Algorithmique des graphes

David Pichardie

5 Avril 2018

Bilan du CM4

- Fermeture transitive
  - Définition
  - Calcul par parcours de graphe
  - Calcul par multiplication de matrices booléennes
  - Calcul par l'algorithme de Warshall
- Composantes fortement connexes
  - Définitions
  - Algorithme de Kosaraju

Algorithme de Warshall

complexité en **O**(S^3)

```
WARSHALL(A) =
    A* <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            pour tout j ∈ S
                A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
    renvoie A*
```

Algorithme de Warshall

complexité en **O**(S^3)

```
WARSHALL(A) =
A* <- copie(A+Id)
pour tout k ∈ S
    pour tout i ∈ S
        pour tout j ∈ S
            A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
renvoie A*
```

Peux-ton éviter certains calculs ?

Algorithme de Warshall

complexité en **O**(S^3)

```
WARSHALL(A) =
    A* <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            si A*[i,k] alors
                pour tout j ∈ S
                    A*[i,j] <- A*[i,j] || A*[k,j]
    renvoie A*
```

oui
(on évite certaines boucles)

Algorithme de Warshall

complexité en **O**(S^3)

```
WARSHALL(A) =
    A* <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            si A*[i,k] alors
                pour tout j ∈ S
                    si A*[k,j] alors A*[i,j] <- vraie
    renvoie A*
```

(moins d'écriture)

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons A_k[i,j] le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

Quel lien avec A* ?

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons $A_k[i,j]$ le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

$A^* = A_n$

les chemins passent parmi 1,..,n

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons $A_k[i,j]$ le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

Que vaut $A_0[i,j]$?

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons A_k[i,j] le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

A_0[i,j] = vraie si et seulement si (i,j) est un arc ou i=j.

Donc A_0 = A+Id

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons $A_k[i,j]$ le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

Comment exprimer A_k en fonction A_{k-1} ?

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons $A_k[i,j]$ le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

Comment exprimer A_k en fonction A_{k-1} ?

- $A_k[i,j]$ = vraie si $A_{k-1}[i,j]$=vraie

un chemin passe par 1,…,k-1, et donc à fortiori par 1,…,k

- $A_k[i,j]$ = vraie si $A_{k-1}[i,k]$=vraie et $A_{k-1}[k,j]$=vraie

un chemin de i à k passe par 1,…,k-1

un chemin de k à j passe par 1,…,k-1

Donc $A_k[i,j] = A_{k-1}[i,j] \,||\, (A_{k-1}[i,k] \,\&\&\, A_{k-1}[k,j])$

Algorithme de Warshall

Correction

Supposons S=[1,n] et notons $A_k[i,j]$ le booléen qui vaut vraie si et seulement si il existe un chemin simple de i à j dont les sommets intermédiaires (autres que i et j) sont dans [1,k], pour k=0,…,n

Donc

$A_0=A+Id$

$A_k[i,j] = A_{k-1}[i,j] \,||\, (A_{k-1}[i,k] \,\&\&\, A_{k-1}[k,j])$ si $k>0$

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
    A <- matrice((n+1) x n x n)
    A[0] <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            pour tout j ∈ S
                A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
    renvoie A[n]
```

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
    A <- matrice(n x n x n)
    A[0] <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            pour tout j ∈ S
                A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
    renvoie A[n]
```

Peux-on gommer les indices ?

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
    A <- matrice(n x n x n)
    A* <- copie(A+Id)
    A[0] <- copie(A+Id)
    pour tout k ∈ S
        pour tout i ∈ S
            pour tout j ∈ S
                A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
                A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
    renvoie A*
```

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
A <- matrice(n x n x n)
A* <- copie(A+Id)
A[0] <- copie(A+Id)
pour tout k ∈ S
   pour tout i ∈ S
      pour tout j ∈ S
         A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
         A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
renvoie A*
```

Montrons que A[k][i,j] = A*[i,j] à chaque passage

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
   A <- matrice(n x n x n)
   A* <- copie(A+Id)
   A[0] <- copie(A+Id)
   pour tout k ∈ S
      pour tout i ∈ S
         pour tout j ∈ S
            A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
            A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
   renvoie A*
```

quand a eu lieu la dernière modification ?

quand a eu lieu la dernière modification ?

quand a eu lieu la dernière modification ?

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
   A <- matrice(n x n x n)
   A* <- copie(A+Id)
   A[0] <- copie(A+Id)
   pour tout k ∈ S
      pour tout i ∈ S
         pour tout j ∈ S
            A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
            A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
   renvoie A*
```

A[k-1][i,j]

A[k-1][i,k] ou A[k][i,k]

A[k-1][k,j] ou A[k][k,j]

Algorithme de Warshall

Correction

Mais

$$A[k-1][i,k] = A[k][i,k]$$

car les chemins simples de i à k passant par 1,…,k ne passent que par 1,…,k-1.

De même

$$A[k-1][k,j] = A[k][k,j]$$

Algorithme de Warshall

Correction

Un premier Warshall correct :

```
WARSHALL(A) =
   A <- matrice(n x n x n)
   A* <- copie(A+Id)
   A[0] <- copie(A+Id)
   pour tout k ∈ S
      pour tout i ∈ S
         pour tout j ∈ S
            A*[i,j] <- A*[i,j] || (A*[i,k] && A*[k,j])
            A[k][i,j] <- A[k-1][i,j] || (A[k-1][i,k] && A[k-1][k,j])
   renvoie A*
```

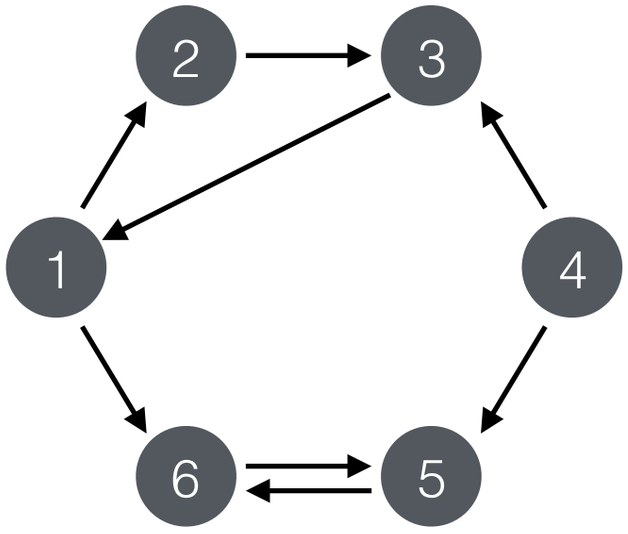
A[k-1][i,j]

A[k-1][i,k]

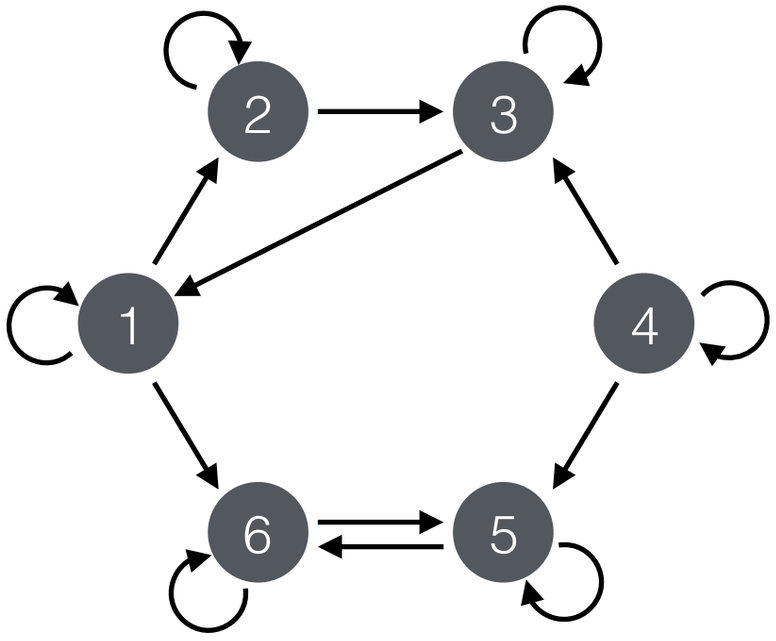
A[k-1][k,j]

Donc A[k][i,j]=A*[i][j] ici

Exemple

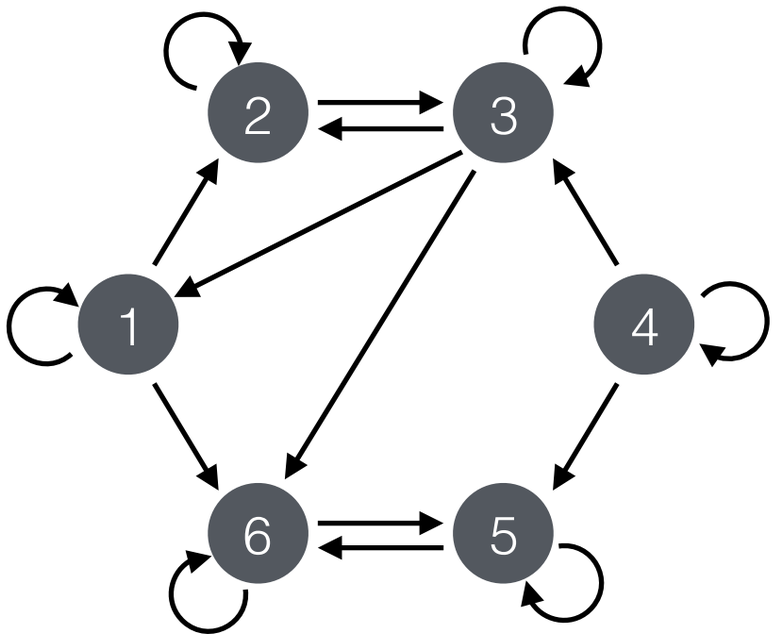


Exemple



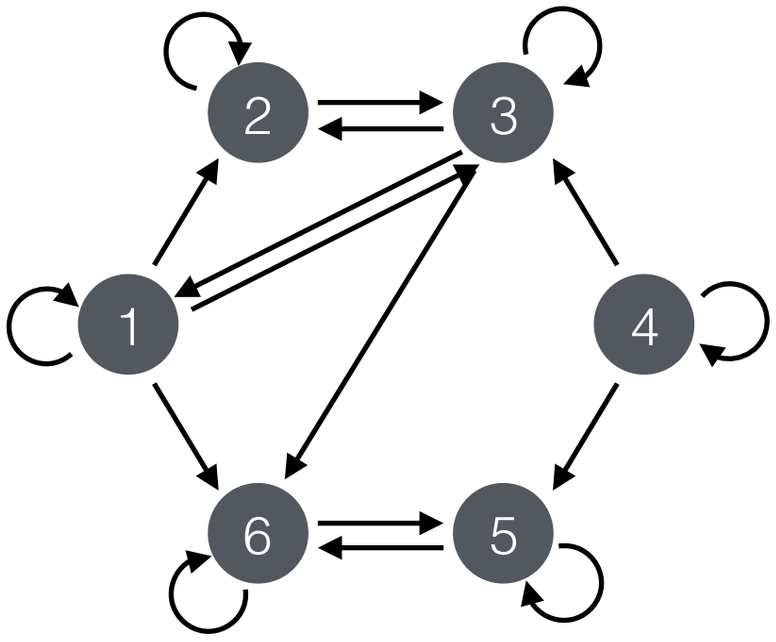
k	arc(s) ajouté(s)
0	
1	
2	
3	
4	
5	
6	

Exemple



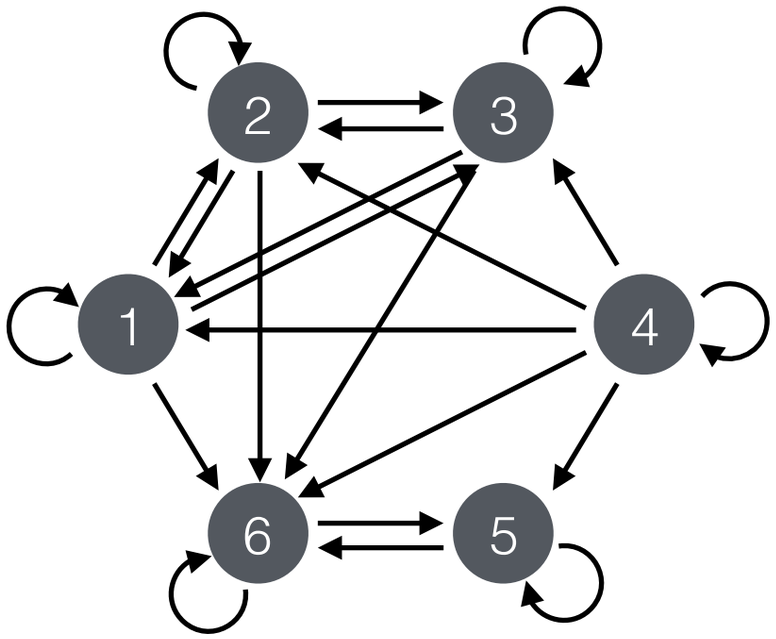
k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	
3	
4	
5	
6	

Exemple



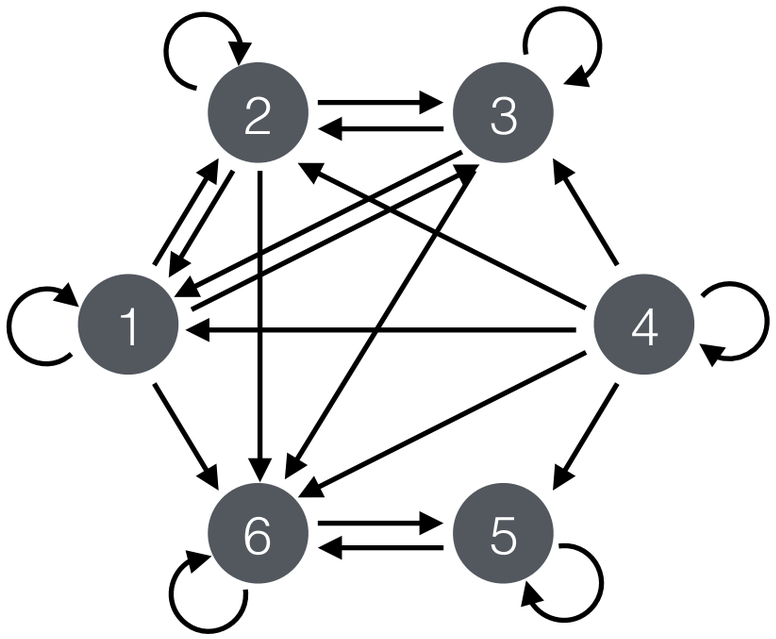
k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	(1,3)
3	
4	
5	
6	

