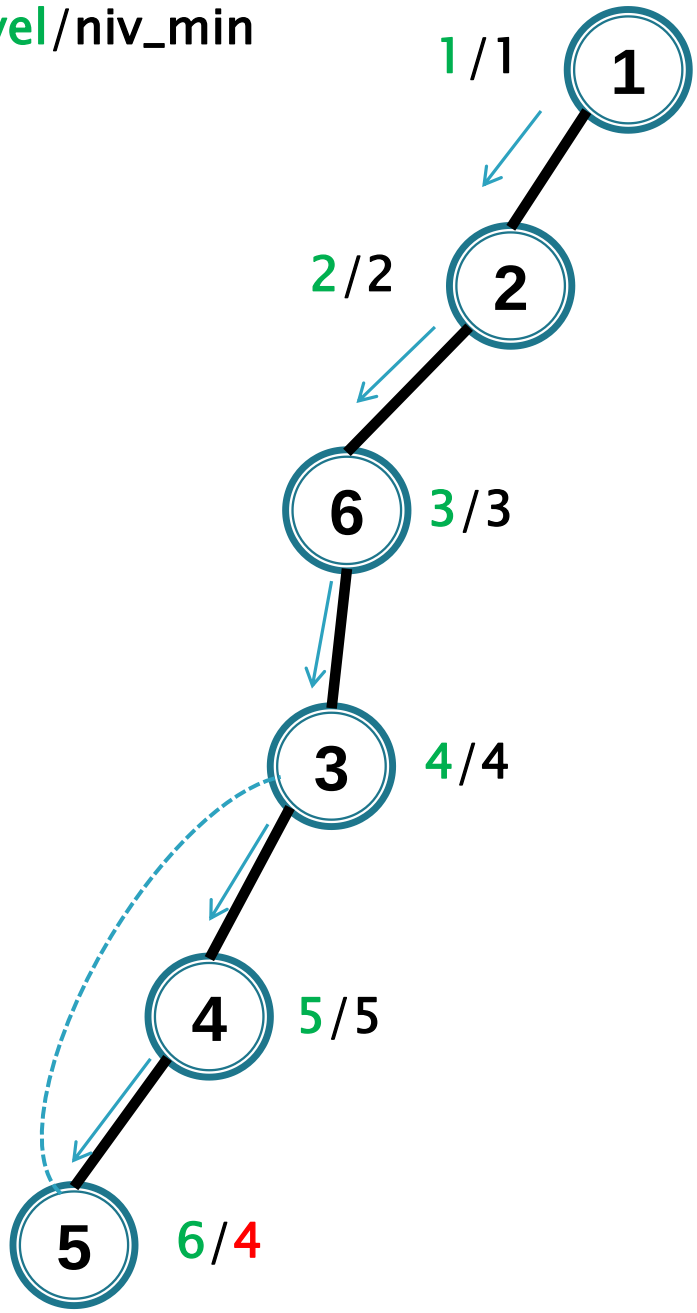
Actualizăm

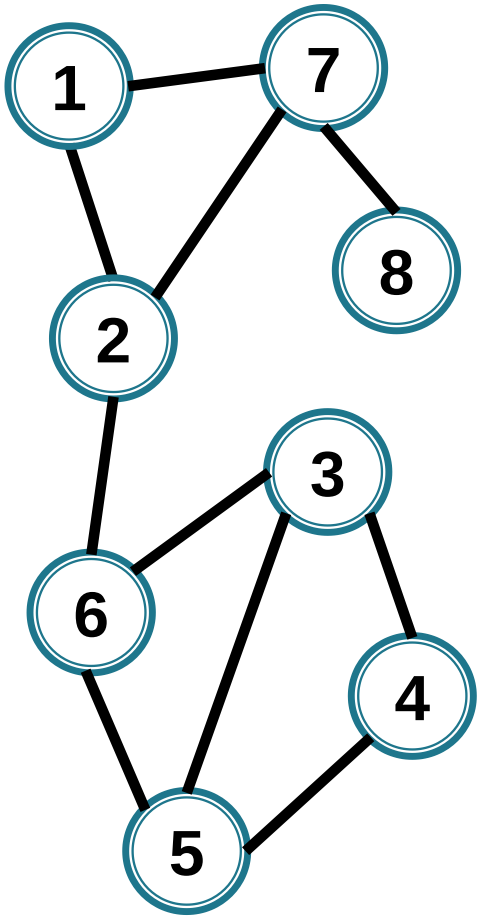
niv_min[5] =

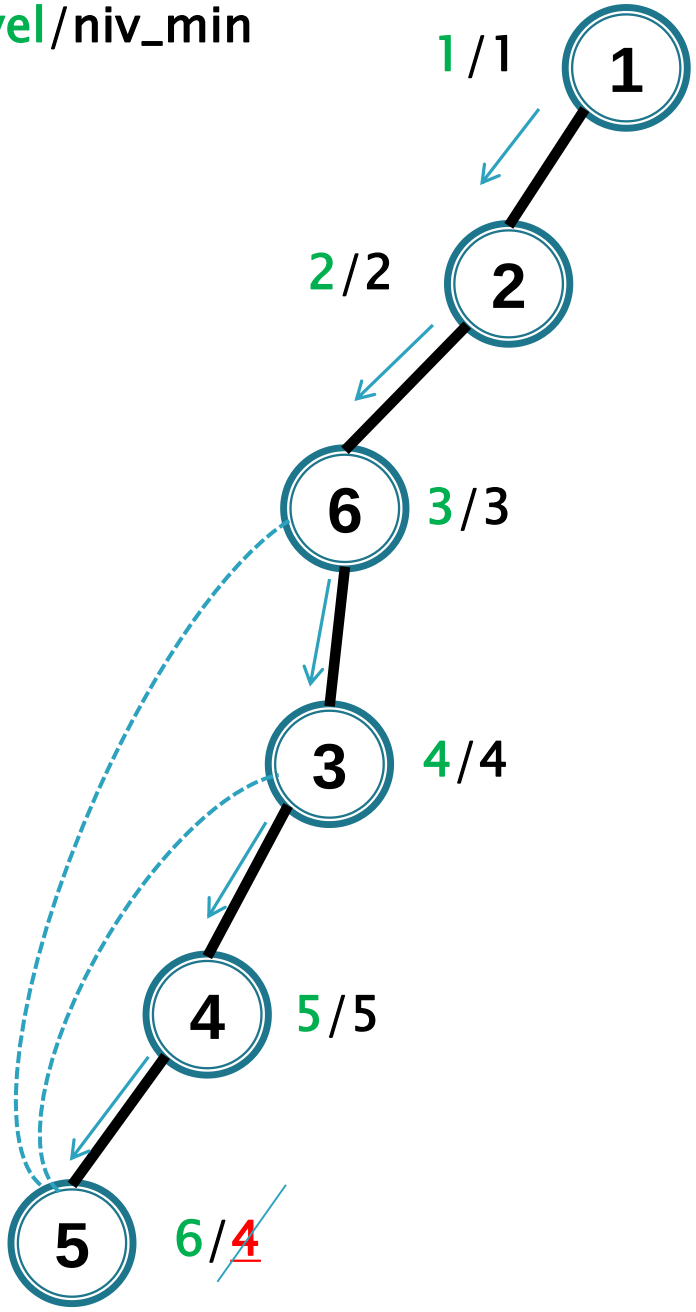
min{niv_min[5], nivel[3] }

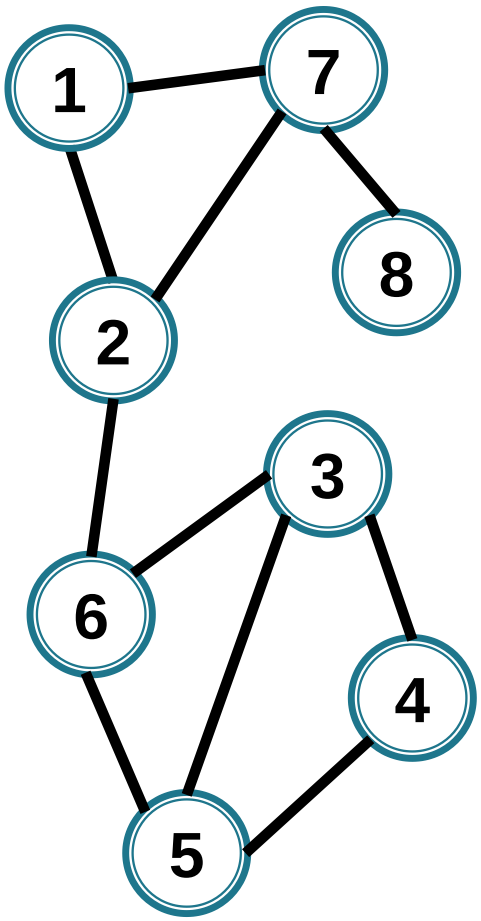
(cazul A)

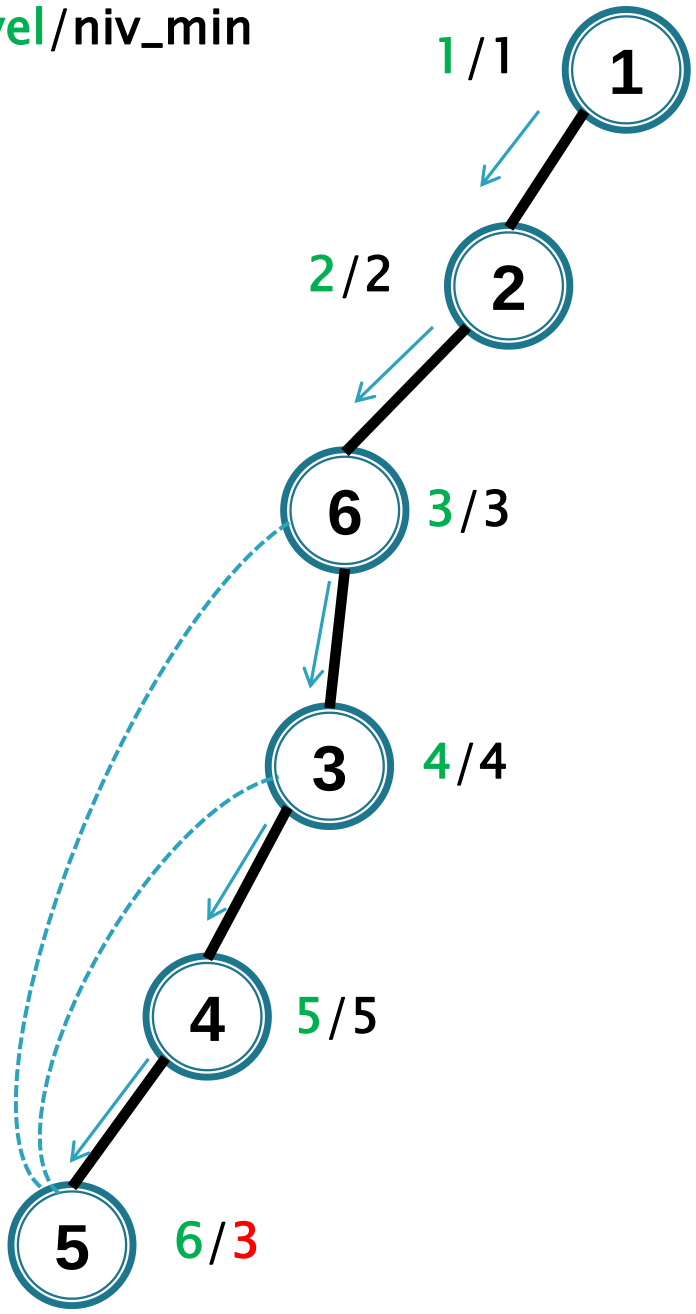

1
7
8
2
3
6
4
5

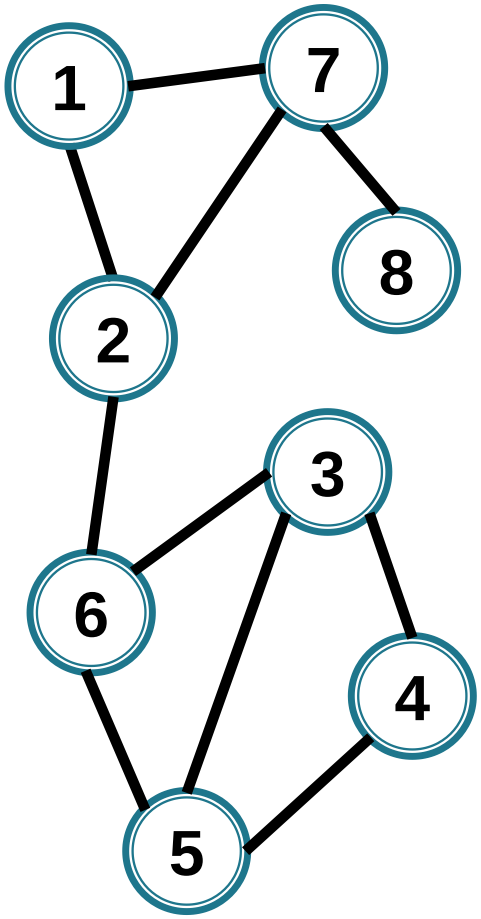

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/5
5
6/4

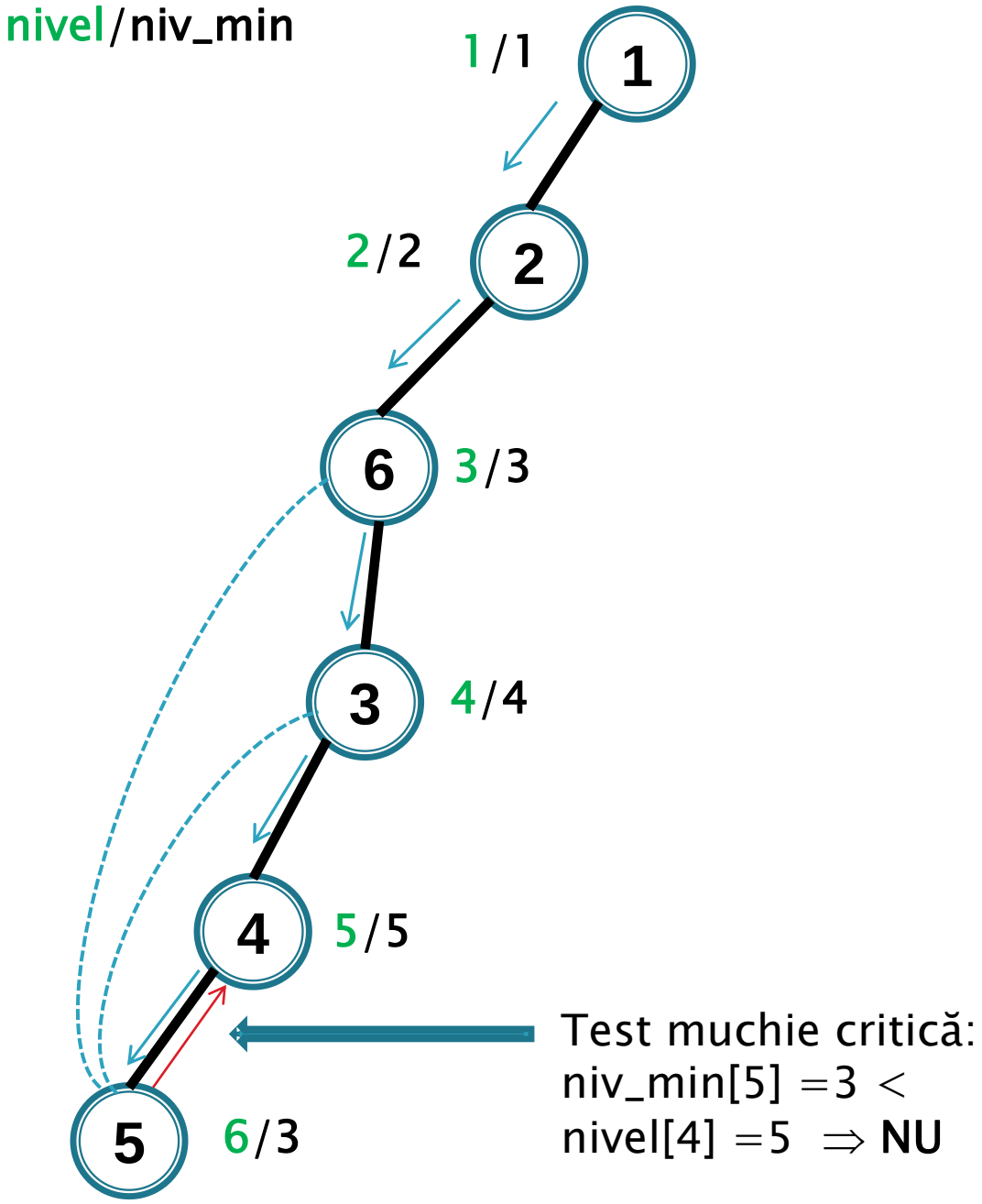

1
7
8
2
3
6
4
5

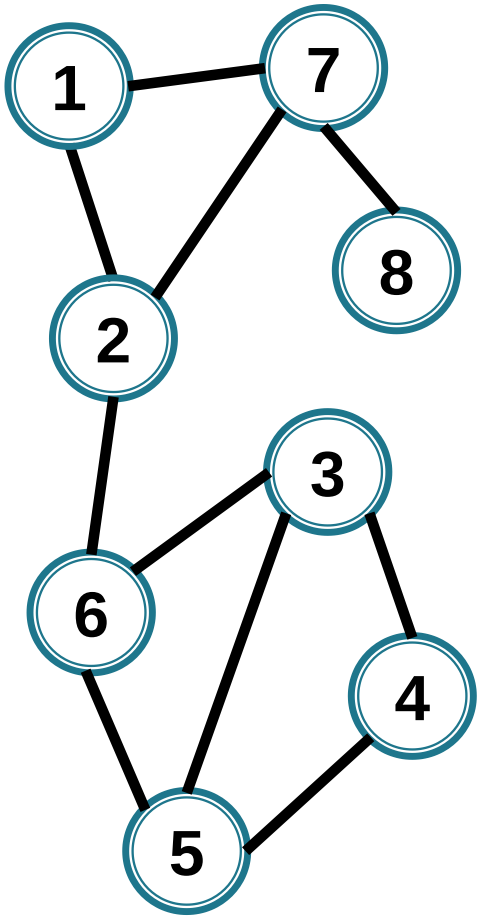

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/5
5
6/4

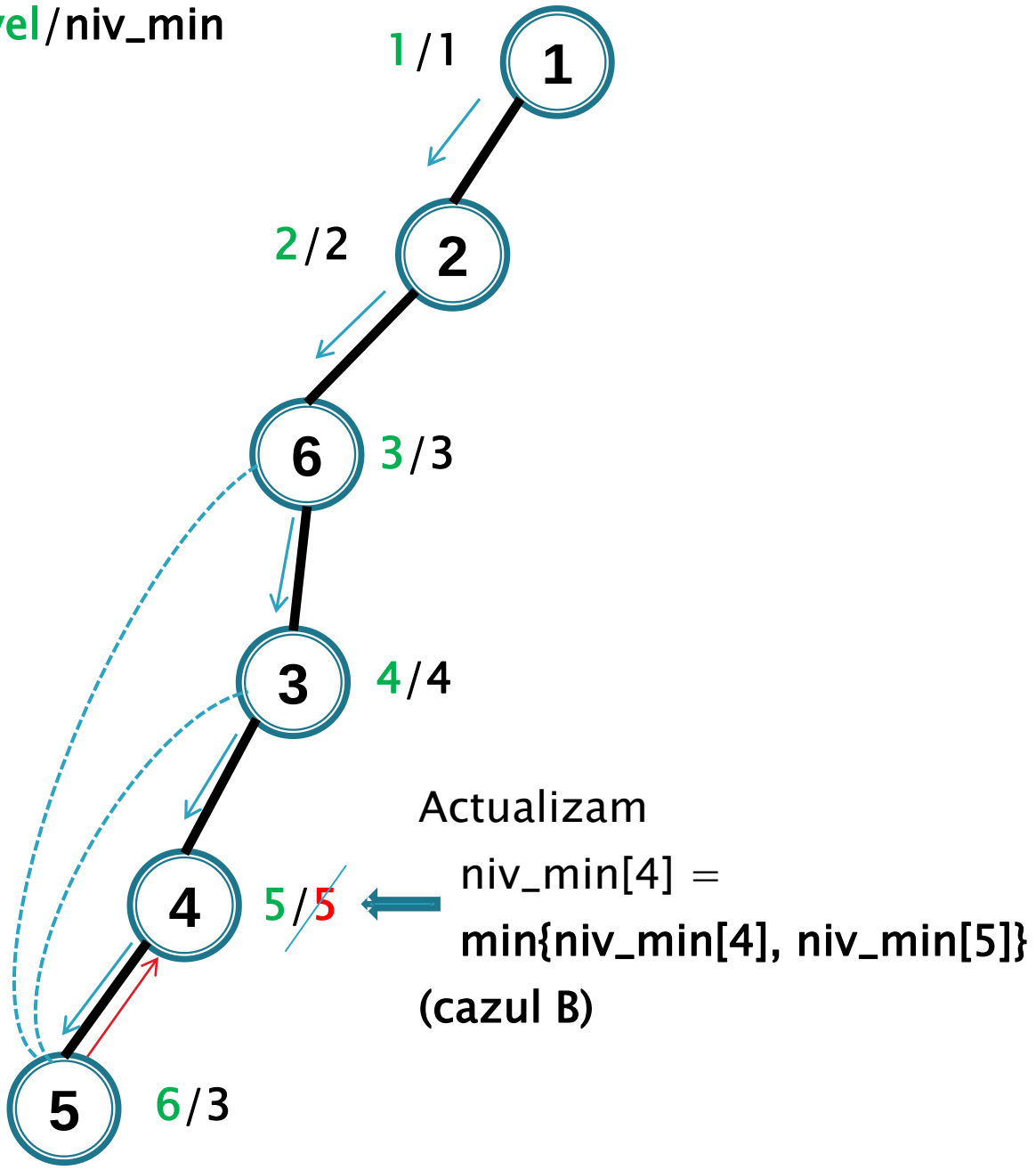

1
7
8
2
3
6
4
5

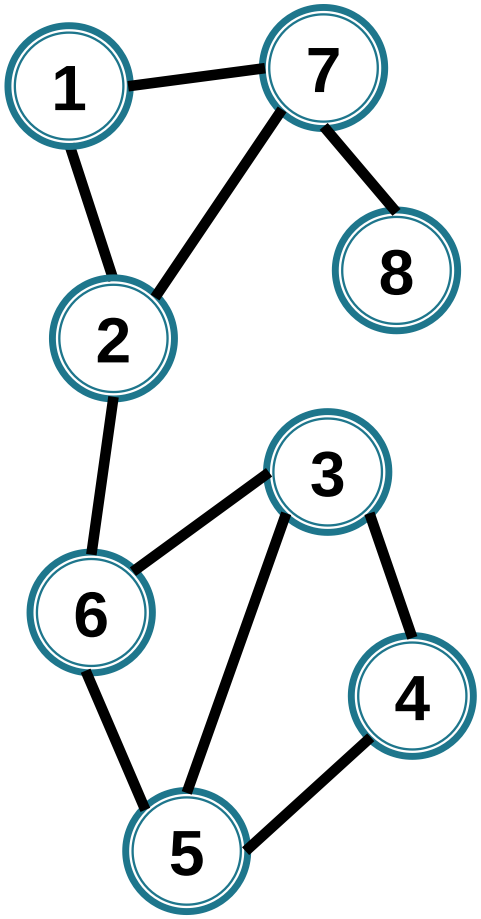

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/5
5
6/3

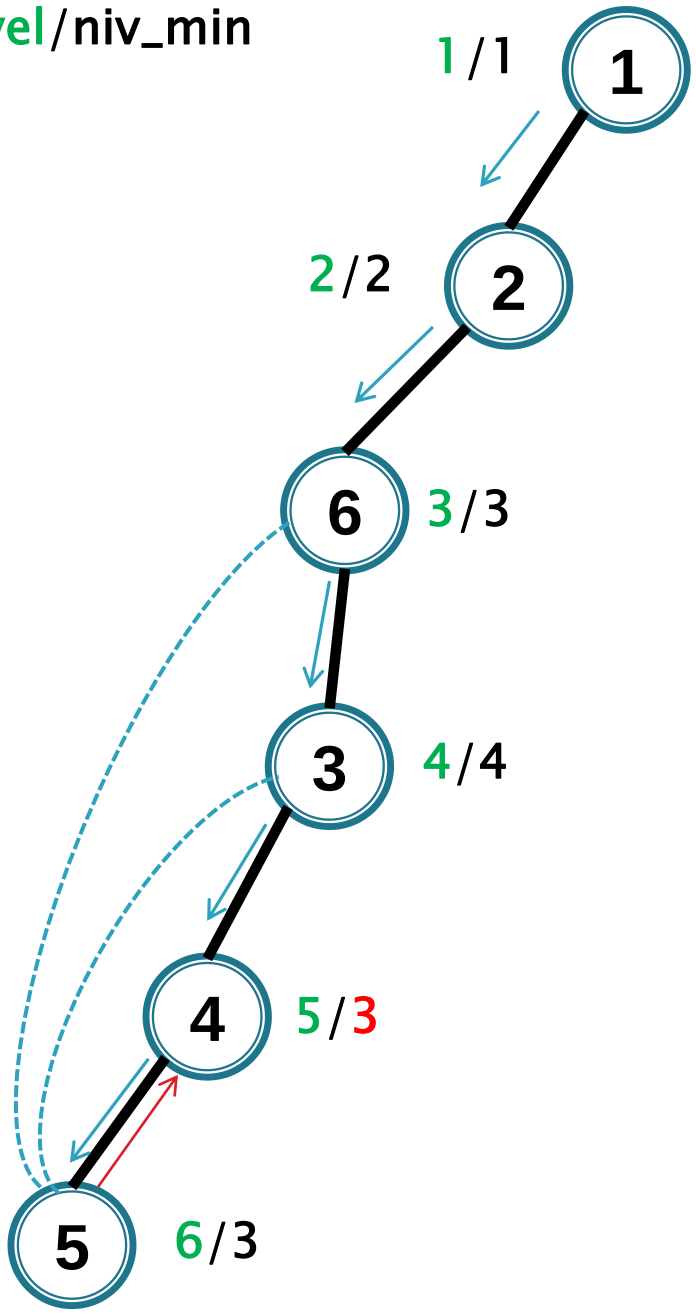

1
7
8
2
3
6
4
5

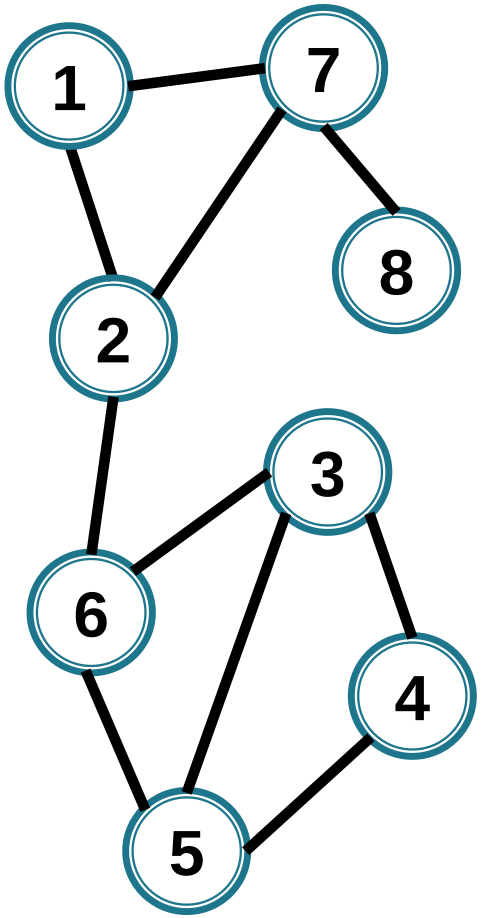

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/5
5
6/3
Test muchie critică:
niv_min[5] =3 <
nivel[4] =5 ⇒ NU

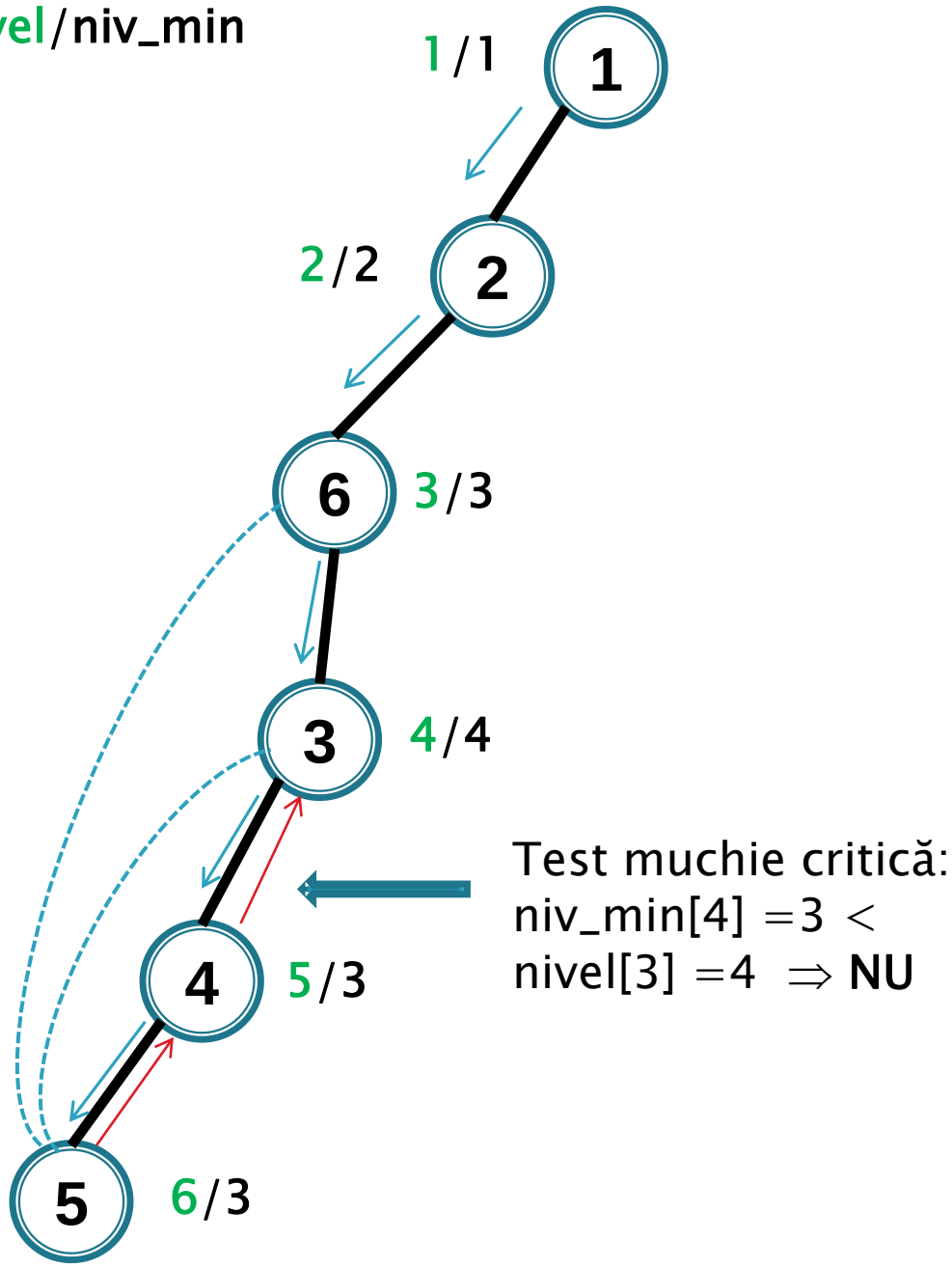

1
7
8
2
3
6
4
5

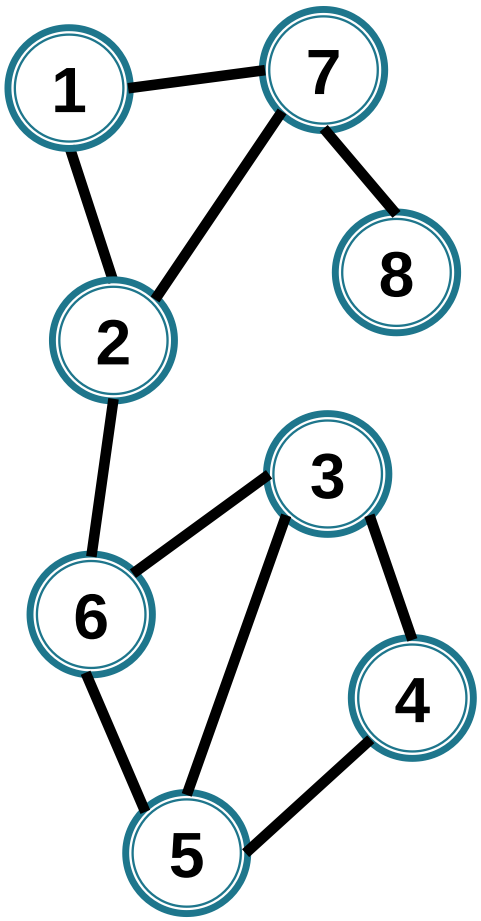

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/5
5
6/3
Actualizam
niv_min[4] =
min{niv_min[4], niv_min[5]}
(cazul B)

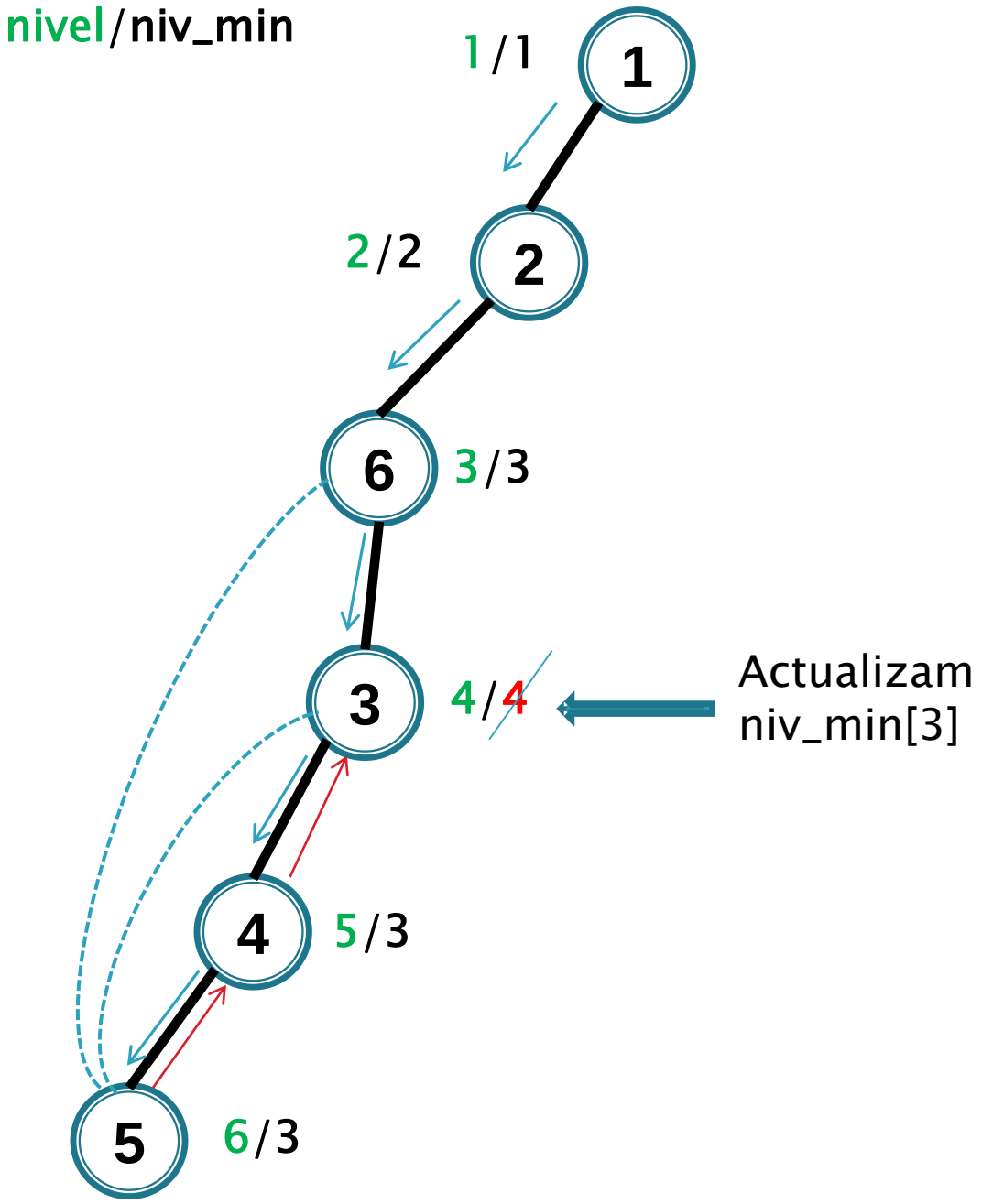

1
7
8
2
3
6
4
5

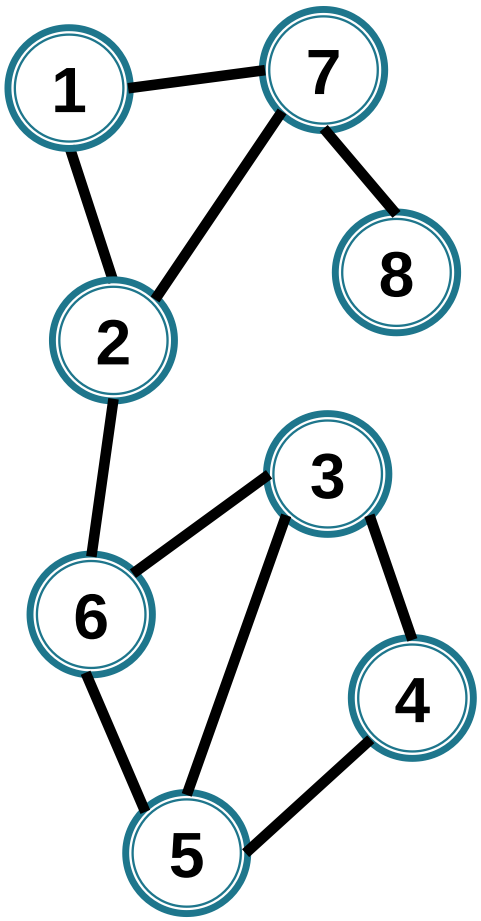

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/3
5
6/3

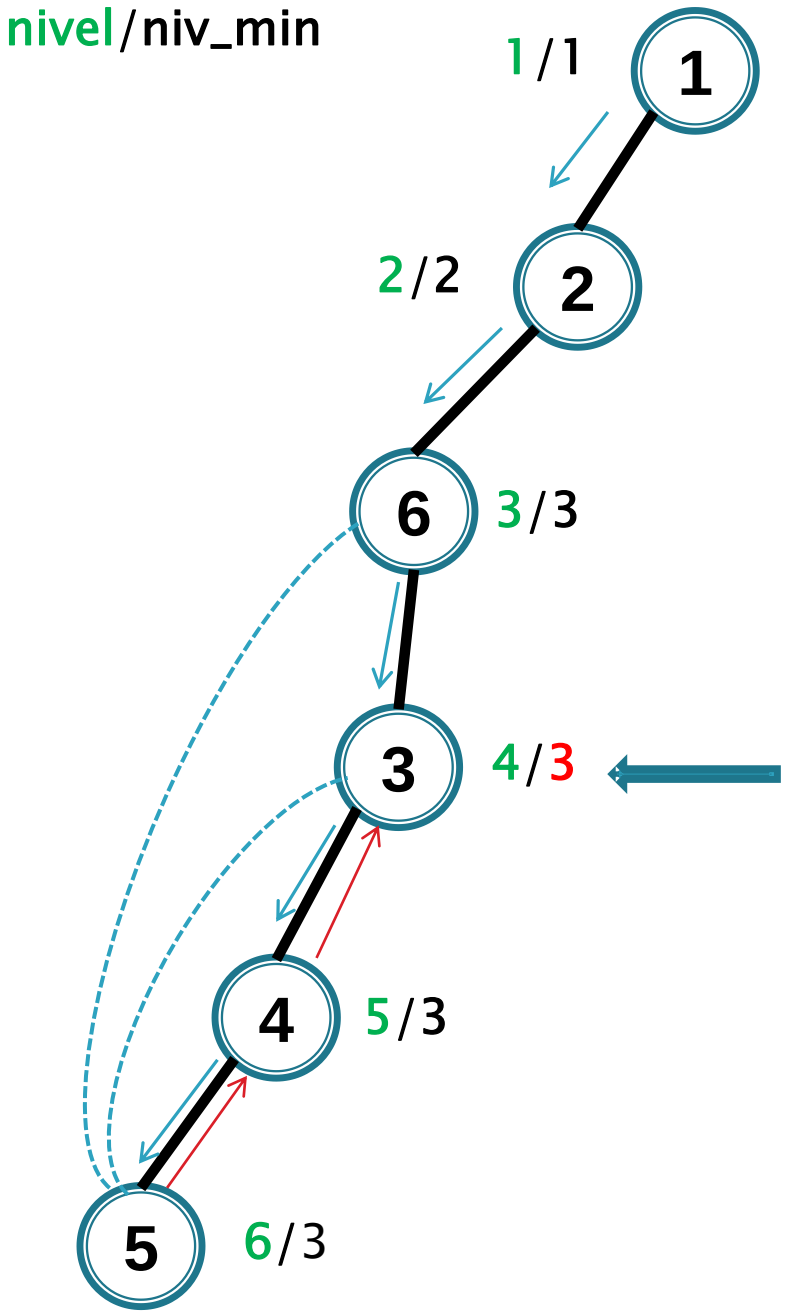

1
7
8
2
3
6
4
5

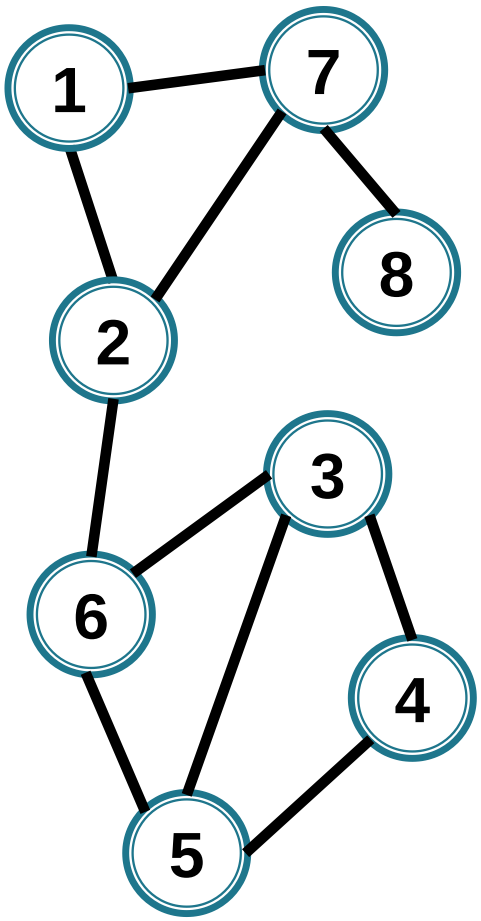

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
4
5/3
5
6/3
Test muchie critică:
niv_min[4] =3 <
nivel[3] =4 ⇒ NU

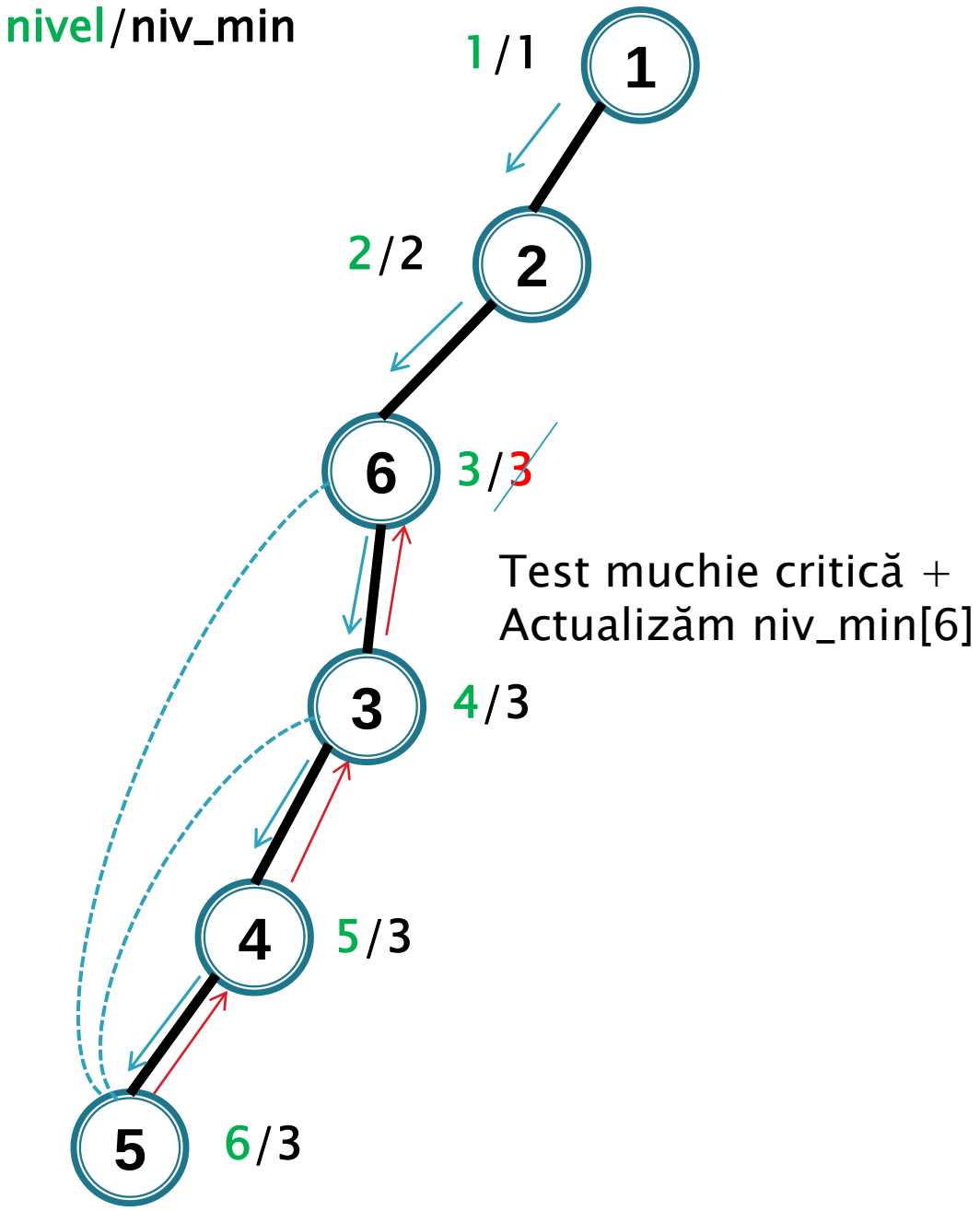

1
7
8
2
3
6
4
5

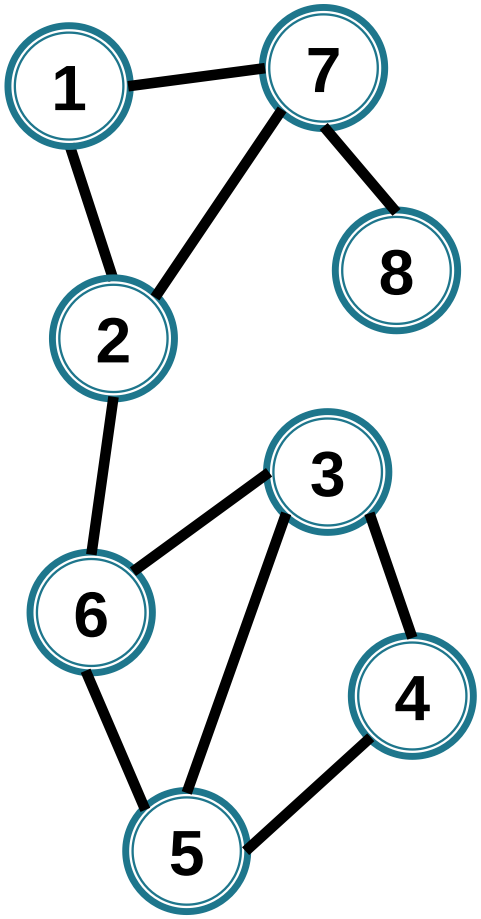

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/4
Actualizam
niv_min[3]
4
5/3
5
6/3

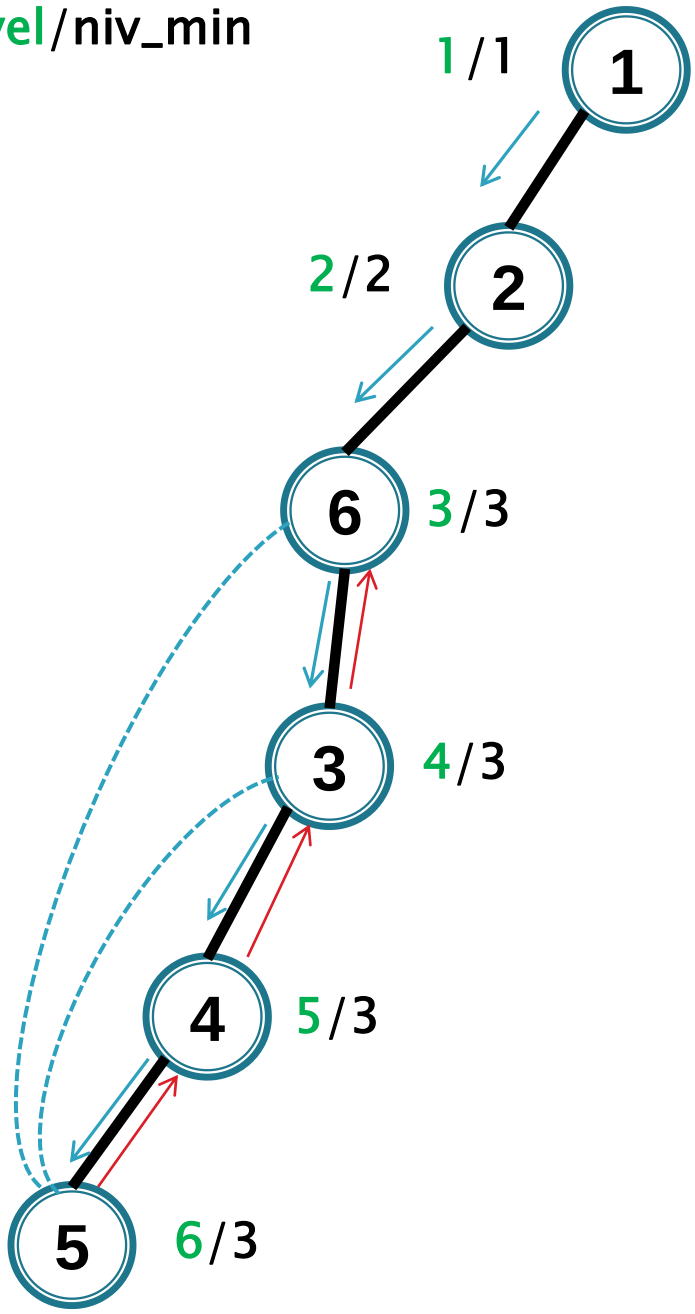

1
7
8
2
3
6
4
5

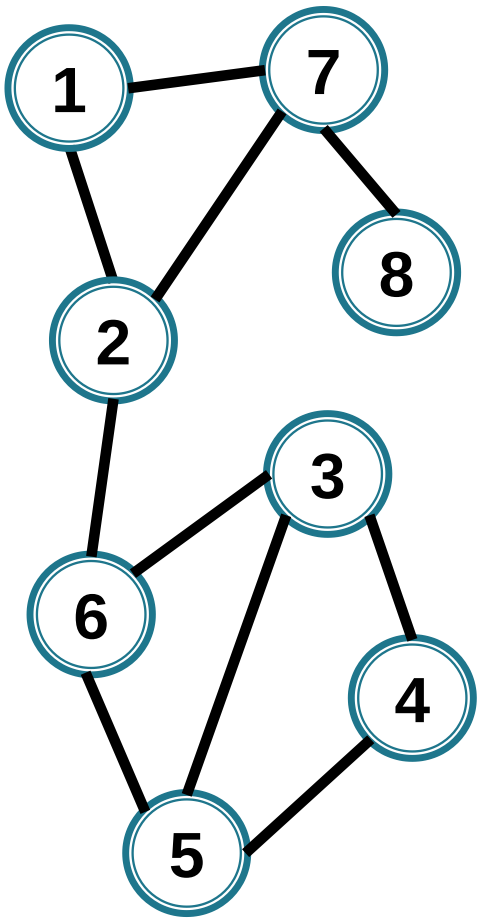

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/3
4
5/3
5
6/3

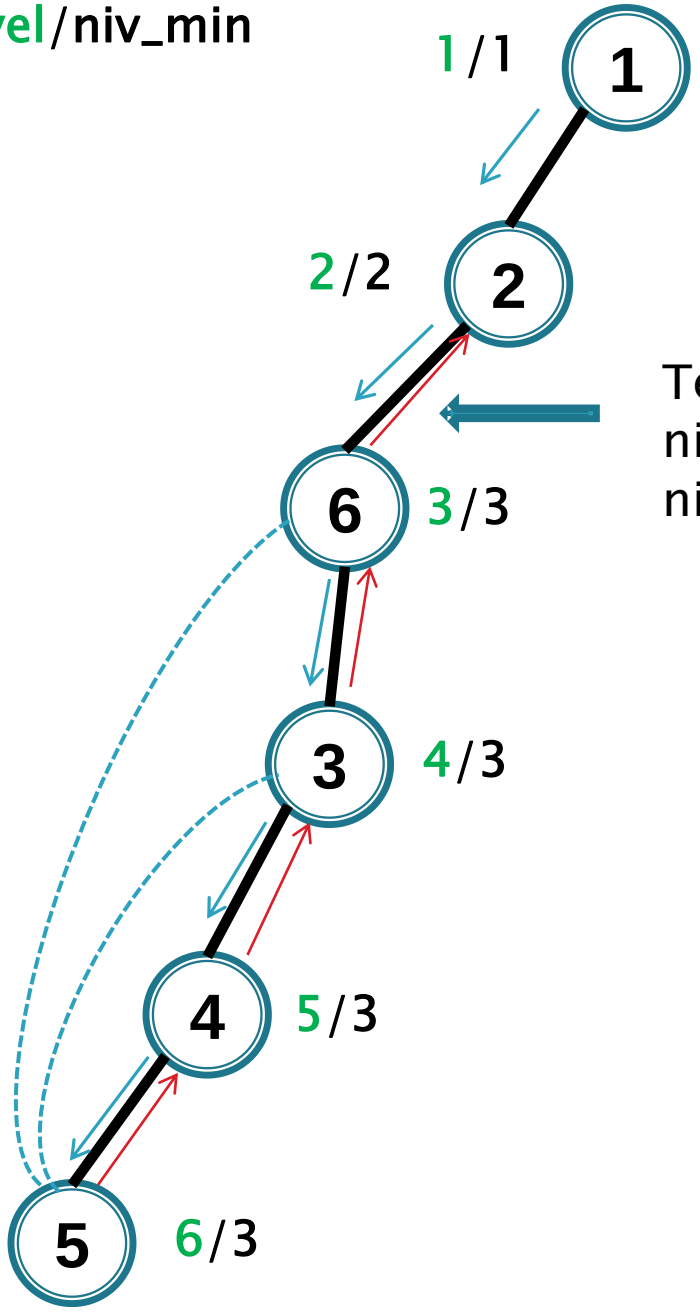

1
7
8
2
3
6
4
5


nivel/niv_min
1/1
1
2/2
2
6
3/3
Test muchie critică +
Actualizăm niv_min[6]
3
4/3
4
5/3
5
6/3

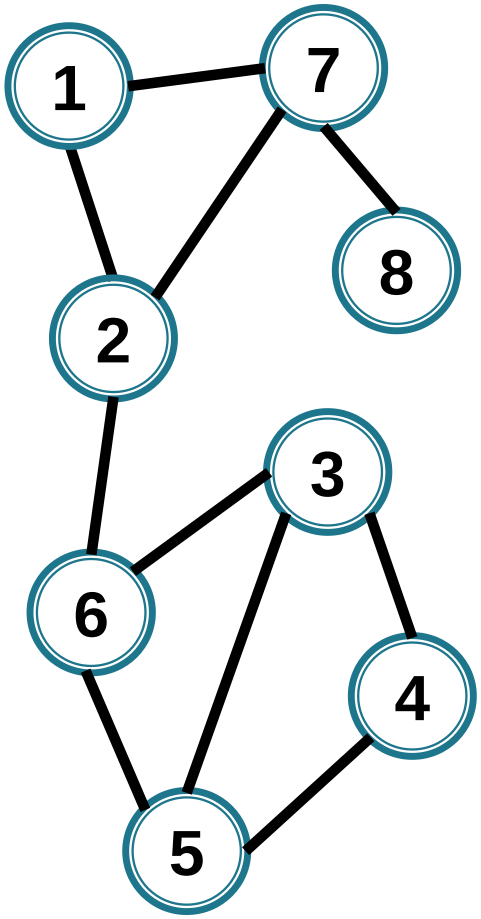

1
7
8
2
3
6
4
5

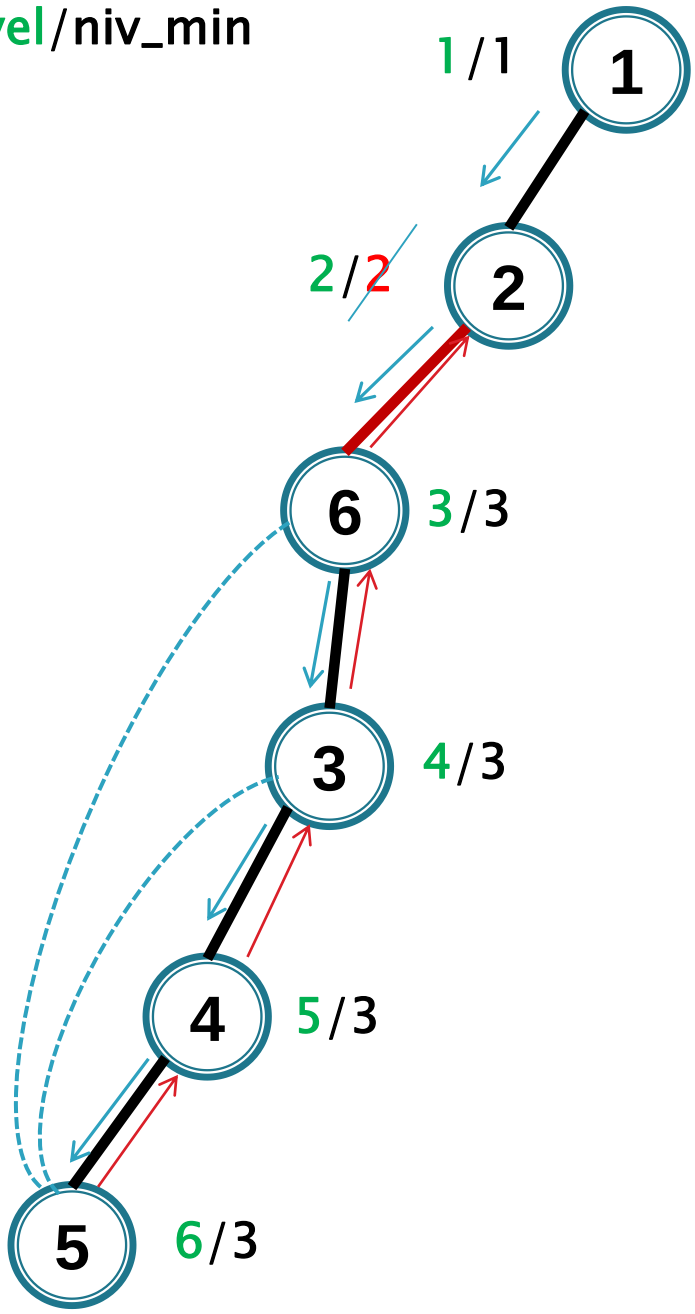

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/3
4
5/3
5
6/3

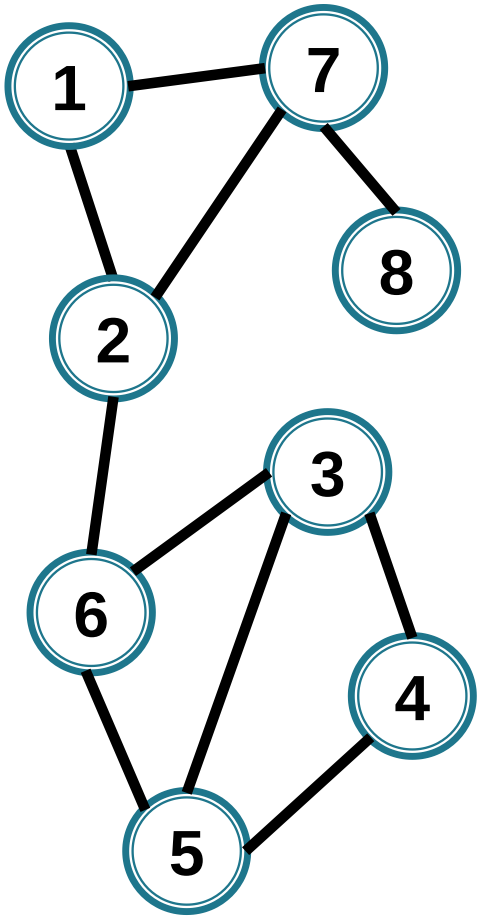


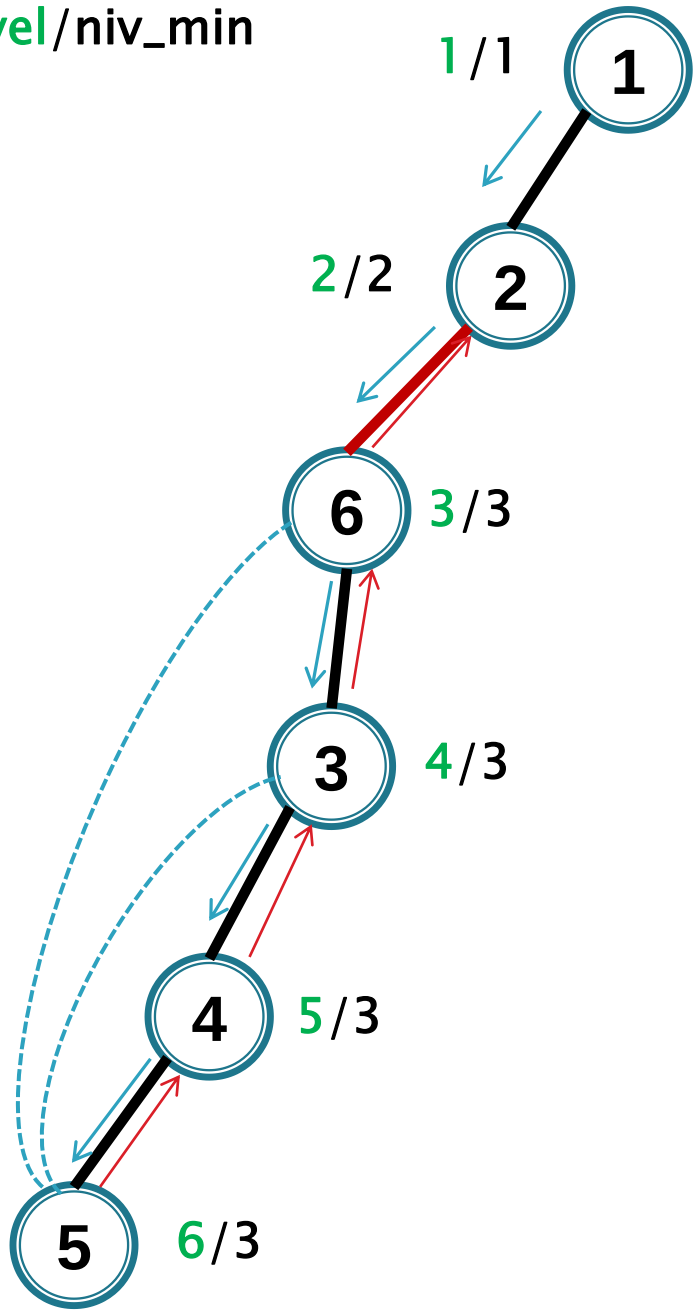


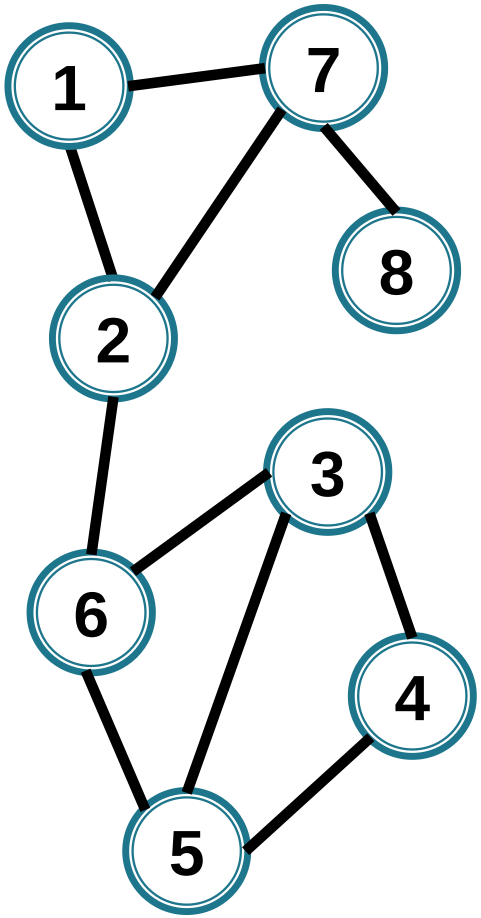
Test muchie critică:
niv_min[6] =3 >
nivel[2] =2 ⇒ **DA**