Exemple



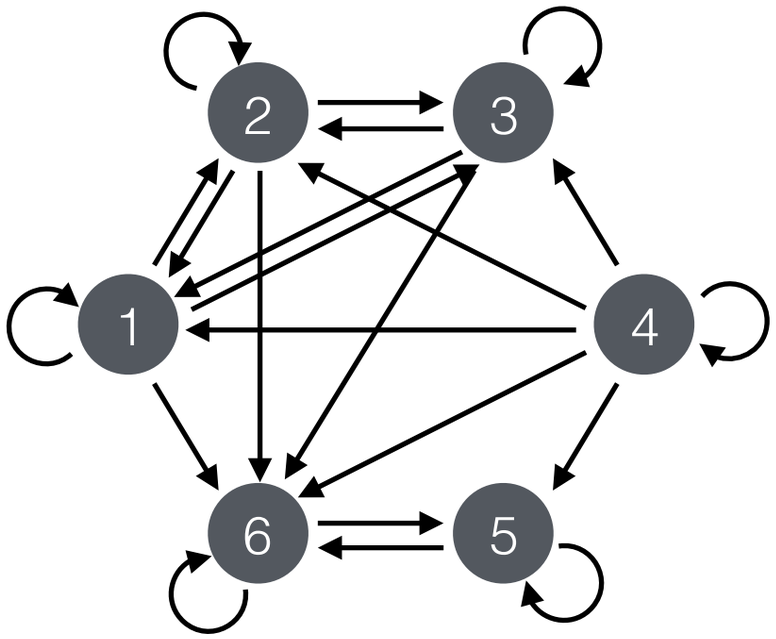
k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	(1,3)
3	(2,1); (4,2); (4,1); (4,6); (2,6)
4	
5	
6	

Exemple



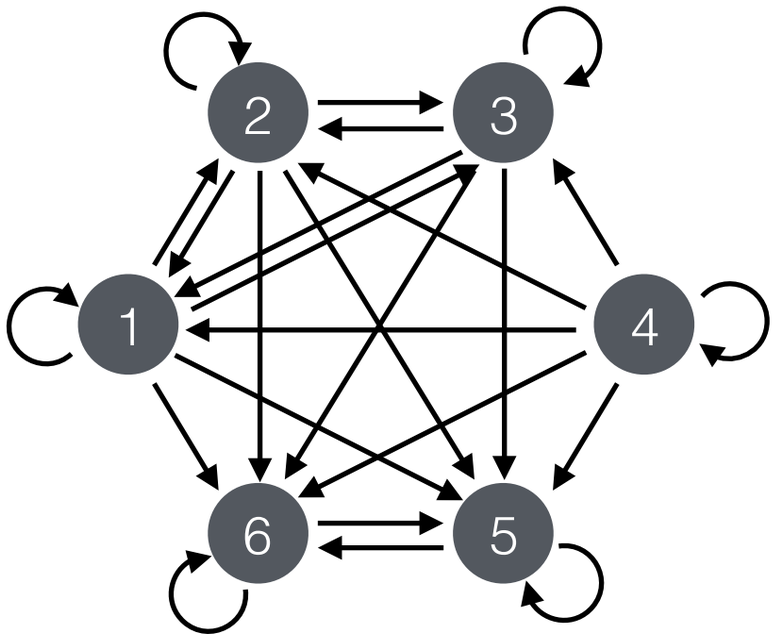
k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	(1,3)
3	(2,1); (4,2); (4,1); (4,6); (2,6)
4	-
5	
6	

Exemple



k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	(1,3)
3	(2,1); (4,2); (4,1); (4,6); (2,6)
4	-
5	-
6	

Exemple



k	arc(s) ajouté(s)
0	
1	(3,2); (3,6)
2	(1,3)
3	(2,1); (4,2); (4,1); (4,6); (2,6)
4	-
5	-
6	(1,5); (2,5); (3,5)

Composantes fortement connexes

Algorithme de Kosaraju

1. parcours en profondeur sur G pour calculer FIN (ordre suffixe)

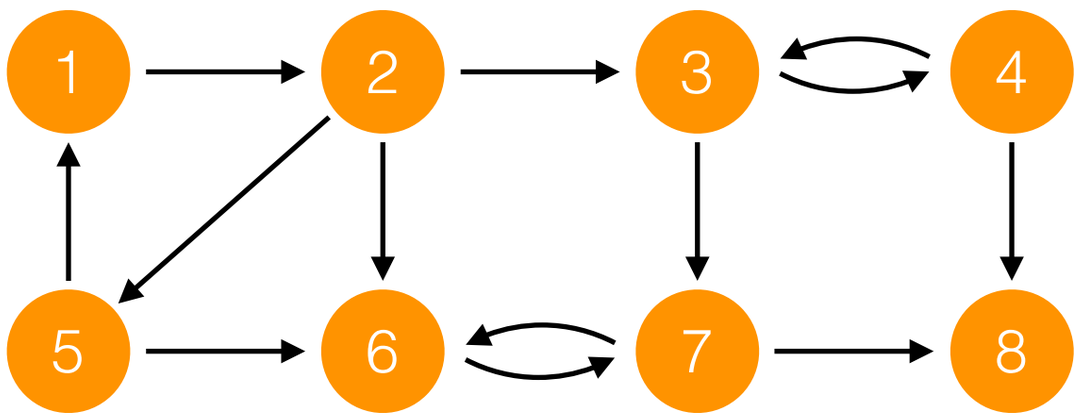
2. calcul de l'inverse G' de G

3. parcours en profondeur sur G', mais en itérant la boucle principale par ordre de FIN décroissant

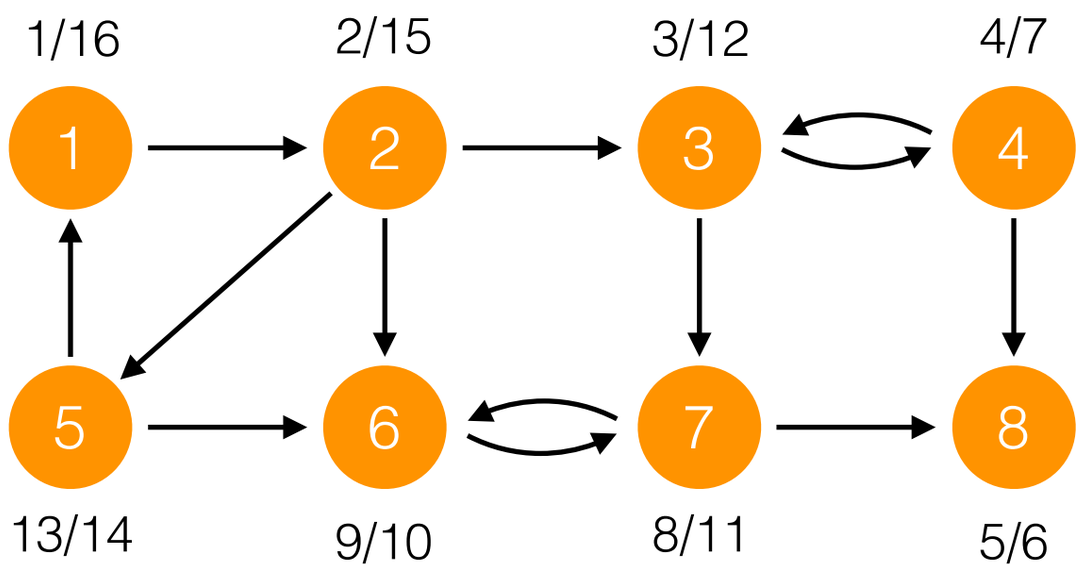
4. les composantes fortement connexes sont les arbres de la forêt du 2e parcours

coût global **O**(A+S)

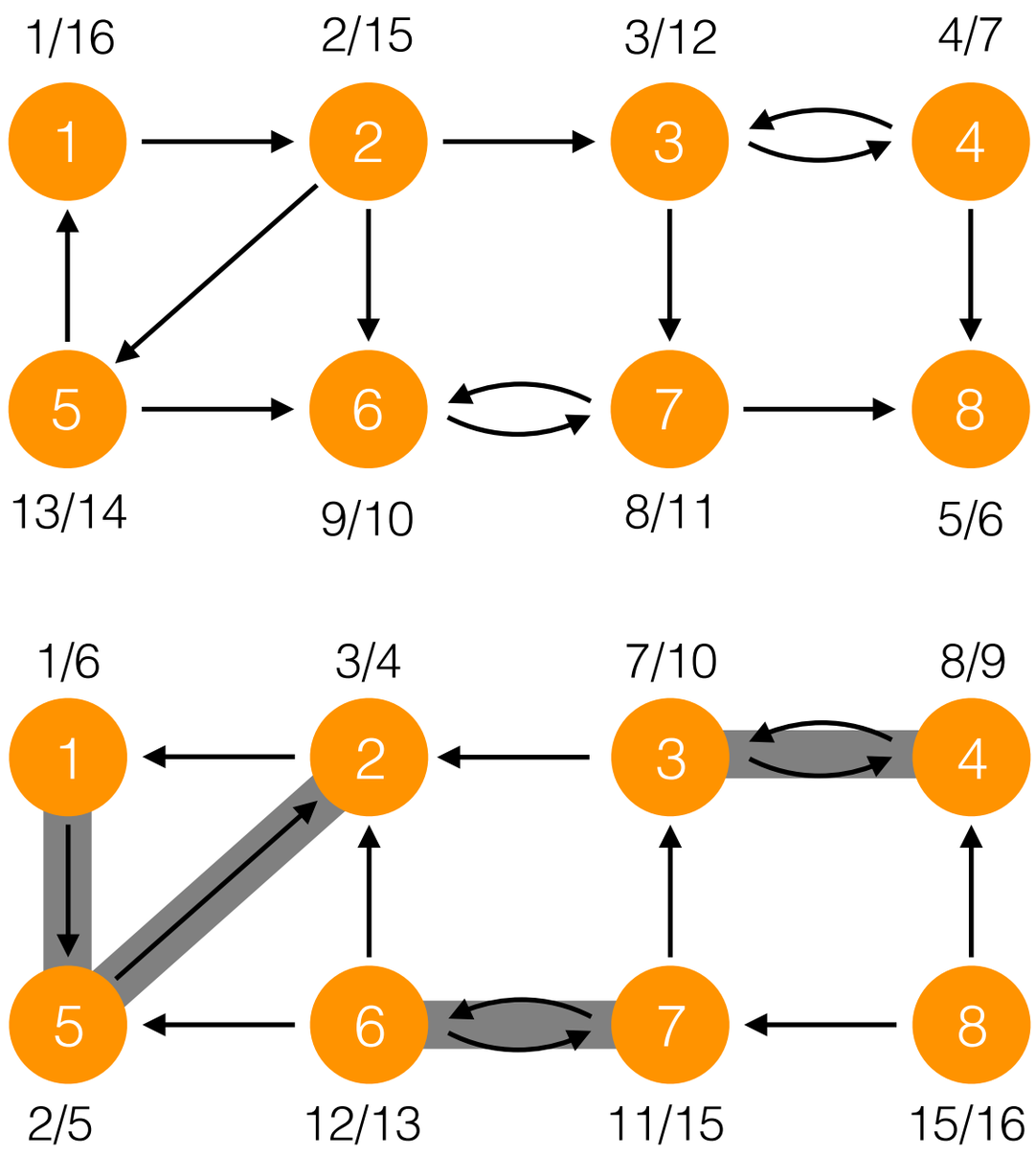
Exemple



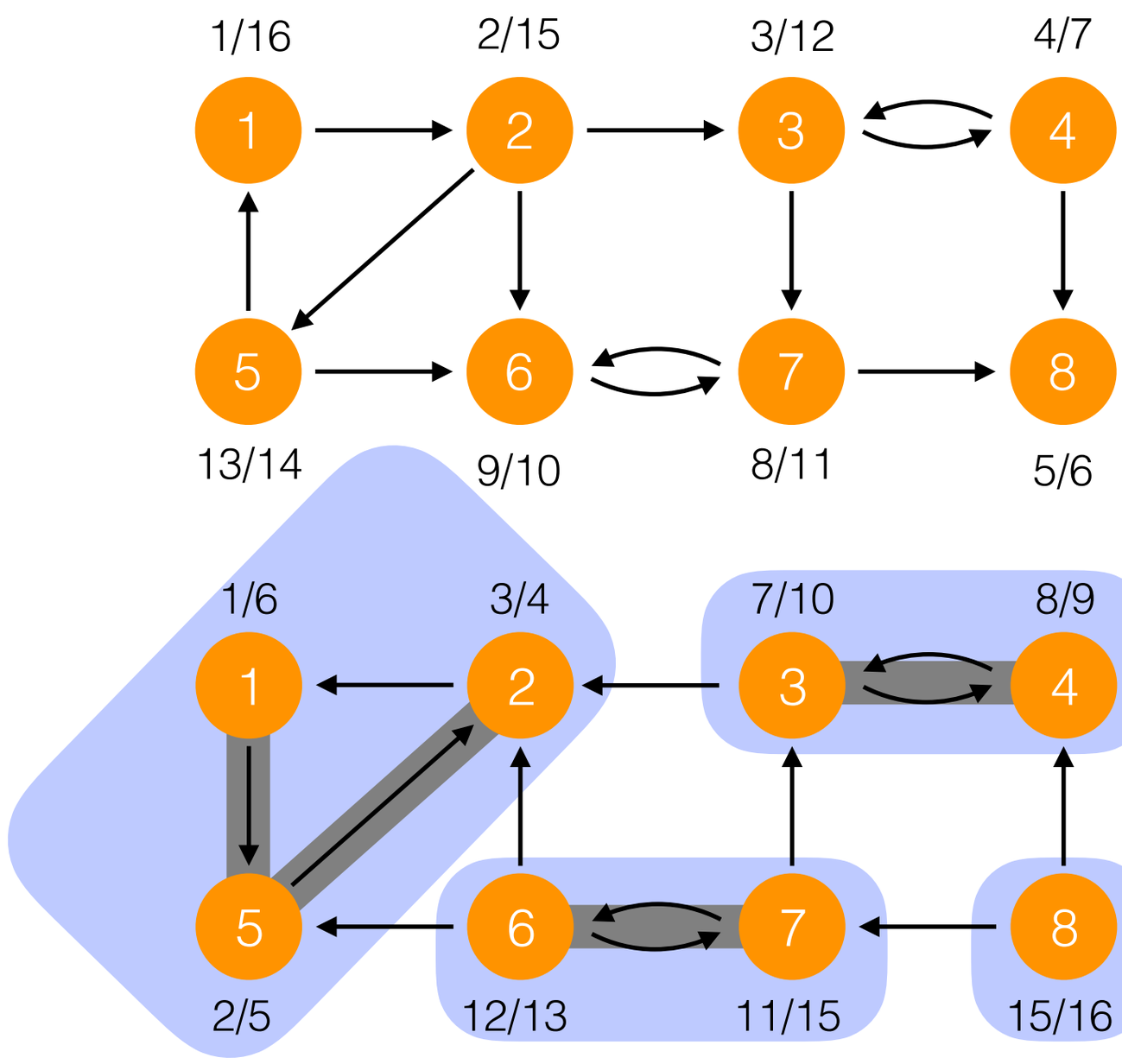
Exemple



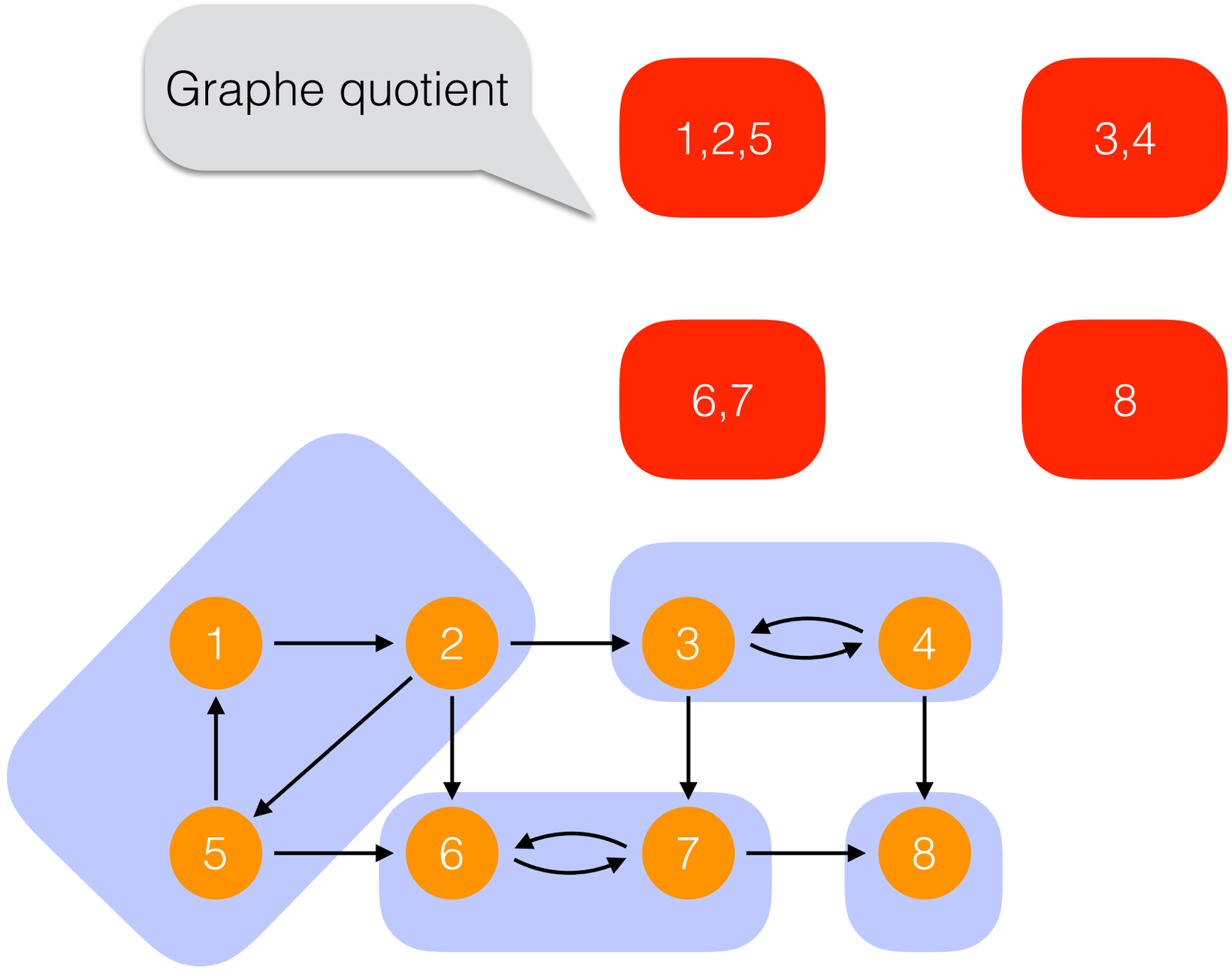
Exemple


1/16
2/15
3/12
4/7
1
2
3
4
5
6
7
8
13/14
9/10
8/11
5/6
1/6
3/4
7/10
8/9
1
2
3
4
5
6
7
8
2/5
12/13
11/15
15/16

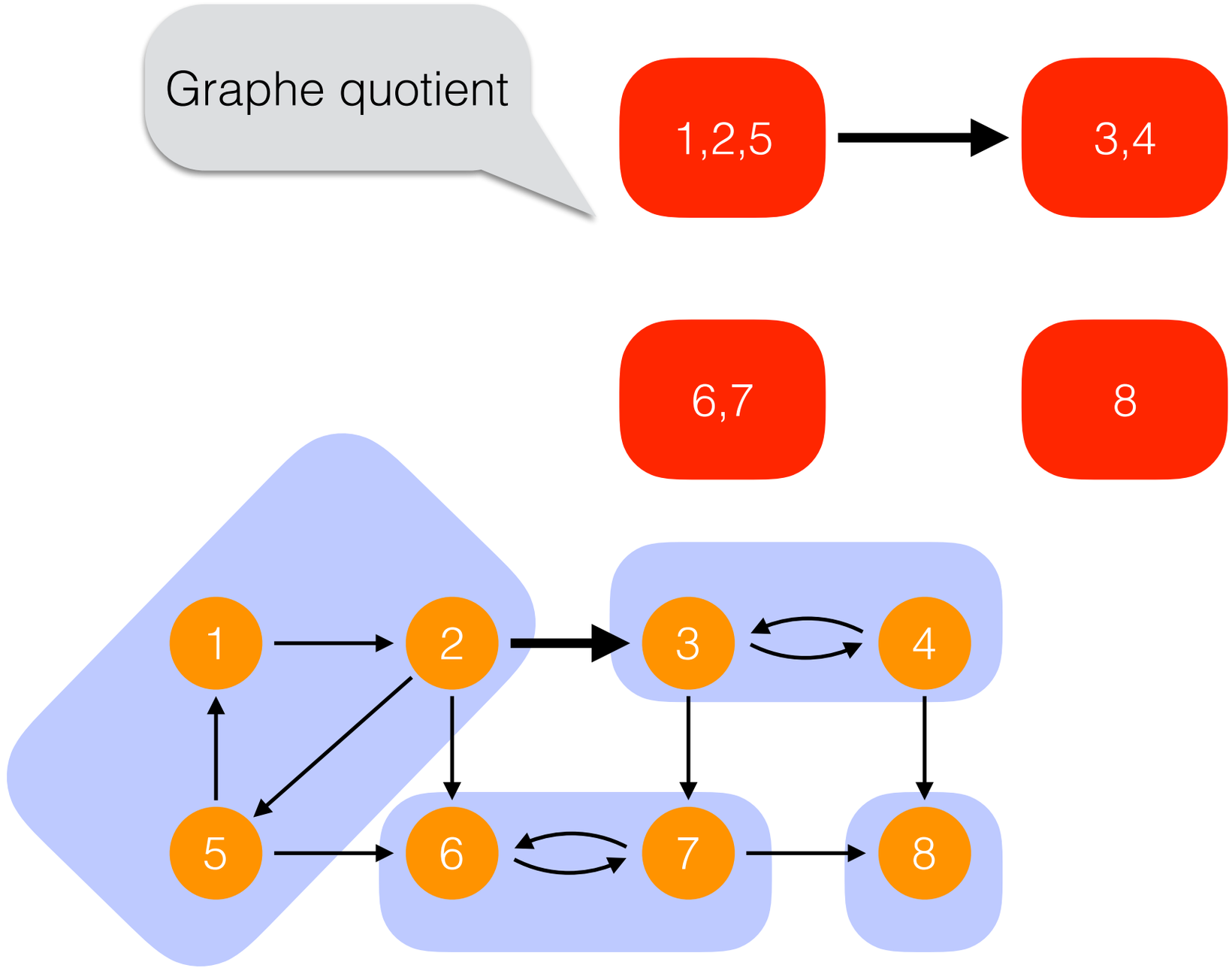
Exemple


1/16
2/15
3/12
4/7
1
2
3
4
5
6
7
8
13/14
9/10
8/11
5/6
1/6
3/4
7/10
8/9
1
2
3
4
5
6
7
8
2/5
12/13
11/15
15/16