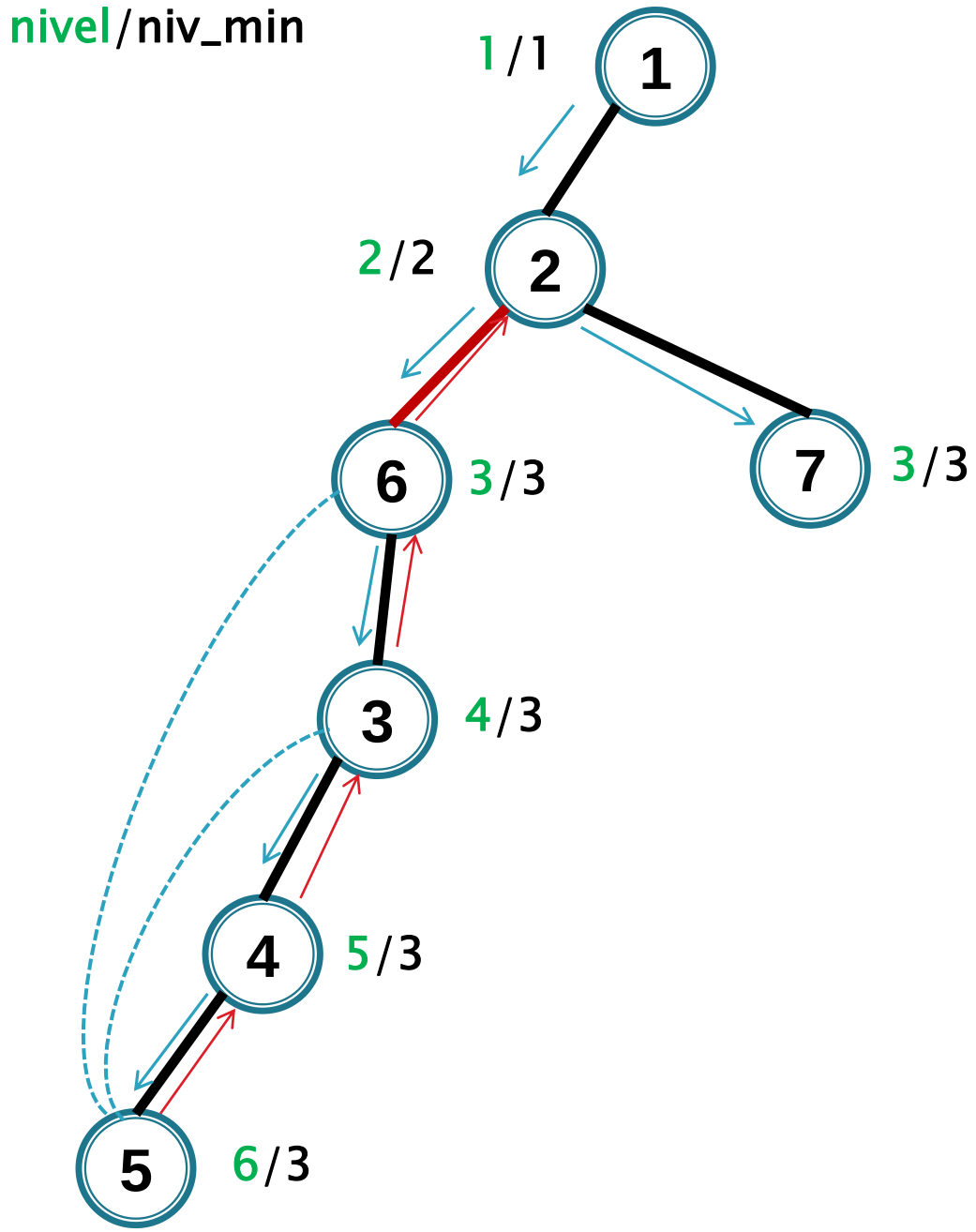

1
7
8
2
3
6
4
5

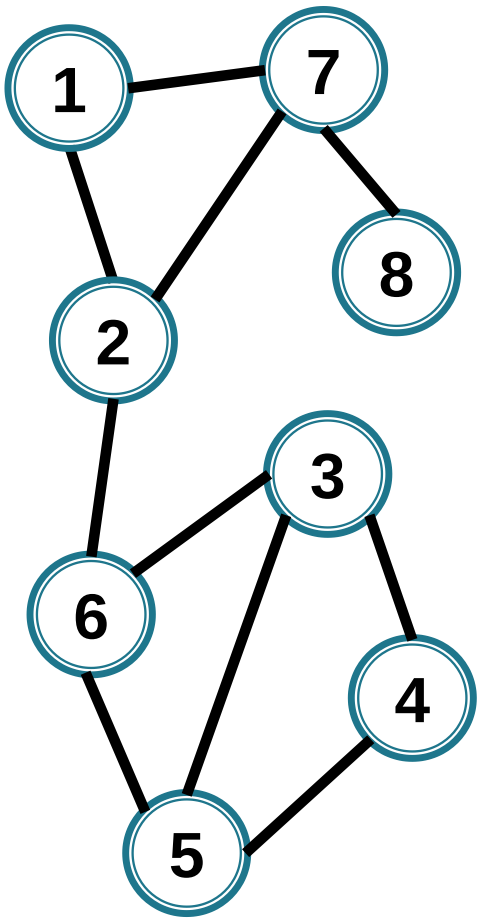

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/3
4
5/3
5
6/3

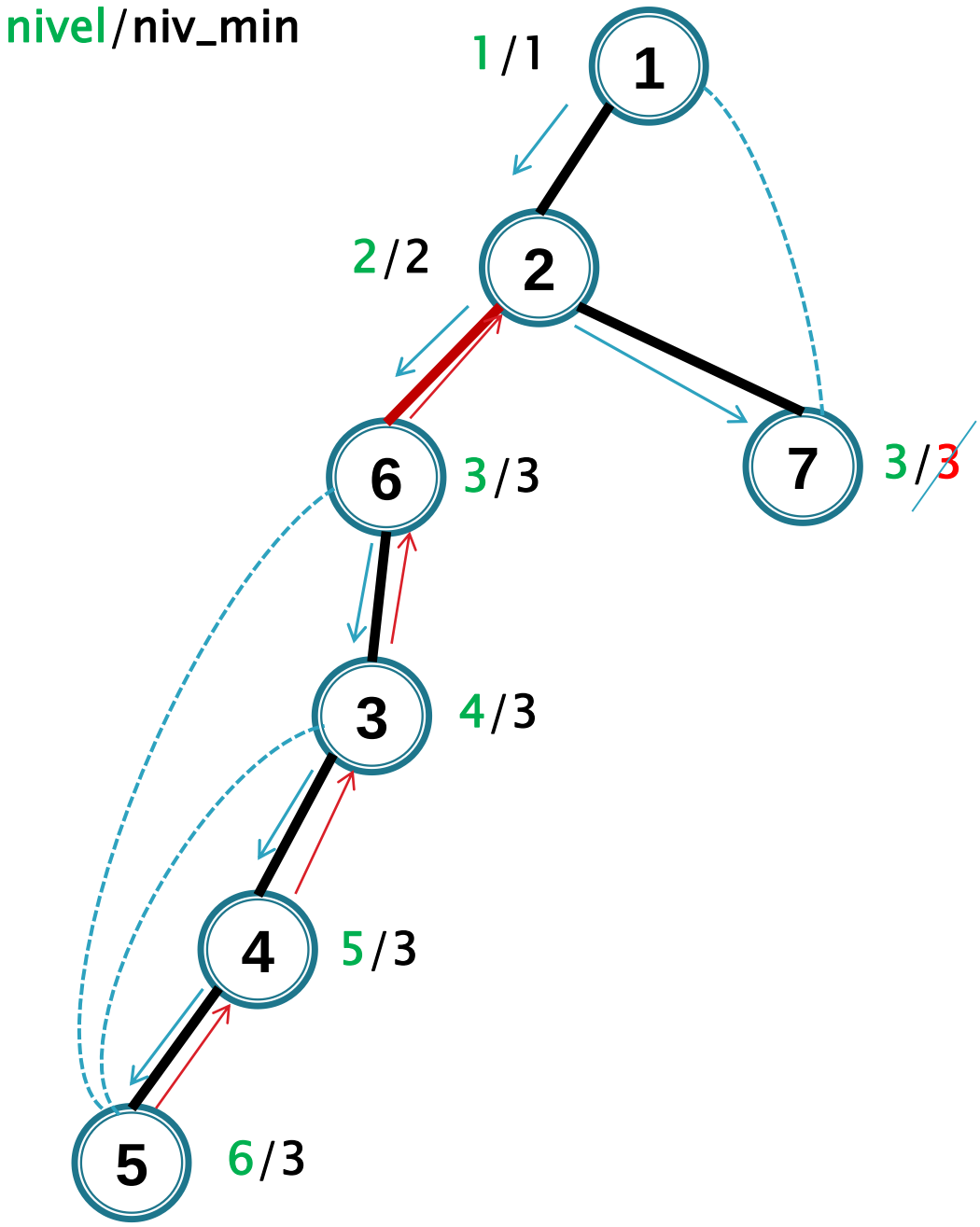

1
7
8
2
3
6
4
5

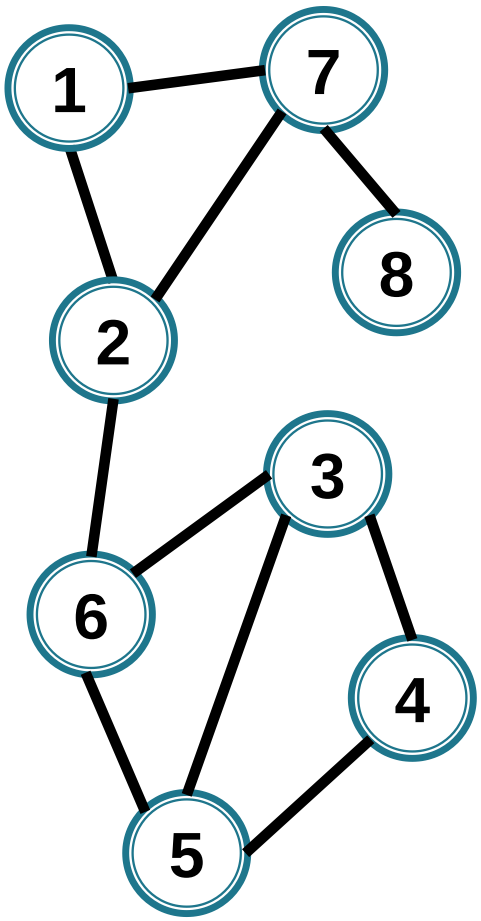

nivel/niv_min
1/1
1
2/2
2
6
3/3
3
4/3
4
5/3
5
6/3

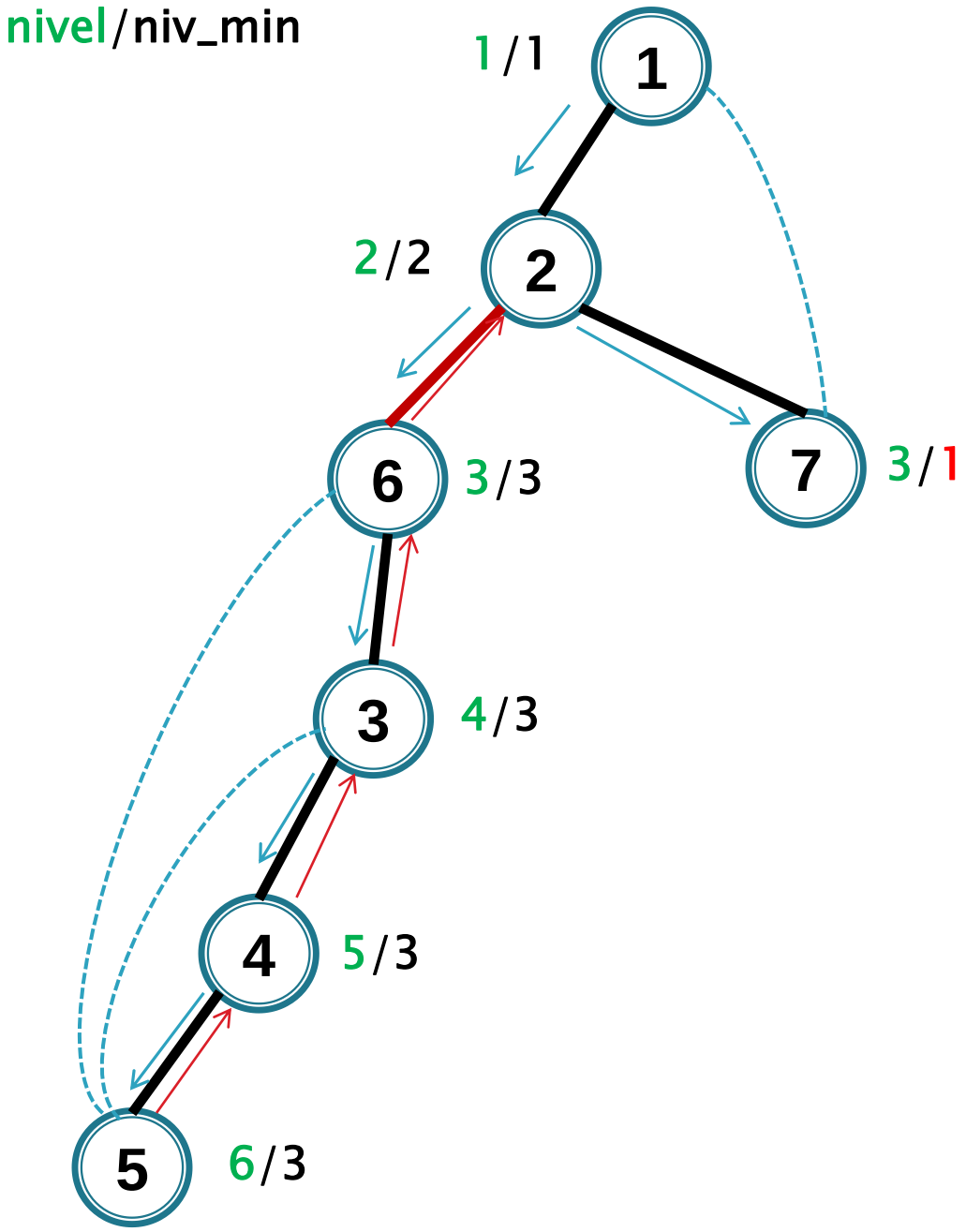

1
7
8
2
3
6
4
5

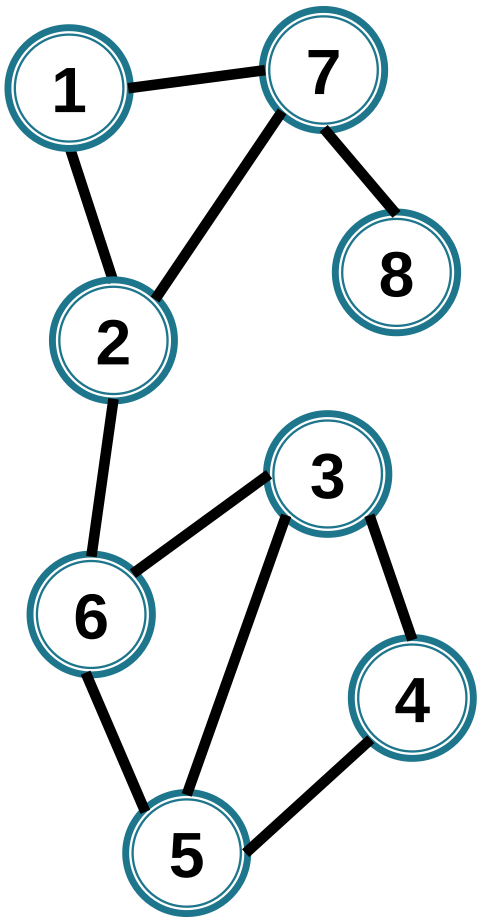

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/3
3
4/3
4
5/3
5
6/3

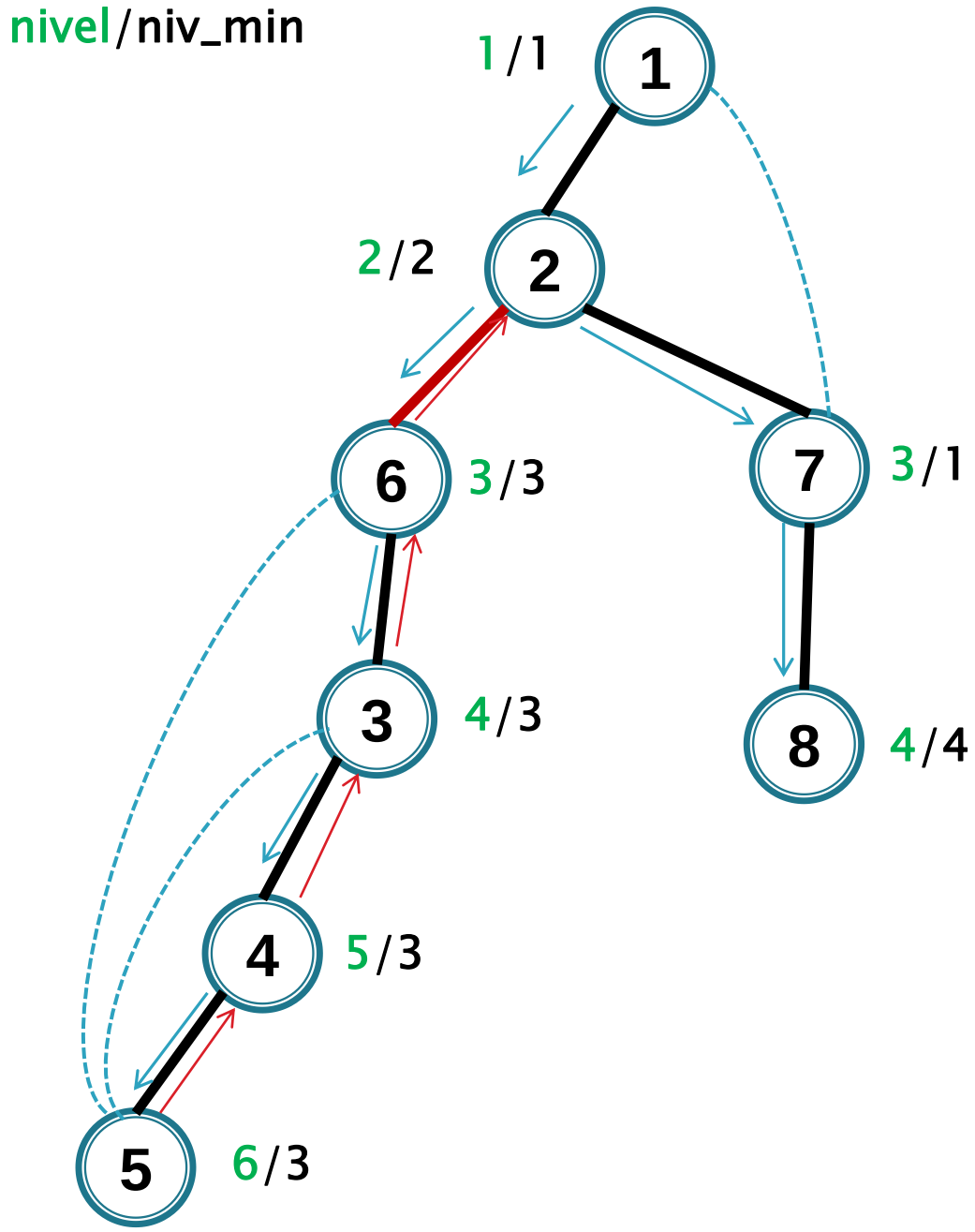

1
7
8
2
3
6
4
5

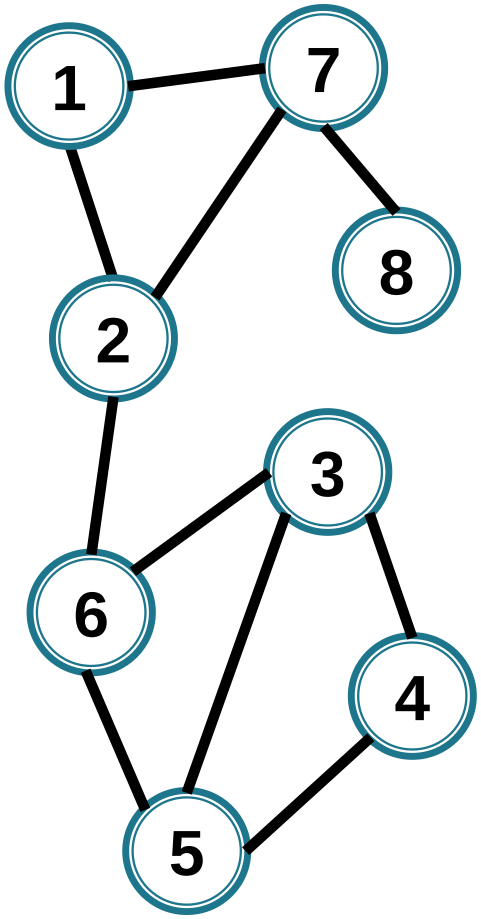

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/3
3
4/3
4
5/3
5
6/3

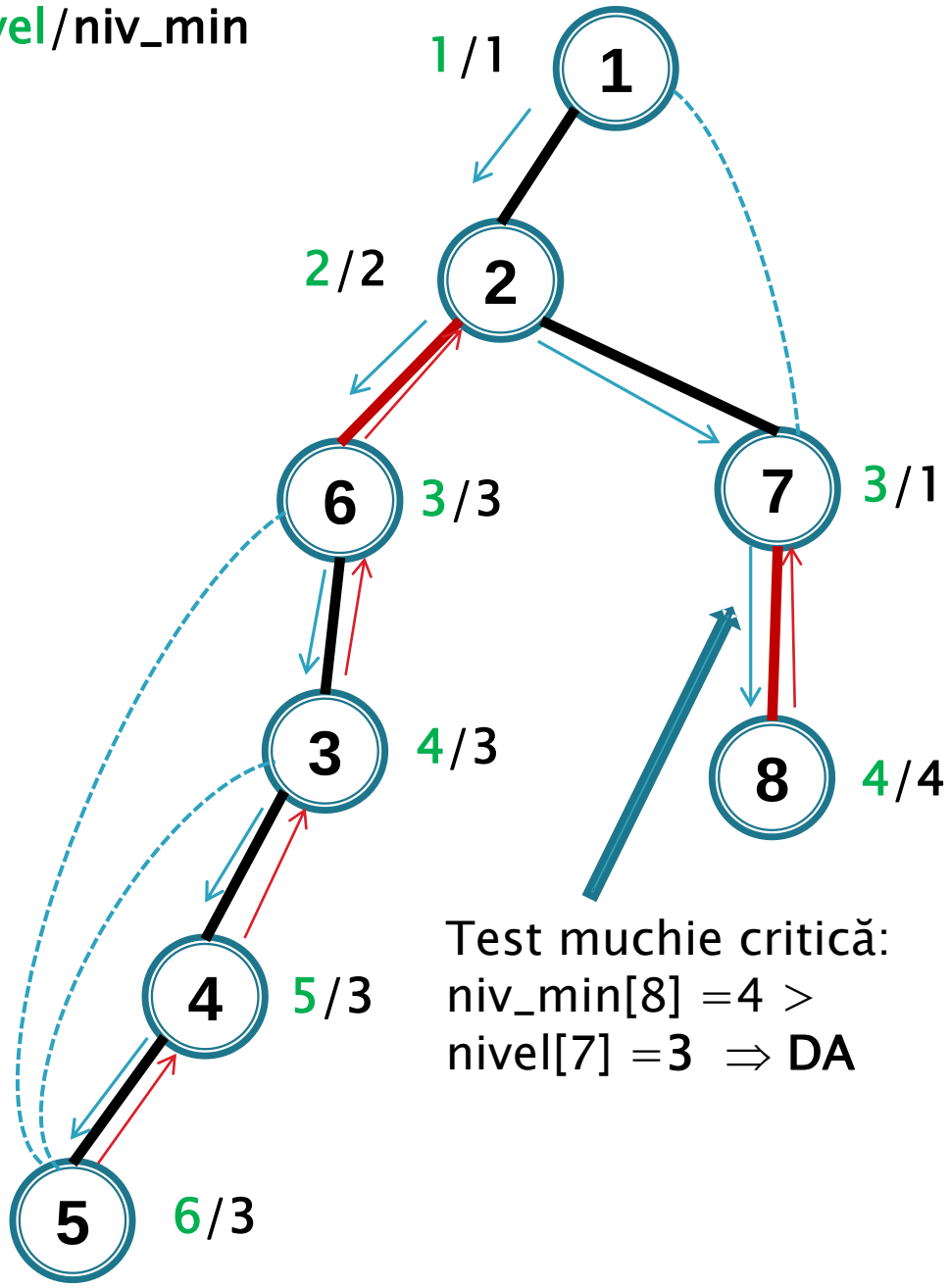

1
7
8
2
3
6
4
5

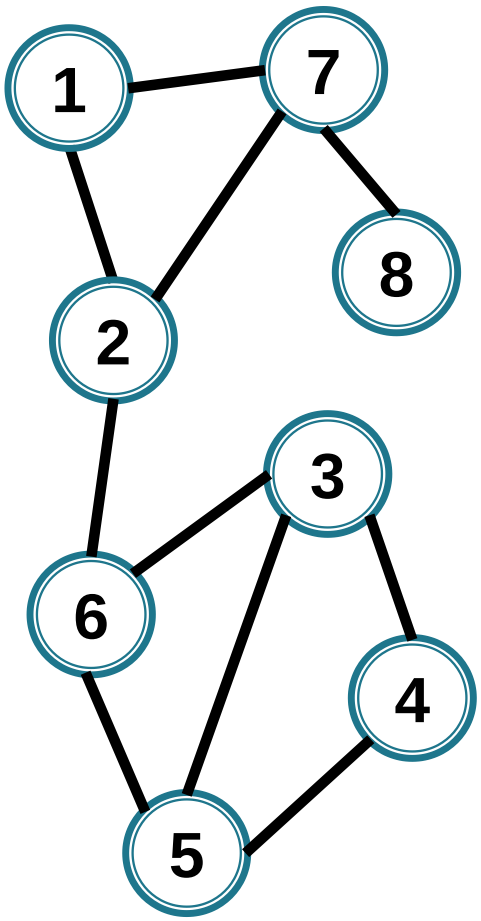

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/1
3
4/3
4
5/3
5
6/3

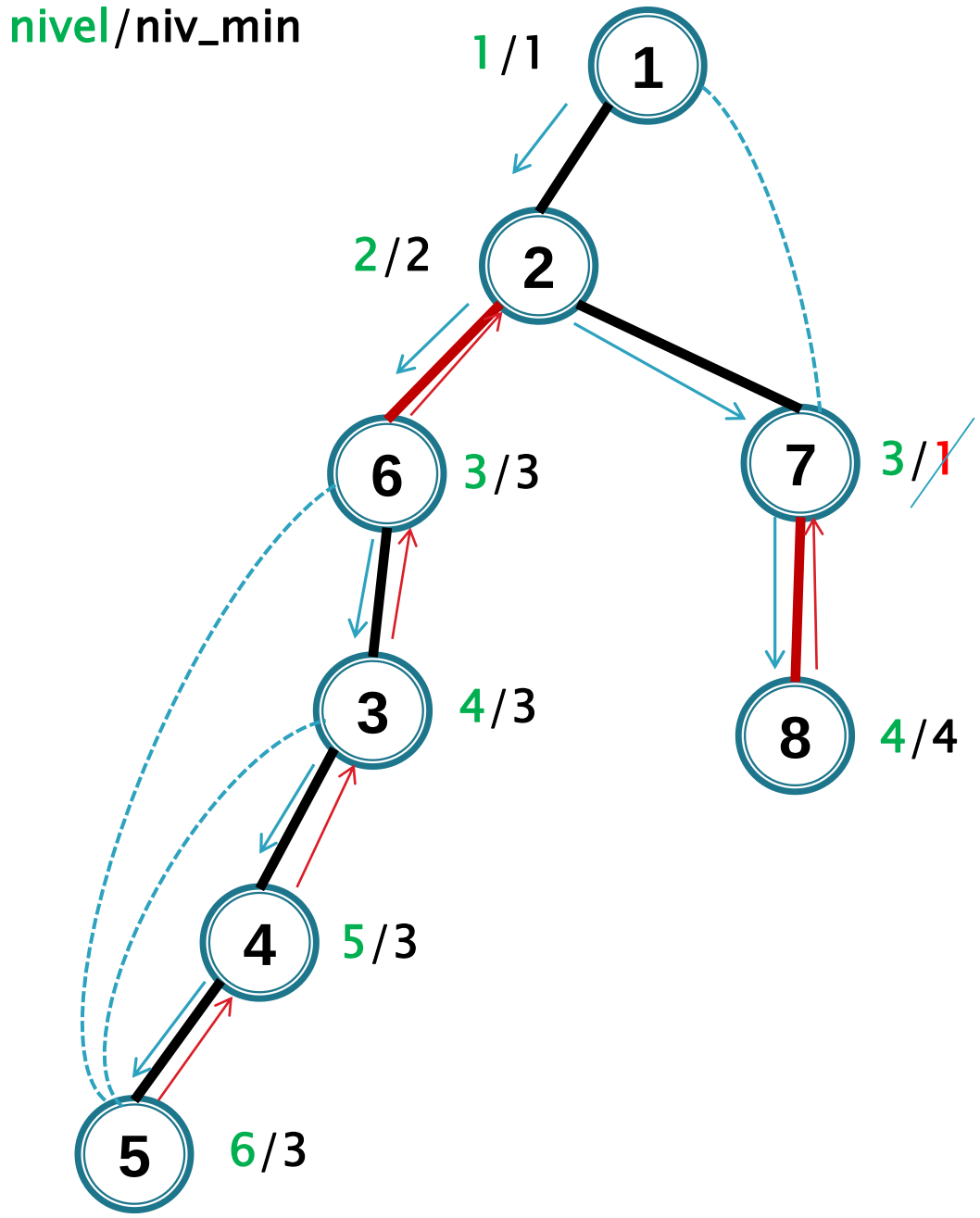

1
7
8
2
3
6
4
5

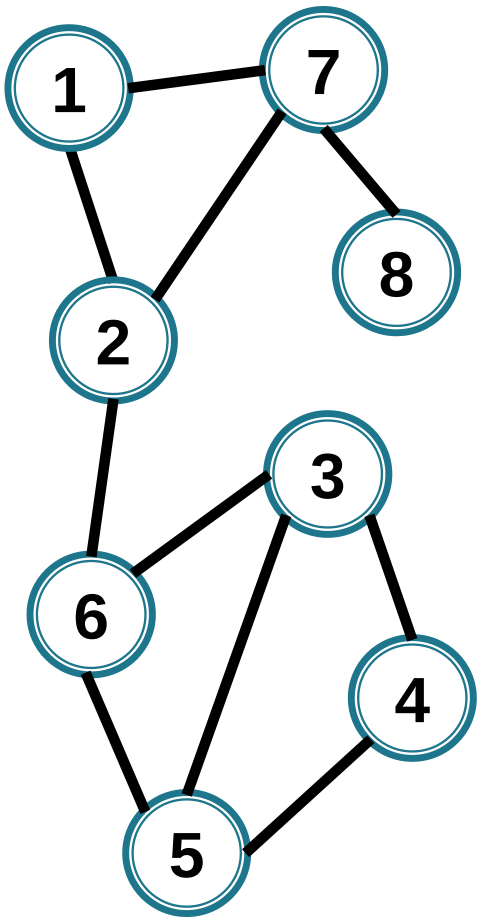

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3

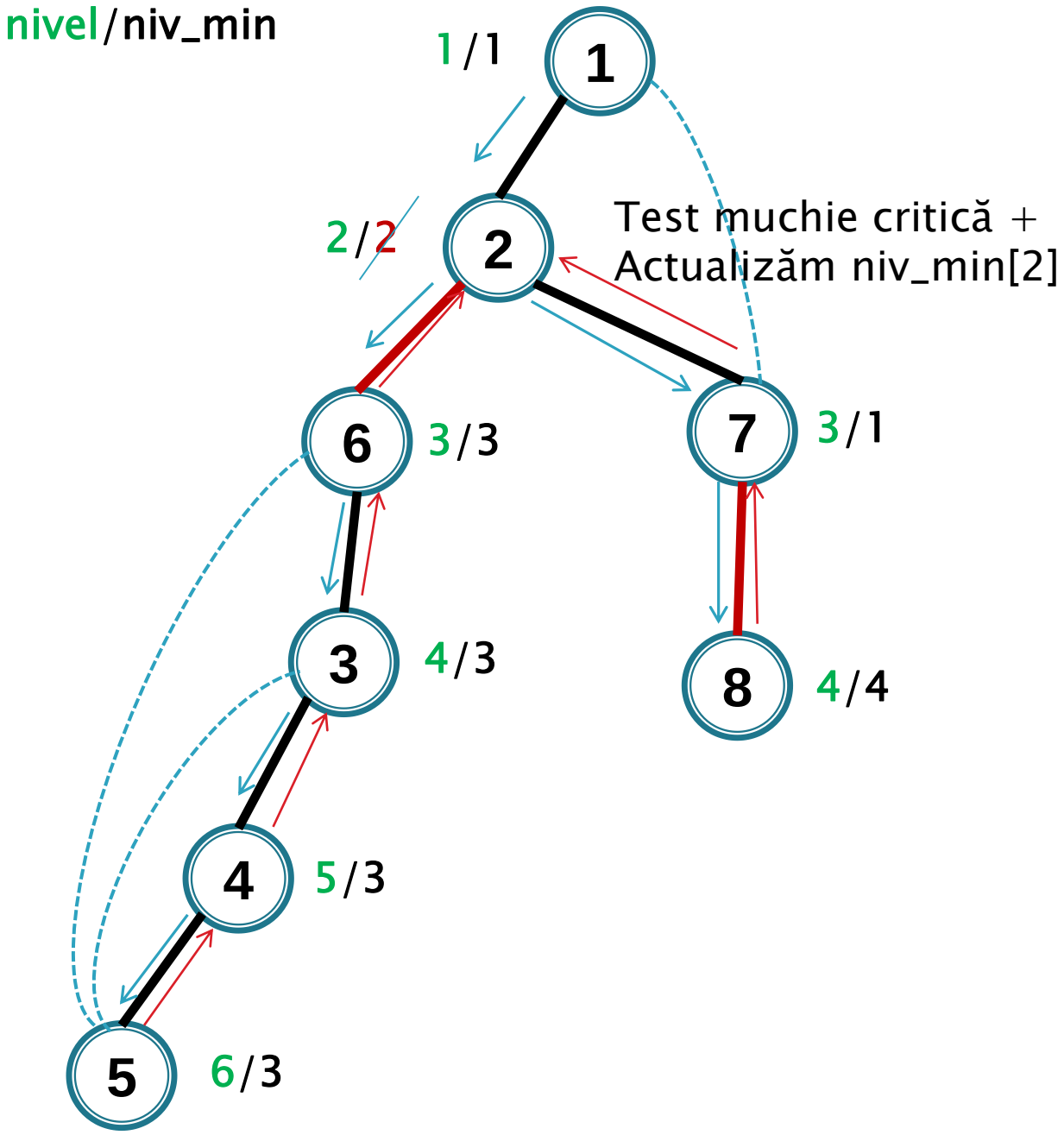

1
7
8
2
3
6
4
5

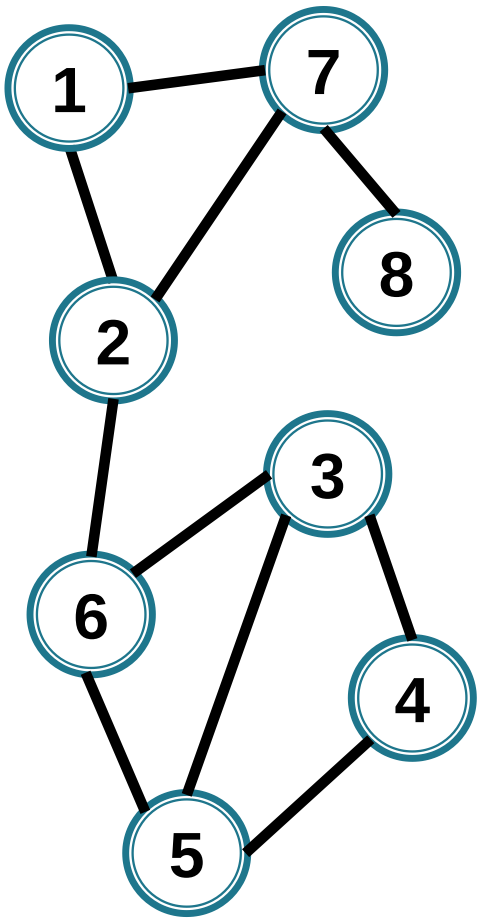

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3
Test muchie critică:
niv_min[8] =4 >
nivel[7] =3 ⇒ DA

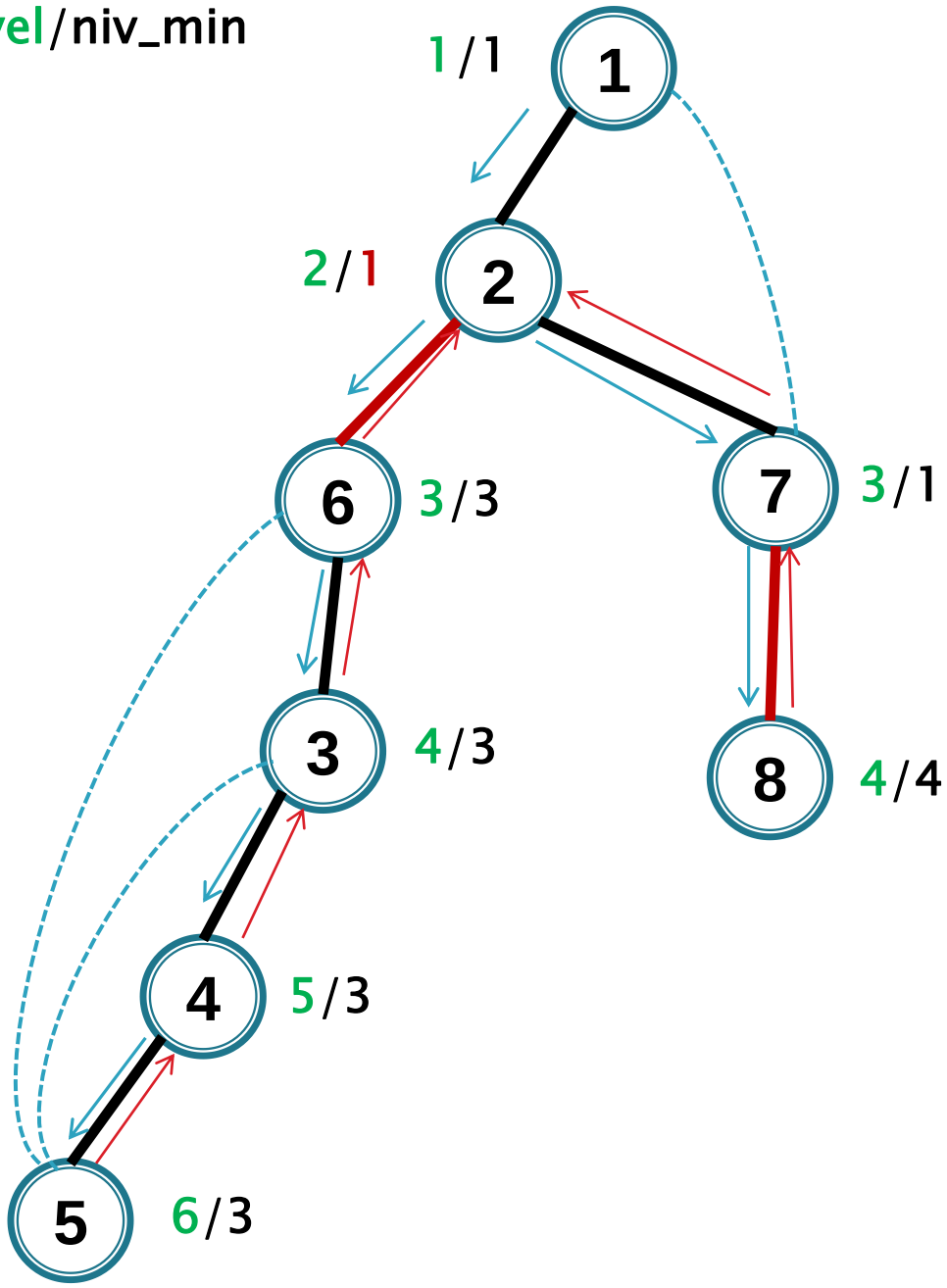

1
7
8
2
3
6
4
5

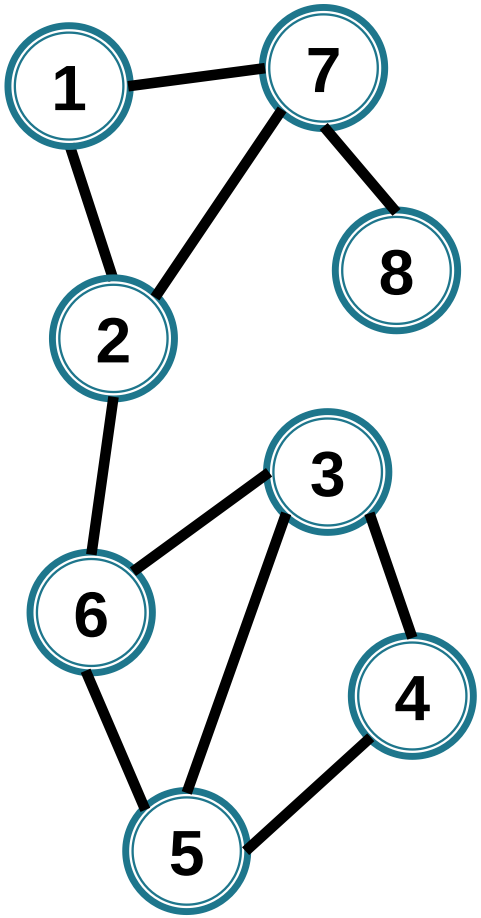

nivel/niv_min
1/1
1
2/2
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3


1
7
8
2
3
6
4
5


nivel/niv_min
1/1
1
2/2
2
Test muchie critică +
Actualizăm niv_min[2]
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3

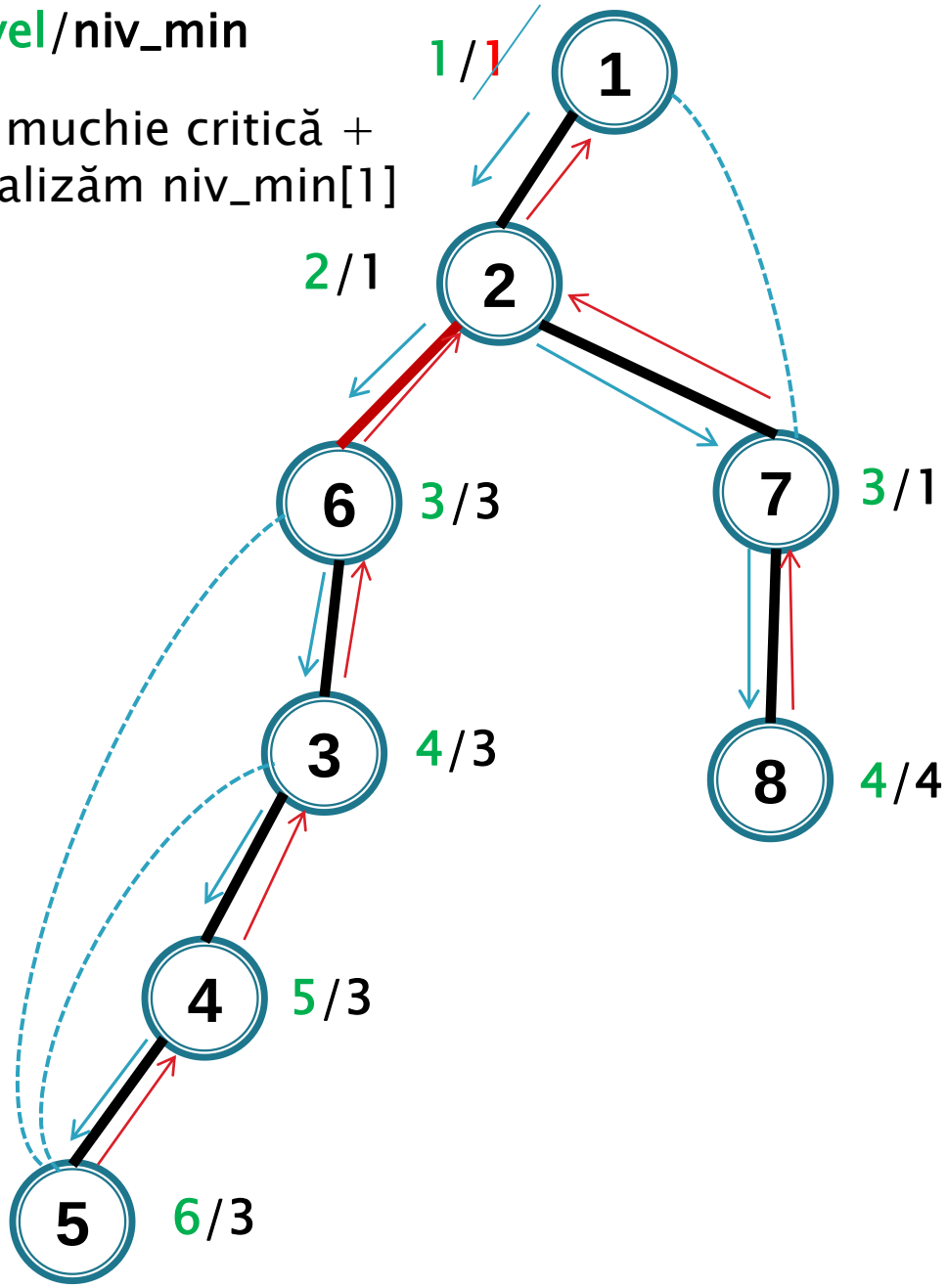

1
7
8
2
3
6
4
5

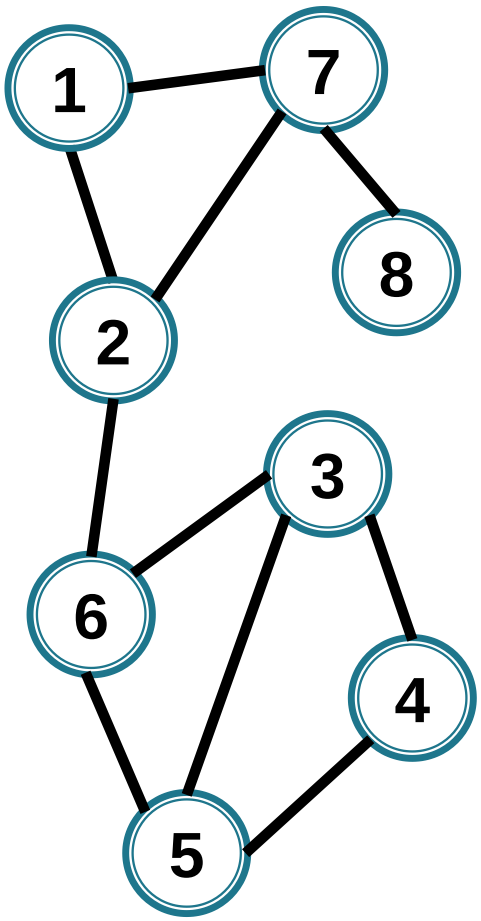

nivel/niv_min
1/1
1
2/1
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3

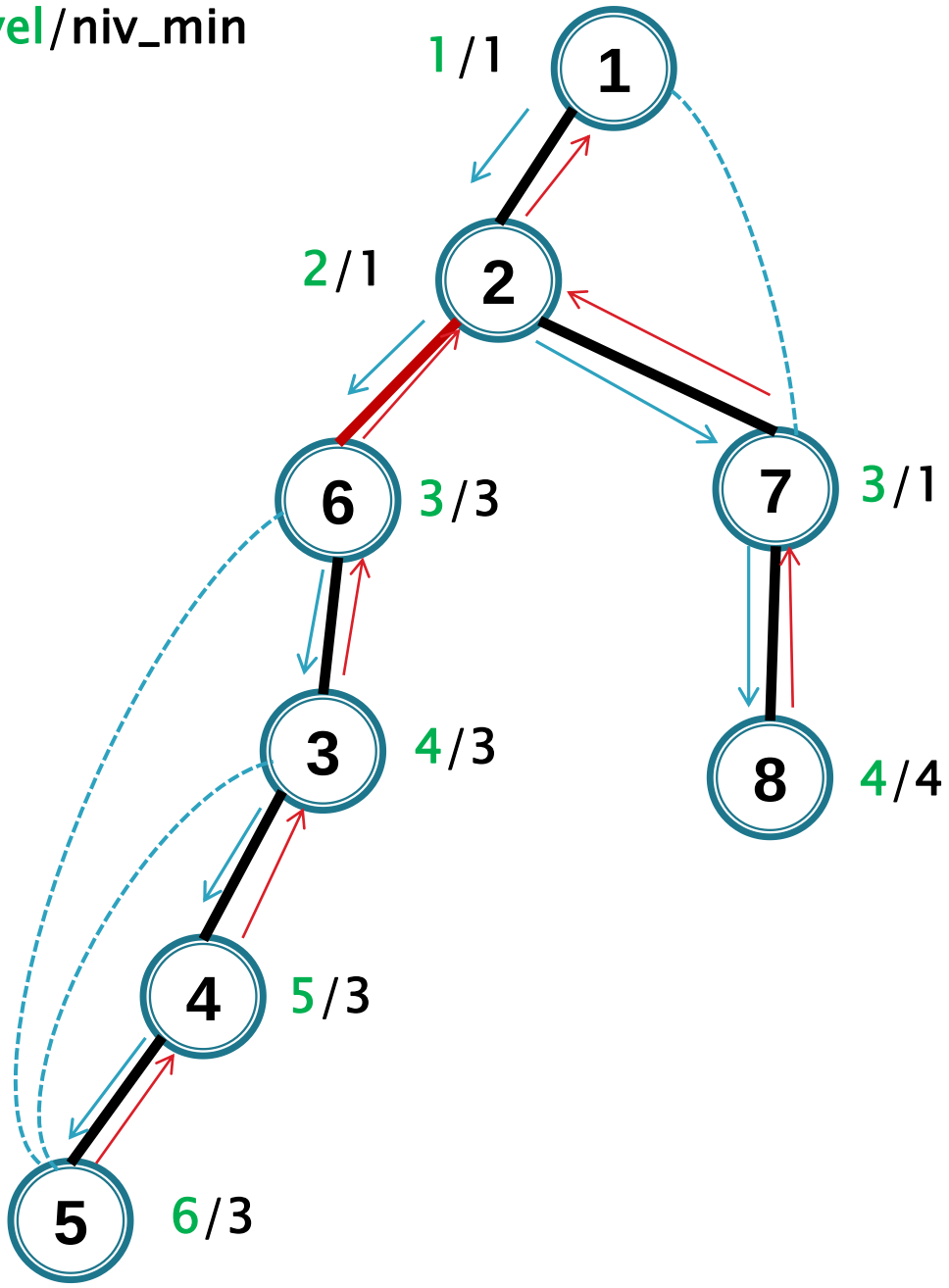


nivel/niv_min

Test muchie critică +
Actualizăm niv_min[1]




1
7
8
2
3
6
4
5


nivel/niv_min
1/1
1
2/1
2
6
3/3
7
3/1
3
4/3
8
4/4
4
5/3
5
6/3

Indicații implementare

```
void df(int i){
      viz[i] = 1;
      niv_min[i] = nivel[i];
      for(j vecin al lui i)
            if(viz[j]==0){ //ij muchie de avansare
                  nivel[j] = nivel[i]+1;
                  df(j);
                  //actualizare niv_min[i]- formula B
                  ...
                  //test ij este muchie critica
                  ...
            }
            else
               if(nivel[j]<nivel[i]-1) //ij muchie de intoarcere
                    //actualizare niv_min[i]- formula A
                     ...
 }
```

Indicații implementare

```
void df(int i){
      viz[i] = 1;
      niv_min[i] = nivel[i];
      for(j vecin al lui i)
            if(viz[j]==0){ //ij muchie de avansare
                  nivel[j] = nivel[i]+1;
                  df(j);

                  //actualizare niv_min[i]- formula B
                  niv_min[i] = min{niv_min[i], niv_min[j] }

                  //test ij este muchie critica
                  ...
            }
            else
               if(nivel[j]<nivel[i]-1) //ij muchie de intoarcere
                    //actualizare niv_min[i]- formula A
                     ...
 }
```

Indicații implementare

```
void df(int i){
      viz[i] = 1;
      niv_min[i] = nivel[i];
      for(j vecin al lui i)
            if(viz[j]==0){ //ij muchie de avansare
                  nivel[j] = nivel[i]+1;
                  df(j);

                  //actualizare niv_min[i]- formula B
                  niv_min[i] = min{niv_min[i], niv_min[j] }

                  //test ij este muchie critica
                  if (niv_min[j]>nivel[i]) scrie muchia ij
            }
            else
               if(nivel[j]<nivel[i]-1) //ij muchie de intoarcere
                    //actualizare niv_min[i]- formula A
                     ...
 }
```

Indicații implementare

```
void df(int i){
      viz[i] = 1;
      niv_min[i] = nivel[i];
      for(j vecin al lui i)
            if(viz[j]==0){ //ij muchie de avansare
                  nivel[j] = nivel[i]+1;
                  df(j);

                  //actualizare niv_min[i]- formula B
                  niv_min[i] = min{niv_min[i], niv_min[j] }

                  //test ij este muchie critica
                  if (niv_min[j]>nivel[i]) scrie muchia ij
            }
            else
               if(nivel[j]<nivel[i]-1) //ij muchie de intoarcere
                    //actualizare niv_min[i]- formula A
                     niv_min[i] = min{niv_min[i], nivel[j] }
 }
```

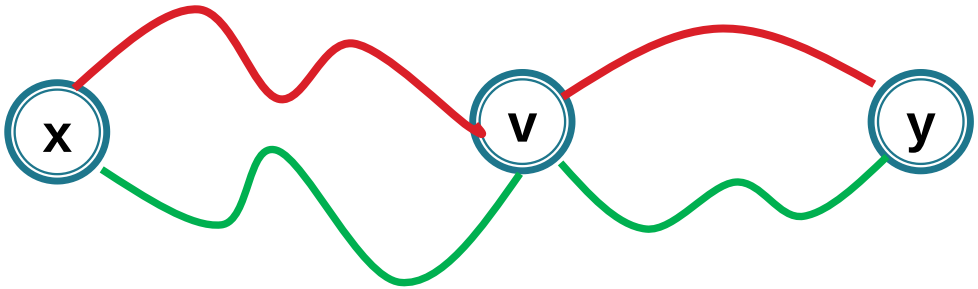
Puncte critice

Puncte critice

- Un vârf v este punct critic $\Leftrightarrow$

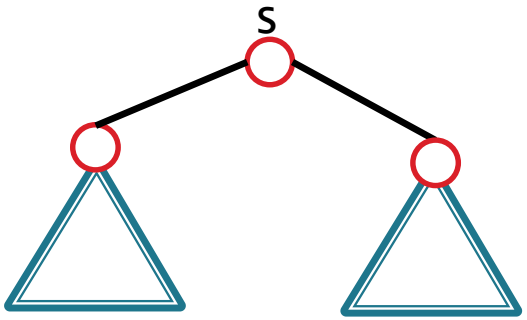
  există două vârfuri x,y $\neq$ v astfel

  încât v aparține oricărui x,y-lanț



Puncte critice

- Arborele DF
  - **rădăcina s este punct critic ⇔** ?