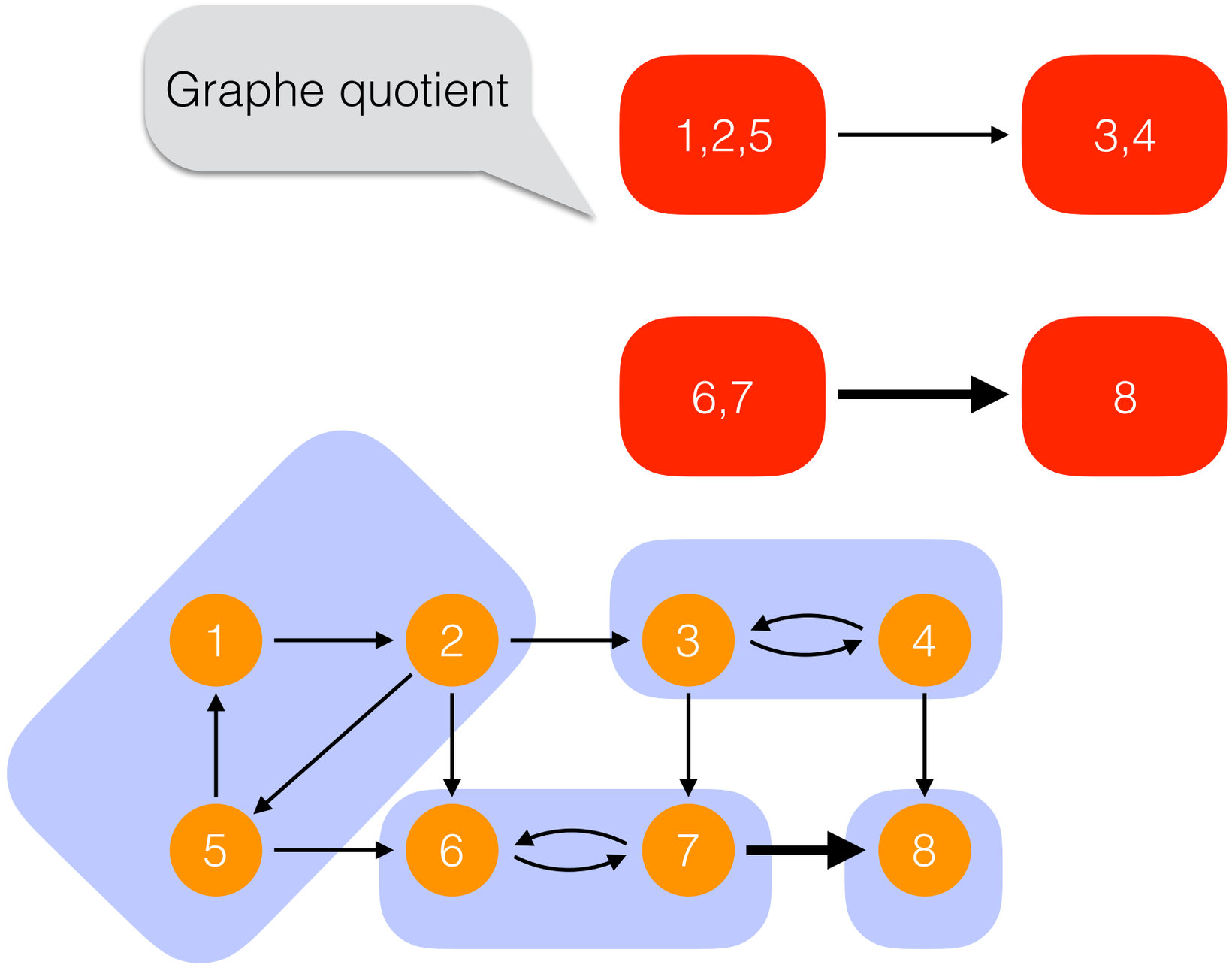
Exemple



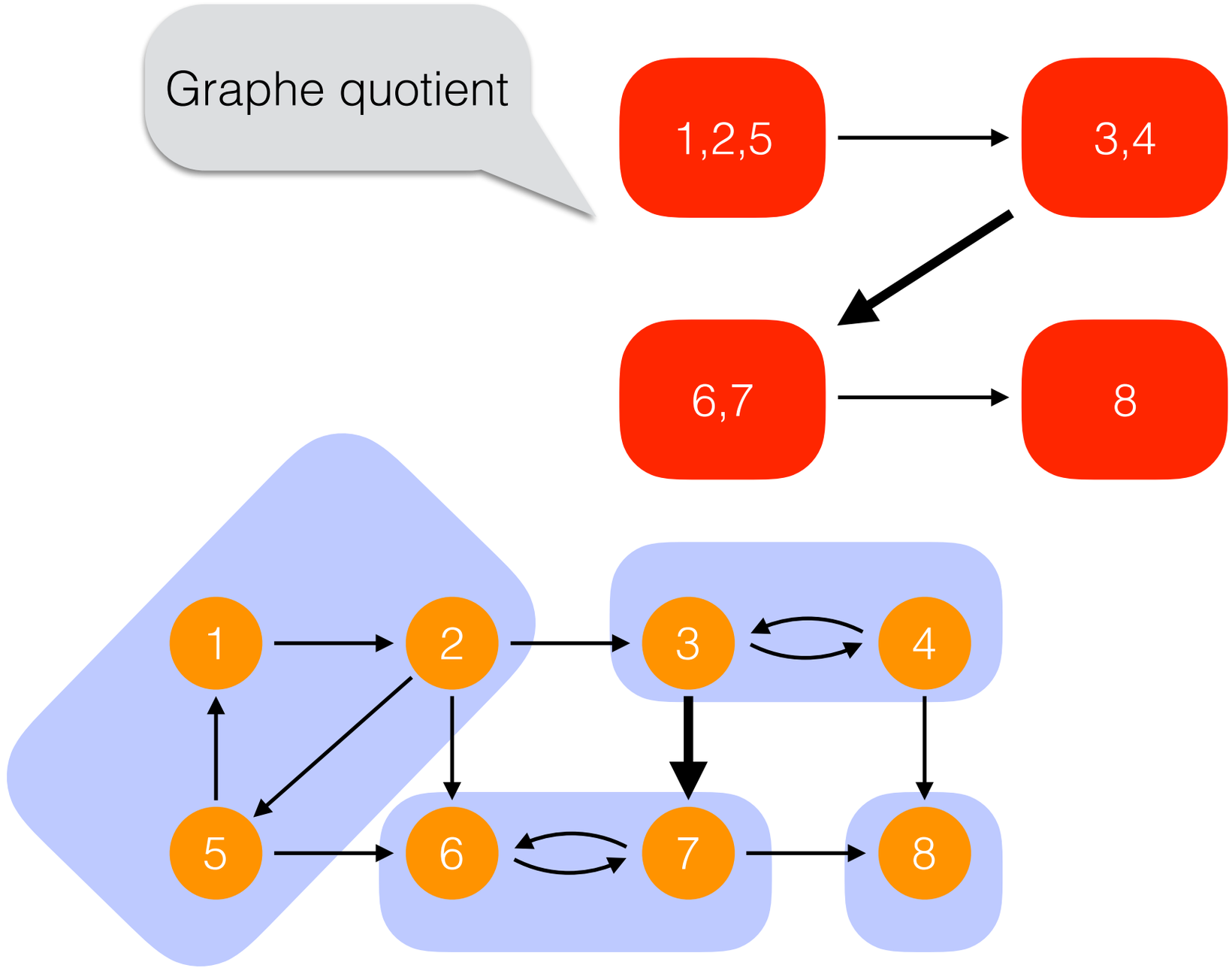
Exemple



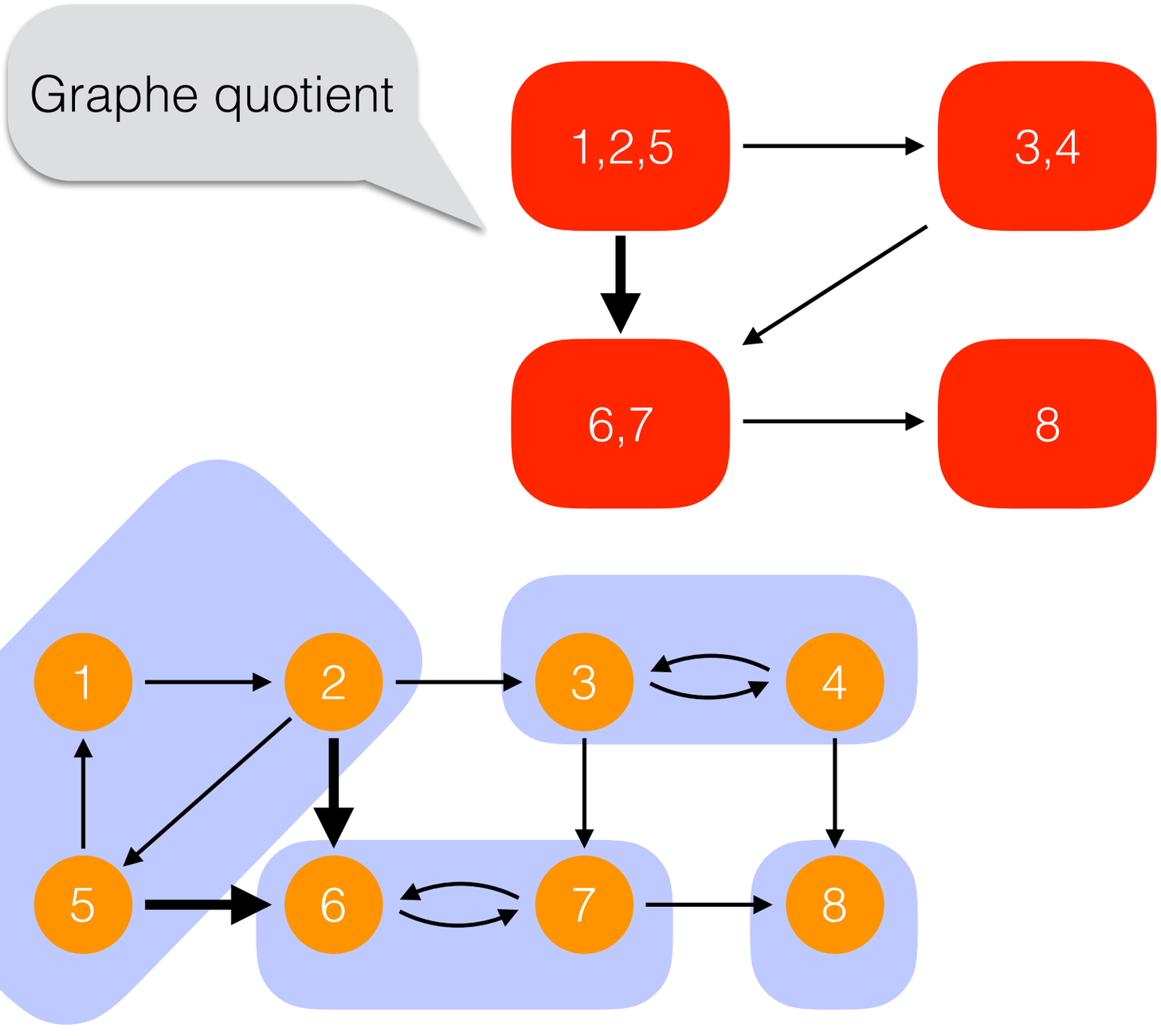
Exemple



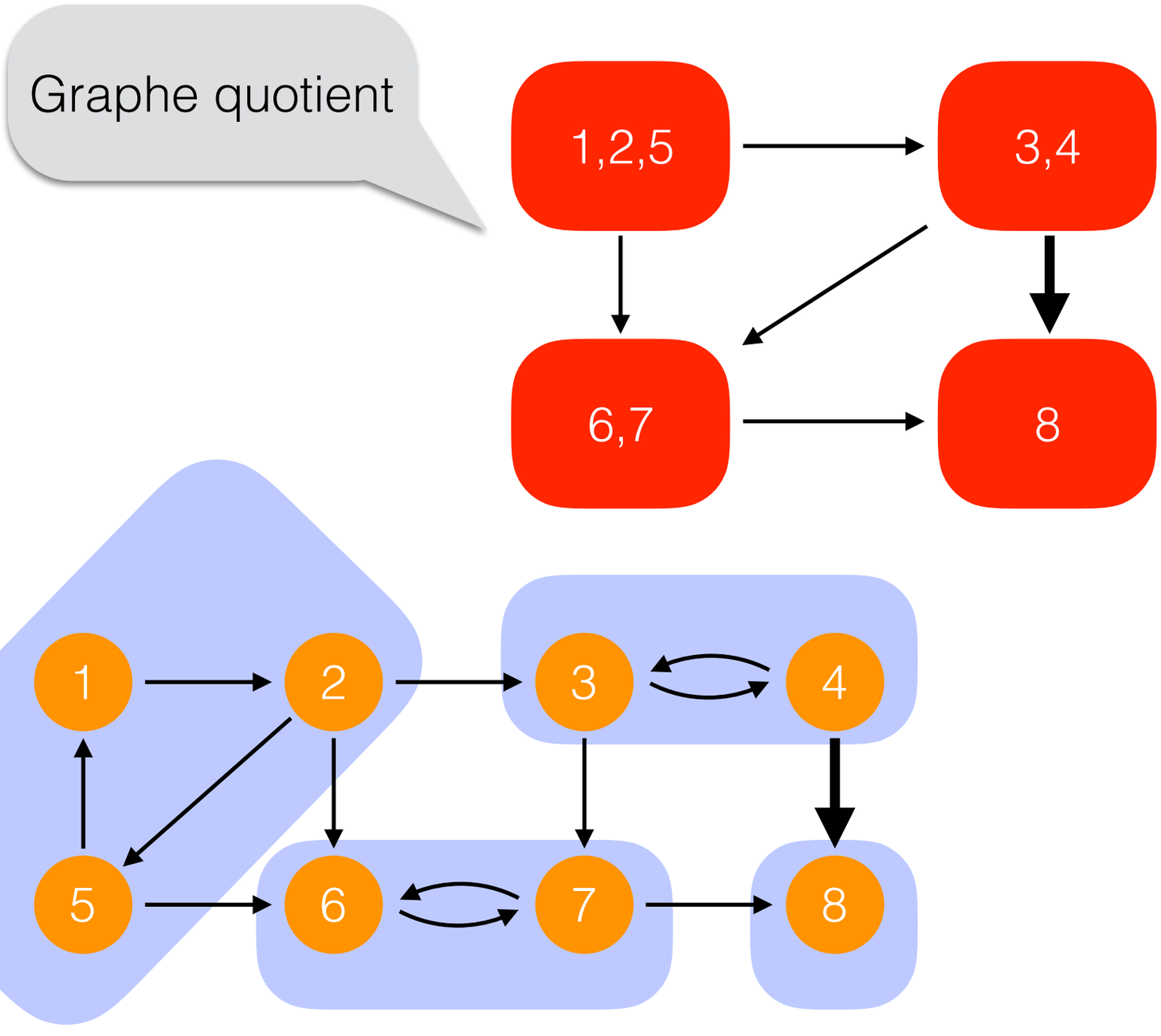
Exemple



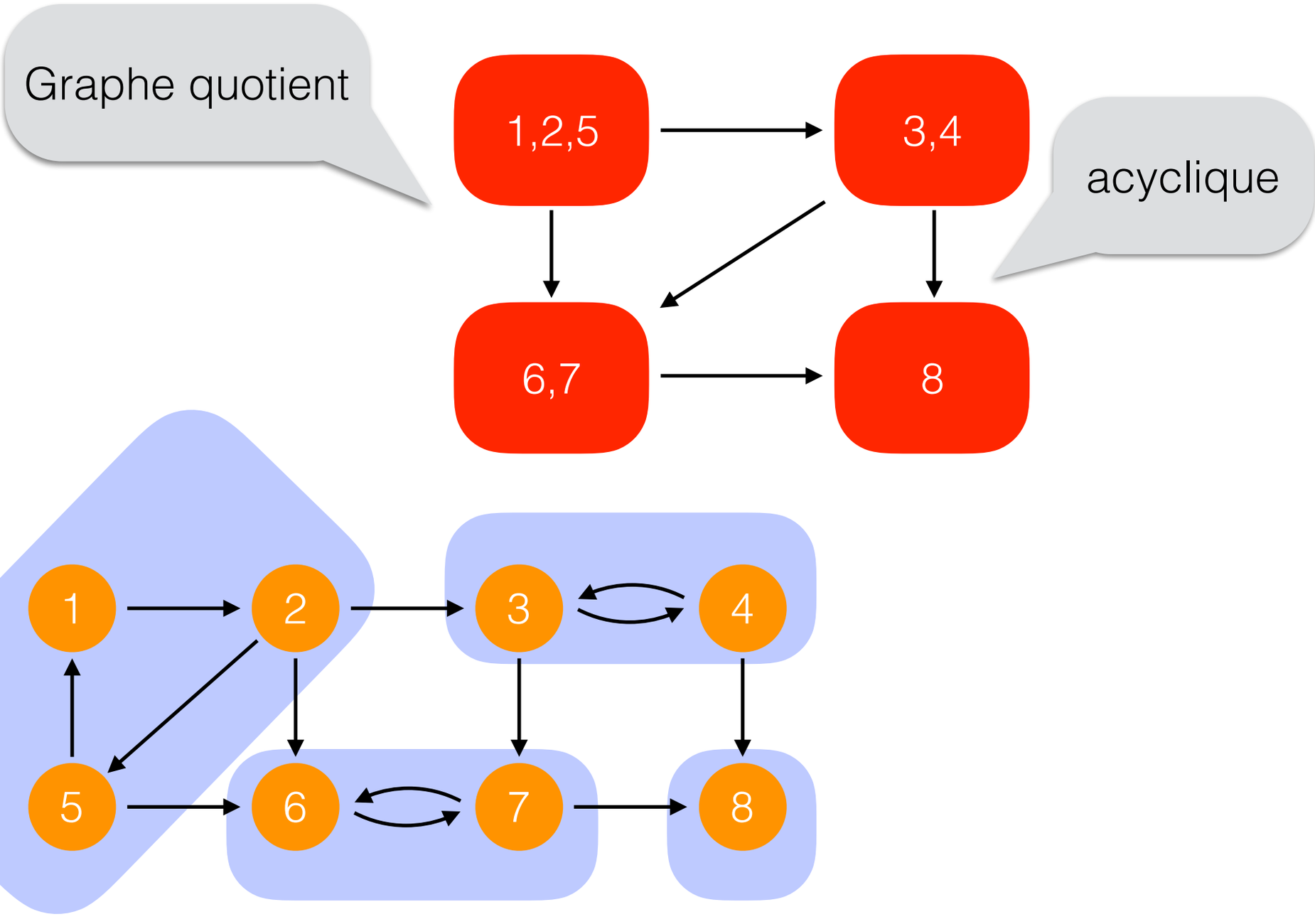
Exemple



Exemple



Exemple



Question

Comment réduire le coût du calcul de la fermeture transitive grâce aux composantes fortement connexes ?

Fermeture transitive par passage au quotient

1. Soit G=(S,A) un graphe orienté

coût linéaire **O**(S+A)

2. On calcule sa table de cfc (un numéro de composante pour chaque sommet). On note S_{cfc} l'ensemble des cfc.

coût linéaire **O**(S+A)

3. On calcule le graphe quotient $G_q=(S_{cfc},A_q)$

coût **O**(S_{cfc}^3)

4. On calcule la fermeture transitive $G_q^*=(S_{cfc},A_q^*)$ de G_q

coût **O**(1)

5. Pour tout $(i,j) \in S^2$,
   $(i,j) \in A^*$ si et seulement si $(cfc[i],cfc[j]) \in A_q^*$

Calcul de composantes fortement connexes

Algorithme de Tarjan

1. un seul parcours en profondeur mais en utilisant une pile auxiliaire pour calculer les composantes

2. la fonction récursive VISITE renvoie un numéro de sommet

3. un peu magique…

Algorithme de Tarjan

Procédure principale

```
CFC()=
date ⟵ 0
DEBUT ⟵ [0, ..., 0]
P ⟵ Pile_Vide()
Ncfc ⟵ 0
CFC ⟵ [0, ..., 0]
pour tout i∈S
  si DEBUT[i]=0 alors TARJAN(i)
renvoie CFC
```

coût global **O**(A+S) mais avec un seul parcours !

dates de DEBUT classiques

pile auxiliaire

nombre de cfc

numéro unique de composante pour chaque sommet

non VU

Algorithme de Tarjan

Procédure récursive

```
TARJAN(i)=
  date ⟵ date + 1
  DEBUT[i] ⟵ date
  min ⟵ DEBUT[i]
  Empile(P,i)
  pour tout j∈Adj[i]
    si DEBUT[j]=0 alors min ⟵ MIN(min,TARJAN(j))
    sinon si CFC[j]=0 alors min ⟵ MIN(min,DEBUT[j])
  si min=DEBUT[i] alors
    Ncfc ⟵ Ncfc + 1
    répète
      k ⟵ Depile(P)
      CFC[k] ⟵ Ncfc
    tant que k<>i
  renvoie min
```

variable locale

on va tenter de faire diminuer min

si min n'a pas diminué, on a trouvé une cfc

les éléments de la cfc sont au dessus de i dans la pile

Algorithme de Tarjan : visualisation

- on numérote les sommets par l'ordre de début de visite dans le parcours en profondeur

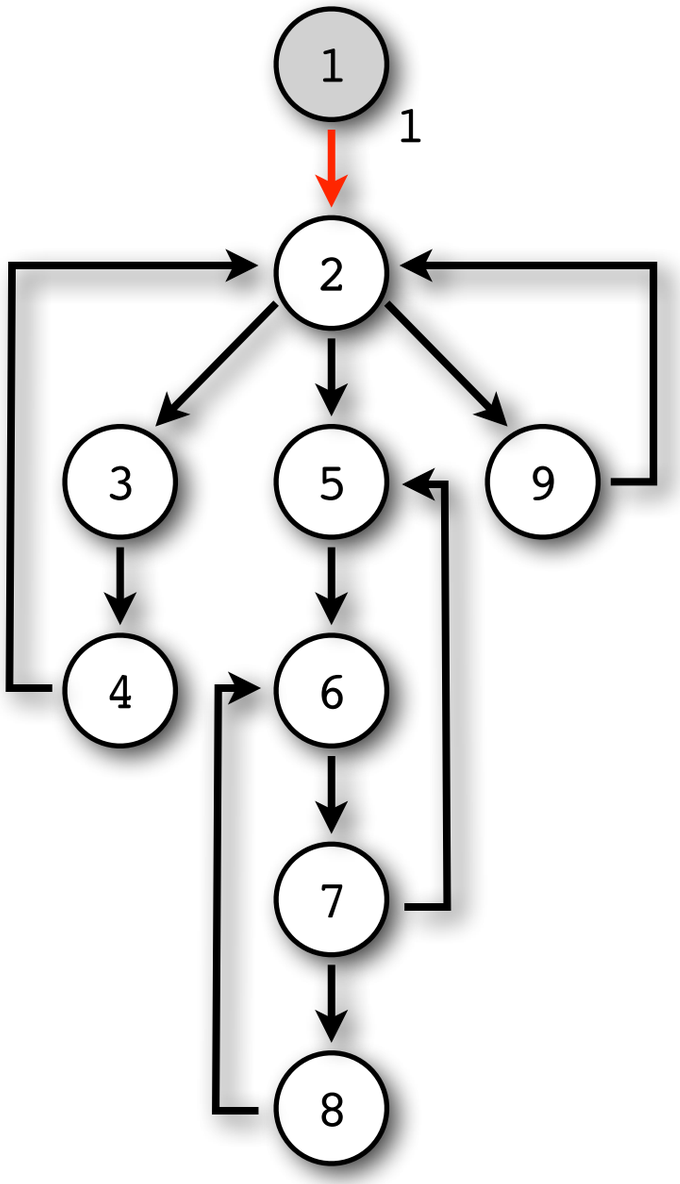
- on grise un sommet dont la visite a commencé

- on noircit un sommet dont la visite est terminée

- on utilise des lassos pour marquer les CFC calculées

- à chaque appel de Tarjan(i), on indique sur le sommet i la valeur courante de min et on indique la valeur de la pile P avant l'appel

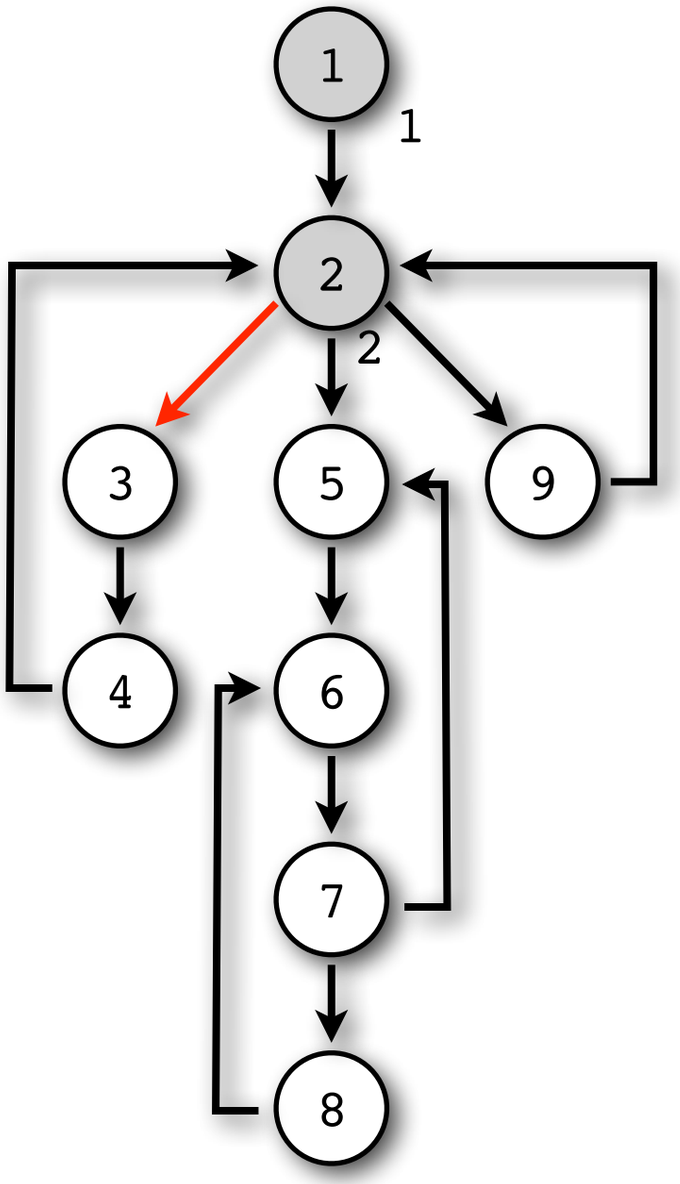
Tarjan(1)

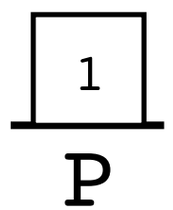


(vide)

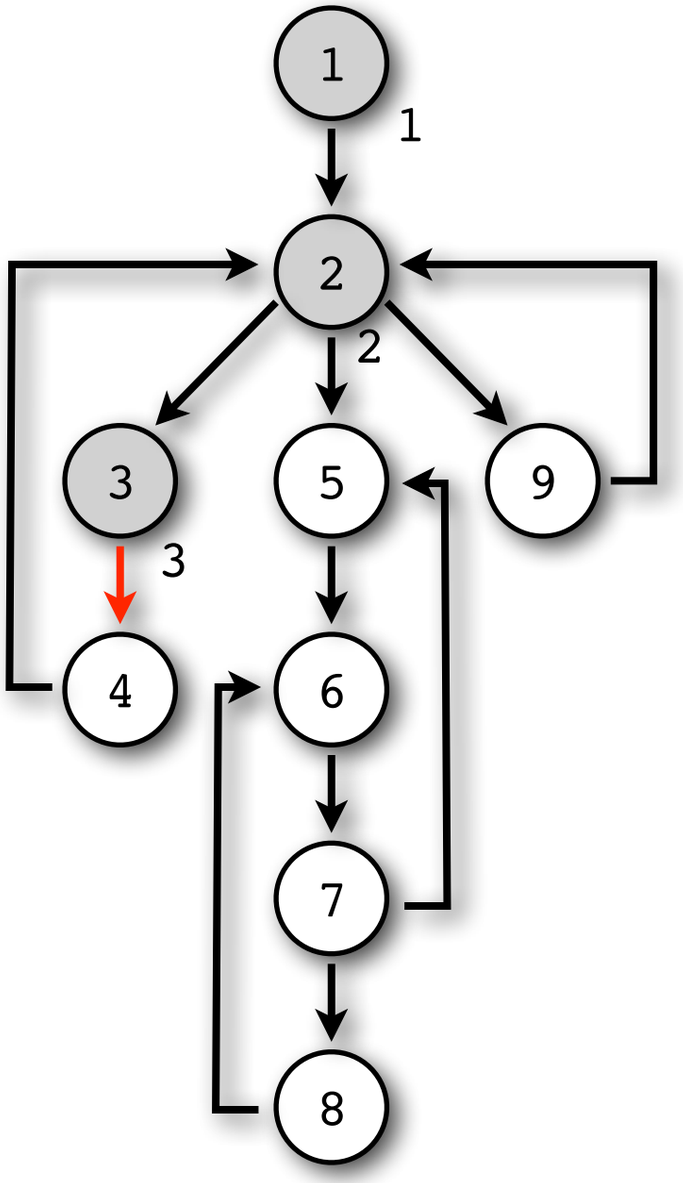
P

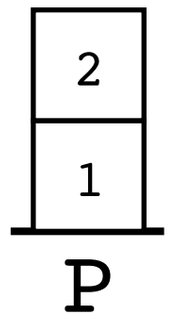
Tarjan(2)



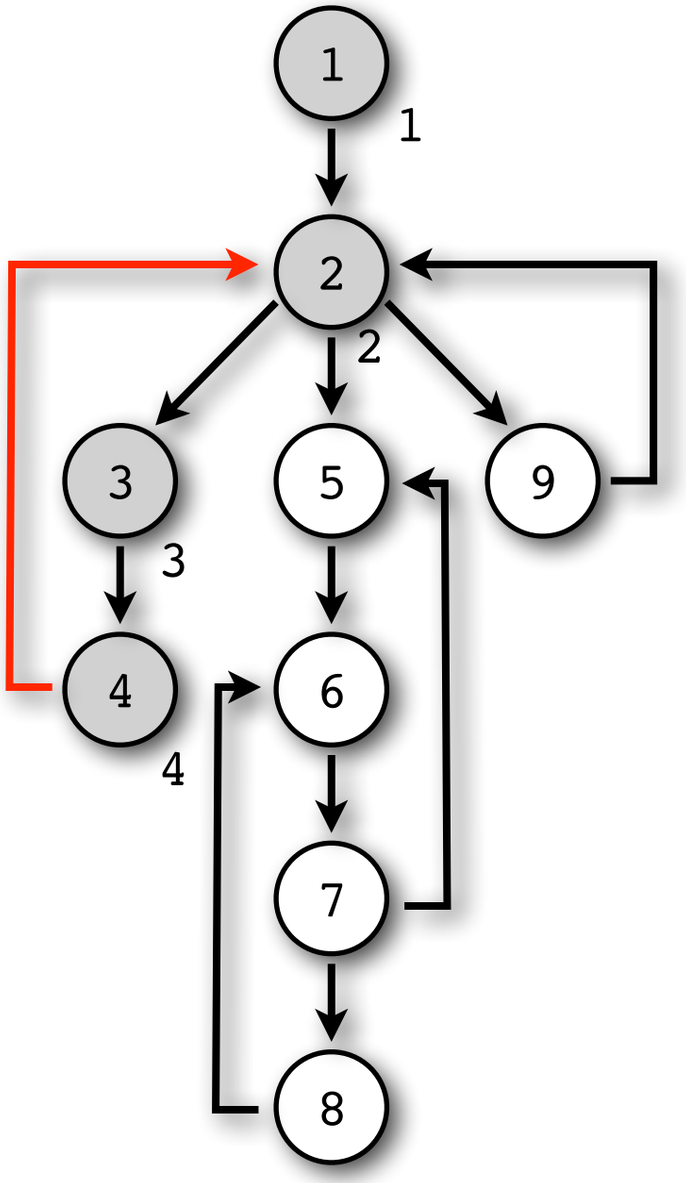


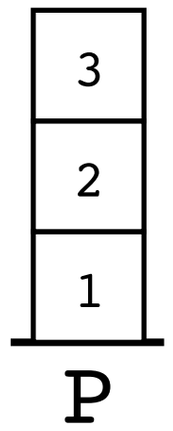
Tarjan(3)



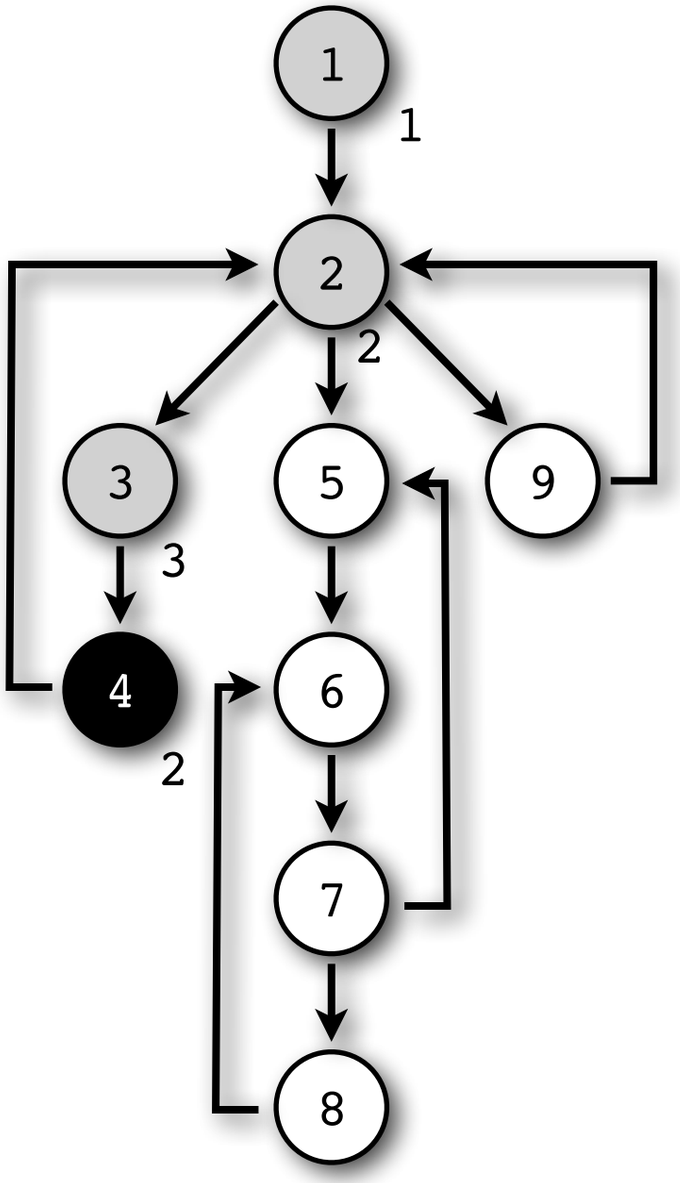


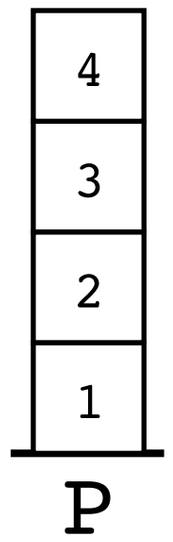
Tarjan(4)



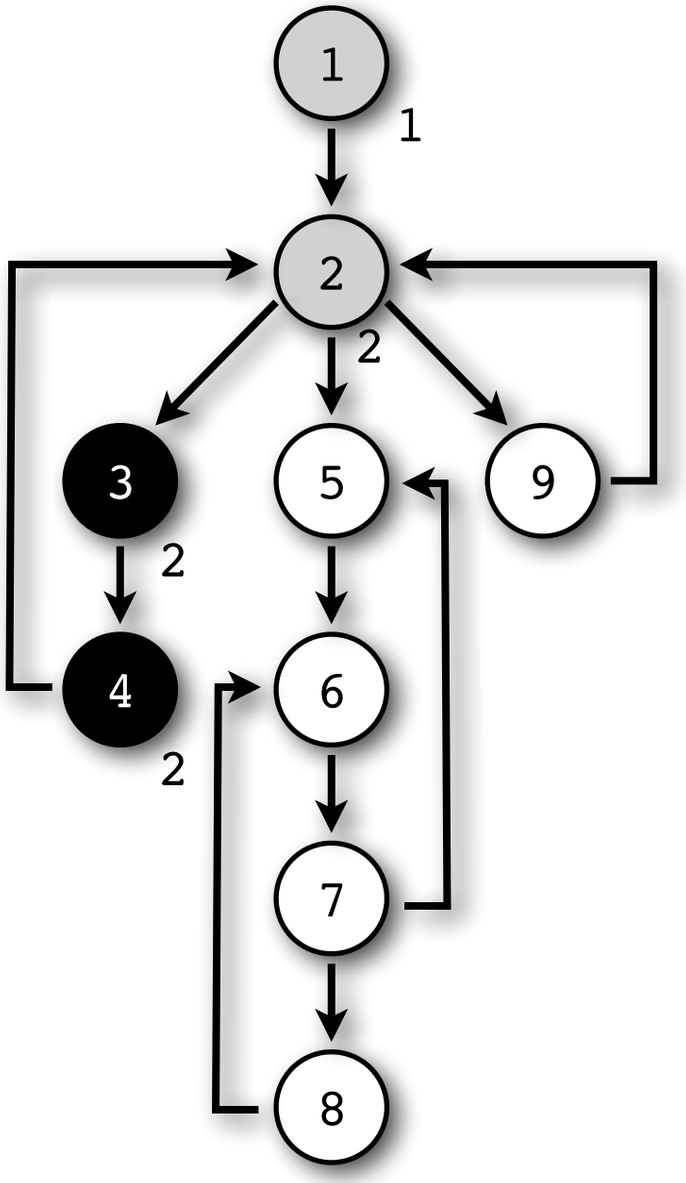


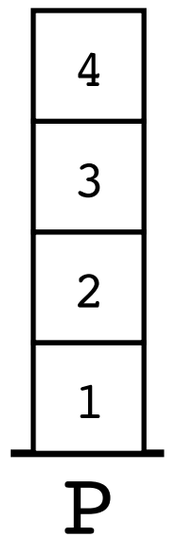
Tarjan(4)