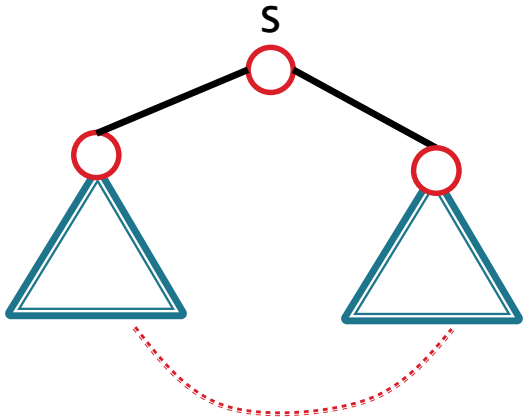


Puncte critice

- Arborele DF
  - **rădăcina s este punct critic ⇔** 

are cel puțin 2 fii în arborele DF

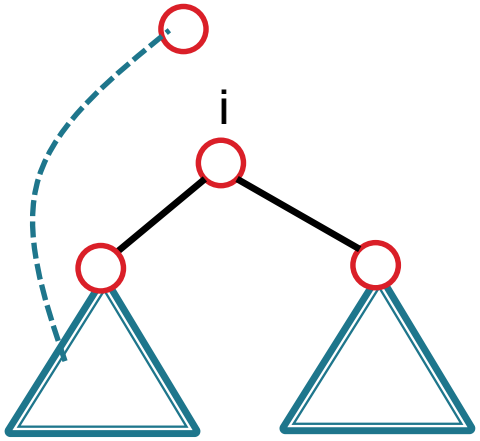
(s aparține oricărui lanț dintre 2 fii ai săi)



nu există muchii între subarbori
(de traversare)

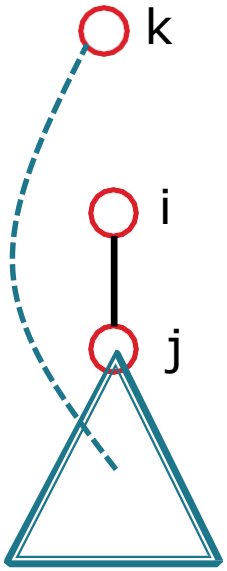
Puncte critice

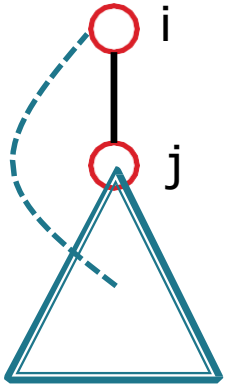
- Arborele DF
  - **un alt vârf i din arbore este critic ⇔ ?**

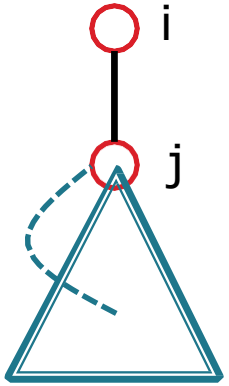


Puncte critice

Pentru $i \neq s$

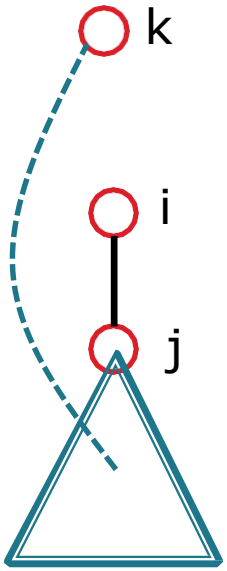






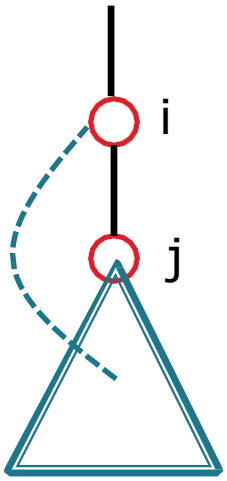
Puncte critice

Pentru i ≠ s



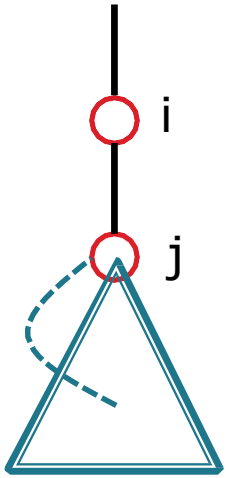
i NU este critic

`niv_min[j]<nivel[i]`



i ESTE critic

`niv_min[j]=nivel[i]`



i ESTE critic

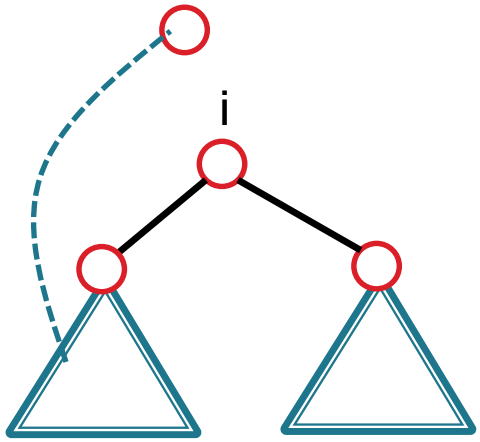
`niv_min[j]>nivel[i]`

i aparține oricărui lanț de la tata[i] la j

Puncte critice

- Arborele DF
  - **un alt vârf i din arbore este critic ⇔**

are cel puțin un fiu j cu
niv_min[j] ≥ nivel[i]

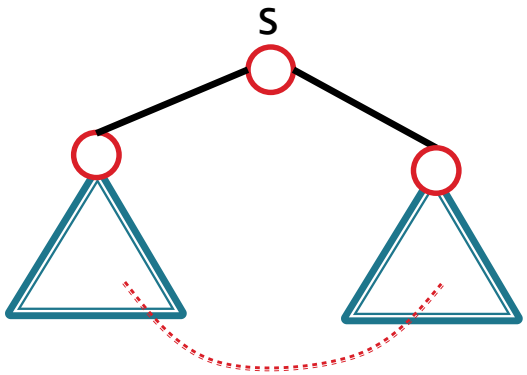


Puncte critice

Concluzii:

- Arbore DF
  - **rădăcina s este punct critic ⇔**

    are cel puțin 2 fii în arborele DF

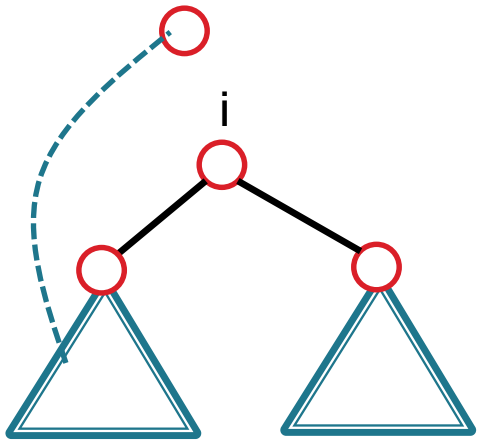


nu există muchii între subarbori
(de traversare)

  - **un alt vârf i din arbore este critic ⇔**

    are cel puțin un fiu j cu

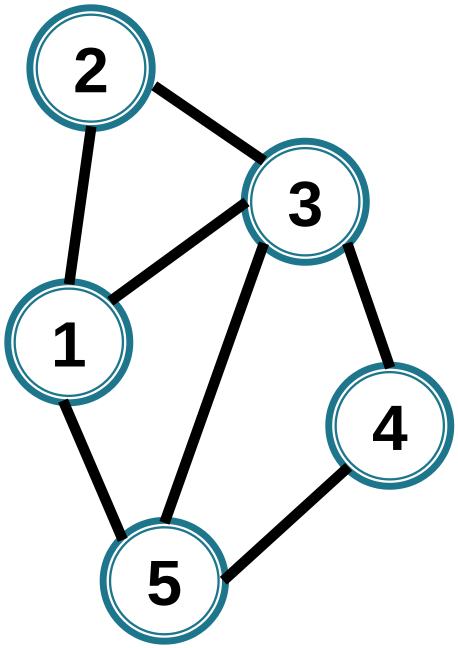
    niv_min[j] ≥ nivel[i]



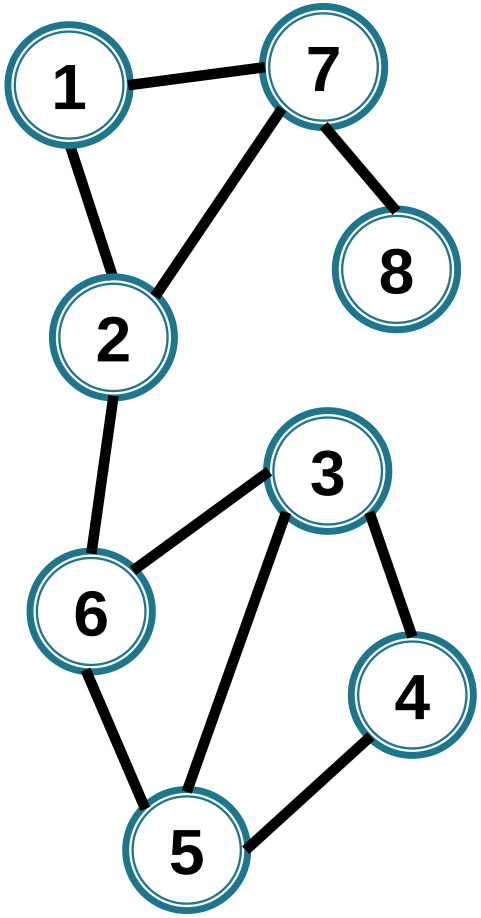
Componente biconexe

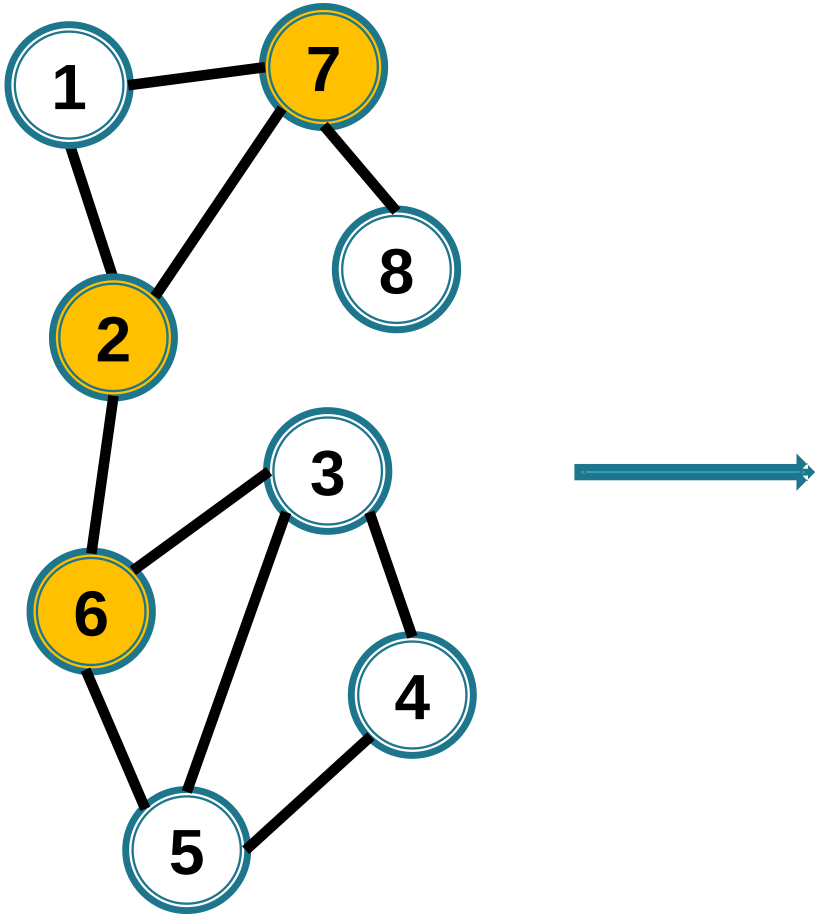
Componente biconexe

- G – graf neorientat
- G=(V,M) biconex = **nu are puncte critice (de articulaţie).**
- **Componentă biconexă** (**bloc**) a lui G = subgraf biconex maximal.

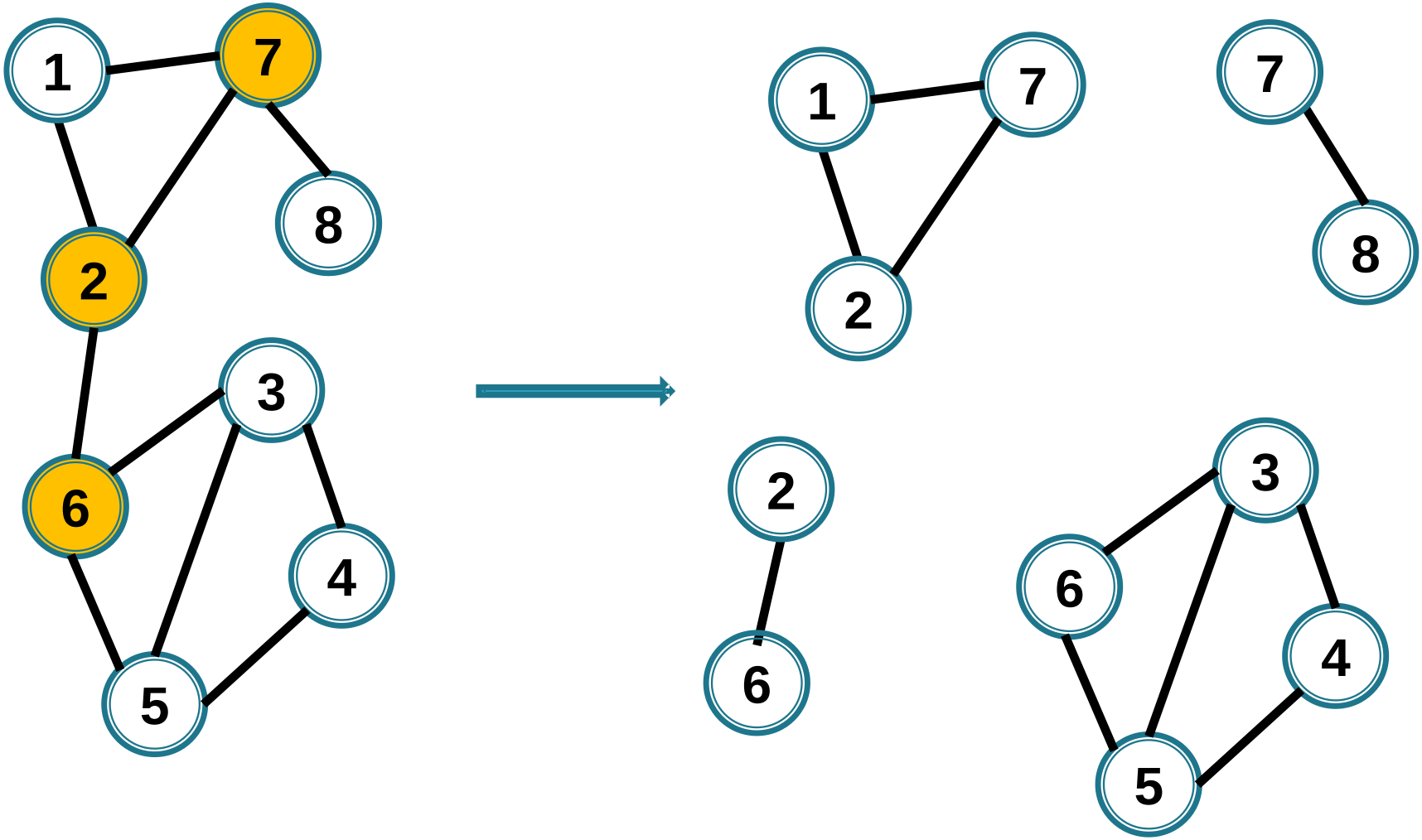


graf biconex


1
7
8
2
3
6
4
5



Puncte critice 2, 6, 7



Puncte critice 2, 6, 7

Componentele biconexe

Componente biconexe

- **Observații**

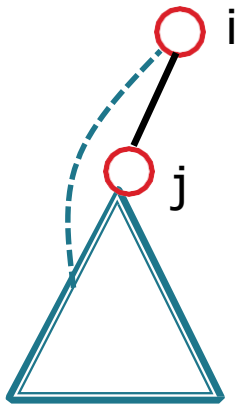
  Într-un graf neorientat:

  - **Componentele biconexe sunt muchie-disjuncte, dar nu neaparat vârf-disjuncte (pot avea în comun un vârf care era punct critic în graful inițial)**
  - Orice vârf aparține unei componente biconexe
  - Orice muchie aparține unei componente biconexe

Componente biconexe

- **Algoritm**

Putem folosi tot algoritmul de determinare a punctelor critice

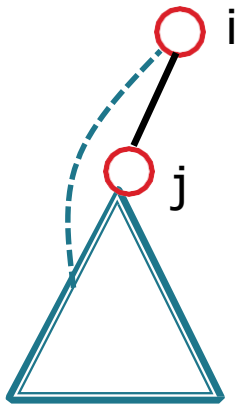


Componente biconexe

- **Algoritm**

Putem folosi tot algoritmul de determinare a punctelor critice

Intuitiv: când un fiu j semnalează că vârful curent i este critic, adică niv_min[j] ≥ niv[i] se poate afișa componenta care conține muchia ij (care “se rupe” din vârful i și conține un subarbore de rădăcină j)

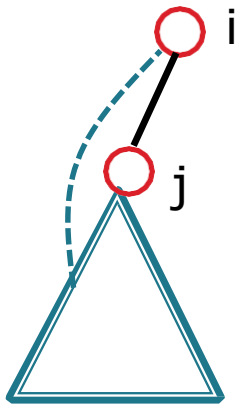


Componente biconexe

- **Algoritm**

Putem folosi tot algoritmul de determinare a punctelor critice

Intuitiv: când un fiu j semnalează că vârful curent i este critic, adică niv_min[j] ≥ niv[i] se poate afișa componenta care conține muchia ij

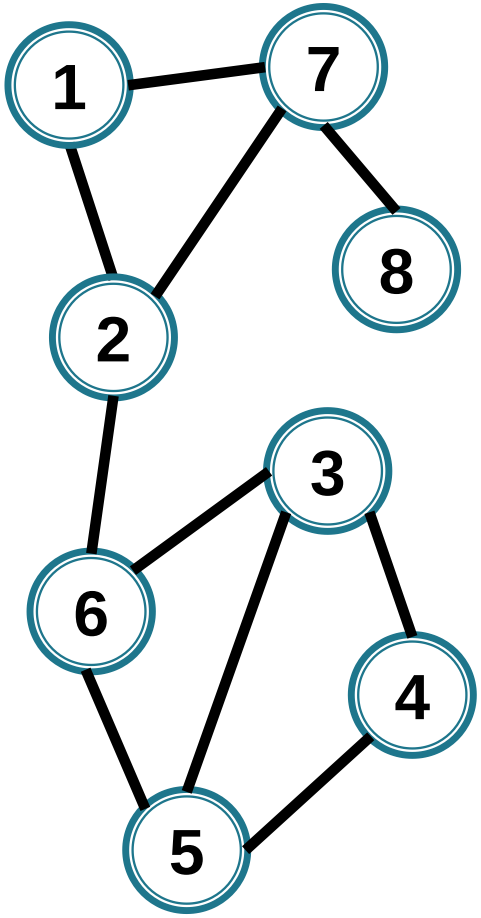


Memorăm muchiile într-o stivă

Când detectăm o componentă biconexă – scoatem muchiile din stivă până la muchia ij => acestea formează o componentă

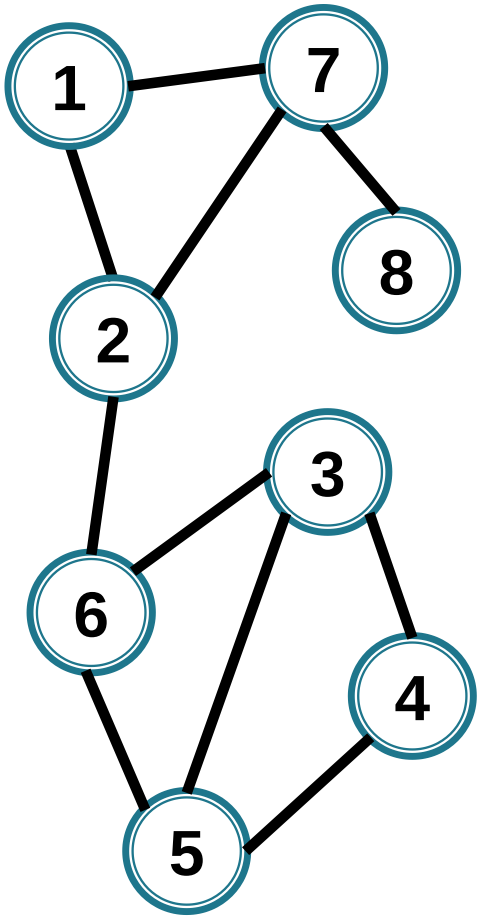
Indicații implementare

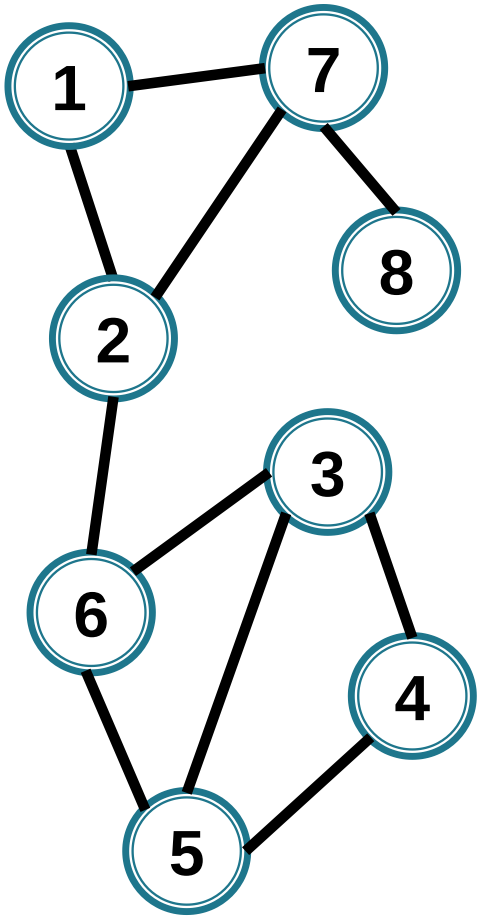
```
void df(int i){
     viz[i] = 1;
     niv_min[i] = nivel[i];
     for(j vecin al lui i)
          if(viz[j]==0){ //ij muchie de avansare
               nivel[j] = nivel[i]+1;
               adauga(S,ij)
               df(j);
               niv_min[i] = min{niv_min[i], niv_min[j] }
               if (niv_min[j] > = nivel[i])
                        elimina din S toate muchiile pana la ij
           }
           else
              if(nivel[j]<nivel[i]-1){ //ij muchie de intoarcere
                  //actualizare niv_min[i]- formula A
                  niv_min[i] = min{niv_min[i], nivel[j]
                  adauga(S,ij)
               }
 }
```


1
7
8
2
3
6
4
5

nivel/niv_min

1/1

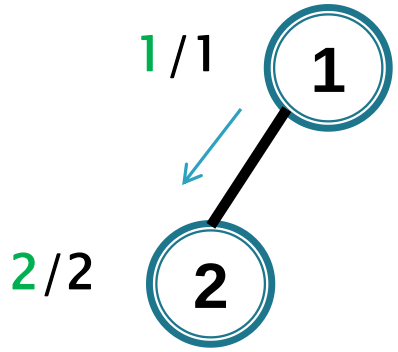


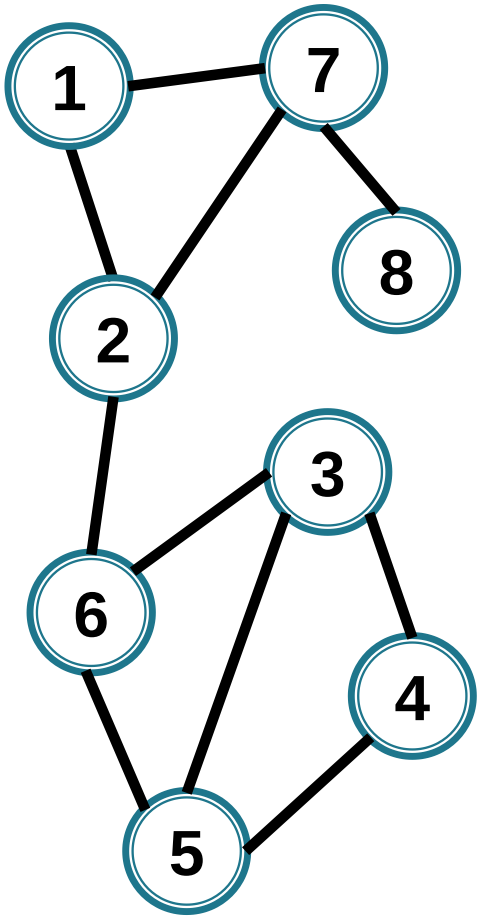


S:

1 2

nivel/niv_min



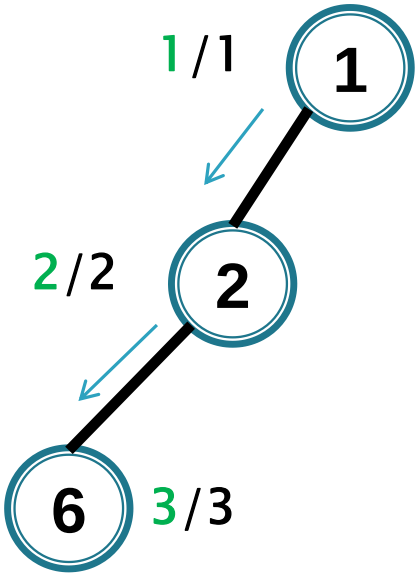


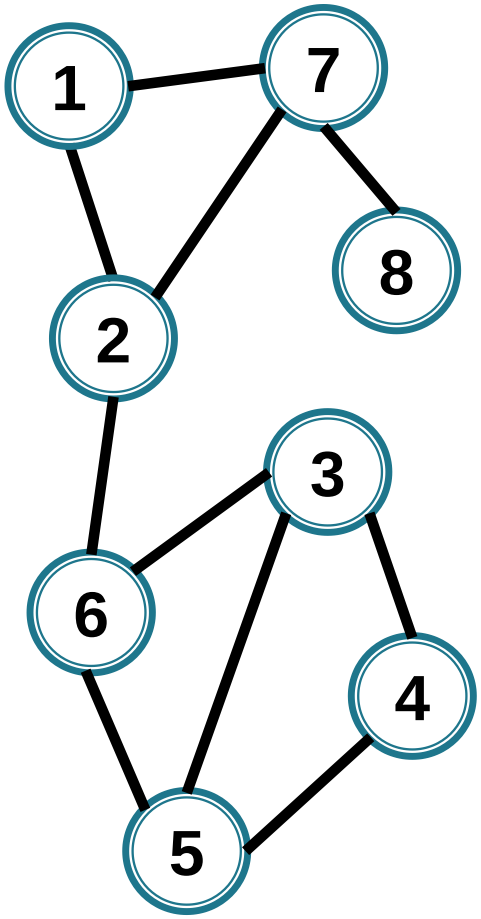
S:

1 2

2 6

nivel/niv_min





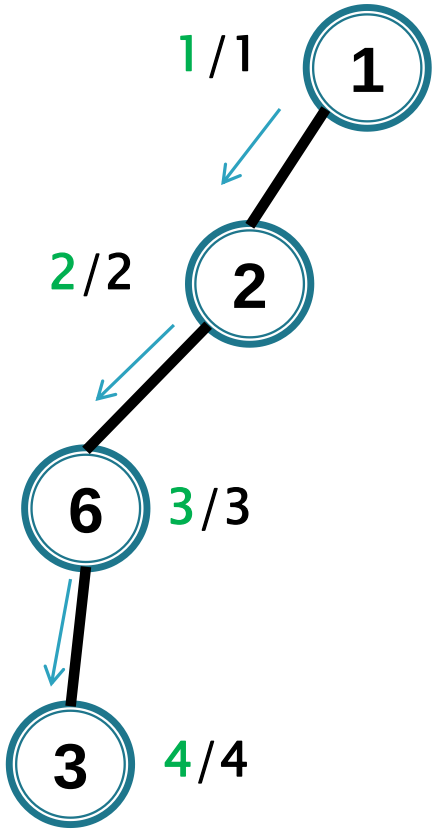
S:

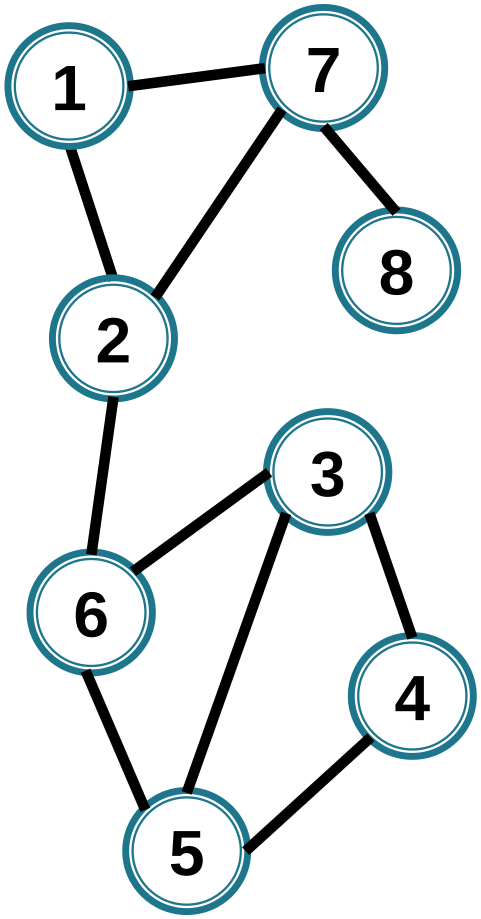
1 2

2 6

6 3

nivel/niv_min





S:

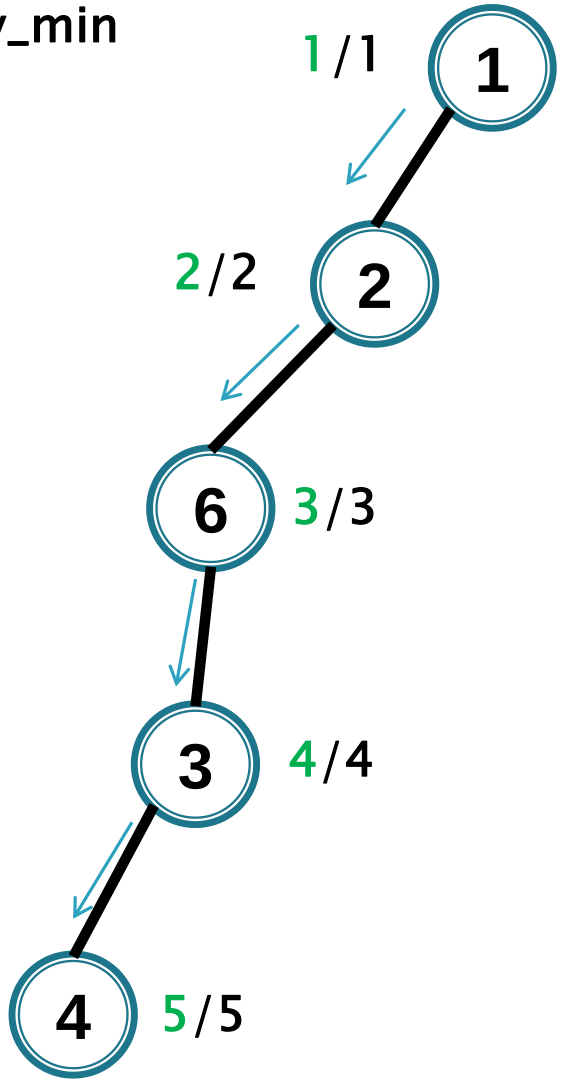
1 2

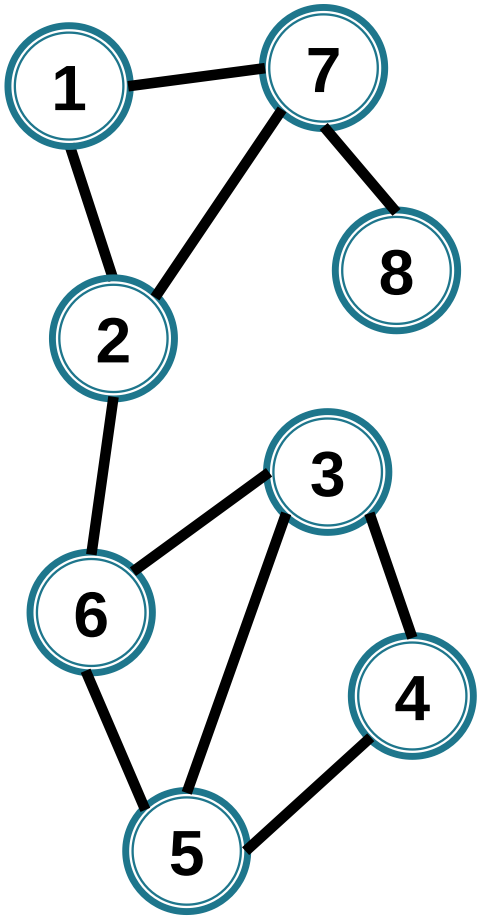
2 6

6 3

3 4

nivel/niv_min





S:

1 2

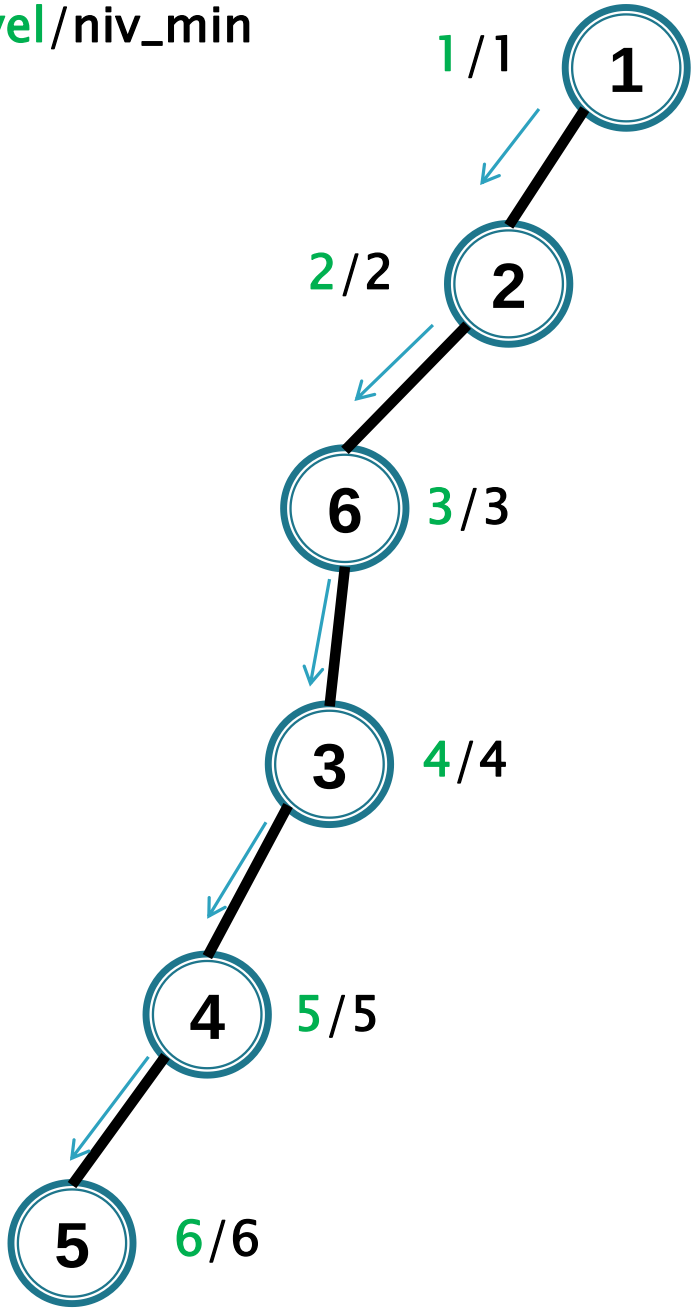
2 6

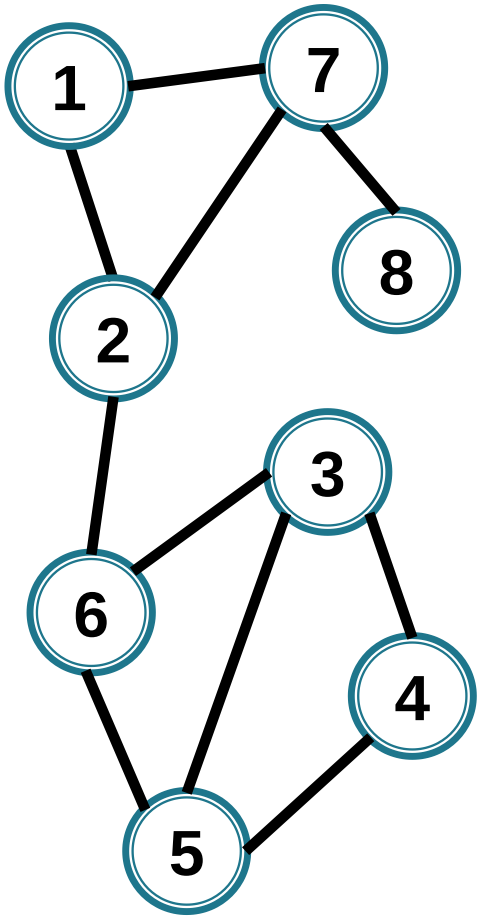
6 3

3 4

4 5

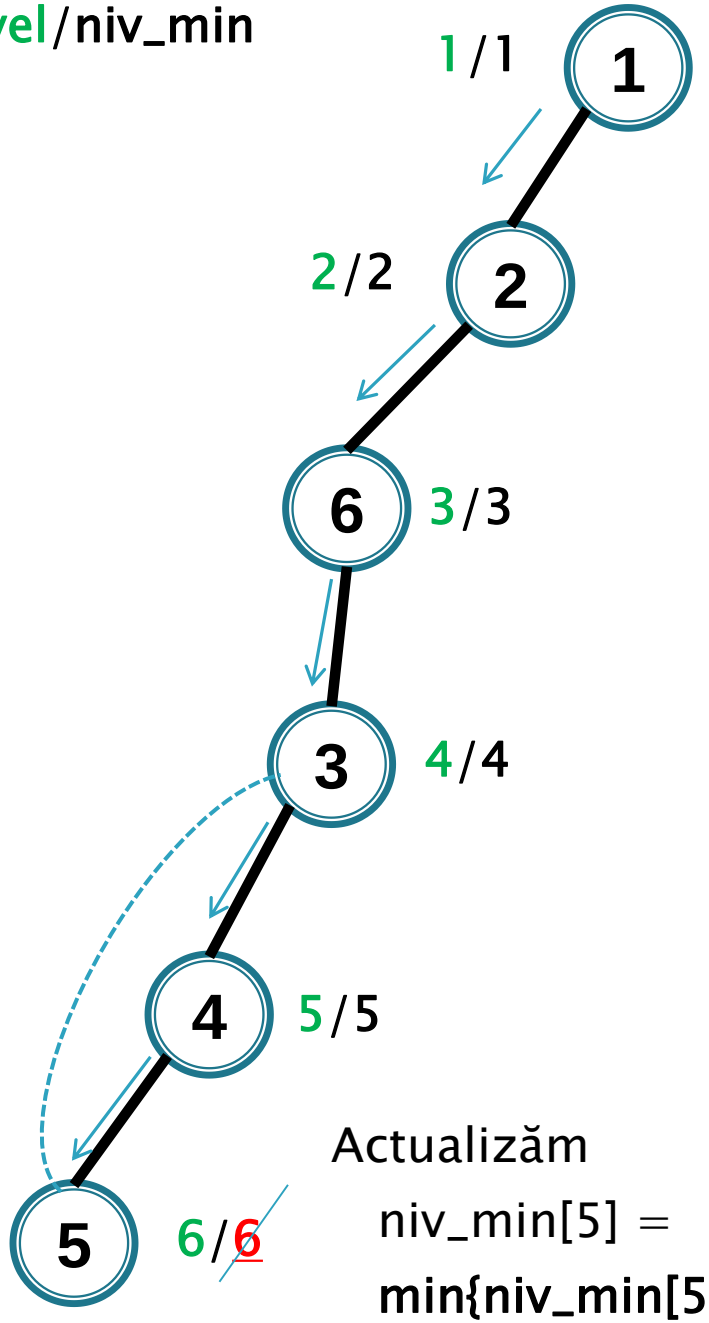
nivel/niv_min



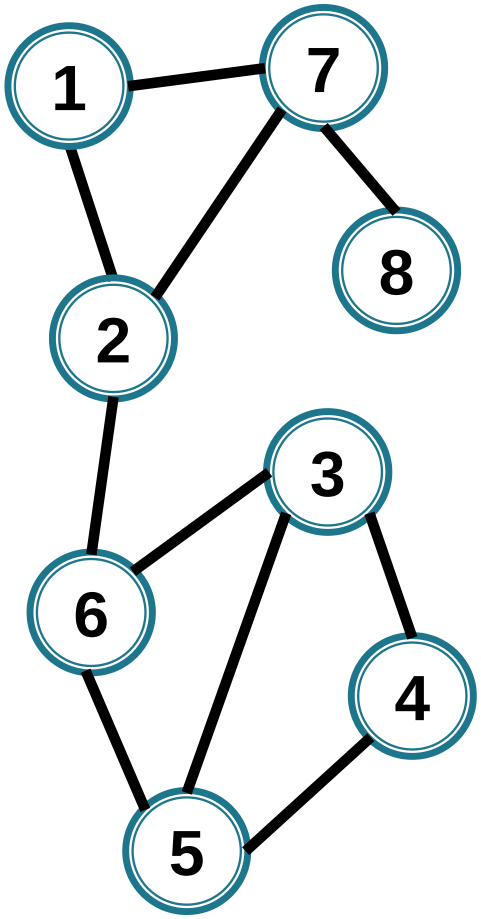


S:
1 2
2 6
6 3
3 4
4 5
5 3

nivel/niv_min



Actualizăm
niv_min[5] =
min{niv_min[5], nivel[3] }
(cazul A)



S:

1 2

2 6

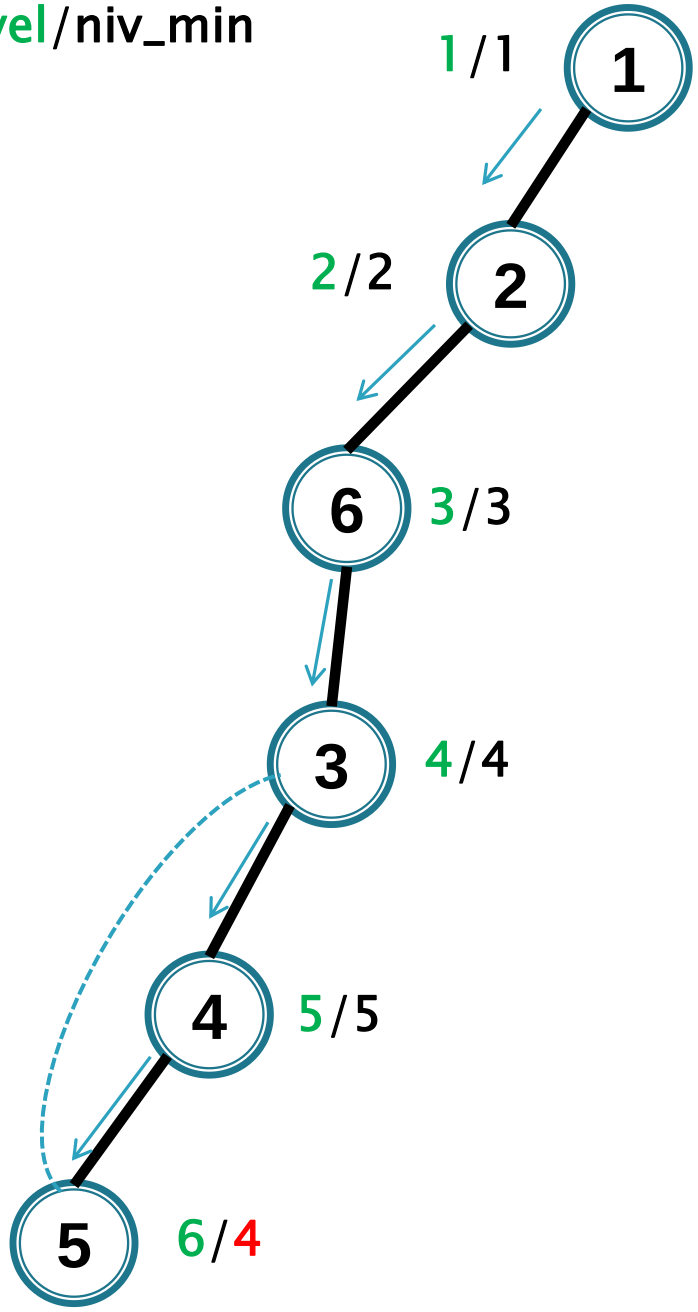
6 3

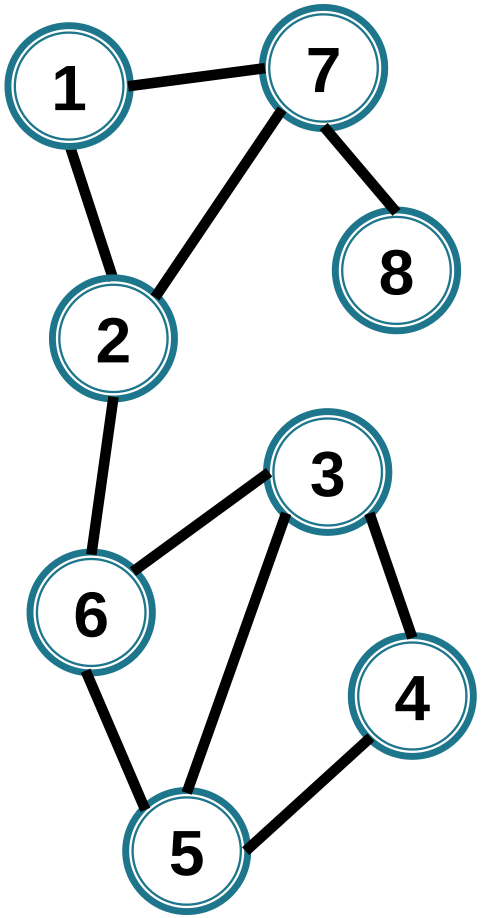
3 4

4 5

5 3

nivel/niv_min





S:

1 2

2 6

6 3

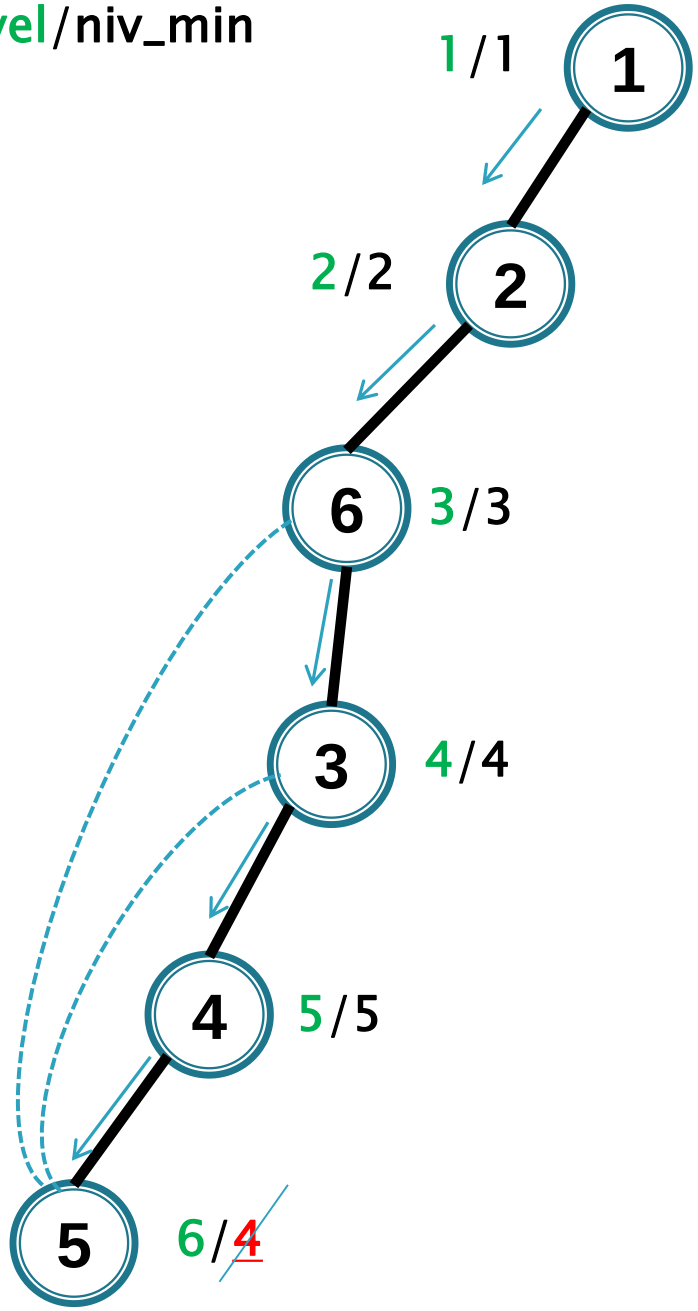
3 4

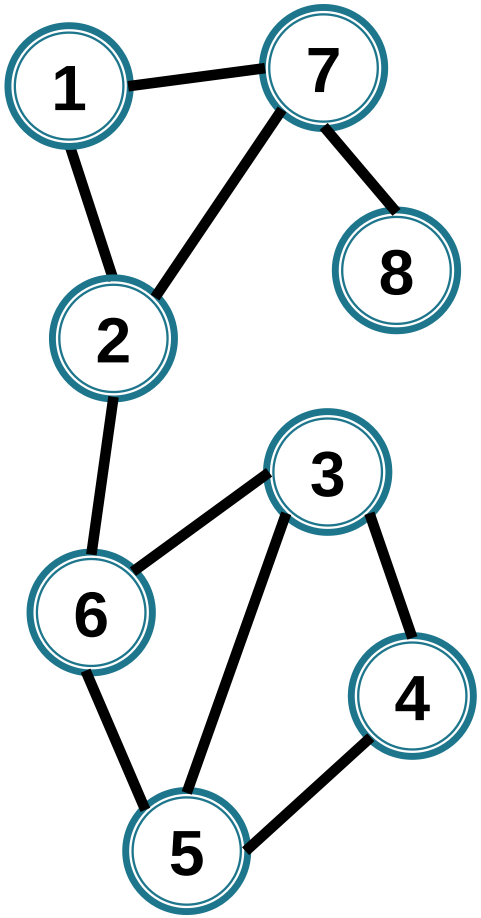
4 5

5 3

5 6

nivel/niv_min





S:

1 2

2 6

6 3

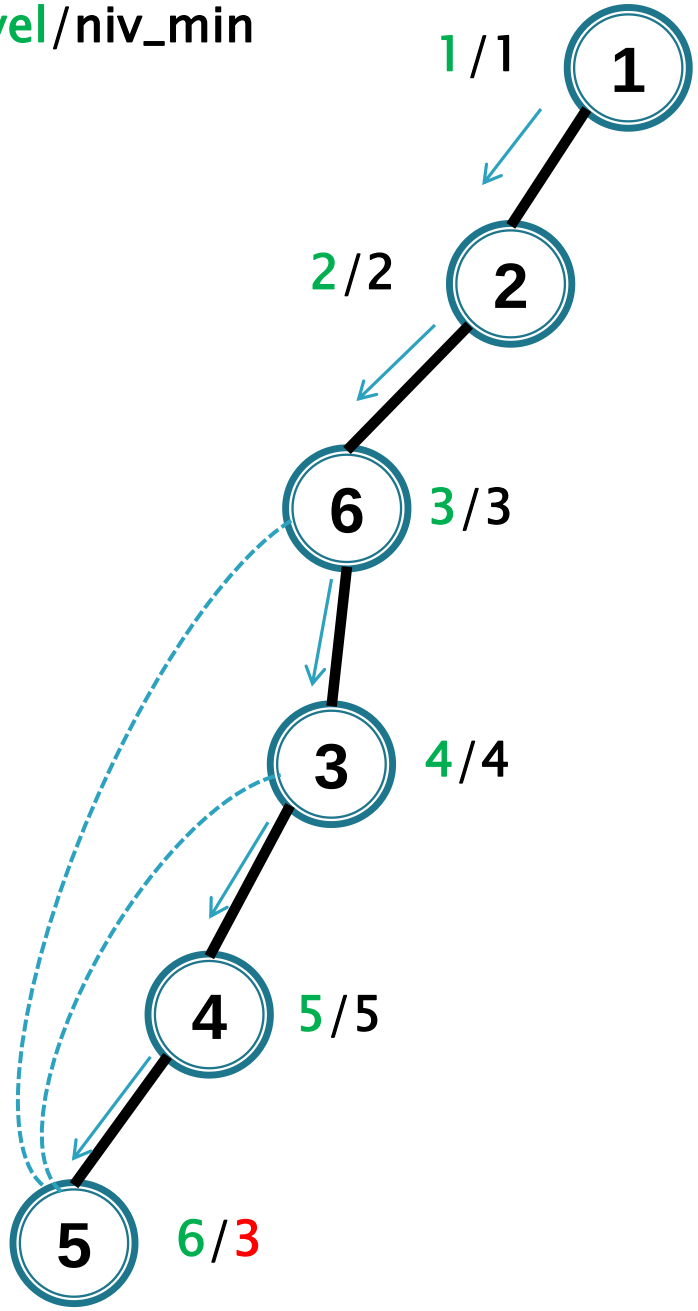
3 4

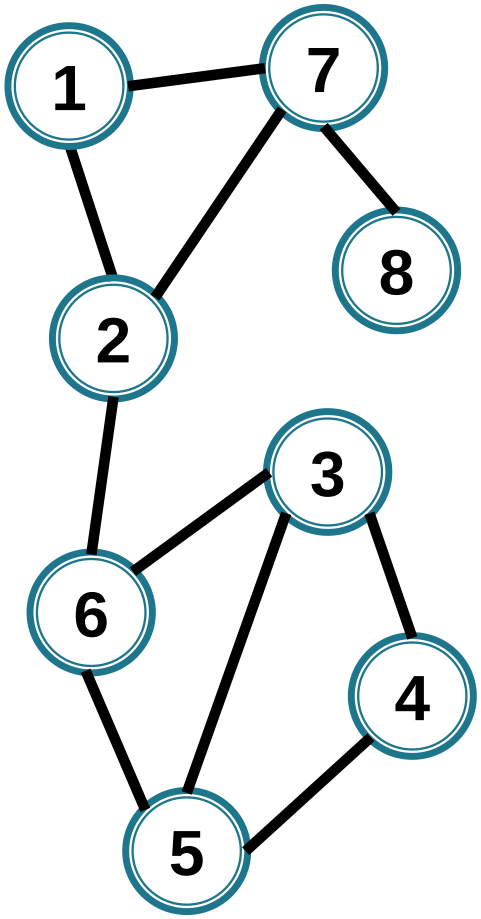
4 5

5 3

5 6

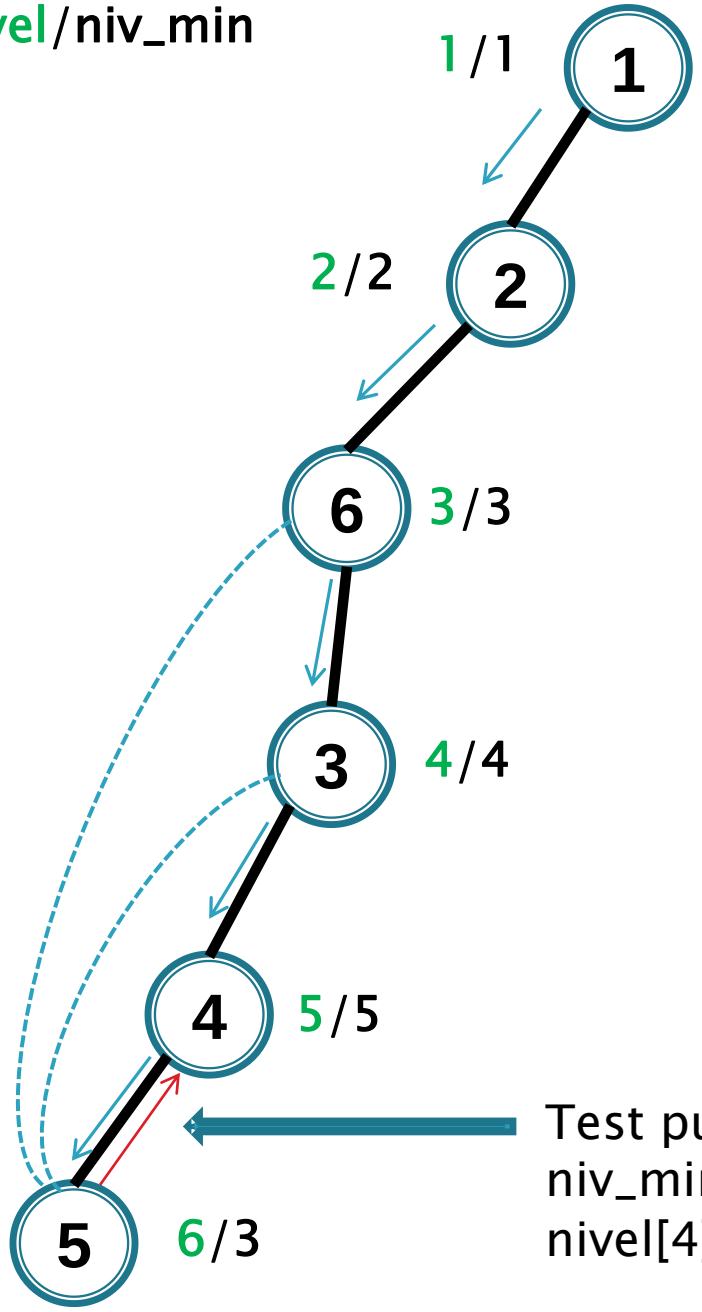
nivel/niv_min



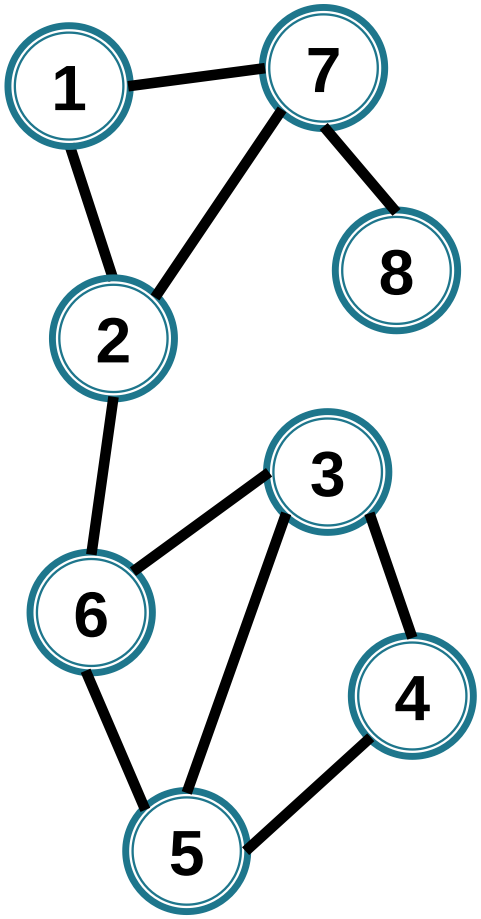


S:
1 2
2 6
6 3
3 4
4 5
5 3
5 6

nivel/niv_min

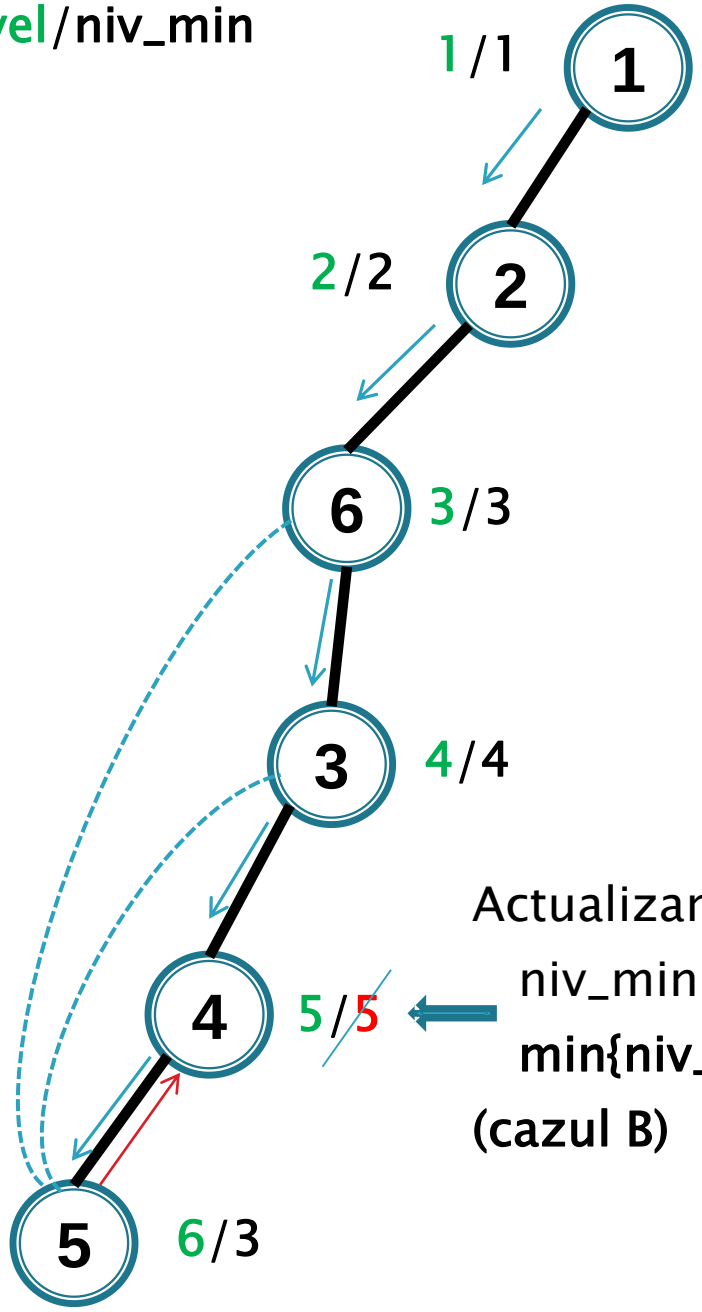


Test punct critic:
niv_min[5] =3 <
nivel[4] =5 ⇒ **NU**

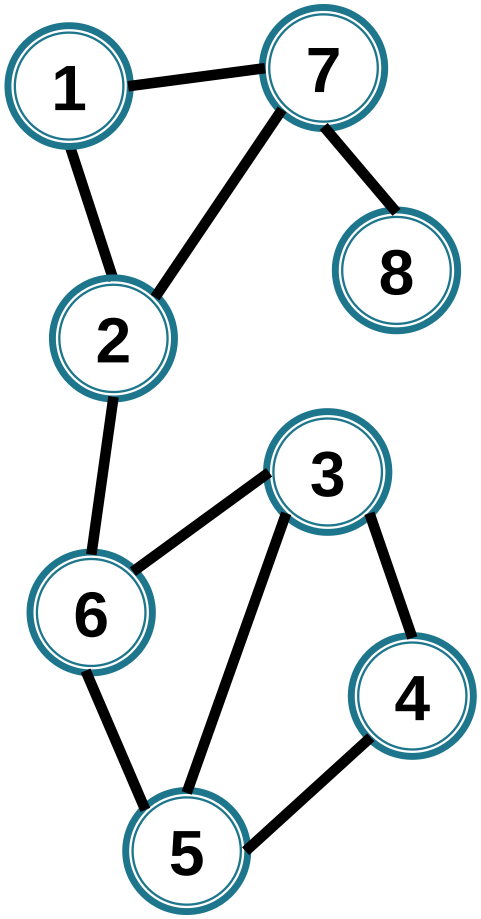


S:
1 2
2 6
6 3
3 4
4 5
5 3
5 6

nivel/niv_min



Actualizam
niv_min[4] =
min{niv_min[4], niv_min[5]}
(cazul B)



S:

1 2

2 6

6 3

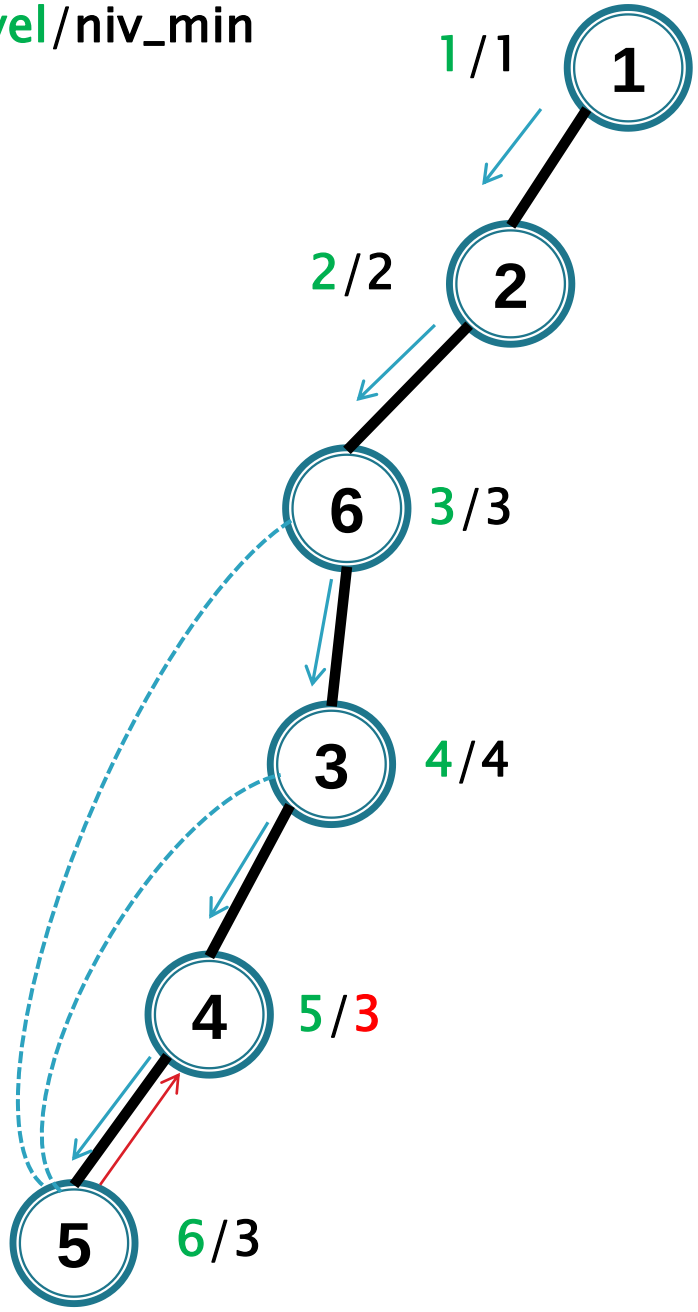
3 4

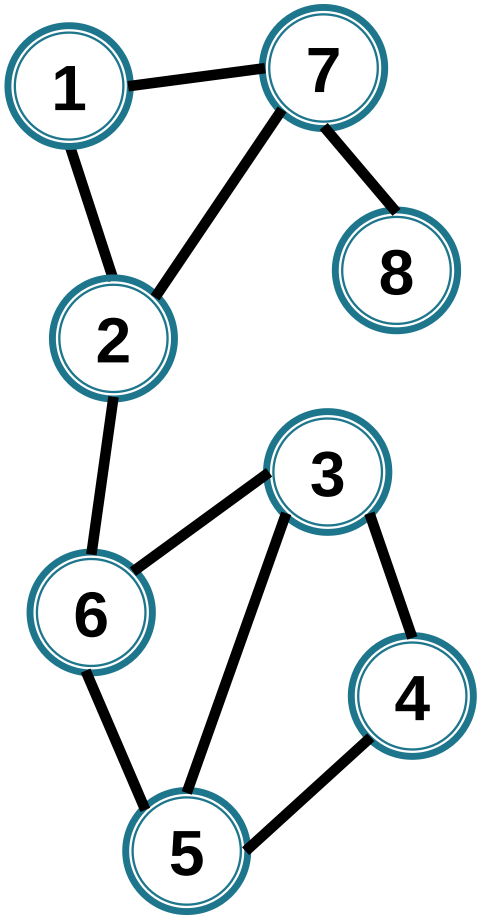
4 5

5 3

5 6

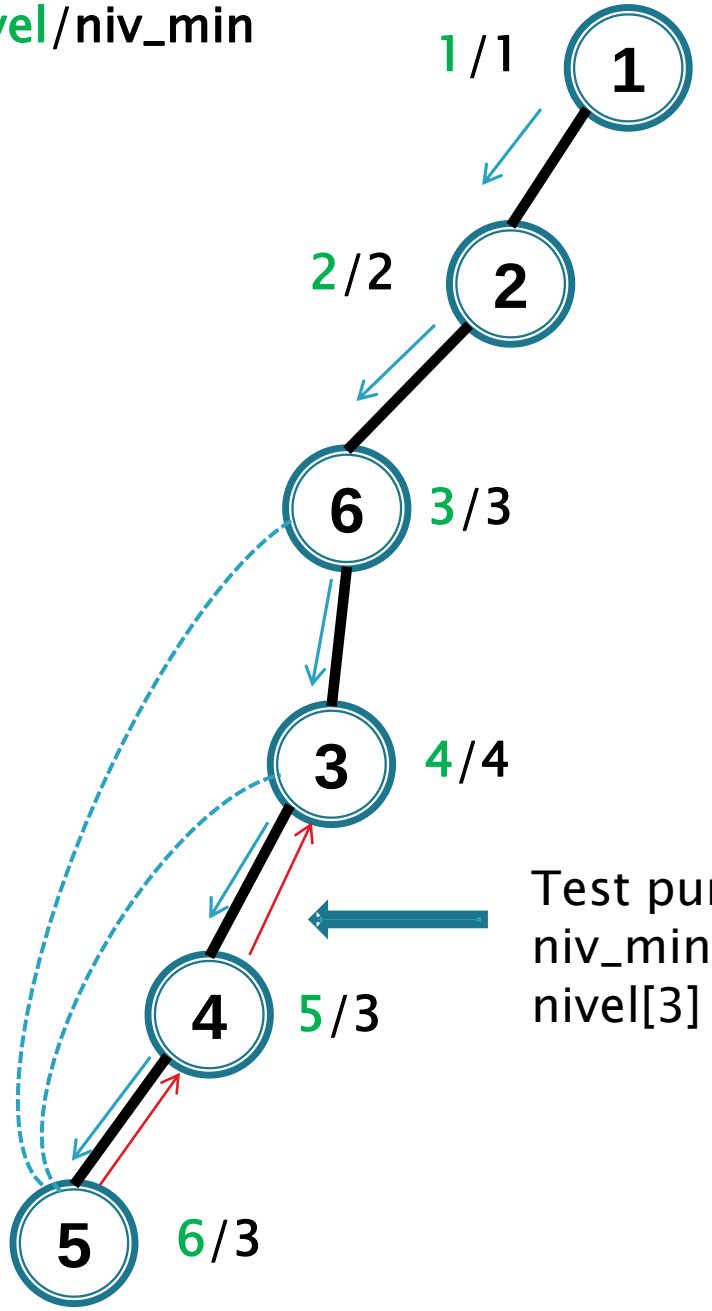
nivel/niv_min



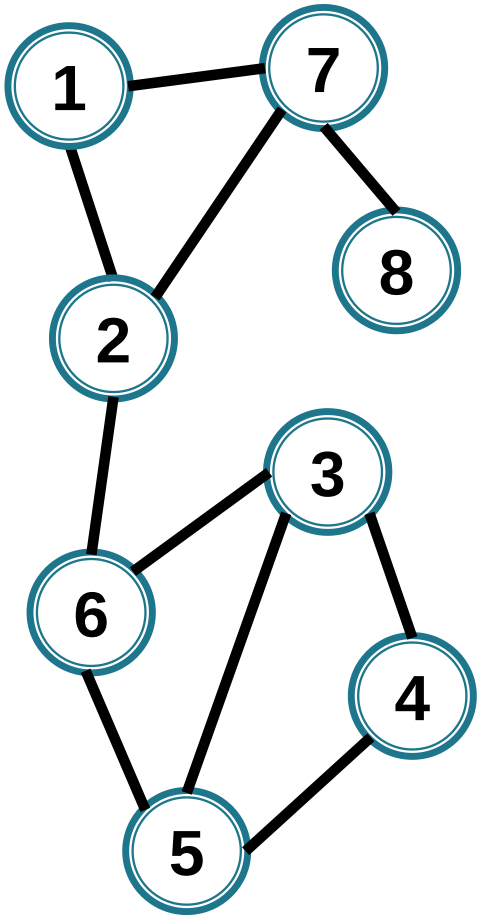


S:
1 2
2 6
6 3
3 4
4 5
5 3
5 6

nivel/niv_min

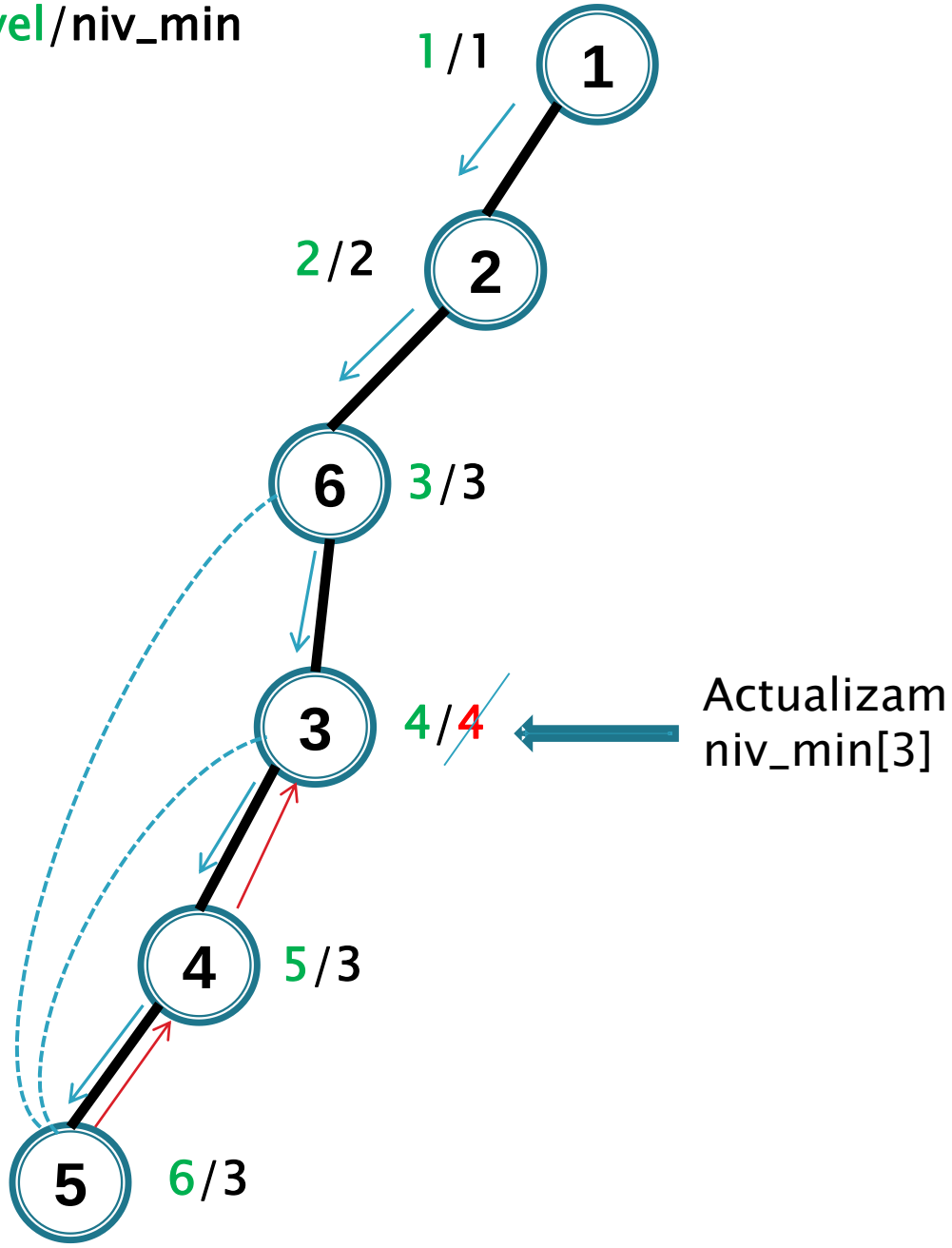


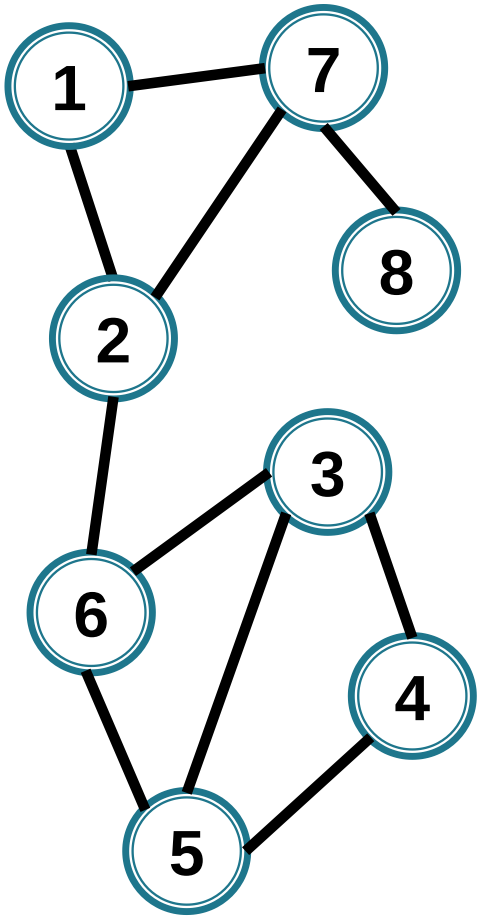
Test punct critic:
niv_min[4] =3 <
nivel[3] =4 ⇒ **NU**