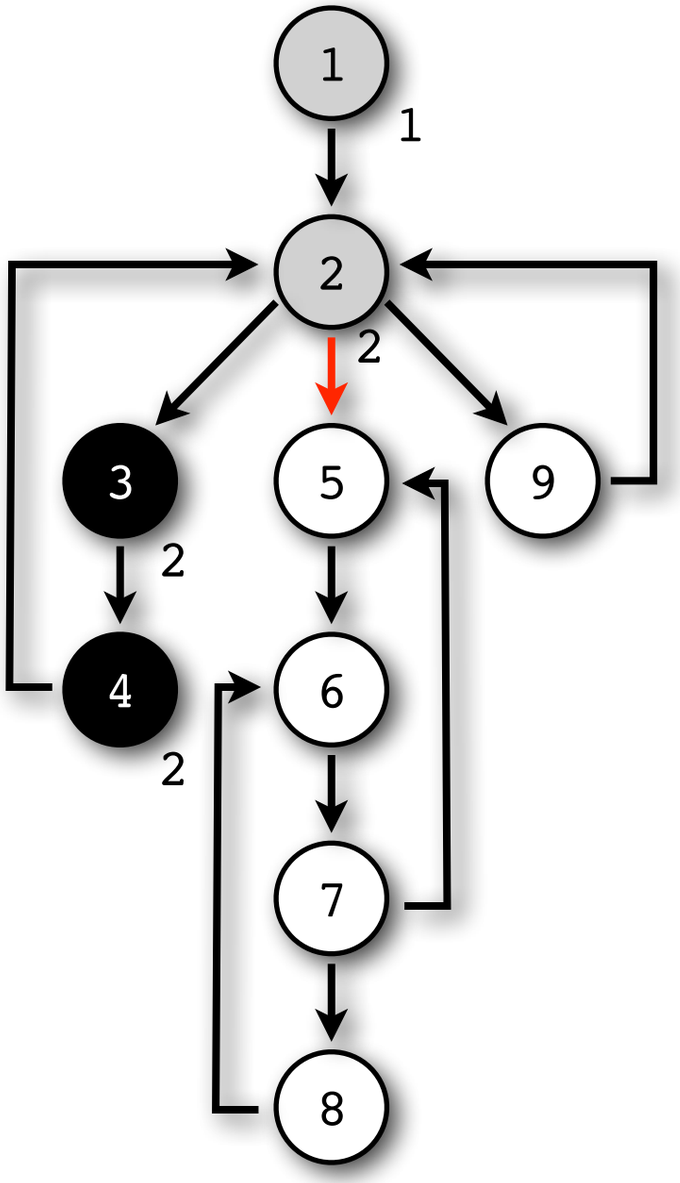


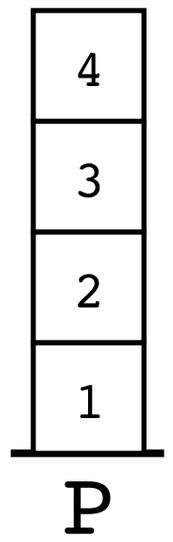
Tarjan(3)



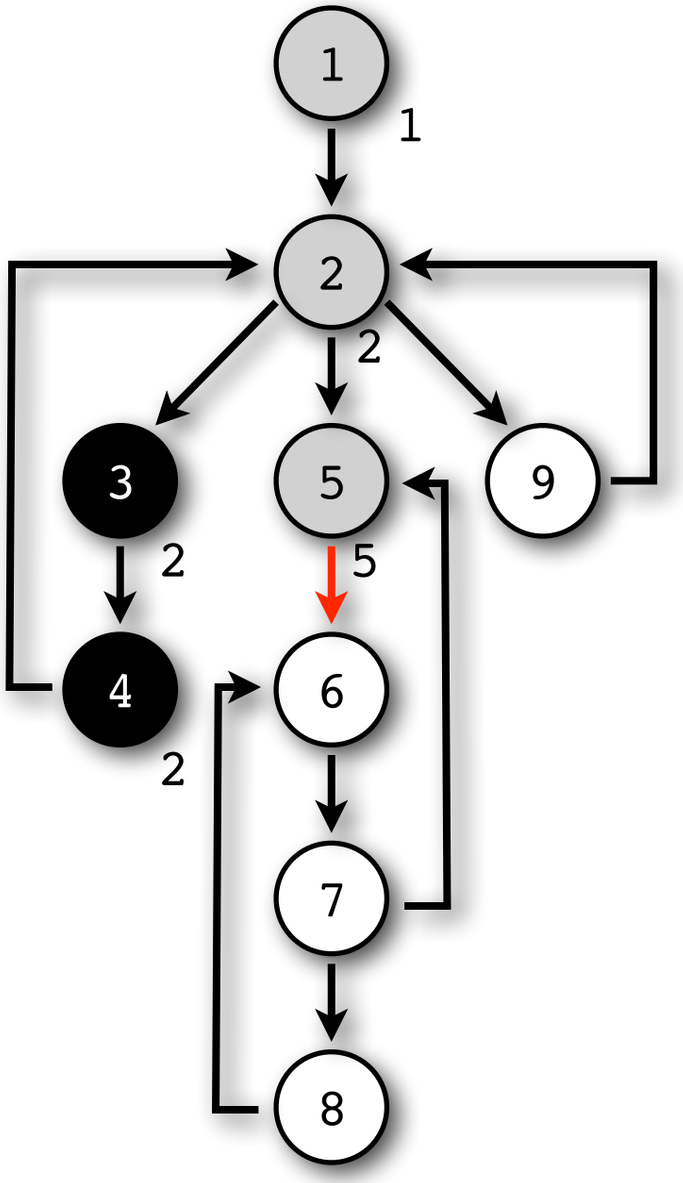


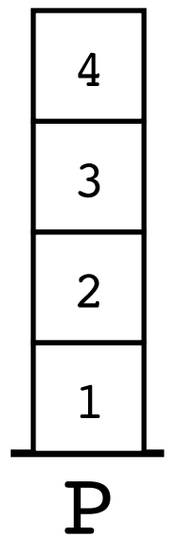
Tarjan(2)



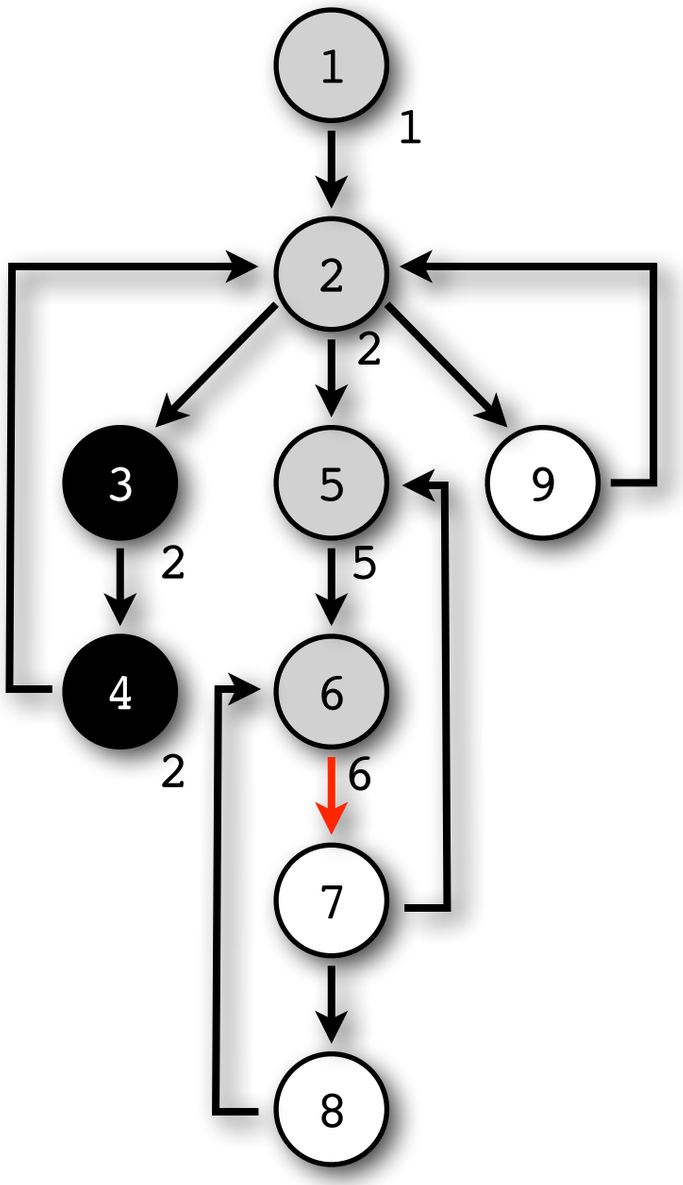


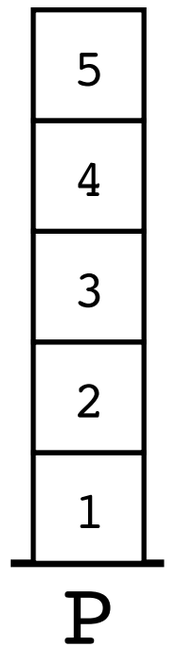
Tarjan(5)



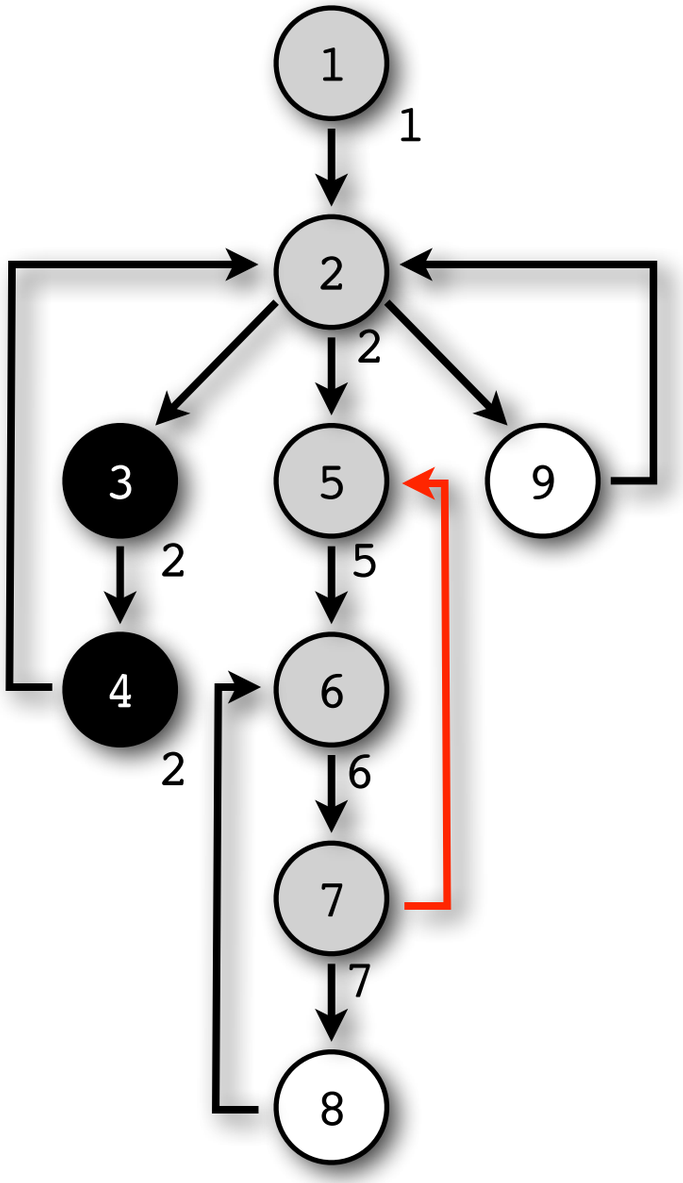


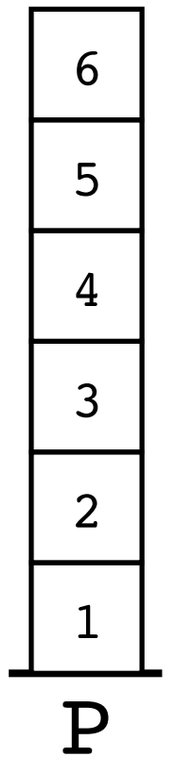
Tarjan(6)



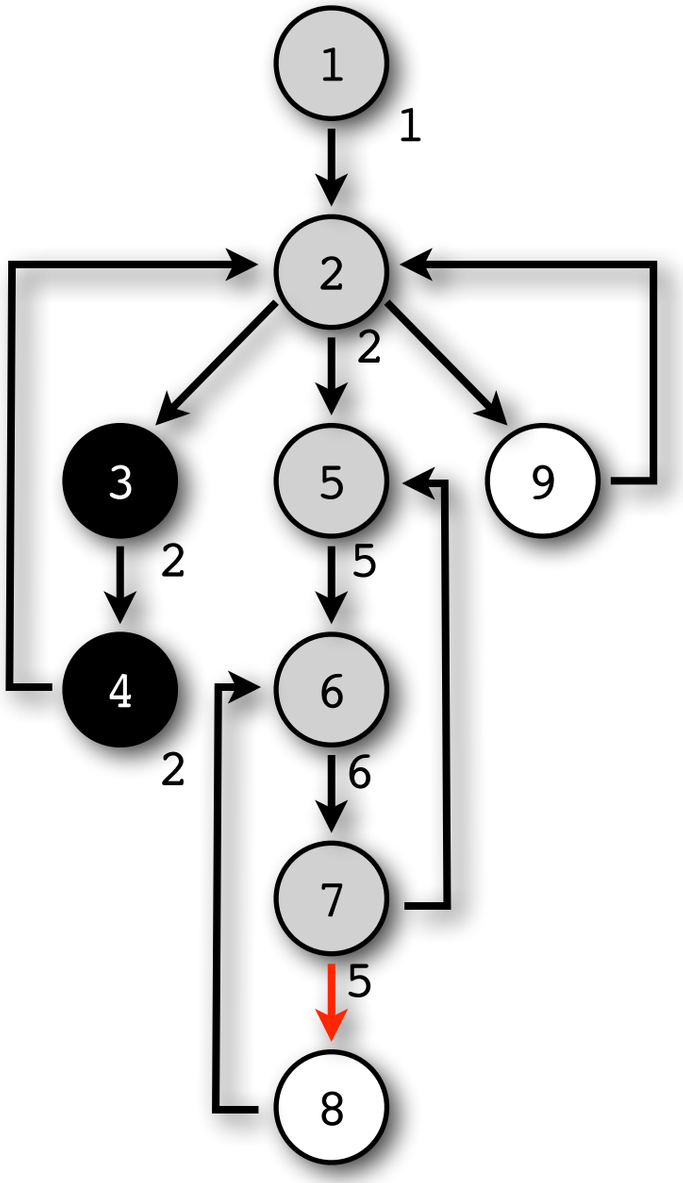


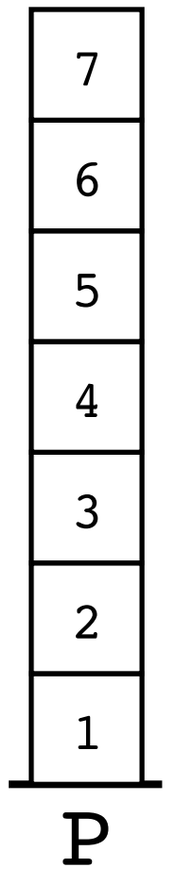
Tarjan(7)