S:
1 2
2 6
6 3
3 4
4 5
5 3
5 6

nivel/niv_min





S:

1 2

2 6

6 3

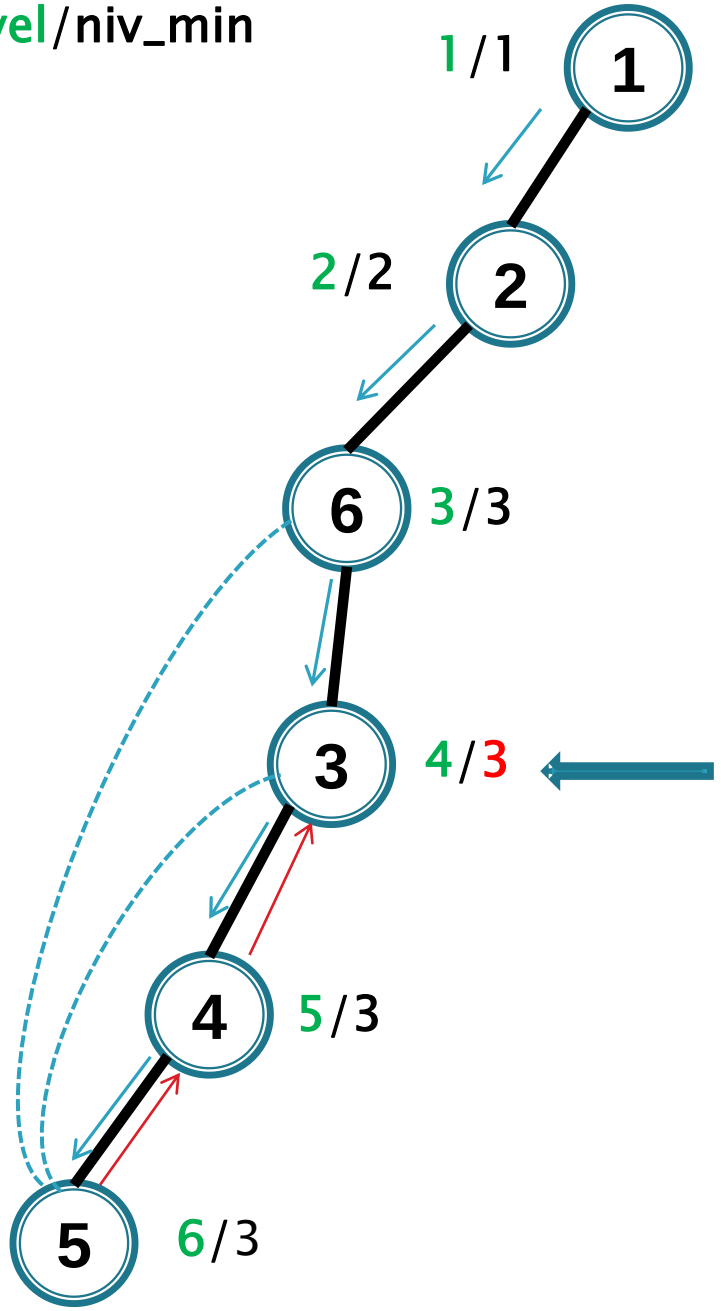
3 4

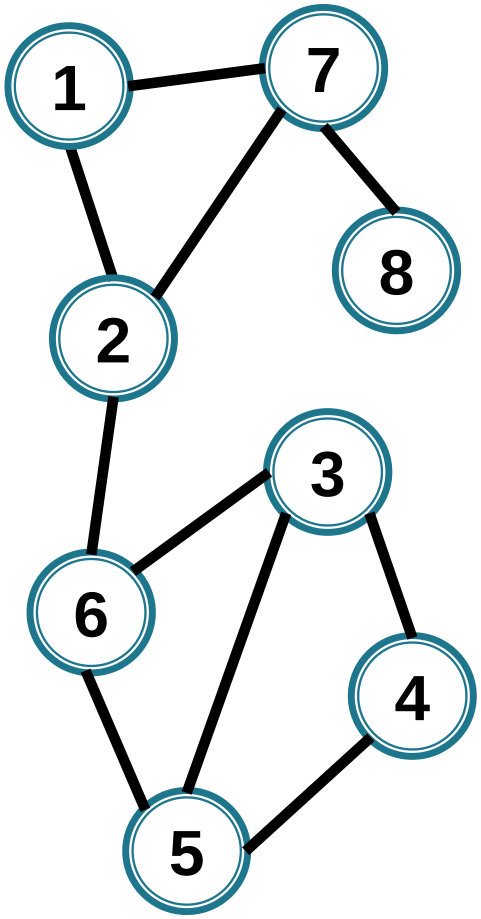
4 5

5 3

5 6

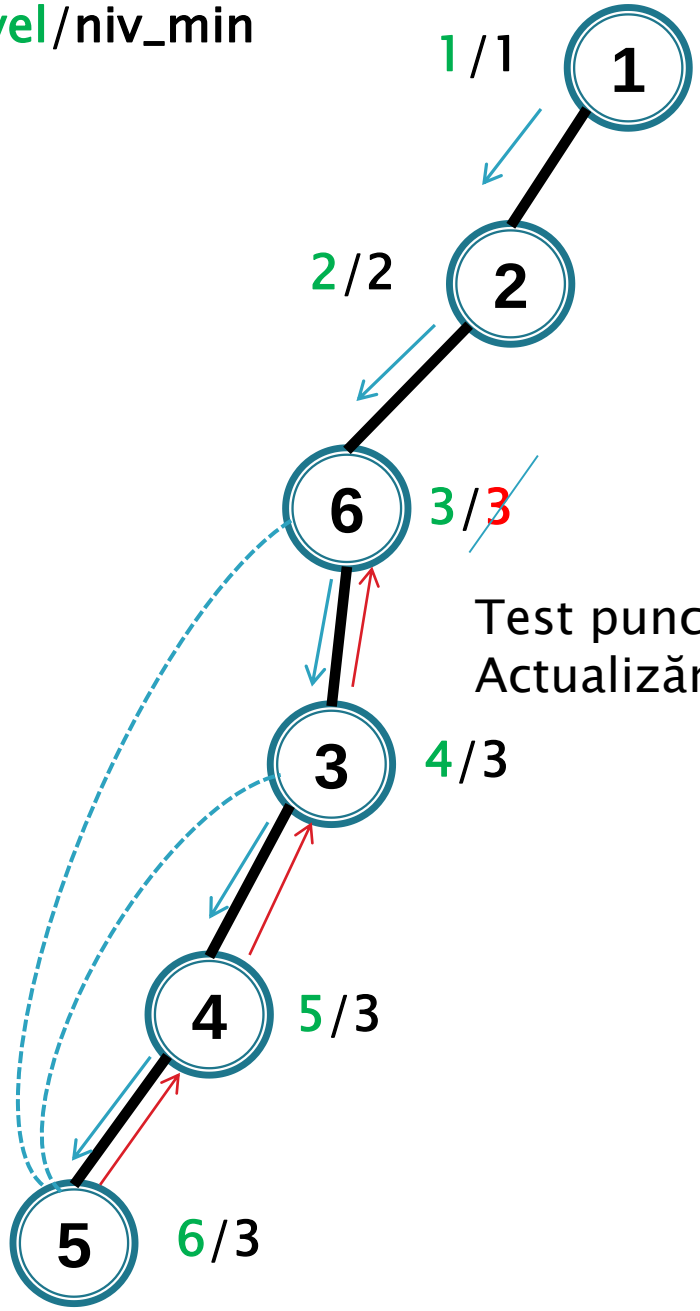
nivel/niv_min

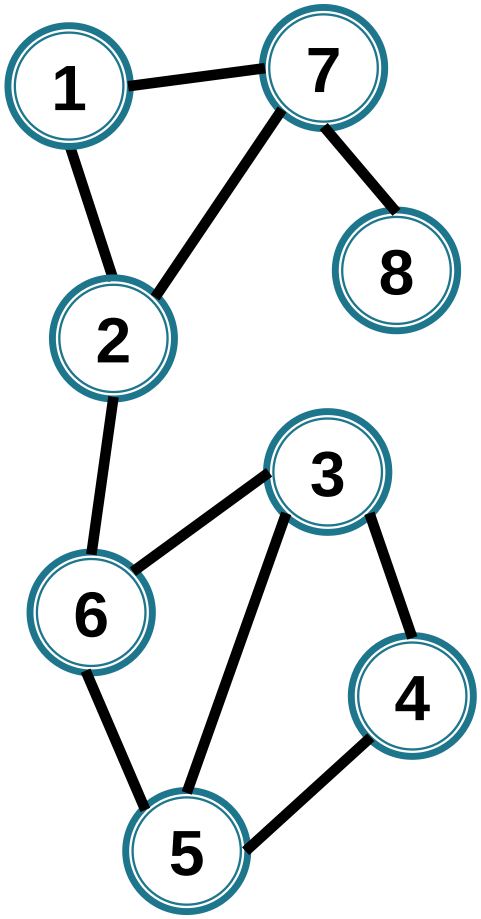




S:
1 2
2 6
6 3
3 4
4 5
5 3
5 6

nivel/niv_min





S:

1 2

2 6

6 3

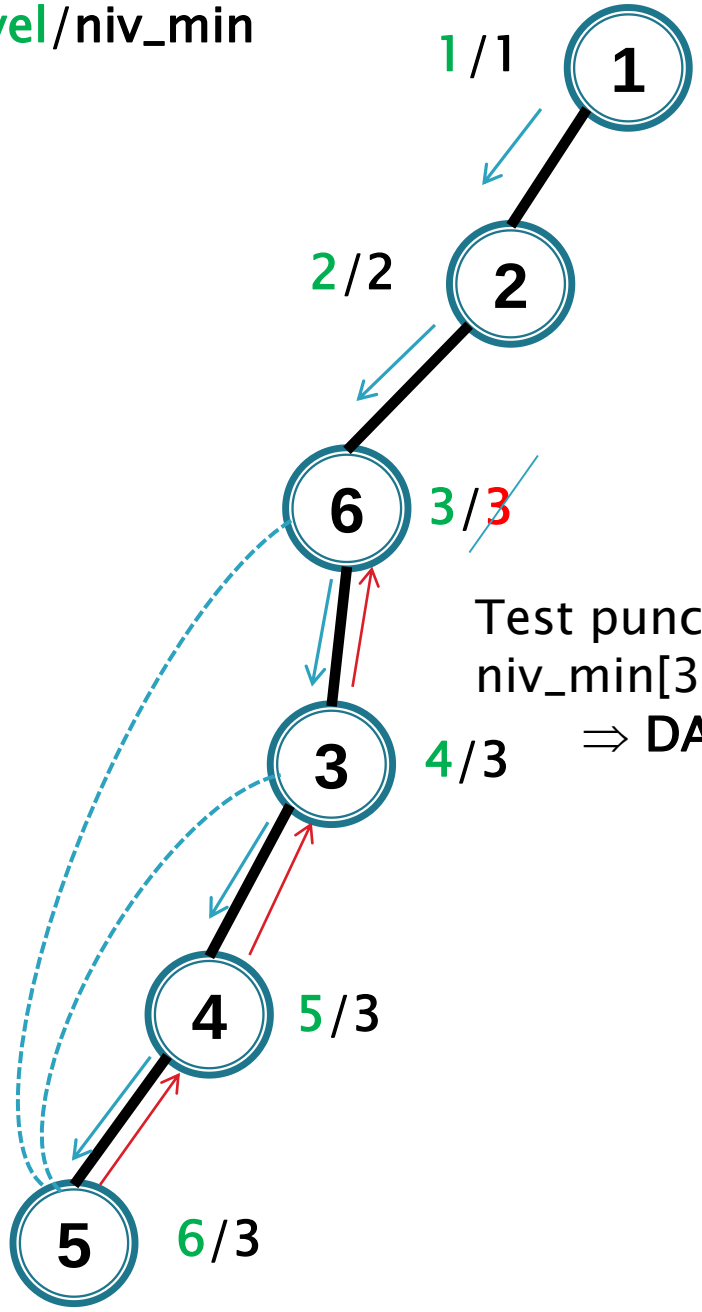
3 4

4 5

5 3

5 6

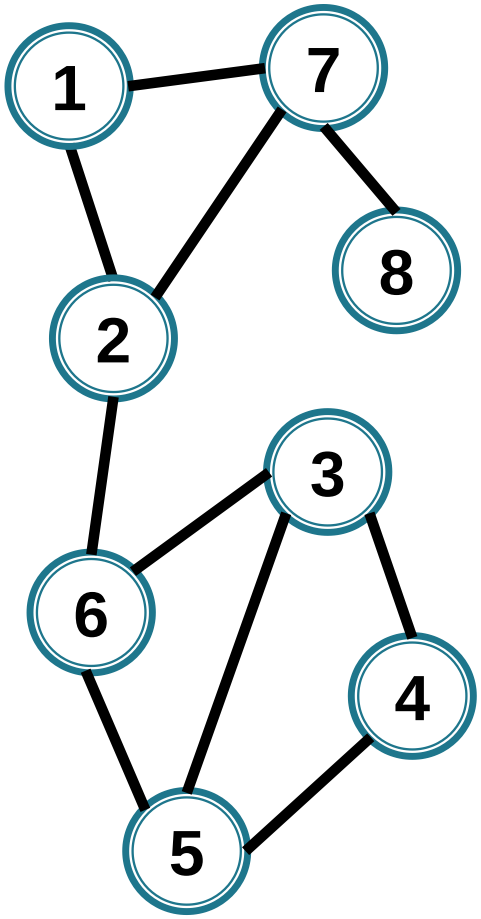
nivel/niv_min



Test punct critic:

niv_min[3] =3 = nivel[6]

$\Rightarrow$ DA



S:

1 2

2 6

6 3

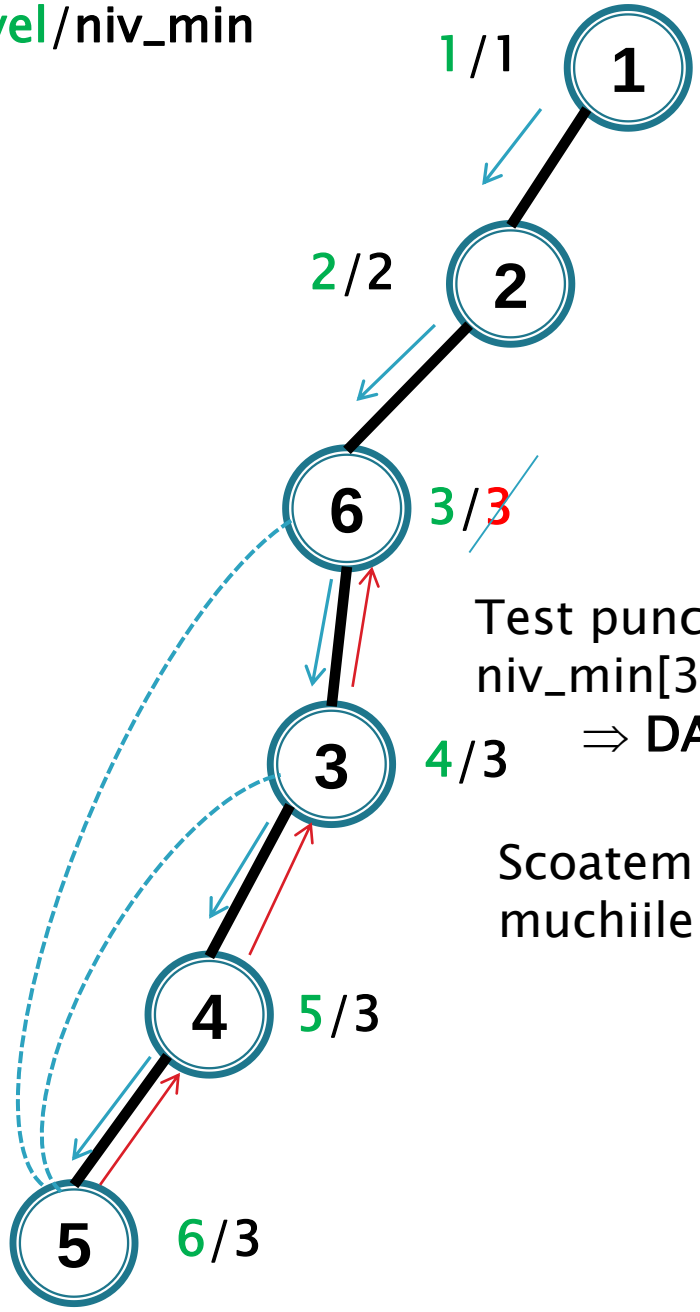
3 4

4 5

5 3

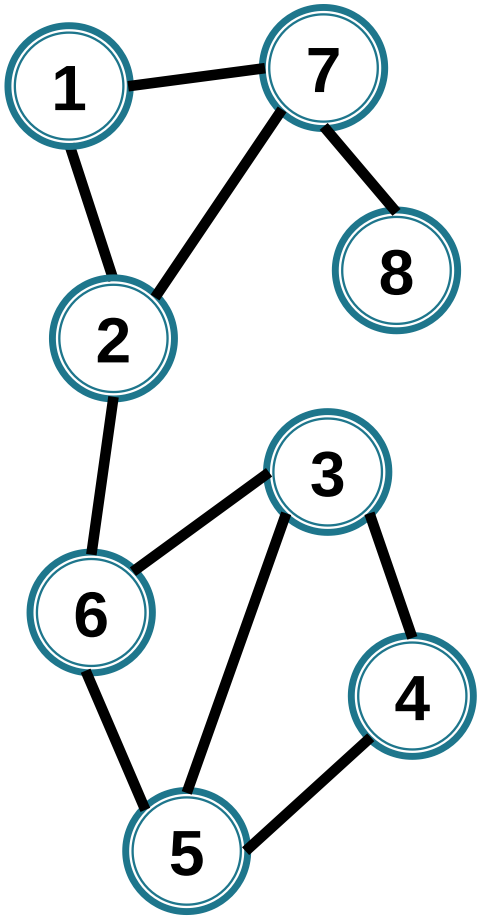
5 6

nivel/niv_min



Test punct critic:
niv_min[3] =3 = nivel[6]
⇒ DA

Scoatem din stiva toate muchiile pana la 3 6 =>

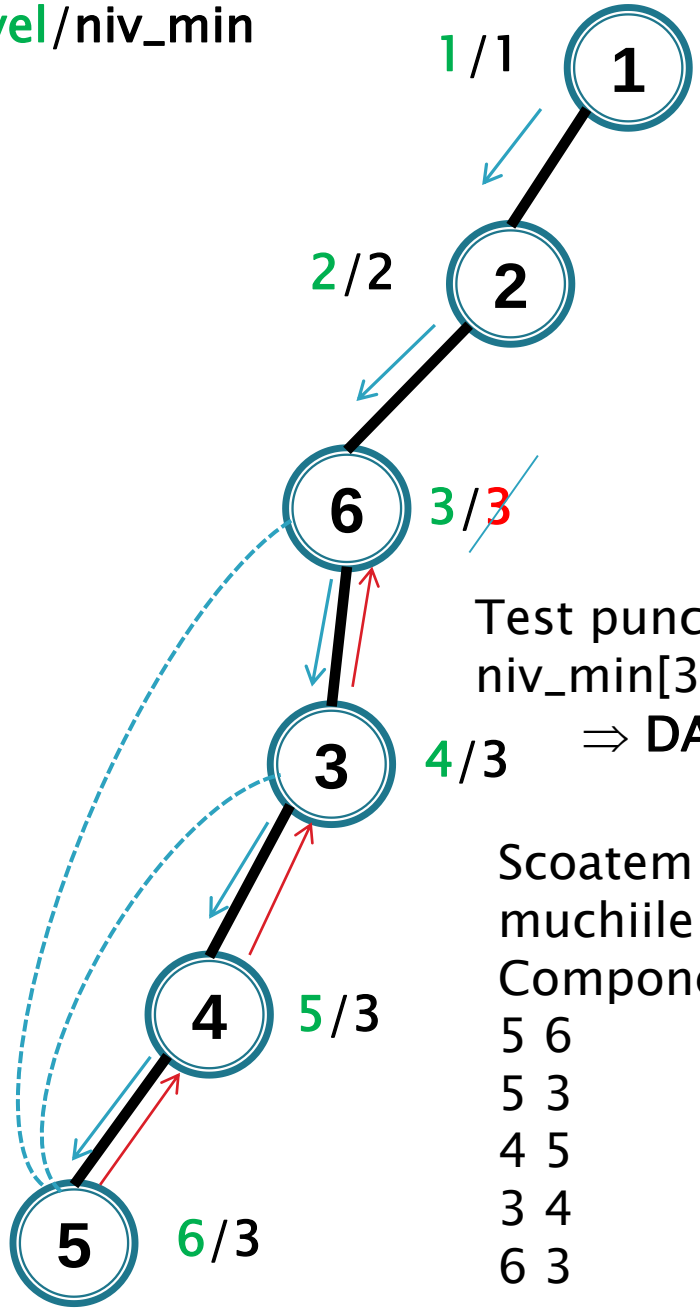


S:

1 2

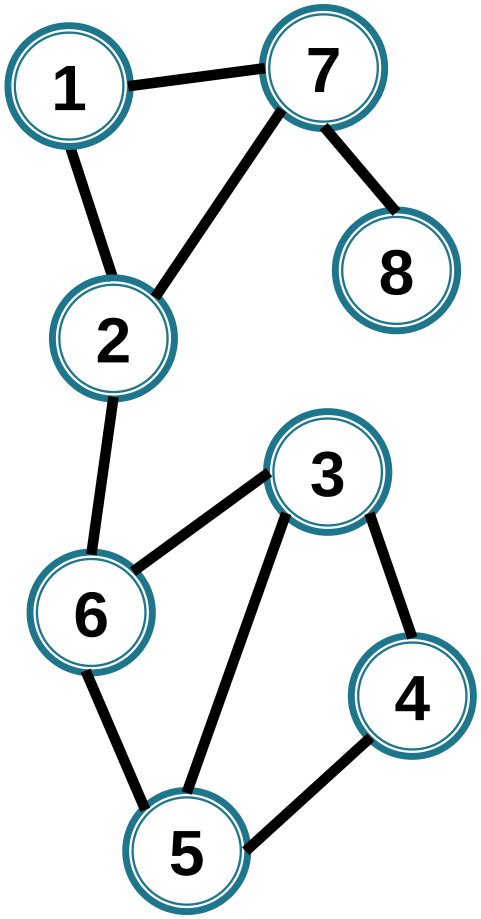
2 6

nivel/niv_min



Test punct critic:
niv_min[3] =3 = nivel[6]
$\Rightarrow$ DA

Scoatem din stiva toate muchiile pana la 3 6 =>
Componenta cu muchiile
5 6
5 3
4 5
3 4
6 3

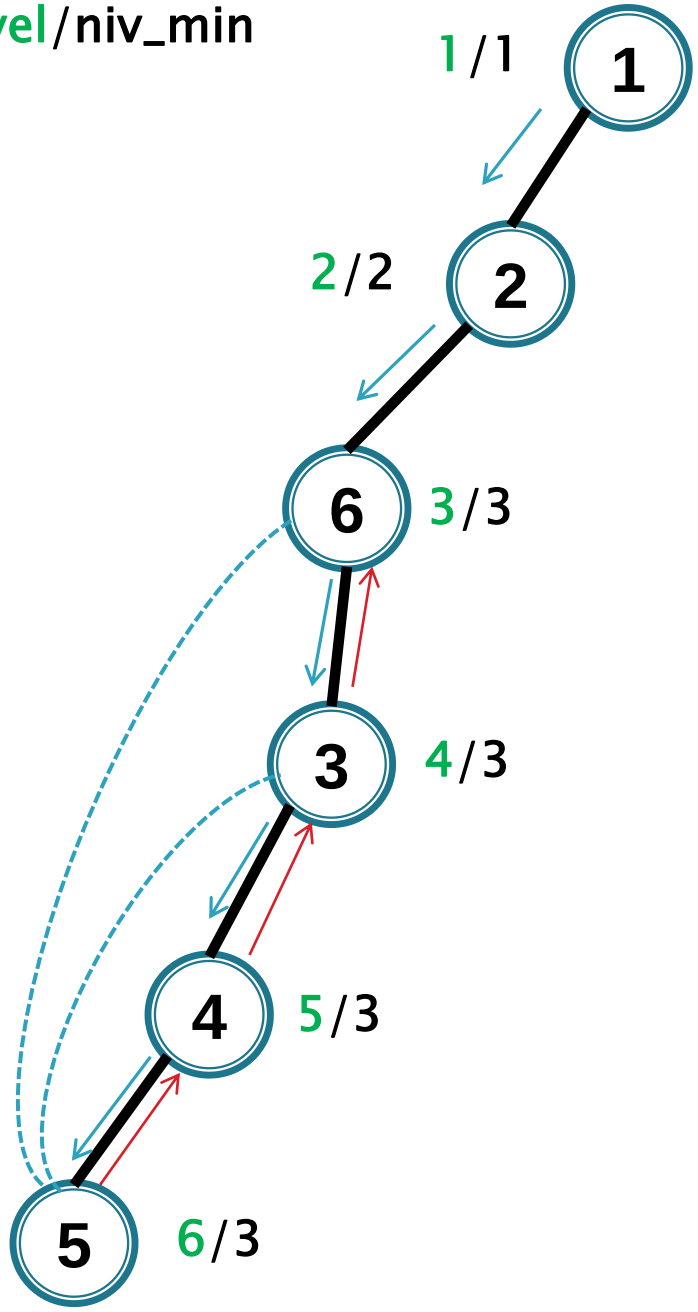


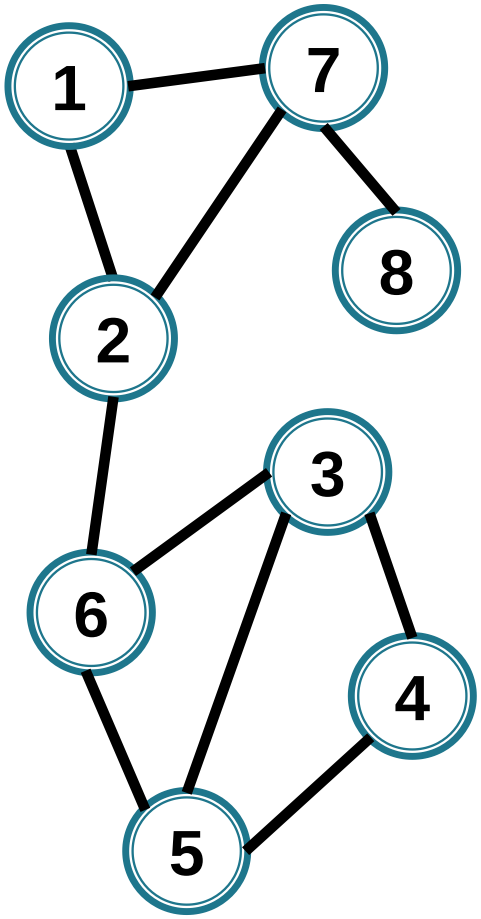
S:

1 2

2 6

nivel/niv_min



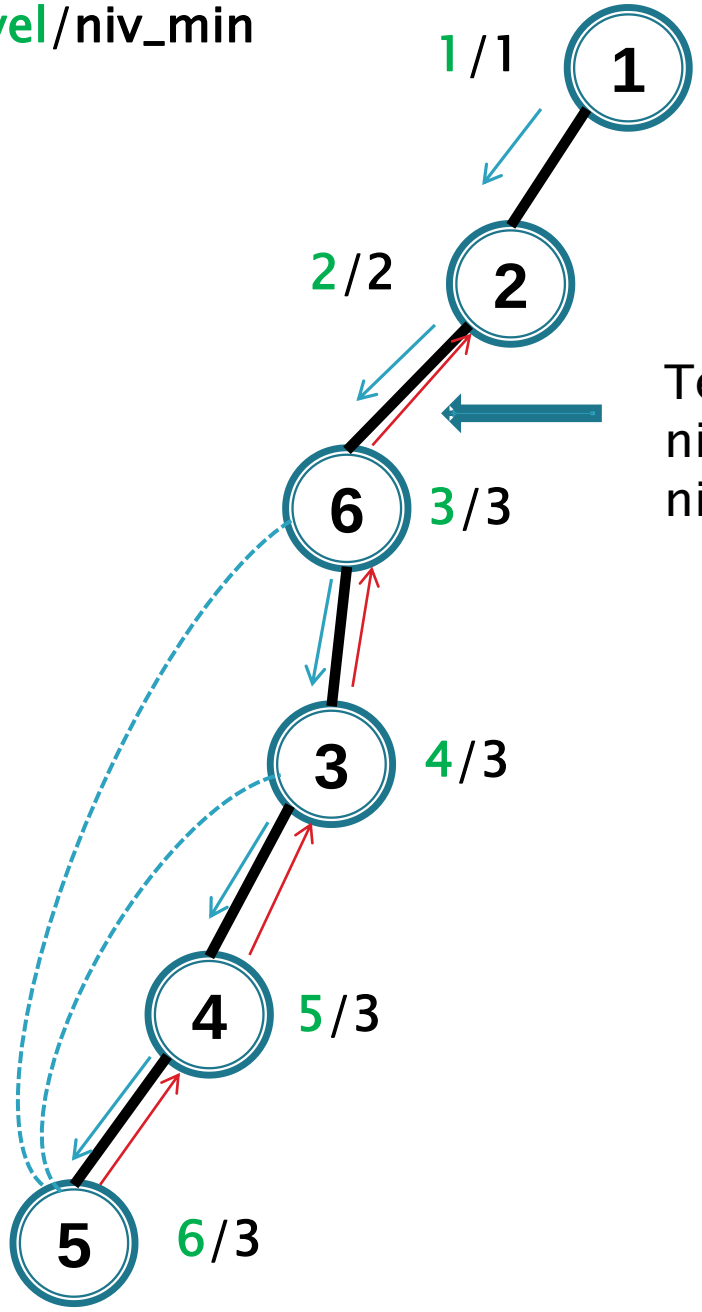


S:

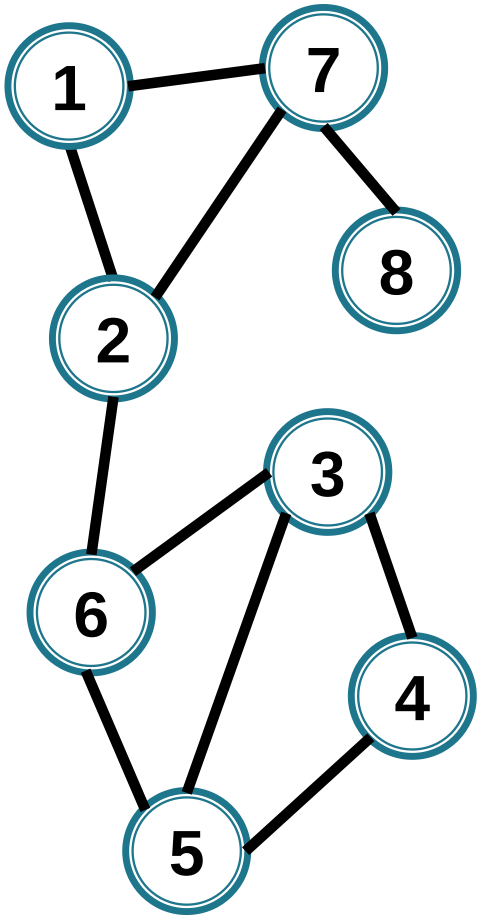
1 2

2 6

nivel/niv_min



Test punct critic:
niv_min[6] =3 >
nivel[2] =2 ⇒ **DA**

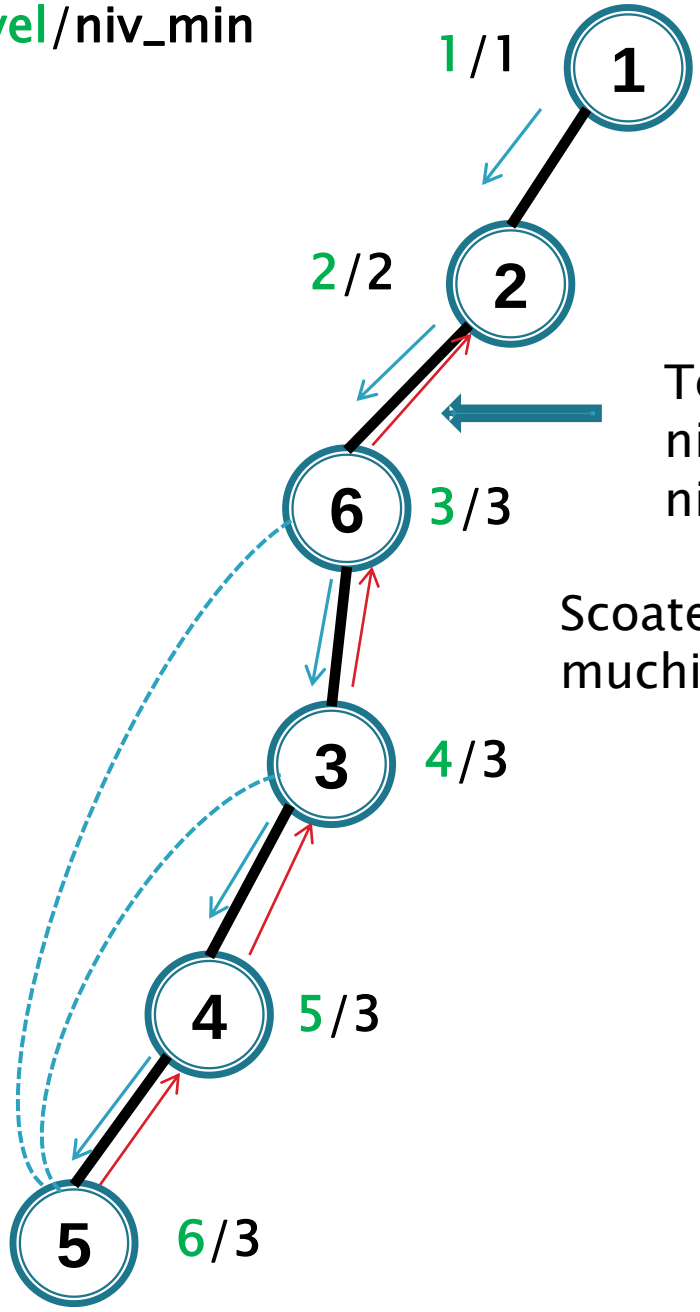


S:

1 2

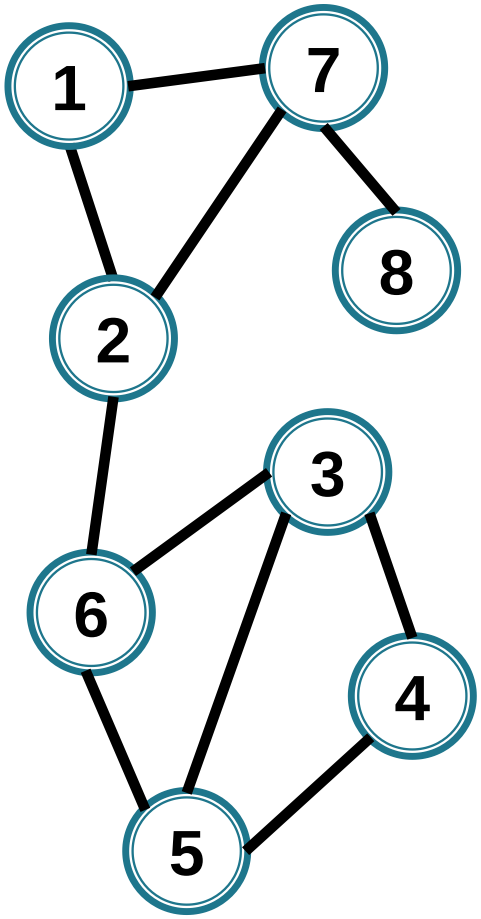
2 6

nivel/niv_min



Test punct critic:
niv_min[6] =3 >
nivel[2] =2 ⇒ **DA**

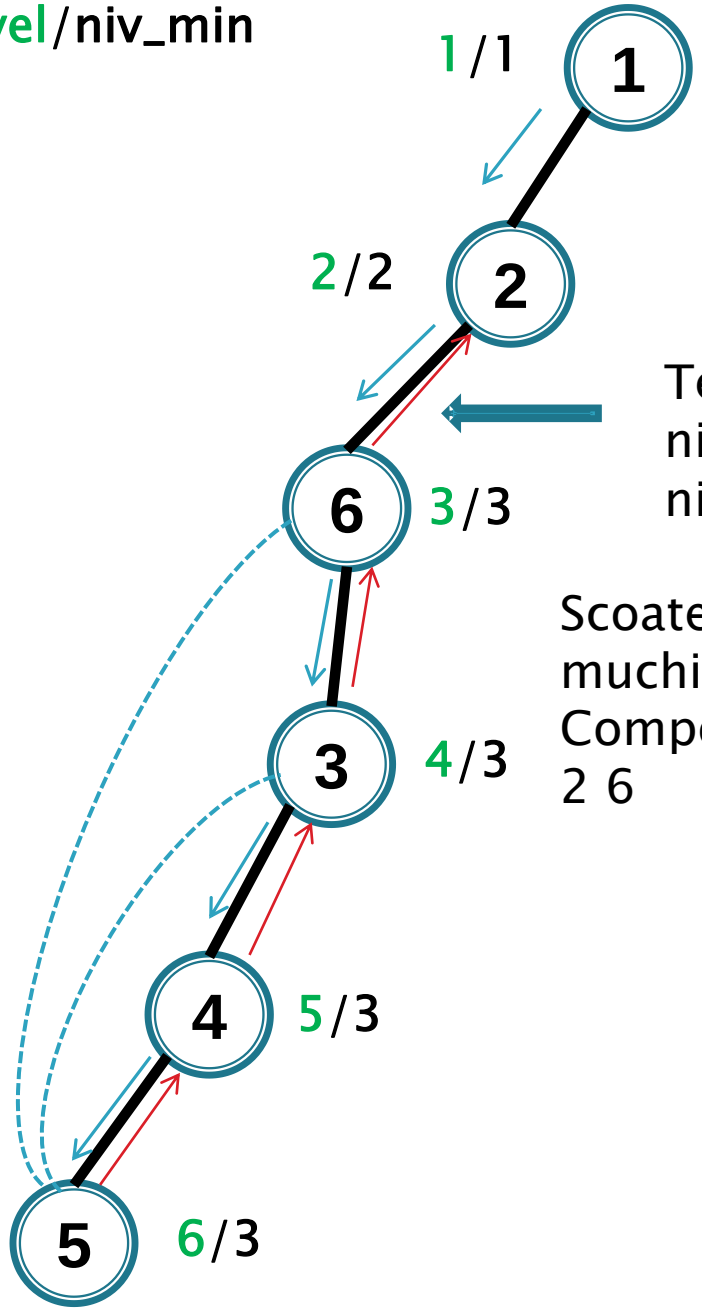
Scoatem din stiva toate muchiile pana la 2 6 =>



S:

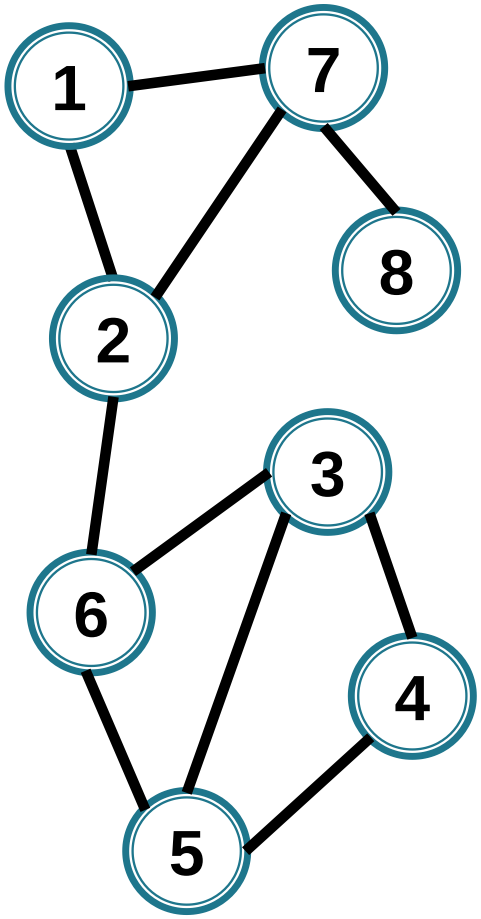
1 2

nivel/niv_min



Test punct critic:
niv_min[6] =3 >
nivel[2] =2 ⇒ **DA**

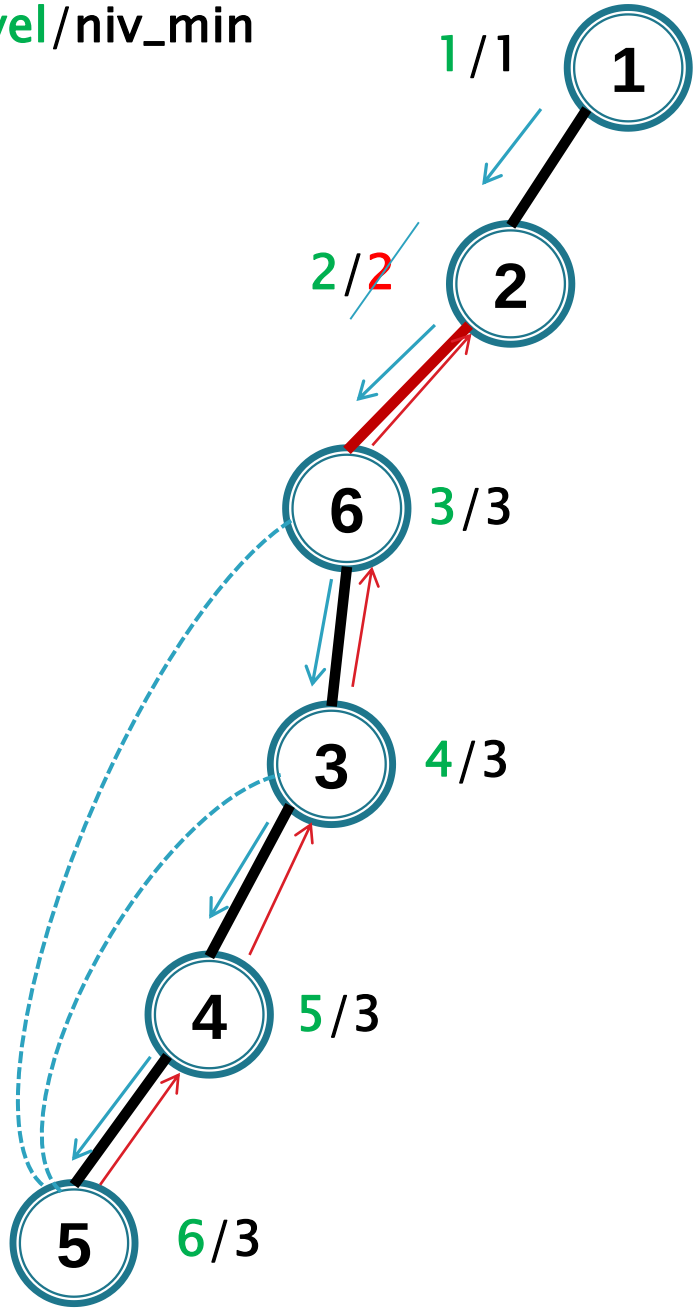
Scoatem din stiva toate muchiile pana la 2 6 =>
Componenta cu muchiile
2 6

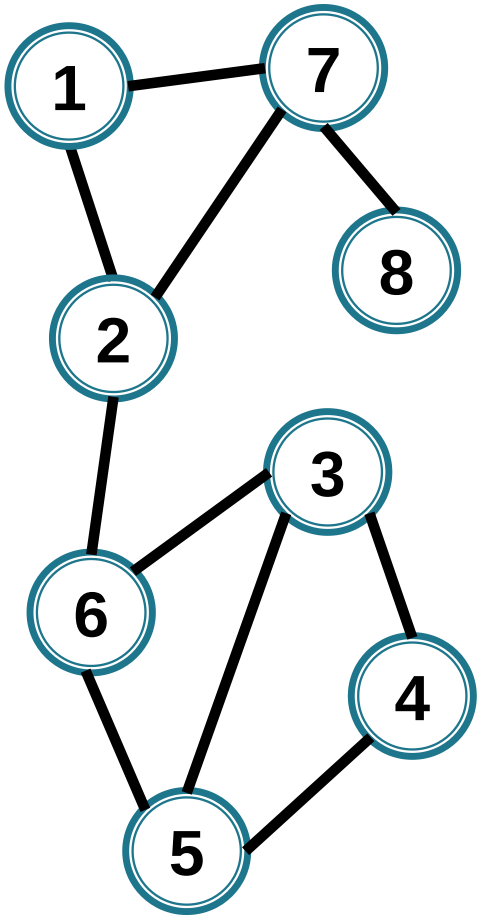


S:

1 2

nivel/niv_min

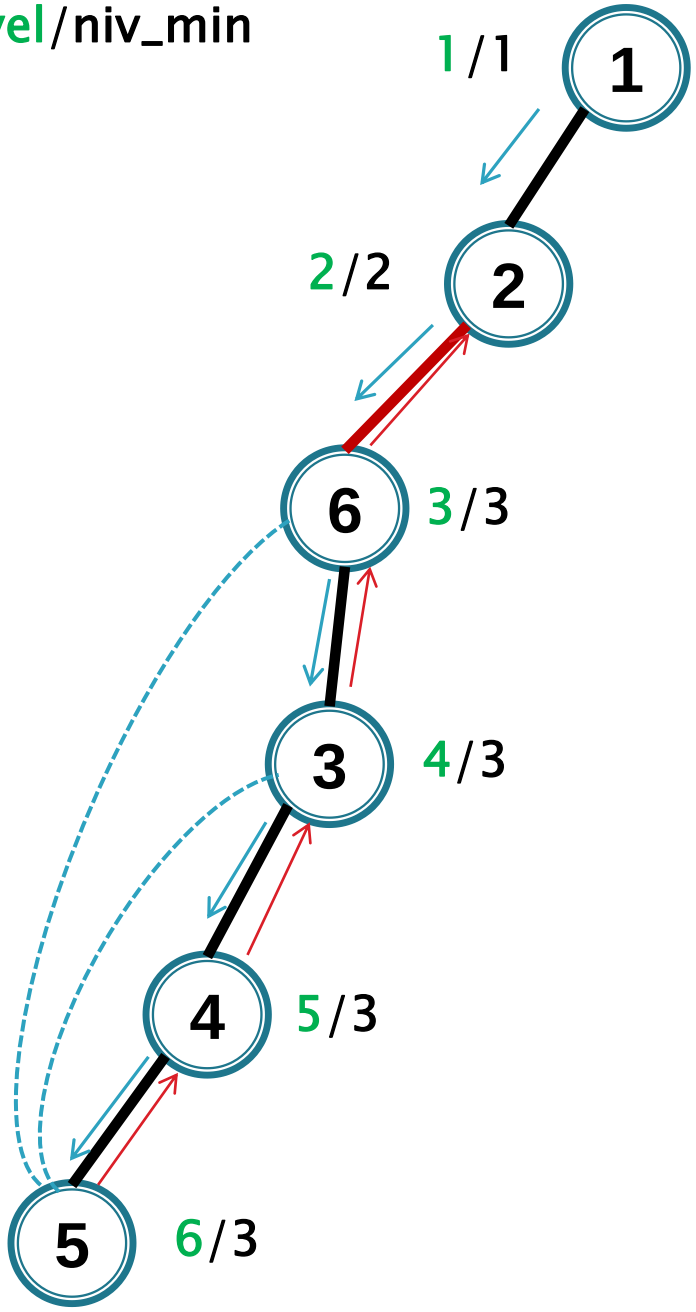


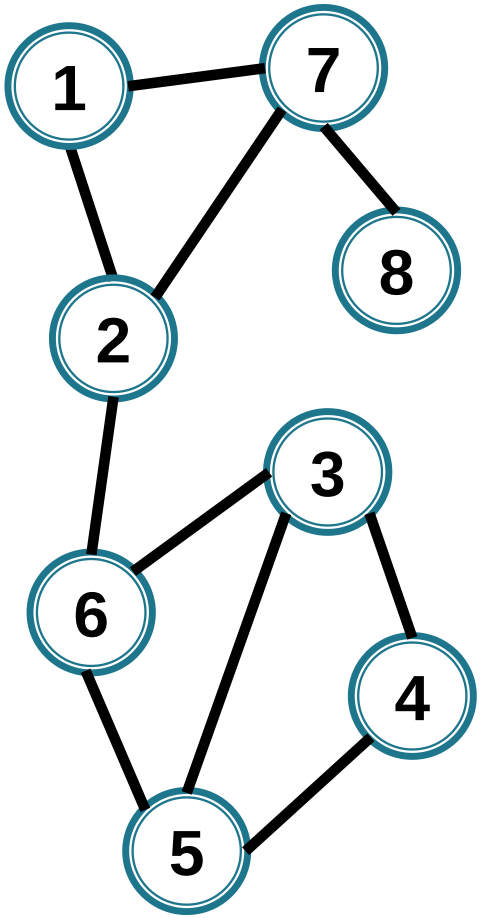


S:

1 2

nivel/niv_min



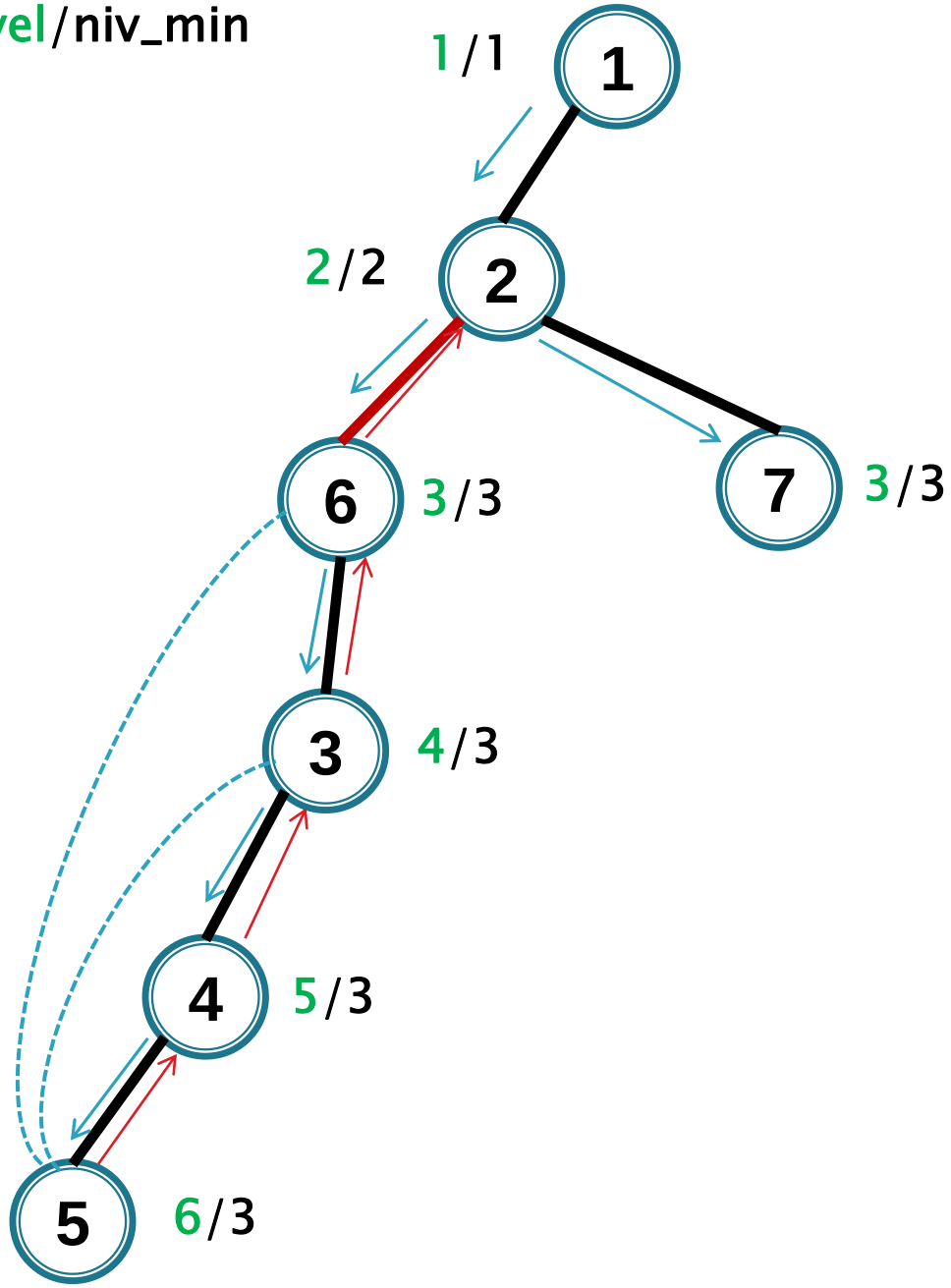


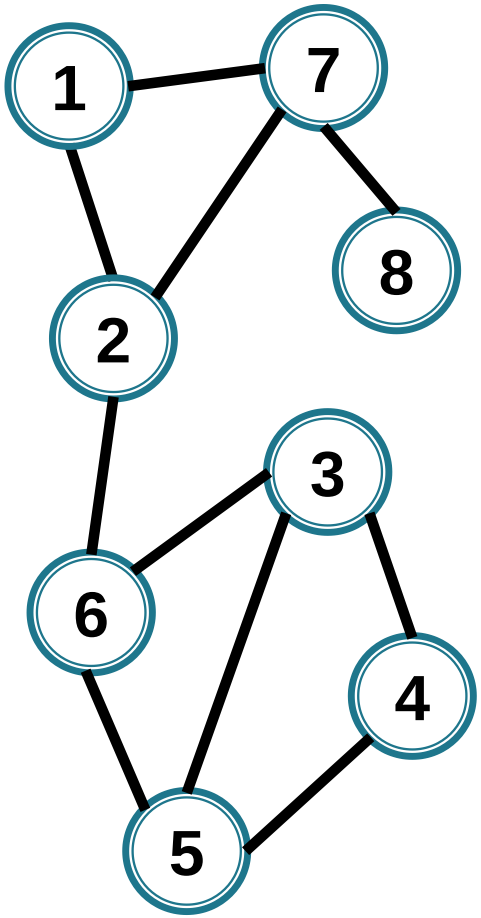
S:

1 2

2 7

nivel/niv_min





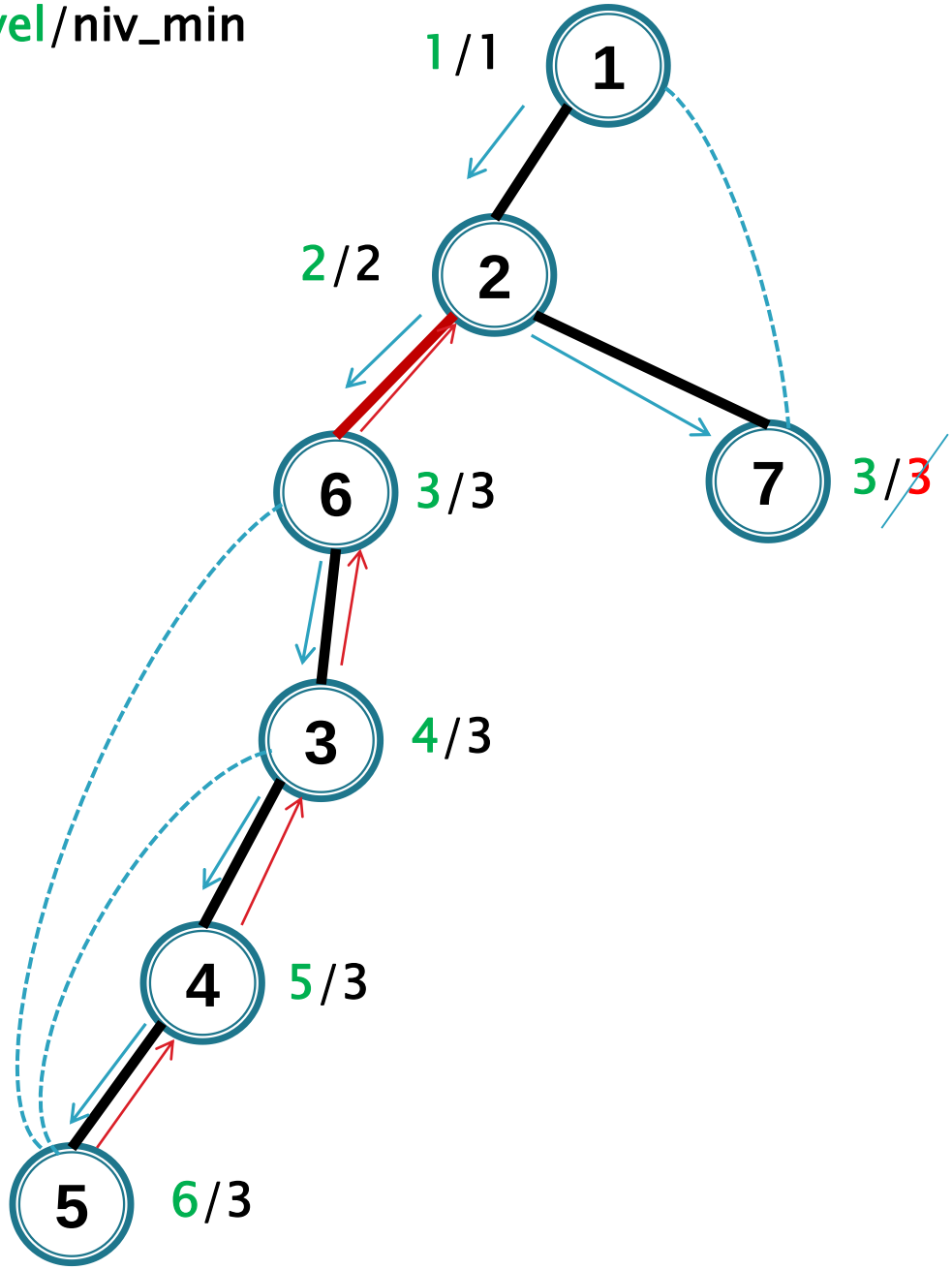
S:

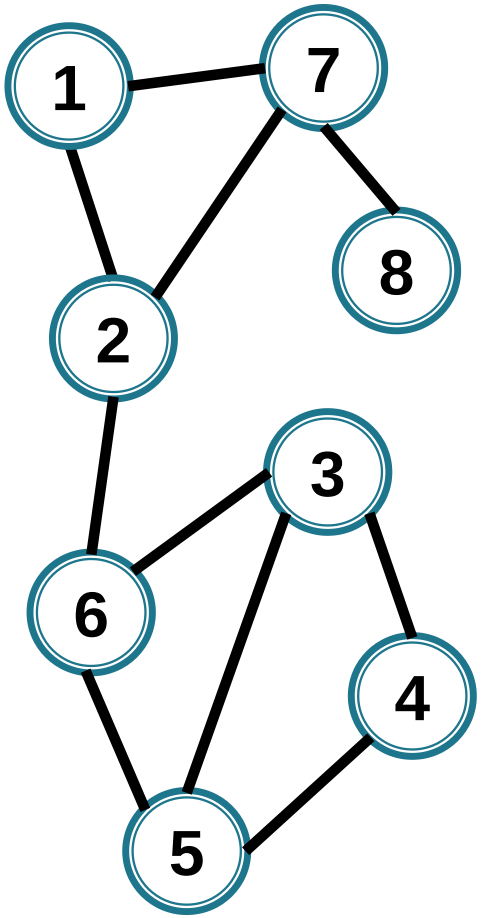
1 2

2 7

1 7

nivel/niv_min





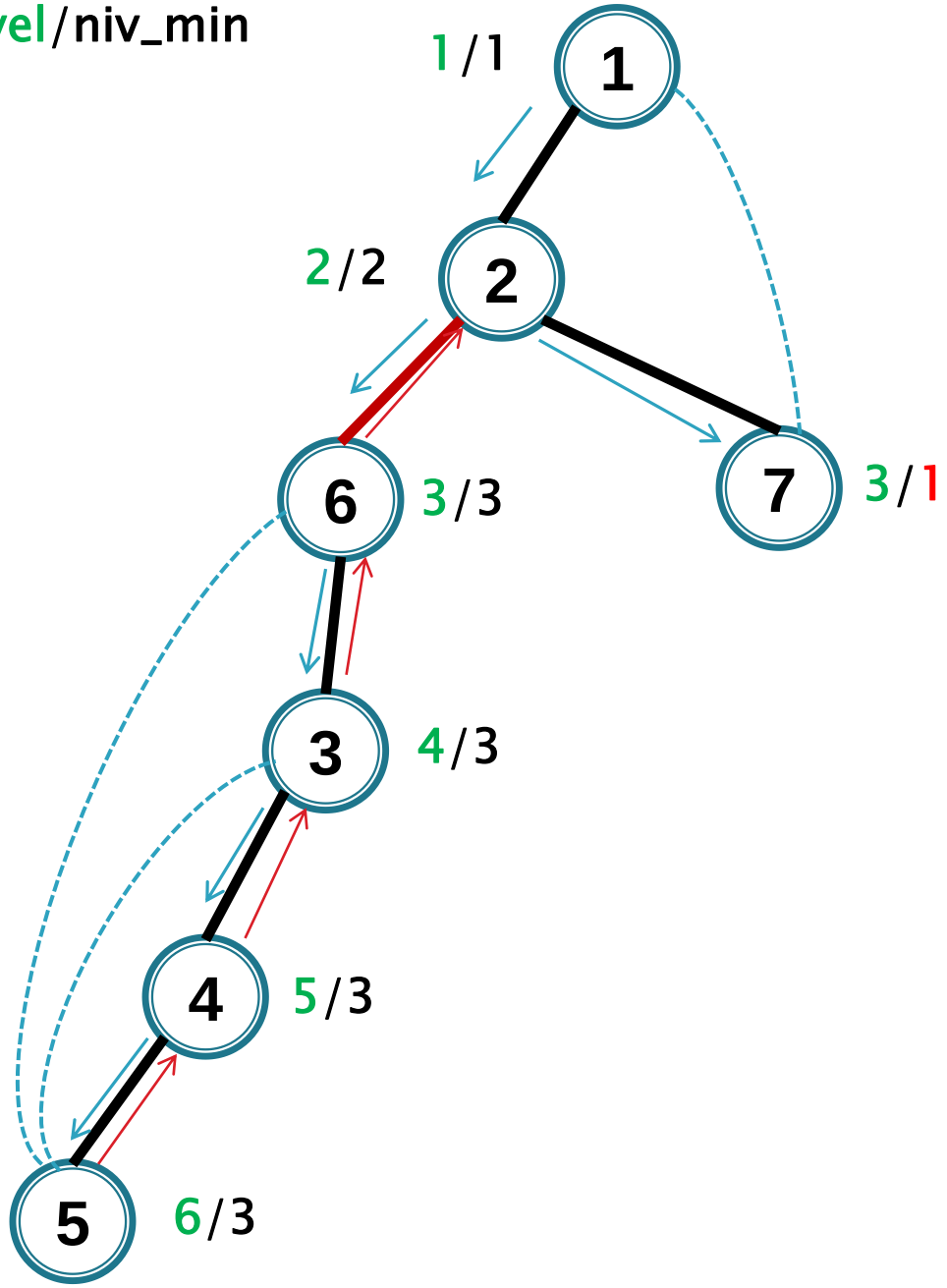
S:

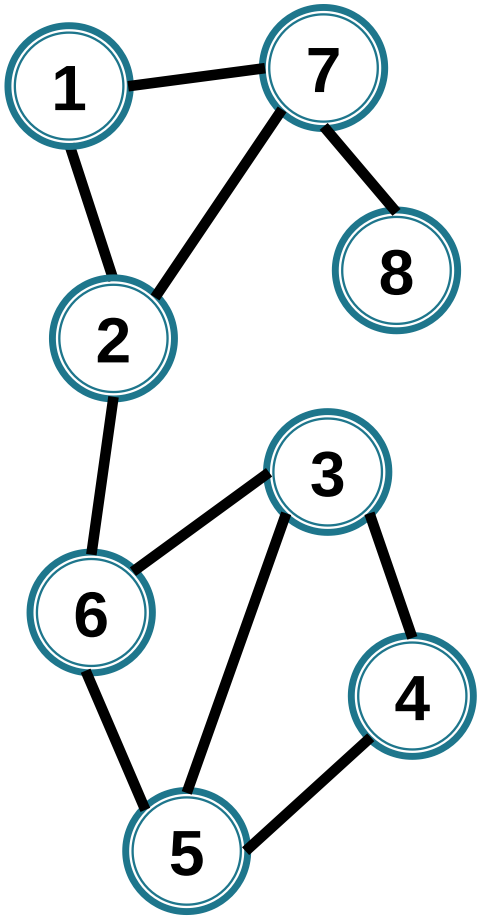
1 2

2 7

1 7

nivel/niv_min





S:

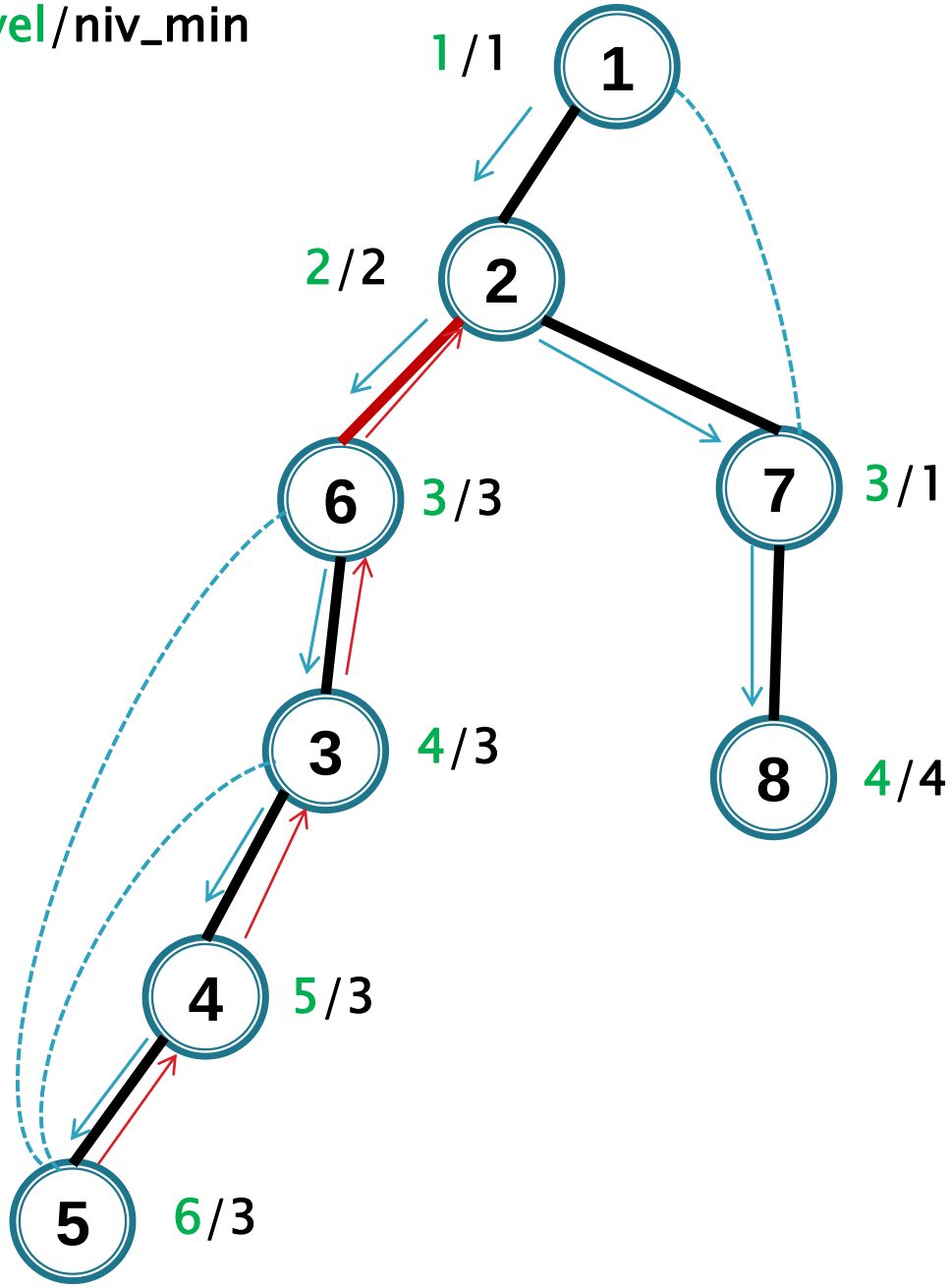
1 2

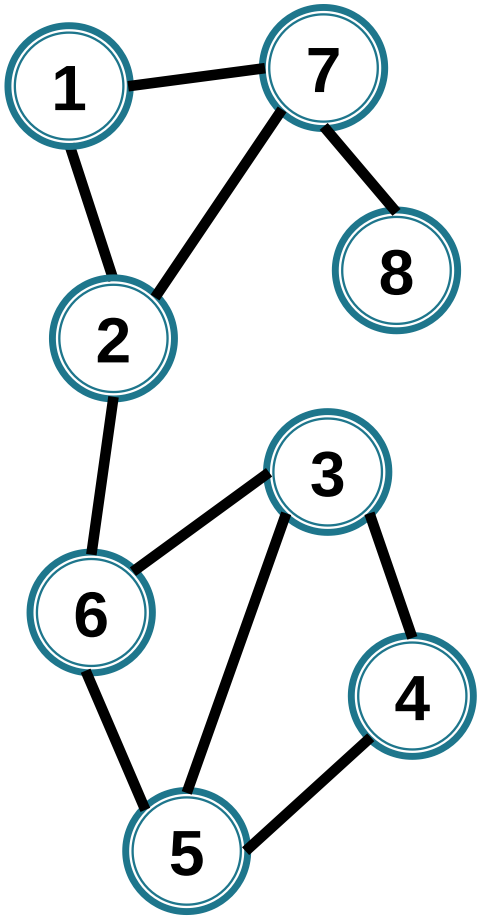
2 7

1 7

7 8

nivel/niv_min





S:

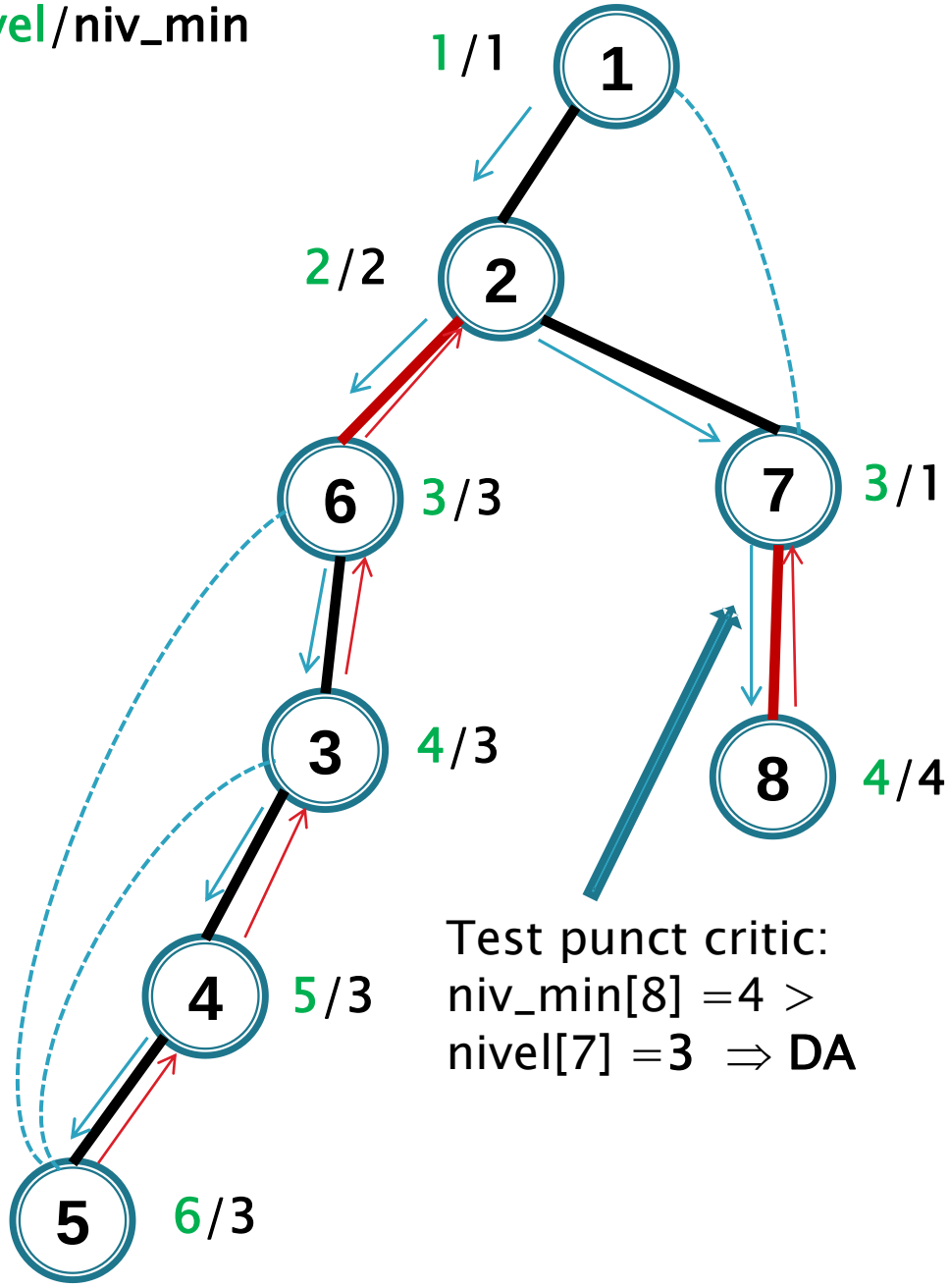
1 2

2 7

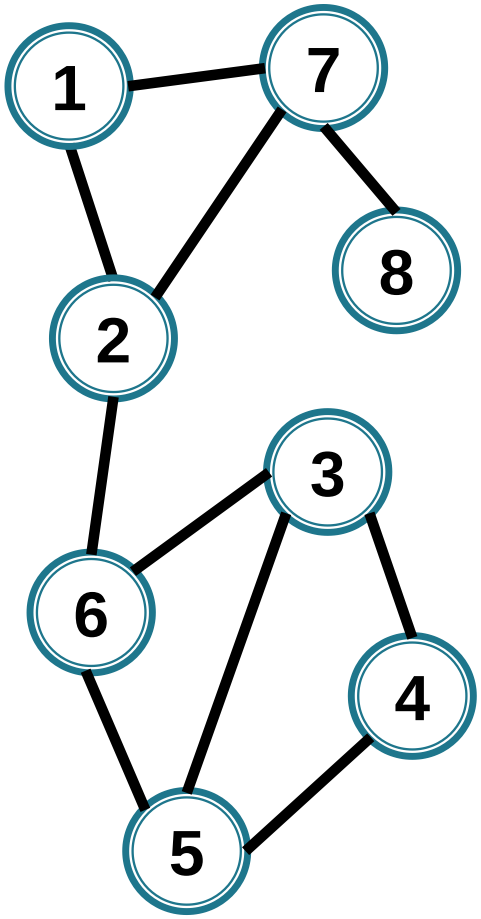
1 7

7 8

nivel/niv_min



Test punct critic:
niv_min[8] =4 >
nivel[7] =3 $\Rightarrow$ **DA**



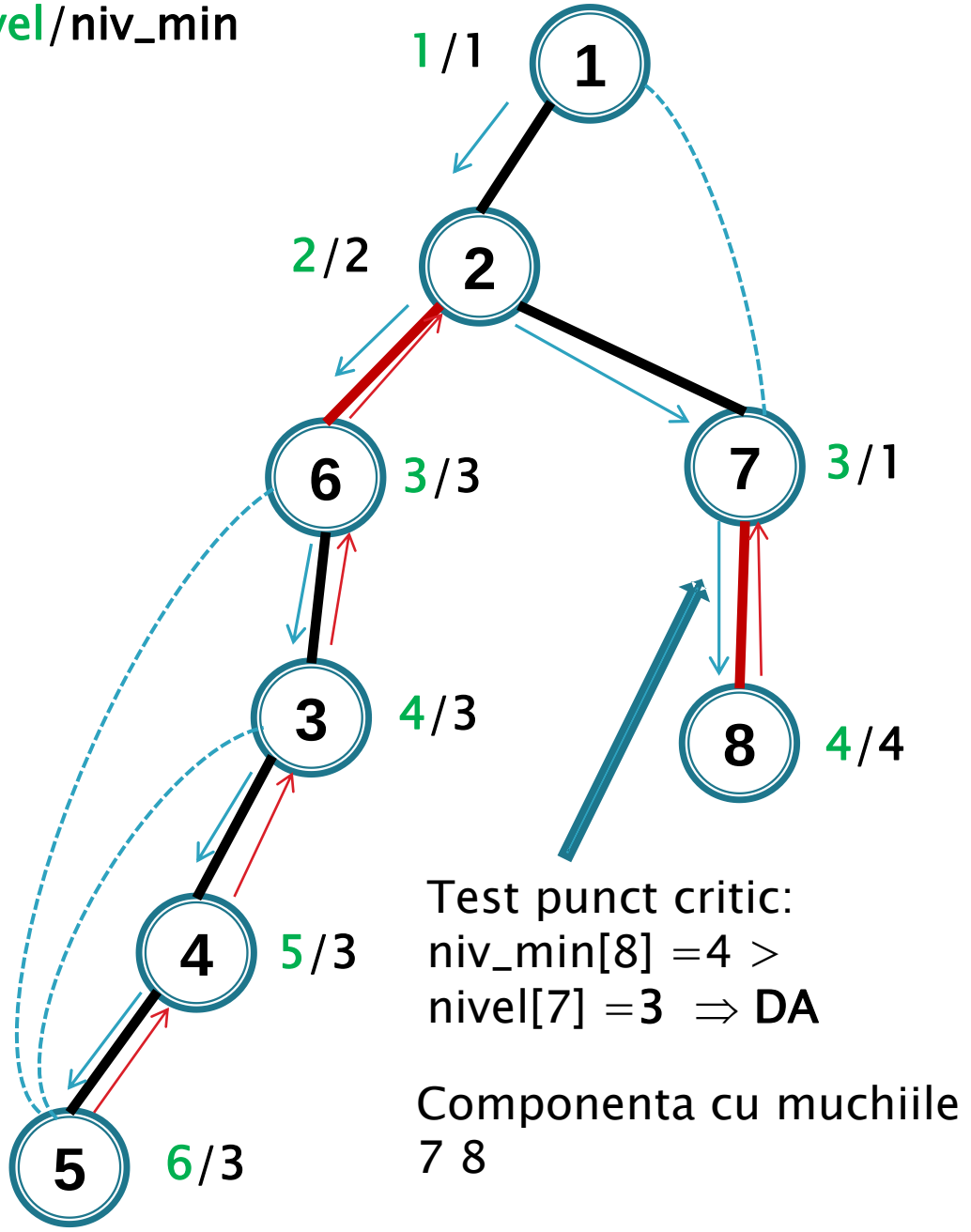
S:

1 2

2 7

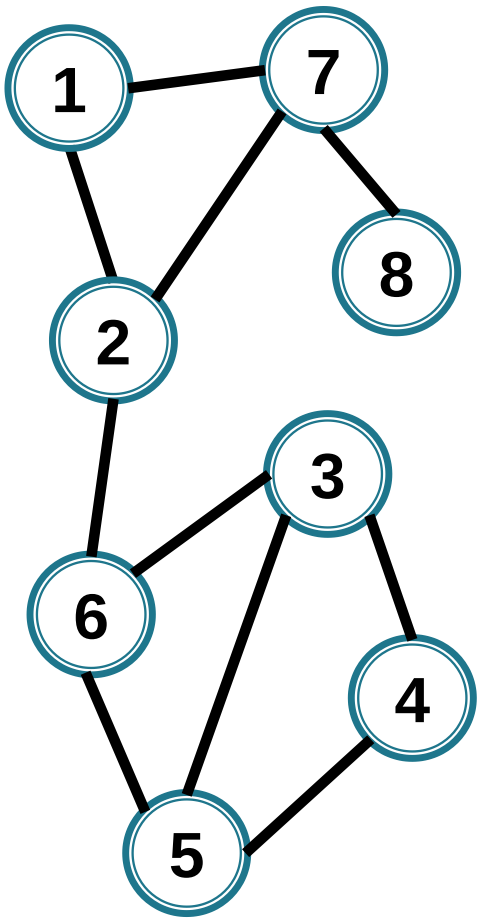
1 7

nivel/niv_min



Test punct critic:
niv_min[8] =4 >
nivel[7] =3 $\Rightarrow$ **DA**

Componenta cu muchiile
7 8



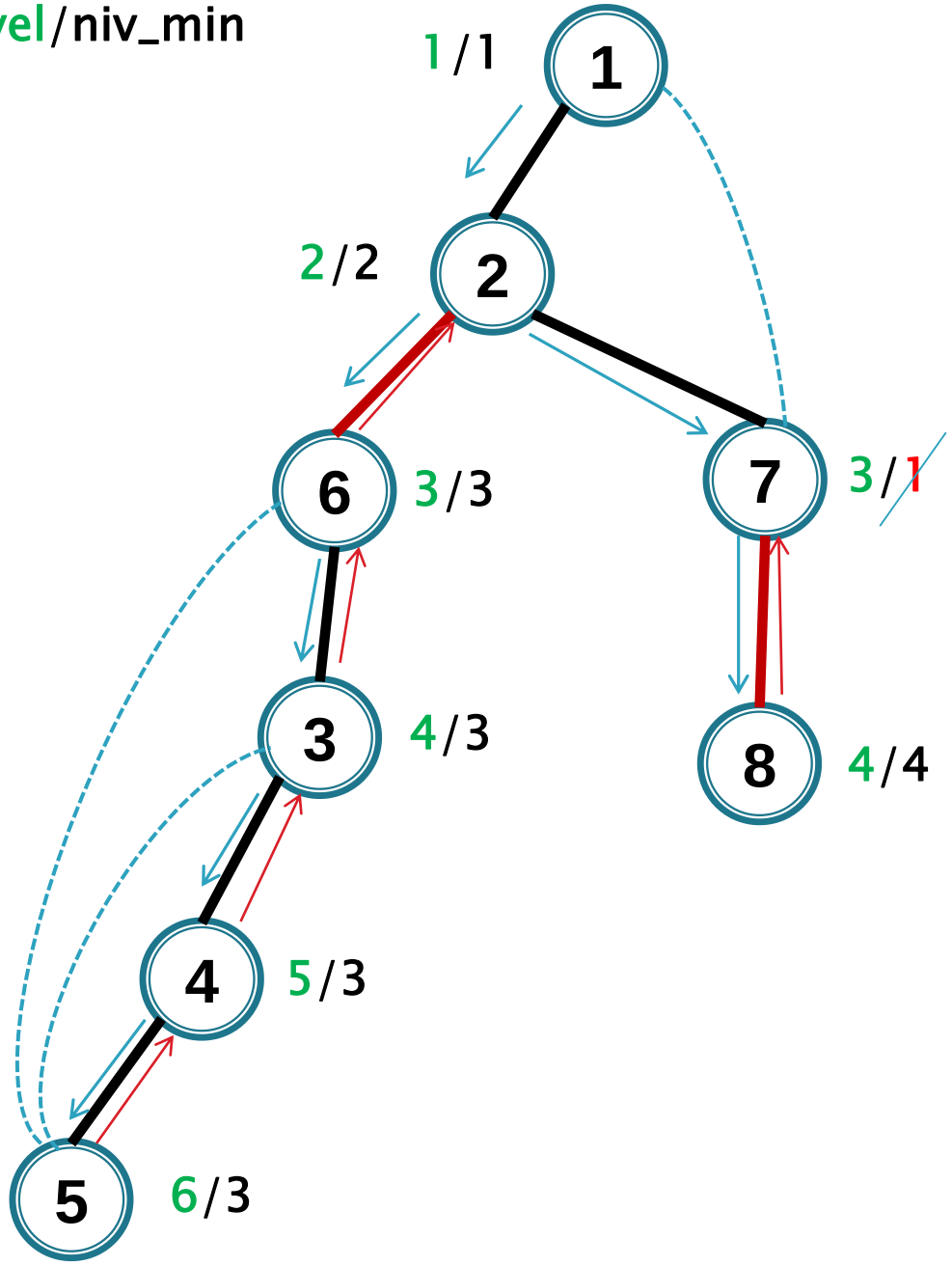
S:

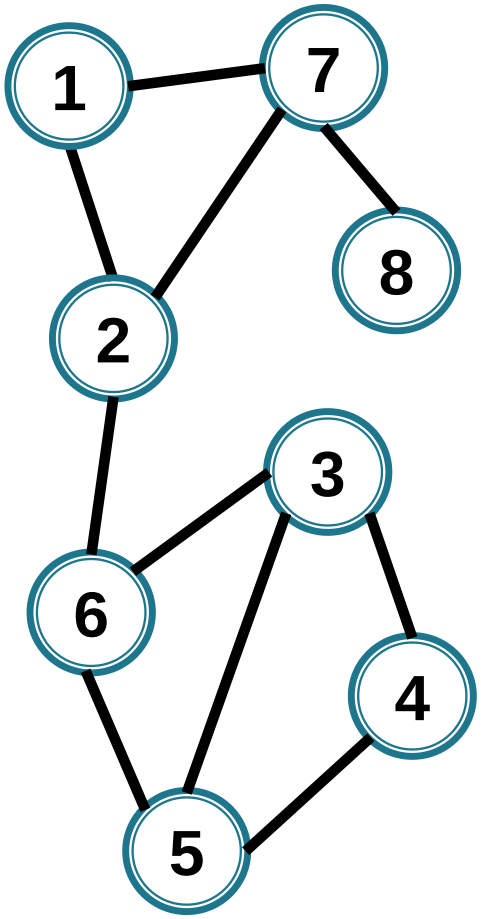
1 2

2 7

1 7

nivel/niv_min





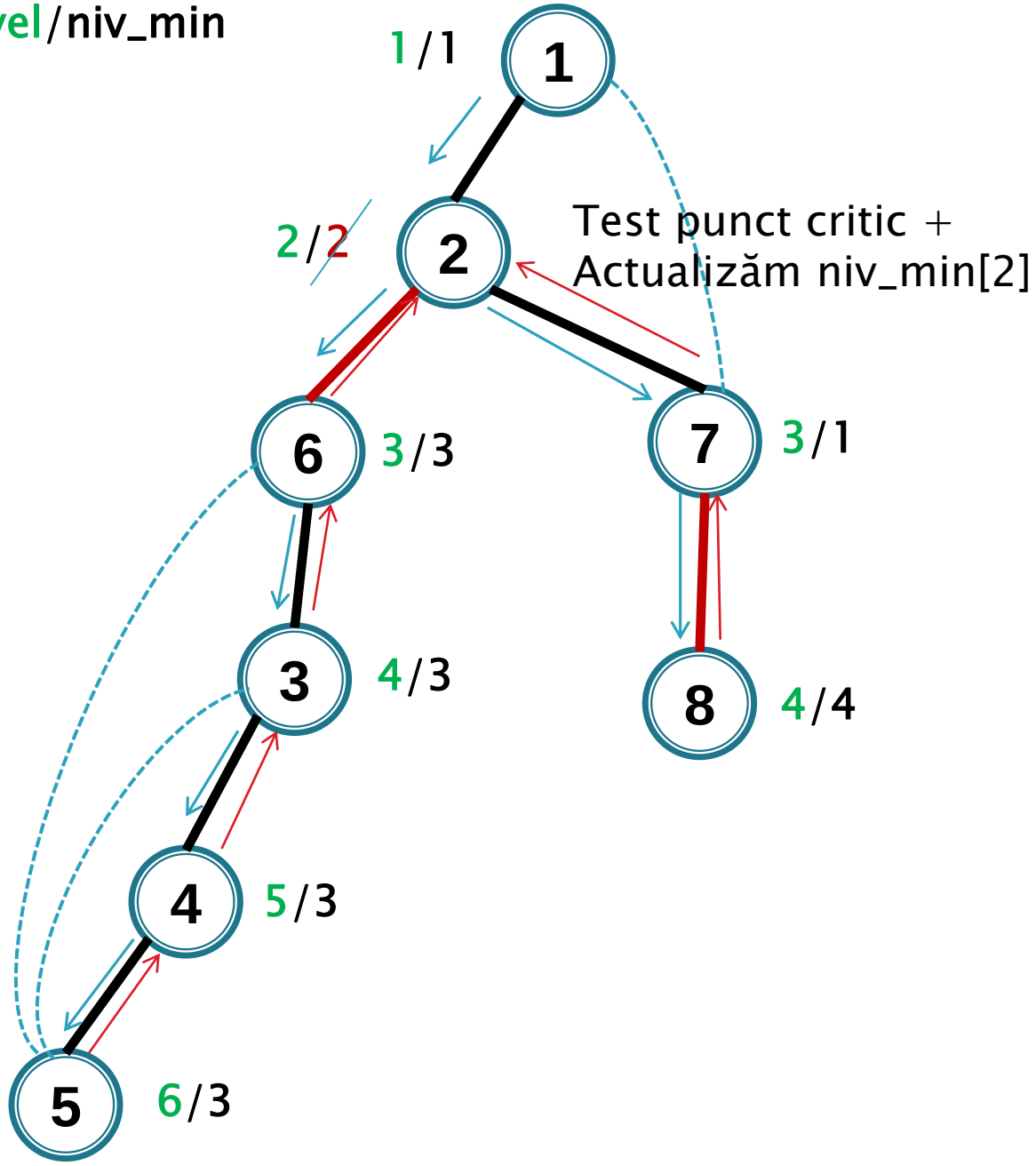
S:

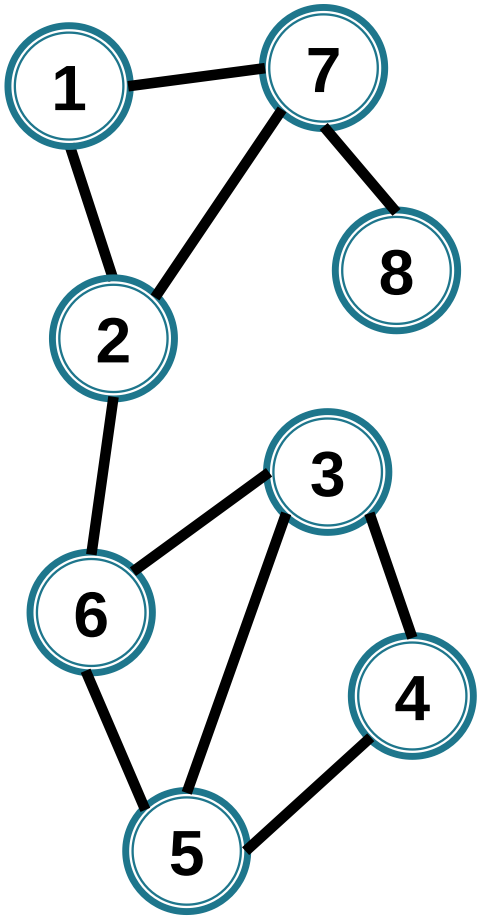
1 2

2 7

1 7

nivel/niv_min





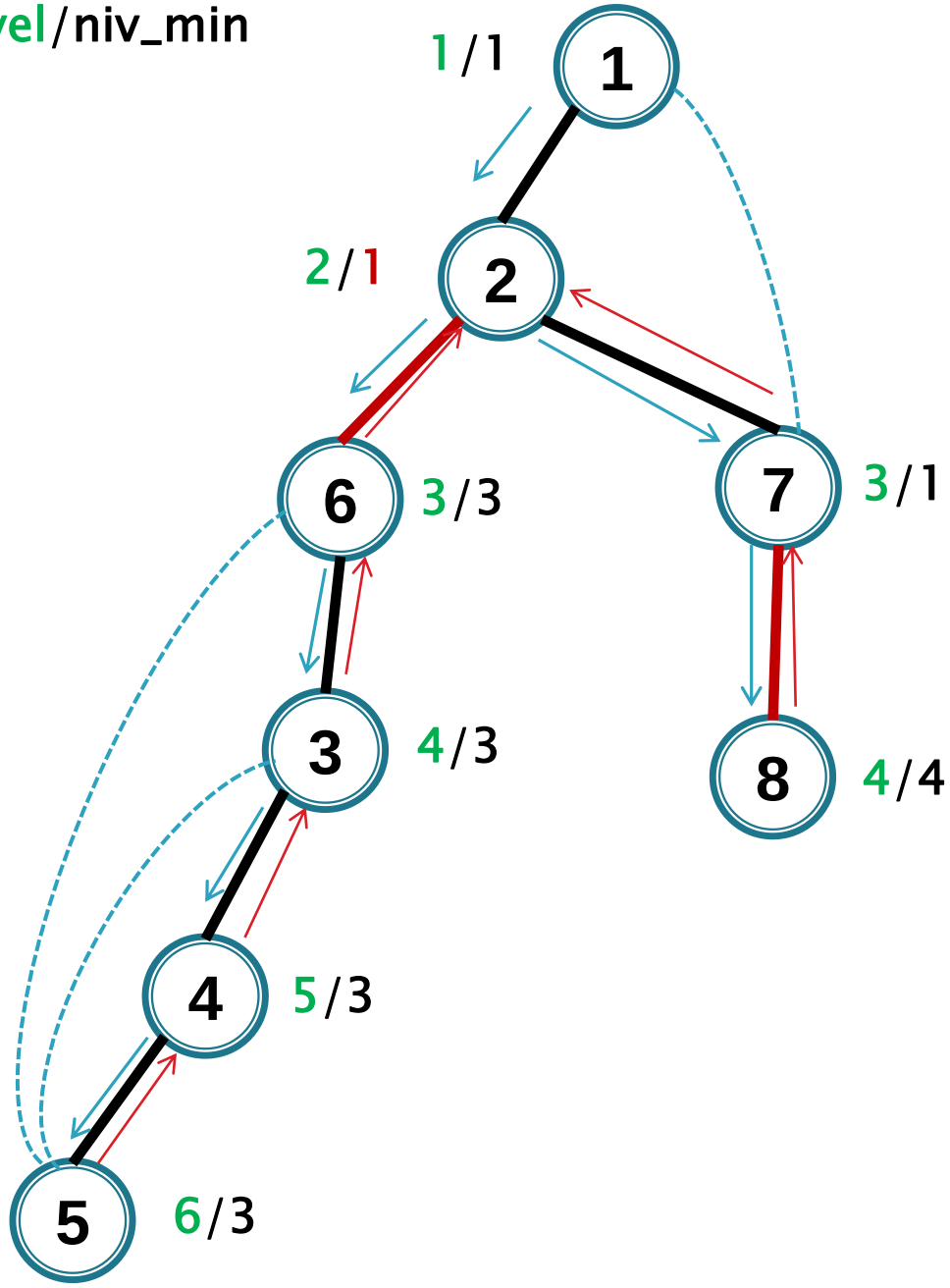
S:

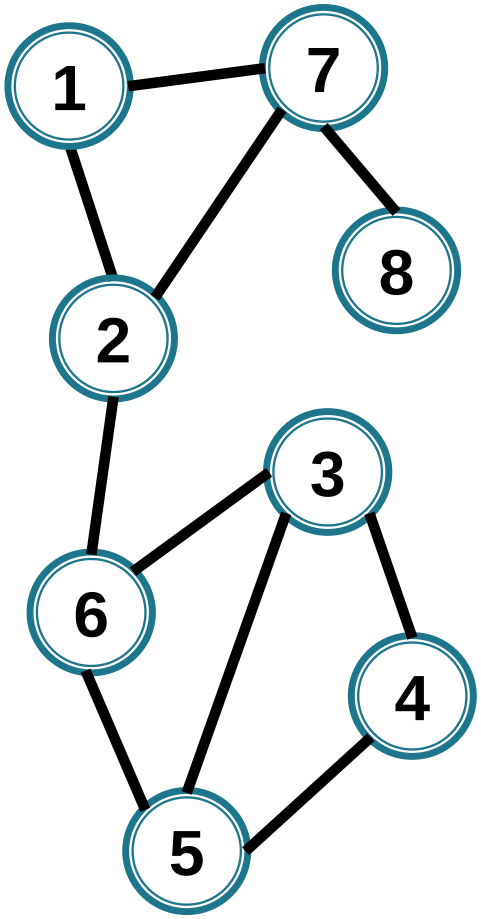
1 2

2 7

1 7

nivel/niv_min





S:

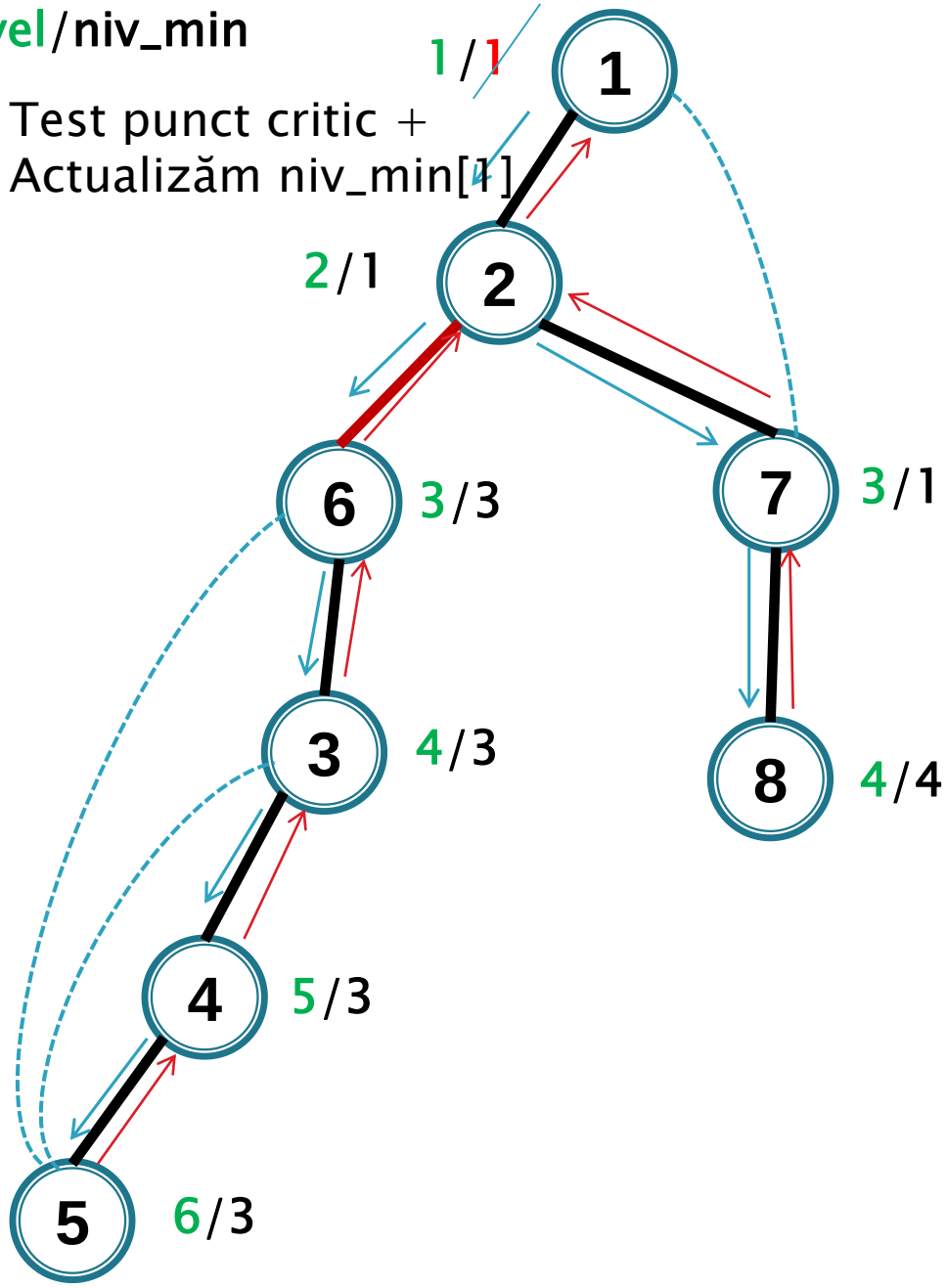
1 2

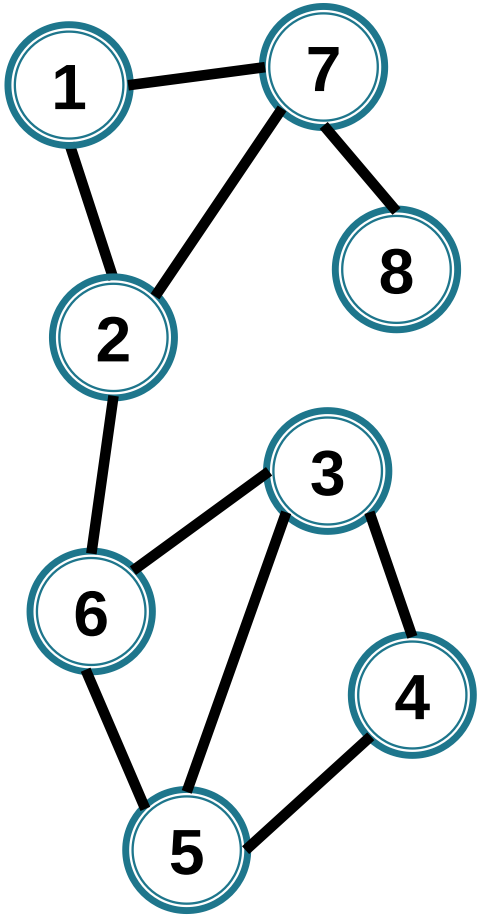
2 7

1 7

nivel/niv_min

Test punct critic +
Actualizăm niv_min[1]





S:

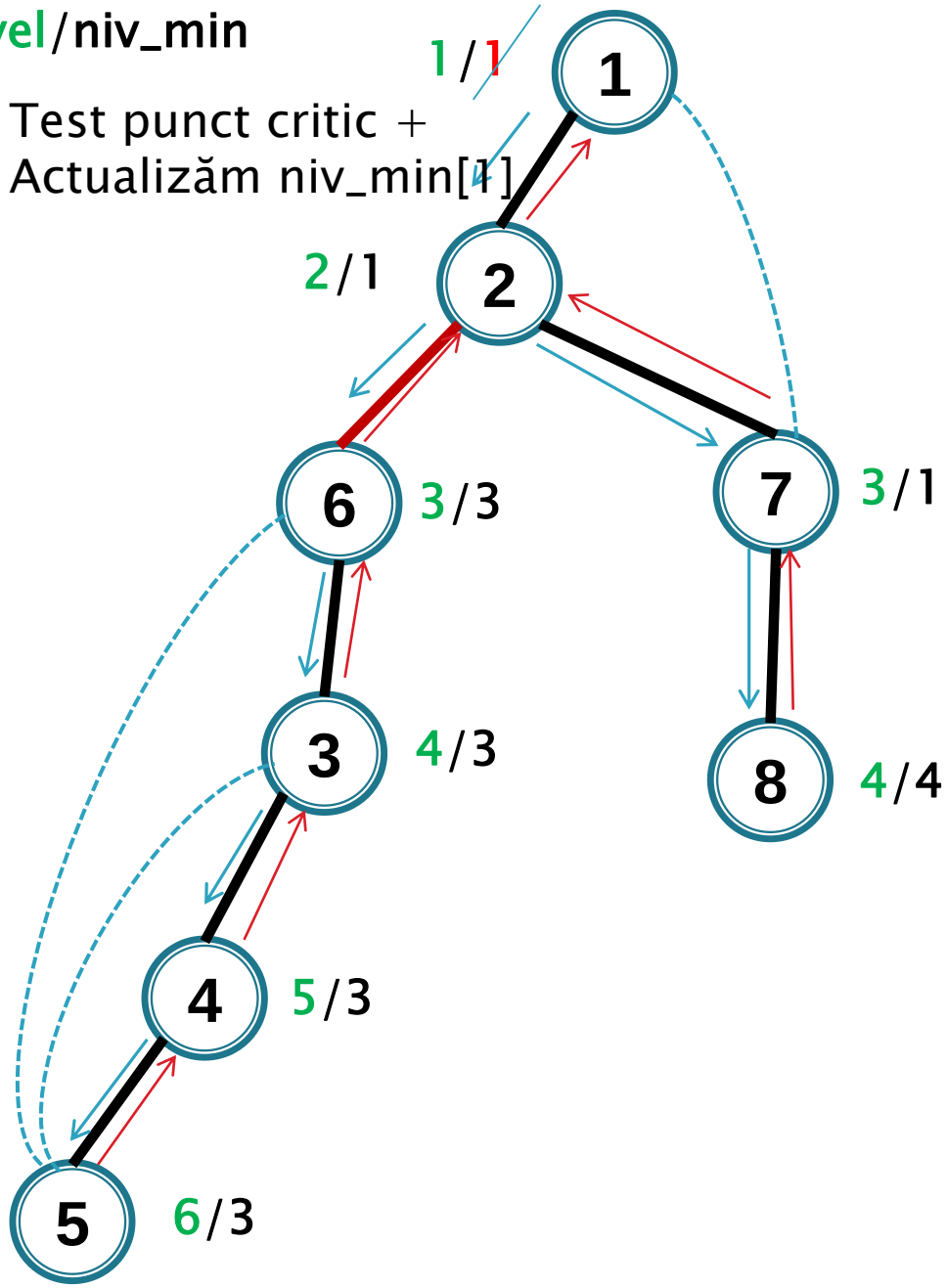
1 2

2 7

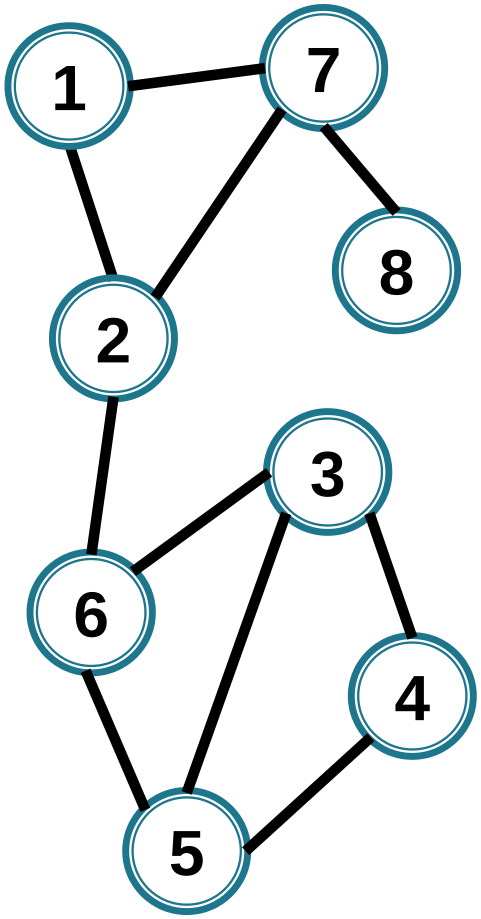
1 7

nivel/niv_min

Test punct critic +
Actualizăm niv_min[1]



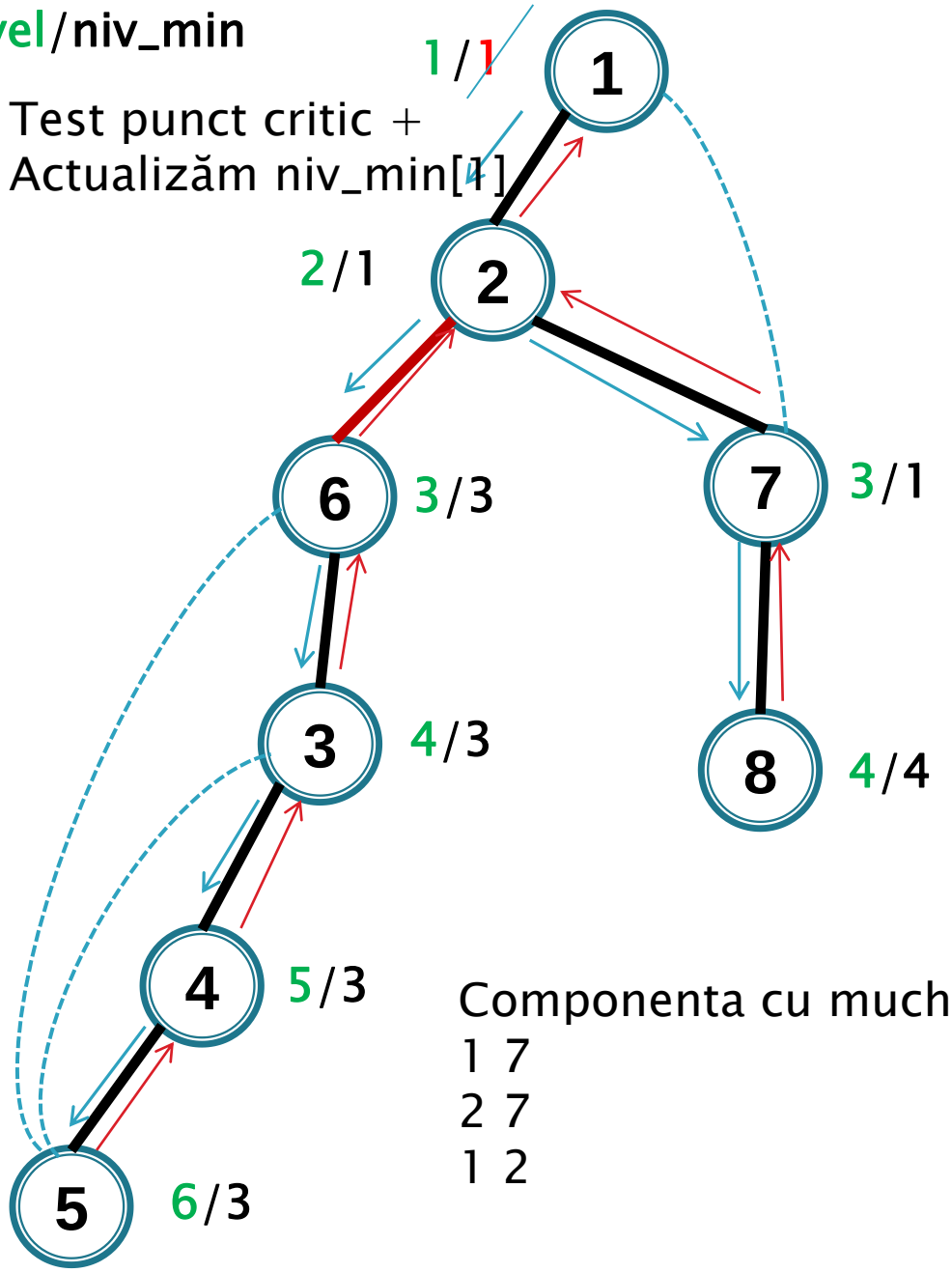
Rădăcina caz particular de test de punct critic – aici nu mai este necesar, se “rupe” o componenta biconexă chiar dacă are un singur fiu



S:
1 2
2 7
1 7

nivel/niv_min

Test punct critic +
Actualizăm niv_min[1]



Componenta cu muchiile
1 7
2 7
1 2

Rădăcina caz particular de test de punct critic – aici nu mai este necesar, se “rupe” o componenta biconexă chiar dacă are un singur fiu