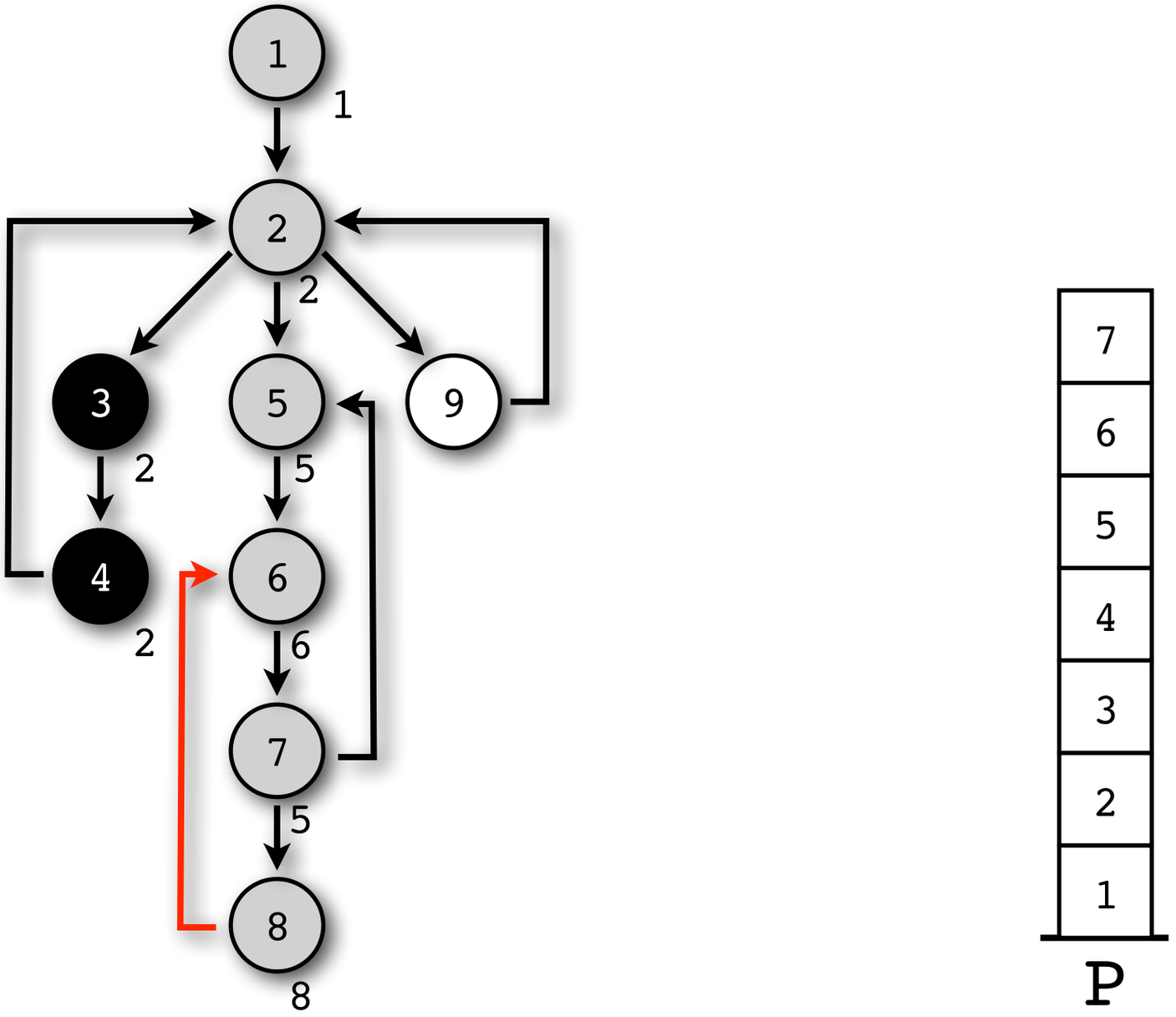


Tarjan(7)

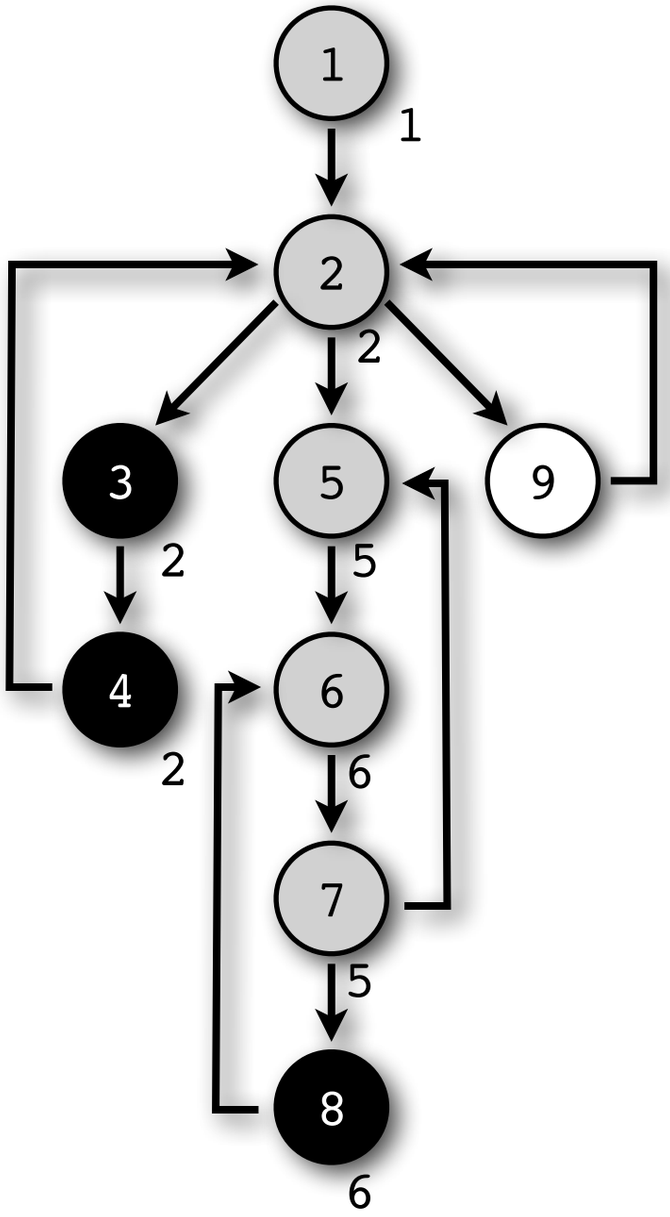


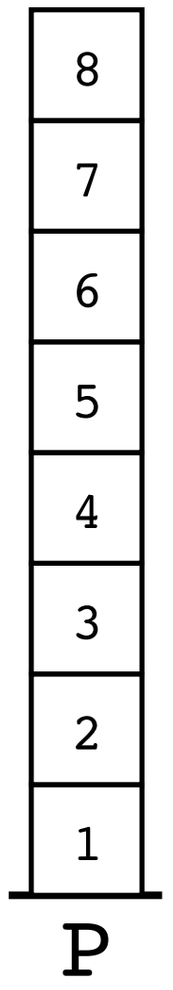


Tarjan(8)

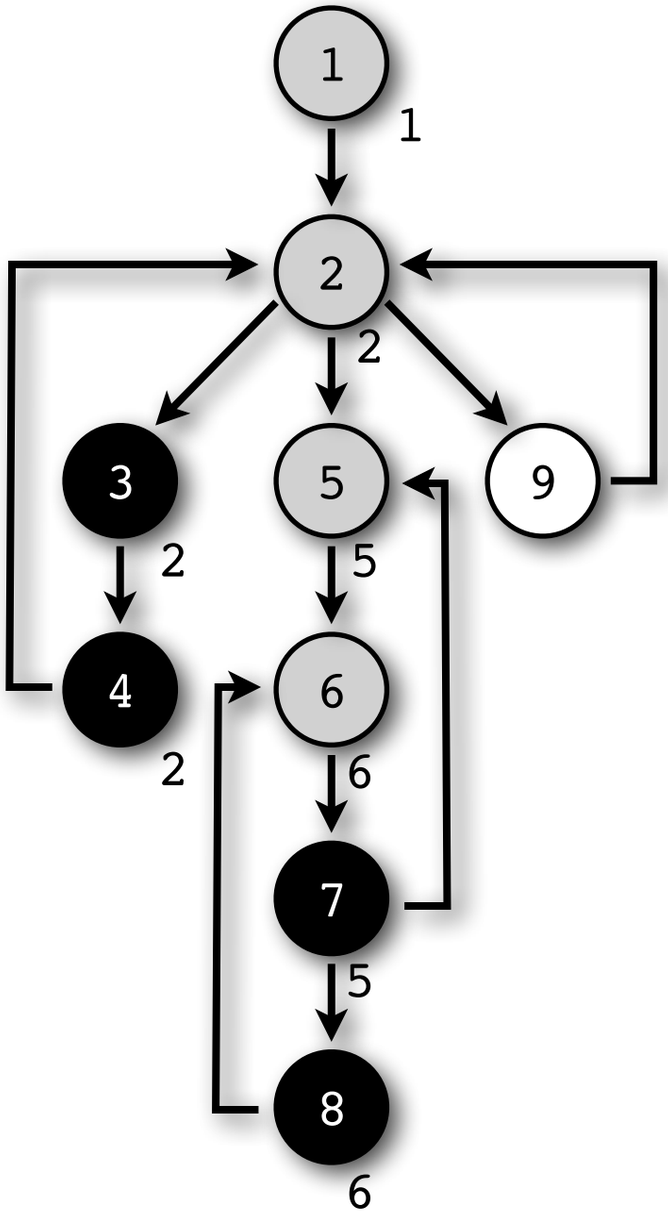


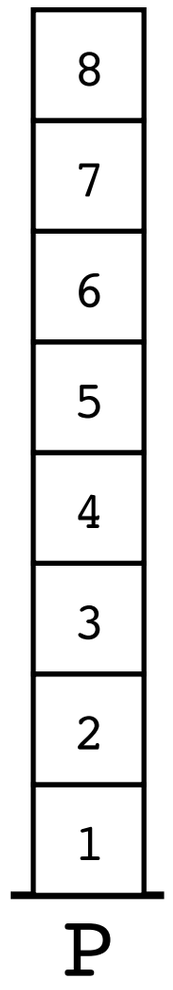
Tarjan(8)



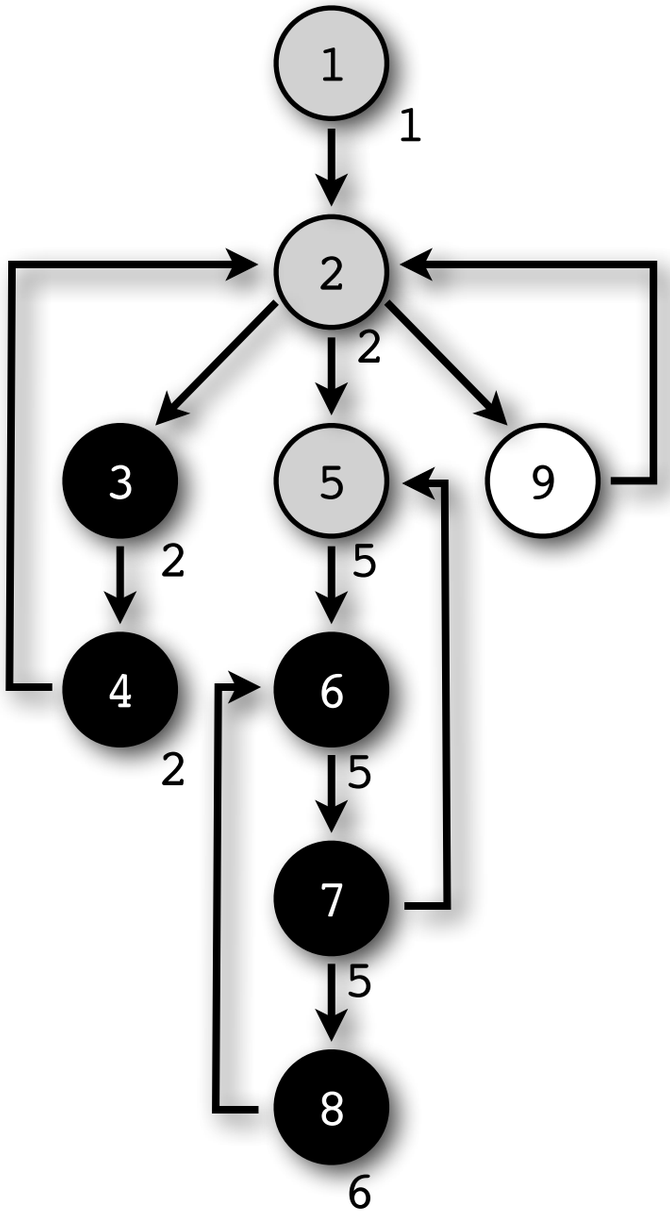


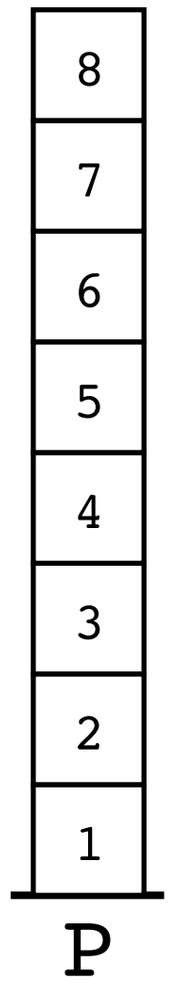
Tarjan(7)



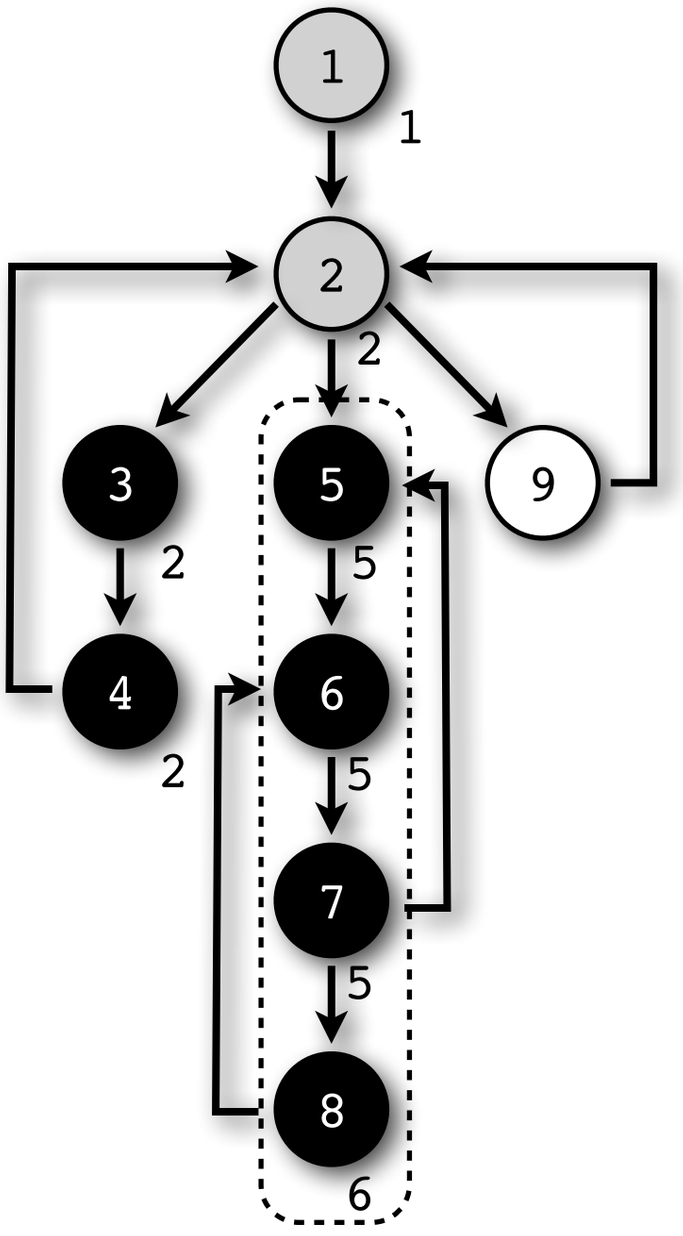


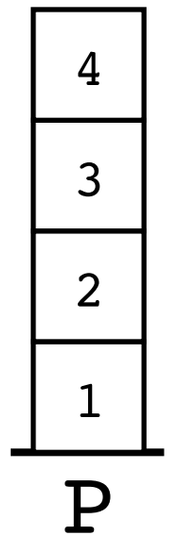
Tarjan(6)



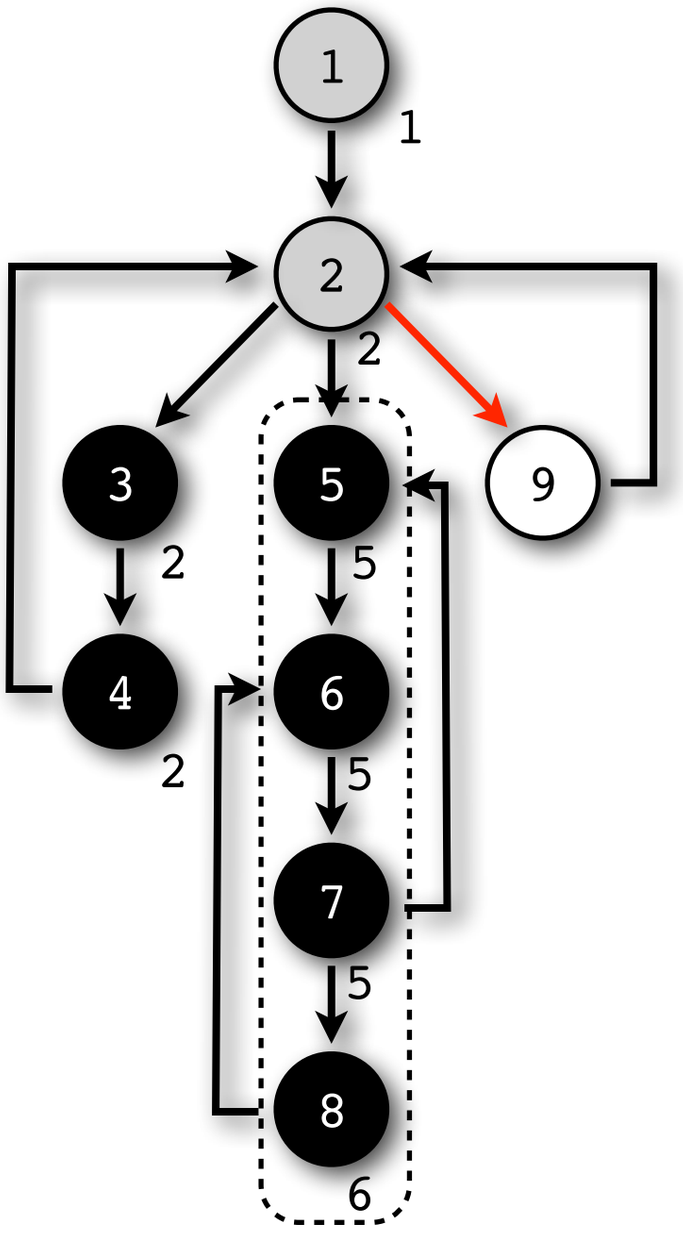


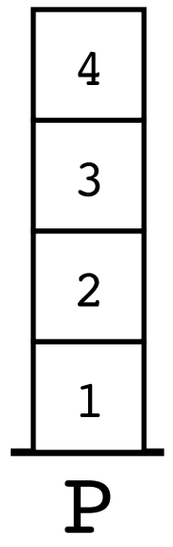
Tarjan(5)



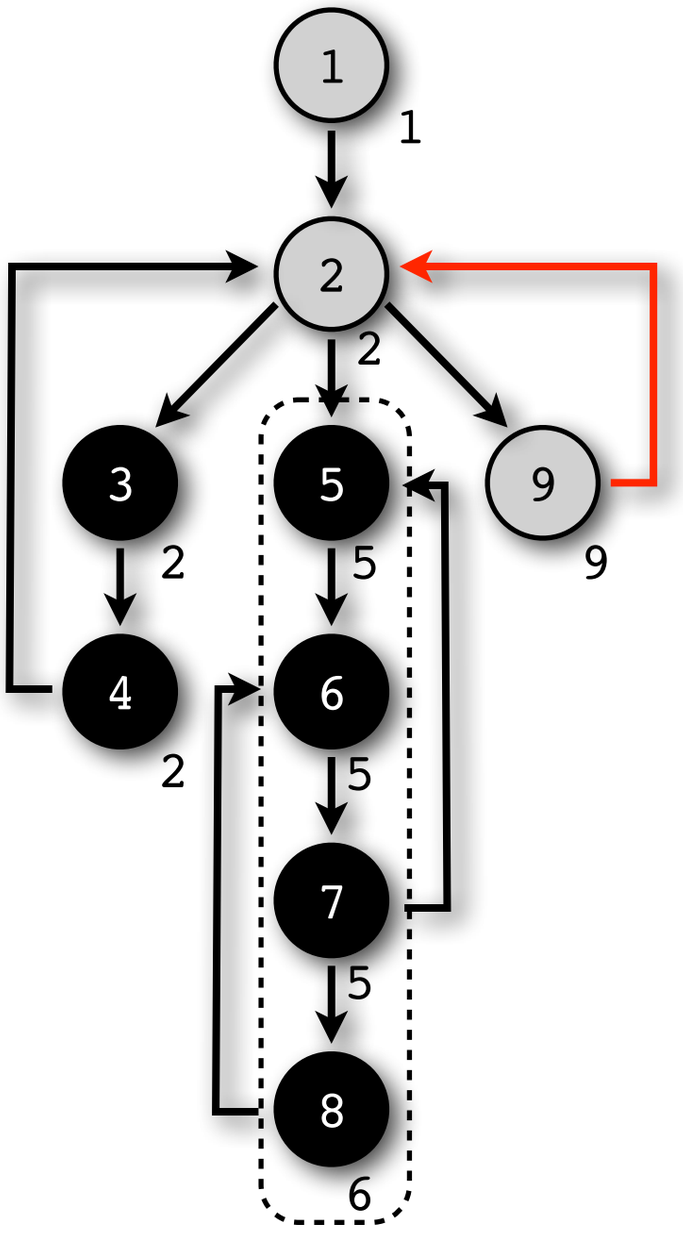


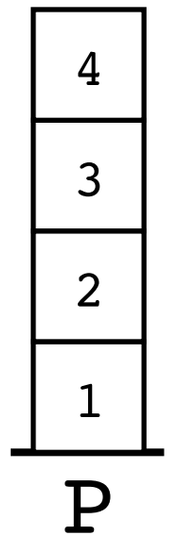
Tarjan(2)



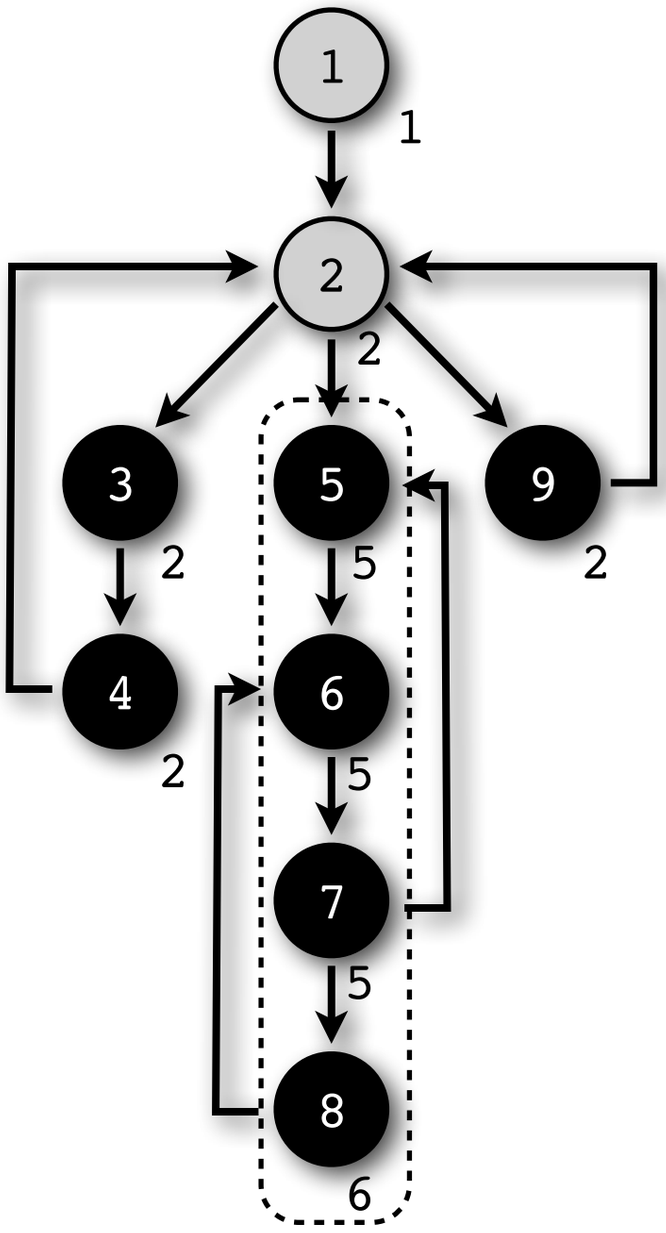


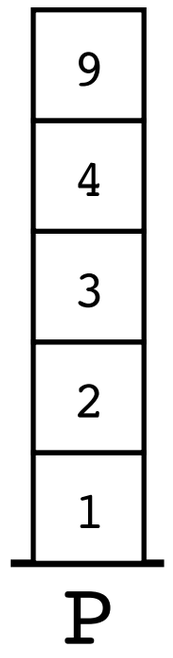
Tarjan(9)



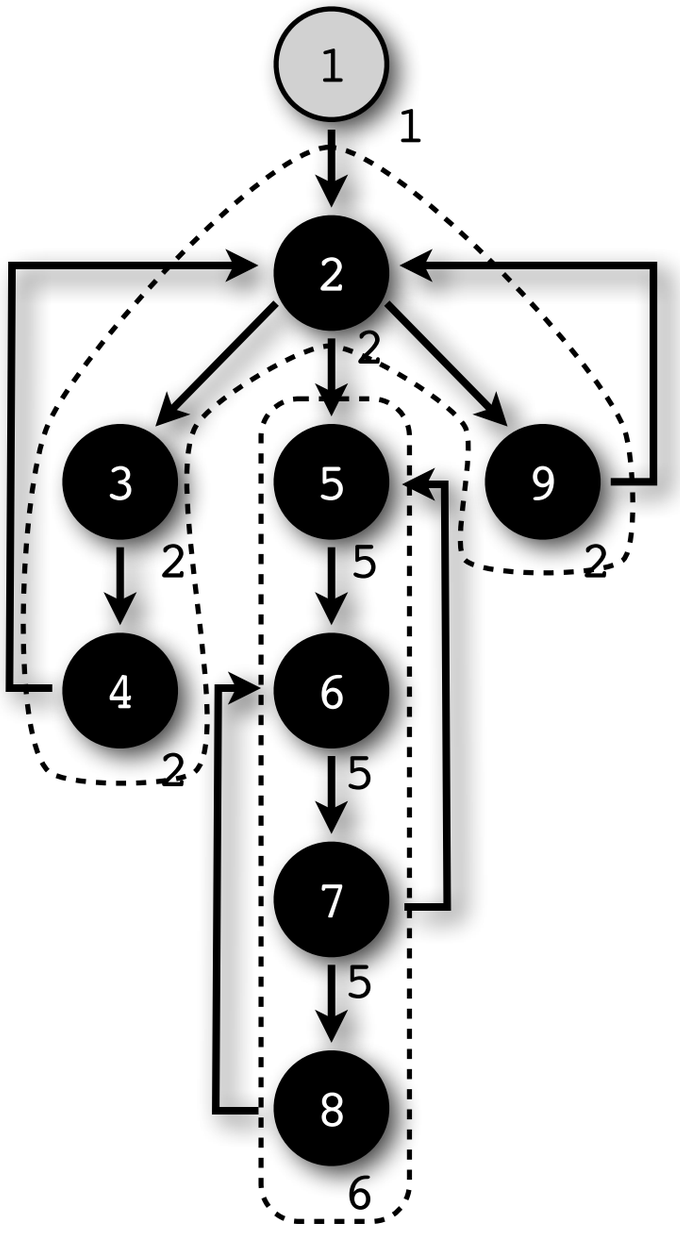


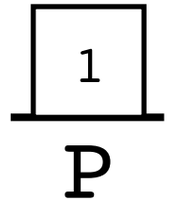
Tarjan(9)



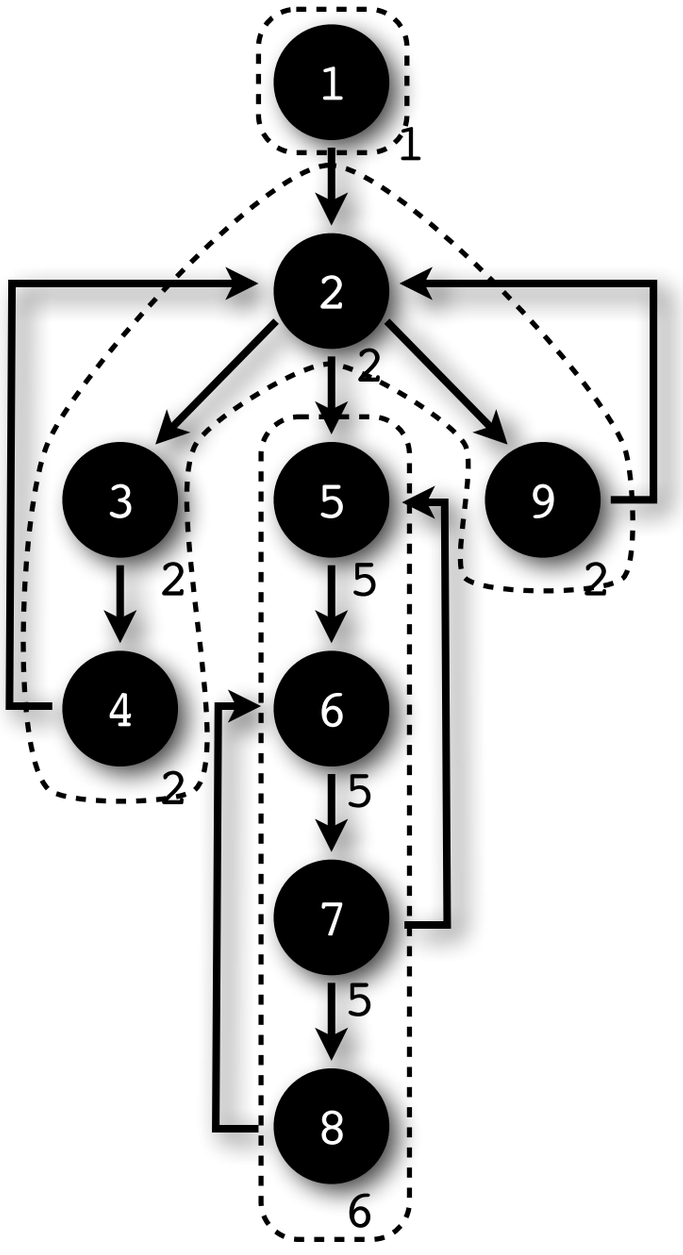


Tarjan(2)





Tarjan(1)



$$\frac{\text{(vide)}}{P}$$

```
cf comparaison_tarjan_kosaraju.pdf
```